

StarPU Handbook

for StarPU 1.2.0

This manual documents the usage of StarPU version 1.2.0. Its contents was last updated on 10 December 2014.

Copyright © 2009–2013 Université de Bordeaux

Copyright © 2010-2014 Centre National de la Recherche Scientifique

Copyright © 2011, 2012 Institut National de Recherche en Informatique et Automatique

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Contents

1	Introduction	3
1.1	Motivation	3
1.2	StarPU in a Nutshell	3
1.2.1	Codelet and Tasks	3
1.2.2	StarPU Data Management Library	4
1.3	Application Taskification	4
1.4	Glossary	4
1.5	Research Papers	5
1.6	StarPU Applications	5
1.7	Further Reading	5
I	StarPU Basics	7
2	Building and Installing StarPU	9
2.1	Installing a Binary Package	9
2.2	Installing from Source	9
2.2.1	Optional Dependencies	9
2.2.2	Getting Sources	9
2.2.3	Configuring StarPU	10
2.2.4	Building StarPU	10
2.2.5	Installing StarPU	10
2.3	Setting up Your Own Code	11
2.3.1	Setting Flags for Compiling, Linking and Running Applications	11
2.3.2	Running a Basic StarPU Application	11
2.3.3	Running a Basic StarPU Application on Microsoft Visual C	11
2.3.4	Kernel Threads Started by StarPU	12
2.3.5	Enabling OpenCL	12
2.4	Benchmarking StarPU	12
2.4.1	Task Size Overhead	13
2.4.2	Data Transfer Latency	13
2.4.3	Matrix-Matrix Multiplication	13

2.4.4	Cholesky Factorization	13
2.4.5	LU Factorization	13
3	Basic Examples	15
3.1	Hello World Using The C Extension	15
3.2	Hello World Using StarPU's API	16
3.2.1	Required Headers	16
3.2.2	Defining A Codelet	16
3.2.3	Submitting A Task	16
3.2.4	Execution Of Hello World	17
3.2.5	Passing Arguments To The Codelet	17
3.2.6	Defining A Callback	18
3.2.7	Where To Execute A Codelet	18
3.3	Vector Scaling Using the C Extension	19
3.3.1	Adding an OpenCL Task Implementation	20
3.3.2	Adding a CUDA Task Implementation	21
3.4	Vector Scaling Using StarPU's API	21
3.4.1	Source Code of Vector Scaling	21
3.4.2	Execution of Vector Scaling	23
3.5	Vector Scaling on an Hybrid CPU/GPU Machine	23
3.5.1	Definition of the CUDA Kernel	23
3.5.2	Definition of the OpenCL Kernel	23
3.5.3	Definition of the Main Code	24
3.5.4	Execution of Hybrid Vector Scaling	26
II	StarPU Quick Programming Guide	27
4	Advanced Examples	29
5	Check List When Performance Are Not There	31
5.1	Data Related Features That May Improve Performance	31
5.2	Task Related Features That May Improve Performance	31
5.3	Scheduling Related Features That May Improve Performance	31
5.4	CUDA-specific Optimizations	31
5.5	OpenCL-specific Optimizations	32
5.6	Detection Stuck Conditions	32
5.7	How to limit memory per node	32
5.8	Performance Model Calibration	33
5.9	Profiling	35

III	StarPU Inside	37
6	Tasks In StarPU	39
6.1	Task Granularity	39
6.2	Task Submission	39
6.3	Task Priorities	39
6.4	Setting The Data Handles For A Task	39
6.5	Using Multiple Implementations Of A Codelet	40
6.6	Enabling Implementation According To Capabilities	41
6.7	Insert Task Utility	42
6.8	Parallel Tasks	43
6.8.1	Fork-mode Parallel Tasks	43
6.8.2	SPMD-mode Parallel Tasks	44
6.8.3	Parallel Tasks Performance	44
6.8.4	Combined Workers	44
6.8.5	Concurrent Parallel Tasks	45
7	Data Management	47
7.1	Data Management	47
7.2	Data Prefetch	48
7.3	Partitioning Data	48
7.4	Data Reduction	49
7.5	Temporary Buffers	50
7.5.1	Temporary Data	50
7.5.2	Scratch Data	51
7.6	The Multiformat Interface	51
7.7	Defining A New Data Interface	52
7.8	Specifying a target node for task data	54
8	Scheduling	55
8.1	Task Scheduling Policy	55
8.2	Task Distribution Vs Data Transfer	56
8.3	Power-based Scheduling	56
8.4	Static Scheduling	56
8.5	Defining A New Scheduling Policy	57
9	Scheduling Contexts	59
9.1	General Ideas	59
9.2	Creating A Context	59
9.3	Modifying A Context	60
9.4	Submitting Tasks To A Context	60

9.5	Deleting A Context	60
9.6	Emptying A Context	60
9.7	Contexts Sharing Workers	61
10	Scheduling Context Hypervisor	63
10.1	What Is The Hypervisor	63
10.2	Start the Hypervisor	63
10.3	Interrogate The Runtime	63
10.4	Trigger the Hypervisor	63
10.5	Resizing Strategies	64
10.6	Defining A New Hypervisor Policy	65
11	Modularized Scheduler	67
11.1	Introduction	67
11.2	Using Modularized Schedulers	67
11.2.1	Existing Modularized Schedulers	67
11.2.2	An Example : The Tree-Eager-Prefetching Strategy	68
11.2.3	Interface	68
11.3	Build a Modularized Scheduler	69
11.3.1	Pre-implemented Components	69
11.3.2	Progression And Validation Rules	69
11.3.3	Implement a Modularized Scheduler	69
11.4	Write a Scheduling Component	70
11.4.1	Generic Scheduling Component	71
11.4.2	Instantiation : Redefine the Interface	71
11.4.3	Detailed Progression and Validation Rules	71
12	Debugging Tools	73
12.1	Using The Temanejo Task Debugger	73
13	Online Performance Tools	75
13.1	On-line Performance Feedback	75
13.1.1	Enabling On-line Performance Monitoring	75
13.1.2	Per-task Feedback	75
13.1.3	Per-codelet Feedback	75
13.1.4	Per-worker Feedback	76
13.1.5	Bus-related Feedback	76
13.1.6	StarPU-Top Interface	76
13.2	Task And Worker Profiling	77
13.3	Performance Model Example	78
13.4	Data trace and tasks length	80

14 Offline Performance Tools	81
14.1 Off-line Performance Feedback	81
14.1.1 Generating Traces With FxT	81
14.1.2 Creating a Gantt Diagram	82
14.1.3 Creating a DAG With Graphviz	83
14.1.4 Monitoring Activity	83
14.2 Performance Of Codelets	83
14.3 Trace statistics	89
14.4 Theoretical Lower Bound On Execution Time	91
14.5 Theoretical Lower Bound On Execution Time Example	91
14.6 Memory Feedback	91
14.7 Data Statistics	92
15 Frequently Asked Questions	95
15.1 How To Initialize A Computation Library Once For Each Worker?	95
15.2 Using The Driver API	96
15.3 On-GPU Rendering	96
15.4 Using StarPU With MKL 11 (Intel Composer XE 2013)	97
15.5 Thread Binding on NetBSD	97
15.6 Interleaving StarPU and non-StarPU code	97
IV StarPU Extensions	99
16 Out Of Core	101
16.1 Introduction	101
16.2 Use a new disk memory	101
16.3 Disk functions	102
16.4 Examples: disk_copy	102
16.5 Examples: disk_compute	103
17 MPI Support	107
17.1 Simple Example	107
17.2 Point To Point Communication	108
17.3 Exchanging User Defined Data Interface	109
17.4 MPI Insert Task Utility	109
17.5 MPI cache support	111
17.6 MPI Data migration	112
17.7 MPI Collective Operations	112
18 FFT Support	115
18.1 Compilation	115

19 MIC/SCC Support	117
19.1 Compilation	117
19.2 Porting Applications To MIC Xeon Phi / SCC	117
19.3 Launching Programs	117
20 C Extensions	119
20.1 Defining Tasks	119
20.2 Initialization, Termination, and Synchronization	121
20.3 Registered Data Buffers	121
20.4 Using C Extensions Conditionally	122
21 SOCL OpenCL Extensions	125
22 SimGrid Support	127
22.1 Preparing your application for simulation.	127
22.2 Calibration	127
22.3 Simulation	128
22.4 Simulation On Another Machine	128
22.5 applications	128
23 OpenMP Runtime Support	131
23.1 Implementation Details and Specificities	131
23.1.1 Main Thread	131
23.1.2 Extended Task Semantics	131
23.2 Configuration	131
23.3 Initialization and Shutdown	131
23.4 Parallel Regions and Worksharing	132
23.4.1 Parallel Regions	132
23.4.2 Parallel For	132
23.4.3 Sections	133
23.4.4 Single	134
23.5 Tasks	134
23.5.1 Explicit Tasks	134
23.5.2 Data Dependencies	135
23.5.3 TaskWait and TaskGroup	135
23.6 Synchronization Support	136
23.6.1 Simple Locks	136
23.6.2 Nestable Locks	136
23.6.3 Critical Sections	137
23.6.4 Barriers	137

V StarPU Reference API	139
24 Execution Configuration Through Environment Variables	141
24.1 Configuring Workers	141
24.2 Configuring The Scheduling Engine	143
24.3 Extensions	144
24.4 Miscellaneous And Debug	144
24.5 Configuring The Hypervisor	146
25 Compilation Configuration	147
25.1 Common Configuration	147
25.2 Configuring Workers	148
25.3 Extension Configuration	149
25.4 Advanced Configuration	149
26 Module Index	151
26.1 Modules	151
27 Module Documentation a.k.a StarPU's API	153
27.1 Versioning	153
27.1.1 Detailed Description	153
27.1.2 Macro Definition Documentation	153
27.1.2.1 STARPU_MAJOR_VERSION	153
27.1.2.2 STARPU_MINOR_VERSION	153
27.1.2.3 STARPU_RELEASE_VERSION	153
27.1.3 Function Documentation	153
27.1.3.1 starpu_get_version	153
27.2 Initialization and Termination	154
27.2.1 Detailed Description	154
27.2.2 Data Structure Documentation	154
27.2.2.1 struct starpu_driver	154
27.2.2.2 union starpu_driver.id	154
27.2.2.3 struct starpu_conf	154
27.2.3 Function Documentation	157
27.2.3.1 starpu_init	157
27.2.3.2 starpu_initialize	157
27.2.3.3 starpu_conf_init	157
27.2.3.4 starpu_shutdown	158
27.2.3.5 starpu_pause	158
27.2.3.6 starpu_resume	158
27.2.3.7 starpu_asynchronous_copy_disabled	158

27.2.3.8	starpu_asynchronous_cuda_copy_disabled	158
27.2.3.9	starpu_asynchronous_opengl_copy_disabled	158
27.2.3.10	starpu_topology_print	158
27.3	Standard Memory Library	158
27.3.1	Detailed Description	159
27.3.2	Macro Definition Documentation	159
27.3.2.1	starpu_data_malloc_pinned_if_possible	159
27.3.2.2	starpu_data_free_pinned_if_possible	159
27.3.2.3	STARPU_MALLOC_PINNED	159
27.3.2.4	STARPU_MALLOC_COUNT	159
27.3.2.5	STARPU_MALLOC_NORECLAIM	159
27.3.3	Function Documentation	159
27.3.3.1	starpu_malloc_flags	159
27.3.3.2	starpu_malloc_set_align	159
27.3.3.3	starpu_malloc	160
27.3.3.4	starpu_free	160
27.3.3.5	starpu_free_flags	160
27.3.3.6	starpu_memory_get_total	160
27.3.3.7	starpu_memory_get_available	160
27.3.3.8	starpu_memory_allocate	160
27.3.3.9	starpu_memory_deallocate	160
27.4	Toolbox	161
27.4.1	Detailed Description	161
27.4.2	Macro Definition Documentation	161
27.4.2.1	STARPU_GNUC_PREREQ	161
27.4.2.2	STARPU_UNLIKELY	162
27.4.2.3	STARPU_LIKELY	162
27.4.2.4	STARPU_ATTRIBUTE_UNUSED	162
27.4.2.5	STARPU_ATTRIBUTE_INTERNAL	162
27.4.2.6	STARPU_ATTRIBUTE_MALLOC	162
27.4.2.7	STARPU_ATTRIBUTE_WARN_UNUSED_RESULT	162
27.4.2.8	STARPU_ATTRIBUTE_PURE	162
27.4.2.9	STARPU_ATTRIBUTE_ALIGNED	162
27.4.2.10	STARPU_WARN_UNUSED_RESULT	162
27.4.2.11	STARPU_POISON_PTR	162
27.4.2.12	STARPU_MIN	162
27.4.2.13	STARPU_MAX	162
27.4.2.14	STARPU_ASSERT	163
27.4.2.15	STARPU_ASSERT_MSG	163
27.4.2.16	STARPU_ABORT	163

27.4.2.17	STARPU_ABORT_MSG	163
27.4.2.18	STARPU_CHECK_RETURN_VALUE	163
27.4.2.19	STARPU_CHECK_RETURN_VALUE_IS	163
27.4.2.20	STARPU_RMB	163
27.4.2.21	STARPU_WMB	163
27.4.3	Function Documentation	163
27.4.3.1	starpu_get_env_number	163
27.5	Threads	163
27.5.1	Detailed Description	165
27.5.2	Macro Definition Documentation	165
27.5.2.1	STARPU_PTHREAD_CREATE_ON	165
27.5.2.2	STARPU_PTHREAD_CREATE	165
27.5.2.3	STARPU_PTHREAD_MUTEX_INIT	165
27.5.2.4	STARPU_PTHREAD_MUTEX_DESTROY	165
27.5.2.5	STARPU_PTHREAD_MUTEX_LOCK	165
27.5.2.6	STARPU_PTHREAD_MUTEX_UNLOCK	165
27.5.2.7	STARPU_PTHREAD_KEY_CREATE	166
27.5.2.8	STARPU_PTHREAD_KEY_DELETE	166
27.5.2.9	STARPU_PTHREAD_SETSPECIFIC	166
27.5.2.10	STARPU_PTHREAD_GETSPECIFIC	166
27.5.2.11	STARPU_PTHREAD_RWLOCK_INIT	166
27.5.2.12	STARPU_PTHREAD_RWLOCK_RDLOCK	166
27.5.2.13	STARPU_PTHREAD_RWLOCK_WRLOCK	166
27.5.2.14	STARPU_PTHREAD_RWLOCK_UNLOCK	166
27.5.2.15	STARPU_PTHREAD_RWLOCK_DESTROY	166
27.5.2.16	STARPU_PTHREAD_COND_INIT	166
27.5.2.17	STARPU_PTHREAD_COND_DESTROY	166
27.5.2.18	STARPU_PTHREAD_COND_SIGNAL	166
27.5.2.19	STARPU_PTHREAD_COND_BROADCAST	167
27.5.2.20	STARPU_PTHREAD_COND_WAIT	167
27.5.2.21	STARPU_PTHREAD_BARRIER_INIT	167
27.5.2.22	STARPU_PTHREAD_BARRIER_DESTROY	167
27.5.2.23	STARPU_PTHREAD_BARRIER_WAIT	167
27.5.2.24	STARPU_PTHREAD_MUTEX_INITIALIZER	167
27.5.2.25	STARPU_PTHREAD_COND_INITIALIZER	167
27.5.3	Function Documentation	167
27.5.3.1	starpu_pthread_create	167
27.5.3.2	starpu_pthread_join	167
27.5.3.3	starpu_pthread_exit	167
27.5.3.4	starpu_pthread_attr_init	167

27.5.3.5	starpu_thread_attr_destroy	168
27.5.3.6	starpu_thread_attr_setdetachstate	168
27.5.3.7	starpu_thread_mutex_init	168
27.5.3.8	starpu_thread_mutex_destroy	168
27.5.3.9	starpu_thread_mutex_lock	168
27.5.3.10	starpu_thread_mutex_unlock	168
27.5.3.11	starpu_thread_mutex_trylock	168
27.5.3.12	starpu_thread_mutexattr_gettype	168
27.5.3.13	starpu_thread_mutexattr_settype	168
27.5.3.14	starpu_thread_mutexattr_destroy	169
27.5.3.15	starpu_thread_mutexattr_init	169
27.5.3.16	starpu_thread_key_create	169
27.5.3.17	starpu_thread_key_delete	169
27.5.3.18	starpu_thread_setspecific	169
27.5.3.19	starpu_thread_getspecific	169
27.5.3.20	starpu_thread_cond_init	169
27.5.3.21	starpu_thread_cond_signal	169
27.5.3.22	starpu_thread_cond_broadcast	169
27.5.3.23	starpu_thread_cond_wait	169
27.5.3.24	starpu_thread_cond_timedwait	170
27.5.3.25	starpu_thread_cond_destroy	170
27.5.3.26	starpu_thread_rwlock_init	170
27.5.3.27	starpu_thread_rwlock_destroy	170
27.5.3.28	starpu_thread_rwlock_rdlock	170
27.5.3.29	starpu_thread_rwlock_tryrdlock	170
27.5.3.30	starpu_thread_rwlock_wrlock	170
27.5.3.31	starpu_thread_rwlock_trywrlock	170
27.5.3.32	starpu_thread_rwlock_unlock	170
27.5.3.33	starpu_thread_barrier_init	170
27.5.3.34	starpu_thread_barrier_destroy	170
27.5.3.35	starpu_thread_barrier_wait	171
27.5.3.36	starpu_thread_spin_init	171
27.5.3.37	starpu_thread_spin_destroy	171
27.5.3.38	starpu_thread_spin_lock	171
27.5.3.39	starpu_thread_spin_trylock	171
27.5.3.40	starpu_thread_spin_unlock	171
27.6	Bitmap	171
27.6.1	Detailed Description	171
27.6.2	Function Documentation	172
27.6.2.1	starpu_bitmap_create	172

27.6.2.2	<code>starpu_bitmap_destroy</code>	172
27.6.2.3	<code>starpu_bitmap_set</code>	172
27.6.2.4	<code>starpu_bitmap_unset</code>	172
27.6.2.5	<code>starpu_bitmap_unset_all</code>	172
27.6.2.6	<code>starpu_bitmap_get</code>	172
27.6.2.7	<code>starpu_bitmap_unset_and</code>	172
27.6.2.8	<code>starpu_bitmap_or</code>	172
27.6.2.9	<code>starpu_bitmap_and_get</code>	172
27.6.2.10	<code>starpu_bitmap_cardinal</code>	172
27.6.2.11	<code>starpu_bitmap_first</code>	172
27.6.2.12	<code>starpu_bitmap_last</code>	172
27.6.2.13	<code>starpu_bitmap_next</code>	173
27.6.2.14	<code>starpu_bitmap_has_next</code>	173
27.7	Workers' Properties	173
27.7.1	Detailed Description	174
27.7.2	Data Structure Documentation	174
27.7.2.1	<code>struct starpu_worker_collection</code>	174
27.7.2.2	<code>struct starpu_sched_ctx_iterator</code>	175
27.7.3	Macro Definition Documentation	175
27.7.3.1	<code>STARPU_NMAXWORKERS</code>	175
27.7.4	Enumeration Type Documentation	175
27.7.4.1	<code>starpu_node_kind</code>	175
27.7.4.2	<code>starpu_worker_archtype</code>	176
27.7.4.3	<code>starpu_worker_collection_type</code>	176
27.7.5	Function Documentation	176
27.7.5.1	<code>starpu_worker_get_count</code>	176
27.7.5.2	<code>starpu_worker_get_count_by_type</code>	176
27.7.5.3	<code>starpu_cpu_worker_get_count</code>	176
27.7.5.4	<code>starpu_cuda_worker_get_count</code>	176
27.7.5.5	<code>starpu_mic_worker_get_count</code>	176
27.7.5.6	<code>starpu_mic_device_get_count</code>	176
27.7.5.7	<code>starpu_scc_worker_get_count</code>	177
27.7.5.8	<code>starpu_opencl_worker_get_count</code>	177
27.7.5.9	<code>starpu_worker_get_id</code>	177
27.7.5.10	<code>starpu_worker_get_ids_by_type</code>	177
27.7.5.11	<code>starpu_worker_get_by_type</code>	177
27.7.5.12	<code>starpu_worker_get_by_devid</code>	177
27.7.5.13	<code>starpu_worker_get_devid</code>	177
27.7.5.14	<code>starpu_worker_get_type</code>	177
27.7.5.15	<code>starpu_worker_get_name</code>	178

27.7.5.16	starpu_worker_get_memory_node	178
27.7.5.17	starpu_node_get_kind	178
27.7.5.18	starpu_worker_get_type_as_string	178
27.8	Data Management	178
27.8.1	Detailed Description	179
27.8.2	Macro Definition Documentation	179
27.8.2.1	STARPU_DATA_ACQUIRE_CB	179
27.8.3	Typedef Documentation	180
27.8.3.1	starpu_data_handle_t	180
27.8.4	Enumeration Type Documentation	180
27.8.4.1	starpu_data_access_mode	180
27.8.5	Function Documentation	180
27.8.5.1	starpu_data_register	180
27.8.5.2	starpu_data_ptr_register	180
27.8.5.3	starpu_data_register_same	181
27.8.5.4	starpu_data_unregister	181
27.8.5.5	starpu_data_unregister_no_coherency	181
27.8.5.6	starpu_data_unregister_submit	181
27.8.5.7	starpu_data_invalidate	181
27.8.5.8	starpu_data_invalidate_submit	181
27.8.5.9	starpu_data_set_wt_mask	181
27.8.5.10	starpu_data_prefetch_on_node	181
27.8.5.11	starpu_data_lookup	181
27.8.5.12	starpu_data_request_allocation	181
27.8.5.13	starpu_data_query_status	182
27.8.5.14	starpu_data_advise_as_important	182
27.8.5.15	starpu_data_set_reduction_methods	182
27.8.5.16	starpu_data_get_interface_ops	182
27.8.5.17	starpu_data_acquire	182
27.8.5.18	starpu_data_acquire_cb	182
27.8.5.19	starpu_data_acquire_cb_sequential_consistency	182
27.8.5.20	starpu_data_acquire_on_node	183
27.8.5.21	starpu_data_acquire_on_node_cb	183
27.8.5.22	starpu_data_acquire_on_node_cb_sequential_consistency	183
27.8.5.23	starpu_data_release	183
27.8.5.24	starpu_data_release_on_node	183
27.9	Data Interfaces	183
27.9.1	Detailed Description	187
27.9.2	Data Structure Documentation	187
27.9.2.1	struct starpu_data_interface_ops	187

27.9.2.2	struct starpu_data_copy_methods	189
27.9.2.3	struct starpu_variable_interface	192
27.9.2.4	struct starpu_vector_interface	192
27.9.2.5	struct starpu_matrix_interface	193
27.9.2.6	struct starpu_block_interface	193
27.9.2.7	struct starpu_bcsr_interface	193
27.9.2.8	struct starpu_csr_interface	194
27.9.2.9	struct starpu_coo_interface	194
27.9.3	Macro Definition Documentation	194
27.9.3.1	STARPU_VARIABLE_GET_PTR	194
27.9.3.2	STARPU_VARIABLE_GET_ELEMSIZE	194
27.9.3.3	STARPU_VARIABLE_GET_DEV_HANDLE	195
27.9.3.4	STARPU_VARIABLE_GET_OFFSET	195
27.9.3.5	STARPU_VECTOR_GET_PTR	195
27.9.3.6	STARPU_VECTOR_GET_DEV_HANDLE	195
27.9.3.7	STARPU_VECTOR_GET_OFFSET	195
27.9.3.8	STARPU_VECTOR_GET_NX	195
27.9.3.9	STARPU_VECTOR_GET_ELEMSIZE	195
27.9.3.10	STARPU_VECTOR_GET_SLICE_BASE	195
27.9.3.11	STARPU_MATRIX_GET_PTR	195
27.9.3.12	STARPU_MATRIX_GET_DEV_HANDLE	195
27.9.3.13	STARPU_MATRIX_GET_OFFSET	196
27.9.3.14	STARPU_MATRIX_GET_NX	196
27.9.3.15	STARPU_MATRIX_GET_NY	196
27.9.3.16	STARPU_MATRIX_GET_LD	196
27.9.3.17	STARPU_MATRIX_GET_ELEMSIZE	196
27.9.3.18	STARPU_BLOCK_GET_PTR	196
27.9.3.19	STARPU_BLOCK_GET_DEV_HANDLE	196
27.9.3.20	STARPU_BLOCK_GET_OFFSET	196
27.9.3.21	STARPU_BLOCK_GET_NX	196
27.9.3.22	STARPU_BLOCK_GET_NY	196
27.9.3.23	STARPU_BLOCK_GET_NZ	196
27.9.3.24	STARPU_BLOCK_GET_LDY	196
27.9.3.25	STARPU_BLOCK_GET_LDZ	197
27.9.3.26	STARPU_BLOCK_GET_ELEMSIZE	197
27.9.3.27	STARPU_BCSR_GET_NNZ	197
27.9.3.28	STARPU_BCSR_GET_NZVAL	197
27.9.3.29	STARPU_BCSR_GET_NZVAL_DEV_HANDLE	197
27.9.3.30	STARPU_BCSR_GET_COLIND	197
27.9.3.31	STARPU_BCSR_GET_COLIND_DEV_HANDLE	197

27.9.3.32	STARPU_BCSR_GET_ROWPTR	197
27.9.3.33	STARPU_BCSR_GET_ROWPTR_DEV_HANDLE	197
27.9.3.34	STARPU_BCSR_GET_OFFSET	197
27.9.3.35	STARPU_CSR_GET_NNZ	197
27.9.3.36	STARPU_CSR_GET_NROW	198
27.9.3.37	STARPU_CSR_GET_NZVAL	198
27.9.3.38	STARPU_CSR_GET_NZVAL_DEV_HANDLE	198
27.9.3.39	STARPU_CSR_GET_COLIND	198
27.9.3.40	STARPU_CSR_GET_COLIND_DEV_HANDLE	198
27.9.3.41	STARPU_CSR_GET_ROWPTR	198
27.9.3.42	STARPU_CSR_GET_ROWPTR_DEV_HANDLE	198
27.9.3.43	STARPU_CSR_GET_OFFSET	198
27.9.3.44	STARPU_CSR_GET_FIRSTENTRY	198
27.9.3.45	STARPU_CSR_GET_ELEMSIZE	198
27.9.3.46	STARPU_COO_GET_COLUMNS	198
27.9.3.47	STARPU_COO_GET_COLUMNS_DEV_HANDLE	199
27.9.3.48	STARPU_COO_GET_ROWS	199
27.9.3.49	STARPU_COO_GET_ROWS_DEV_HANDLE	199
27.9.3.50	STARPU_COO_GET_VALUES	199
27.9.3.51	STARPU_COO_GET_VALUES_DEV_HANDLE	199
27.9.3.52	STARPU_COO_GET_OFFSET	199
27.9.3.53	STARPU_COO_GET_NX	199
27.9.3.54	STARPU_COO_GET_NY	199
27.9.3.55	STARPU_COO_GET_NVALUES	199
27.9.3.56	STARPU_COO_GET_ELEMSIZE	199
27.9.4	Enumeration Type Documentation	199
27.9.4.1	starpu_data_interface_id	199
27.9.5	Function Documentation	200
27.9.5.1	starpu_void_data_register	200
27.9.5.2	starpu_variable_data_register	200
27.9.5.3	starpu_variable_ptr_register	200
27.9.5.4	starpu_vector_data_register	200
27.9.5.5	starpu_vector_ptr_register	201
27.9.5.6	starpu_matrix_data_register	201
27.9.5.7	starpu_matrix_ptr_register	201
27.9.5.8	starpu_block_data_register	201
27.9.5.9	starpu_block_ptr_register	201
27.9.5.10	starpu_bcsr_data_register	201
27.9.5.11	starpu_csr_data_register	202
27.9.5.12	starpu_coo_data_register	202

27.9.5.13 starpu_data_get_interface_on_node	202
27.9.5.14 starpu_data_handle_to_pointer	202
27.9.5.15 starpu_data_get_local_ptr	202
27.9.5.16 starpu_data_get_interface_id	202
27.9.5.17 starpu_data_get_size	202
27.9.5.18 starpu_data_pack	202
27.9.5.19 starpu_data_unpack	202
27.9.5.20 starpu_variable_get_elemsize	202
27.9.5.21 starpu_variable_get_local_ptr	203
27.9.5.22 starpu_vector_get_nx	203
27.9.5.23 starpu_vector_get_elemsize	203
27.9.5.24 starpu_vector_get_local_ptr	203
27.9.5.25 starpu_matrix_get_nx	203
27.9.5.26 starpu_matrix_get_ny	203
27.9.5.27 starpu_matrix_get_local_id	203
27.9.5.28 starpu_matrix_get_local_ptr	203
27.9.5.29 starpu_matrix_get_elemsize	203
27.9.5.30 starpu_block_get_nx	203
27.9.5.31 starpu_block_get_ny	203
27.9.5.32 starpu_block_get_nz	203
27.9.5.33 starpu_block_get_local_idy	204
27.9.5.34 starpu_block_get_local_idz	204
27.9.5.35 starpu_block_get_local_ptr	204
27.9.5.36 starpu_block_get_elemsize	204
27.9.5.37 starpu_bcsr_get_nnz	204
27.9.5.38 starpu_bcsr_get_nrow	204
27.9.5.39 starpu_bcsr_get_firstentry	204
27.9.5.40 starpu_bcsr_get_local_nzval	204
27.9.5.41 starpu_bcsr_get_local_colind	204
27.9.5.42 starpu_bcsr_get_local_rowptr	204
27.9.5.43 starpu_bcsr_get_r	204
27.9.5.44 starpu_bcsr_get_c	204
27.9.5.45 starpu_bcsr_get_elemsize	205
27.9.5.46 starpu_csr_get_nnz	205
27.9.5.47 starpu_csr_get_nrow	205
27.9.5.48 starpu_csr_get_firstentry	205
27.9.5.49 starpu_csr_get_local_nzval	205
27.9.5.50 starpu_csr_get_local_colind	205
27.9.5.51 starpu_csr_get_local_rowptr	205
27.9.5.52 starpu_csr_get_elemsize	205

27.9.5.53 starpu_malloc_on_node	205
27.9.5.54 starpu_free_on_node	205
27.9.5.55 starpu_interface_copy	205
27.9.5.56 starpu_hash_crc32c_be_n	206
27.9.5.57 starpu_hash_crc32c_be	206
27.9.5.58 starpu_hash_crc32c_string	206
27.9.5.59 starpu_data_interface_get_next_id	206
27.10 Data Partition	206
27.10.1 Detailed Description	207
27.10.2 Data Structure Documentation	207
27.10.2.1 struct starpu_data_filter	207
27.10.3 Function Documentation	208
27.10.3.1 starpu_data_partition	208
27.10.3.2 starpu_data_unpartition	209
27.10.3.3 starpu_data_get_nb_children	209
27.10.3.4 starpu_data_get_child	209
27.10.3.5 starpu_data_get_sub_data	209
27.10.3.6 starpu_data_vget_sub_data	209
27.10.3.7 starpu_data_map_filters	209
27.10.3.8 starpu_data_vmap_filters	209
27.10.3.9 starpu_vector_filter_block	209
27.10.3.10 starpu_vector_filter_block_shadow	210
27.10.3.11 starpu_vector_filter_list	210
27.10.3.12 starpu_vector_filter_divide_in_2	210
27.10.3.13 starpu_matrix_filter_block	210
27.10.3.14 starpu_matrix_filter_block_shadow	210
27.10.3.15 starpu_matrix_filter_vertical_block	210
27.10.3.16 starpu_matrix_filter_vertical_block_shadow	210
27.10.3.17 starpu_block_filter_block	211
27.10.3.18 starpu_block_filter_block_shadow	211
27.10.3.19 starpu_block_filter_vertical_block	211
27.10.3.20 starpu_block_filter_vertical_block_shadow	211
27.10.3.21 starpu_block_filter_depth_block	211
27.10.3.22 starpu_block_filter_depth_block_shadow	211
27.10.3.23 starpu_bcsr_filter_canonical_block	211
27.10.3.24 starpu_csr_filter_vertical_block	211
27.11 Out Of Core	212
27.11.1 Detailed Description	212
27.11.2 Data Structure Documentation	212
27.11.2.1 struct starpu_disk_ops	212

27.11.3 Function Documentation	214
27.11.3.1 starpu_disk_open	214
27.11.3.2 starpu_disk_close	214
27.11.4 Variable Documentation	214
27.11.4.1 starpu_disk_stdio_ops	214
27.11.4.2 starpu_disk_unistd_ops	214
27.11.4.3 starpu_disk_unistd_o_direct_ops	214
27.11.4.4 starpu_disk_leveldb_ops	215
27.12 Multiformat Data Interface	215
27.12.1 Detailed Description	215
27.12.2 Data Structure Documentation	215
27.12.2.1 struct starpu_multiformat_data_interface_ops	215
27.12.2.2 struct starpu_multiformat_interface	216
27.12.3 Macro Definition Documentation	216
27.12.3.1 STARPU_MULTIFORMAT_GET_CPU_PTR	216
27.12.3.2 STARPU_MULTIFORMAT_GET_CUDA_PTR	216
27.12.3.3 STARPU_MULTIFORMAT_GET_OPENCL_PTR	216
27.12.3.4 STARPU_MULTIFORMAT_GET_MIC_PTR	216
27.12.3.5 STARPU_MULTIFORMAT_GET_NX	216
27.12.4 Function Documentation	216
27.12.4.1 starpu_multiformat_data_register	217
27.13 Codelet And Tasks	217
27.13.1 Detailed Description	218
27.13.2 Data Structure Documentation	218
27.13.2.1 struct starpu_codelet	218
27.13.2.2 struct starpu_data_descr	222
27.13.2.3 struct starpu_task	222
27.13.3 Macro Definition Documentation	227
27.13.3.1 STARPU_CPU	227
27.13.3.2 STARPU_CUDA	227
27.13.3.3 STARPU_OPENCL	227
27.13.3.4 STARPU_MIC	227
27.13.3.5 STARPU_SCC	228
27.13.3.6 STARPU_MAIN_RAM	228
27.13.3.7 STARPU_MULTIPLE_CPU_IMPLEMENTATIONS	228
27.13.3.8 STARPU_MULTIPLE_CUDA_IMPLEMENTATIONS	228
27.13.3.9 STARPU_MULTIPLE_OPENCL_IMPLEMENTATIONS	228
27.13.3.10 STARPU_NMAXBUFS	228
27.13.3.11 STARPU_VARIABLE_NBUFFERS	228
27.13.3.12 STARPU_TASK_INITIALIZER	228

27.13.3.13	STARPU_TASK_GET_NBUFFERS	228
27.13.3.14	STARPU_TASK_GET_HANDLE	229
27.13.3.15	STARPU_TASK_SET_HANDLE	229
27.13.3.16	STARPU_CODELET_GET_MODE	229
27.13.3.17	STARPU_CODELET_SET_MODE	229
27.13.3.18	STARPU_TASK_GET_MODE	229
27.13.3.19	STARPU_TASK_SET_MODE	229
27.13.3.20	STARPU_TASK_INVALID	229
27.13.4	Typedef Documentation	229
27.13.4.1	starpu_cpu_func_t	229
27.13.4.2	starpu_cuda_func_t	229
27.13.4.3	starpu_openccl_func_t	230
27.13.4.4	starpu_mic_func_t	230
27.13.4.5	starpu_scc_func_t	230
27.13.4.6	starpu_mic_kernel_t	230
27.13.4.7	starpu_scc_kernel_t	230
27.13.5	Enumeration Type Documentation	230
27.13.5.1	starpu_codelet_type	230
27.13.5.2	starpu_task_status	230
27.13.6	Function Documentation	231
27.13.6.1	starpu_codelet_init	231
27.13.6.2	starpu_task_init	231
27.13.6.3	starpu_task_create	231
27.13.6.4	starpu_task_dup	231
27.13.6.5	starpu_task_clean	231
27.13.6.6	starpu_task_destroy	231
27.13.6.7	starpu_task_wait	231
27.13.6.8	starpu_task_submit	231
27.13.6.9	starpu_task_submit_to_ctx	232
27.13.6.10	starpu_task_wait_for_all	232
27.13.6.11	starpu_task_wait_for_all_in_ctx	232
27.13.6.12	starpu_task_nready	232
27.13.6.13	starpu_task_nsubmitted	232
27.13.6.14	starpu_task_get_current	232
27.13.6.15	starpu_codelet_display_stats	232
27.13.6.16	starpu_task_wait_for_no_ready	232
27.13.6.17	starpu_task_set_implementation	232
27.13.6.18	starpu_task_get_implementation	233
27.13.6.19	starpu_create_sync_task	233
27.14	Insert_Task	233

27.14.1 Detailed Description	233
27.14.2 Macro Definition Documentation	233
27.14.2.1 STARPU_VALUE	233
27.14.2.2 STARPU_CALLBACK	233
27.14.2.3 STARPU_CALLBACK_WITH_ARG	234
27.14.2.4 STARPU_CALLBACK_ARG	234
27.14.2.5 STARPU_PRIORITY	234
27.14.2.6 STARPU_DATA_ARRAY	234
27.14.2.7 STARPU_DATA_MODE_ARRAY	234
27.14.2.8 STARPU_EXECUTE_ON_WORKER	234
27.14.2.9 STARPU_WORKER_ORDER	234
27.14.2.10 STARPU_TAG	234
27.14.2.11 STARPU_TAG_ONLY	234
27.14.2.12 STARPU_FLOPS	234
27.14.2.13 STARPU_SCHED_CTX	234
27.14.3 Function Documentation	235
27.14.3.1 starpu_insert_task	235
27.14.3.2 starpu_task_insert	235
27.14.3.3 starpu_codelet_pack_args	235
27.14.3.4 starpu_codelet_unpack_args	235
27.14.3.5 starpu_task_build	235
27.15 Explicit Dependencies	235
27.15.1 Detailed Description	236
27.15.2 Typedef Documentation	236
27.15.2.1 starpu_tag_t	236
27.15.3 Function Documentation	236
27.15.3.1 starpu_task_declare_deps_array	236
27.15.3.2 starpu_tag_declare_deps	236
27.15.3.3 starpu_tag_declare_deps_array	237
27.15.3.4 starpu_tag_wait	237
27.15.3.5 starpu_tag_wait_array	237
27.15.3.6 starpu_tag_restart	237
27.15.3.7 starpu_tag_remove	237
27.15.3.8 starpu_tag_notify_from_apps	237
27.16 Implicit Data Dependencies	237
27.16.1 Detailed Description	238
27.16.2 Function Documentation	238
27.16.2.1 starpu_data_set_default_sequential_consistency_flag	238
27.16.2.2 starpu_data_get_default_sequential_consistency_flag	238
27.16.2.3 starpu_data_set_sequential_consistency_flag	238

27.16.2.4 starpu_data_get_sequential_consistency_flag	238
27.17 Performance Model	238
27.17.1 Detailed Description	239
27.17.2 Data Structure Documentation	239
27.17.2.1 struct starpu_perfmodel	239
27.17.2.2 struct starpu_perfmodel_regression_model	240
27.17.2.3 struct starpu_perfmodel_per_arch	241
27.17.2.4 struct starpu_perfmodel_history_list	241
27.17.2.5 struct starpu_perfmodel_history_entry	242
27.17.3 Enumeration Type Documentation	242
27.17.3.1 starpu_perfmodel_type	242
27.17.4 Function Documentation	242
27.17.4.1 starpu_perfmodel_init	242
27.17.4.2 starpu_perfmodel_free_sampling_directories	242
27.17.4.3 starpu_perfmodel_load_symbol	243
27.17.4.4 starpu_perfmodel_unload_model	243
27.17.4.5 starpu_perfmodel_debugfilepath	243
27.17.4.6 starpu_perfmodel_get_archtype_name	243
27.17.4.7 starpu_perfmodel_get_arch_name	243
27.17.4.8 starpu_worker_get_perf_archtype	243
27.17.4.9 starpu_perfmodel_list	243
27.17.4.10 starpu_perfmodel_directory	243
27.17.4.11 starpu_perfmodel_print	243
27.17.4.12 starpu_perfmodel_print_all	243
27.17.4.13 starpu_bus_print_bandwidth	243
27.17.4.14 starpu_bus_print_affinity	244
27.17.4.15 starpu_perfmodel_update_history	244
27.17.4.16 starpu_transfer_bandwidth	244
27.17.4.17 starpu_transfer_latency	244
27.17.4.18 starpu_transfer_predict	244
27.17.4.19 starpu_permodel_history_based_expected_perf	244
27.18 Profiling	244
27.18.1 Detailed Description	245
27.18.2 Data Structure Documentation	245
27.18.2.1 struct starpu_profiling_task_info	245
27.18.2.2 struct starpu_profiling_worker_info	245
27.18.2.3 struct starpu_profiling_bus_info	246
27.18.3 Macro Definition Documentation	246
27.18.3.1 STARPU_PROFILING_DISABLE	246
27.18.3.2 STARPU_PROFILING_ENABLE	246

27.18.4 Function Documentation	246
27.18.4.1 starpu_profiling_status_set	246
27.18.4.2 starpu_profiling_status_get	246
27.18.4.3 starpu_profiling_init	246
27.18.4.4 starpu_profiling_set_id	247
27.18.4.5 starpu_profiling_worker_get_info	247
27.18.4.6 starpu_bus_get_profiling_info	247
27.18.4.7 starpu_bus_get_count	247
27.18.4.8 starpu_bus_get_id	247
27.18.4.9 starpu_bus_get_src	247
27.18.4.10 starpu_bus_get_dst	247
27.18.4.11 starpu_timing_timespec_delay_us	247
27.18.4.12 starpu_timing_timespec_to_us	247
27.18.4.13 starpu_profiling_bus_helper_display_summary	247
27.18.4.14 starpu_profiling_worker_helper_display_summary	247
27.18.4.15 starpu_data_display_memory_stats	248
27.19 Theoretical Lower Bound on Execution Time	248
27.19.1 Detailed Description	248
27.19.2 Function Documentation	248
27.19.2.1 starpu_bound_start	248
27.19.2.2 starpu_bound_stop	248
27.19.2.3 starpu_bound_print_dot	248
27.19.2.4 starpu_bound_compute	248
27.19.2.5 starpu_bound_print_lp	248
27.19.2.6 starpu_bound_print_mps	248
27.19.2.7 starpu_bound_print	249
27.20 CUDA Extensions	249
27.20.1 Detailed Description	249
27.20.2 Macro Definition Documentation	249
27.20.2.1 STARPU_USE_CUDA	249
27.20.2.2 STARPU_MAXCUDADEVES	249
27.20.2.3 STARPU_CUDA_REPORT_ERROR	249
27.20.2.4 STARPU_CUBLAS_REPORT_ERROR	249
27.20.3 Function Documentation	250
27.20.3.1 starpu_cuda_get_local_stream	250
27.20.3.2 starpu_cuda_get_device_properties	250
27.20.3.3 starpu_cuda_report_error	250
27.20.3.4 starpu_cuda_copy_async_sync	250
27.20.3.5 starpu_cuda_set_device	250
27.20.3.6 starpu_cublas_init	250

27.20.3.7 starpu_cublas_shutdown	250
27.21 OpenCL Extensions	250
27.21.1 Detailed Description	252
27.21.2 Data Structure Documentation	252
27.21.2.1 struct starpu_opencil_program	252
27.21.3 Macro Definition Documentation	252
27.21.3.1 STARPU_USE_OPENCL	252
27.21.3.2 STARPU_MAXOPENCLDEVS	252
27.21.3.3 STARPU_OPENCL_DATADIR	252
27.21.3.4 STARPU_OPENCL_DISPLAY_ERROR	252
27.21.3.5 STARPU_OPENCL_REPORT_ERROR	252
27.21.3.6 STARPU_OPENCL_REPORT_ERROR_WITH_MSG	252
27.21.4 Function Documentation	253
27.21.4.1 starpu_opencil_get_context	253
27.21.4.2 starpu_opencil_get_device	253
27.21.4.3 starpu_opencil_get_queue	253
27.21.4.4 starpu_opencil_get_current_context	253
27.21.4.5 starpu_opencil_get_current_queue	253
27.21.4.6 starpu_opencil_set_kernel_args	253
27.21.4.7 starpu_opencil_load_opencil_from_file	253
27.21.4.8 starpu_opencil_load_opencil_from_string	253
27.21.4.9 starpu_opencil_unload_opencil	253
27.21.4.10 starpu_opencil_load_program_source	254
27.21.4.11 starpu_opencil_compile_opencil_from_file	254
27.21.4.12 starpu_opencil_compile_opencil_from_string	254
27.21.4.13 starpu_opencil_load_binary_opencil	254
27.21.4.14 starpu_opencil_load_kernel	254
27.21.4.15 starpu_opencil_release_kernel	254
27.21.4.16 starpu_opencil_collect_stats	254
27.21.4.17 starpu_opencil_error_string	254
27.21.4.18 starpu_opencil_display_error	255
27.21.4.19 starpu_opencil_report_error	255
27.21.4.20 starpu_opencil_allocate_memory	255
27.21.4.21 starpu_opencil_copy_ram_to_opencil	255
27.21.4.22 starpu_opencil_copy_opencil_to_ram	255
27.21.4.23 starpu_opencil_copy_opencil_to_opencil	255
27.21.4.24 starpu_opencil_copy_async_sync	255
27.22 OpenMP Runtime Support	256
27.22.1 Detailed Description	258
27.22.2 Data Structure Documentation	258

27.22.2.1 struct starpu_omp_lock_t	258
27.22.2.2 struct starpu_omp_nest_lock_t	258
27.22.2.3 struct starpu_omp_parallel_region_attr	259
27.22.2.4 struct starpu_omp_task_region_attr	259
27.22.3 Macro Definition Documentation	260
27.22.3.1 STARPU_OPENMP	260
27.22.4 Enumeration Type Documentation	260
27.22.4.1 starpu_omp_sched_value	260
27.22.4.2 starpu_omp_proc_bind_value	260
27.22.5 Function Documentation	261
27.22.5.1 starpu_omp_init	261
27.22.5.2 starpu_omp_shutdown	261
27.22.5.3 starpu_omp_parallel_region	261
27.22.5.4 starpu_omp_master	261
27.22.5.5 starpu_omp_master_inline	261
27.22.5.6 starpu_omp_barrier	261
27.22.5.7 starpu_omp_critical	262
27.22.5.8 starpu_omp_critical_inline_begin	262
27.22.5.9 starpu_omp_critical_inline_end	262
27.22.5.10 starpu_omp_single	262
27.22.5.11 starpu_omp_single_inline	262
27.22.5.12 starpu_omp_single_copyprivate	262
27.22.5.13 starpu_omp_single_copyprivate_inline_begin	263
27.22.5.14 starpu_omp_single_copyprivate_inline_end	263
27.22.5.15 starpu_omp_for	263
27.22.5.16 starpu_omp_for_inline_first	263
27.22.5.17 starpu_omp_for_inline_next	264
27.22.5.18 starpu_omp_for_alt	264
27.22.5.19 starpu_omp_for_inline_first_alt	264
27.22.5.20 starpu_omp_for_inline_next_alt	265
27.22.5.21 starpu_omp_ordered	265
27.22.5.22 starpu_omp_ordered_inline_begin	265
27.22.5.23 starpu_omp_ordered_inline_end	265
27.22.5.24 starpu_omp_sections	265
27.22.5.25 starpu_omp_sections_combined	266
27.22.5.26 starpu_omp_task_region	266
27.22.5.27 starpu_omp_taskwait	266
27.22.5.28 starpu_omp_taskgroup	266
27.22.5.29 starpu_omp_taskgroup_inline_begin	266
27.22.5.30 starpu_omp_taskgroup_inline_end	266

27.22.5.31	<code>starpu_omp_set_num_threads</code>	267
27.22.5.32	<code>starpu_omp_get_num_threads</code>	267
27.22.5.33	<code>starpu_omp_get_thread_num</code>	267
27.22.5.34	<code>starpu_omp_get_max_threads</code>	267
27.22.5.35	<code>starpu_omp_get_num_procs</code>	268
27.22.5.36	<code>starpu_omp_in_parallel</code>	268
27.22.5.37	<code>starpu_omp_set_dynamic</code>	268
27.22.5.38	<code>starpu_omp_get_dynamic</code>	268
27.22.5.39	<code>starpu_omp_set_nested</code>	269
27.22.5.40	<code>starpu_omp_get_nested</code>	269
27.22.5.41	<code>starpu_omp_get_cancellation</code>	269
27.22.5.42	<code>starpu_omp_set_schedule</code>	269
27.22.5.43	<code>starpu_omp_get_schedule</code>	269
27.22.5.44	<code>starpu_omp_get_thread_limit</code>	270
27.22.5.45	<code>starpu_omp_set_max_active_levels</code>	270
27.22.5.46	<code>starpu_omp_get_max_active_levels</code>	270
27.22.5.47	<code>starpu_omp_get_level</code>	270
27.22.5.48	<code>starpu_omp_get_ancestor_thread_num</code>	271
27.22.5.49	<code>starpu_omp_get_team_size</code>	271
27.22.5.50	<code>starpu_omp_get_active_level</code>	271
27.22.5.51	<code>starpu_omp_in_final</code>	271
27.22.5.52	<code>starpu_omp_get_proc_bind</code>	271
27.22.5.53	<code>starpu_omp_set_default_device</code>	272
27.22.5.54	<code>starpu_omp_get_default_device</code>	272
27.22.5.55	<code>starpu_omp_get_num_devices</code>	272
27.22.5.56	<code>starpu_omp_get_num_teams</code>	272
27.22.5.57	<code>starpu_omp_get_team_num</code>	272
27.22.5.58	<code>starpu_omp_is_initial_device</code>	273
27.22.5.59	<code>starpu_omp_init_lock</code>	273
27.22.5.60	<code>starpu_omp_destroy_lock</code>	273
27.22.5.61	<code>starpu_omp_set_lock</code>	273
27.22.5.62	<code>starpu_omp_unset_lock</code>	273
27.22.5.63	<code>starpu_omp_test_lock</code>	274
27.22.5.64	<code>starpu_omp_init_nest_lock</code>	274
27.22.5.65	<code>starpu_omp_destroy_nest_lock</code>	274
27.22.5.66	<code>starpu_omp_set_nest_lock</code>	274
27.22.5.67	<code>starpu_omp_unset_nest_lock</code>	275
27.22.5.68	<code>starpu_omp_test_nest_lock</code>	275
27.22.5.69	<code>starpu_omp_atomic_fallback_inline_begin</code>	275
27.22.5.70	<code>starpu_omp_atomic_fallback_inline_end</code>	275

27.22.5.71	starpu_omp_get_wtime	276
27.22.5.72	starpu_omp_get_wtick	276
27.22.5.73	starpu_omp_vector_annotate	276
27.23	MIC Extensions	276
27.23.1	Detailed Description	277
27.23.2	Macro Definition Documentation	277
27.23.2.1	STARPU_USE_MIC	277
27.23.2.2	STARPU_MAXMICDEVS	277
27.23.3	Typedef Documentation	277
27.23.3.1	starpu_mic_func_symbol_t	277
27.23.4	Function Documentation	277
27.23.4.1	starpu_mic_register_kernel	277
27.23.4.2	starpu_mic_get_kernel	277
27.24	SCC Extensions	277
27.24.1	Detailed Description	278
27.24.2	Macro Definition Documentation	278
27.24.2.1	STARPU_USE_SCC	278
27.24.2.2	STARPU_MAXSCCDEVS	278
27.24.3	Typedef Documentation	278
27.24.3.1	starpu_scc_func_symbol_t	278
27.24.4	Function Documentation	278
27.24.4.1	starpu_scc_register_kernel	278
27.24.4.2	starpu_scc_get_kernel	278
27.25	Miscellaneous Helpers	278
27.25.1	Detailed Description	278
27.25.2	Function Documentation	278
27.25.2.1	starpu_data_cpy	278
27.25.2.2	starpu_execute_on_each_worker	279
27.25.2.3	starpu_execute_on_each_worker_ex	279
27.25.2.4	starpu_execute_on_specific_workers	279
27.25.2.5	starpu_timing_now	279
27.26	FxT Support	279
27.26.1	Detailed Description	279
27.26.2	Data Structure Documentation	279
27.26.2.1	struct starpu_fxt_codelet_event	279
27.26.2.2	struct starpu_fxt_options	280
27.26.3	Function Documentation	280
27.26.3.1	starpu_fxt_options_init	280
27.26.3.2	starpu_fxt_generate_trace	281
27.26.3.3	starpu_fxt_start_profiling	281

27.26.3.4 starpu_fxt_stop_profiling	281
27.26.3.5 starpu_fxt_write_data_trace	281
27.26.3.6 starpu_fxt_trace_user_event	281
27.27 FFT Support	281
27.27.1 Detailed Description	281
27.27.2 Function Documentation	281
27.27.2.1 starpufft_malloc	281
27.27.2.2 starpufft_free	281
27.27.2.3 starpufft_plan_dft_1d	282
27.27.2.4 starpufft_plan_dft_2d	282
27.27.2.5 starpufft_start	282
27.27.2.6 starpufft_start_handle	282
27.27.2.7 starpufft_execute	282
27.27.2.8 starpufft_execute_handle	282
27.27.2.9 starpufft_cleanup	282
27.27.2.10 starpufft_destroy_plan	282
27.28 MPI Support	282
27.28.1 Detailed Description	284
27.28.2 Macro Definition Documentation	284
27.28.2.1 STARPU_USE_MPI	284
27.28.2.2 STARPU_EXECUTE_ON_NODE	284
27.28.2.3 STARPU_EXECUTE_ON_DATA	284
27.28.3 Function Documentation	284
27.28.3.1 starpu_mpi_init	284
27.28.3.2 starpu_mpi_initialize	284
27.28.3.3 starpu_mpi_initialize_extended	284
27.28.3.4 starpu_mpi_shutdown	284
27.28.3.5 starpu_mpi_comm_amounts_retrieve	285
27.28.3.6 starpu_mpi_send	285
27.28.3.7 starpu_mpi_recv	285
27.28.3.8 starpu_mpi_isend	285
27.28.3.9 starpu_mpi_irecv	285
27.28.3.10 starpu_mpi_isend_detached	285
27.28.3.11 starpu_mpi_irecv_detached	285
27.28.3.12 starpu_mpi_irecv_detached_sequential_consistency	285
27.28.3.13 starpu_mpi_issend	286
27.28.3.14 starpu_mpi_wait	286
27.28.3.15 starpu_mpi_test	286
27.28.3.16 starpu_mpi_barrier	286
27.28.3.17 starpu_mpi_isend_detached_unlock_tag	286

27.28.3.18	starpu_mpi_irecv_detached_unlock_tag	286
27.28.3.19	starpu_mpi_isend_array_detached_unlock_tag	286
27.28.3.20	starpu_mpi_irecv_array_detached_unlock_tag	287
27.28.3.21	starpu_mpi_get_communication_tag	287
27.28.3.22	starpu_mpi_set_communication_tag	287
27.28.3.23	starpu_mpi_cache_flush	287
27.28.3.24	starpu_mpi_cache_flush_all_data	287
27.28.3.25	starpu_data_set_tag	287
27.28.3.26	starpu_data_get_tag	287
27.28.3.27	starpu_mpi_data_register	287
27.28.3.28	starpu_data_set_rank	287
27.28.3.29	starpu_data_get_rank	287
27.28.3.30	starpu_mpi_insert_task	288
27.28.3.31	starpu_mpi_task_insert	288
27.28.3.32	starpu_mpi_task_build	288
27.28.3.33	starpu_mpi_task_post_build	288
27.28.3.34	starpu_mpi_get_data_on_node	288
27.28.3.35	starpu_mpi_get_data_on_node_detached	289
27.28.3.36	starpu_mpi_node_selection_get_default_policy	289
27.28.3.37	starpu_mpi_node_selection_set_default_policy	289
27.28.3.38	starpu_mpi_redux_data	289
27.28.3.39	starpu_mpi_scatter_detached	289
27.28.3.40	starpu_mpi_gather_detached	289
27.29	Task Bundles	289
27.29.1	Detailed Description	290
27.29.2	Typedef Documentation	290
27.29.2.1	starpu_task_bundle_t	290
27.29.3	Function Documentation	290
27.29.3.1	starpu_task_bundle_create	290
27.29.3.2	starpu_task_bundle_insert	290
27.29.3.3	starpu_task_bundle_remove	290
27.29.3.4	starpu_task_bundle_close	290
27.29.3.5	starpu_task_bundle_expected_length	290
27.29.3.6	starpu_task_bundle_expected_power	290
27.29.3.7	starpu_task_bundle_expected_data_transfer_time	291
27.30	Task Lists	291
27.30.1	Detailed Description	291
27.30.2	Data Structure Documentation	291
27.30.2.1	struct starpu_task_list	291
27.30.3	Function Documentation	291

27.30.3.1	starpu_task_list_init	291
27.30.3.2	starpu_task_list_push_front	292
27.30.3.3	starpu_task_list_push_back	292
27.30.3.4	starpu_task_list_front	292
27.30.3.5	starpu_task_list_back	292
27.30.3.6	starpu_task_list_empty	292
27.30.3.7	starpu_task_list_erase	292
27.30.3.8	starpu_task_list_pop_front	292
27.30.3.9	starpu_task_list_pop_back	292
27.30.3.10	starpu_task_list_begin	292
27.30.3.11	starpu_task_list_next	292
27.31	Parallel Tasks	292
27.31.1	Detailed Description	293
27.31.2	Function Documentation	293
27.31.2.1	starpu_combined_worker_get_size	293
27.31.2.2	starpu_combined_worker_get_rank	293
27.31.2.3	starpu_combined_worker_get_count	293
27.31.2.4	starpu_combined_worker_get_id	293
27.31.2.5	starpu_combined_worker_assign_workerid	293
27.31.2.6	starpu_combined_worker_get_description	293
27.31.2.7	starpu_combined_worker_can_execute_task	293
27.31.2.8	starpu_parallel_task_barrier_init	293
27.31.2.9	starpu_parallel_task_barrier_init_n	293
27.32	Running Drivers	294
27.32.1	Detailed Description	294
27.32.2	Function Documentation	294
27.32.2.1	starpu_driver_run	294
27.32.2.2	starpu_driver_init	294
27.32.2.3	starpu_driver_run_once	294
27.32.2.4	starpu_driver_deinit	294
27.32.2.5	starpu_drivers_request_termination	294
27.33	Expert Mode	294
27.33.1	Detailed Description	294
27.33.2	Function Documentation	294
27.33.2.1	starpu_wake_all_blocked_workers	295
27.33.2.2	starpu_progression_hook_register	295
27.33.2.3	starpu_progression_hook_deregister	295
27.34	StarPU-Top Interface	295
27.34.1	Detailed Description	296
27.34.2	Data Structure Documentation	296

27.34.2.1 struct starpu_top_data	296
27.34.2.2 struct starpu_top_param	296
27.34.3 Enumeration Type Documentation	297
27.34.3.1 starpu_top_data_type	297
27.34.3.2 starpu_top_param_type	298
27.34.3.3 starpu_top_message_type	298
27.34.4 Function Documentation	298
27.34.4.1 starpu_top_add_data_boolean	298
27.34.4.2 starpu_top_add_data_integer	298
27.34.4.3 starpu_top_add_data_float	298
27.34.4.4 starpu_top_register_parameter_boolean	298
27.34.4.5 starpu_top_register_parameter_float	299
27.34.4.6 starpu_top_register_parameter_integer	299
27.34.4.7 starpu_top_register_parameter_enum	299
27.34.4.8 starpu_top_init_and_wait	299
27.34.4.9 starpu_top_update_parameter	299
27.34.4.10 starpu_top_update_data_boolean	299
27.34.4.11 starpu_top_update_data_integer	299
27.34.4.12 starpu_top_update_data_float	299
27.34.4.13 starpu_top_task_prevision	299
27.34.4.14 starpu_top_debug_log	299
27.34.4.15 starpu_top_debug_lock	300
27.35 Scheduling Contexts	300
27.35.1 Detailed Description	301
27.35.2 Data Structure Documentation	301
27.35.2.1 struct starpu_sched_ctx_performance_counters	301
27.35.3 Macro Definition Documentation	302
27.35.3.1 STARPU_SCHED_CTX_POLICY_NAME	302
27.35.3.2 STARPU_SCHED_CTX_POLICY_STRUCT	302
27.35.3.3 STARPU_SCHED_CTX_POLICY_MIN_PRIO	302
27.35.3.4 STARPU_SCHED_CTX_POLICY_MAX_PRIO	302
27.35.3.5 STARPU_MIN_PRIO	302
27.35.3.6 STARPU_MAX_PRIO	302
27.35.3.7 STARPU_DEFAULT_PRIO	303
27.35.4 Function Documentation	303
27.35.4.1 starpu_sched_ctx_create	303
27.35.4.2 starpu_sched_ctx_create_inside_interval	303
27.35.4.3 starpu_sched_ctx_register_close_callback	303
27.35.4.4 starpu_sched_ctx_add_workers	303
27.35.4.5 starpu_sched_ctx_remove_workers	303

27.35.4.6	starpu_sched_ctx_display_workers	303
27.35.4.7	starpu_sched_ctx_delete	303
27.35.4.8	starpu_sched_ctx_set_inheritor	304
27.35.4.9	starpu_sched_ctx_set_context	304
27.35.4.10	starpu_sched_ctx_get_context	304
27.35.4.11	starpu_sched_ctx_stop_task_submission	304
27.35.4.12	starpu_sched_ctx_finished_submit	304
27.35.4.13	starpu_sched_ctx_get_workers_list	304
27.35.4.14	starpu_sched_ctx_get_nworkers	304
27.35.4.15	starpu_sched_ctx_get_nshared_workers	304
27.35.4.16	starpu_sched_ctx_contains_worker	304
27.35.4.17	starpu_sched_ctx_worker_get_id	304
27.35.4.18	starpu_sched_ctx_overlapping_ctxs_on_worker	304
27.35.4.19	starpu_sched_ctx_set_min_priority	305
27.35.4.20	starpu_sched_ctx_set_max_priority	305
27.35.4.21	starpu_sched_ctx_get_min_priority	305
27.35.4.22	starpu_sched_ctx_get_max_priority	305
27.35.4.23	starpu_sched_ctx_min_priority_is_set	305
27.35.4.24	starpu_sched_ctx_max_priority_is_set	305
27.35.4.25	starpu_sched_ctx_create_worker_collection	305
27.35.4.26	starpu_sched_ctx_delete_worker_collection	305
27.35.4.27	starpu_sched_ctx_get_worker_collection	305
27.35.4.28	starpu_sched_ctx_set_perf_counters	305
27.35.4.29	starpu_sched_ctx_call_pushed_task_cb	305
27.35.4.30	starpu_sched_ctx_notify_hypervisor_exists	306
27.35.4.31	starpu_sched_ctx_check_if_hypervisor_exists	306
27.35.4.32	starpu_sched_ctx_set_policy_data	306
27.35.4.33	starpu_sched_ctx_get_policy_data	306
27.35.4.34	starpu_sched_ctx_exec_parallel_code	306
27.35.4.35	starpu_sched_ctx_get_nready_tasks	306
27.35.4.36	starpu_sched_ctx_get_nready_flops	306
27.36	Scheduling Policy	306
27.36.1	Detailed Description	307
27.36.2	Data Structure Documentation	307
27.36.2.1	struct starpu_sched_policy	307
27.36.3	Function Documentation	308
27.36.3.1	starpu_sched_get_predefined_policies	308
27.36.3.2	starpu_worker_get_sched_condition	309
27.36.3.3	starpu_sched_set_min_priority	309
27.36.3.4	starpu_sched_set_max_priority	309

27.36.3.5 starpu_sched_get_min_priority	309
27.36.3.6 starpu_sched_get_max_priority	309
27.36.3.7 starpu_push_local_task	309
27.36.3.8 starpu_push_task_end	309
27.36.3.9 starpu_worker_can_execute_task	309
27.36.3.10 starpu_worker_can_execute_task_impl	309
27.36.3.11 starpu_worker_can_execute_task_first_impl	310
27.36.3.12 starpu_task_footprint	310
27.36.3.13 starpu_task_data_footprint	310
27.36.3.14 starpu_task_expected_length	310
27.36.3.15 starpu_worker_get_relative_speedup	310
27.36.3.16 starpu_task_expected_data_transfer_time	310
27.36.3.17 starpu_data_expected_transfer_time	310
27.36.3.18 starpu_task_expected_power	310
27.36.3.19 starpu_task_expected_conversion_time	310
27.36.3.20 starpu_get_prefetch_flag	310
27.36.3.21 starpu_prefetch_task_input_on_node	311
27.36.3.22 starpu_sched_ctx_worker_shares_tasks_lists	311
27.37 Tree	311
27.37.1 Detailed Description	311
27.37.2 Data Structure Documentation	311
27.37.2.1 struct starpu_tree	311
27.37.3 Function Documentation	311
27.37.3.1 starpu_tree_reset_visited	311
27.37.3.2 starpu_tree_insert	312
27.37.3.3 starpu_tree_get	312
27.37.3.4 starpu_tree_get_neighbour	312
27.37.3.5 starpu_tree_free	312
27.38 Scheduling Context Hypervisor - Regular usage	312
27.38.1 Detailed Description	313
27.38.2 Macro Definition Documentation	313
27.38.2.1 SC_HYPERVISOR_MAX_IDLE	313
27.38.2.2 SC_HYPERVISOR_PRIORITY	313
27.38.2.3 SC_HYPERVISOR_MIN_WORKERS	313
27.38.2.4 SC_HYPERVISOR_MAX_WORKERS	313
27.38.2.5 SC_HYPERVISOR_GRANULARITY	313
27.38.2.6 SC_HYPERVISOR_FIXED_WORKERS	313
27.38.2.7 SC_HYPERVISOR_MIN_TASKS	313
27.38.2.8 SC_HYPERVISOR_NEW_WORKERS_MAX_IDLE	314
27.38.2.9 SC_HYPERVISOR_TIME_TO_APPLY	314

27.38.2.10	SC_HYPERVISOR_ISPEED_W_SAMPLE	314
27.38.2.11	SC_HYPERVISOR_ISPEED_CTX_SAMPLE	314
27.38.2.12	SC_HYPERVISOR_NULL	314
27.38.3	Function Documentation	314
27.38.3.1	sc_hypervisor_init	314
27.38.3.2	sc_hypervisor_shutdown	314
27.38.3.3	sc_hypervisor_register_ctx	314
27.38.3.4	sc_hypervisor_unregister_ctx	315
27.38.3.5	sc_hypervisor_resize_ctxs	315
27.38.3.6	sc_hypervisor_stop_resize	315
27.38.3.7	sc_hypervisor_start_resize	315
27.38.3.8	sc_hypervisor_get_policy	315
27.38.3.9	sc_hypervisor_add_workers_to_sched_ctx	315
27.38.3.10	sc_hypervisor_remove_workers_from_sched_ctx	315
27.38.3.11	sc_hypervisor_move_workers	315
27.38.3.12	sc_hypervisor_size_ctxs	315
27.38.3.13	sc_hypervisor_set_type_of_task	315
27.38.3.14	sc_hypervisor_update_diff_total_flops	315
27.38.3.15	sc_hypervisor_update_diff_elapsed_flops	316
27.38.3.16	sc_hypervisor_ctl	316
27.39	Scheduling Context Hypervisor - Building a new resizing policy	316
27.39.1	Detailed Description	316
27.39.2	Data Structure Documentation	316
27.39.2.1	struct sc_hypervisor_policy	316
27.39.2.2	struct sc_hypervisor_policy_config	317
27.39.2.3	struct sc_hypervisor_wrapper	318
27.39.2.4	struct sc_hypervisor_resize_ack	320
27.39.2.5	struct sc_hypervisor_policy_task_pool	320
27.39.3	Function Documentation	320
27.39.3.1	sc_hypervisor_post_resize_request	320
27.39.3.2	sc_hypervisor_get_size_req	320
27.39.3.3	sc_hypervisor_save_size_req	321
27.39.3.4	sc_hypervisor_free_size_req	321
27.39.3.5	sc_hypervisor_can_resize	321
27.39.3.6	sc_hypervisor_get_config	321
27.39.3.7	sc_hypervisor_set_config	321
27.39.3.8	sc_hypervisor_get_sched_ctxs	321
27.39.3.9	sc_hypervisor_get_nsched_ctxs	321
27.39.3.10	sc_hypervisor_get_wrapper	321
27.39.3.11	sc_hypervisor_get_elapsed_flops_per_sched_ctx	321

27.40	Modularized Scheduler Interface	321
27.40.1	Detailed Description	324
27.40.2	Data Structure Documentation	324
27.40.2.1	struct starpu_sched_component	324
27.40.2.2	struct starpu_sched_tree	326
27.40.2.3	struct starpu_sched_component_composed_recipe	326
27.40.2.4	struct starpu_sched_component_specs	326
27.40.3	Macro Definition Documentation	327
27.40.3.1	STARPU_SCHED_COMPONENT_IS_HOMOGENEOUS	327
27.40.3.2	STARPU_SCHED_COMPONENT_IS_SINGLE_MEMORY_NODE	327
27.40.4	Enumeration Type Documentation	327
27.40.4.1	starpu_sched_component_properties	327
27.40.5	Function Documentation	327
27.40.5.1	starpu_sched_tree_create	327
27.40.5.2	starpu_sched_tree_destroy	327
27.40.5.3	starpu_sched_tree_update_workers	327
27.40.5.4	starpu_sched_tree_update_workers_in_ctx	328
27.40.5.5	starpu_sched_tree_push_task	328
27.40.5.6	starpu_sched_tree_pop_task	328
27.40.5.7	starpu_sched_tree_add_workers	328
27.40.5.8	starpu_sched_tree_remove_workers	328
27.40.5.9	starpu_sched_component_create	328
27.40.5.10	starpu_sched_component_destroy	328
27.40.5.11	starpu_sched_component_destroy_rec	328
27.40.5.12	starpu_sched_component_can_execute_task	328
27.40.5.13	starpu_sched_component_execute_preds	328
27.40.5.14	starpu_sched_component_transfer_length	329
27.40.5.15	starpu_sched_component_worker_get	329
27.40.5.16	starpu_sched_component_worker_get_workerid	329
27.40.5.17	starpu_sched_component_is_worker	329
27.40.5.18	starpu_sched_component_is_simple_worker	329
27.40.5.19	starpu_sched_component_is_combined_worker	329
27.40.5.20	starpu_sched_component_worker_pre_exec_hook	329
27.40.5.21	starpu_sched_component_worker_post_exec_hook	329
27.40.5.22	starpu_sched_component_fifo_create	329
27.40.5.23	starpu_sched_component_is_fifo	329
27.40.5.24	starpu_sched_component_prio_create	329
27.40.5.25	starpu_sched_component_is_prio	330
27.40.5.26	starpu_sched_tree_work_stealing_push_task	330
27.40.5.27	starpu_sched_component_is_work_stealing	330

27.40.5.28	starpu_sched_component_is_random	330
27.40.5.29	starpu_sched_component_is_eager	330
27.40.5.30	starpu_sched_component_is_eager_calibration	330
27.40.5.31	starpu_sched_component_mct_create	330
27.40.5.32	starpu_sched_component_is_mct	330
27.40.5.33	starpu_sched_component_heft_create	330
27.40.5.34	starpu_sched_component_is_heft	330
27.40.5.35	starpu_sched_component_perfmodel_select_create	330
27.40.5.36	starpu_sched_component_is_perfmodel_select	331
27.40.5.37	starpu_sched_component_composed_recipe_create	331
27.40.5.38	starpu_sched_component_composed_recipe_create_singleton	331
27.40.5.39	starpu_sched_component_composed_recipe_add	331
27.40.5.40	starpu_sched_component_composed_recipe_destroy	331
27.40.5.41	starpu_sched_component_composed_component_create	331
27.40.5.42	starpu_sched_component_make_scheduler	331
28	File Index	333
28.1	File List	333
29	File Documentation	335
29.1	starpu.h File Reference	335
29.2	starpu_bitmap.h File Reference	336
29.3	starpu_bound.h File Reference	336
29.4	starpu_config.h File Reference	337
29.5	starpu_cublas.h File Reference	338
29.6	starpu_cuda.h File Reference	338
29.7	starpu_data.h File Reference	339
29.8	starpu_data_filters.h File Reference	341
29.9	starpu_data_interfaces.h File Reference	342
29.10	starpu_deprecated_api.h File Reference	346
29.11	starpu_disk.h File Reference	346
29.12	starpu_driver.h File Reference	347
29.13	starpu_expert.h File Reference	347
29.14	starpu_fxt.h File Reference	347
29.15	starpu_hash.h File Reference	348
29.16	starpu_mic.h File Reference	348
29.17	starpu_opencl.h File Reference	349
29.18	starpu_openmp.h File Reference	350
29.19	starpu_perfmodel.h File Reference	352
29.19.1	Data Structure Documentation	354
29.19.1.1	struct starpu_perfmodel_device	354

29.19.1.2 struct starpu_perfmodel_arch	354
29.20starpu_profiling.h File Reference	354
29.21starpu_rand.h File Reference	355
29.22starpu_scc.h File Reference	355
29.23starpu_sched_ctx.h File Reference	356
29.24starpu_sched_ctx_hypervisor.h File Reference	357
29.25starpu_scheduler.h File Reference	358
29.26starpu_sink.h File Reference	358
29.27starpu_stdlib.h File Reference	359
29.28starpu_task.h File Reference	359
29.28.1 Macro Definition Documentation	361
29.28.1.1 STARPU_CUDA_ASYNC	361
29.28.1.2 STARPU_OPENCL_ASYNC	361
29.29starpu_task_bundle.h File Reference	361
29.30starpu_task_list.h File Reference	361
29.31starpu_task_util.h File Reference	362
29.32starpu_thread.h File Reference	363
29.32.1 Data Structure Documentation	364
29.32.1.1 struct starpu_pthread_barrier_t	364
29.32.1.2 struct starpu_pthread_spinlock_t	365
29.33starpu_thread_util.h File Reference	365
29.34starpu_top.h File Reference	366
29.35starpu_tree.h File Reference	367
29.36starpu_util.h File Reference	367
29.37starpu_worker.h File Reference	368
29.38starpu_mpi.h File Reference	369
29.39sc_hypervisor.h File Reference	370
29.40sc_hypervisor_config.h File Reference	371
29.41sc_hypervisor_lp.h File Reference	372
29.42sc_hypervisor_monitoring.h File Reference	373
29.43sc_hypervisor_policy.h File Reference	373
29.43.1 Data Structure Documentation	374
29.43.1.1 struct types_of_workers	374
30 Deprecated List	375
VI Appendix	377
31 Full Source Code for the 'Scaling a Vector' Example	379
31.1 Main Application	379

CONTENTS	1
31.2 CPU Kernel	380
31.3 CUDA Kernel	381
31.4 OpenCL Kernel	381
31.4.1 Invoking the Kernel	381
31.4.2 Source of the Kernel	382
32 GNU Free Documentation License	383
32.1 ADDENDUM: How to use this License for your documents	387
VII Index	389

Chapter 1

Introduction

1.1 Motivation

The use of specialized hardware such as accelerators or coprocessors offers an interesting approach to overcome the physical limits encountered by processor architects. As a result, many machines are now equipped with one or several accelerators (e.g. a GPU), in addition to the usual processor(s). While a lot of efforts have been devoted to offload computation onto such accelerators, very little attention has been paid to portability concerns on the one hand, and to the possibility of having heterogeneous accelerators and processors to interact on the other hand.

StarPU is a runtime system that offers support for heterogeneous multicore architectures, it not only offers a unified view of the computational resources (i.e. CPUs and accelerators at the same time), but it also takes care of efficiently mapping and executing tasks onto an heterogeneous machine while transparently handling low-level issues such as data transfers in a portable fashion.

1.2 StarPU in a Nutshell

StarPU is a software tool aiming to allow programmers to exploit the computing power of the available CPUs and GPUs, while relieving them from the need to specially adapt their programs to the target machine and processing units.

At the core of StarPU is its run-time support library, which is responsible for scheduling application-provided tasks on heterogeneous CPU/GPU machines. In addition, StarPU comes with programming language support, in the form of extensions to languages of the C family ([C Extensions](#)), as well as an OpenCL front-end ([SOCL OpenCL Extensions](#)).

StarPU's run-time and programming language extensions support a task-based programming model. Applications submit computational tasks, with CPU and/or GPU implementations, and StarPU schedules these tasks and associated data transfers on available CPUs and GPUs. The data that a task manipulates are automatically transferred among accelerators and the main memory, so that programmers are freed from the scheduling issues and technical details associated with these transfers.

StarPU takes particular care of scheduling tasks efficiently, using well-known algorithms from the literature ([Task Scheduling Policy](#)). In addition, it allows scheduling experts, such as compiler or computational library developers, to implement custom scheduling policies in a portable fashion ([Defining A New Scheduling Policy](#)).

The remainder of this section describes the main concepts used in StarPU.

1.2.1 Codelet and Tasks

One of the StarPU primary data structures is the **codelet**. A codelet describes a computational kernel that can possibly be implemented on multiple architectures such as a CPU, a CUDA device or an OpenCL device.

Another important data structure is the **task**. Executing a StarPU task consists in applying a codelet on a data set,

on one of the architectures on which the codelet is implemented. A task thus describes the codelet that it uses, but also which data are accessed, and how they are accessed during the computation (read and/or write). StarPU tasks are asynchronous: submitting a task to StarPU is a non-blocking operation. The task structure can also specify a **callback** function that is called once StarPU has properly executed the task. It also contains optional fields that the application may use to give hints to the scheduler (such as priority levels).

By default, task dependencies are inferred from data dependency (sequential coherency) by StarPU. The application can however disable sequential coherency for some data, and dependencies can be specifically expressed. A task may be identified by a unique 64-bit number chosen by the application which we refer as a **tag**. Task dependencies can be enforced either by the means of callback functions, by submitting other tasks, or by expressing dependencies between tags (which can thus correspond to tasks that have not yet been submitted).

1.2.2 StarPU Data Management Library

Because StarPU schedules tasks at runtime, data transfers have to be done automatically and “just-in-time” between processing units, relieving application programmers from explicit data transfers. Moreover, to avoid unnecessary transfers, StarPU keeps data where it was last needed, even if it was modified there, and it allows multiple copies of the same data to reside at the same time on several processing units as long as it is not modified.

1.3 Application Taskification

TODO

1.4 Glossary

A **codelet** records pointers to various implementations of the same theoretical function.

A **memory node** can be either the main RAM, GPU-embedded memory or a disk memory.

A **bus** is a link between memory nodes.

A **data handle** keeps track of replicates of the same data (**registered** by the application) over various memory nodes. The data management library manages to keep them coherent.

The **home** memory node of a data handle is the memory node from which the data was registered (usually the main memory node).

A **task** represents a scheduled execution of a codelet on some data handles.

A **tag** is a rendez-vous point. Tasks typically have their own tag, and can depend on other tags. The value is chosen by the application.

A **worker** execute tasks. There is typically one per CPU computation core and one per accelerator (for which a whole CPU core is dedicated).

A **driver** drives a given kind of workers. There are currently CPU, CUDA, and OpenCL drivers. They usually start several workers to actually drive them.

A **performance model** is a (dynamic or static) model of the performance of a given codelet. Codelets can have execution time performance model as well as power consumption performance models.

A data **interface** describes the layout of the data: for a vector, a pointer for the start, the number of elements and the size of elements ; for a matrix, a pointer for the start, the number of elements per row, the offset between rows, and the size of each element ; etc. To access their data, codelet functions are given interfaces for the local memory node replicates of the data handles of the scheduled task.

Partitioning data means dividing the data of a given data handle (called **father**) into a series of **children** data handles which designate various portions of the former.

A **filter** is the function which computes children data handles from a father data handle, and thus describes how the partitioning should be done (horizontal, vertical, etc.)

Acquiring a data handle can be done from the main application, to safely access the data of a data handle from its home node, without having to unregister it.

1.5 Research Papers

Research papers about StarPU can be found at <http://runtime.bordeaux.inria.fr/Publis/Keyword/STARPU.html>.

A good overview is available in the research report at <http://hal.archives-ouvertes.fr/inria-00467677>.

1.6 StarPU Applications

You can first have a look at the chapters [Basic Examples](#) and [Advanced Examples](#). A tutorial is also installed in the directory `share/doc/starpu/tutorial/`.

Many examples are also available in the StarPU sources in the directory `examples/`. Simple examples include:

incrementer/ Trivial incrementation test.

basic_examples/ Simple documented Hello world and vector/scalar product (as shown in [Basic Examples](#)), matrix product examples (as shown in [Performance Model Example](#)), an example using the blocked matrix data interface, an example using the variable data interface, and an example using different formats on CPUs and GPUs.

matvecmult/ OpenCL example from NVidia, adapted to StarPU.

axpy/ AXPY CUBLAS operation adapted to StarPU.

fortran/ Example of Fortran bindings.

More advanced examples include:

filters/ Examples using filters, as shown in [Partitioning Data](#).

lu/ LU matrix factorization, see for instance `xlu_implicit.c`

cholesky/ Cholesky matrix factorization, see for instance `cholesky_implicit.c`.

1.7 Further Reading

The documentation chapters include

- Part 1: StarPU Basics
 - [Building and Installing StarPU](#)
 - [Basic Examples](#)
- Part 2: StarPU Quick Programming Guide
 - [Advanced Examples](#)
 - [Check List When Performance Are Not There](#)
- Part 3: StarPU Inside
 - [Tasks In StarPU](#)
 - [Data Management](#)

- [Scheduling](#)
 - [Scheduling Contexts](#)
 - [Scheduling Context Hypervisor](#)
 - [Debugging Tools](#)
 - [Online Performance Tools](#)
 - [Offline Performance Tools](#)
 - [Frequently Asked Questions](#)
- [Part 4: StarPU Extensions](#)
 - [Out Of Core](#)
 - [MPI Support](#)
 - [FFT Support](#)
 - [MIC Xeon Phi / SCC Support](#)
 - [C Extensions](#)
 - [SOCL OpenCL Extensions](#)
 - [SimGrid Support](#)
- [Part 5: StarPU Reference API](#)
 - [Execution Configuration Through Environment Variables](#)
 - [Compilation Configuration](#)
 - [ModuleDocumentation](#)
 - [FileDocumentation](#)
 - [Deprecated List](#)
- [Part: Appendix](#)
 - [Full source code for the 'Scaling a Vector' example](#)
 - [The GNU Free Documentation License](#)

Make sure to have had a look at those too!

Part I

StarPU Basics

Chapter 2

Building and Installing StarPU

2.1 Installing a Binary Package

One of the StarPU developers being a Debian Developer, the packages are well integrated and very up-to-date. To see which packages are available, simply type:

```
$ apt-cache search starpu
```

To install what you need, type for example:

```
$ sudo apt-get install libstarpu-1.2 libstarpu-dev
```

2.2 Installing from Source

StarPU can be built and installed by the standard means of the GNU autotools. The following chapter is intended to briefly remind how these tools can be used to install StarPU.

2.2.1 Optional Dependencies

The `hwloc` (<http://www.open-mpi.org/software/hwloc>) topology discovery library is not mandatory to use StarPU but strongly recommended. It allows for topology aware scheduling, which improves performance. `hwloc` is available in major free operating system distributions, and for most operating systems.

If `hwloc` is not available on your system, the option `--without-hwloc` should be explicitly given when calling the `configure` script. If `hwloc` is installed with a `pkg-config` file, no option is required, it will be detected automatically, otherwise `--with-hwloc` should be used to specify its location.

2.2.2 Getting Sources

StarPU's sources can be obtained from the download page of the StarPU website (<http://runtime.bordeaux.inria.fr/StarPU/files/>).

All releases and the development tree of StarPU are freely available on INRIA's gforge under the LGPL license. Some releases are available under the BSD license.

The latest release can be downloaded from the INRIA's gforge (http://gforge.inria.fr/frs/?group_id=1570) or directly from the StarPU download page (<http://runtime.bordeaux.inria.fr/StarPU/files/>).

The latest nightly snapshot can be downloaded from the StarPU gforge website (<http://starpu.gforge.inria.fr/testing/>).

```
$ wget http://starpu.gforge.inria.fr/testing/starpu-nightly-latest.tar.gz
```

And finally, current development version is also accessible via svn. It should be used only if you need the very latest changes (i.e. less than a day!). Note that the client side of the software Subversion can be obtained from <http://subversion.tigris.org>. If you are running on Windows, you will probably prefer to use TortoiseSVN (<http://tortoisesvn.tigris.org/>).

```
$ svn checkout svn://scm.gforge.inria.fr/svn/starpu/trunk StarPU
```

2.2.3 Configuring StarPU

Running `autogen.sh` is not necessary when using the tarball releases of StarPU. If you are using the source code from the svn repository, you first need to generate the configure scripts and the Makefiles. This requires the availability of `autoconf` and `automake` ≥ 2.60 .

```
$ ./autogen.sh
```

You then need to configure StarPU. Details about options that are useful to give to `./configure` are given in [Compilation Configuration](#).

```
$ ./configure
```

If `configure` does not detect some software or produces errors, please make sure to post the content of `config.log` when reporting the issue.

By default, the files produced during the compilation are placed in the source directory. As the compilation generates a lot of files, it is advised to put them all in a separate directory. It is then easier to cleanup, and this allows to compile several configurations out of the same source tree. For that, simply enter the directory where you want the compilation to produce its files, and invoke the `configure` script located in the StarPU source directory.

```
$ mkdir build
$ cd build
$ ../configure
```

By default, StarPU will be installed in `/usr/local/bin`, `/usr/local/lib`, etc. You can specify an installation prefix other than `/usr/local` using the option `--prefix`, for instance:

```
$ ../configure --prefix=$HOME/starpu
```

2.2.4 Building StarPU

```
$ make
```

Once everything is built, you may want to test the result. An extensive set of regression tests is provided with StarPU. Running the tests is done by calling `make check`. These tests are run every night and the result from the main profile is publicly available (<http://starpu.gforge.inria.fr/testing/>).

```
$ make check
```

2.2.5 Installing StarPU

In order to install StarPU at the location that was specified during configuration:

```
$ make install
```

Libtool interface versioning information are included in libraries names (`libstarpu-1.2.so`, `libstarpumpi-1.2.so` and `libstarpufft-1.2.so`).

2.3 Setting up Your Own Code

2.3.1 Setting Flags for Compiling, Linking and Running Applications

StarPU provides a `pkg-config` executable to obtain relevant compiler and linker flags. As compiling and linking an application against StarPU may require to use specific flags or libraries (for instance `CUDA` or `libspe2`).

If StarPU was not installed at some standard location, the path of StarPU's library must be specified in the environment variable `PKG_CONFIG_PATH` so that `pkg-config` can find it. For example if StarPU was installed in `$STARPU_PATH`:

```
$ PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$STARPU_PATH/lib/pkgconfig
```

The flags required to compile or link against StarPU are then accessible with the following commands:

```
$ pkg-config --cflags starpu-1.2 # options for the compiler
$ pkg-config --libs starpu-1.2  # options for the linker
```

Note that it is still possible to use the API provided in the version 1.0 of StarPU by calling `pkg-config` with the `starpu-1.0` package. Similar packages are provided for `starpumpi-1.0` and `starpufft-1.0`. It is also possible to use the API provided in the version 0.9 of StarPU by calling `pkg-config` with the `libstarpu` package. Similar packages are provided for `libstarpumpi` and `libstarpufft`.

Make sure that `pkg-config --libs starpu-1.2` actually produces some output before going further: `PKG_CONFIG_PATH` has to point to the place where `starpu-1.2.pc` was installed during `make install`.

Also pass the option `-static` if the application is to be linked statically.

It is also necessary to set the environment variable `LD_LIBRARY_PATH` to locate dynamic libraries at runtime.

```
$ LD_LIBRARY_PATH=$STARPU_PATH/lib:$LD_LIBRARY_PATH
```

When using a Makefile, the following lines can be added to set the options for the compiler and the linker:

```
CFLAGS      += $(shell pkg-config --cflags starpu-1.2)
LDFLAGS     += $(shell pkg-config --libs starpu-1.2)
```

2.3.2 Running a Basic StarPU Application

Basic examples using StarPU are built in the directory `examples/basic_examples/` (and installed in `$STARPU_PATH/lib/starpu/examples/`). You can for example run the example `vector_scal`.

```
$ ./examples/basic_examples/vector_scal
BEFORE: First element was 1.000000
AFTER:  First element is 3.140000
```

When StarPU is used for the first time, the directory `$STARPU_HOME/.starpu/` is created, performance models will be stored in that directory ([STARPU_HOME](#)).

Please note that buses are benchmarked when StarPU is launched for the first time. This may take a few minutes, or less if `hwloc` is installed. This step is done only once per user and per machine.

2.3.3 Running a Basic StarPU Application on Microsoft Visual C

Batch files are provided to run StarPU applications under Microsoft Visual C. They are installed in `$STARPU_PATH/bin/msvc`.

To execute a StarPU application, you first need to set the environment variable `STARPU_PATH`.

```
c:\....> cd c:\cygwin\home\ci\starpu\
c:\....> set STARPU_PATH=c:\cygwin\home\ci\starpu\
c:\....> cd bin\msvc
c:\....> starpu_open.bat starpu_simple.c
```

The batch script will run Microsoft Visual C with a basic project file to run the given application.

The batch script `starpu_clean.bat` can be used to delete all compilation generated files.

The batch script `starpu_exec.bat` can be used to compile and execute a StarPU application from the command prompt.

```
c:\....> cd c:\cygwin\home\ci\starpu\
c:\....> set STARPU_PATH=c:\cygwin\home\ci\starpu\
c:\....> cd bin\msvc
c:\....> starpu_exec.bat ..\..\..\examples\basic_examples\hello_world.c
```

```
MSVC StarPU Execution
...
/out:hello_world.exe
...
Hello world (params = {1, 2.00000})
Callback function got argument 0000042
c:\....>
```

2.3.4 Kernel Threads Started by StarPU

StarPU automatically binds one thread per CPU core. It does not use SMT/hyperthreading because kernels are usually already optimized for using a full core, and using hyperthreading would make kernel calibration rather random.

Since driving GPUs is a CPU-consuming task, StarPU dedicates one core per GPU.

While StarPU tasks are executing, the application is not supposed to do computations in the threads it starts itself, tasks should be used instead.

TODO: add a StarPU function to bind an application thread (e.g. the main thread) to a dedicated core (and thus disable the corresponding StarPU CPU worker).

2.3.5 Enabling OpenCL

When both CUDA and OpenCL drivers are enabled, StarPU will launch an OpenCL worker for NVIDIA GPUs only if CUDA is not already running on them. This design choice was necessary as OpenCL and CUDA can not run at the same time on the same NVIDIA GPU, as there is currently no interoperability between them.

To enable OpenCL, you need either to disable CUDA when configuring StarPU:

```
$ ./configure --disable-cuda
```

or when running applications:

```
$ STARPU_NCUDA=0 ./application
```

OpenCL will automatically be started on any device not yet used by CUDA. So on a machine running 4 GPUS, it is therefore possible to enable CUDA on 2 devices, and OpenCL on the 2 other devices by doing so:

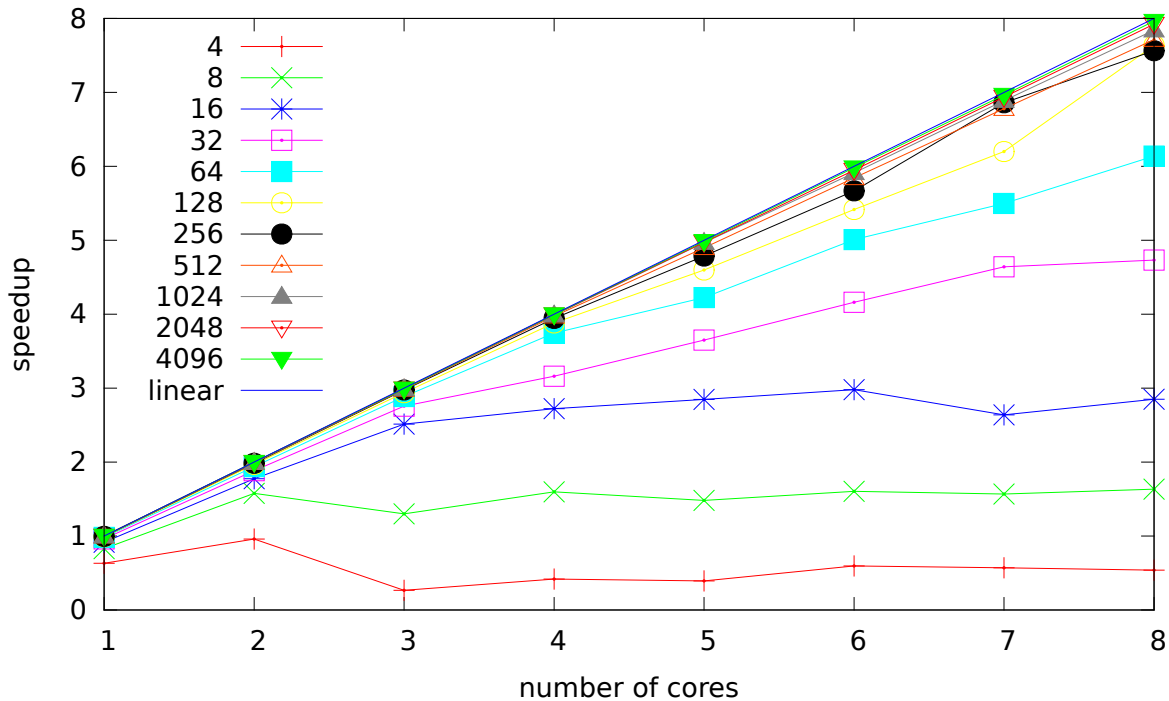
```
$ STARPU_NCUDA=2 ./application
```

2.4 Benchmarking StarPU

Some interesting benchmarks are installed among examples in `$STARPU_PATH/lib/starpu/examples/`. Make sure to try various schedulers, for instance `STARPU_SCHED=dmda`.

2.4.1 Task Size Overhead

This benchmark gives a glimpse into how long a task should be (in μ s) for StarPU overhead to be low enough to keep efficiency. Running `tasks_size_overhead.sh` generates a plot of the speedup of tasks of various sizes, depending on the number of CPUs being used.



2.4.2 Data Transfer Latency

`local_pingpong` performs a ping-pong between the first two CUDA nodes, and prints the measured latency.

2.4.3 Matrix-Matrix Multiplication

`sgemm` and `dgemm` perform a blocked matrix-matrix multiplication using BLAS and cuBLAS. They output the obtained GFlops.

2.4.4 Cholesky Factorization

`cholesky_*` perform a Cholesky factorization (single precision). They use different dependency primitives.

2.4.5 LU Factorization

`lu_*` perform an LU factorization. They use different dependency primitives.

Chapter 3

Basic Examples

3.1 Hello World Using The C Extension

This section shows how to implement a simple program that submits a task to StarPU using the StarPU C extension ([C Extensions](#)). The complete example, and additional examples, is available in the directory `gcc-plugin/examples` of the StarPU distribution. A similar example showing how to directly use the StarPU's API is shown in [Hello World Using StarPU's API](#).

GCC from version 4.5 permit to use the StarPU GCC plug-in ([C Extensions](#)). This makes writing a task both simpler and less error-prone. In a nutshell, all it takes is to declare a task, declare and define its implementations (for CPU, OpenCL, and/or CUDA), and invoke the task like a regular C function. The example below defines `my_task` which has a single implementation for CPU:

```
#include <stdio.h>

/* Task declaration. */
static void my_task (int x) __attribute__ ((task));

/* Definition of the CPU implementation of 'my_task'. */
static void my_task (int x)
{
    printf ("Hello, world! With x = %d\n", x);
}

int main ()
{
    /* Initialize StarPU. */
    #pragma starpu initialize

    /* Do an asynchronous call to 'my_task'. */
    my_task (42);

    /* Wait for the call to complete. */
    #pragma starpu wait

    /* Terminate. */
    #pragma starpu shutdown

    return 0;
}
```

The code can then be compiled and linked with GCC and the flag `-fplugin`:

```
$ gcc `pkg-config starpu-1.2 --cflags` hello-starpu.c \
    -fplugin=`pkg-config starpu-1.2 --variable=gccplugin` \
    `pkg-config starpu-1.2 --libs`
```

The code can also be compiled without the StarPU C extension and will behave as a normal sequential code.

```
$ gcc hello-starpu.c
hello-starpu.c:33:1: warning: 'task' attribute directive ignored [-Wattributes]
$ ./a.out
Hello, world! With x = 42
```

As can be seen above, the C extensions allows programmers to use StarPU tasks by essentially annotating “regular” C code.

3.2 Hello World Using StarPU’s API

This section shows how to achieve the same result as in the previous section using StarPU’s standard C API.

3.2.1 Required Headers

The header `starpu.h` should be included in any code using StarPU.

```
#include <starpu.h>
```

3.2.2 Defining A Codelet

A codelet is a structure that represents a computational kernel. Such a codelet may contain an implementation of the same kernel on different architectures (e.g. CUDA, x86, ...). For compatibility, make sure that the whole structure is properly initialized to zero, either by using the function `starpu_codelet_init()`, or by letting the compiler implicitly do it as exemplified above.

The field `starpu_codelet::nbuffers` specifies the number of data buffers that are manipulated by the codelet: here the codelet does not access or modify any data that is controlled by our data management library.

We create a codelet which may only be executed on the CPUs. When a CPU core will execute a codelet, it will call the function `cpu_func`, which *must* have the following prototype:

```
void (*cpu_func)(void *buffers[], void *cl_arg);
```

In this example, we can ignore the first argument of this function which gives a description of the input and output buffers (e.g. the size and the location of the matrices) since there is none. We also ignore the second argument which is a pointer to optional arguments for the codelet.

```
void cpu_func(void *buffers[], void *cl_arg)
{
    printf("Hello world\n");
}

struct starpu_codelet cl =
{
    .cpu_funcs = { cpu_func },
    .nbuffers = 0
};
```

3.2.3 Submitting A Task

Before submitting any tasks to StarPU, `starpu_init()` must be called. The `NULL` argument specifies that we use the default configuration. Tasks cannot be submitted after the termination of StarPU by a call to `starpu_shutdown()`.

In the example above, a task structure is allocated by a call to `starpu_task_create()`. This function only allocates and fills the corresponding structure with the default settings, but it does not submit the task to StarPU.

The field `starpu_task::cl` is a pointer to the codelet which the task will execute: in other words, the codelet structure describes which computational kernel should be offloaded on the different architectures, and the task structure is a wrapper containing a codelet and the piece of data on which the codelet should operate.

If the field `starpu_task::synchronous` is non-zero, task submission will be synchronous: the function `starpu_task_submit()` will not return until the task has been executed. Note that the function `starpu_shutdown()` does not guarantee that asynchronous tasks have been executed before it returns, `starpu_task_wait_for_all()` can be used to that effect, or data can be unregistered (`starpu_data_unregister()`), which will implicitly wait for all the tasks scheduled to work on it, unless explicitly disabled thanks to `starpu_data_set_default_sequential_consistency_flag()` or `starpu_data_set_sequential_consistency_flag()`.

```

int main(int argc, char **argv)
{
    /* initialize StarPU */
    starpu_init(NULL);

    struct starpu_task *task = starpu_task_create();

    task->cl = &cl; /* Pointer to the codelet defined above */

    /* starpu_task_submit will be a blocking call. If unset,
    starpu_task_wait() needs to be called after submitting the task. */
    task->synchronous = 1;

    /* submit the task to StarPU */
    starpu_task_submit(task);

    /* terminate StarPU */
    starpu_shutdown();

    return 0;
}

```

3.2.4 Execution Of Hello World

```

$ make hello_world
cc $(pkg-config --cflags starpu-1.2) hello_world.c -o hello_world $(pkg-config --libs starpu-1.2)
$ ./hello_world
Hello world

```

3.2.5 Passing Arguments To The Codelet

The optional field `starpu_task::cl_arg` field is a pointer to a buffer (of size `starpu_task::cl_arg_size`) with some parameters for the kernel described by the codelet. For instance, if a codelet implements a computational kernel that multiplies its input vector by a constant, the constant could be specified by the means of this buffer, instead of registering it as a StarPU data. It must however be noted that StarPU avoids making copy whenever possible and rather passes the pointer as such, so the buffer which is pointed at must be kept allocated until the task terminates, and if several tasks are submitted with various parameters, each of them must be given a pointer to their own buffer.

```

struct params
{
    int i;
    float f;
};

void cpu_func(void *buffers[], void *cl_arg)
{
    struct params *params = cl_arg;

    printf("Hello world (params = {%i, %f})\n", params->i, params->f);
}

```

As said before, the field `starpu_codelet::nbuffers` specifies the number of data buffers that are manipulated by the codelet. It does not count the argument — the parameter `cl_arg` of the function `cpu_func` — since it is not managed by our data management library, but just contains trivial parameters.

Be aware that this may be a pointer to a *copy* of the actual buffer, and not the pointer given by the programmer: if the codelet modifies this buffer, there is no guarantee that the initial buffer will be modified as well: this for instance implies that the buffer cannot be used as a synchronization medium. If synchronization is needed, data has to be registered to StarPU, see [Vector Scaling Using StarPU's API](#).

```

int main(int argc, char **argv)
{
    /* initialize StarPU */
    starpu_init(NULL);

    struct starpu_task *task = starpu_task_create();

    task->cl = &cl; /* Pointer to the codelet defined above */

    struct params params = { 1, 2.0f };
    task->cl_arg = &params;
    task->cl_arg_size = sizeof(params);
}

```

```

/* starpu_task_submit will be a blocking call */
task->synchronous = 1;

/* submit the task to StarPU */
starpu_task_submit(task);

/* terminate StarPU */
starpu_shutdown();

return 0;
}

$ make hello_world
cc $(pkg-config --cflags starpu-1.2) hello_world.c -o hello_world $(pkg-config --libs starpu-1.2)
$ ./hello_world
Hello world (params = {1, 2.000000} )

```

3.2.6 Defining A Callback

Once a task has been executed, an optional callback function `starpu_task::callback_func` is called when defined. While the computational kernel could be offloaded on various architectures, the callback function is always executed on a CPU. The pointer `starpu_task::callback_arg` is passed as an argument of the callback function. The prototype of a callback function must be:

```

void (*callback_function)(void *);

void callback_func(void *callback_arg)
{
    printf("Callback function (arg %x)\n", callback_arg);
}

int main(int argc, char **argv)
{
    /* initialize StarPU */
    starpu_init(NULL);

    struct starpu_task *task = starpu_task_create();

    task->cl = &cl; /* Pointer to the codelet defined above */

    task->callback_func = callback_func;
    task->callback_arg = 0x42;

    /* starpu_task_submit will be a blocking call */
    task->synchronous = 1;

    /* submit the task to StarPU */
    starpu_task_submit(task);

    /* terminate StarPU */
    starpu_shutdown();

    return 0;
}

$ make hello_world
cc $(pkg-config --cflags starpu-1.2) hello_world.c -o hello_world $(pkg-config --libs starpu-1.2)
$ ./hello_world
Hello world
Callback function (arg 42)

```

3.2.7 Where To Execute A Codelet

```

struct starpu_codelet cl =
{
    .where = STARPU_CPU,
    .cpu_funcs = { cpu_func },
    .cpu_funcs_name = { "cpu_func" },
    .nbuffers = 0
};

```

We create a codelet which may only be executed on the CPUs. The optional field `starpu_codelet::where` is a bitmask that defines where the codelet may be executed. Here, the value `STARPU_CPU` means that only CPUs

can execute this codelet. When the optional field `starpu_codelet::where` is unset, its value is automatically set based on the availability of the different fields `XXX_funcs`.

TODO: explain `starpu_codelet::cpu_funcs_name`

3.3 Vector Scaling Using the C Extension

The previous example has shown how to submit tasks. In this section, we show how StarPU tasks can manipulate data.

We will first show how to use the C language extensions provided by the GCC plug-in ([C Extensions](#)). The complete example, and additional examples, is available in the directory `gcc-plugin/examples` of the StarPU distribution. These extensions map directly to StarPU's main concepts: tasks, task implementations for CPU, OpenCL, or CUDA, and registered data buffers. The standard C version that uses StarPU's standard C programming interface is given in [Vector Scaling Using StarPU's API](#).

First of all, the vector-scaling task and its simple CPU implementation has to be defined:

```
/* Declare the 'vector_scal' task. */
static void vector_scal (unsigned size, float vector[size],
                        float factor)
    __attribute__((task));

/* Define the standard CPU implementation. */
static void
vector_scal (unsigned size, float vector[size], float factor)
{
    unsigned i;
    for (i = 0; i < size; i++)
        vector[i] *= factor;
}
```

Next, the body of the program, which uses the task defined above, can be implemented:

```
int main (void)
{
    #pragma starpu initialize

    #define NX      0x100000
    #define FACTOR  3.14

    {
        float vector[NX]
            __attribute__((heap_allocated, registered));

        size_t i;
        for (i = 0; i < NX; i++)
            vector[i] = (float) i;

        vector_scal (NX, vector, FACTOR);
    }

    #pragma starpu wait
    /* VECTOR is automatically freed here. */

    #pragma starpu shutdown

    return valid ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

The function `main` above does several things:

- It initializes StarPU.
- It allocates `vector` in the heap; it will automatically be freed when its scope is left. Alternatively, good old `malloc` and `free` could have been used, but they are more error-prone and require more typing.
- It registers the memory pointed to by `vector`. Eventually, when OpenCL or CUDA task implementations are added, this will allow StarPU to transfer that memory region between GPUs and the main memory. Removing this `pragma` is an error.

- It invokes the task `vector_scal`. The invocation looks the same as a standard C function call. However, it is an asynchronous invocation, meaning that the actual call is performed in parallel with the caller's continuation.
- It waits for the termination of the asynchronous call `vector_scal`.
- Finally, StarPU is shut down.

The program can be compiled and linked with GCC and the flag `-fplugin`:

```
$ gcc `pkg-config starpu-1.2 --cflags` vector_scal.c \
    -fplugin=`pkg-config starpu-1.2 --variable=gccplugin` \
    `pkg-config starpu-1.2 --libs`
```

And voilà!

3.3.1 Adding an OpenCL Task Implementation

Now, this is all fine and great, but you certainly want to take advantage of these newfangled GPUs that your lab just bought, don't you?

So, let's add an OpenCL implementation of the task `vector_scal`. We assume that the OpenCL kernel is available in a file, `vector_scal_openccl_kernel.cl`, not shown here. The OpenCL task implementation is similar to that used with the standard C API ([Definition of the OpenCL Kernel](#)). It is declared and defined in our C file like this:

```
/* The OpenCL programs, loaded from 'main' (see below). */
static struct starpu_openccl_program cl_programs;

static void vector_scal_openccl (unsigned size, float vector[size],
                                float factor)
    __attribute__((task_implementation ("openccl", vector_scal)));

static void
vector_scal_openccl (unsigned size, float vector[size], float factor)
{
    int id, devid, err;
    cl_kernel kernel;
    cl_command_queue queue;
    cl_event event;

    /* VECTOR is GPU memory pointer, not a main memory pointer. */
    cl_mem val = (cl_mem) vector;

    id = starpu_worker_get_id ();
    devid = starpu_worker_get_devid (id);

    /* Prepare to invoke the kernel. In the future, this will be largely automated. */
    err = starpu_openccl_load_kernel (&kernel, &queue, &cl_programs,
                                     "vector_mult_openccl", devid);
    if (err != CL_SUCCESS)
        STARPU_OPENCCL_REPORT_ERROR (err);

    err = clSetKernelArg (kernel, 0, sizeof (size), &size);
    err |= clSetKernelArg (kernel, 1, sizeof (val), &val);
    err |= clSetKernelArg (kernel, 2, sizeof (factor), &factor);
    if (err)
        STARPU_OPENCCL_REPORT_ERROR (err);

    size_t global = 1, local = 1;
    err = clEnqueueNDRangeKernel (queue, kernel, 1, NULL, &global,
                                  &local, 0, NULL, &event);
    if (err != CL_SUCCESS)
        STARPU_OPENCCL_REPORT_ERROR (err);

    clFinish (queue);
    starpu_openccl_collect_stats (event);
    clReleaseEvent (event);

    /* Done with KERNEL. */
    starpu_openccl_release_kernel (kernel);
}
```

The OpenCL kernel itself must be loaded from `main`, sometime after the `pragma initialize`:

```
starpu_opengl_load_opengl_from_file ("vector_scal_opengl_kernel.cl",
                                     &cl_programs, "");
```

And that's it. The task `vector_scal` now has an additional implementation, for OpenCL, which StarPU's scheduler may choose to use at run-time. Unfortunately, the `vector_scal_opengl` above still has to go through the common OpenCL boilerplate; in the future, additional extensions will automate most of it.

3.3.2 Adding a CUDA Task Implementation

Adding a CUDA implementation of the task is very similar, except that the implementation itself is typically written in CUDA, and compiled with `nvcc`. Thus, the C file only needs to contain an external declaration for the task implementation:

```
extern void vector_scal_cuda (unsigned size, float vector[size],
                             float factor)
    __attribute__((task_implementation ("cuda", vector_scal)));
```

The actual implementation of the CUDA task goes into a separate compilation unit, in a `.cu` file. It is very close to the implementation when using StarPU's standard C API ([Definition of the CUDA Kernel](#)).

```
1 /* CUDA implementation of the 'vector_scal' task, to be compiled with 'nvcc'. */
2
3 #include <starpu.h>
4 #include <stdlib.h>
5
6 static __global__ void
7 vector_mult_cuda (unsigned n, float *val, float factor)
8 {
9     unsigned i = blockIdx.x * blockDim.x + threadIdx.x;
10
11     if (i < n)
12         val[i] *= factor;
13 }
14
15 /* Definition of the task implementation declared in the C file. */
16 extern "C" void
17 vector_scal_cuda (size_t size, float vector[], float factor)
18 {
19     unsigned threads_per_block = 64;
20     unsigned nblocks = (size + threads_per_block - 1) / threads_per_block;
21
22     vector_mult_cuda <<< nblocks, threads_per_block, 0,
23         starpu_cuda_get_local_stream () >>> (size, vector, factor);
24
25     cudaStreamSynchronize (starpu_cuda_get_local_stream ());
26 }
```

The complete source code, in the directory `gcc-plugin/examples/vector_scal` of the StarPU distribution, also shows how an SSE-specialized CPU task implementation can be added.

For more details on the C extensions provided by StarPU's GCC plug-in, see [C Extensions](#).

3.4 Vector Scaling Using StarPU's API

This section shows how to achieve the same result as explained in the previous section using StarPU's standard C API.

The full source code for this example is given in [Full source code for the 'Scaling a Vector' example](#).

3.4.1 Source Code of Vector Scaling

Programmers can describe the data layout of their application so that StarPU is responsible for enforcing data coherency and availability across the machine. Instead of handling complex (and non-portable) mechanisms to perform data movements, programmers only declare which piece of data is accessed and/or modified by a task,

and StarPU makes sure that when a computational kernel starts somewhere (e.g. on a GPU), its data are available locally.

Before submitting those tasks, the programmer first needs to declare the different pieces of data to StarPU using the functions `starpu*_data_register`. To ease the development of applications for StarPU, it is possible to describe multiple types of data layout. A type of data layout is called an **interface**. There are different predefined interfaces available in StarPU: here we will consider the **vector interface**.

The following lines show how to declare an array of `NX` elements of type `float` using the vector interface:

```
float vector[NX];

starpu_data_handle_t vector_handle;
starpu_vector_data_register(&vector_handle,
    STARPU_MAIN_RAM, (uintptr_t)vector, NX,
    sizeof(vector[0]));
```

The first argument, called the **data handle**, is an opaque pointer which designates the array in StarPU. This is also the structure which is used to describe which data is used by a task. The second argument is the node number where the data originally resides. Here it is `STARPU_MAIN_RAM` since the array `vector` is in the main memory. Then comes the pointer `vector` where the data can be found in main memory, the number of elements in the vector and the size of each element. The following shows how to construct a StarPU task that will manipulate the vector and a constant factor.

```
float factor = 3.14;
struct starpu_task *task = starpu_task_create();

task->cl = &cl; /* Pointer to the codelet defined below */
task->handles[0] = vector_handle; /* First parameter of the codelet */
task->cl_arg = &factor;
task->cl_arg_size = sizeof(factor);
task->synchronous = 1;

starpu_task_submit(task);
```

Since the factor is a mere constant float value parameter, it does not need a preliminary registration, and can just be passed through the pointer `starpu_task::cl_arg` like in the previous example. The vector parameter is described by its handle. `starpu_task::handles` should be set with the handles of the data, the access modes for the data are defined in the field `starpu_codelet::modes` (`STARPU_R` for read-only, `STARPU_W` for write-only and `STARPU_RW` for read and write access).

The definition of the codelet can be written as follows:

```
void scal_cpu_func(void *buffers[], void *cl_arg)
{
    unsigned i;
    float *factor = cl_arg;

    /* length of the vector */
    unsigned n = STARPU_VECTOR_GET_NX(buffers[0]);
    /* CPU copy of the vector pointer */
    float *val = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);

    for (i = 0; i < n; i++)
        val[i] *= *factor;
}

struct starpu_codelet cl =
{
    .cpu_funcs = { scal_cpu_func },
    .cpu_funcs_name = { "scal_cpu_func" },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
```

The first argument is an array that gives a description of all the buffers passed in the array `starpu_task::handles`. The size of this array is given by the field `starpu_codelet::nbuffers`. For the sake of genericity, this array contains pointers to the different interfaces describing each buffer. In the case of the **vector interface**, the location of the vector (resp. its length) is accessible in the `starpu_vector_interface::ptr` (resp. `starpu_vector_interface::nx`) of this interface. Since the vector is accessed in a read-write fashion, any modification will automatically affect future accesses to this vector made by other tasks.

The second argument of the function `scal_cpu_func` contains a pointer to the parameters of the codelet (given in `starpu_task::cl_arg`), so that we read the constant factor from this pointer.

3.4.2 Execution of Vector Scaling

```
$ make vector_scal
cc $(pkg-config --cflags starpu-1.2) vector_scal.c -o vector_scal $(pkg-config --libs starpu-1.2)
$ ./vector_scal
0.000000 3.000000 6.000000 9.000000 12.000000
```

3.5 Vector Scaling on an Hybrid CPU/GPU Machine

Contrary to the previous examples, the task submitted in this example may not only be executed by the CPUs, but also by a CUDA device.

3.5.1 Definition of the CUDA Kernel

The CUDA implementation can be written as follows. It needs to be compiled with a CUDA compiler such as `nvcc`, the NVIDIA CUDA compiler driver. It must be noted that the vector pointer returned by `STARPU_VECTOR_GET_PTR` is here a pointer in GPU memory, so that it can be passed as such to the kernel call `vector_mult_cuda`.

```
1 #include <starpu.h>
2
3 static __global__ void vector_mult_cuda(unsigned n, float *val,
4                                       float factor)
5 {
6     unsigned i = blockIdx.x*blockDim.x + threadIdx.x;
7     if (i < n)
8         val[i] *= factor;
9 }
10
11 extern "C" void scal_cuda_func(void *buffers[], void *_args)
12 {
13     float *factor = (float *)_args;
14
15     /* length of the vector */
16     unsigned n = STARPU_VECTOR_GET_NX(buffers[0]);
17     /* local copy of the vector pointer */
18     float *val = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
19     unsigned threads_per_block = 64;
20     unsigned nblocks = (n + threads_per_block - 1) / threads_per_block;
21
22     vector_mult_cuda<<<nblocks, threads_per_block, 0, starpu_cuda_get_local_stream()>>>
23         (n, val, *factor);
24
25     cudaStreamSynchronize(starpu_cuda_get_local_stream());
26 }
```

3.5.2 Definition of the OpenCL Kernel

The OpenCL implementation can be written as follows. StarPU provides tools to compile a OpenCL kernel stored in a file.

```
__kernel void vector_mult_opencl(int nx, __global float* val, float factor)
{
    const int i = get_global_id(0);
    if (i < nx) {
        val[i] *= factor;
    }
}
```

Contrary to CUDA and CPU, `STARPU_VECTOR_GET_DEV_HANDLE` has to be used, which returns a `cl_mem` (which is not a device pointer, but an OpenCL handle), which can be passed as such to the OpenCL kernel. The difference is important when using partitioning, see [Partitioning Data](#).

```

#include <starpu.h>

extern struct starpu_opencil_program programs;

void scal_opencil_func(void *buffers[], void *_args)
{
    float *factor = _args;
    int id, devid, err;
    cl_kernel kernel;
    cl_command_queue queue;
    cl_event event;

    /* length of the vector */
    unsigned n = STARPU_VECTOR_GET_NX(buffers[0]);
    /* OpenCL copy of the vector pointer */
    cl_mem val = (cl_mem)STARPU_VECTOR_GET_DEV_HANDLE(buffers[0]);

    { /* OpenCL specific code */
        id = starpu_worker_get_id();
        devid = starpu_worker_get_devid(id);

        err = starpu_opencil_load_kernel(&kernel, &queue,
                                         &programs,
                                         "vector_mult_opencil", /* Name of the codelet */
                                         devid);

        if (err != CL_SUCCESS) STARPU_OPENCIL_REPORT_ERROR(err);

        err = clSetKernelArg(kernel, 0, sizeof(n), &n);
        err |= clSetKernelArg(kernel, 1, sizeof(val), &val);
        err |= clSetKernelArg(kernel, 2, sizeof(*factor), factor);
        if (err) STARPU_OPENCIL_REPORT_ERROR(err);
    }

    { /* OpenCL specific code */
        size_t global=n;
        size_t local;
        size_t s;
        cl_device_id device;

        starpu_opencil_get_device(devid, &device);
        err = clGetKernelWorkGroupInfo(kernel, device, CL_KERNEL_WORK_GROUP_SIZE,
                                         sizeof(local), &local, &s);
        if (err != CL_SUCCESS) STARPU_OPENCIL_REPORT_ERROR(err);
        if (local > global) local=global;

        err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &global, &local, 0,
                                         NULL, &event);
        if (err != CL_SUCCESS) STARPU_OPENCIL_REPORT_ERROR(err);
    }

    { /* OpenCL specific code */
        clFinish(queue);
        starpu_opencil_collect_stats(event);
        clReleaseEvent(event);

        starpu_opencil_release_kernel(kernel);
    }
}

```

3.5.3 Definition of the Main Code

The CPU implementation is the same as in the previous section.

Here is the source of the main application. You can notice that the fields `starpu_codelet::cuda_funcs` and `starpu_codelet::opencil_funcs` are set to define the pointers to the CUDA and OpenCL implementations of the task.

```

/*
 * This example demonstrates how to use StarPU to scale an array by a factor.
 * It shows how to manipulate data with StarPU's data management library.
 * 1- how to declare a piece of data to StarPU (starpu_vector_data_register)
 * 2- how to describe which data are accessed by a task (task->handles[0])
 * 3- how a kernel can manipulate the data (buffers[0].vector.ptr)
 */
#include <starpu.h>

#define NX 2048

extern void scal_cpu_func(void *buffers[], void *_args);
extern void scal_sse_func(void *buffers[], void *_args);
extern void scal_cuda_func(void *buffers[], void *_args);
extern void scal_opencil_func(void *buffers[], void *_args);

```

```

static struct starpu_codelet cl = {
    .where = STARPU_CPU | STARPU_CUDA | STARPU_OPENCL,
    /* CPU implementation of the codelet */
    .cpu_funcs = { scal_cpu_func, scal_sse_func },
    .cpu_funcs_name = { "scal_cpu_func", "scal_sse_func" },
#ifdef STARPU_USE_CUDA
    /* CUDA implementation of the codelet */
    .cuda_funcs = { scal_cuda_func },
#endif
#ifdef STARPU_USE_OPENCL
    /* OpenCL implementation of the codelet */
    .opencl_funcs = { scal_opencl_func },
#endif
    .nbuffers = 1,
    .modes = { STARPU_RW }
};

#ifdef STARPU_USE_OPENCL
struct starpu_opencl_program programs;
#endif

int main(int argc, char **argv)
{
    /* We consider a vector of float that is initialized just as any of C
     * data */
    float vector[NX];
    unsigned i;
    for (i = 0; i < NX; i++)
        vector[i] = 1.0f;

    fprintf(stderr, "BEFORE: First element was %f\n", vector[0]);

    /* Initialize StarPU with default configuration */
    starpu_init(NULL);

#ifdef STARPU_USE_OPENCL
    starpu_opencl_load_opencl_from_file(
        "examples/basic_examples/vector_scal_opencl_kernel.cl", &programs, NULL);
#endif

    /* Tell StaPU to associate the "vector" vector with the "vector_handle"
     * identifier. When a task needs to access a piece of data, it should
     * refer to the handle that is associated to it.
     * In the case of the "vector" data interface:
     * - the first argument of the registration method is a pointer to the
     *   handle that should describe the data
     * - the second argument is the memory node where the data (ie. "vector")
     *   resides initially: STARPU_MAIN_RAM stands for an address in main memory, as
     *   opposed to an adress on a GPU for instance.
     * - the third argument is the adress of the vector in RAM
     * - the fourth argument is the number of elements in the vector
     * - the fifth argument is the size of each element.
     */
    starpu_data_handle_t vector_handle;
    starpu_vector_data_register(&vector_handle,
        STARPU_MAIN_RAM, (uintptr_t)vector,
        NX, sizeof(vector[0]));

    float factor = 3.14;

    /* create a synchronous task: any call to starpu_task_submit will block
     * until it is terminated */
    struct starpu_task *task = starpu_task_create();
    task->synchronous = 1;

    task->cl = &cl;

    /* the codelet manipulates one buffer in RW mode */
    task->handles[0] = vector_handle;

    /* an argument is passed to the codelet, beware that this is a
     * READ-ONLY buffer and that the codelet may be given a pointer to a
     * COPY of the argument */
    task->cl_arg = &factor;
    task->cl_arg_size = sizeof(factor);

    /* execute the task on any eligible computational ressource */
    starpu_task_submit(task);

    /* StarPU does not need to manipulate the array anymore so we can stop
     * monitoring it */
    starpu_data_unregister(vector_handle);

#ifdef STARPU_USE_OPENCL
    starpu_opencl_unload_opencl(&programs);
#endif
}

```

```

/* terminate StarPU, no task can be submitted after */
starpu_shutdown();

fprintf(stderr, "AFTER First element is %f\n", vector[0]);

return 0;
}

```

3.5.4 Execution of Hybrid Vector Scaling

The Makefile given at the beginning of the section must be extended to give the rules to compile the CUDA source code. Note that the source file of the OpenCL kernel does not need to be compiled now, it will be compiled at run-time when calling the function [starpu_opencil_load_opencil_from_file\(\)](#).

```

CFLAGS += $(shell pkg-config --cflags starpu-1.2)
LDLAGS += $(shell pkg-config --libs starpu-1.2)
CC      = gcc

vector_scal: vector_scal.o vector_scal_cpu.o vector_scal_cuda.o vector_scal_opencil.o

%.o: %.cu
    nvcc $(CFLAGS) $< -c $@

clean:
    rm -f vector_scal *.o

$ make

```

and to execute it, with the default configuration:

```

$ ./vector_scal
0.000000 3.000000 6.000000 9.000000 12.000000

```

or for example, by disabling CPU devices:

```

$ STARPU_NCPU=0 ./vector_scal
0.000000 3.000000 6.000000 9.000000 12.000000

```

or by disabling CUDA devices (which may permit to enable the use of OpenCL, see [Enabling OpenCL](#)) :

```

$ STARPU_NCUDA=0 ./vector_scal
0.000000 3.000000 6.000000 9.000000 12.000000

```


Part II

StarPU Quick Programming Guide

Chapter 4

Advanced Examples

TODO

Chapter 5

Check List When Performance Are Not There

TODO: improve!

Simply encapsulating application kernels into tasks already permits to seamlessly support CPU and GPUs at the same time. To achieve good performance, we give below a list of features which should be checked.

5.1 Data Related Features That May Improve Performance

link to [Data Management](#)

link to [Data Prefetch](#)

5.2 Task Related Features That May Improve Performance

link to [Task Granularity](#)

link to [Task Submission](#)

link to [Task Priorities](#)

5.3 Scheduling Related Features That May Improve Performance

link to [Task Scheduling Policy](#)

link to [Task Distribution Vs Data Transfer](#)

link to [Power-based Scheduling](#)

link to [Static Scheduling](#)

5.4 CUDA-specific Optimizations

Due to CUDA limitations, StarPU will have a hard time overlapping its own communications and the codelet computations if the application does not use a dedicated CUDA stream for its computations instead of the default stream, which synchronizes all operations of the GPU. StarPU provides one by the use of [starpu_cuda_get_local_stream\(\)](#) which can be used by all CUDA codelet operations to avoid this issue. For instance:

```
func <<<grid,block,0,starpu_cuda_get_local_stream()>>> (foo, bar);  
cudaStreamSynchronize(starpu_cuda_get_local_stream());
```

Unfortunately, some CUDA libraries do not have stream variants of kernels. That will lower the potential for overlapping.

Calling `starpu_cublas_init()` makes StarPU already do appropriate calls for the CUBLAS library. Some libraries like Magma may however change the current stream, one then has to call `cublasSetKernelStream(starpu_cuda_get_local_stream())`; at the beginning of the codelet to make sure that CUBLAS is really using the proper stream.

If the kernel can be made to only use this local stream or other self-allocated streams, i.e. the whole kernel submission can be made asynchronous, then one should enable asynchronous execution of the kernel. That means setting the flag `STARPU_CUDA_ASYNC` in the corresponding field `starpu_codelet::cuda_flags`, and dropping the `cudaStreamSynchronize()` call at the end of the `cuda_func` function, so that it returns immediately after having queued the kernel to the local stream. That way, StarPU will be able to submit and complete data transfers while kernels are executing, instead of only at each kernel submission. The kernel just has to make sure that StarPU can use the local stream to synchronize with the kernel startup and completion.

Using the flag `STARPU_CUDA_ASYNC` also permits to enable concurrent kernel execution, on cards which support it (Kepler and later, notably). This is enabled by setting the environment variable `STARPU_NWORKER_PER_CUDA` to the number of kernels to execute concurrently. This is useful when kernels are small and do not feed the whole GPU with threads to run.

5.5 OpenCL-specific Optimizations

If the kernel can be made to only use the StarPU-provided command queue or other self-allocated queues, i.e. the whole kernel submission can be made asynchronous, then one should enable asynchronous execution of the kernel. This means setting the flag `STARPU_OPENCL_ASYNC` in the corresponding field `starpu_codelet::opencl_flags` and dropping the `clFinish()` and `starpu_opencil_collect_stats()` calls at the end of the kernel, so that it returns immediately after having queued the kernel to the provided queue. That way, StarPU will be able to submit and complete data transfers while kernels are executing, instead of only at each kernel submission. The kernel just has to make sure that StarPU can use the command queue it has provided to synchronize with the kernel startup and completion.

5.6 Detection Stuck Conditions

It may happen that for some reason, StarPU does not make progress for a long period of time. Reason are sometimes due to contention inside StarPU, but sometimes this is due to external reasons, such as stuck MPI driver, or CUDA driver, etc.

```
export STARPU_WATCHDOG_TIMEOUT=10000 (STARPU_WATCHDOG_TIMEOUT)
```

allows to make StarPU print an error message whenever StarPU does not terminate any task for 10ms. In addition to that,

```
export STARPU_WATCHDOG_CRASH=1 (STARPU_WATCHDOG_CRASH)
```

raises SIGABRT in that condition, thus allowing to catch the situation in gdb. It can also be useful to type "handle SIGABRT nopass" in gdb to be able to let the process continue, after inspecting the state of the process.

5.7 How to limit memory per node

TODO

Talk about `STARPU_LIMIT_CUDA_devid_MEM`, `STARPU_LIMIT_CUDA_MEM`, `STARPU_LIMIT_OPENCL_devid_MEM`, `STARPU_LIMIT_OPENCL_MEM` and `STARPU_LIMIT_CPU_MEM`

`starpu_memory_get_total()`

`starpu_memory_get_available()`

5.8 Performance Model Calibration

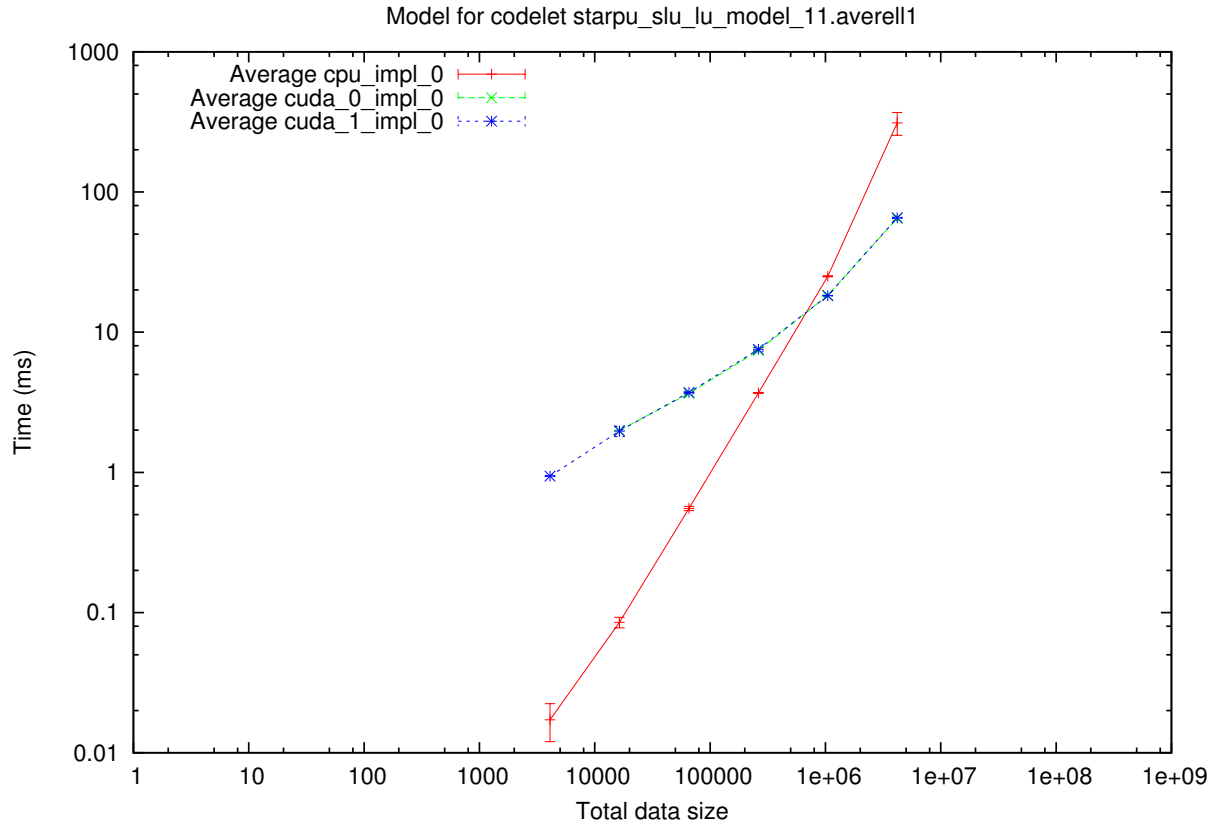
Most schedulers are based on an estimation of codelet duration on each kind of processing unit. For this to be possible, the application programmer needs to configure a performance model for the codelets of the application (see [Performance Model Example](#) for instance). History-based performance models use on-line calibration. StarPU will automatically calibrate codelets which have never been calibrated yet, and save the result in `$STARPU_HOME/.starpu/sampling/codelets`. The models are indexed by machine name. To share the models between machines (e.g. for a homogeneous cluster), use `export STARPU_HOSTNAME=some_global_name (STARPU_HOSTNAME)`. To force continuing calibration, use `export STARPU_CALIBRATE=1 (STARPU_CALIBRATE)`. This may be necessary if your application has not-so-stable performance. StarPU will force calibration (and thus ignore the current result) until 10 (`_STARPU_CALIBRATION_MINIMUM`) measurements have been made on each architecture, to avoid badly scheduling tasks just because the first measurements were not so good. Details on the current performance model status can be obtained from the command `starpu_perfmodel_display`: the `-l` option lists the available performance models, and the `-s` option permits to choose the performance model to be displayed. The result looks like:

```
$ starpu_perfmodel_display -s starpu_slv_lu_model_11
performance model for cpu_impl_0
# hash      size      flops      mean      dev      n
914f3bef    1048576    0.000000e+00  2.503577e+04  1.982465e+02  8
3e921964    65536     0.000000e+00  5.527003e+02  1.848114e+01  7
e5a07e31    4096      0.000000e+00  1.717457e+01  5.190038e+00  14
...
```

Which shows that for the LU 11 kernel with a 1MiB matrix, the average execution time on CPUs was about 25ms, with a 0.2ms standard deviation, over 8 samples. It is a good idea to check this before doing actual performance measurements.

A graph can be drawn by using the tool `starpu_perfmodel_plot`:

```
$ starpu_perfmodel_plot -s starpu_slv_lu_model_11
4096 16384 65536 262144 1048576 4194304
$ gnuplot starpu_starpu_slv_lu_model_11.gp
$ gv starpu_starpu_slv_lu_model_11.eps
```



If a kernel source code was modified (e.g. performance improvement), the calibration information is stale and should be dropped, to re-calibrate from start. This can be done by using `export STARPU_CALIBRATE=2` ([STARPU_CALIBRATE](#)).

Note: history-based performance models get calibrated only if a performance-model-based scheduler is chosen.

The history-based performance models can also be explicitly filled by the application without execution, if e.g. the application already has a series of measurements. This can be done by using [starpu_perfmodel_update_history\(\)](#), for instance:

```
static struct starpu_perfmodel perf_model = {
    .type = STARPU_HISTORY_BASED,
    .symbol = "my_perfmodel",
};

struct starpu_codelet cl = {
    .cuda_funcs = { cuda_func1, cuda_func2 },
    .nbuffers = 1,
    .modes = {STARPU_W},
    .model = &perf_model
};

void feed(void) {
    struct my_measure *measure;
    struct starpu_task task;
    starpu_task_init(&task);

    task.cl = &cl;

    for (measure = &measures[0]; measure < measures[last]; measure++) {
        starpu_data_handle_t handle;
        starpu_vector_data_register(&handle, -1, 0, measure->size, sizeof(float))
    };
    task.handles[0] = handle;
    starpu_perfmodel_update_history(&perf_model, &task,
                                  STARPU_CUDA_DEFAULT + measure->cuda_dev, 0,
                                  measure->implementation, measure->time);

    starpu_task_clean(&task);
    starpu_data_unregister(handle);
}
}
```


Measurement has to be provided in milliseconds for the completion time models, and in Joules for the energy consumption models.

5.9 Profiling

A quick view of how many tasks each worker has executed can be obtained by setting `export STARPU_WORKER_STATS=1` ([STARPU_WORKER_STATS](#)). This is a convenient way to check that execution did happen on accelerators, without penalizing performance with the profiling overhead.

A quick view of how much data transfers have been issued can be obtained by setting `export STARPU_BUS_STATS=1` ([STARPU_BUS_STATS](#)).

More detailed profiling information can be enabled by using `export STARPU_PROFILING=1` ([STARPU_PROFILING](#)) or by calling `starpuprofilingstatus_set()` from the source code. Statistics on the execution can then be obtained by using `export STARPU_BUS_STATS=1` and `export STARPU_WORKER_STATS=1`. More details on performance feedback are provided in the next chapter.

Part III

StarPU Inside

Chapter 6

Tasks In StarPU

6.1 Task Granularity

Like any other runtime, StarPU has some overhead to manage tasks. Since it does smart scheduling and data management, that overhead is not always neglectable. The order of magnitude of the overhead is typically a couple of microseconds, which is actually quite smaller than the CUDA overhead itself. The amount of work that a task should do should thus be somewhat bigger, to make sure that the overhead becomes neglectable. The offline performance feedback can provide a measure of task length, which should thus be checked if bad performance are observed. To get a grasp at the scalability possibility according to task size, one can run `tests/microbenchs/tasks_size_overhead.sh` which draws curves of the speedup of independent tasks of very small sizes.

The choice of scheduler also has impact over the overhead: for instance, the scheduler `dmda` takes time to make a decision, while `eager` does not. `tasks_size_overhead.sh` can again be used to get a grasp at how much impact that has on the target machine.

6.2 Task Submission

To let StarPU make online optimizations, tasks should be submitted asynchronously as much as possible. Ideally, all the tasks should be submitted, and mere calls to `starpu_task_wait_for_all()` or `starpu_data_unregister()` be done to wait for termination. StarPU will then be able to rework the whole schedule, overlap computation with communication, manage accelerator local memory usage, etc.

6.3 Task Priorities

By default, StarPU will consider the tasks in the order they are submitted by the application. If the application programmer knows that some tasks should be performed in priority (for instance because their output is needed by many other tasks and may thus be a bottleneck if not executed early enough), the field `starpu_task::priority` should be set to transmit the priority information to StarPU.

6.4 Setting The Data Handles For A Task

The number of data a task can manage is fixed by the environment variable `STARPU_NMAXBUFS` which has a default value which can be changed through the configure option `--enable-maxbuffers`.

However, it is possible to define tasks managing more data by using the field `starpu_task::dyn_handles` when defining a task and the field `starpu_codelet::dyn_modes` when defining the corresponding codelet.

```
enum starpu_data_access_mode modes[STARPU_NMAXBUFS+1] = {  
    STARPU_R, STARPU_R, ...  
}
```

```

};

struct starpu_codelet dummy_big_cl =
{
    .cuda_funcs = { dummy_big_kernel },
    .opencl_funcs = { dummy_big_kernel },
    .cpu_funcs = { dummy_big_kernel },
    .cpu_funcs_name = { "dummy_big_kernel" },
    .nbuffers = STARPU_NMAXBUFS+1,
    .dyn_modes = modes
};

task = starpu_task_create();
task->cl = &dummy_big_cl;
task->dyn_handles = malloc(task->cl->nbuffers * sizeof(
    starpu_data_handle_t));
for(i=0 ; i<task->cl->nbuffers ; i++)
{
    task->dyn_handles[i] = handle;
}
starpu_task_submit(task);

starpu_data_handle_t *handles = malloc(dummy_big_cl.nbuffers * sizeof(
    starpu_data_handle_t));
for(i=0 ; i<dummy_big_cl.nbuffers ; i++)
{
    handles[i] = handle;
}
starpu_task_insert(&dummy_big_cl,
    STARPU_VALUE, &dummy_big_cl.nbuffers, sizeof(dummy_big_cl.
    nbuffers),
    STARPU_DATA_ARRAY, handles, dummy_big_cl.
    nbuffers,
    0);

```

The whole code for this complex data interface is available in the directory `examples/basic_↵`
`examples/dynamic_handles.c`.

6.5 Using Multiple Implementations Of A Codelet

One may want to write multiple implementations of a codelet for a single type of device and let StarPU choose which one to run. As an example, we will show how to use SSE to scale a vector. The codelet can be written as follows:

```

#include <xmmintrin.h>

void scal_sse_func(void *buffers[], void *cl_arg)
{
    float *vector = (float *) STARPU_VECTOR_GET_PTR(buffers[0]);
    unsigned int n = STARPU_VECTOR_GET_NX(buffers[0]);
    unsigned int n_iterations = n/4;
    if (n % 4 != 0)
        n_iterations++;

    __m128 *VECTOR = (__m128*) vector;
    __m128 factor = __attribute__((aligned(16)));
    factor = _mm_set1_ps(*(float *) cl_arg);

    unsigned int i;
    for (i = 0; i < n_iterations; i++)
        VECTOR[i] = _mm_mul_ps(factor, VECTOR[i]);
}

struct starpu_codelet cl = {
    .cpu_funcs = { scal_cpu_func, scal_sse_func },
    .cpu_funcs_name = { "scal_cpu_func", "scal_sse_func" },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};

```

Schedulers which are multi-implementation aware (only `dmda` and `pheft` for now) will use the performance models of all the implementations it was given, and pick the one that seems to be the fastest.

6.6 Enabling Implementation According To Capabilities

Some implementations may not run on some devices. For instance, some CUDA devices do not support double floating point precision, and thus the kernel execution would just fail; or the device may not have enough shared memory for the implementation being used. The field `starpu_codelet::can_execute` permits to express this. For instance:

```
static int can_execute(unsigned workerid, struct starpu_task *task, unsigned nimpl)
{
    const struct cudaDeviceProp *props;
    if (starpu_worker_get_type(workerid) ==
        STARPU_CPU_WORKER)
        return 1;
    /* Cuda device */
    props = starpu_cuda_get_device_properties(workerid);
    if (props->major >= 2 || props->minor >= 3)
        /* At least compute capability 1.3, supports doubles */
        return 1;
    /* Old card, does not support doubles */
    return 0;
}

struct starpu_codelet cl = {
    .can_execute = can_execute,
    .cpu_funcs = { cpu_func },
    .cpu_funcs_name = { "cpu_func" },
    .cuda_funcs = { gpu_func },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
```

This can be essential e.g. when running on a machine which mixes various models of CUDA devices, to take benefit from the new models without crashing on old models.

Note: the function `starpu_codelet::can_execute` is called by the scheduler each time it tries to match a task with a worker, and should thus be very fast. The function `starpu_cuda_get_device_properties()` provides a quick access to CUDA properties of CUDA devices to achieve such efficiency.

Another example is to compile CUDA code for various compute capabilities, resulting with two CUDA functions, e.g. `scal_gpu_13` for compute capability 1.3, and `scal_gpu_20` for compute capability 2.0. Both functions can be provided to StarPU by using `starpu_codelet::cuda_funcs`, and `starpu_codelet::can_execute` can then be used to rule out the `scal_gpu_20` variant on a CUDA device which will not be able to execute it:

```
static int can_execute(unsigned workerid, struct starpu_task *task, unsigned nimpl)
{
    const struct cudaDeviceProp *props;
    if (starpu_worker_get_type(workerid) ==
        STARPU_CPU_WORKER)
        return 1;
    /* Cuda device */
    if (nimpl == 0)
        /* Trying to execute the 1.3 capability variant, we assume it is ok in all cases. */
        return 1;
    /* Trying to execute the 2.0 capability variant, check that the card can do it. */
    props = starpu_cuda_get_device_properties(workerid);
    if (props->major >= 2 || props->minor >= 0)
        /* At least compute capability 2.0, can run it */
        return 1;
    /* Old card, does not support 2.0, will not be able to execute the 2.0 variant. */
    return 0;
}

struct starpu_codelet cl = {
    .can_execute = can_execute,
    .cpu_funcs = { cpu_func },
    .cpu_funcs_name = { "cpu_func" },
    .cuda_funcs = { scal_gpu_13, scal_gpu_20 },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
```

Another example is having specialized implementations for some given common sizes, for instance here we have a specialized implementation for 1024x1024 matrices:

```
static int can_execute(unsigned workerid, struct starpu_task *task, unsigned nimpl)
```

```

{
    const struct cudaDeviceProp *props;
    if (starpu_worker_get_type(workerid) ==
        STARPU_CPU_WORKER)
        return 1;
    /* Cuda device */
    switch (nimpl)
    {
        case 0:
            /* Trying to execute the generic capability variant. */
            return 1;
        case 1:
            {
                /* Trying to execute the size == 1024 specific variant. */
                struct starpu_matrix_interface *interface =
                    starpu_data_get_interface_on_node(task->handles[0]);
                return STARPU_MATRIX_GET_NX(interface) == 1024 &&
                    STARPU_MATRIX_GET_NY(interface) == 1024;
            }
    }
}

struct starpu_codelet cl = {
    .can_execute = can_execute,
    .cpu_funcs = { cpu_func },
    .cpu_funcs_name = { "cpu_func" },
    .cuda_funcs = { potrf_gpu_generic, potrf_gpu_1024 },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};

```

Note: the most generic variant should be provided first, as some schedulers are not able to try the different variants.

6.7 Insert Task Utility

StarPU provides the wrapper function `starpu_task_insert()` to ease the creation and submission of tasks.

Here the implementation of the codelet:

```

void func_cpu(void *descr[], void *_args)
{
    int *x0 = (int *)STARPU_VARIABLE_GET_PTR(descr[0]);
    float *x1 = (float *)STARPU_VARIABLE_GET_PTR(descr[1]);
    int ifactor;
    float ffactor;

    starpu_codelet_unpack_args(_args, &ifactor, &ffactor);
    *x0 = *x0 * ifactor;
    *x1 = *x1 * ffactor;
}

struct starpu_codelet mycodelet = {
    .cpu_funcs = { func_cpu },
    .cpu_funcs_name = { "func_cpu" },
    .nbuffers = 2,
    .modes = { STARPU_RW, STARPU_RW }
};

```

And the call to the function `starpu_task_insert()`:

```

starpu_task_insert(&mycodelet,
    STARPU_VALUE, &ifactor, sizeof(ifactor),
    STARPU_VALUE, &ffactor, sizeof(ffactor),
    STARPU_RW, data_handles[0], STARPU_RW, data_handles[1],
    0);

```

The call to `starpu_task_insert()` is equivalent to the following code:

```

struct starpu_task *task = starpu_task_create();
task->cl = &mycodelet;
task->handles[0] = data_handles[0];
task->handles[1] = data_handles[1];
char *arg_buffer;
size_t arg_buffer_size;
starpu_codelet_pack_args(&arg_buffer, &arg_buffer_size,
    STARPU_VALUE, &ifactor, sizeof(ifactor),

```



```

        STARPU_VALUE, &ffactor, sizeof(ffactor),
        0);
task->cl_arg = arg_buffer;
task->cl_arg_size = arg_buffer_size;
int ret = starpu_task_submit(task);

```

Here a similar call using `STARPU_DATA_ARRAY`.

```

starpu_task_insert(&mycodelet,
    STARPU_DATA_ARRAY, data_handles, 2,
    STARPU_VALUE, &ifactor, sizeof(ifactor),
    STARPU_VALUE, &ffactor, sizeof(ffactor),
    0);

```

If some part of the task insertion depends on the value of some computation, the macro `STARPU_DATA_ACQUIRE_CB` can be very convenient. For instance, assuming that the index variable `i` was registered as handle `A_handle[i]`:

```

/* Compute which portion we will work on, e.g. pivot */
starpu_task_insert(&which_index, STARPU_W, i_handle, 0);

/* And submit the corresponding task */
STARPU_DATA_ACQUIRE_CB(i_handle, STARPU_R,
    starpu_task_insert(&work, STARPU_RW, A_handle[i], 0));

```

The macro `STARPU_DATA_ACQUIRE_CB` submits an asynchronous request for acquiring data `i` for the main application, and will execute the code given as third parameter when it is acquired. In other words, as soon as the value of `i` computed by the codelet `which_index` can be read, the portion of code passed as third parameter of `STARPU_DATA_ACQUIRE_CB` will be executed, and is allowed to read from `i` to use it e.g. as an index. Note that this macro is only available when compiling StarPU with the compiler `gcc`.

6.8 Parallel Tasks

StarPU can leverage existing parallel computation libraries by the means of parallel tasks. A parallel task is a task which gets worked on by a set of CPUs (called a parallel or combined worker) at the same time, by using an existing parallel CPU implementation of the computation to be achieved. This can also be useful to improve the load balance between slow CPUs and fast GPUs: since CPUs work collectively on a single task, the completion time of tasks on CPUs become comparable to the completion time on GPUs, thus relieving from granularity discrepancy concerns. `hwloc` support needs to be enabled to get good performance, otherwise StarPU will not know how to better group cores.

Two modes of execution exist to accomodate with existing usages.

6.8.1 Fork-mode Parallel Tasks

In the Fork mode, StarPU will call the codelet function on one of the CPUs of the combined worker. The codelet function can use `starpu_combined_worker_get_size()` to get the number of threads it is allowed to start to achieve the computation. The CPU binding mask for the whole set of CPUs is already enforced, so that threads created by the function will inherit the mask, and thus execute where StarPU expected, the OS being in charge of choosing how to schedule threads on the corresponding CPUs. The application can also choose to bind threads by hand, using e.g. `sched_getaffinity` to know the CPU binding mask that StarPU chose.

For instance, using OpenMP (full source is available in `examples/openmp/vector_scal.c`):

```

void scal_cpu_func(void *buffers[], void *_args)
{
    unsigned i;
    float *factor = _args;
    struct starpu_vector_interface *vector = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector);
    float *val = (float *)STARPU_VECTOR_GET_PTR(vector);

#pragma omp parallel for num_threads(starpu_combined_worker_get_size())
    for (i = 0; i < n; i++)

```

```

        val[i] *= *factor;
    }

    static struct starpu_codelet cl =
    {
        .modes = { STARPU_RW },
        .where = STARPU_CPU,
        .type = STARPU_FORKJOIN,
        .max_parallelism = INT_MAX,
        .cpu_funcs = { scal_cpu_func },
        .cpu_funcs_name = { "scal_cpu_func" },
        .nbuffers = 1,
    };

```

Other examples include for instance calling a BLAS parallel CPU implementation (see `examples/mult/xgemv.c`).

6.8.2 SPMD-mode Parallel Tasks

In the SPMD mode, StarPU will call the codelet function on each CPU of the combined worker. The codelet function can use `starpu_combined_worker_get_size()` to get the total number of CPUs involved in the combined worker, and thus the number of calls that are made in parallel to the function, and `starpu_combined_worker_get_rank()` to get the rank of the current CPU within the combined worker. For instance:

```

static void func(void *buffers[], void *args)
{
    unsigned i;
    float *factor = _args;
    struct starpu_vector_interface *vector = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX(vector);
    float *val = (float *)STARPU_VECTOR_GET_PTR(vector);

    /* Compute slice to compute */
    unsigned m = starpu_combined_worker_get_size();
    unsigned j = starpu_combined_worker_get_rank();
    unsigned slice = (n+m-1)/m;

    for (i = j * slice; i < (j+1) * slice && i < n; i++)
        val[i] *= *factor;
}

static struct starpu_codelet cl =
{
    .modes = { STARPU_RW },
    .type = STARPU_SPMD,
    .max_parallelism = INT_MAX,
    .cpu_funcs = { func },
    .cpu_funcs_name = { "func" },
    .nbuffers = 1,
}

```

Of course, this trivial example will not really benefit from parallel task execution, and was only meant to be simple to understand. The benefit comes when the computation to be done is so that threads have to e.g. exchange intermediate results, or write to the data in a complex but safe way in the same buffer.

6.8.3 Parallel Tasks Performance

To benefit from parallel tasks, a parallel-task-aware StarPU scheduler has to be used. When exposed to codelets with a flag `STARPU_FORKJOIN` or `STARPU_SPMD`, the schedulers `pheft` (parallel-heft) and `peager` (parallel eager) will indeed also try to execute tasks with several CPUs. It will automatically try the various available combined worker sizes (making several measurements for each worker size) and thus be able to avoid choosing a large combined worker if the codelet does not actually scale so much.

6.8.4 Combined Workers

By default, StarPU creates combined workers according to the architecture structure as detected by `hwloc`. It means that for each object of the `hwloc` topology (NUMA node, socket, cache, ...) a combined worker will be

created. If some nodes of the hierarchy have a big arity (e.g. many cores in a socket without a hierarchy of shared caches), StarPU will create combined workers of intermediate sizes. The variable `STARPU_SYNTHESIZE_ARY_←
TY_COMBINED_WORKER` permits to tune the maximum arity between levels of combined workers.

The combined workers actually produced can be seen in the output of the tool `starpu_machine_display` (the environment variable `STARPU_SCHED` has to be set to a combined worker-aware scheduler such as `pheft` or `peager`).

6.8.5 Concurrent Parallel Tasks

Unfortunately, many environments and librairies do not support concurrent calls.

For instance, most OpenMP implementations (including the main ones) do not support concurrent `pragma omp parallel` statements without nesting them in another `pragma omp parallel` statement, but StarPU does not yet support creating its CPU workers by using such pragma.

Other parallel libraries are also not safe when being invoked concurrently from different threads, due to the use of global variables in their sequential sections for instance.

The solution is then to use only one combined worker at a time. This can be done by setting the field `starpu_conf_←
::single_combined_worker` to 1, or setting the environment variable `STARPU_SINGLE_COMBINED_WORKER` to 1. StarPU will then run only one parallel task at a time (but other CPU and GPU tasks are not affected and can be run concurrently). The parallel task scheduler will however still try varying combined worker sizes to look for the most efficient ones.

Chapter 7

Data Management

intro qui parle de coherency entre autres

7.1 Data Management

When the application allocates data, whenever possible it should use the function `starpu_malloc()`, which will ask CUDA or OpenCL to make the allocation itself and pin the corresponding allocated memory. This is needed to permit asynchronous data transfer, i.e. permit data transfer to overlap with computations. Otherwise, the trace will show that the `DriverCopyAsync` state takes a lot of time, this is because CUDA or OpenCL then reverts to synchronous transfers.

By default, StarPU leaves replicates of data wherever they were used, in case they will be re-used by other tasks, thus saving the data transfer time. When some task modifies some data, all the other replicates are invalidated, and only the processing unit which ran that task will have a valid replicate of the data. If the application knows that this data will not be re-used by further tasks, it should advise StarPU to immediately replicate it to a desired list of memory nodes (given through a bitmask). This can be understood like the write-through mode of CPU caches.

```
starpu_data_set_wt_mask(img_handle, 1<<0);
```

will for instance request to always automatically transfer a replicate into the main memory (node 0), as bit 0 of the write-through bitmask is being set.

```
starpu_data_set_wt_mask(img_handle, ~0U);
```

will request to always automatically broadcast the updated data to all memory nodes.

Setting the write-through mask to `~0U` can also be useful to make sure all memory nodes always have a copy of the data, so that it is never evicted when memory gets scarce.

Implicit data dependency computation can become expensive if a lot of tasks access the same piece of data. If no dependency is required on some piece of data (e.g. because it is only accessed in read-only mode, or because write accesses are actually commutative), use the function `starpu_data_set_sequential_consistency_flag()` to disable implicit dependencies on that data.

In the same vein, accumulation of results in the same data can become a bottleneck. The use of the mode `STARPU_REDUX` permits to optimize such accumulation (see [Data Reduction](#)). To a lesser extent, the use of the flag `STARPU_COMMUTE` keeps the bottleneck, but at least permits the accumulation to happen in any order.

Applications often need a data just for temporary results. In such a case, registration can be made without an initial value, for instance this produces a vector data:

```
starpu_vector_data_register(&handle, -1, 0, n, sizeof(float));
```

StarPU will then allocate the actual buffer only when it is actually needed, e.g. directly on the GPU without allocating in main memory.

In the same vein, once the temporary results are not useful any more, the data should be thrown away. If the handle is not to be reused, it can be unregistered:

```
starpu_data_unregister_submit(handle);
```

actual unregistration will be done after all tasks working on the handle terminate.

If the handle is to be reused, instead of unregistering it, it can simply be invalidated:

```
starpu_data_invalidate_submit(handle);
```

the buffers containing the current value will then be freed, and reallocated only when another task writes some value to the handle.

7.2 Data Prefetch

The scheduling policies `heft`, `dmda` and `pheft` perform data prefetch (see [STARPU_PREFETCH](#)): as soon as a scheduling decision is taken for a task, requests are issued to transfer its required data to the target processing unit, if needed, so that when the processing unit actually starts the task, its data will hopefully be already available and it will not have to wait for the transfer to finish.

The application may want to perform some manual prefetching, for several reasons such as excluding initial data transfers from performance measurements, or setting up an initial statically-computed data distribution on the machine before submitting tasks, which will thus guide StarPU toward an initial task distribution (since StarPU will try to avoid further transfers).

This can be achieved by giving the function `starpu_data_prefetch_on_node()` the handle and the desired target memory node.

7.3 Partitioning Data

An existing piece of data can be partitioned in sub parts to be used by different tasks, for instance:

```
int vector[NX];
starpu_data_handle_t handle;

/* Declare data to StarPU */
starpu_vector_data_register(&handle, STARPU_MAIN_RAM, (uintptr_t)
    vector,
    NX, sizeof(vector[0]));

/* Partition the vector in PARTS sub-vectors */
struct starpu_data_filter f =
{
    .filter_func = starpu_vector_filter_block,
    .nchildren = PARTS
};
starpu_data_partition(handle, &f);
```

The task submission then uses the function `starpu_data_get_sub_data()` to retrieve the sub-handles to be passed as tasks parameters.

```
/* Submit a task on each sub-vector */
for (i=0; i<starpu_data_get_nb_children(handle); i++) {
    /* Get subdata number i (there is only 1 dimension) */
    starpu_data_handle_t sub_handle =
        starpu_data_get_sub_data(handle, 1, i);
    struct starpu_task *task = starpu_task_create();

    task->handles[0] = sub_handle;
    task->cl = &cl;
    task->synchronous = 1;
    task->cl_arg = &factor;
    task->cl_arg_size = sizeof(factor);

    starpu_task_submit(task);
}
```

Partitioning can be applied several times, see `examples/basic_examples/mult.c` and `examples/filters/`.

Wherever the whole piece of data is already available, the partitioning will be done in-place, i.e. without allocating new buffers but just using pointers inside the existing copy. This is particularly important to be aware of when using OpenCL, where the kernel parameters are not pointers, but handles. The kernel thus needs to be also passed the offset within the OpenCL buffer:

```
void opencil_func(void *buffers[], void *cl_arg)
{
    cl_mem vector = (cl_mem) STARPU_VECTOR_GET_DEV_HANDLE(buffers[0]);
    unsigned offset = STARPU_BLOCK_GET_OFFSET(buffers[0]);

    ...
    clSetKernelArg(kernel, 0, sizeof(vector), &vector);
    clSetKernelArg(kernel, 1, sizeof(offset), &offset);
    ...
}
```

And the kernel has to shift from the pointer passed by the OpenCL driver:

```
__kernel void opencil_kernel(__global int *vector, unsigned offset)
{
    block = (__global void *)block + offset;
    ...
}
```

StarPU provides various interfaces and filters for matrices, vectors, etc., but applications can also write their own data interfaces and filters, see `examples/interface` and `examples/filters/custom_mf` for an example.

7.4 Data Reduction

In various cases, some piece of data is used to accumulate intermediate results. For instances, the dot product of a vector, maximum/minimum finding, the histogram of a photograph, etc. When these results are produced along the whole machine, it would not be efficient to accumulate them in only one place, incurring data transmission each and access concurrency.

StarPU provides a mode `STARPU_REDUX`, which permits to optimize that case: it will allocate a buffer on each memory node, and accumulate intermediate results there. When the data is eventually accessed in the normal mode `STARPU_R`, StarPU will collect the intermediate results in just one buffer.

For this to work, the user has to use the function `starpu_data_set_reduction_methods()` to declare how to initialize these buffers, and how to assemble partial results.

For instance, `cg` uses that to optimize its dot product: it first defines the codelets for initialization and reduction:

```
struct starpu_codelet bzero_variable_cl =
{
    .cpu_funcs = { bzero_variable_cpu },
    .cpu_funcs_name = { "bzero_variable_cpu" },
    .cuda_funcs = { bzero_variable_cuda },
    .nbuffers = 1,
}

static void accumulate_variable_cpu(void *descr[], void *cl_arg)
{
    double *v_dst = (double *)STARPU_VARIABLE_GET_PTR(descr[0]);
    double *v_src = (double *)STARPU_VARIABLE_GET_PTR(descr[1]);
    *v_dst = *v_dst + *v_src;
}

static void accumulate_variable_cuda(void *descr[], void *cl_arg)
{
    double *v_dst = (double *)STARPU_VARIABLE_GET_PTR(descr[0]);
    double *v_src = (double *)STARPU_VARIABLE_GET_PTR(descr[1]);
    cublasaxpy(1, (double)1.0, v_src, 1, v_dst, 1);
    cudaStreamSynchronize(starpu_cuda_get_local_stream());
}

struct starpu_codelet accumulate_variable_cl =
{

```

```

    .cpu_funcs = { accumulate_variable_cpu },
    .cpu_funcs_name = { "accumulate_variable_cpu" },
    .cuda_funcs = { accumulate_variable_cuda },
    .nbuffers = 1,
}

```

and attaches them as reduction methods for its handle `dtq`:

```

starpu_variable_data_register(&dtq_handle, -1, NULL, sizeof(
    type));
starpu_data_set_reduction_methods(dtq_handle,
    &accumulate_variable_cl, &bzero_variable_cl);

```

and `dtq_handle` can now be used in mode `STARPU_REDUX` for the dot products with partitioned vectors:

```

for (b = 0; b < nblocks; b++)
    starpu_task_insert(&dot_kernel_cl,
        STARPU_REDUX, dtq_handle,
        STARPU_R, starpu_data_get_sub_data(v1, 1, b),
        STARPU_R, starpu_data_get_sub_data(v2, 1, b),
        0);

```

During registration, we have here provided `NULL`, i.e. there is no initial value to be taken into account during reduction. StarPU will thus only take into account the contributions from the tasks `dot_kernel_cl`. Also, it will not allocate any memory for `dtq_handle` before tasks `dot_kernel_cl` are ready to run.

If another dot product has to be performed, one could unregister `dtq_handle`, and re-register it. But one can also call `starpu_data_invalidate_submit()` with the parameter `dtq_handle`, which will clear all data from the handle, thus resetting it back to the initial status `register(NULL)`.

The example `cg` also uses reduction for the blocked `gemv` kernel, leading to yet more relaxed dependencies and more parallelism.

`STARPU_REDUX` can also be passed to `starpu_mpi_task_insert()` in the MPI case. That will however not produce any MPI communication, but just pass `STARPU_REDUX` to the underlying `starpu_task_insert()`. It is up to the application to call `starpu_mpi_redux_data()`, which posts tasks that will reduce the partial results among MPI nodes into the MPI node which owns the data. For instance, some hypothetical application which collects partial results into data `res`, then uses it for other computation, before looping again with a new reduction:

```

for (i = 0; i < 100; i++) {
    starpu_mpi_task_insert(MPI_COMM_WORLD, &init_res,
        STARPU_W, res, 0);
    starpu_mpi_task_insert(MPI_COMM_WORLD, &work,
        STARPU_RW, A,
        STARPU_R, B, STARPU_REDUX, res, 0);
    starpu_mpi_redux_data(MPI_COMM_WORLD, res);
    starpu_mpi_task_insert(MPI_COMM_WORLD, &work2,
        STARPU_RW, B, STARPU_R, res, 0);
}

```

7.5 Temporary Buffers

There are two kinds of temporary buffers: temporary data which just pass results from a task to another, and scratch data which are needed only internally by tasks.

7.5.1 Temporary Data

Data can sometimes be entirely produced by a task, and entirely consumed by another task, without the need for other parts of the application to access it. In such case, registration can be done without prior allocation, by using the special memory node number `-1`, and passing a zero pointer. StarPU will actually allocate memory only when the task creating the content gets scheduled, and destroy it on unregistration.

In addition to that, it can be tedious for the application to have to unregister the data, since it will not use its content anyway. The unregistration can be done lazily by using the function `starpu_data_unregister_submit()`, which will

record that no more tasks accessing the handle will be submitted, so that it can be freed as soon as the last task accessing it is over.

The following code exemplifies both points: it registers the temporary data, submits three tasks accessing it, and records the data for automatic unregistration.

```
starpu_vector_data_register(&handle, -1, 0, n, sizeof(float));
starpu_task_insert(&produce_data, STARPU_W, handle, 0);
starpu_task_insert(&compute_data, STARPU_RW, handle, 0);
starpu_task_insert(&summarize_data, STARPU_R, handle,
    STARPU_W, result_handle, 0);
starpu_data_unregister_submit(handle);
```

The application may also want to see the temporary data initialized on the fly before being used by the task. This can be done by using `starpu_data_set_reduction_methods()` to set an initialization codelet (no redux codelet is needed).

7.5.2 Scratch Data

Some kernels sometimes need temporary data to achieve the computations, i.e. a workspace. The application could allocate it at the start of the codelet function, and free it at the end, but that would be costly. It could also allocate one buffer per worker (similarly to [How To Initialize A Computation Library Once For Each Worker?](#)), but that would make them systematic and permanent. A more optimized way is to use the data access mode `STARPU_SCRATCH`, as exemplified below, which provides per-worker buffers without content consistency.

```
starpu_vector_data_register(&workspace, -1, 0, sizeof(float));
for (i = 0; i < N; i++)
    starpu_task_insert(&compute, STARPU_R, input[i],
        STARPU_SCRATCH, workspace, STARPU_W, output[i], 0);
```

StarPU will make sure that the buffer is allocated before executing the task, and make this allocation per-worker: for CPU workers, notably, each worker has its own buffer. This means that each task submitted above will actually have its own workspace, which will actually be the same for all tasks running one after the other on the same worker. Also, if for instance GPU memory becomes scarce, StarPU will notice that it can free such buffers easily, since the content does not matter.

The example `examples/pi` uses scratches for some temporary buffer.

7.6 The Multiformat Interface

It may be interesting to represent the same piece of data using two different data structures: one that would only be used on CPUs, and one that would only be used on GPUs. This can be done by using the multiformat interface. StarPU will be able to convert data from one data structure to the other when needed. Note that the scheduler `dmda` is the only one optimized for this interface. The user must provide StarPU with conversion codelets:

```
#define NX 1024
struct point array_of_structs[NX];
starpu_data_handle_t handle;

/*
 * The conversion of a piece of data is itself a task, though it is created,
 * submitted and destroyed by StarPU internals and not by the user. Therefore,
 * we have to define two codelets.
 * Note that for now the conversion from the CPU format to the GPU format has to
 * be executed on the GPU, and the conversion from the GPU to the CPU has to be
 * executed on the CPU.
 */
#ifdef STARPU_USE_OPENCL
void cpu_to_opengl_opengl_func(void *buffers[], void *args);
struct starpu_codelet cpu_to_opengl_cl = {
    .where = STARPU_OPENGL,
    .opengl_funcs = { cpu_to_opengl_opengl_func },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};

void opengl_to_cpu_func(void *buffers[], void *args);
```

```

struct starpu_codelet opengl_to_cpu_cl = {
    .where = STARPU_CPU,
    .cpu_funcs = { opengl_to_cpu_func },
    .cpu_funcs_name = { "opengl_to_cpu_func" },
    .nbuffers = 1,
    .modes = { STARPU_RW }
};
#endif

struct starpu_multiformat_data_interface_ops format_ops = {
#ifdef STARPU_USE_OPENCL
    .opengl_elsize = 2 * sizeof(float),
    .cpu_to_opengl_cl = &cpu_to_opengl_cl,
    .opengl_to_cpu_cl = &opengl_to_cpu_cl,
#endif
    .cpu_elsize = 2 * sizeof(float),
    ...
};

starpu_multiformat_data_register(handle,
    STARPU_MAIN_RAM, &array_of_structs, NX, &format_ops);

```

Kernels can be written almost as for any other interface. Note that `STARPU_MULTIFORMAT_GET_CPU_PTR` shall only be used for CPU kernels. CUDA kernels must use `STARPU_MULTIFORMAT_GET_CUDA_PTR`, and OpenCL kernels must use `STARPU_MULTIFORMAT_GET_OPENGL_PTR`. `STARPU_MULTIFORMAT_GET_NX` may be used in any kind of kernel.

```

static void
multiformat_scal_cpu_func(void *buffers[], void *args)
{
    struct point *aos;
    unsigned int n;

    aos = STARPU_MULTIFORMAT_GET_CPU_PTR(buffers[0]);
    n = STARPU_MULTIFORMAT_GET_NX(buffers[0]);
    ...
}

extern "C" void multiformat_scal_cuda_func(void *buffers[], void *_args)
{
    unsigned int n;
    struct struct_of_arrays *soa;

    soa = (struct struct_of_arrays *) STARPU_MULTIFORMAT_GET_CUDA_PTR(
        buffers[0]);
    n = STARPU_MULTIFORMAT_GET_NX(buffers[0]);
    ...
}

```

A full example may be found in `examples/basic_examples/multiformat.c`.

7.7 Defining A New Data Interface

Let's define a new data interface to manage complex numbers.

```

/* interface for complex numbers */
struct starpu_complex_interface
{
    double *real;
    double *imaginary;
    int nx;
};

```

Registering such a data to StarPU is easily done using the function `starpu_data_register()`. The last parameter of the function, `interface_complex_ops`, will be described below.

```

void starpu_complex_data_register(starpu_data_handle_t *handle,
    unsigned home_node, double *real, double *imaginary, int nx)
{
    struct starpu_complex_interface complex =
    {
        .real = real,
        .imaginary = imaginary,
    }
}

```

```

        .nx = nx
    };

    if (interface_complex_ops.interfaceid ==
        STARPU_UNKNOWN_INTERFACE_ID)
    {
        interface_complex_ops.interfaceid =
            starpu_data_interface_get_next_id();
    }

    starpu_data_register(handleptr, home_node, &complex, &interface_complex_ops);
}

```

Different operations need to be defined for a data interface through the type `starpu_data_interface_ops`. We only define here the basic operations needed to run simple applications. The source code for the different functions can be found in the file `examples/interface/complex_interface.c`.

```

static struct starpu_data_interface_ops interface_complex_ops =
{
    .register_data_handle = complex_register_data_handle,
    .allocate_data_on_node = complex_allocate_data_on_node,
    .copy_methods = &complex_copy_methods,
    .get_size = complex_get_size,
    .footprint = complex_footprint,
    .interfaceid = STARPU_UNKNOWN_INTERFACE_ID,
    .interface_size = sizeof(struct starpu_complex_interface),
};

```

Functions need to be defined to access the different fields of the complex interface from a StarPU data handle.

```

double *starpu_complex_get_real(starpu_data_handle_t handle)
{
    struct starpu_complex_interface *complex_interface =
        (struct starpu_complex_interface *) starpu_data_get_interface_on_node
        (handle, STARPU_MAIN_RAM);
    return complex_interface->real;
}

double *starpu_complex_get_imaginary(starpu_data_handle_t handle);
int starpu_complex_get_nx(starpu_data_handle_t handle);

```

Similar functions need to be defined to access the different fields of the complex interface from a `void *` pointer to be used within codelet implementations.

```

#define STARPU_COMPLEX_GET_REAL(interface) \
    ((struct starpu_complex_interface *) (interface))->real)
#define STARPU_COMPLEX_GET_IMAGINARY(interface) \
    ((struct starpu_complex_interface *) (interface))->imaginary)
#define STARPU_COMPLEX_GET_NX(interface) \
    ((struct starpu_complex_interface *) (interface))->nx)

```

Complex data interfaces can then be registered to StarPU.

```

double real = 45.0;
double imaginary = 12.0; starpu_complex_data_register(&handle1, STARPU_MAIN_RAM, &real, &
    imaginary, 1);
starpu_task_insert(&cl_display, STARPU_R, handle1, 0);

```

and used by codelets.

```

void display_complex_codelet(void *descr[], __attribute__((unused)) void *_args)
{
    int nx = STARPU_COMPLEX_GET_NX(descr[0]);
    double *real = STARPU_COMPLEX_GET_REAL(descr[0]);
    double *imaginary = STARPU_COMPLEX_GET_IMAGINARY(descr[0]);
    int i;

    for(i=0 ; i<nx ; i++)
    {
        fprintf(stderr, "Complex[%d] = %3.2f + %3.2f i\n", i, real[i], imaginary[i]);
    }
}

```

The whole code for this complex data interface is available in the directory `examples/interface/`.

7.8 Specifying a target node for task data

When executing a task on a GPU for instance, StarPU would normally copy all the needed data for the tasks on the embedded memory of the GPU. It may however happen that the task kernel would rather have some of the datas kept in the main memory instead of copied in the GPU, a pivoting vector for instance. This can be achieved by setting the `starpu_codelet::specific_nodes` flag to 1, and then fill the `starpu_codelet::nodes` array (or `starpu_codelet::dyn_nodes` when `starpu_codelet::nbuffers` is greater than `STARPU_NMAXBUFS`) with the node numbers where data should be copied to, or -1 to let StarPU copy it to the memory node where the task will be executed. For instance, with the following codelet:

```
struct starpu_codelet cl =
{
    .cuda_funcs = { kernel },
    .nbuffers = 2,
    .modes = { STARPU_RW, STARPU_RW },
    .specific_nodes = 1,
    .nodes = { STARPU_MAIN_RAM, -1 },
};
```

the first data of the task will be kept in the main memory, while the second data will be copied to the CUDA GPU as usual.

Chapter 8

Scheduling

8.1 Task Scheduling Policy

The basics of the scheduling policy are that

- The scheduler gets to schedule tasks (`push` operation) when they become ready to be executed, i.e. they are not waiting for some tags, data dependencies or task dependencies.
- Workers pull tasks (`pop` operation) one by one from the scheduler.

This means scheduling policies usually contain at least one queue of tasks to store them between the time when they become available, and the time when a worker gets to grab them.

By default, StarPU uses the simple greedy scheduler `eager`. This is because it provides correct load balance even if the application codelets do not have performance models. If your application codelets have performance models ([Performance Model Example](#)), you should change the scheduler thanks to the environment variable `STARPU_SCHED`. For instance `export STARPU_SCHED=dmda`. Use `help` to get the list of available schedulers.

The **eager** scheduler uses a central task queue, from which all workers draw tasks to work on concurrently. This however does not permit to prefetch data since the scheduling decision is taken late. If a task has a non-0 priority, it is put at the front of the queue.

The **prio** scheduler also uses a central task queue, but sorts tasks by priority (between -5 and 5).

The **random** scheduler uses a queue per worker, and distributes tasks randomly according to assumed worker overall performance.

The **ws** (work stealing) scheduler uses a queue per worker, and schedules a task on the worker which released it by default. When a worker becomes idle, it steals a task from the most loaded worker.

The **lws** (locality work stealing) scheduler uses a queue per worker, and schedules a task on the worker which released it by default. When a worker becomes idle, it steals a task from neighbour workers. It also takes into account priorities.

The **dm** (deque model) scheduler uses task execution performance models into account to perform a HEFT-similar scheduling strategy: it schedules tasks where their termination time will be minimal. The difference with HEFT is that **dm** schedules tasks as soon as they become available, and thus in the order they become available, without taking priorities into account.

The **dmda** (deque model data aware) scheduler is similar to **dm**, but it also takes into account data transfer time.

The **dmdar** (deque model data aware ready) scheduler is similar to **dmda**, but it also sorts tasks on per-worker queues by number of already-available data buffers on the target device.

The **dmdas** (deque model data aware sorted) scheduler is similar to **dmdar**, except that it sorts tasks by priority order, which allows to become even closer to HEFT by respecting priorities after having made the scheduling decision (but it still schedules tasks in the order they become available).

The **heft** (heterogeneous earliest finish time) scheduler is a deprecated alias for **dmda**.

The **pheft** (parallel HEFT) scheduler is similar to **dmda**, it also supports parallel tasks (still experimental). Should not be used when several contexts using it are being executed simultaneously.

The **peager** (parallel eager) scheduler is similar to **eager**, it also supports parallel tasks (still experimental). Should not be used when several contexts using it are being executed simultaneously.

TODO: describe modular schedulers

8.2 Task Distribution Vs Data Transfer

Distributing tasks to balance the load induces data transfer penalty. StarPU thus needs to find a balance between both. The target function that the scheduler **dmda** of StarPU tries to minimize is $\alpha * T_{\text{execution}} + \beta * T_{\text{data_transfer}}$, where $T_{\text{execution}}$ is the estimated execution time of the codelet (usually accurate), and $T_{\text{data_transfer}}$ is the estimated data transfer time. The latter is estimated based on bus calibration before execution start, i.e. with an idle machine, thus without contention. You can force bus re-calibration by running the tool `starpu_calibrate_bus`. The β parameter defaults to 1, but it can be worth trying to tweak it by using `export STARPU_SCHED_BETA=2` for instance, since during real application execution, contention makes transfer times bigger. This is of course imprecise, but in practice, a rough estimation already gives the good results that a precise estimation would give.

8.3 Power-based Scheduling

If the application can provide some power performance model (through the field `starpu_codelet::power_model`), StarPU will take it into account when distributing tasks. The target function that the scheduler **dmda** minimizes becomes $\alpha * T_{\text{execution}} + \beta * T_{\text{data_transfer}} + \gamma * \text{Consumption}$, where Consumption is the estimated task consumption in Joules. To tune this parameter, use `export STARPU_SCHED_GAMMA=3000` for instance, to express that each Joule (i.e kW during 1000us) is worth 3000us execution time penalty. Setting α and β to zero permits to only take into account power consumption.

This is however not sufficient to correctly optimize power: the scheduler would simply tend to run all computations on the most energy-conservative processing unit. To account for the consumption of the whole machine (including idle processing units), the idle power of the machine should be given by setting `export STARPU_IDLE_POWER=200` for 200W, for instance. This value can often be obtained from the machine power supplier.

The power actually consumed by the total execution can be displayed by setting `export STARPU_PROFILING=1 STARPU_WORKER_STATS=1`.

On-line task consumption measurement is currently only supported through the `CL_PROFILING_POWER_CONSUMED` OpenCL extension, implemented in the `Movisim` simulator. Applications can however provide explicit measurements by using the function `starpu_perfmodel_update_history()` (exemplified in [Performance Model Example](#) with the `power_model` performance model). Fine-grain measurement is often not feasible with the feedback provided by the hardware, so the user can for instance run a given task a thousand times, measure the global consumption for that series of tasks, divide it by a thousand, repeat for varying kinds of tasks and task sizes, and eventually feed StarPU with these manual measurements through `starpu_perfmodel_update_history()`. For instance, for CUDA devices, `nvidia-smi -q -d POWER` can be used to get the current consumption in Watt. Multiplying that value by the average duration of a single task gives the consumption of the task in Joules, which can be given to `starpu_perfmodel_update_history()`.

8.4 Static Scheduling

In some cases, one may want to force some scheduling, for instance force a given set of tasks to GPU0, another set to GPU1, etc. while letting some other tasks be scheduled on any other device. This can indeed be useful to guide StarPU into some work distribution, while still letting some degree of dynamism. For instance, to force execution of a task on CUDA0:

```
task->execute_on_a_specific_worker = 1;
task->worker = starpu_worker_get_by_type(
    STARPU_CUDA_WORKER, 0);
```

One can also specify the order in which tasks must be executed by setting the `starpu_task::workerorder` field. If this field is set to a non-zero value, it provides the per-worker consecutive order in which tasks will be executed, starting from 1. For a given of such task, the worker will thus not execute it before all the tasks with smaller order value have been executed, notably in case those tasks are not available yet due to some dependencies. This eventually gives total control of task scheduling, and StarPU will only serve as a "self-timed" task runtime. Of course, the provided order has to be runnable, i.e. a task should not depend on another task bound to the same worker with a bigger order.

Note however that using scheduling contexts while statically scheduling tasks on workers could be tricky. Be careful to schedule the tasks exactly on the workers of the corresponding contexts, otherwise the workers' corresponding scheduling structures may not be allocated or the execution of the application may deadlock. Moreover, the hypervisor should not be used when statically scheduling tasks.

8.5 Defining A New Scheduling Policy

A full example showing how to define a new scheduling policy is available in the StarPU sources in the directory `examples/scheduler/`.

See [Scheduling Policy](#)

```
static struct starpu_sched_policy dummy_sched_policy = {
    .init_sched = init_dummy_sched,
    .deinit_sched = deinit_dummy_sched,
    .add_workers = dummy_sched_add_workers,
    .remove_workers = dummy_sched_remove_workers,
    .push_task = push_task_dummy,
    .push_prio_task = NULL,
    .pop_task = pop_task_dummy,
    .post_exec_hook = NULL,
    .pop_every_task = NULL,
    .policy_name = "dummy",
    .policy_description = "dummy scheduling strategy"
};
```


Chapter 9

Scheduling Contexts

TODO: improve!

9.1 General Ideas

Scheduling contexts represent abstracts sets of workers that allow the programmers to control the distribution of computational resources (i.e. CPUs and GPUs) to concurrent parallel kernels. The main goal is to minimize interferences between the execution of multiple parallel kernels, by partitioning the underlying pool of workers using contexts.

9.2 Creating A Context

By default, the application submits tasks to an initial context, which disposes of all the computation resources available to StarPU (all the workers). If the application programmer plans to launch several parallel kernels simultaneously, by default these kernels will be executed within this initial context, using a single scheduler policy(see [Task Scheduling Policy](#)). Meanwhile, if the application programmer is aware of the demands of these kernels and of the specificity of the machine used to execute them, the workers can be divided between several contexts. These scheduling contexts will isolate the execution of each kernel and they will permit the use of a scheduling policy proper to each one of them.

Scheduling Contexts may be created in two ways: either the programmers indicates the set of workers corresponding to each context (providing he knows the identifiers of the workers running within StarPU), or the programmer does not provide any worker list and leaves the Hypervisor assign workers to each context according to their needs ([Scheduling Context Hypervisor](#))

Both cases require a call to the function `starpu_sched_ctx_create`, which requires as input the worker list (the exact list or a NULL pointer) and the scheduling policy. The latter one can be a character list corresponding to the name of a StarPU predefined policy or the pointer to a custom policy. The function returns an identifier of the context created which you will use to indicate the context you want to submit the tasks to.

```
/* the list of resources the context will manage */
int workerids[3] = {1, 3, 10};

/* indicate the scheduling policy to be used within the context, the list of
workers assigned to it, the number of workers, the name of the context */
int id_ctx = starpu_sched_ctx_create("dmda", workerids, 3, "my_ctx");

/* let StarPU know that the following tasks will be submitted to this context */
starpu_sched_ctx_set_task_context(id);

/* submit the task to StarPU */
starpu_task_submit(task);
```

Note: Parallel greedy and parallel heft scheduling policies do not support the existence of several disjoint contexts

on the machine. Combined workers are constructed depending on the entire topology of the machine, not only the one belonging to a context.

9.3 Modifying A Context

A scheduling context can be modified dynamically. The applications may change its requirements during the execution and the programmer can add additional workers to a context or remove if no longer needed. In the following example we have two scheduling contexts `sched_ctx1` and `sched_ctx2`. After executing a part of the tasks some of the workers of `sched_ctx1` will be moved to context `sched_ctx2`.

```
/* the list of ressources that context 1 will give away */
int workerids[3] = {1, 3, 10};

/* add the workers to context 1 */
starpu_sched_ctx_add_workers(workerids, 3, sched_ctx2);

/* remove the workers from context 2 */
starpu_sched_ctx_remove_workers(workerids, 3, sched_ctx1);
```

9.4 Submitting Tasks To A Context

The application may submit tasks to several contexts either simultaneously or sequentially. If several threads of submission are used the function `starpu_sched_ctx_set_context` may be called just before `starpu_task_submit`. Thus StarPU considers that the current thread will submit tasks to the corresponding context.

When the application may not assign a thread of submission to each context, the id of the context must be indicated by using the function `starpu_task_submit_to_ctx` or the field `STARPU_SCHED_CTX` for `starpu_task_insert()`.

9.5 Deleting A Context

When a context is no longer needed it must be deleted. The application can indicate which context should keep the resources of a deleted one. All the tasks of the context should be executed before doing this. Thus, the programmer may use either a barrier and then delete the context directly, or just indicate that other tasks will not be submitted later on to the context (such that when the last task is executed its workers will be moved to the inheritor) and delete the context at the end of the execution (when a barrier will be used eventually).

```
/* when the context 2 is deleted context 1 inherits its resources */
starpu_sched_ctx_set_inheritor(sched_ctx2, sched_ctx1);

/* submit tasks to context 2 */
for (i = 0; i < ntasks; i++)
    starpu_task_submit_to_ctx(task[i], sched_ctx2);

/* indicate that context 2 finished submitting and that */
/* as soon as the last task of context 2 finished executing */
/* its workers can be moved to the inheritor context */
starpu_sched_ctx_finished_submit(sched_ctx2);

/* wait for the tasks of both contexts to finish */
starpu_task_wait_for_all();

/* delete context 2 */
starpu_sched_ctx_delete(sched_ctx2);

/* delete context 1 */
starpu_sched_ctx_delete(sched_ctx1);
```

9.6 Emptying A Context

A context may have no resources at the beginning or at a certain moment of the execution. Task can still be submitted to these contexts and they will be executed as soon as the contexts will have resources. A list of tasks pending

to be executed is kept and when workers are added to the contexts these tasks start being submitted. However, if resources are never allocated to the context the program will not terminate. If these tasks have low priority the programmer can forbid the application to submit them by calling the function `starpu_sched_ctx_stop_↵task_submission()`.

9.7 Contexts Sharing Workers

Contexts may share workers when a single context cannot execute efficiently enough alone on these workers or when the application decides to express a hierarchy of contexts. The workers apply an algorithm of “Round-Robin” to chose the context on which they will “pop” next. By using the function `starpu_sched_ctx_set_turn_↵to_other_ctx`, the programmer can impose the `workerid` to “pop” in the context `sched_ctx_id` next.

Chapter 10

Scheduling Context Hypervisor

10.1 What Is The Hypervisor

StarPU proposes a platform to construct Scheduling Contexts, to delete and modify them dynamically. A parallel kernel, can thus be isolated into a scheduling context and interferences between several parallel kernels are avoided. If the user knows exactly how many workers each scheduling context needs, he can assign them to the contexts at their creation time or modify them during the execution of the program.

The Scheduling Context Hypervisor Plugin is available for the users who do not dispose of a regular parallelism, who cannot know in advance the exact size of the context and need to resize the contexts according to the behavior of the parallel kernels.

The Hypervisor receives information from StarPU concerning the execution of the tasks, the efficiency of the resources, etc. and it decides accordingly when and how the contexts can be resized. Basic strategies of resizing scheduling contexts already exist but a platform for implementing additional custom ones is available.

10.2 Start the Hypervisor

The Hypervisor must be initialized once at the beginning of the application. At this point a resizing policy should be indicated. This strategy depends on the information the application is able to provide to the hypervisor as well as on the accuracy needed for the resizing procedure. For example, the application may be able to provide an estimation of the workload of the contexts. In this situation the hypervisor may decide what resources the contexts need. However, if no information is provided the hypervisor evaluates the behavior of the resources and of the application and makes a guess about the future. The hypervisor resizes only the registered contexts.

10.3 Interrogate The Runtime

The runtime provides the hypervisor with information concerning the behavior of the resources and the application. This is done by using the `performance_counters` which represent callbacks indicating when the resources are idle or not efficient, when the application submits tasks or when it becomes to slow.

10.4 Trigger the Hypervisor

The resizing is triggered either when the application requires it (`sc_hypervisor_resize_ctxs`) or when the initial distribution of resources alters the performance of the application (the application is too slow or the resource are idle for too long time). If the environment variable `SC_HYPERVISOR_TRIGGER_RESIZE` is set to `speed` the monitored speed of the contexts is compared to a theoretical value computed with a linear program, and the resizing is triggered whenever the two values do not correspond. Otherwise, if the environment variable is set to `idle` the

hypervisor triggers the resizing algorithm whenever the workers are idle for a period longer than the threshold indicated by the programmer. When this happens different resizing strategy are applied that target minimizing the total execution of the application, the instant speed or the idle time of the resources.

10.5 Resizing Strategies

The plugin proposes several strategies for resizing the scheduling context.

The **Application driven** strategy uses the user's input concerning the moment when he wants to resize the contexts. Thus, the users tags the task that should trigger the resizing process. We can set directly the field `starpu_task::hypervisor_tag` or use the macro `STARPU_HYPERVISOR_TAG` in the function `starpu_task_insert()`.

```
task.hypervisor_tag = 2;
```

or

```
starpu_task_insert(&codelet,
    ...,
    STARPU_HYPERVISOR_TAG, 2,
    0);
```

Then the user has to indicate that when a task with the specified tag is executed the contexts should resize.

```
sc_hypervisor_resize(sched_ctx, 2);
```

The user can use the same tag to change the resizing configuration of the contexts if he considers it necessary.

```
sc_hypervisor_ctl(sched_ctx,
    SC_HYPERVISOR_MIN_WORKERS, 6,
    SC_HYPERVISOR_MAX_WORKERS, 12,
    SC_HYPERVISOR_TIME_TO_APPLY, 2,
    NULL);
```

The **Idleness** based strategy moves workers unused in a certain context to another one needing them. (see Users' Input In The Resizing Process)

```
int workerids[3] = {1, 3, 10};
int workerids2[9] = {0, 2, 4, 5, 6, 7, 8, 9, 11};
sc_hypervisor_ctl(sched_ctx_id,
    SC_HYPERVISOR_MAX_IDLE, workerids, 3, 10000.0,
    SC_HYPERVISOR_MAX_IDLE, workerids2, 9, 50000.0,
    NULL);
```

The **Gflops rate** based strategy resizes the scheduling contexts such that they all finish at the same time. The speed of each of them is computed and once one of them is significantly slower the resizing process is triggered. In order to do these computations the user has to input the total number of instructions needed to be executed by the parallel kernels and the number of instruction to be executed by each task.

The number of flops to be executed by a context are passed as parameter when they are registered to the hypervisor, (`sc_hypervisor_register_ctx(sched_ctx_id, flops)`) and the one to be executed by each task are passed when the task is submitted. The corresponding field is `starpu_task::flops` and the corresponding macro in the function `starpu_task_insert()` is `STARPU_FLOPS` (**Caution**: but take care of passing a double, not an integer, otherwise parameter passing will be bogus). When the task is executed the resizing process is triggered.

```
task.flops = 100;
```

or

```
starpu_task_insert(&codelet,
    ...,
    STARPU_FLOPS, (double) 100,
    0);
```

The **Feft** strategy uses a linear program to predict the best distribution of resources such that the application finishes in a minimum amount of time. As for the **Gflops rate** strategy the programmers has to indicate the total number of flops to be executed when registering the context. This number of flops may be updated dynamically during the execution of the application whenever this information is not very accurate from the beginning. The function `sc_hypervisor_update_diff_total_flop` is called in order add or remove a difference to the flops left to be executed. Tasks are provided also the number of flops corresponding to each one of them. During the execution of the application the hypervisor monitors the consumed flops and recomputes the time left and the number of resources to use. The speed of each type of resource is (re)evaluated and inserter in the linear program in order to better adapt to the needs of the application.

The **Teft** strategy uses a linear program too, that considers all the types of tasks and the number of each of them and it tries to allocates resources such that the application finishes in a minimum amount of time. A previous calibration of StarPU would be useful in order to have good predictions of the execution time of each type of task.

The types of tasks may be determines directly by the hypervisor when they are submitted. However there are applications that do not expose all the graph of tasks from the beginning. In this case in order to let the hypervisor know about all the tasks the function `sc_hypervisor_set_type_of_task` will just inform the hypervisor about future tasks without submitting them right away.

The **Ispeed** strategy divides the execution of the application in several frames. For each frame the hypervisor computes the speed of the contexts and tries making them run at the same speed. The strategy requires less contribution from the user as the hypervisor requires only the size of the frame in terms of flops.

```
int workerids[3] = {1, 3, 10};
int workerids2[9] = {0, 2, 4, 5, 6, 7, 8, 9, 11};
sc_hypervisor_ctl(sched_ctx_id,
                  SC_HYPERSVISOR_ISPEED_W_SAMPLE, workerids, 3, 20000000000.0,
                  SC_HYPERSVISOR_ISPEED_W_SAMPLE, workerids2, 9, 200000000000.0
                  ,
                  SC_HYPERSVISOR_ISPEED_CTX_SAMPLE, 60000000000.0,
                  NULL);
```

The **Throughput** strategy focuses on maximizing the throughput of the resources and resizes the contexts such that the machine is running at its maximum efficiency (maximum instant speed of the workers).

10.6 Defining A New Hypervisor Policy

While Scheduling Context Hypervisor Plugin comes with a variety of resizing policies (see [Resizing Strategies](#)), it may sometimes be desirable to implement custom policies to address specific problems. The API described below allows users to write their own resizing policy.

Here an example of how to define a new policy

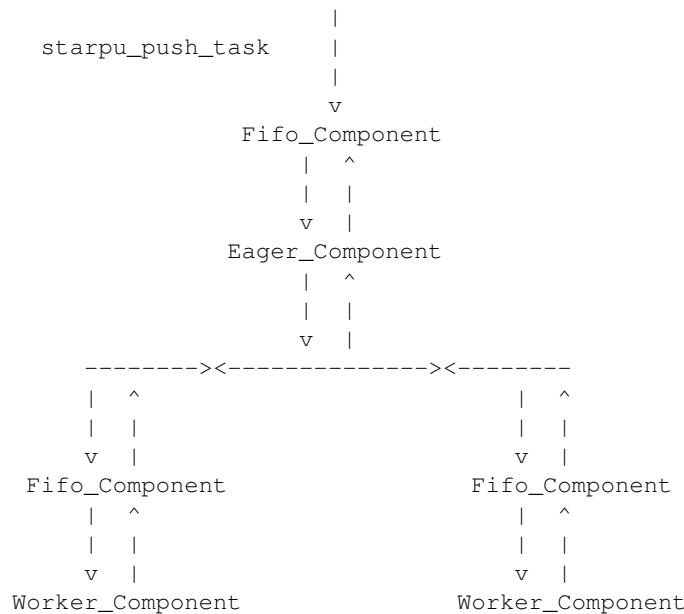
```
struct sc_hypervisor_policy dummy_policy =
{
    .handle_poped_task = dummy_handle_poped_task,
    .handle_pushed_task = dummy_handle_pushed_task,
    .handle_idle_cycle = dummy_handle_idle_cycle,
    .handle_idle_end = dummy_handle_idle_end,
    .handle_post_exec_hook = dummy_handle_post_exec_hook,
    .custom = 1,
    .name = "dummy"
};
```


Chapter 11

Modularized Scheduler

11.1 Introduction

StarPU's Modularized Schedulers are made of individual Scheduling Components Modularizedly assembled as a Scheduling Tree. Each Scheduling Component has an unique purpose, such as prioritizing tasks or mapping tasks over resources. A typical Scheduling Tree is shown below.



When a task is pushed by StarPU in a Modularized Scheduler, the task moves from a Scheduling Component to an other, following the hierarchy of the Scheduling Tree, and is stored in one of the Scheduling Components of the strategy. When a worker wants to pop a task from the Modularized Scheduler, the corresponding Worker Component of the Scheduling Tree tries to pull a task from its parents, following the hierarchy, and gives it to the worker if it succeeded to get one.

11.2 Using Modularized Schedulers

11.2.1 Existing Modularized Schedulers

StarPU is currently shipped with the following pre-defined Modularized Schedulers :

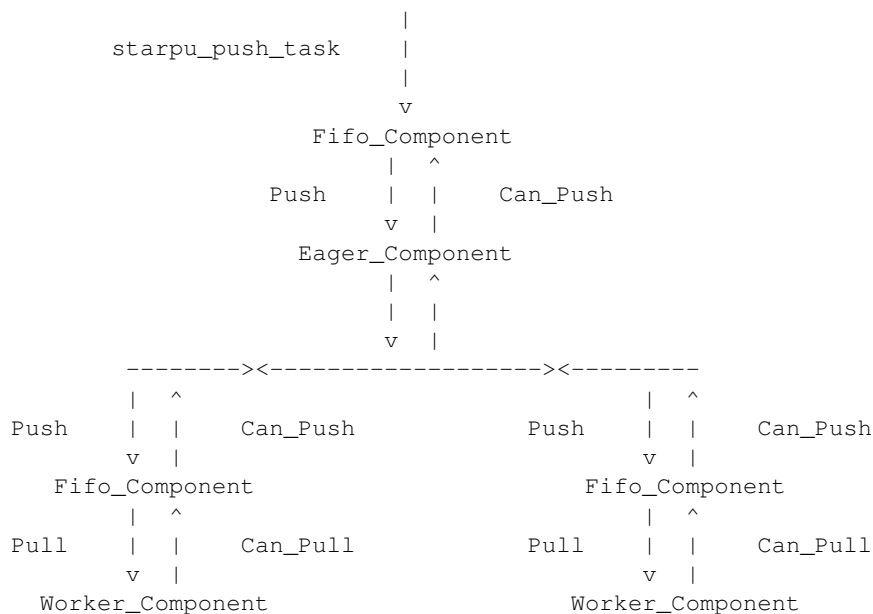
- Eager-based Schedulers (with/without prefetching) :

Naive scheduler, which tries to map a task on the first available resource it finds.

- Prio-based Schedulers (with/without prefetching) :
Similar to Eager-Based Schedulers. Can handle tasks which have a defined priority and schedule them accordingly.
- Random-based Schedulers (with/without prefetching) :
Selects randomly a resource to be mapped on for each task.
- HEFT Scheduler :
Heterogeneous Earliest Finish Time Scheduler. This scheduler needs that every task submitted to StarPU have a defined performance model ([Performance Model Calibration](#)) to work efficiently, but can handle tasks without a performance model.

It is currently needed to set the environment variable `STARPU_SCHED` to use those Schedulers. Modularized Schedulers' naming is tree-*

11.2.2 An Example : The Tree-Eager-Prefetching Strategy



11.2.3 Interface

Each Scheduling Component must follow the following pre-defined Interface to be able to interact with other Scheduling Components.

- Push (Caller_Component, Child_Component, Task)
The calling Scheduling Component transfers a task to its Child Component. When the Push function returns, the task no longer belongs to the calling Component. The Modularized Schedulers' model relies on this function to perform prefetching.
- Pull (Caller_Component, Parent_Component) -> Task
The calling Scheduling Component requests a task from its Parent Component. When the Pull function ends, the returned task belongs to the calling Component.
- Can_Push (Caller_Component, Parent_Component)
The calling Scheduling Component notifies its Parent Component that it is ready to accept new tasks.
- Can_Pull (Caller_Component, Child_Component)
The calling Scheduling Component notifies its Child Component that it is ready to give new tasks.

11.3 Build a Modularized Scheduler

11.3.1 Pre-implemented Components

StarPU is currently shipped with the following four Scheduling Components :

- Flow-control Components : Fifo, Prio
Components which store tasks. They can also prioritize them if they have a defined priority. It is possible to define a threshold for those Components following two criterias : the number of tasks stored in the Component, or the sum of the expected length of all tasks stored in the Component.
- Resource-Mapping Components : Mct, Heft, Eager, Random, Work-Stealing
"Core" of the Scheduling Strategy, those Components are the ones who make scheduling choices.
- Worker Components : Worker
Each Worker Component modelize a concrete worker.
- Special-Purpose Components : Perfmodel_Select, Best_Implementation
Components dedicated to original purposes. The Perfmodel_Select Component decides which Resource-Mapping Component should be used to schedule a task. The Best_Implementation Component chooses which implementation of a task should be used on the choosen resource.

11.3.2 Progression And Validation Rules

Some rules must be followed to ensure the correctness of a Modularized Scheduler :

- At least one Flow-control Component without threshold per Worker Component is needed in a Modularized Scheduler, to store incoming tasks from StarPU and to give tasks to Worker Components who asks for it. It is possible to use one Flow-control Component per Worker Component, or one for all Worker Components, depending on how the Scheduling Tree is defined.
- At least one Resource-Mapping Component is needed in a Modularized Scheduler. Resource-Mapping Components are the only ones who can make scheduling choices, and so the only ones who can have several child.

11.3.3 Implement a Modularized Scheduler

The following code shows how the Tree-Eager-Prefetching Scheduler shown in Section [An Example : The Tree-Eager-Prefetching Strategy](#) is implemented :

```
#define _STARPU_SCHED_NTASKS_THRESHOLD_DEFAULT 2
#define _STARPU_SCHED_EXP_LEN_THRESHOLD_DEFAULT 1000000000.0

static void initialize_eager_prefetching_center_policy(unsigned sched_ctx_id)
{
    unsigned ntasks_threshold = _STARPU_SCHED_NTASKS_THRESHOLD_DEFAULT;
    double exp_len_threshold = _STARPU_SCHED_EXP_LEN_THRESHOLD_DEFAULT;

    [...]

    starpu_sched_ctx_create_worker_collection
        (sched_ctx_id, STARPU_WORKER_LIST);

    /* Create the Scheduling Tree */
    struct starpu_sched_tree * t =
        starpu_sched_tree_create(sched_ctx_id);

    /* The Root Component is a Flow-control Fifo Component */
    t->root = starpu_sched_component_fifo_create(NULL);

    /* The Resource-mapping Component of the strategy is an Eager Component */
    struct starpu_sched_component * eager_component =
        starpu_sched_component_eager_create(NULL);
```

```

/* Create links between Components : the Eager Component is the child
 * of the Root Component */
t->root->add_child
    (t->root, eager_component);
eager_component->add_father
    (eager_component, t->root);

/* A task threshold is set for the Flow-control Components which will
 * be connected to Worker Components. By doing so, this Modularized
 * Scheduler will be able to perform some prefetching on the resources
 */
struct starpu_sched_component_fifo_data fifo_data =
{
    .ntasks_threshold = ntasks_threshold,
    .exp_len_threshold = exp_len_threshold,
};

unsigned i;
for(i = 0;
    i < starpu_worker_get_count() +
    starpu_combined_worker_get_count();
    i++)
{
    /* Each Worker Component has a Flow-control Fifo Component as
     * father */
    struct starpu_sched_component * worker_component =
        starpu_sched_component_worker_get(i);
    struct starpu_sched_component * fifo_component =
        starpu_sched_component_fifo_create(&fifo_data);
    fifo_component->add_child
        (fifo_component, worker_component);
    worker_component->add_father
        (worker_component, fifo_component);

    /* Each Flow-control Fifo Component associated to a Worker
     * Component is linked to the Eager Component as one of its
     * children */
    eager_component->add_child
        (eager_component, fifo_component);
    fifo_component->add_father
        (fifo_component, eager_component);
}

starpu_sched_tree_update_workers(t);
starpu_sched_ctx_set_policy_data
    (sched_ctx_id, (void*)t);
}

/* Properly destroy the Scheduling Tree and all its Components */
static void deinitialize_eager_prefetching_center_policy(unsigned sched_ctx_id)
{
    struct starpu_sched_tree * tree =
        (struct starpu_sched_tree*)
        starpu_sched_ctx_get_policy_data(sched_ctx_id);
    starpu_sched_tree_destroy(tree);
    starpu_sched_ctx_delete_worker_collection
        (sched_ctx_id);
}

/* Initializing the starpu_sched_policy struct associated to the Modularized
 * Scheduler : only the init_sched and deinit_sched needs to be defined to
 * implement a Modularized Scheduler */
struct starpu_sched_policy _starpu_sched_tree_eager_prefetching_policy =
{
    .init_sched = initialize_eager_prefetching_center_policy,
    .deinit_sched = deinitialize_eager_prefetching_center_policy,
    .add_workers = starpu_sched_tree_add_workers,
    .remove_workers = starpu_sched_tree_remove_workers,
    .push_task = starpu_sched_tree_push_task,
    .pop_task = starpu_sched_tree_pop_task,
    .pre_exec_hook = starpu_sched_component_worker_pre_exec_hook,
    .post_exec_hook = starpu_sched_component_worker_post_exec_hook,

    .pop_every_task = NULL,
    .policy_name = "tree-eager-prefetching",
    .policy_description = "eager with prefetching tree policy"
};

```

11.4 Write a Scheduling Component

Chapter 12

Debugging Tools

StarPU provides several tools to help debugging applications. Execution traces can be generated and displayed graphically, see [Generating Traces With FxT](#).

Some gdb helpers are also provided to show the whole StarPU state:

```
(gdb) source tools/gdbinit
(gdb) help starpu
```

Valgrind can be used on StarPU: valgrind.h just needs to be found at ./configure time, to tell valgrind about some known false positives and disable host memory pinning. Other known false positives can be suppressed by giving the suppression files in tools/valgrind/ *.suppr to valgrind's `--suppressions` option.

The environment variable [STARPU_DISABLE_KERNELS](#) can also be set to 1 to make StarPU do everything (schedule tasks, transfer memory, etc.) except actually calling the application-provided kernel functions, i.e. the computation will not happen. This permits to quickly check that the task scheme is working properly.

The Temanejo task debugger can also be used, see [Using The Temanejo Task Debugger](#).

12.1 Using The Temanejo Task Debugger

StarPU can connect to Temanejo `>= 1.0rc2` (see <http://www.hlr.s.de/temanejo>), to permit nice visual task debugging. To do so, build Temanejo's `libayudame.so`, install `Ayudame.h` to e.g. `/usr/local/include`, apply the `tools/patch-ayudame` to it to fix C build, re-`./configure`, make sure that it found it, rebuild StarPU. Run the Temanejo GUI, give it the path to your application, any options you want to pass it, the path to `libayudame.so`.

Make sure to specify at least the same number of CPUs in the dialog box as your machine has, otherwise an error will happen during execution. Future versions of Temanejo should be able to tell StarPU the number of CPUs to use.

Tag numbers have to be below `4000000000000000000ULL` to be usable for Temanejo (so as to distinguish them from tasks).

Chapter 13

Online Performance Tools

13.1 On-line Performance Feedback

13.1.1 Enabling On-line Performance Monitoring

In order to enable online performance monitoring, the application can call `starpu_profiling_status_set()` with the parameter `STARPU_PROFILING_ENABLE`. It is possible to detect whether monitoring is already enabled or not by calling `starpu_profiling_status_get()`. Enabling monitoring also reinitialize all previously collected feedback. The environment variable `STARPU_PROFILING` can also be set to 1 to achieve the same effect. The function `starpu_profiling_init()` can also be called during the execution to reinitialize performance counters and to start the profiling if the environment variable `STARPU_PROFILING` is set to 1.

Likewise, performance monitoring is stopped by calling `starpu_profiling_status_set()` with the parameter `STARPU_PROFILING_DISABLE`. Note that this does not reset the performance counters so that the application may consult them later on.

More details about the performance monitoring API are available in [Profiling](#).

13.1.2 Per-task Feedback

If profiling is enabled, a pointer to a structure `starpu_profiling_task_info` is put in the field `starpu_task::profiling_info` when a task terminates. This structure is automatically destroyed when the task structure is destroyed, either automatically or by calling `starpu_task_destroy()`.

The structure `starpu_profiling_task_info` indicates the date when the task was submitted (`starpu_profiling_task_info::submit_time`), started (`starpu_profiling_task_info::start_time`), and terminated (`starpu_profiling_task_info::end_time`), relative to the initialization of StarPU with `starpu_init()`. It also specifies the identifier of the worker that has executed the task (`starpu_profiling_task_info::workerid`). These date are stored as `timespec` structures which the user may convert into micro-seconds using the helper function `starpu_timing_timespec_to_us()`.

It is worth noting that the application may directly access this structure from the callback executed at the end of the task. The structure `starpu_task` associated to the callback currently being executed is indeed accessible with the function `starpu_task_get_current()`.

13.1.3 Per-codelet Feedback

The field `starpu_codelet::per_worker_stats` is an array of counters. The *i*-th entry of the array is incremented every time a task implementing the codelet is executed on the *i*-th worker. This array is not reinitialized when profiling is enabled or disabled.

13.1.4 Per-worker Feedback

The second argument returned by the function `starpu_profiling_worker_get_info()` is a structure `starpu_profiling_worker_info` that gives statistics about the specified worker. This structure specifies when StarPU started collecting profiling information for that worker (`starpu_profiling_worker_info::start_time`), the duration of the profiling measurement interval (`starpu_profiling_worker_info::total_time`), the time spent executing kernels (`starpu_profiling_worker_info::executing_time`), the time spent sleeping because there is no task to execute at all (`starpu_profiling_worker_info::sleeping_time`), and the number of tasks that were executed while profiling was enabled. These values give an estimation of the proportion of time spent do real work, and the time spent either sleeping because there are not enough executable tasks or simply wasted in pure StarPU overhead.

Calling `starpu_profiling_worker_get_info()` resets the profiling information associated to a worker.

When an FxT trace is generated (see [Generating Traces With FxT](#)), it is also possible to use the tool `starpu_workers_activity` (see [Monitoring Activity](#)) to generate a graphic showing the evolution of these values during the time, for the different workers.

13.1.5 Bus-related Feedback

TODO: ajouter `STARPU_BUS_STATS`

The bus speed measured by StarPU can be displayed by using the tool `starpu_machine_display`, for instance:

```
StarPU has found:
  3 CUDA devices
      CUDA 0 (Tesla C2050 02:00.0)
      CUDA 1 (Tesla C2050 03:00.0)
      CUDA 2 (Tesla C2050 84:00.0)
from   to RAM      to CUDA 0      to CUDA 1      to CUDA 2
RAM    0.000000    5176.530428    5176.492994    5191.710722
CUDA 0 4523.732446    0.000000      2414.074751    2417.379201
CUDA 1 4523.718152    2414.078822    0.000000      2417.375119
CUDA 2 4534.229519    2417.069025    2417.060863    0.000000
```

13.1.6 StarPU-Top Interface

StarPU-Top is an interface which remotely displays the on-line state of a StarPU application and permits the user to change parameters on the fly.

Variables to be monitored can be registered by calling the functions `starpu_top_add_data_boolean()`, `starpu_top_add_data_integer()`, `starpu_top_add_data_float()`, e.g.:

```
starpu_top_data *data = starpu_top_add_data_integer("mynum", 0, 1
00, 1);
```

The application should then call `starpu_top_init_and_wait()` to give its name and wait for StarPU-Top to get a start request from the user. The name is used by StarPU-Top to quickly reload a previously-saved layout of parameter display.

```
starpu_top_init_and_wait("the application");
```

The new values can then be provided thanks to `starpu_top_update_data_boolean()`, `starpu_top_update_data_integer()`, `starpu_top_update_data_float()`, e.g.:

```
starpu_top_update_data_integer(data, mynum);
```

Updateable parameters can be registered thanks to `starpu_top_register_parameter_boolean()`, `starpu_top_register_parameter_integer()`, `starpu_top_register_parameter_float()`, e.g.:

```
float alpha;
starpu_top_register_parameter_float("alpha", &alpha, 0, 10, modif_hook);
```

modif_hook is a function which will be called when the parameter is being modified, it can for instance print the new value:

```
void modif_hook(struct starpu_top_param *d) {
    fprintf(stderr, "%s has been modified: %f\n", d->name, alpha);
}
```

Task schedulers should notify StarPU-Top when it has decided when a task will be scheduled, so that it can show it in its Gantt chart, for instance:

```
starpu_top_task_prevision(task, workerid, begin, end);
```

Starting StarPU-Top (StarPU-Top is started via the binary `starpu_top`.) and the application can be done two ways:

- The application is started by hand on some machine (and thus already waiting for the start event). In the Preference dialog of StarPU-Top, the SSH checkbox should be unchecked, and the hostname and port (default is 2011) on which the application is already running should be specified. Clicking on the connection button will thus connect to the already-running application.
- StarPU-Top is started first, and clicking on the connection button will start the application itself (possibly on a remote machine). The SSH checkbox should be checked, and a command line provided, e.g.:

```
$ ssh myserver STARPU_SCHED=dmda ./application
```

If port 2011 of the remote machine can not be accessed directly, an ssh port bridge should be added:

```
$ ssh -L 2011:localhost:2011 myserver STARPU_SCHED=dmda ./application
```

and "localhost" should be used as IP Address to connect to.

13.2 Task And Worker Profiling

A full example showing how to use the profiling API is available in the StarPU sources in the directory `examples/profiling/`.

```
struct starpu_task *task = starpu_task_create();
task->cl = &cl;
task->synchronous = 1;
/* We will destroy the task structure by hand so that we can
 * query the profiling info before the task is destroyed. */
task->destroy = 0;

/* Submit and wait for completion (since synchronous was set to 1) */
starpu_task_submit(task);

/* The task is finished, get profiling information */
struct starpu_profiling_task_info *info = task->
    profiling_info;

/* How much time did it take before the task started ? */
double delay += starpu_timing_timespec_delay_us(&info->
    submit_time, &info->start_time);

/* How long was the task execution ? */
double length += starpu_timing_timespec_delay_us(&info->
    start_time, &info->end_time);

/* We don't need the task structure anymore */
starpu_task_destroy(task);
```

```

/* Display the occupancy of all workers during the test */
int worker;
for (worker = 0; worker < starpu_worker_get_count(); worker++)
{
    struct starpu_profiling_worker_info worker_info;
    int ret = starpu_profiling_worker_get_info(worker, &worker_info);
    STARPU_ASSERT(!ret);

    double total_time = starpu_timing_timespec_to_us(&worker_info
.total_time);
    double executing_time = starpu_timing_timespec_to_us(&
worker_info.executing_time);
    double sleeping_time = starpu_timing_timespec_to_us(&
worker_info.sleeping_time);
    double overhead_time = total_time - executing_time - sleeping_time;

    float executing_ratio = 100.0*executing_time/total_time;
    float sleeping_ratio = 100.0*sleeping_time/total_time;
    float overhead_ratio = 100.0 - executing_ratio - sleeping_ratio;

    char workername[128];
    starpu_worker_get_name(worker, workername, 128);
    fprintf(stderr, "Worker %s:\n", workername);
    fprintf(stderr, "\ttotal time: %.21f ms\n", total_time*1e-3);
    fprintf(stderr, "\texec time: %.21f ms (%.2f %%)\n",
        executing_time*1e-3, executing_ratio);
    fprintf(stderr, "\tblocked time: %.21f ms (%.2f %%)\n",
        sleeping_time*1e-3, sleeping_ratio);
    fprintf(stderr, "\toverhead time: %.21f ms (%.2f %%)\n",
        overhead_time*1e-3, overhead_ratio);
}

```

13.3 Performance Model Example

To achieve good scheduling, StarPU scheduling policies need to be able to estimate in advance the duration of a task. This is done by giving to codelets a performance model, by defining a structure `starpu_perfmodel` and providing its address in the field `starpu_codelet::model`. The fields `starpu_perfmodel::symbol` and `starpu_perfmodel::type` are mandatory, to give a name to the model, and the type of the model, since there are several kinds of performance models. For compatibility, make sure to initialize the whole structure to zero, either by using explicit `memset()`, or by letting the compiler implicitly do it as exemplified below.

- Measured at runtime (model type `STARPU_HISTORY_BASED`). This assumes that for a given set of data input/output sizes, the performance will always be about the same. This is very true for regular kernels on GPUs for instance (<0.1% error), and just a bit less true on CPUs (~=1% error). This also assumes that there are few different sets of data input/output sizes. StarPU will then keep record of the average time of previous executions on the various processing units, and use it as an estimation. History is done per task size, by using a hash of the input and output sizes as an index. It will also save it in `$STARPU_HOME/.starpu/sampling/codelets` for further executions, and can be observed by using the tool `starpu_perfmodel_display`, or drawn by using the tool `starpu_perfmodel_plot` ([Performance Model Calibration](#)). The models are indexed by machine name. To share the models between machines (e.g. for a homogeneous cluster), use `export STARPU_HOSTNAME=some_global_name`. Measurements are only done when using a task scheduler which makes use of it, such as `dmda`. Measurements can also be provided explicitly by the application, by using the function `starpu_perfmodel_update_history()`.

The following is a small code example.

If e.g. the code is recompiled with other compilation options, or several variants of the code are used, the symbol string should be changed to reflect that, in order to recalibrate a new model from zero. The symbol string can even be constructed dynamically at execution time, as long as this is done before submitting any task using it.

```

static struct starpu_perfmodel mult_perf_model = {
    .type = STARPU_HISTORY_BASED,
    .symbol = "mult_perf_model"
};

struct starpu_codelet cl = {
    .cpu_funcs = { cpu_mult },
    .cpu_funcs_name = { "cpu_mult" },
    .nbuffers = 3,

```

```
.modes = { STARPU_R, STARPU_R, STARPU_W },
/* for the scheduling policy to be able to use performance models */
.model = &mult_perf_model
};
```

- Measured at runtime and refined by regression (model types [STARPU_REGRESSION_BASED](#) and [STARPU_NL_REGRESSION_BASED](#)). This still assumes performance regularity, but works with various data input sizes, by applying regression over observed execution times. [STARPU_REGRESSION_BASED](#) uses an $a \cdot n^b$ regression form, [STARPU_NL_REGRESSION_BASED](#) uses an $a \cdot n^b + c$ (more precise than [STARPU_REGRESSION_BASED](#), but costs a lot more to compute).

For instance, `tests/perfmodels/regression_based.c` uses a regression-based performance model for the function `memset()`.

Of course, the application has to issue tasks with varying size so that the regression can be computed. StarPU will not trust the regression unless there is at least 10% difference between the minimum and maximum observed input size. It can be useful to set the environment variable [STARPU_CALIBRATE](#) to 1 and run the application on varying input sizes with [STARPU_SCHED](#) set to `dmda` scheduler, so as to feed the performance model for a variety of inputs. The application can also provide the measurements explicitly by using the function `starpu_perfmodel_update_history()`. The tools `starpu_perfmodel_display` and `starpu_perfmodel_plot` can be used to observe how much the performance model is calibrated ([Performance Model Calibration](#)); when their output look good, [STARPU_CALIBRATE](#) can be reset to 0 to let StarPU use the resulting performance model without recording new measures, and [STARPU_SCHED](#) can be set to `dmda` to benefit from the performance models. If the data input sizes vary a lot, it is really important to set [STARPU_CALIBRATE](#) to 0, otherwise StarPU will continue adding the measures, and result with a very big performance model, which will take time a lot of time to load and save.

For non-linear regression, since computing it is quite expensive, it is only done at termination of the application. This means that the first execution of the application will use only history-based performance model to perform scheduling, without using regression.

- Provided as an estimation from the application itself (model type [STARPU_COMMON](#) and field `starpu_perfmodel::cost_function`), see for instance `examples/common/blas_model.h` and `examples/common/blas_model.c`.
- Provided explicitly by the application (model type [STARPU_PER_ARCH](#)): the fields `.per_arch[arch][nimpl].cost_function` have to be filled with pointers to functions which return the expected duration of the task in micro-seconds, one per architecture.

For [STARPU_HISTORY_BASED](#), [STARPU_REGRESSION_BASED](#), and [STARPU_NL_REGRESSION_BASED](#), the dimensions of task data (both input and output) are used as an index by default. [STARPU_HISTORY_BASED](#) uses a CRC hash of the dimensions as an index to distinguish histories, and [STARPU_REGRESSION_BASED](#) and [STARPU_NL_REGRESSION_BASED](#) use the total size as an index for the regression.

The `starpu_perfmodel::size_base` and `starpu_perfmodel::footprint` fields however permit the application to override that, when for instance some of the data do not matter for task cost (e.g. mere reference table), or when using sparse structures (in which case it is the number of non-zeros which matter), or when there is some hidden parameter such as the number of iterations, or when the application actually has a very good idea of the complexity of the algorithm, and just not the speed of the processor, etc. The example in the directory `examples/pi` uses this to include the number of iterations in the base size. `starpu_perfmodel::size_base` should be used when the variance of the actual performance is known (i.e. bigger returned value is longer execution time), and thus particularly useful for [STARPU_REGRESSION_BASED](#) or [STARPU_NL_REGRESSION_BASED](#). `starpu_perfmodel::footprint` can be used when the variance of the actual performance is unknown (irregular performance behavior, etc.), and thus only useful for [STARPU_HISTORY_BASED](#). `starpu_task_data_footprint()` can be used as a base and combined with other parameters through `starpu_hash_crc32c_be` for instance.

StarPU will automatically determine when the performance model is calibrated, or rather, it will assume the performance model is calibrated until the application submits a task for which the performance can not be predicted. For [STARPU_HISTORY_BASED](#), StarPU will require 10 (`STARPU_CALIBRATE_MINIMUM`) measurements for a given size before estimating that an average can be taken as estimation for further executions with the same size. For [STARPU_REGRESSION_BASED](#) and [STARPU_NL_REGRESSION_BASED](#), StarPU will require 10 (`STARPU_CALIBRATE_MINIMUM`) measurements, and that the minimum measured data size is smaller than 90% of the maximum measured data size (i.e. the measurement interval is large enough for a regression to have a meaning).

Calibration can also be forced by setting the `STARPU_CALIBRATE` environment variable to 1, or even reset by setting it to 2.

How to use schedulers which can benefit from such performance model is explained in [Task Scheduling Policy](#).

The same can be done for task power consumption estimation, by setting the field `starpu_codelet::power_model` the same way as the field `starpu_codelet::model`. Note: for now, the application has to give to the power consumption performance model a name which is different from the execution time performance model.

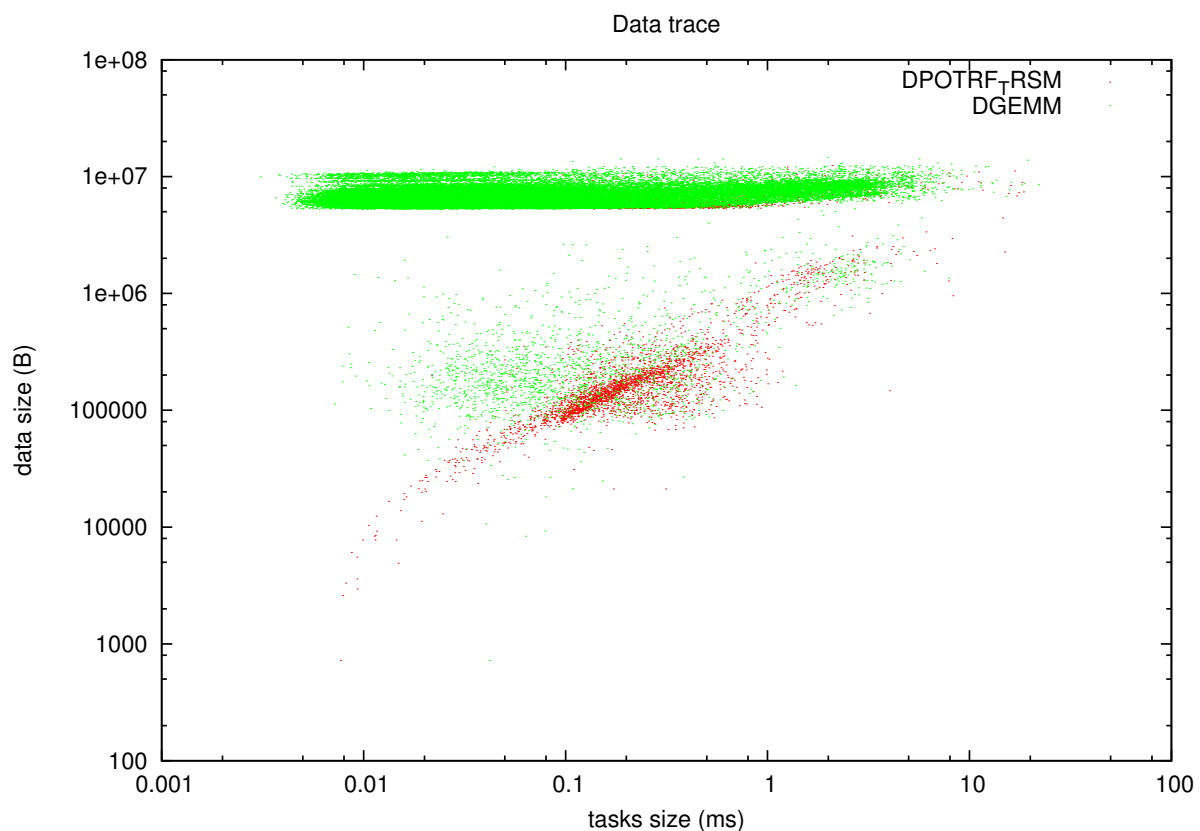
The application can request time estimations from the StarPU performance models by filling a task structure as usual without actually submitting it. The data handles can be created by calling any of the functions `starpu_*_data_register` with a `NULL` pointer and `-1` node and the desired data sizes, and need to be unregistered as usual. The functions `starpu_task_expected_length()` and `starpu_task_expected_power()` can then be called to get an estimation of the task cost on a given arch. `starpu_task_footprint()` can also be used to get the footprint used for indexing history-based performance models. `starpu_task_destroy()` needs to be called to destroy the dummy task afterwards. See `tests/perfmodels/regression_based.c` for an example.

13.4 Data trace and tasks length

It is possible to get statistics about tasks length and data size by using :

```
$ starpu_fxt_data_trace filename [codelet1 codelet2 ... codeletn]
```

Where filename is the FxT trace file and codeletX the names of the codelets you want to profile (if no names are specified, `starpu_fxt_data_trace` will profile them all). This will create a file, `data_trace.gp` which can be executed to get a `.eps` image of these results. On the image, each point represents a task, and each color corresponds to a codelet.



Chapter 14

Offline Performance Tools

To get an idea of what is happening, a lot of performance feedback is available, detailed in this chapter. The various informations should be checked for.

- What does the Gantt diagram look like? (see [Creating a Gantt Diagram](#))
 - If it's mostly green (tasks running in the initial context) or context specific color prevailing, then the machine is properly utilized, and perhaps the codelets are just slow. Check their performance, see [Performance Of Codelets](#).
 - If it's mostly purple (FetchingInput), tasks keep waiting for data transfers, do you perhaps have far more communication than computation? Did you properly use CUDA streams to make sure communication can be overlapped? Did you use data-locality aware schedulers to avoid transfers as much as possible?
 - If it's mostly red (Blocked), tasks keep waiting for dependencies, do you have enough parallelism? It might be a good idea to check what the DAG looks like (see [Creating a DAG With Graphviz](#)).
 - If only some workers are completely red (Blocked), for some reason the scheduler didn't assign tasks to them. Perhaps the performance model is bogus, check it (see [Performance Of Codelets](#)). Do all your codelets have a performance model? When some of them don't, the schedulers switches to a greedy algorithm which thus performs badly.

You can also use the Temanejo task debugger (see [Using The Temanejo Task Debugger](#)) to visualize the task graph more easily.

14.1 Off-line Performance Feedback

14.1.1 Generating Traces With FxT

StarPU can use the FxT library (see <https://savannah.nongnu.org/projects/fkt/>) to generate traces with a limited runtime overhead.

You can either get a tarball:

```
$ wget http://download.savannah.gnu.org/releases/fkt/fxt-0.2.11.tar.gz
```

or use the FxT library from CVS (autotools are required):

```
$ cvs -d :pserver:anonymous@cvs.sv.gnu.org:/sources/fkt co FxT
$ ./bootstrap
```

Compiling and installing the FxT library in the `$FXTDIR` path is done following the standard procedure:

```
$ ./configure --prefix=$FXTDIR
$ make
$ make install
```

In order to have StarPU to generate traces, StarPU should be configured with the option `--with-fxt` :

```
$ ./configure --with-fxt=$FXTDIR
```

Or you can simply point the `PKG_CONFIG_PATH` to `$FXTDIR/lib/pkgconfig` and pass `--with-fxt` to `./configure`

When FxT is enabled, a trace is generated when StarPU is terminated by calling `starpu_shutdown()`. The trace is a binary file whose name has the form `prof_file_XXX_YYY` where `XXX` is the user name, and `YYY` is the pid of the process that used StarPU. This file is saved in the `/tmp/` directory by default, or by the directory specified by the environment variable `STARPU_FXT_PREFIX`.

The additional configure option `--enable-fxt-lock` can be used to generate trace events which describes the locks behaviour during the execution.

14.1.2 Creating a Gantt Diagram

When the FxT trace file `filename` has been generated, it is possible to generate a trace in the Paje format by calling:

```
$ starpu_fxt_tool -i filename
```

Or alternatively, setting the environment variable `STARPU_GENERATE_TRACE` to 1 before application execution will make StarPU do it automatically at application shutdown.

This will create a file `paje.trace` in the current directory that can be inspected with the ViTE (<http://vite.gforge.inria.fr/>) trace visualizing open-source tool. It is possible to open the file `paje.trace` with ViTE by using the following command:

```
$ vite paje.trace
```

To get names of tasks instead of "unknown", fill the optional `starpu_codelet::name`, or use a performance model for them. Details of the codelet execution can be obtained by passing `--enable-paje-codelet-details` when configuring StarPU and using a recent enough version of ViTE (at least r1430).

In the MPI execution case, collect the trace files from the MPI nodes, and specify them all on the command `starpu_fxt_tool`, for instance:

```
$ starpu_fxt_tool -i filename1 -i filename2
```

By default, all tasks are displayed using a green color. To display tasks with varying colors, pass option `-c` to `starpu_fxt_tool`.

To identify tasks precisely, the application can set the `starpu_task::tag_id` field of the task (or use `STARPU_TAG_ONLY` when using `starpu_task_insert()`), and with a recent enough version of vite (`>= r1430`) and the `--enable-paje-codelet-details` StarPU configure option, the value of the tag will show up in the trace.

Traces can also be inspected by hand by using the tool `fxt_print`, for instance:

```
$ fxt_print -o -f filename
```

Timings are in nanoseconds (while timings as seen in `vite` are in milliseconds).

14.1.3 Creating a DAG With Graphviz

When the FxT trace file `filename` has been generated, it is possible to generate a task graph in the DOT format by calling:

```
$ starpu_fxt_tool -i filename
```

This will create a `dag.dot` file in the current directory. This file is a task graph described using the DOT language. It is possible to get a graphical output of the graph by using the graphviz library:

```
$ dot -Tpdf dag.dot -o output.pdf
```

14.1.4 Monitoring Activity

When the FxT trace file `filename` has been generated, it is possible to generate an activity trace by calling:

```
$ starpu_fxt_tool -i filename
```

This will create a file `activity.data` in the current directory. A profile of the application showing the activity of StarPU during the execution of the program can be generated:

```
$ starpu_workers_activity activity.data
```

This will create a file named `activity.eps` in the current directory. This picture is composed of two parts. The first part shows the activity of the different workers. The green sections indicate which proportion of the time was spent executed kernels on the processing unit. The red sections indicate the proportion of time spent in StartPU: an important overhead may indicate that the granularity may be too low, and that bigger tasks may be appropriate to use the processing unit more efficiently. The black sections indicate that the processing unit was blocked because there was no task to process: this may indicate a lack of parallelism which may be alleviated by creating more tasks when it is possible.

The second part of the picture `activity.eps` is a graph showing the evolution of the number of tasks available in the system during the execution. Ready tasks are shown in black, and tasks that are submitted but not schedulable yet are shown in grey.

14.2 Performance Of Codelets

The performance model of codelets (see [Performance Model Example](#)) can be examined by using the tool `starpu_perfmodel_display`:

```
$ starpu_perfmodel_display -l
file: <malloc_pinned.hannibal>
file: <starpu_slu_lu_model_21.hannibal>
file: <starpu_slu_lu_model_11.hannibal>
file: <starpu_slu_lu_model_22.hannibal>
file: <starpu_slu_lu_model_12.hannibal>
```

Here, the codelets of the example `lu` are available. We can examine the performance of the kernel 22 (in micro-seconds), which is history-based:

```
$ starpu_perfmodel_display -s starpu_slu_lu_model_22
performance model for cpu
# hash      size      mean      dev      n
57618ab0    19660800    2.851069e+05  1.829369e+04  109
performance model for cuda_0
# hash      size      mean      dev      n
57618ab0    19660800    1.164144e+04  1.556094e+01  315
```

```

performance model for cuda_1
# hash      size      mean      dev      n
57618ab0    19660800    1.164271e+04    1.330628e+01    360
performance model for cuda_2
# hash      size      mean      dev      n
57618ab0    19660800    1.166730e+04    3.390395e+02    456

```

We can see that for the given size, over a sample of a few hundreds of execution, the GPUs are about 20 times faster than the CPUs (numbers are in us). The standard deviation is extremely low for the GPUs, and less than 10% for CPUs.

This tool can also be used for regression-based performance models. It will then display the regression formula, and in the case of non-linear regression, the same performance log as for history-based performance models:

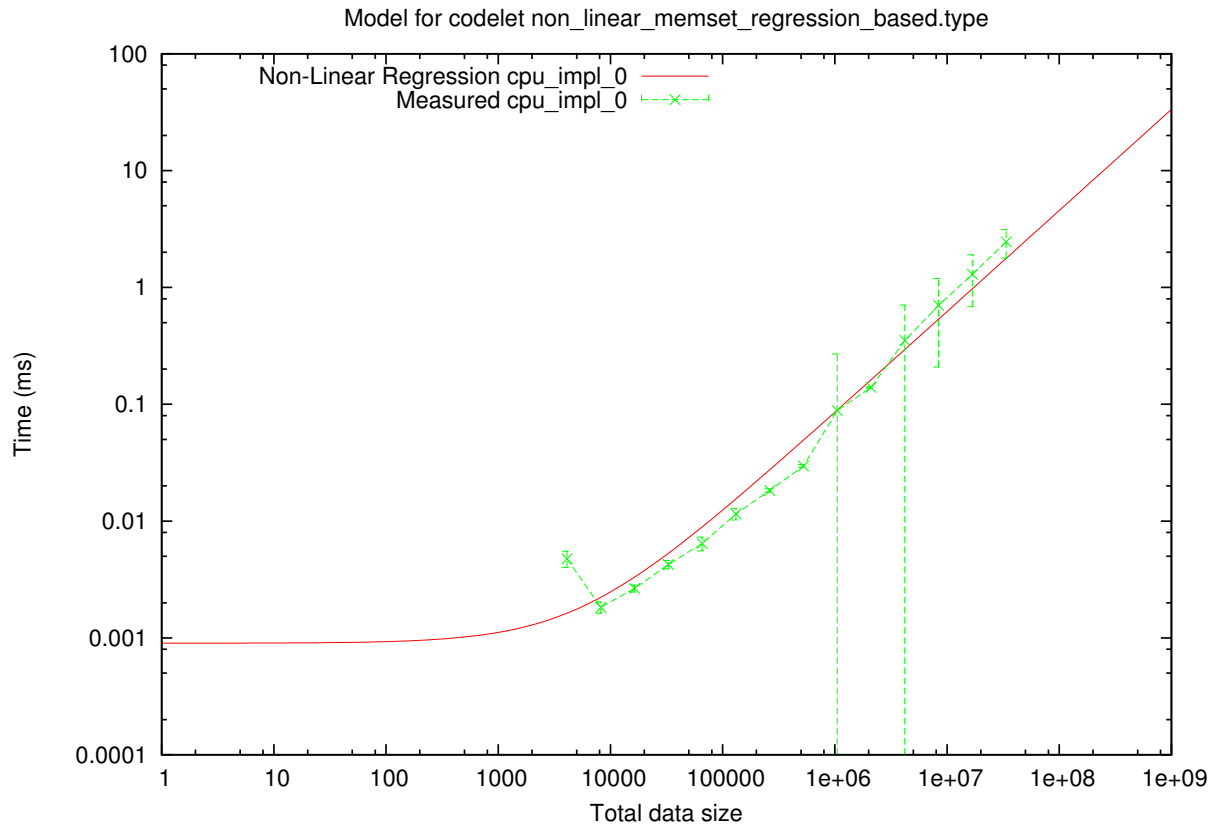
```

$ starpu_perfmodel_display -s non_linear_memset_regression_based
performance model for cpu_impl_0
Regression : #sample = 1400
Linear: y = alpha size ^ beta
alpha = 1.335973e-03
beta = 8.024020e-01
Non-Linear: y = a size ^b + c
a = 5.429195e-04
b = 8.654899e-01
c = 9.009313e-01
# hash size mean stddev n
a3d3725e 4096          4.763200e+00    7.650928e-01    100
870a30aa 8192          1.827970e+00    2.037181e-01    100
48e988e9 16384          2.652800e+00    1.876459e-01    100
961e65d2 32768          4.255530e+00    3.518025e-01    100
...

```

The same can also be achieved by using StarPU's library API, see [Performance Model](#) and notably the function `starpu_perfmodel_load_symbol()`. The source code of the tool `starpu_perfmodel_display` can be a useful example.

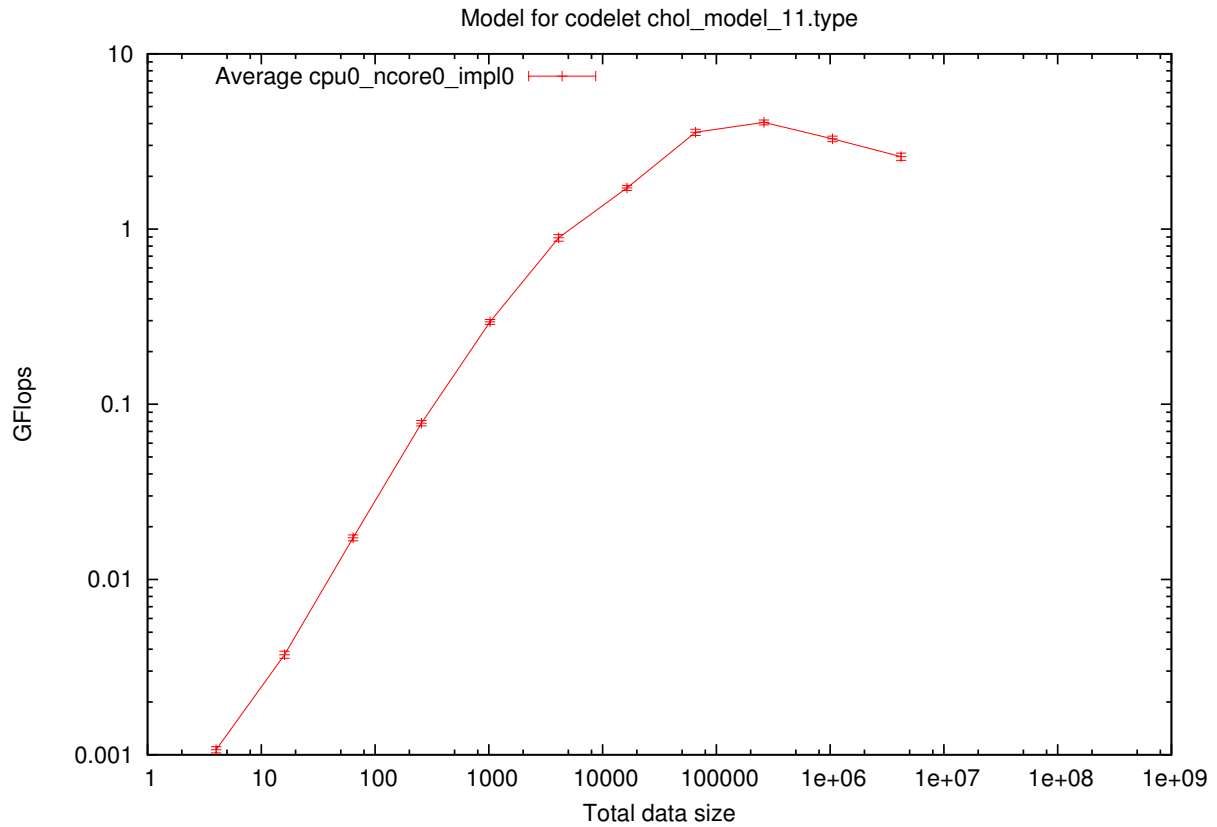
The tool `starpu_perfmodel_plot` can be used to draw performance models. It writes a `.gp` file in the current directory, to be run with the tool `gnuplot`, which shows the corresponding curve.



When the field `starpu_task::flops` is set, `starpu_perfmodel_plot` can directly draw a GFlops curve, by simply adding the `-f` option:

```
$ starpu_perfmodel_plot -f -s chol_model_11
```

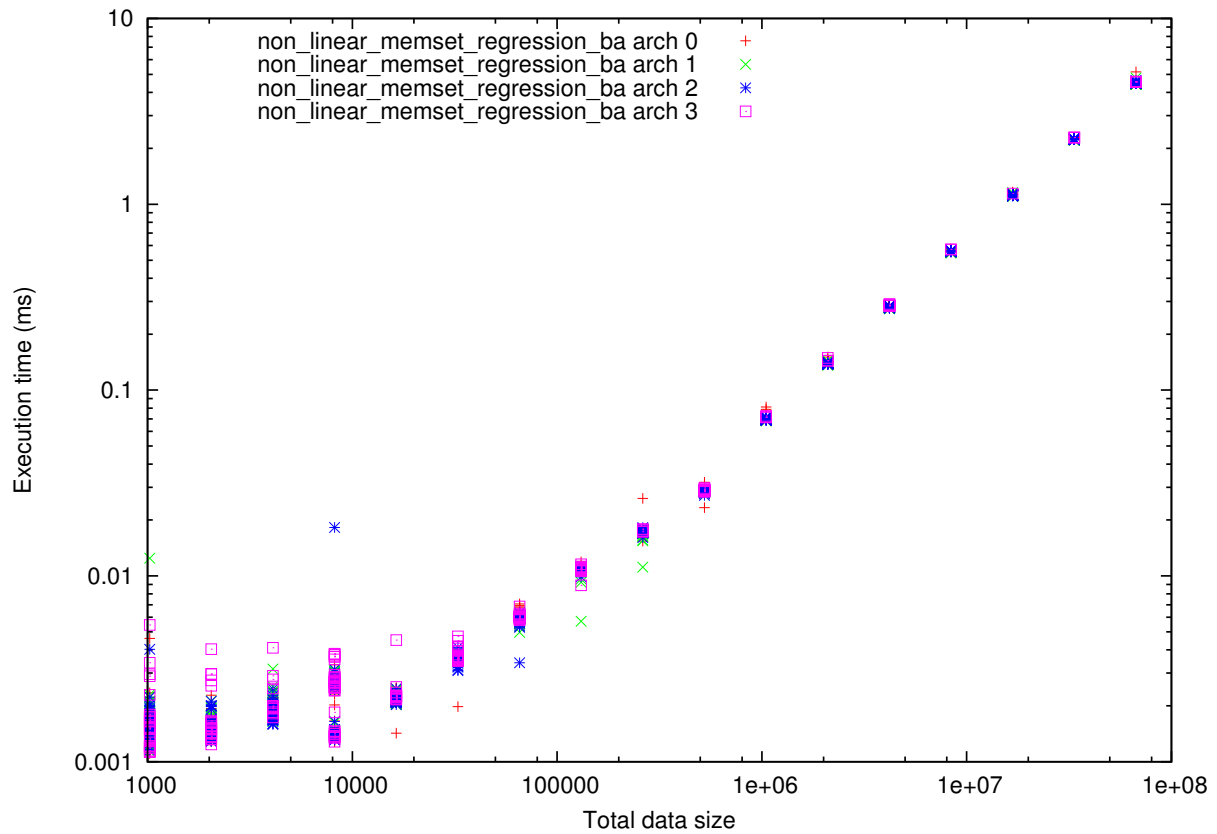
This will however disable displaying the regression model, for which we can not compute GFlops.



When the FxT trace file `filename` has been generated, it is possible to get a profiling of each codelet by calling:

```
$ starpu_fxt_tool -i filename
$ starpu_codelet_profile distrib.data codelet_name
```

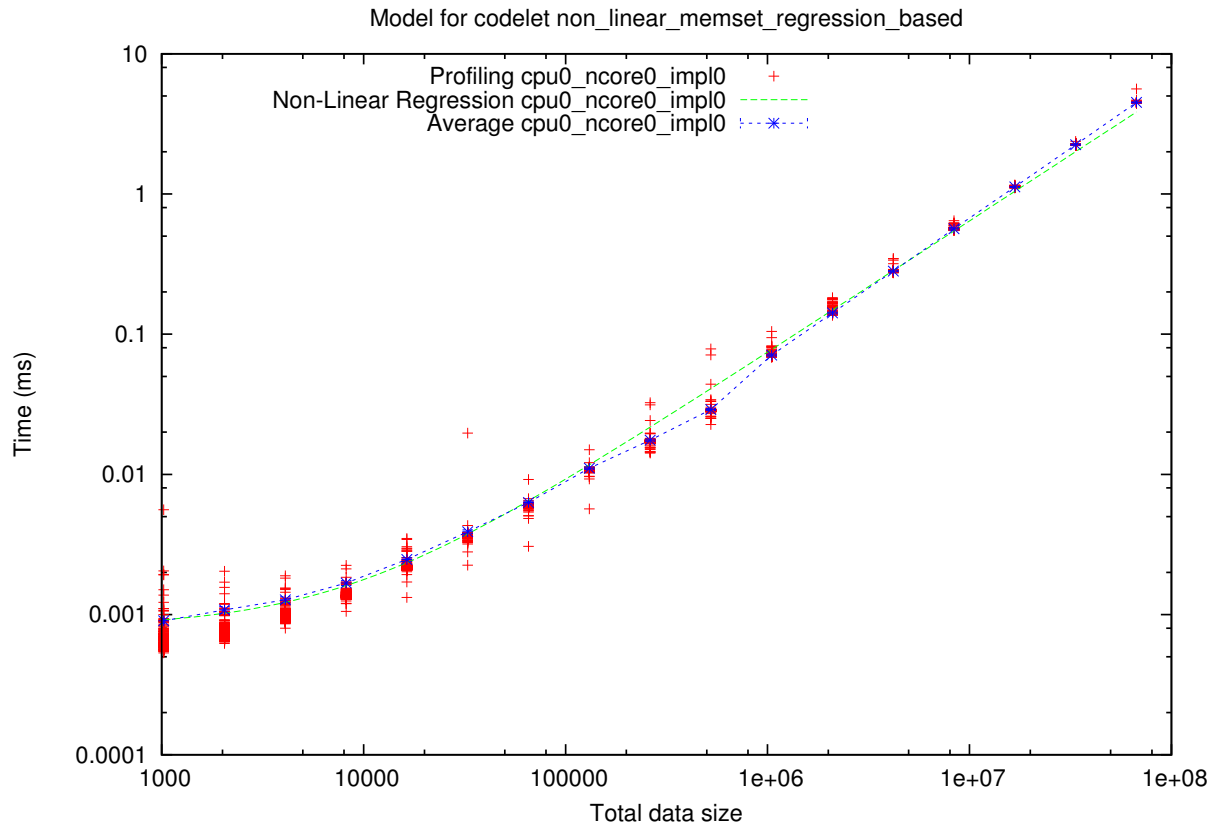
This will create profiling data files, and a `.gp` file in the current directory, which draws the distribution of codelet time over the application execution, according to data input size.



This is also available in the tool `starpu_perfmodel_plot`, by passing it the fxt trace:

```
$ starpu_perfmodel_plot -s non_linear_memset_regression_based -i /tmp/prof_file_foo_0
```

It will produce a `.gp` file which contains both the performance model curves, and the profiling measurements.

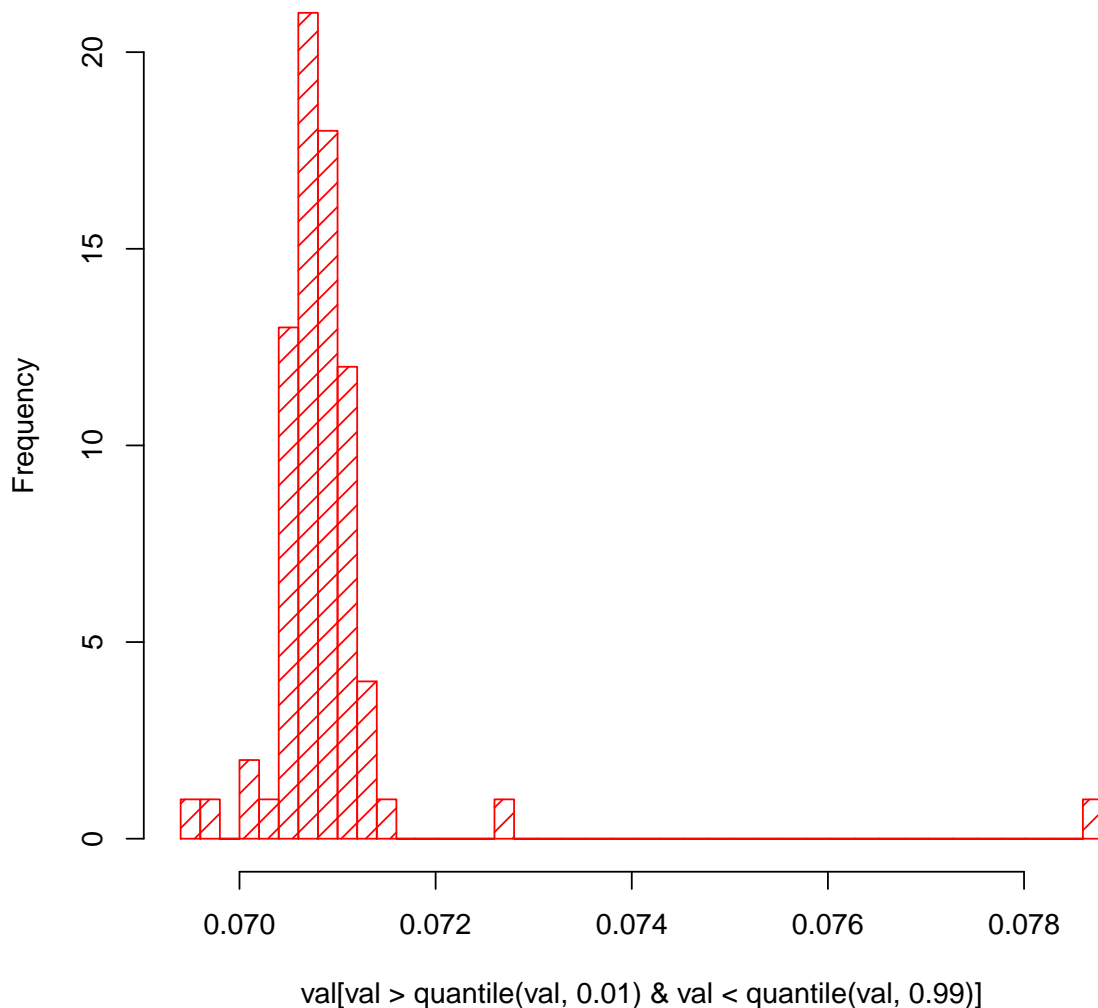


If you have the statistical tool R installed, you can additionally use

```
$ starpu_codelet_histo_profile distrib.data
```

Which will create one .pdf file per codelet and per input size, showing a histogram of the codelet execution time distribution.

Histogram of $\text{val}[\text{val} > \text{quantile}(\text{val}, 0.01) \ \& \ \text{val} < \text{quantile}(\text{val}, 0.99)]$



14.3 Trace statistics

More than just codelet performance, it is interesting to get statistics over all kinds of StarPU states (allocations, data transfers, etc.). This is particularly useful to check what may have gone wrong in the accuracy of the simgrid simulation.

This requires the R statistical tool, with the `plyr`, `ggplot2` and `data.table` packages. If your system distribution does not have packages for these, one can fetch them from CRAN:

```
$ R
> install.packages("plyr")
> install.packages("ggplot2")
> install.packages("data.table")
> install.packages("knitr")
```

The `pj_dump` tool from `pajeng` is also needed (see <https://github.com/schnorr/pajeng>)

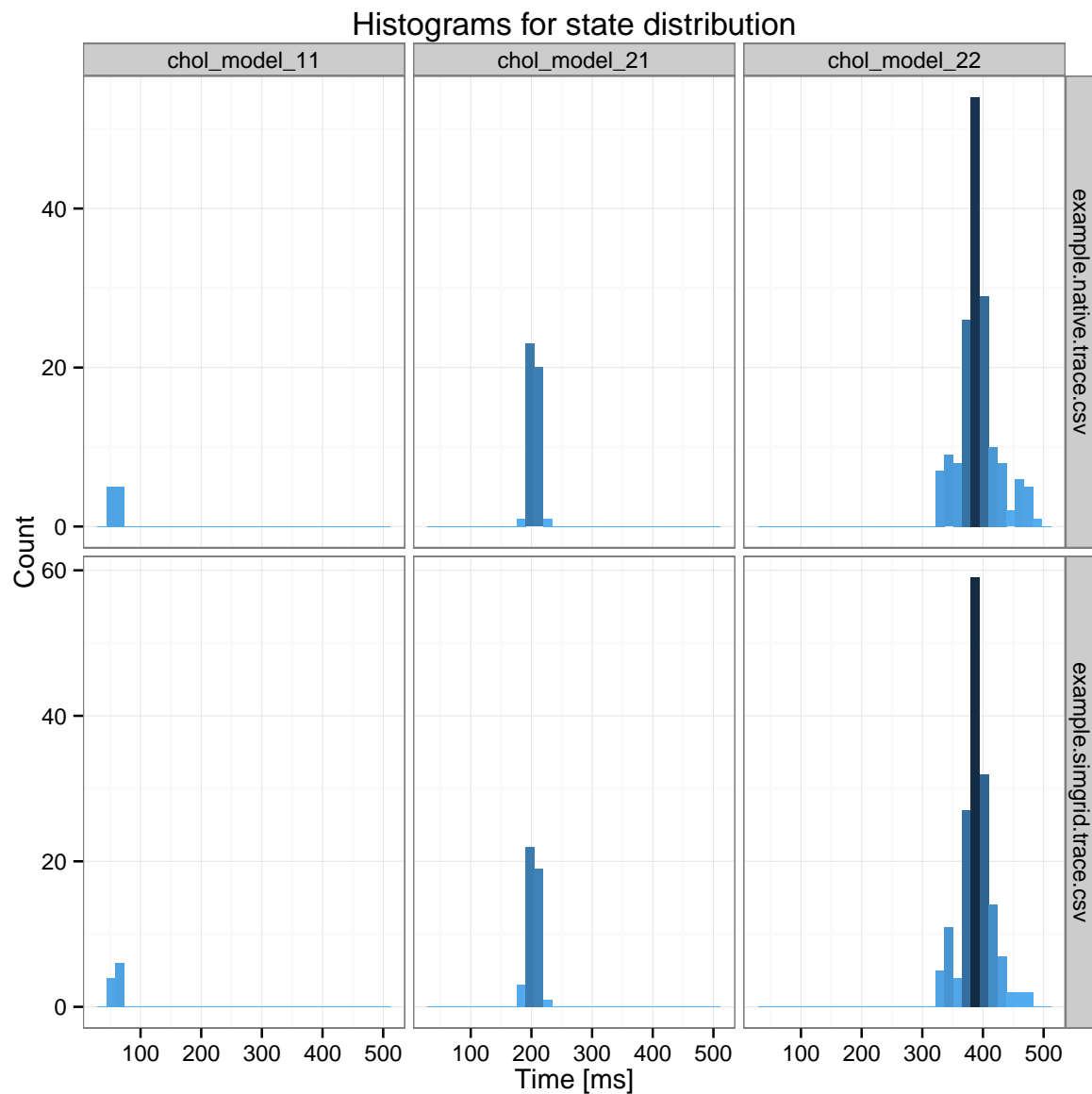
One can then get textual or .csv statistics over the trace states:

```
$ starpu_paje_state_stats -v native.trace simgrid.trace
"Value"          "Events_native.csv" "Duration_native.csv" "Events_simgrid.csv" "Duration_simgrid.csv"
"Callback"       220                0.075978             220                0
"chol_model_11"  10                565.176              10                572.8695
"chol_model_21"  45                9184.828             45                9170.719
"chol_model_22" 165                64712.07             165               64299.203
$ starpu_paje_state_stats native.trace simgrid.trace
```

And one can plot histograms of execution times, of several states for instance:

```
$ starpu_paje_draw_histogram -n chol_model_11,chol_model_21,chol_model_22 native.trace simgrid.trace
```

and see the resulting pdf file:



A quick statistical report can be generated by using:

```
$ starpu_paje_summary native.trace simgrid.trace
```

it includes gantt charts, execution summaries, as well as state duration charts and time distribution histograms.

14.4 Theoretical Lower Bound On Execution Time

StarPU can record a trace of what tasks are needed to complete the application, and then, by using a linear system, provide a theoretical lower bound of the execution time (i.e. with an ideal scheduling).

The computed bound is not really correct when not taking into account dependencies, but for an application which have enough parallelism, it is very near to the bound computed with dependencies enabled (which takes a huge lot more time to compute), and thus provides a good-enough estimation of the ideal execution time.

[Theoretical Lower Bound On Execution Time Example](#) provides an example on how to use this.

14.5 Theoretical Lower Bound On Execution Time Example

For kernels with history-based performance models (and provided that they are completely calibrated), StarPU can very easily provide a theoretical lower bound for the execution time of a whole set of tasks. See for instance `examples/lu/lu_example.c`: before submitting tasks, call the function `starpu_bound_start()`, and after complete execution, call `starpu_bound_stop()`. `starpu_bound_print_lp()` or `starpu_bound_print_mps()` can then be used to output a Linear Programming problem corresponding to the schedule of your tasks. Run it through `lp_solve` or any other linear programming solver, and that will give you a lower bound for the total execution time of your tasks. If StarPU was compiled with the library `glpk` installed, `starpu_bound_compute()` can be used to solve it immediately and get the optimized minimum, in ms. Its parameter `integer` allows to decide whether integer resolution should be computed and returned

The `deps` parameter tells StarPU whether to take tasks, implicit data, and tag dependencies into account. Tags released in a callback or similar are not taken into account, only tags associated with a task are. It must be understood that the linear programming problem size is quadratic with the number of tasks and thus the time to solve it will be very long, it could be minutes for just a few dozen tasks. You should probably use `lp_solve -timeout 1 test.pl -wmps test.mps` to convert the problem to MPS format and then use a better solver, `glpsol` might be better than `lp_solve` for instance (the `-pcost` option may be useful), but sometimes doesn't manage to converge. `cbc` might look slower, but it is parallel. For `lp_solve`, be sure to try at least all the `-B` options. For instance, we often just use `lp_solve -cc -B1 -Bb -Bg -Bp -Bf -Br -BG -Bd -Bs -BB -Bo -Bc -Bi`, and the `-gr` option can also be quite useful. The resulting schedule can be observed by using the tool `starpu_lp2paje`, which converts it into the Paje format.

Data transfer time can only be taken into account when `deps` is set. Only data transfers inferred from implicit data dependencies between tasks are taken into account. Other data transfers are assumed to be completely overlapped.

Setting `deps` to 0 will only take into account the actual computations on processing units. It however still properly takes into account the varying performances of kernels and processing units, which is quite more accurate than just comparing StarPU performances with the fastest of the kernels being used.

The `prio` parameter tells StarPU whether to simulate taking into account the priorities as the StarPU scheduler would, i.e. schedule prioritized tasks before less prioritized tasks, to check to which extend this results to a less optimal solution. This increases even more computation time.

14.6 Memory Feedback

It is possible to enable memory statistics. To do so, you need to pass the option `--enable-memory-stats` when running `configure`. It is then possible to call the function `starpu_data_display_memory_stats()` to display statistics about the current data handles registered within StarPU.

Moreover, statistics will be displayed at the end of the execution on data handles which have not been cleared out. This can be disabled by setting the environment variable `STARPU_MEMORY_STATS` to 0.

For example, if you do not unregister data at the end of the complex example, you will get something similar to:

```
$ STARPU_MEMORY_STATS=0 ./examples/interface/complex
Complex[0] = 45.00 + 12.00 i
```

```

Complex[0] = 78.00 + 78.00 i
Complex[0] = 45.00 + 12.00 i
Complex[0] = 45.00 + 12.00 i

$ STARPU_MEMORY_STATS=1 ./examples/interface/complex
Complex[0] = 45.00 + 12.00 i
Complex[0] = 78.00 + 78.00 i
Complex[0] = 45.00 + 12.00 i
Complex[0] = 45.00 + 12.00 i

#-----
Memory stats:
#-----
Data on Node #3
#-----
Data : 0x553ff40
Size : 16

#--
Data access stats
/!\ Work Underway
Node #0
Direct access : 4
Loaded (Owner) : 0
Loaded (Shared) : 0
Invalidated (was Owner) : 0

Node #3
Direct access : 0
Loaded (Owner) : 0
Loaded (Shared) : 1
Invalidated (was Owner) : 0

#-----
Data : 0x5544710
Size : 16

#--
Data access stats
/!\ Work Underway
Node #0
Direct access : 2
Loaded (Owner) : 0
Loaded (Shared) : 1
Invalidated (was Owner) : 1

Node #3
Direct access : 0
Loaded (Owner) : 1
Loaded (Shared) : 0
Invalidated (was Owner) : 0

```

14.7 Data Statistics

Different data statistics can be displayed at the end of the execution of the application. To enable them, you need to pass the option `--enable-stats` when calling `configure`. When calling `starpu_shutdown()` various statistics will be displayed, execution, MSI cache statistics, allocation cache statistics, and data transfer statistics. The display can be disabled by setting the environment variable `STARPU_STATS` to 0.

```

$ ./examples/cholesky/cholesky_tag
Computation took (in ms)
518.16
Synthetic GFlops : 44.21
#-----
MSI cache stats :
TOTAL MSI stats hit 1622 (66.23 %) miss 827 (33.77 %)
...

$ STARPU_STATS=0 ./examples/cholesky/cholesky_tag

```

Computation took (in ms)
518.16
Synthetic GFlops : 44.21

Chapter 15

Frequently Asked Questions

15.1 How To Initialize A Computation Library Once For Each Worker?

Some libraries need to be initialized once for each concurrent instance that may run on the machine. For instance, a C++ computation class which is not thread-safe by itself, but for which several instantiated objects of that class can be used concurrently. This can be used in StarPU by initializing one such object per worker. For instance, the libstarpufft example does the following to be able to use FFTW on CPUs.

Some global array stores the instantiated objects:

```
fftw_plan plan_cpu[STARPU_NMAXWORKERS];
```

At initialisation time of libstarpup, the objects are initialized:

```
int workerid;
for (workerid = 0; workerid < starpu_worker_get_count(); workerid++) {
    switch (starpu_worker_get_type(workerid)) {
        case STARPU_CPU_WORKER:
            plan_cpu[workerid] = fftw_plan(...);
            break;
    }
}
```

And in the codelet body, they are used:

```
static void fft(void *descr[], void *_args)
{
    int workerid = starpu_worker_get_id();
    fftw_plan plan = plan_cpu[workerid];
    ...

    fftw_execute(plan, ...);
}
```

This however is not sufficient for FFT on CUDA: initialization has to be done from the workers themselves. This can be done thanks to [starpu_execute_on_each_worker\(\)](#). For instance libstarpufft does the following.

```
static void fft_plan_gpu(void *args)
{
    plan plan = args;
    int n2 = plan->n2[0];
    int workerid = starpu_worker_get_id();

    cufftPlan1d(&plan->plans[workerid].plan_cuda, n, _CUFFT_C2C, 1);
    cufftSetStream(plan->plans[workerid].plan_cuda, starpu_cuda_get_local_stream(
        ));
}
void starpufft_plan(void)
{
    starpu_execute_on_each_worker(fft_plan_gpu, plan,
        STARPU_CUDA);
}
```

15.2 Using The Driver API

Running Drivers

```
int ret;
struct starpu_driver = {
    .type = STARPU_CUDA_WORKER,
    .id.cuda_id = 0
};
ret = starpu_driver_init(&d);
if (ret != 0)
    error();
while (some_condition) {
    ret = starpu_driver_run_once(&d);
    if (ret != 0)
        error();
}
ret = starpu_driver_deinit(&d);
if (ret != 0)
    error();
```

To add a new kind of device to the structure `starpu_driver`, one needs to:

1. Add a member to the union `starpu_driver::id`
2. Modify the internal function `_starpu_launch_drivers()` to make sure the driver is not always launched.
3. Modify the function `starpu_driver_run()` so that it can handle another kind of architecture.
4. Write the new function `_starpu_run_foobar()` in the corresponding driver.

15.3 On-GPU Rendering

Graphical-oriented applications need to draw the result of their computations, typically on the very GPU where these happened. Technologies such as OpenGL/CUDA interoperability permit to let CUDA directly work on the OpenGL buffers, making them thus immediately ready for drawing, by mapping OpenGL buffer, textures or renderbuffer objects into CUDA. CUDA however imposes some technical constraints: peer memcopy has to be disabled, and the thread that runs OpenGL has to be the one that runs CUDA computations for that GPU.

To achieve this with StarPU, pass the option `--disable-cuda-memcpy-peer` to `./configure` (TODO: make it dynamic), OpenGL/GLUT has to be initialized first, and the interoperability mode has to be enabled by using the field `starpu_conf::cuda_opengl_interoperability`, and the driver loop has to be run by the application, by using the field `starpu_conf::not_launched_drivers` to prevent StarPU from running it in a separate thread, and by using `starpu_driver_run()` to run the loop. The examples `gl_interop` and `gl_interop_idle` show how it articulates in a simple case, where rendering is done in task callbacks. The former uses `glutMainLoopEvent` to make GLUT progress from the StarPU driver loop, while the latter uses `glutIdleFunc` to make StarPU progress from the GLUT main loop.

Then, to use an OpenGL buffer as a CUDA data, StarPU simply needs to be given the CUDA pointer at registration, for instance:

```
/* Get the CUDA worker id */
for (workerid = 0; workerid < starpu_worker_get_count(); workerid++)
    if (starpu_worker_get_type(workerid) ==
        STARPU_CUDA_WORKER)
        break;

/* Build a CUDA pointer pointing at the OpenGL buffer */
cudaGraphicsResourceGetMappedPointer((void**)&output, &num_bytes, resource);

/* And register it to StarPU */
starpu_vector_data_register(&handle,
    starpu_worker_get_memory_node(workerid),
    output, num_bytes / sizeof(float4), sizeof(float4));

/* The handle can now be used as usual */
starpu_task_insert(&cl, STARPU_RW, handle, 0);
```

```
/* ... */
/* This gets back data into the OpenGL buffer */
starpu_data_unregister(handle);
```

and display it e.g. in the callback function.

15.4 Using StarPU With MKL 11 (Intel Composer XE 2013)

Some users had issues with MKL 11 and StarPU (versions 1.1rc1 and 1.0.5) on Linux with MKL, using 1 thread for MKL and doing all the parallelism using StarPU (no multithreaded tasks), setting the environment variable `MKL_NUM_THREADS` to 1, and using the threaded MKL library, with `imlp5`.

Using this configuration, StarPU uses only 1 core, no matter the value of `STARPU_NCPU`. The problem is actually a thread pinning issue with MKL.

The solution is to set the environment variable `KMP_AFFINITY` to disabled (http://software.intel.com/sites/products/documentation/studio/composer/en-us/2011Update/compiler_c/optaps/common/optaps_openmp_thread_affinity.htm).

15.5 Thread Binding on NetBSD

When using StarPU on a NetBSD machine, if the topology discovery library `hwloc` is used, thread binding will fail. To prevent the problem, you should at least use the version 1.7 of `hwloc`, and also issue the following call:

```
$ sysctl -w security.models.extensions.user_set_cpu_affinity=1
```

Or add the following line in the file `/etc/sysctl.conf`

```
security.models.extensions.user_set_cpu_affinity=1
```

15.6 Interleaving StarPU and non-StarPU code

If your application only partially uses StarPU, and you do not want to call `starpu_init()` / `starpu_shutdown()` at the beginning/end of each section, StarPU workers will poll for work between the sections. To avoid this behavior, you can "pause" StarPU with the `starpu_pause()` function. This will prevent the StarPU workers from accepting new work (tasks that are already in progress will not be frozen), and stop them from polling for more work.

Note that this does not prevent you from submitting new tasks, but they won't execute until `starpu_resume()` is called. Also note that StarPU must not be paused when you call `starpu_shutdown()`, and that this function pair works in a push/pull manner, ie you need to match the number of calls to these functions to clear their effect.

One way to use these functions could be:

```
starpu_init(NULL);
starpu_pause(); // To submit all the tasks without a single one executing
submit_some_tasks();
starpu_resume(); // The tasks start executing

starpu_task_wait_for_all();
starpu_pause(); // Stop the workers from polling

starpu_resume();
starpu_shutdown();
```


Part IV

StarPU Extensions

Chapter 16

Out Of Core

16.1 Introduction

When using StarPU, one may need to store more data than what the main memory (RAM) can store. This part describes the method to add a new memory node on a disk and to use it.

The principle is that one first registers a disk location, seen by StarPU as a `void*`, which can be for instance a Unix path for the `stdio` or `unistd` case, or a database file path for a `leveldb` case, etc. The disk backend opens this place with the `plug` method.

If the disk backend provides an `alloc` method, StarPU can then start using it to allocate room and store data there with the `write` method, without user intervention.

The user can also use `starpu_disk_open` to explicitly open an object within the disk, e.g. a file name in the `stdio` or `unistd` cases, or a database key in the `leveldb` case, and then use `starpu_*_register` functions to turn it into a StarPU data handle. StarPU will then automatically read and write data as appropriate.

16.2 Use a new disk memory

To use a disk memory node, you have to register it with this function:

```
int new_dd = starpu_disk_register(&starpu_disk_unistd_ops, (void *) "/tmp/", 1024*1024*200);
```

Here, we use the `unistd` library to realize the read/write operations, i.e. `fread/fwrite`. This structure must have a path where to store files, as well as the maximum size the software can afford storing on the disk.

Don't forget to check if the result is correct!

This can also be achieved by just setting environment variables:

```
export STARPU_DISK_SWAP=/tmp
export STARPU_DISK_SWAP_BACKEND=unistd
export STARPU_DISK_SWAP_SIZE=$((200*1024*1024))
```

When the register function is called, StarPU will benchmark the disk. This can take some time.

Warning: the size thus has to be at least 1 MB!

StarPU will automatically try to evict unused data to this new disk. One can also use the standard StarPU memory node API, see the [Standard Memory Library](#) and the [Data Interfaces](#).

The disk is unregistered during the `starpu_shutdown()`.

16.3 Disk functions

There are various ways to operate a disk memory node, described by the structure [starpu_disk_ops](#). For instance, the variable [starpu_disk_unistd_ops](#) uses read/write functions.

All structures are in [Out Of Core](#) .

16.4 Examples: disk_copy

```

/* Try to write into disk memory
 * Use mechanism to push datas from main ram to disk ram
 */

#include <starpu.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

/* size of one vector */
#define NX (30*1000000/sizeof(double))
#define FPRINTF(o, file, fmt, ...) do { if (!getenv("STARPU_SSILENT")) { fprintf(o, file, fmt, ## __VA_ARGS__); } } while(0)

int main(int argc, char **argv)
{
    double * A,*B,*C,*D,*E,*F;

    /* limit main ram to force to push in disk */
    setenv("STARPU_LIMIT_CPU_MEM", "160", 1);

    /* Initialize StarPU with default configuration */
    int ret = starpu_init(NULL);

    if (ret == -ENODEV) goto enodev;

    /* register a disk */
    int new_dd = starpu_disk_register(&starpu_disk_unistd_ops, (void *) "/tmp/",
    1024*1024*200);
    /* can't write on /tmp/ */
    if (new_dd == -ENOENT) goto enoent;

    /* allocate two memory spaces */
    starpu_malloc_flags((void **)&A, NX*sizeof(double),
    STARPU_MALLOC_COUNT);
    starpu_malloc_flags((void **)&F, NX*sizeof(double),
    STARPU_MALLOC_COUNT);

    FPRINTF(stderr, "TEST DISK MEMORY \n");

    unsigned int j;
    /* initialization with bad values */
    for(j = 0; j < NX; ++j)
    {
        A[j] = j;
        F[j] = -j;
    }

    starpu_data_handle_t vector_handleA, vector_handleB, vector_handleC,
    vector_handleD, vector_handleE, vector_handleF;

    /* register vector in starpu */
    starpu_vector_data_register(&vector_handleA,
    STARPU_MAIN_RAM, (uintptr_t)A, NX, sizeof(double));
    starpu_vector_data_register(&vector_handleB, -1, (uintptr_t) NULL, NX,
    sizeof(double));
    starpu_vector_data_register(&vector_handleC, -1, (uintptr_t) NULL, NX,
    sizeof(double));
    starpu_vector_data_register(&vector_handleD, -1, (uintptr_t) NULL, NX,
    sizeof(double));
    starpu_vector_data_register(&vector_handleE, -1, (uintptr_t) NULL, NX,
    sizeof(double));
    starpu_vector_data_register(&vector_handleF,
    STARPU_MAIN_RAM, (uintptr_t)F, NX, sizeof(double));

    /* copy vector A->B, B->C... */
    starpu_data_cpy(vector_handleB, vector_handleA, 0, NULL, NULL);
    starpu_data_cpy(vector_handleC, vector_handleB, 0, NULL, NULL);
    starpu_data_cpy(vector_handleD, vector_handleC, 0, NULL, NULL);
    starpu_data_cpy(vector_handleE, vector_handleD, 0, NULL, NULL);

```

```

    starpu_data_cpy(vector_handleF, vector_handleE, 0, NULL, NULL);

    /* StarPU does not need to manipulate the array anymore so we can stop
     * monitoring it */

    /* free them */
    starpu_data_unregister(vector_handleA);
    starpu_data_unregister(vector_handleB);
    starpu_data_unregister(vector_handleC);
    starpu_data_unregister(vector_handleD);
    starpu_data_unregister(vector_handleE);
    starpu_data_unregister(vector_handleF);

    /* check if computation is correct */
    int try = 1;
    for (j = 0; j < NX; ++j)
        if (A[j] != F[j])
        {
            printf("Fail A %f != F %f \n", A[j], F[j]);
            try = 0;
        }

    /* free last vectors */
    starpu_free_flags(A, NX*sizeof(double),
    STARPU_MALLOC_COUNT);
    starpu_free_flags(F, NX*sizeof(double),
    STARPU_MALLOC_COUNT);

    /* terminate StarPU, no task can be submitted after */
    starpu_shutdown();

    if(try)
        FPRINTF(stderr, "TEST SUCCESS\n");
    else
        FPRINTF(stderr, "TEST FAIL\n");
    return (try ? EXIT_SUCCESS : EXIT_FAILURE);
}

enodev:
    return 77;
enoent:
    return 77;
}

```

16.5 Examples: disk_compute

```

/* Try to write into disk memory
 * Use mechanism to push datas from main ram to disk ram
 */

#include <starpu.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <math.h>

#define NX (1024)

int main(int argc, char **argv)
{
    /* Initialize StarPU with default configuration */
    int ret = starpu_init(NULL);

    if (ret == -ENODEV) goto enodev;

    /* Initialize path and name */
    char pid_str[16];
    int pid = getpid();
    snprintf(pid_str, 16, "%d", pid);

    const char *name_file_start = "STARPU_DISK_COMPUTE_DATA_";
    const char *name_file_end = "STARPU_DISK_COMPUTE_DATA_RESULT_";

    char * path_file_start = malloc(strlen(base) + 1 + strlen(name_file_start) + 1);
    strcpy(path_file_start, base);
    strcat(path_file_start, "/");
    strcat(path_file_start, name_file_start);

    char * path_file_end = malloc(strlen(base) + 1 + strlen(name_file_end) + 1);
    strcpy(path_file_end, base);
    strcat(path_file_end, "/");
    strcat(path_file_end, name_file_end);
}

```

```

/* register a disk */
int new_dd = starpu_disk_register(&starpu_disk_unistd_ops, (void *) base, 102
4*1024*1);
/* can't write on /tmp/ */
if (new_dd == -ENOENT) goto enoent;

unsigned dd = (unsigned) new_dd;

printf("TEST DISK MEMORY \n");

/* Imagine, you want to compute datas */
int *A;
int *C;

starpu_malloc_flags((void **)&A, NX*sizeof(int),
STARPU_MALLOC_COUNT);
starpu_malloc_flags((void **)&C, NX*sizeof(int),
STARPU_MALLOC_COUNT);

unsigned int j;
/* you register them in a vector */
for(j = 0; j < NX; ++j)
{
    A[j] = j;
    C[j] = 0;
}

/* you create a file to store the vector ON the disk */
FILE * f = fopen(path_file_start, "wb+");
if (f == NULL)
    goto enoent2;

/* store it in the file */
fwrite(A, sizeof(int), NX, f);

/* close the file */
fclose(f);

/* create a file to store result */
f = fopen(path_file_end, "wb+");
if (f == NULL)
    goto enoent2;

/* replace all datas by 0 */
fwrite(C, sizeof(int), NX, f);

/* close the file */
fclose(f);

/* And now, you want to use your datas in StarPU */
/* Open the file ON the disk */
void * data = starpu_disk_open(dd, (void *) name_file_start, NX*sizeof(int));
void * data_result = starpu_disk_open(dd, (void *) name_file_end, NX*sizeof(int));

starpu_data_handle_t vector_handleA, vector_handleC;

/* register vector in starpu */
starpu_vector_data_register(&vector_handleA, dd, (uintptr_t) data, NX,
sizeof(int));

/* and do what you want with it, here we copy it into an other vector */
starpu_vector_data_register(&vector_handleC, dd, (uintptr_t) data_result
, NX, sizeof(int));

starpu_data_copy(vector_handleC, vector_handleA, 0, NULL, NULL);

/* free them */
starpu_data_unregister(vector_handleA);
starpu_data_unregister(vector_handleC);

/* close them in StarPU */
starpu_disk_close(dd, data, NX*sizeof(int));
starpu_disk_close(dd, data_result, NX*sizeof(int));

/* check results */
f = fopen(path_file_end, "rb+");
if (f == NULL)
    goto enoent;
/* take datas */
int size = fread(C, sizeof(int), NX, f);

/* close the file */

```

```

fclose(f);

int try = 1;
for (j = 0; j < NX; ++j)
    if (A[j] != C[j])
    {
        printf("Fail A %d != C %d \n", A[j], C[j]);
        try = 0;
    }

    starpu_free_flags(A, NX*sizeof(int),
STARPU_MALLOC_COUNT);
    starpu_free_flags(C, NX*sizeof(int),
STARPU_MALLOC_COUNT);

    unlink(path_file_start);
    unlink(path_file_end);

    free(path_file_start);
    free(path_file_end);

    /* terminate StarPU, no task can be submitted after */
    starpu_shutdown();

    if(try)
        printf("TEST SUCCESS\n");
    else
        printf("TEST FAIL\n");
    return (try ? EXIT_SUCCESS : EXIT_FAILURE);

enodev:
    return 77;
enoent2:
    starpu_free_flags(A, NX*sizeof(int),
STARPU_MALLOC_COUNT);
    starpu_free_flags(C, NX*sizeof(int),
STARPU_MALLOC_COUNT);
enoent:
    unlink(path_file_start);
    unlink(path_file_end);

    free(path_file_start);
    free(path_file_end);

    starpu_shutdown();
    return 77;
}

```


Chapter 17

MPI Support

The integration of MPI transfers within task parallelism is done in a very natural way by the means of asynchronous interactions between the application and StarPU. This is implemented in a separate libstarpumpi library which basically provides "StarPU" equivalents of `MPI_*` functions, where `void *` buffers are replaced with `starpu_↔data_handle_t`, and all GPU-RAM-NIC transfers are handled efficiently by StarPU-MPI. The user has to use the usual `mpirun` command of the MPI implementation to start StarPU on the different MPI nodes.

An MPI Insert Task function provides an even more seamless transition to a distributed application, by automatically issuing all required data transfers according to the task graph and an application-provided distribution.

17.1 Simple Example

The flags required to compile or link against the MPI layer are accessible with the following commands:

```
$ pkg-config --cflags starpumpi-1.2 # options for the compiler
$ pkg-config --libs starpumpi-1.2   # options for the linker
```

You also need pass the option `-static` if the application is to be linked statically.

```
void increment_token(void)
{
    struct starpu_task *task = starpu_task_create();

    task->cl = &increment_cl;
    task->handles[0] = token_handle;

    starpu_task_submit(task);
}

int main(int argc, char **argv)
{
    int rank, size;

    starpu_init(NULL);
    starpu_mpi_initialize_extended(&rank, &size);

    starpu_vector_data_register(&token_handle,
        STARPU_MAIN_RAM, (uintptr_t)&token, 1, sizeof(unsigned));

    unsigned nloops = NITER;
    unsigned loop;

    unsigned last_loop = nloops - 1;
    unsigned last_rank = size - 1;

    for (loop = 0; loop < nloops; loop++) {
        int tag = loop*size + rank;

        if (loop == 0 && rank == 0)
        {
            token = 0;
            fprintf(stdout, "Start with token value %d\n", token);
        }
    }
}
```

```

else
{
    starpu_mpi_irecv_detached(token_handle, (rank+size-1)%size, tag,
        MPI_COMM_WORLD, NULL, NULL);
}

increment_token();

if (loop == last_loop && rank == last_rank)
{
    starpu_data_acquire(token_handle, STARPU_R);
    fprintf(stdout, "Finished: token value %d\n", token);
    starpu_data_release(token_handle);
}
else
{
    starpu_mpi_isend_detached(token_handle, (rank+1)%size, tag+1,
        MPI_COMM_WORLD, NULL, NULL);
}
}

starpu_task_wait_for_all();

starpu_mpi_shutdown();
starpu_shutdown();

if (rank == last_rank)
{
    fprintf(stderr, "[%d] token = %d == %d * %d ?\n", rank, token, nloops, size);
    STARPU_ASSERT(token == nloops*size);
}

```

17.2 Point To Point Communication

The standard point to point communications of MPI have been implemented. The semantic is similar to the MPI one, but adapted to the DSM provided by StarPU. A MPI request will only be submitted when the data is available in the main memory of the node submitting the request.

There is two types of asynchronous communications: the classic asynchronous communications and the detached communications. The classic asynchronous communications ([starpu_mpi_isend\(\)](#) and [starpu_mpi_irecv\(\)](#)) need to be followed by a call to [starpu_mpi_wait\(\)](#) or to [starpu_mpi_test\(\)](#) to wait for or to test the completion of the communication. Waiting for or testing the completion of detached communications is not possible, this is done internally by StarPU-MPI, on completion, the resources are automatically released. This mechanism is similar to the pthread detach state attribute which determines whether a thread will be created in a joinable or a detached state.

Internally, all communication are divided in 2 communications, a first message is used to exchange an envelope describing the data (i.e its tag and its size), the data itself is sent in a second message. All MPI communications submitted by StarPU uses a unique tag which has a default value, and can be accessed with the functions [starpu_mpi_get_communication_tag\(\)](#) and [starpu_mpi_set_communication_tag\(\)](#). The matching of tags with corresponding requests is done within StarPU-MPI.

For any userland communication, the call of the corresponding function (e.g [starpu_mpi_isend\(\)](#)) will result in the creation of a StarPU-MPI request, the function [starpu_data_acquire_cb\(\)](#) is then called to asynchronously request StarPU to fetch the data in main memory; when the data is ready and the corresponding buffer has already been received by MPI, it will be copied in the memory of the data, otherwise the request is stored in the *early requests list*. Sending requests are stored in the *ready requests list*.

While requests need to be processed, the StarPU-MPI progression thread does the following:

1. it polls the *ready requests list*. For all the ready requests, the appropriate function is called to post the corresponding MPI call. For example, an initial call to [starpu_mpi_isend\(\)](#) will result in a call to `MPI_Isend`. If the request is marked as detached, the request will then be added in the *detached requests list*.
2. it posts a `MPI_Irecv()` to retrieve a data envelope.
3. it polls the *detached requests list*. For all the detached requests, it tests its completion of the MPI request by calling `MPI_Test`. On completion, the data handle is released, and if a callback was defined, it is called.

4. finally, it checks if a data envelope has been received. If so, if the data envelope matches a request in the *early requests list* (i.e the request has already been posted by the application), the corresponding MPI call is posted (similarly to the first step above).

If the data envelope does not match any application request, a temporary handle is created to receive the data, a StarPU-MPI request is created and added into the *ready requests list*, and thus will be processed in the first step of the next loop.

[Communication](#) gives the list of all the point to point communications defined in StarPU-MPI.

17.3 Exchanging User Defined Data Interface

New data interfaces defined as explained in [Defining A New Data Interface](#) can also be used within StarPU-MPI and exchanged between nodes. Two functions needs to be defined through the type `starpu_data_interface_ops`. The function `starpu_data_interface_ops::pack_data` takes a handle and returns a contiguous memory buffer allocated with `starpu_malloc_flags(ptr, size, 0)` along with its size where data to be conveyed to another node should be copied. The reversed operation is implemented in the function `starpu_data_interface_ops::unpack_data` which takes a contiguous memory buffer and recreates the data handle.

```
static int complex_pack_data(starpu_data_handle_t handle, unsigned node, void **ptr,
                             ssize_t *count)
{
    STARPU_ASSERT(starpu_data_test_if_allocated_on_node(handle, node));

    struct starpu_complex_interface *complex_interface =
        (struct starpu_complex_interface *) starpu_data_get_interface_on_node(
            handle, node);

    *count = complex_get_size(handle);
    starpu_malloc_flags(ptr, *count, 0);
    memcpy(*ptr, complex_interface->real, complex_interface->nx*sizeof(double));
    memcpy(*ptr+complex_interface->nx*sizeof(double), complex_interface->imaginary,
        complex_interface->nx*sizeof(double));

    return 0;
}

static int complex_unpack_data(starpu_data_handle_t handle, unsigned node, void *ptr,
                               size_t count)
{
    STARPU_ASSERT(starpu_data_test_if_allocated_on_node(handle, node));

    struct starpu_complex_interface *complex_interface =
        (struct starpu_complex_interface *) starpu_data_get_interface_on_node(
            handle, node);

    memcpy(complex_interface->real, ptr, complex_interface->nx*sizeof(double));
    memcpy(complex_interface->imaginary, ptr+complex_interface->nx*sizeof(double),
        complex_interface->nx*sizeof(double));

    return 0;
}

static struct starpu_data_interface_ops interface_complex_ops =
{
    ...
    .pack_data = complex_pack_data,
    .unpack_data = complex_unpack_data
};
```

17.4 MPI Insert Task Utility

To save the programmer from having to explicit all communications, StarPU provides an "MPI Insert Task Utility". The principle is that the application decides a distribution of the data over the MPI nodes by allocating it and notifying StarPU of that decision, i.e. tell StarPU which MPI node "owns" which data. It also decides, for each handle, an MPI tag which will be used to exchange the content of the handle. All MPI nodes then process the whole task graph, and StarPU automatically determines which node actually execute which task, and trigger the required MPI transfers.

The list of functions is described in [MPI Insert Task](#).

Here an stencil example showing how to use `starpu_mpi_task_insert()`. One first needs to define a distribution function which specifies the locality of the data. Note that the data needs to be registered to MPI by calling `starpu_mpi_data_register()`. This function allows to set the distribution information and the MPI tag which should be used when communicating the data. The function `starpu_mpi_data_register()` should be preferred to `starpu_data_set_rank()` and `starpu_data_set_tag()` as it also allows to automatically clear the MPI communication cache when unregistering the data.

```
/* Returns the MPI node number where data is */
int my_distrib(int x, int y, int nb_nodes) {
    /* Block distrib */
    return ((int)(x / sqrt(nb_nodes) + (y / sqrt(nb_nodes)) * sqrt(nb_nodes))) % nb_nodes;

    // /* Other examples useful for other kinds of computations */
    // /* / distrib */
    // return (x+y) % nb_nodes;

    // /* Block cyclic distrib */
    // unsigned side = sqrt(nb_nodes);
    // return x % side + (y % side) * side;
}
```

Now the data can be registered within StarPU. Data which are not owned but will be needed for computations can be registered through the lazy allocation mechanism, i.e. with a `home_node` set to `-1`. StarPU will automatically allocate the memory when it is used for the first time.

One can note an optimization here (the `else if` test): we only register data which will be needed by the tasks that we will execute.

```
unsigned matrix[X][Y];
starpu_data_handle_t data_handles[X][Y];

for(x = 0; x < X; x++) {
    for (y = 0; y < Y; y++) {
        int mpi_rank = my_distrib(x, y, size);
        if (mpi_rank == my_rank)
            /* Owning data */
            starpu_variable_data_register(&data_handles[x][y],
            STARPU_MAIN_RAM,
            (uintptr_t)&(matrix[x][y]), sizeof(unsigned));
        else if (my_rank == my_distrib(x+1, y, size) || my_rank == my_distrib(x-1, y, size)
            || my_rank == my_distrib(x, y+1, size) || my_rank == my_distrib(x, y-1, size))
            /* I don't own that index, but will need it for my computations */
            starpu_variable_data_register(&data_handles[x][y], -1,
            (uintptr_t)NULL, sizeof(unsigned));
        else
            /* I know it's useless to allocate anything for this */
            data_handles[x][y] = NULL;
        if (data_handles[x][y]) {
            starpu_mpi_data_register(data_handles[x][y], x*X+y, mpi_rank);
        }
    }
}
```

Now `starpu_mpi_task_insert()` can be called for the different steps of the application.

```
for(loop=0; loop<niter; loop++)
    for (x = 1; x < X-1; x++)
        for (y = 1; y < Y-1; y++)
            starpu_mpi_task_insert(MPI_COMM_WORLD, &stencil5_c1,
            STARPU_RW, data_handles[x][y],
            STARPU_R, data_handles[x-1][y],
            STARPU_R, data_handles[x+1][y],
            STARPU_R, data_handles[x][y-1],
            STARPU_R, data_handles[x][y+1],
            0);

starpu_task_wait_for_all();
```

I.e. all MPI nodes process the whole task graph, but as mentioned above, for each task, only the MPI node which owns the data being written to (here, `data_handles[x][y]`) will actually run the task. The other MPI nodes will automatically send the required data.

This can be a concern with a growing number of nodes. To avoid this, the application can prune the task for loops according to the data distribution, so as to only submit tasks on nodes which have to care about them (either to execute them, or to send the required data).

A way to do some of this quite easily can be to just add an `if` like this:

```

for(loop=0 ; loop<niter; loop++)
    for (x = 1; x < X-1; x++)
        for (y = 1; y < Y-1; y++)
            if (my_distrib(x,y,size) == my_rank
                || my_distrib(x-1,y,size) == my_rank
                || my_distrib(x+1,y,size) == my_rank
                || my_distrib(x,y-1,size) == my_rank
                || my_distrib(x,y+1,size) == my_rank)
                starpu_mpi_insert_task(MPI_COMM_WORLD, &stencil5_c1,
                                      STARPU_RW, data_handles[x][y],
                                      STARPU_R, data_handles[x-1][y],
                                      STARPU_R, data_handles[x+1][y],
                                      STARPU_R, data_handles[x][y-1],
                                      STARPU_R, data_handles[x][y+1],
                                      0);

starpu_task_wait_for_all();

```

This permits to drop the cost of function call argument passing and parsing.

If the `my_distrib` function can be inlined by the compiler, the latter can improve the test.

If the `size` can be made a compile-time constant, the compiler can considerably improve the test further.

If the distribution function is not too complex and the compiler is very good, the latter can even optimize the `for` loops, thus dramatically reducing the cost of task submission.

A function `starpu_mpi_task_build()` is also provided with the aim to only construct the task structure. All MPI nodes need to call the function, only the node which is to execute the task will return a valid task structure. Following the execution of the task, all nodes need to call the function `starpu_mpi_task_post_build()` – with the same list of arguments as `starpu_mpi_task_build()` – to post all the necessary data communications.

```

struct starpu_task *task;
task = starpu_mpi_task_build(MPI_COMM_WORLD, &c1,
                             STARPU_RW, data_handles[0],
                             STARPU_R, data_handles[1],
                             0);
if (task) starpu_task_submit(task);
starpu_mpi_task_post_build(MPI_COMM_WORLD, &c1,
                           STARPU_RW, data_handles[0],
                           STARPU_R, data_handles[1],
                           0);

```

17.5 MPI cache support

StarPU-MPI automatically optimizes duplicate data transmissions: if an MPI node B needs a piece of data D from MPI node A for several tasks, only one transmission of D will take place from A to B, and the value of D will be kept on B as long as no task modifies D.

If a task modifies D, B will wait for all tasks which need the previous value of D, before invalidating the value of D. As a consequence, it releases the memory occupied by D. Whenever a task running on B needs the new value of D, allocation will take place again to receive it.

Since tasks can be submitted dynamically, StarPU-MPI can not know whether the current value of data D will again be used by a newly-submitted task before being modified by another newly-submitted task, so until a task is submitted to modify the current value, it can not decide by itself whether to flush the cache or not. The application can however explicitly tell StarPU-MPI to flush the cache by calling `starpu_mpi_cache_flush()` or `starpu_mpi_cache_flush_all_data()`, for instance in case the data will not be used at all any more (see for instance the cholesky example in `mpi/examples/matrix_decomposition`), or at least not in the close future. If a newly-submitted task actually needs the value again, another transmission of D will be initiated from A to B. A mere `starpu_mpi_cache_flush_all_data()` can for instance be added at the end of the whole algorithm, to express that no data will be reused after that (or at least that it is not interesting to keep them in cache). It may however be interesting to add fine-grained `starpu_mpi_cache_flush()` calls during the algorithm; the effect for the data deallocation will be the same, but it will additionally release some pressure from the StarPU-MPI cache hash table during task submission.

The whole caching behavior can be disabled thanks to the `STARPU_MPI_CACHE` environment variable. The variable `STARPU_MPI_CACHE_STATS` can be set to 1 to enable the runtime to display messages when data are added or removed from the cache holding the received data.

17.6 MPI Data migration

The application can dynamically change its mind about the data distribution, to balance the load over MPI nodes for instance. This can be done very simply by requesting an explicit move and then change the registered rank. For instance, we here switch to a new distribution function `my_distrib2`: we first register any data that wasn't registered already and will be needed, then migrate the data, and register the new location.

```
for(x = 0; x < X; x++) {
    for (y = 0; y < Y; y++) {
        int mpi_rank = my_distrib2(x, y, size);
        if (!data_handles[x][y] && (mpi_rank == my_rank
            || my_rank == my_distrib(x+1, y, size) || my_rank == my_distrib(x-1, y, size)
            || my_rank == my_distrib(x, y+1, size) || my_rank == my_distrib(x, y-1, size)))
            /* Register newly-needed data */
            starpu_variable_data_register(&data_handles[x][y], -1,
                (uintptr_t) NULL, sizeof(unsigned));

        if (data_handles[x][y]) {
            /* Migrate the data */
            starpu_mpi_get_data_on_node_detached(MPI_COMM_WORLD,
                data_handles[x][y], mpi_rank, NULL, NULL);
            /* And register the new rank of the matrix */
            starpu_data_set_rank(data_handles[x][y], mpi_rank);
        }
    }
}
```

From then on, further tasks submissions will use the new data distribution, which will thus change both MPI communications and task assignments.

Very importantly, since all nodes have to agree on which node owns which data so as to determine MPI communications and task assignments the same way, all nodes have to perform the same data migration, and at the same point among task submissions. It thus does not require a strict synchronization, just a clear separation of task submissions before and after the data redistribution.

Before data unregistration, it has to be migrated back to its original home node (the value, at least), since that is where the user-provided buffer resides. Otherwise the unregistration will complain that it does not have the latest value on the original home node.

```
for(x = 0; x < X; x++) {
    for (y = 0; y < Y; y++) {
        if (data_handles[x][y]) {
            int mpi_rank = my_distrib(x, y, size);
            /* Get back data to original place where the user-provided buffer is. */
            starpu_mpi_get_data_on_node_detached(MPI_COMM_WORLD,
                data_handles[x][y], mpi_rank, NULL, NULL);
            /* And unregister it */
            starpu_data_unregister(data_handles[x][y]);
        }
    }
}
```

17.7 MPI Collective Operations

The functions are described in [MPI Collective Operations](#).

```
if (rank == root)
{
    /* Allocate the vector */
    vector = malloc(nblocks * sizeof(float *));
    for(x=0 ; x<nblocks ; x++)
    {
        starpu_malloc((void **)&vector[x], block_size*sizeof(float));
    }
}

/* Allocate data handles and register data to StarPU */
data_handles = malloc(nblocks*sizeof(starpu_data_handle_t *));
for(x = 0; x < nblocks ; x++)
{
    int mpi_rank = my_distrib(x, nodes);
    if (rank == root) {
        starpu_vector_data_register(&data_handles[x],
```

```
        STARPU_MAIN_RAM, (uintptr_t)vector[x],
                           blocks_size, sizeof(float));
    }
    else if ((mpi_rank == rank) || ((rank == mpi_rank+1 || rank == mpi_rank-1))) {
        /* I own that index, or i will need it for my computations */
        starpu_vector_data_register(&data_handles[x], -1, (uintptr_t)NULL,
                                   block_size, sizeof(float));
    }
    else {
        /* I know it's useless to allocate anything for this */
        data_handles[x] = NULL;
    }
    if (data_handles[x]) {
        starpu_mpi_data_register(data_handles[x], x*nblocks+y, mpi_rank);
    }
}

/* Scatter the matrix among the nodes */
starpu_mpi_scatter_detached(data_handles, nblocks, root, MPI_COMM_WORLD);

/* Calculation */
for(x = 0; x < nblocks ; x++) {
    if (data_handles[x]) {
        int owner = starpu_data_get_rank(data_handles[x]);
        if (owner == rank) {
            starpu_task_insert(&c1, STARPU_RW, data_handles[x], 0);
        }
    }
}

/* Gather the matrix on main node */
starpu_mpi_gather_detached(data_handles, nblocks, 0, MPI_COMM_WORLD);
```


Chapter 18

FFT Support

StarPU provides `libstarpuffft`, a library whose design is very similar to both `fftw` and `cufft`, the difference being that it takes benefit from both CPUs and GPUs. It should however be noted that GPUs do not have the same precision as CPUs, so the results may differ by a negligible amount.

Different precisions are available, namely float, double and long double precisions, with the following `fftw` naming conventions:

- double precision structures and functions are named e.g. `starpufft_execute()`
- float precision structures and functions are named e.g. `starpufftf_execute()`
- long double precision structures and functions are named e.g. `starpufftl_execute()`

The documentation below is given with names for double precision, replace `starpufft_` with `starpufftf_` or `starpufftl_` as appropriate.

Only complex numbers are supported at the moment.

The application has to call `starpufft_init()` before calling `starpufft` functions.

Either main memory pointers or data handles can be provided.

- To provide main memory pointers, use `starpufft_start()` or `starpufft_execute()`. Only one FFT can be performed at a time, because StarPU will have to register the data on the fly. In the `starpufft_start()` case, `starpufft_cleanup()` needs to be called to unregister the data.
- To provide data handles (which is preferable), use `starpufft_start_handle()` (preferred) or `starpufft_execute_handle()`. Several FFTs tasks can be submitted for a given plan, which permits e.g. to start a series of FFT with just one plan. `starpufft_start_handle()` is preferable since it does not wait for the task completion, and thus permits to enqueue a series of tasks.

All functions are defined in [FFT Support](#).

18.1 Compilation

The flags required to compile or link against the FFT library are accessible with the following commands:

```
$ pkg-config --cflags starpufft-1.2 # options for the compiler
$ pkg-config --libs starpufft-1.2  # options for the linker
```

Also pass the option `-static` if the application is to be linked statically.

Chapter 19

MIC/SCC Support

19.1 Compilation

SCC support just needs the presence of the RCCE library.

MIC Xeon Phi support actually needs two compilations of StarPU, one for the host and one for the device. The `PATH` environment variable has to include the path to the cross-compilation toolchain, for instance `/usr/linux-k10m-4.7/bin`. The script `mic-configure` can then be used to achieve the two compilations: it basically calls `configure` as appropriate from two new directories: `build_mic` and `build_host`. `make` and `make install` can then be used as usual and will recurse into both directories.

19.2 Porting Applications To MIC Xeon Phi / SCC

The simplest way to port an application to MIC Xeon Phi or SCC is to set the field `starpu_codelet::cpu_funcs_name`, to provide StarPU with the function name of the CPU implementation. StarPU will thus simply use the existing CPU implementation (cross-rebuilt in the MIC Xeon Phi case). The functions have to be globally-visible (i.e. not `static`) for StarPU to be able to look them up.

For SCC execution, the function `starpu_initialize()` also has to be used instead of `starpu_init()`, so as to pass `argc` and `argv`.

19.3 Launching Programs

SCC programs are started through RCCE.

MIC programs are started from the host. StarPU automatically starts the same program on MIC devices. It however needs to get the MIC-cross-built binary. It will look for the file given by the environment variable `STARPU_MIC_SINK_PROGRAM_NAME` or in the directory given by the environment variable `STARPU_MIC_SINK_PROGRAM_PATH`, or in the field `starpu_conf::mic_sink_program_path`. It will also look in the current directory for the same binary name plus the suffix `-mic` or `_mic`.

The testsuite can be started by simply running `make check` from the top directory. It will recurse into both `build_host` to run tests with only the host, and into `build_mic` to run tests with both the host and the MIC devices. Single tests with the host and the MIC can be run by starting `./loader-cross.sh ./the_test` from `build_mic/tests`.

Chapter 20

C Extensions

When GCC plug-in support is available, StarPU builds a plug-in for the GNU Compiler Collection (GCC), which defines extensions to languages of the C family (C, C++, Objective-C) that make it easier to write StarPU code. This feature is only available for GCC 4.5 and later; it is known to work with GCC 4.5, 4.6, and 4.7. You may need to install a specific `-dev` package of your distro, such as `gcc-4.6-plugin-dev` on Debian and derivatives. In addition, the plug-in's test suite is only run when GNU Guile (<http://www.gnu.org/software/guile/>) is found at `configure-time`. Building the GCC plug-in can be disabled by configuring with `--disable-gcc-extensions`.

Those extensions include syntactic sugar for defining tasks and their implementations, invoking a task, and manipulating data buffers. Use of these extensions can be made conditional on the availability of the plug-in, leading to valid C sequential code when the plug-in is not used ([Using C Extensions Conditionally](#)).

When StarPU has been installed with its GCC plug-in, programs that use these extensions can be compiled this way:

```
$ gcc -c -fplugin='pkg-config starpu-1.2 --variable=gccplugin' foo.c
```

When the plug-in is not available, the above `pkg-config` command returns the empty string.

In addition, the `-fplugin-arg-starpu-verbose` flag can be used to obtain feedback from the compiler as it analyzes the C extensions used in source files.

This section describes the C extensions implemented by StarPU's GCC plug-in. It does not require detailed knowledge of the StarPU library.

Note: this is still an area under development and subject to change.

20.1 Defining Tasks

The StarPU GCC plug-in views tasks as “extended” C functions:

- tasks may have several implementations—e.g., one for CPUs, one written in OpenCL, one written in CUDA;
- tasks may have several implementations of the same target—e.g., several CPU implementations;
- when a task is invoked, it may run in parallel, and StarPU is free to choose any of its implementations.

Tasks and their implementations must be *declared*. These declarations are annotated with attributes (<http://gcc.gnu.org/onlinedocs/gcc/Attribute-Syntax.html#Attribute-Syntax>): the declaration of a task is a regular C function declaration with an additional `task` attribute, and task implementations are declared with a `task_implementation` attribute.

The following function attributes are provided:

task Declare the given function as a StarPU task. Its return type must be `void`. When a function declared as `task` has a user-defined body, that body is interpreted as the implicit definition of the task's CPU implementation (see example below). In all cases, the actual definition of a task's body is automatically generated by the compiler.

Under the hood, declaring a task leads to the declaration of the corresponding `codelet` ([Codelet and Tasks](#)). If one or more task implementations are declared in the same compilation unit, then the `codelet` and the function itself are also defined; they inherit the scope of the task.

Scalar arguments to the task are passed by value and copied to the target device if need be—technically, they are passed as the buffer `starpu_task::cl_arg` ([Codelet and Tasks](#)).

Pointer arguments are assumed to be registered data buffers—the handles argument of a task (`starpu_task::handles`) ; `const`-qualified pointer arguments are viewed as read-only buffers ([STARPU_R](#)), and non-`const`-qualified buffers are assumed to be used read-write ([STARPU_RW](#)). In addition, the `output` type attribute can be as a type qualifier for output pointer or array parameters ([STARPU_W](#)).

task_implementation (target, task) Declare the given function as an implementation of `task` to run on `target`. `target` must be a string, currently one of "cpu", "opencl", or "cuda".

Here is an example:

```
#define __output __attribute__((output))

static void matmul (const float *A, const float *B,
                  __output float *C,
                  unsigned nx, unsigned ny, unsigned nz)
    __attribute__((task));

static void matmul_cpu (const float *A, const float *B,
                      __output float *C,
                      unsigned nx, unsigned ny, unsigned nz)
    __attribute__((task_implementation ("cpu", matmul)));

static void
matmul_cpu (const float *A, const float *B, __output float *C,
            unsigned nx, unsigned ny, unsigned nz)
{
    unsigned i, j, k;

    for (j = 0; j < ny; j++)
        for (i = 0; i < nx; i++)
        {
            for (k = 0; k < nz; k++)
                C[j * nx + i] += A[j * nz + k] * B[k * nx + i];
        }
}
```

A `matmult` task is defined; it has only one implementation, `matmult_cpu`, which runs on the CPU. Variables `A` and `B` are input buffers, whereas `C` is considered an input/output buffer.

For convenience, when a function declared with the `task` attribute has a user-defined body, that body is assumed to be that of the CPU implementation of a task, which we call an implicit task CPU implementation. Thus, the above snippet can be simplified like this:

```
#define __output __attribute__((output))

static void matmul (const float *A, const float *B,
                  __output float *C,
                  unsigned nx, unsigned ny, unsigned nz)
    __attribute__((task));

/* Implicit definition of the CPU implementation of the
   'matmul' task. */
static void
matmul (const float *A, const float *B, __output float *C,
        unsigned nx, unsigned ny, unsigned nz)
{
    unsigned i, j, k;

    for (j = 0; j < ny; j++)
        for (i = 0; i < nx; i++)
        {
            for (k = 0; k < nz; k++)
                C[j * nx + i] += A[j * nz + k] * B[k * nx + i];
        }
}
```

Use of implicit CPU task implementations as above has the advantage that the code is valid sequential code when StarPU's GCC plug-in is not used ([Using C Extensions Conditionally](#)).

CUDA and OpenCL implementations can be declared in a similar way:

```
static void matmul_cuda (const float *A, const float *B, float *C,
                        unsigned nx, unsigned ny, unsigned nz)
    __attribute__((task_implementation ("cuda", matmul)));

static void matmul_opengl (const float *A, const float *B, float *C,
                           unsigned nx, unsigned ny, unsigned nz)
    __attribute__((task_implementation ("opengl", matmul)));
```

The CUDA and OpenCL implementations typically either invoke a kernel written in CUDA or OpenCL (for similar code, [CUDA Kernel](#), and [OpenCL Kernel](#)), or call a library function that uses CUDA or OpenCL under the hood, such as CUBLAS functions:

```
static void
matmul_cuda (const float *A, const float *B, float *C,
             unsigned nx, unsigned ny, unsigned nz)
{
    cublasSgemm ('n', 'n', nx, ny, nz,
                1.0f, A, 0, B, 0,
                0.0f, C, 0);
    cudaStreamSynchronize (starpu_cuda_get_local_stream ());
}
```

A task can be invoked like a regular C function:

```
matmul (&A[i * zdim * bydim + k * bzdime * bydim],
        &B[k * xdim * bzdime + j * bxdime * bzdime],
        &C[i * xdim * bydim + j * bxdime * bydim],
        bxdime, bydim, bzdime);
```

This leads to an asynchronous invocation, whereby `matmul`'s implementation may run in parallel with the continuation of the caller.

The next section describes how memory buffers must be handled in StarPU-GCC code. For a complete example, see the `gcc-plugin/examples` directory of the source distribution, and [Vector Scaling Using the C Extension](#).

20.2 Initialization, Termination, and Synchronization

The following pragmas allow user code to control StarPU's life time and to synchronize with tasks.

#pragma starpu initialize Initialize StarPU. This call is compulsory and is *never* added implicitly. One of the reasons this has to be done explicitly is that it provides greater control to user code over its resource usage.

#pragma starpu shutdown Shut down StarPU, giving it an opportunity to write profiling info to a file on disk, for instance ([Off-line Performance Feedback](#)).

#pragma starpu wait Wait for all task invocations to complete, as with [starpu_task_wait_for_all\(\)](#).

20.3 Registered Data Buffers

Data buffers such as matrices and vectors that are to be passed to tasks must be registered. Registration allows StarPU to handle data transfers among devices—e.g., transferring an input buffer from the CPU's main memory to a task scheduled to run a GPU ([StarPU Data Management Library](#)).

The following pragmas are provided:

#pragma starpu register ptr [size] Register `ptr` as a `size`-element buffer. When `ptr` has an array type whose size is known, `size` may be omitted. Alternatively, the `registered` attribute can be used (see below.)

#pragma starpu unregister ptr Unregister the previously-registered memory area pointed to by `ptr`. As a side-effect, `ptr` points to a valid copy in main memory.

#pragma starpu acquire ptr Acquire in main memory an up-to-date copy of the previously-registered memory area pointed to by `ptr`, for read-write access.

#pragma starpu release ptr Release the previously-registered memory area pointed to by `ptr`, making it available to the tasks.

Additionally, the following attributes offer a simple way to allocate and register storage for arrays:

registered This attribute applies to local variables with an array type. Its effect is to automatically register the array's storage, as per `#pragma starpu register`. The array is automatically unregistered when the variable's scope is left. This attribute is typically used in conjunction with the `heap_allocated` attribute, described below.

heap_allocated This attribute applies to local variables with an array type. Its effect is to automatically allocate the array's storage on the heap, using `starpu_malloc()` under the hood. The heap-allocated array is automatically freed when the variable's scope is left, as with automatic variables.

The following example illustrates use of the `heap_allocated` attribute:

```
extern void cholesky(unsigned nblocks, unsigned size,
                    float mat[nblocks][nblocks][size])
    __attribute__((task));

int
main (int argc, char *argv[])
{
    #pragma starpu initialize

    /* ... */

    int nblocks, size;
    parse_args (&nblocks, &size);

    /* Allocate an array of the required size on the heap,
       and register it. */

    {
        float matrix[nblocks][nblocks][size]
            __attribute__((heap_allocated, registered));

        cholesky (nblocks, size, matrix);
    }

    #pragma starpu wait

    /* MATRIX is automatically unregistered & freed here. */

    #pragma starpu shutdown

    return EXIT_SUCCESS;
}
```

20.4 Using C Extensions Conditionally

The C extensions described in this chapter are only available when GCC and its StarPU plug-in are in use. Yet, it is possible to make use of these extensions when they are available—leading to hybrid CPU/GPU code—and discard them when they are not available—leading to valid sequential code.

To that end, the GCC plug-in defines the C preprocessor macro — `STARPU_GCC_PLUGIN` — when it is being used. When defined, this macro expands to an integer denoting the version of the supported C extensions.

The code below illustrates how to define a task and its implementations in a way that allows it to be compiled without the GCC plug-in:

```
/* This program is valid, whether or not StarPU's GCC plug-in
   is being used. */
```



```

#include <stdlib.h>

/* The attribute below is ignored when GCC is not used. */
static void matmul (const float *A, const float *B, float * C,
                   unsigned nx, unsigned ny, unsigned nz)
    __attribute__ ((task));

static void
matmul (const float *A, const float *B, float * C,
        unsigned nx, unsigned ny, unsigned nz)
{
    /* Code of the CPU kernel here... */
}

#ifdef STARPU_GCC_PLUGIN
/* Optional OpenCL task implementation. */

static void matmul_opencl (const float *A, const float *B, float * C,
                           unsigned nx, unsigned ny, unsigned nz)
    __attribute__ ((task_implementation ("opencl", matmul)));

static void
matmul_opencl (const float *A, const float *B, float * C,
               unsigned nx, unsigned ny, unsigned nz)
{
    /* Code that invokes the OpenCL kernel here... */
}
#endif

int
main (int argc, char *argv[])
{
    /* The pragmas below are simply ignored when StarPU-GCC
       is not used. */
    #pragma starpu initialize

    float A[123][42][7], B[123][42][7], C[123][42][7];

    #pragma starpu register A
    #pragma starpu register B
    #pragma starpu register C

    /* When StarPU-GCC is used, the call below is asynchronous;
       otherwise, it is synchronous. */
    matmul ((float *) A, (float *) B, (float *) C, 123, 42, 7);

    #pragma starpu wait
    #pragma starpu shutdown

    return EXIT_SUCCESS;
}

```

The above program is a valid StarPU program when StarPU's GCC plug-in is used; it is also a valid sequential program when the plug-in is not used.

Note that attributes such as `task` as well as `starpu` pragmas are simply ignored by GCC when the StarPU-GCC plug-in is not loaded. However, `gcc -Wall` emits a warning for unknown attributes and pragmas, which can be inconvenient. In addition, other compilers may be unable to parse the attribute syntax (In practice, Clang and several proprietary compilers implement attributes.), so you may want to wrap attributes in macros like this:

```

/* Use the 'task' attribute only when StarPU's GCC plug-in
   is available. */
#ifdef STARPU_GCC_PLUGIN
# define __task __attribute__ ((task))
#else
# define __task
#endif

static void matmul (const float *A, const float *B, float *C,
                   unsigned nx, unsigned ny, unsigned nz) __task;

```


Chapter 21

SOCL OpenCL Extensions

SOCL is an OpenCL implementation based on StarPU. It gives a unified access to every available OpenCL device↔: applications can now share entities such as Events, Contexts or Command Queues between several OpenCL implementations.

In addition, command queues that are created without specifying a device provide automatic scheduling of the submitted commands on OpenCL devices contained in the context to which the command queue is attached.

Note: this is still an area under development and subject to change.

When compiling StarPU, SOCL will be enabled if a valid OpenCL implementation is found on your system. To be able to run the SOCL test suite, the environment variable `SOCL_OCL_LIB_OPENCL` needs to be defined to the location of the file `libOpenCL.so` of the OCL ICD implementation. You should for example add the following line in your file `.bashrc`

```
export SOCL_OCL_LIB_OPENCL=/usr/lib/x86_64-linux-gnu/libOpenCL.so
```

You can then run the test suite in the directory `socl/examples`.

```
$ make check
...
PASS: basic/basic
PASS: testmap/testmap
PASS: clinfo/clinfo
PASS: matmul/matmul
PASS: mansched/mansched
=====
All 5 tests passed
=====
```

The environment variable `OCL_ICD_VENDORS` has to point to the directory where the ICD files are installed. When compiling StarPU, the files are in the directory `socl/vendors`. With an installed version of StarPU, the files are installed in the directory `$prefix/share/starpu/opencl/vendors`.

To run the tests by hand, you have to call for example,

```
$ LD_PRELOAD=$SOCL_OCL_LIB_OPENCL OCL_ICD_VENDORS=socl/vendors/ socl/examples/clinfo/clinfo
Number of platforms: 2
  Platform Profile: FULL_PROFILE
  Platform Version: OpenCL 1.1 CUDA 4.2.1
  Platform Name: NVIDIA CUDA
  Platform Vendor: NVIDIA Corporation
  Platform Extensions: cl_khr_byte_addressable_store cl_khr_icd cl_khr_gl_sharing cl_nv_compiler_options cl_nv_

  Platform Profile: FULL_PROFILE
  Platform Version: OpenCL 1.0 SOCL Edition (0.1.0)
  Platform Name: SOCL Platform
  Platform Vendor: INRIA
  Platform Extensions: cl_khr_icd
....
$
```

To enable the use of CPU cores via OpenCL, one can set the `STARPU_OPENCL_ON_CPUS` environment variable to 1 and `STARPU_NCPUS` to 0 (to avoid using CPUs both via the OpenCL driver and the normal CPU driver).

Chapter 22

SimGrid Support

StarPU can use Simgrid in order to simulate execution on an arbitrary platform.

22.1 Preparing your application for simulation.

There are a few technical details which need to be handled for an application to be simulated through Simgrid.

If the application uses `gettimeofday` to make its performance measurements, the real time will be used, which will be bogus. To get the simulated time, it has to use `starpu_timing_now()` which returns the virtual timestamp in us.

For some technical reason, the application's .c file which contains `main()` has to be recompiled with `starpu_simgrid↔_wrap.h`, which in the simgrid case will `# define main()` into `starpu_main()`, and it is `libstarpu` which will provide the real `main()` and will call the application's `main()`.

To be able to test with crazy data sizes, one may want to only allocate application data if `STARPU_SIMGRID` is not defined. Passing a `NULL` pointer to `starpu_data_register` functions is fine, data will never be read/written to by StarPU in Simgrid mode anyway.

To be able to run the application with e.g. CUDA simulation on a system which does not have CUDA installed, one can fill the `cuda_funcs` with `(void*)1`, to express that there is a CUDA implementation, even if one does not actually provide it. StarPU will never actually run it in Simgrid mode anyway.

22.2 Calibration

The idea is to first compile StarPU normally, and run the application, so as to automatically benchmark the bus and the codelets.

```
$ ./configure && make
$ STARPU_SCHED=dmda ./examples/matvecmult/matvecmult
[starpu][_starpu_load_history_based_model] Warning: model matvecmult
  is not calibrated, forcing calibration for this run. Use the
  STARPU_CALIBRATE environment variable to control this.
$ ...
$ STARPU_SCHED=dmda ./examples/matvecmult/matvecmult
TEST PASSED
```

Note that we force to use the scheduler `dmda` to generate performance models for the application. The application may need to be run several times before the model is calibrated.

22.3 Simulation

Then, recompile StarPU, passing `--enable-simgrid` to `./configure`.

```
$ ./configure --enable-simgrid
```

To specify the location of SimGrid, you can either set the environment variables `SIMGRID_CFLAGS` and `SIMGRID_LIBS`, or use the configure options `--with-simgrid-dir`, `--with-simgrid-include-dir` and `--with-simgrid-lib-dir`, for example

```
$ ./configure --with-simgrid-dir=/opt/local/simgrid
```

You can then re-run the application.

```
$ make
$ STARPU_SCHED=dmda ./examples/matvecmult/matvecmult
TEST FAILED !!!
```

It is normal that the test fails: since the computation are not actually done (that is the whole point of simgrid), the result is wrong, of course.

If the performance model is not calibrated enough, the following error message will be displayed

```
$ STARPU_SCHED=dmda ./examples/matvecmult/matvecmult
[starpu][_starpu_load_history_based_model] Warning: model matvecmult
is not calibrated, forcing calibration for this run. Use the
STARPU_CALIBRATE environment variable to control this.
[starpu][_starpu_simgrid_execute_job][assert failure] Codelet
matvecmult does not have a perfmodel, or is not calibrated enough
```

The number of devices can be chosen as usual with `STARPU_NCPU`, `STARPU_NCUDA`, and `STARPU_NOPE_NCL`, and the amount of GPU memory with `STARPU_LIMIT_CUDA_MEM`, `STARPU_LIMIT_CUDA_devid_MEM`, `STARPU_LIMIT_OPENCL_MEM`, and `STARPU_LIMIT_OPENCL_devid_MEM`.

22.4 Simulation On Another Machine

The simgrid support even permits to perform simulations on another machine, your desktop, typically. To achieve this, one still needs to perform the Calibration step on the actual machine to be simulated, then copy them to your desktop machine (the `$STARPU_HOME/.starpu` directory). One can then perform the Simulation step on the desktop machine, by setting the environment variable `STARPU_HOSTNAME` to the name of the actual machine, to make StarPU use the performance models of the simulated machine even on the desktop machine.

If the desktop machine does not have CUDA or OpenCL, StarPU is still able to use simgrid to simulate execution with CUDA/OpenCL devices, but the application source code will probably disable the CUDA and OpenCL codelets in that case. Since during simgrid execution, the functions of the codelet are actually not called, one can use dummy functions such as the following to still permit CUDA or OpenCL execution:

22.5 applications

By default, simgrid uses its own implementation of threads, which prevents gdb from being able to inspect stacks of all threads. To be able to fully debug an application running with simgrid, pass the `-cfg=contexts/factory:thread` option to the application, to make simgrid use system threads, which gdb will be able to manipulate as usual.

```
static struct starpu_codelet c111 =
{
    .cpu_funcs = {chol_cpu_codelet_update_u11},
```

```
        .cpu_funcs_name = {"chol_cpu_codelet_update_u11"},
#ifdef STARPU_USE_CUDA
        .cuda_funcs = {chol_cublas_codelet_update_u11},
#elif defined(STARPU_SIMGRID)
        .cuda_funcs = {(void*)1},
#endif
        .nbuffers = 1,
        .modes = {STARPU_RW},
        .model = &chol_model_11
};
```


Chapter 23

OpenMP Runtime Support

StarPU provides the necessary routines and support to implement an OpenMP (<http://www.openmp.org/>) runtime compliant with the revision 3.1 of the language specification, and compliant with the task-related data dependency functionalities introduced in the revision 4.0 of the language. This StarPU OpenMP Runtime Support (SORS) has been designed to be targetted by OpenMP compilers such as the Klang-OMP compiler. Most supported OpenMP directives can both be implemented inline or as outlined functions.

All functions are defined in [OpenMP Runtime Support](#).

23.1 Implementation Details and Specificities

23.1.1 Main Thread

When using the SORS, the main thread gets involved in executing OpenMP tasks just like every other threads, in order to be compliant with the specification execution model. This contrasts with StarPU's usual execution model where the main thread submit tasks but does not take part in executing them.

23.1.2 Extended Task Semantics

The semantics of tasks generated by the SORS are extended with respect to regular StarPU tasks in that SORS' tasks may block and be preempted by SORS call, whereas regular StarPU tasks cannot. SORS tasks may coexist with regular StarPU tasks. However, only the tasks created using SORS API functions inherit from extended semantics.

23.2 Configuration

The SORS can be compiled into `libstarpu` through the configure option `--enable-openmp`. Conditional compiled source codes may check for the availability of the OpenMP Runtime Support by testing whether the C preprocessor macro `STARPU_OPENMP` is defined or not.

23.3 Initialization and Shutdown

The SORS needs to be executed/terminated by the `starpu_omp_init()` / `starpu_omp_shutdown()` instead of `starpu_init()` / `starpu_shutdown()`. This requirement is necessary to make sure that the main thread gets the proper execution environment to run OpenMP tasks. These calls will usually be performed by a compiler runtime. Thus, they can be executed from a constructor/destructor such as this:

```
__attribute__((constructor))
```

```
static void omp_constructor(void)
{
    int ret = starpu_omp_init();
    STARPU_CHECK_RETURN_VALUE(ret, "starpu_omp_init");
}

__attribute__((destructor))
static void omp_destructor(void)
{
    starpu_omp_shutdown();
}
```

See also

[starpu_omp_init\(\)](#)
[starpu_omp_shutdown\(\)](#)

23.4 Parallel Regions and Worksharing

The SORS provides functions to create OpenMP parallel regions as well as mapping work on participating workers. The current implementation does not provide nested active parallel regions: Parallel regions may be created recursively, however only the first level parallel region may have more than one worker. From an internal point-of-view, the SORS' parallel regions are implemented as a set of implicit, extended semantics StarPU tasks, following the execution model of the OpenMP specification. Thus the SORS' parallel region tasks may block and be preempted, by SORS calls, enabling constructs such as barriers.

23.4.1 Parallel Regions

Parallel regions can be created with the function [starpu_omp_parallel_region\(\)](#) which accepts a set of attributes as parameter. The execution of the calling task is suspended until the parallel region completes. The field [starpu_omp_parallel_region_attr::cl](#) is a regular StarPU codelet. However only CPU codelets are supported for parallel regions. Here is an example of use:

```
void parallel_region_f(void *buffers[], void *args)
{
    (void) buffers;
    (void) args;
    pthread_t tid = pthread_self();
    int worker_id = starpu_worker_get_id();
    printf("[tid %p] task thread = %d\n", (void *)tid, worker_id);
}

void f(void)
{
    struct starpu_omp_parallel_region_attr attr;
    memset(&attr, 0, sizeof(attr));
    attr.cl.cpu_funcs[0] = parallel_region_f;
    attr.cl.where = STARPU_CPU;
    attr.if_clause = 1;
    starpu_omp_parallel_region(&attr);
    return 0;
}
```

See also

[struct starpu_omp_parallel_region_attr](#)
[starpu_omp_parallel_region\(\)](#)

23.4.2 Parallel For

OpenMP `for` loops are provided by the [starpu_omp_for\(\)](#) group of functions. Variants are available for inline or outlined implementations. The SORS supports `static`, `dynamic`, and `guided` loop scheduling clauses. The `auto` scheduling clause is implemented as `static`. The `runtime` scheduling clause honors the scheduling mode selected through the environment variable `OMP_SCHEDULE` or the [starpu_omp_set_schedule\(\)](#) function.

For loops with the `ordered` clause are also supported. An implicit barrier can be enforced or skipped at the end of the worksharing construct, according to the value of the `nowait` parameter.

The canonical family of [starpu_omp_for\(\)](#) functions provide each instance with the first iteration number and the number of iterations (possibly zero) to perform. The alternate family of [starpu_omp_for_alt\(\)](#) functions provide each instance with the (possibly empty) range of iterations to perform, including the first and excluding the last.

The family of [starpu_omp_ordered\(\)](#) functions enable to implement OpenMP's ordered construct, a region with a parallel for loop that is guaranteed to be executed in the sequential order of the loop iterations.

```
void for_g(unsigned long long i, unsigned long long nb_i, void *arg)
{
    (void) arg;
    for (; nb_i > 0; i++, nb_i--)
    {
        array[i] = 1;
    }
}

void parallel_region_f(void *buffers[], void *args)
{
    (void) buffers;
    (void) args;
    starpu_omp_for(for_g, NULL, NB_ITERS, CHUNK,
    starpu_omp_sched_static, 0, 0);
}
```

See also

[starpu_omp_for\(\)](#)
[starpu_omp_for_inline_first\(\)](#)
[starpu_omp_for_inline_next\(\)](#)
[starpu_omp_for_alt\(\)](#)
[starpu_omp_for_inline_first_alt\(\)](#)
[starpu_omp_for_inline_next_alt\(\)](#)
[starpu_omp_ordered\(\)](#)
[starpu_omp_ordered_inline_begin\(\)](#)
[starpu_omp_ordered_inline_end\(\)](#)

23.4.3 Sections

OpenMP sections worksharing constructs are supported using the set of [starpu_omp_sections\(\)](#) variants. The general principle is either to provide an array of per-section functions or a single function that will redirect to execution to the suitable per-section functions. An implicit barrier can be enforced or skipped at the end of the worksharing construct, according to the value of the `nowait` parameter.

```
void parallel_region_f(void *buffers[], void *args)
{
    (void) buffers;
    (void) args;

    section_funcs[0] = f;
    section_funcs[1] = g;
    section_funcs[2] = h;
    section_funcs[3] = i;

    section_args[0] = arg_f;
    section_args[1] = arg_g;
    section_args[2] = arg_h;
    section_args[3] = arg_i;

    starpu_omp_sections(4, section_f, section_args, 0);
}
```

See also

[starpu_omp_sections\(\)](#)
[starpu_omp_sections_combined\(\)](#)

23.4.4 Single

OpenMP `single` worksharing constructs are supported using the set of `starpu_omp_single()` variants. An implicit barrier can be enforced or skipped at the end of the worksharing construct, according to the value of the `nowait` parameter.

```
void single_f(void *arg)
{
    (void) arg;
    pthread_t tid = pthread_self();
    int worker_id = starpu_worker_get_id();
    printf("[tid %p] task thread = %d -- single\n", (void *)tid, worker_id);
}

void parallel_region_f(void *buffers[], void *args)
{
    (void) buffers;
    (void) args;
    starpu_omp_single(single_f, NULL, 0);
}
```

The SORS also provides dedicated support for `single` sections with `copyprivate` clauses through the `starpu_omp_single_copyprivate()` function variants. The OpenMP `master` directive is supported as well using the `starpu_omp_master()` function variants.

See also

- [starpu_omp_master\(\)](#)
- [starpu_omp_master_inline\(\)](#)
- [starpu_omp_single\(\)](#)
- [starpu_omp_single_inline\(\)](#)
- [starpu_omp_single_copyprivate\(\)](#)
- [starpu_omp_single_copyprivate_inline_begin\(\)](#)
- [starpu_omp_single_copyprivate_inline_end\(\)](#)

23.5 Tasks

The SORS implements the necessary support of OpenMP 3.1 and OpenMP 4.0's so-called explicit tasks, together with OpenMP 4.0's data dependency management.

23.5.1 Explicit Tasks

Explicit OpenMP tasks are created with the SORS using the `starpu_omp_task_region()` function. The implementation supports `if`, `final`, `untied` and `mergeable` clauses as defined in the OpenMP specification. Unless specified otherwise by the appropriate clause(s), the created task may be executed by any participating worker of the current parallel region.

The current SORS implementation requires explicit tasks to be created within the context of an active parallel region. In particular, an explicit task cannot be created by the main thread outside of a parallel region. Explicit OpenMP tasks created using `starpu_omp_task_region()` are implemented as StarPU tasks with extended semantics, and may as such be blocked and preempted by SORS routines.

The current SORS implementation supports recursive explicit tasks creation, to ensure compliance with the OpenMP specification. However, it should be noted that StarPU is not designed nor optimized for efficiently scheduling of recursive task applications.

The code below shows how to create 4 explicit tasks within a parallel region.

```
void task_region_g(void *buffers[], void *args)
{
    (void) buffers;
    (void) args;
    pthread_t tid = pthread_self();
    int worker_id = starpu_worker_get_id();
```

```

        printf("[tid %p] task thread = %d: explicit task \"%g\"\n", (void *)tid, worker_id);
    }

void parallel_region_f(void *buffers[], void *args)
{
    (void) buffers;
    (void) args;
    struct starpu_omp_task_region_attr attr;

    memset(&attr, 0, sizeof(attr));
    attr.cl.cpu_funcs[0] = task_region_g;
    attr.cl.where        = STARPU_CPU;
    attr.if_clause       = 1;
    attr.final_clause    = 0;
    attr.untied_clause   = 1;
    attr.mergeable_clause = 0;
    starpu_omp_task_region(&attr);
    starpu_omp_task_region(&attr);
    starpu_omp_task_region(&attr);
    starpu_omp_task_region(&attr);
}

```

See also

[struct starpu_omp_task_region_attr](#)
[starpu_omp_task_region\(\)](#)

23.5.2 Data Dependencies

The SORS implements inter-tasks data dependencies as specified in OpenMP 4.0. Data dependencies are expressed using regular StarPU data handles ([starpu_data_handle_t](#)) plugged into the task's `attr.cl` codelet. The family of [starpu_vector_data_register\(\)](#) -like functions and the [starpu_data_lookup\(\)](#) function may be used to register a memory area and to retrieve the current data handle associated with a pointer respectively. The testcase `./tests/openmp/task_02.c` gives a detailed example of using OpenMP 4.0 tasks dependencies with the SORS implementation.

Note: the OpenMP 4.0 specification only supports data dependencies between sibling tasks, that is tasks created by the same implicit or explicit parent task. The current SORS implementation also only supports data dependencies between sibling tasks. Consequently the behaviour is unspecified if dependencies are expressed between tasks that have not been created by the same parent task.

23.5.3 TaskWait and TaskGroup

The SORS implements both the `taskwait` and `taskgroup` OpenMP task synchronization constructs specified in OpenMP 4.0, with the [starpu_omp_taskwait\(\)](#) and [starpu_omp_taskgroup\(\)](#) functions respectively.

An example of [starpu_omp_taskwait\(\)](#) use, creating two explicit tasks and waiting for their completion:

```

void task_region_g(void *buffers[], void *args)
{
    (void) buffers;
    (void) args;
    printf("Hello, World!\n");
}

void parallel_region_f(void *buffers[], void *args)
{
    (void) buffers;
    (void) args;
    struct starpu_omp_task_region_attr attr;
    memset(&attr, 0, sizeof(attr));
    attr.cl.cpu_funcs[0] = task_region_g;
    attr.cl.where        = STARPU_CPU;
    attr.if_clause       = 1;
    attr.final_clause    = 0;
    attr.untied_clause   = 1;
    attr.mergeable_clause = 0;
    starpu_omp_task_region(&attr);
    starpu_omp_task_region(&attr);
    starpu_omp_taskwait();
}

```

An example of [starpu_omp_taskgroup\(\)](#) use, creating a task group of two explicit tasks:

```

void task_region_g(void *buffers[], void *args)
{
    (void) buffers;
    (void) args;
    printf("Hello, World!\n");
}

void taskgroup_f(void *arg)
{
    (void) arg;
    struct starpu_omp_task_region_attr attr;
    memset(&attr, 0, sizeof(attr));
    attr.cl.cpu_funcs[0] = task_region_g;
    attr.cl.where        = STARPU_CPU;
    attr.if_clause       = 1;
    attr.final_clause    = 0;
    attr.untied_clause   = 1;
    attr.mergeable_clause = 0;
    starpu_omp_task_region(&attr);
    starpu_omp_task_region(&attr);
}

void parallel_region_f(void *buffers[], void *args)
{
    (void) buffers;
    (void) args;
    starpu_omp_taskgroup(taskgroup_f, (void *)NULL);
}

```

See also

[starpu_omp_task_region\(\)](#)
[starpu_omp_taskwait\(\)](#)
[starpu_omp_taskgroup\(\)](#)
[starpu_omp_taskgroup_inline_begin\(\)](#)
[starpu_omp_taskgroup_inline_end\(\)](#)

23.6 Synchronization Support

The SORS implements objects and method to build common OpenMP synchronization constructs.

23.6.1 Simple Locks

The SORS Simple Locks are opaque [starpu_omp_lock_t](#) objects enabling multiple tasks to synchronize with each others, following the Simple Lock constructs defined by the OpenMP specification. In accordance with such specification, simple locks may not be acquired multiple times by the same task, without being released in-between; otherwise, deadlocks may result. Codes requiring the possibility to lock multiple times recursively should use Nestable Locks ([Nestable Locks](#)). Codes NOT requiring the possibility to lock multiple times recursively should use Simple Locks as they incur less processing overhead than Nestable Locks.

See also

[starpu_omp_lock_t](#)
[starpu_omp_init_lock\(\)](#)
[starpu_omp_destroy_lock\(\)](#)
[starpu_omp_set_lock\(\)](#)
[starpu_omp_unset_lock\(\)](#)
[starpu_omp_test_lock\(\)](#)

23.6.2 Nestable Locks

The SORS Nestable Locks are opaque [starpu_omp_nest_lock_t](#) objects enabling multiple tasks to synchronize with each others, following the Nestable Lock constructs defined by the OpenMP specification. In accordance with such specification, nestable locks may be acquired multiple times recursively by the same task without deadlocking.

Nested locking and unlocking operations must be well parenthesized at any time, otherwise deadlock and/or undefined behaviour may occur. Codes requiring the possibility to lock multiple times recursively should use Nestable Locks. Codes NOT requiring the possibility to lock multiple times recursively should use Simple Locks ([Simple Locks](#)) instead, as they incur less processing overhead than Nestable Locks.

See also

```
starpu_omp_nest_lock_t  
starpu_omp_init_nest_lock()  
starpu_omp_destroy_nest_lock()  
starpu_omp_set_nest_lock()  
starpu_omp_unset_nest_lock()  
starpu_omp_test_nest_lock()
```

23.6.3 Critical Sections

The SORS implements support for OpenMP critical sections through the family of `starpu_omp_critical` functions. Critical sections may optionally be named. There is a single, common anonymous critical section. Mutual exclusion only occur within the scope of single critical section, either a named one or the anonymous one.

See also

```
starpu_omp_critical()  
starpu_omp_critical_inline_begin()  
starpu_omp_critical_inline_end()
```

23.6.4 Barriers

The SORS provides the `starpu_omp_barrier()` function to implement barriers over parallel region teams. In accordance with the OpenMP specification, the `starpu_omp_barrier()` function waits for every implicit task of the parallel region to reach the barrier and every explicit task launched by the parallel region to complete, before returning.

See also

```
starpu_omp_barrier()
```


Part V

StarPU Reference API

Chapter 24

Execution Configuration Through Environment Variables

The behavior of the StarPU library and tools may be tuned thanks to the following environment variables.

24.1 Configuring Workers

STARPU_NCPU Specify the number of CPU workers (thus not including workers dedicated to control accelerators). Note that by default, StarPU will not allocate more CPU workers than there are physical CPUs, and that some CPUs are used to control the accelerators.

STARPU_NCPUS This variable is deprecated. You should use [STARPU_NCPU](#).

STARPU_NCUDA Specify the number of CUDA devices that StarPU can use. If [STARPU_NCUDA](#) is lower than the number of physical devices, it is possible to select which CUDA devices should be used by the means of the environment variable [STARPU_WORKERS_CUDAID](#). By default, StarPU will create as many CUDA workers as there are CUDA devices.

STARPU_NWORKER_PER_CUDA Specify the number of workers per CUDA device, and thus the number of kernels which will be concurrently running on the devices. The default value is 1.

STARPU_CUDA_PIPELINE Specify how many asynchronous tasks are submitted in advance on CUDA devices. This for instance permits to overlap task management with the execution of previous tasks, but it also allows concurrent execution on Fermi cards, which otherwise bring spurious synchronizations. The default is 2. Setting the value to 0 forces a synchronous execution of all tasks.

STARPU_NOPENCL OpenCL equivalent of the environment variable [STARPU_NCUDA](#).

STARPU_OPENCL_PIPELINE Specify how many asynchronous tasks are submitted in advance on OpenCL devices. This for instance permits to overlap task management with the execution of previous tasks, but it also allows concurrent execution on Fermi cards, which otherwise bring spurious synchronizations. The default is 2. Setting the value to 0 forces a synchronous execution of all tasks.

STARPU_NMICDEVS MIC equivalent of the environment variable [STARPU_NCUDA](#).

STARPU_OPENCL_ON_CPUS By default, the OpenCL driver only enables GPU and accelerator devices. By setting the environment variable [STARPU_OPENCL_ON_CPUS](#) to 1, the OpenCL driver will also enable CPU devices.

STARPU_OPENCL_ONLY_ON_CPUS By default, the OpenCL driver enables GPU and accelerator devices. By setting the environment variable [STARPU_OPENCL_ONLY_ON_CPUS](#) to 1, the OpenCL driver will ONLY enable CPU devices.

STARPU_NMIC MIC equivalent of the environment variable [STARPU_NCUDA](#).

STARPU_NSCC SCC equivalent of the environment variable [STARPU_NCUDA](#).

STARPU_WORKERS_NOBIND Setting it to non-zero will prevent StarPU from binding its threads to CPUs. This is for instance useful when running the testsuite in parallel.

STARPU_WORKERS_CPUID Passing an array of integers (starting from 0) in [STARPU_WORKERS_CPUID](#) specifies on which logical CPU the different workers should be bound. For instance, if `STARPU_WORKERS_CPUID = "0 1 4 5"`, the first worker will be bound to logical CPU #0, the second CPU worker will be bound to logical CPU #1 and so on. Note that the logical ordering of the CPUs is either determined by the OS, or provided by the library `hwloc` in case it is available.

Note that the first workers correspond to the CUDA workers, then come the OpenCL workers, and finally the CPU workers. For example if we have `STARPU_NCUDA=1`, `STARPU_NOPENCL=1`, `STARPU_NCPU=2` and `STARPU_WORKERS_CPUID = "0 2 1 3"`, the CUDA device will be controlled by logical CPU #0, the OpenCL device will be controlled by logical CPU #2, and the logical CPUs #1 and #3 will be used by the CPU workers.

If the number of workers is larger than the array given in [STARPU_WORKERS_CPUID](#), the workers are bound to the logical CPUs in a round-robin fashion: if `STARPU_WORKERS_CPUID = "0 1"`, the first and the third (resp. second and fourth) workers will be put on CPU #0 (resp. CPU #1).

This variable is ignored if the field `starpu_conf::use_explicit_workers_bindid` passed to `starpu_init()` is set.

STARPU_WORKERS_CUDAID Similarly to the [STARPU_WORKERS_CPUID](#) environment variable, it is possible to select which CUDA devices should be used by StarPU. On a machine equipped with 4 GPUs, setting `STARPU_WORKERS_CUDAID = "1 3"` and `STARPU_NCUDA=2` specifies that 2 CUDA workers should be created, and that they should use CUDA devices #1 and #3 (the logical ordering of the devices is the one reported by CUDA).

This variable is ignored if the field `starpu_conf::use_explicit_workers_cuda_gpuid` passed to `starpu_init()` is set.

STARPU_WORKERS_OPENCLID OpenCL equivalent of the [STARPU_WORKERS_CUDAID](#) environment variable.

This variable is ignored if the field `starpu_conf::use_explicit_workers_opencl_gpuid` passed to `starpu_init()` is set.

STARPU_WORKERS_MICID MIC equivalent of the [STARPU_WORKERS_CUDAID](#) environment variable.

This variable is ignored if the field `starpu_conf::use_explicit_workers_mic_deviceid` passed to `starpu_init()` is set.

STARPU_WORKERS_SCCID SCC equivalent of the [STARPU_WORKERS_CUDAID](#) environment variable.

This variable is ignored if the field `starpu_conf::use_explicit_workers_scc_deviceid` passed to `starpu_init()` is set.

STARPU_SINGLE_COMBINED_WORKER If set, StarPU will create several workers which won't be able to work concurrently. It will by default create combined workers which size goes from 1 to the total number of CPU workers in the system. [STARPU_MIN_WORKERSIZE](#) and [STARPU_MAX_WORKERSIZE](#) can be used to change this default.

STARPU_MIN_WORKERSIZE [STARPU_MIN_WORKERSIZE](#) permits to specify the minimum size of the combined workers (instead of the default 2)

STARPU_MAX_WORKERSIZE [STARPU_MAX_WORKERSIZE](#) permits to specify the minimum size of the combined workers (instead of the number of CPU workers in the system)

STARPU_SYNTHESIZE_ARITY_COMBINED_WORKER Let the user decide how many elements are allowed between combined workers created from `hwloc` information. For instance, in the case of sockets with 6 cores without shared L2 caches, if [STARPU_SYNTHESIZE_ARITY_COMBINED_WORKER](#) is set to 6, no combined worker will be synthesized beyond one for the socket and one per core. If it is set to 3, 3 intermediate combined workers will be synthesized, to divide the socket cores into 3 chunks of 2 cores. If it is set to 2, 2 intermediate combined workers will be synthesized, to divide the the socket cores into 2 chunks of 3 cores, and then 3 additional combined workers will be synthesized, to divide the former synthesized workers into a

bunch of 2 cores, and the remaining core (for which no combined worker is synthesized since there is already a normal worker for it).

The default, 2, thus makes StarPU tend to building a binary trees of combined workers.

STARPU_DISABLE_ASYNCHRONOUS_COPY Disable asynchronous copies between CPU and GPU devices. The AMD implementation of OpenCL is known to fail when copying data asynchronously. When using this implementation, it is therefore necessary to disable asynchronous data transfers.

STARPU_DISABLE_ASYNCHRONOUS_CUDA_COPY Disable asynchronous copies between CPU and CUDA devices.

STARPU_DISABLE_ASYNCHRONOUS_OPENCL_COPY Disable asynchronous copies between CPU and OpenCL devices. The AMD implementation of OpenCL is known to fail when copying data asynchronously. When using this implementation, it is therefore necessary to disable asynchronous data transfers.

STARPU_DISABLE_ASYNCHRONOUS_MIC_COPY Disable asynchronous copies between CPU and MIC devices.

STARPU_ENABLE_CUDA_GPU_GPU_DIRECT Enable (1) or Disable (0) direct CUDA transfers from GPU to GPU, without copying through RAM. The default is Enabled. This permits to test the performance effect of GPU-Direct.

STARPU_DISABLE_PINNING Disable (1) or Enable (0) pinning host memory allocated through `starpu_malloc` and friends. The default is Enabled. This permits to test the performance effect of memory pinning.

24.2 Configuring The Scheduling Engine

STARPU_SCHED Choose between the different scheduling policies proposed by StarPU: work random, stealing, greedy, with performance models, etc.

Use `STARPU_SCHED=help` to get the list of available schedulers.

STARPU_CALIBRATE If this variable is set to 1, the performance models are calibrated during the execution. If it is set to 2, the previous values are dropped to restart calibration from scratch. Setting this variable to 0 disable calibration, this is the default behaviour.

Note: this currently only applies to `dm` and `dmda` scheduling policies.

STARPU_CALIBRATE_MINIMUM This defines the minimum number of calibration measurements that will be made before considering that the performance model is calibrated. The default value is 10.

STARPU_BUS_CALIBRATE If this variable is set to 1, the bus is recalibrated during initialization.

STARPU_PREFETCH This variable indicates whether data prefetching should be enabled (0 means that it is disabled). If prefetching is enabled, when a task is scheduled to be executed e.g. on a GPU, StarPU will request an asynchronous transfer in advance, so that data is already present on the GPU when the task starts. As a result, computation and data transfers are overlapped. Note that prefetching is enabled by default in StarPU.

STARPU_SCHED_ALPHA To estimate the cost of a task StarPU takes into account the estimated computation time (obtained thanks to performance models). The alpha factor is the coefficient to be applied to it before adding it to the communication part.

STARPU_SCHED_BETA To estimate the cost of a task StarPU takes into account the estimated data transfer time (obtained thanks to performance models). The beta factor is the coefficient to be applied to it before adding it to the computation part.

STARPU_SCHED_GAMMA Define the execution time penalty of a joule ([Power-based Scheduling](#)).

STARPU_IDLE_POWER Define the idle power of the machine ([Power-based Scheduling](#)).

STARPU_PROFILING Enable on-line performance monitoring ([Enabling On-line Performance Monitoring](#)).

24.3 Extensions

SOCL_OCL_LIB_OPENCL THE SOCL test suite is only run when the environment variable **SOCL_OCL_LIB_OPENCL** is defined. It should contain the location of the file `libOpenCL.so` of the OCL ICD implementation.

OCL_ICD_VENDORS When using SOCL with OpenCL ICD (<https://forge.imag.fr/projects/ocl-icd/>), this variable may be used to point to the directory where ICD files are installed. The default directory is `/etc/OpenCL/vendors`. StarPU installs ICD files in the directory `$prefix/share/starpu/opencl/vendors`.

STARPU_COMM_STATS Communication statistics for `starpumpi` (MPI Support) will be enabled when the environment variable **STARPU_COMM_STATS** is defined to an value other than 0.

STARPU_MPI_CACHE Communication cache for `starpumpi` (MPI Support) will be disabled when the environment variable **STARPU_MPI_CACHE** is set to 0. It is enabled by default or for any other values of the variable **STARPU_MPI_CACHE**.

STARPU_MPI_CACHE_STATS When set to 1, statistics are enabled for the communication cache (MPI Support). For now, it prints messages on the standard output when data are added or removed from the received communication cache.

24.4 Miscellaneous And Debug

STARPU_HOME This specifies the main directory in which StarPU stores its configuration files. The default is `$HOME` on Unix environments, and `$USERPROFILE` on Windows environments.

STARPU_PERF_MODEL_DIR This specifies the main directory in which StarPU stores its performance model files. The default is `$STARPU_HOME/.starpu/sampling`.

STARPU_HOSTNAME When set, force the hostname to be used when dealing performance model files. Models are indexed by machine name. When running for example on a homogenous cluster, it is possible to share the models between machines by setting `export STARPU_HOSTNAME=some_global_name`.

STARPU_OPENCL_PROGRAM_DIR This specifies the directory where the OpenCL codelet source files are located. The function `starpu_opencl_load_program_source()` looks for the codelet in the current directory, in the directory specified by the environment variable **STARPU_OPENCL_PROGRAM_DIR**, in the directory `share/starpu/opencl` of the installation directory of StarPU, and finally in the source directory of StarPU.

STARPU_SILENT This variable allows to disable verbose mode at runtime when StarPU has been configured with the option `--enable-verbose`. It also disables the display of StarPU information and warning messages.

STARPU_LOGFILENAME This variable specifies in which file the debugging output should be saved to.

STARPU_FXT_PREFIX This variable specifies in which directory to save the trace generated if FxT is enabled. It needs to have a trailing `'/'` character.

STARPU_LIMIT_CUDA_devid_MEM This variable specifies the maximum number of megabytes that should be available to the application on the CUDA device with the identifier `devid`. This variable is intended to be used for experimental purposes as it emulates devices that have a limited amount of memory. When defined, the variable overwrites the value of the variable **STARPU_LIMIT_CUDA_MEM**.

STARPU_LIMIT_CUDA_MEM This variable specifies the maximum number of megabytes that should be available to the application on each CUDA devices. This variable is intended to be used for experimental purposes as it emulates devices that have a limited amount of memory.

STARPU_LIMIT_OPENCL_devid_MEM This variable specifies the maximum number of megabytes that should be available to the application on the OpenCL device with the identifier `devid`. This variable is intended to be used for experimental purposes as it emulates devices that have a limited amount of memory. When defined, the variable overwrites the value of the variable **STARPU_LIMIT_OPENCL_MEM**.

STARPU_LIMIT_OPENCL_MEM This variable specifies the maximum number of megabytes that should be available to the application on each OpenCL devices. This variable is intended to be used for experimental purposes as it emulates devices that have a limited amount of memory.

STARPU_LIMIT_CPU_MEM This variable specifies the maximum number of megabytes that should be available to the application on each CPU device. Setting it enables allocation cache in main memory

STARPU_MINIMUM_AVAILABLE_MEM This specifies the minimum percentage of memory that should be available in GPUs (or in main memory, when using out of core), below which a reclaiming pass is performed. The default is 5%.

STARPU_TARGET_AVAILABLE_MEM This specifies the target percentage of memory that should be reached in GPUs (or in main memory, when using out of core), when performing a periodic reclaiming pass. The default is 10%.

STARPU_DISK_SWAP This specifies a path where StarPU can push data when the main memory is getting full.

STARPU_DISK_SWAP_BACKEND This specifies then backend to be used by StarPU to push data when the main memory is getting full. The default is `unistd` (i.e. using read/write functions), other values are `stdio` (i.e. using `fread/fwrite`), `unistd_o_direct` (i.e. using read/write with `O_DIRECT`), and `leveldb` (i.e. using a leveldb database).

STARPU_DISK_SWAP_SIZE This specifies then size to be used by StarPU to push data when the main memory is getting full. The default is unlimited.

STARPU_TRACE_BUFFER_SIZE This sets the buffer size for recording trace events in MiB. Setting it to a big size allows to avoid pauses in the trace while it is recorded on the disk. This however also consumes memory, of course. The default value is 64.

STARPU_GENERATE_TRACE When set to 1, this variable indicates that StarPU should automatically generate a Paje trace when `starpu_shutdown()` is called.

STARPU_MEMORY_STATS When set to 0, disable the display of memory statistics on data which have not been unregistered at the end of the execution ([Memory Feedback](#)).

STARPU_BUS_STATS When defined, statistics about data transfers will be displayed when calling `starpu_shutdown()` ([Profiling](#)).

STARPU_WORKER_STATS When defined, statistics about the workers will be displayed when calling `starpu_shutdown()` ([Profiling](#)). When combined with the environment variable `STARPU_PROFILING`, it displays the power consumption ([Power-based Scheduling](#)).

STARPU_STATS When set to 0, data statistics will not be displayed at the end of the execution of an application ([Data Statistics](#)).

STARPU_WATCHDOG_TIMEOUT When set to a value other than 0, allows to make StarPU print an error message whenever StarPU does not terminate any task for the given time (in μ s). Should be used in combination with `STARPU_WATCHDOG_CRASH` (see [Detection Stuck Conditions](#)).

STARPU_WATCHDOG_CRASH When set to a value other than 0, it triggers a crash when the watch dog is reached, thus allowing to catch the situation in gdb, etc (see [Detection Stuck Conditions](#))

STARPU_DISABLE_KERNELS When set to a value other than 1, it disables actually calling the kernel functions, thus allowing to quickly check that the task scheme is working properly, without performing the actual application-provided computation.

STARPU_HISTORY_MAX_ERROR History-based performance models will drop measurements which are really far from the measured average. This specifies the allowed variation. The default is 50 (%), i.e. the measurement is allowed to be x1.5 faster or /1.5 slower than the average.

24.5 Configuring The Hypervisor

SC_HYPERVISOR_POLICY Choose between the different resizing policies proposed by StarPU for the hypervisor: `idle`, `app_driven`, `feft_lp`, `teft_lp`; `ispeed_lp`, `throughput_lp` etc.

Use `SC_HYPERVISOR_POLICY=help` to get the list of available policies for the hypervisor

SC_HYPERVISOR_TRIGGER_RESIZE Choose how should the hypervisor be triggered: `speed` if the resizing algorithm should be called whenever the speed of the context does not correspond to an optimal precomputed value, `idle` if the resizing algorithm should be called whenever the workers are idle for a period longer than the value indicated when configuring the hypervisor.

SC_HYPERVISOR_START_RESIZE Indicate the moment when the resizing should be available. The value correspond to the percentage of the total time of execution of the application. The default value is the resizing frame.

SC_HYPERVISOR_MAX_SPEED_GAP Indicate the ratio of speed difference between contexts that should trigger the hypervisor. This situation may occur only when a theoretical speed could not be computed and the hypervisor has no value to compare the speed to. Otherwise the resizing of a context is not influenced by the the speed of the other contexts, but only by the the value that a context should have.

SC_HYPERVISOR_STOP_PRINT By default the values of the speed of the workers is printed during the execution of the application. If the value 1 is given to this environment variable this printing is not done.

SC_HYPERVISOR_LAZY_RESIZE By default the hypervisor resizes the contexts in a lazy way, that is workers are firstly added to a new context before removing them from the previous one. Once this workers are clearly taken into account into the new context (a task was popped there) we remove them from the previous one. However if the application would like that the change in the distribution of workers should change right away this variable should be set to 0

SC_HYPERVISOR_SAMPLE_CRITERIA By default the hypervisor uses a sample of flops when computing the speed of the contexts and of the workers. If this variable is set to `time` the hypervisor uses a sample of time (10% of an approximation of the total execution time of the application)

Chapter 25

Compilation Configuration

The behavior of the StarPU library and tools may be tuned thanks to the following configure options.

25.1 Common Configuration

-enable-debug Enable debugging messages.

-enable-spinlock-check Enable checking that spinlocks are taken and released properly.

-enable-fast Disable assertion checks, which saves computation time.

-enable-verbose Increase the verbosity of the debugging messages. This can be disabled at runtime by setting the environment variable `STARPU_SILENT` to any value. `-enable-verbose=extra` increase even more the verbosity.

```
$ STARPU_SILENT=1 ./vector_scal
```

-enable-coverage Enable flags for the coverage tool `gcov`.

-enable-quick-check Specify tests and examples should be run on a smaller data set, i.e allowing a faster execution time

-enable-long-check Enable some exhaustive checks which take a really long time.

-enable-new-check Enable new testcases which are known to fail.

-with-hwloc Specify `hwloc` should be used by StarPU. `hwloc` should be found by the means of the tool `pkg-config`.

-with-hwloc=prefix Specify `hwloc` should be used by StarPU. `hwloc` should be found in the directory specified by `prefix`

-without-hwloc Specify `hwloc` should not be used by StarPU.

-disable-build-doc Disable the creation of the documentation. This should be done on a machine which does not have the tools `doxygen` and `latex` (plus the packages `latex-xcolor` and `texlive-latex-extra`).

Additionally, the script `configure` recognize many variables, which can be listed by typing `./configure -help`. For example, `./configure NVCCFLAGS="-arch sm_13"` adds a flag for the compilation of CUDA kernels.

25.2 Configuring Workers

- enable-maxcpus=count** Use at most `count` CPU cores. This information is then available as the macro `::STARPU_MAXCPUS`.
- disable-cpu** Disable the use of CPUs of the machine. Only GPUs etc. will be used.
- enable-maxcudadev=count** Use at most `count` CUDA devices. This information is then available as the macro `STARPU_MAXCUDADEVs`.
- disable-cuda** Disable the use of CUDA, even if a valid CUDA installation was detected.
- with-cuda-dir=prefix** Search for CUDA under `prefix`, which should notably contain the file `include/cuda.h`.
- with-cuda-include-dir=dir** Search for CUDA headers under `dir`, which should notably contain the file `cuda.h`. This defaults to `/include` appended to the value given to `--with-cuda-dir`.
- with-cuda-lib-dir=dir** Search for CUDA libraries under `dir`, which should notably contain the CUDA shared libraries—e.g., `libcuda.so`. This defaults to `/lib` appended to the value given to `--with-cuda-dir`.
- disable-cuda-memcpy-peer** Explicitly disable peer transfers when using CUDA 4.0.
- enable-maxopencldev=count** Use at most `count` OpenCL devices. This information is then available as the macro `STARPU_MAXOPENCLDEVs`.
- disable-opencl** Disable the use of OpenCL, even if the SDK is detected.
- with-opencl-dir=prefix** Search for an OpenCL implementation under `prefix`, which should notably contain `include/CL/cl.h` (or `include/OpenCL/cl.h` on Mac OS).
- with-opencl-include-dir=dir** Search for OpenCL headers under `dir`, which should notably contain `CL/cl.h` (or `OpenCL/cl.h` on Mac OS). This defaults to `/include` appended to the value given to `--with-opencl-dir`.
- with-opencl-lib-dir=dir** Search for an OpenCL library under `dir`, which should notably contain the OpenCL shared libraries—e.g. `libOpenCL.so`. This defaults to `/lib` appended to the value given to `--with-opencl-dir`.
- enable-opencl-simulator** Enable considering the provided OpenCL implementation as a simulator, i.e. use the kernel duration returned by OpenCL profiling information as wallclock time instead of the actual measured real time. This requires `simgrid` support.
- enable-maximplementations=count** Allow for at most `count` codelet implementations for the same target device. This information is then available as the macro `::STARPU_MAXIMPLEMENTATIONS` macro.
- enable-max-sched-ctxs=count** Allow for at most `count` scheduling contexts This information is then available as the macro `::STARPU_NMAX_SCHED_CTXS`.
- disable-asynchronous-copy** Disable asynchronous copies between CPU and GPU devices. The AMD implementation of OpenCL is known to fail when copying data asynchronously. When using this implementation, it is therefore necessary to disable asynchronous data transfers.
- disable-asynchronous-cuda-copy** Disable asynchronous copies between CPU and CUDA devices.
- disable-asynchronous-opencl-copy** Disable asynchronous copies between CPU and OpenCL devices. The AMD implementation of OpenCL is known to fail when copying data asynchronously. When using this implementation, it is therefore necessary to disable asynchronous data transfers.
- enable-maxmicthreads** Specify the maximum number of MIC threads
- disable-asynchronous-mic-copy** Disable asynchronous copies between CPU and MIC devices.

25.3 Extension Configuration

- disable-socl** Disable the SOCL extension ([SOCL OpenCL Extensions](#)). By default, it is enabled when an OpenCL implementation is found.
- disable-starpu-top** Disable the StarPU-Top interface ([StarPU-Top Interface](#)). By default, it is enabled when the required dependencies are found.
- disable-gcc-extensions** Disable the GCC plug-in ([C Extensions](#)). By default, it is enabled when the GCC compiler provides a plug-in support.
- with-mpicc=path** Use the compiler `mpicc` at `path`, for StarPU-MPI. ([MPI Support](#)).
- enable-mpi-progression-hook** Enable the activity polling method for StarPU-MPI.
- with-coi-dir** Specify the directory to the COI library for MIC support. The default value is `/opt/intel/mic/coi`
- mic-host** Specify the precise MIC architecture host identifier. The default value is `x86_64-k10m-linux`
- enable-openmp** Enable OpenMP Support ([The StarPU OpenMP Runtime Support \(SORS\)](#))

25.4 Advanced Configuration

- enable-perf-debug** Enable performance debugging through `gprof`.
- enable-model-debug** Enable performance model debugging.
- enable-paje-codelet-details** Enable details about codelets in the paje trace. This requires a recent enough version of ViTE (at least r1430).
- enable-fxt-lock** Enable additional trace events which describes locks behaviour.
- enable-stats** (see `../src/datawizard/datastats.c`) Enable gathering of various data statistics ([Data Statistics](#)).
- enable-maxbuffers** Define the maximum number of buffers that tasks will be able to take as parameters, then available as the macro `STARPU_NMAXBUFS`.
- enable-allocation-cache** Enable the use of a data allocation cache to avoid the cost of it with CUDA. Still experimental.
- enable-opengl-render** Enable the use of OpenGL for the rendering of some examples.
- enable-blas-lib** Specify the blas library to be used by some of the examples. The library has to be 'atlas' or 'goto'.
- disable-starpufft** Disable the build of `libstarpufft`, even if `fftw` or `cuFFT` is available.
- enable-starpufft-examples** Enable the compilation and the execution of the `libstarpufft` examples. By default, they are neither compiled nor checked.
- with-magma=prefix** Search for MAGMA under `prefix`. `prefix` should notably contain `include/magmablas.h`.
- with-fxt=prefix** Search for FxT under `prefix`. FxT (<http://savannah.nongnu.org/projects/fkt>) is used to generate traces of scheduling events, which can then be rendered them using ViTE ([Off-line Performance Feedback](#)). `prefix` should notably contain `include/fxt/fxt.h`.
- with-perf-model-dir=dir** Store performance models under `dir`, instead of the current user's home.
- with-goto-dir=prefix** Search for GotoBLAS under `prefix`, which should notably contain `libgoto.so` or `libgoto2.so`.
- with-atlas-dir=prefix** Search for ATLAS under `prefix`, which should notably contain `include/cblas.h`.

- with-mkl-cflags=cflags** Use `cflags` to compile code that uses the MKL library.
- with-mkl-ldflags=ldflags** Use `ldflags` when linking code that uses the MKL library. Note that the MKL website (<http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/>) provides a script to determine the linking flags.
- disable-build-examples** Disable the build of examples.
- enable-sc-hypervisor** Enable the Scheduling Context Hypervisor plugin([Scheduling Context Hypervisor](#)). By default, it is disabled.
- enable-memory-stats** Enable memory statistics ([Memory Feedback](#)).
- enable-simgrid** Enable simulation of execution in simgrid, to allow easy experimentation with various numbers of cores and GPUs, or amount of memory, etc. Experimental.
- The path to simgrid can be specified through the `SIMGRID_CFLAGS` and `SIMGRID_LIBS` environment variables, for instance:
- ```
export SIMGRID_CFLAGS="-I/usr/local/simgrid/include"
export SIMGRID_LIBS="-L/usr/local/simgrid/lib -lsimgrid"
```
- with-simgrid-dir** Similar to the option `--enable-simgrid` but also allows to specify the location to the SimGrid library.
- with-simgrid-include-dir** Similar to the option `--enable-simgrid` but also allows to specify the location to the SimGrid include directory.
- with-simgrid-lib-dir** Similar to the option `--enable-simgrid` but also allows to specify the location to the SimGrid lib directory.
- enable-calibration-heuristic** Allows to set the maximum authorized percentage of deviation for the history-based calibrator of StarPU. A correct value of this parameter must be in `[0..100]`. The default value of this parameter is 10. Experimental.

## Chapter 26

# Module Index

### 26.1 Modules

Here is a list of all modules:

|                                                     |     |
|-----------------------------------------------------|-----|
| Bitmap . . . . .                                    | 171 |
| Codelet And Tasks . . . . .                         | 217 |
| CUDA Extensions . . . . .                           | 249 |
| Data Interfaces . . . . .                           | 183 |
| Data Management . . . . .                           | 178 |
| Out Of Core . . . . .                               | 212 |
| Data Partition . . . . .                            | 206 |
| Expert Mode . . . . .                               | 294 |
| Explicit Dependencies . . . . .                     | 235 |
| FFT Support . . . . .                               | 281 |
| FxT Support . . . . .                               | 279 |
| Implicit Data Dependencies . . . . .                | 237 |
| Initialization and Termination . . . . .            | 154 |
| Insert_Task . . . . .                               | 233 |
| Theoretical Lower Bound on Execution Time . . . . . | 248 |
| MIC Extensions . . . . .                            | 276 |
| Miscellaneous Helpers . . . . .                     | 278 |
| Modularized Scheduler Interface . . . . .           | 321 |
| MPI Support . . . . .                               | 282 |
| Multiformat Data Interface . . . . .                | 215 |
| OpenCL Extensions . . . . .                         | 250 |
| OpenMP Runtime Support . . . . .                    | 256 |
| Parallel Tasks . . . . .                            | 292 |
| Performance Model . . . . .                         | 238 |
| Profiling . . . . .                                 | 244 |
| Running Drivers . . . . .                           | 294 |
| SCC Extensions . . . . .                            | 277 |
| Scheduling Contexts . . . . .                       | 300 |
| Scheduling Policy . . . . .                         | 306 |
| Standard Memory Library . . . . .                   | 158 |
| Task Bundles . . . . .                              | 289 |
| Task Lists . . . . .                                | 291 |
| Threads . . . . .                                   | 163 |
| Toolbox . . . . .                                   | 161 |
| StarPU-Top Interface . . . . .                      | 295 |
| Tree . . . . .                                      | 311 |
| Versioning . . . . .                                | 153 |
| Workers' Properties . . . . .                       | 173 |

|                                                                          |     |
|--------------------------------------------------------------------------|-----|
| Scheduling Context Hypervisor - Building a new resizing policy . . . . . | 316 |
| Scheduling Context Hypervisor - Regular usage . . . . .                  | 312 |

## Chapter 27

# Module Documentation a.k.a StarPU's API

### 27.1 Versioning

#### Macros

- `#define STARPU_MAJOR_VERSION`
- `#define STARPU_MINOR_VERSION`
- `#define STARPU_RELEASE_VERSION`

#### Functions

- `void starpu_get_version (int *major, int *minor, int *release)`

#### 27.1.1 Detailed Description

#### 27.1.2 Macro Definition Documentation

##### 27.1.2.1 `#define STARPU_MAJOR_VERSION`

Define the major version of StarPU. This is the version used when compiling the application.

##### 27.1.2.2 `#define STARPU_MINOR_VERSION`

Define the minor version of StarPU. This is the version used when compiling the application.

##### 27.1.2.3 `#define STARPU_RELEASE_VERSION`

Define the release version of StarPU. This is the version used when compiling the application.

#### 27.1.3 Function Documentation

##### 27.1.3.1 `void starpu_get_version ( int * major, int * minor, int * release )`

Return as 3 integers the version of StarPU used when running the application.

## 27.2 Initialization and Termination

### Data Structures

- struct [starpu\\_driver](#)
- union [starpu\\_driver.id](#)
- struct [starpu\\_conf](#)

### Functions

- int [starpu\\_init](#) (struct [starpu\\_conf](#) \*conf) [STARPU\\_WARN\\_UNUSED\\_RESULT](#)
- int [starpu\\_initialize](#) (struct [starpu\\_conf](#) \*user\_conf, int \*argc, char \*\*\*argv)
- int [starpu\\_conf\\_init](#) (struct [starpu\\_conf](#) \*conf)
- void [starpu\\_shutdown](#) (void)
- void [starpu\\_pause](#) (void)
- void [starpu\\_resume](#) (void)
- int [starpu\\_asynchronous\\_copy\\_disabled](#) (void)
- int [starpu\\_asynchronous\\_cuda\\_copy\\_disabled](#) (void)
- int [starpu\\_asynchronous\\_opencl\\_copy\\_disabled](#) (void)
- void [starpu\\_topology\\_print](#) (FILE \*f)

#### 27.2.1 Detailed Description

#### 27.2.2 Data Structure Documentation

##### 27.2.2.1 struct [starpu\\_driver](#)

structure for a driver

##### Data Fields

|                                             |      |                                                                                                                                                                       |
|---------------------------------------------|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enum <a href="#">starpu_worker_archtype</a> | type | The type of the driver. Only <a href="#">STARPU_CPU_WORKER</a> , <a href="#">STARPU_CUDA_WORKER</a> and <a href="#">STARPU_OPENCL_WORKER</a> are currently supported. |
| union <a href="#">starpu_driver</a>         | id   | The identifier of the driver.                                                                                                                                         |

##### 27.2.2.2 union [starpu\\_driver.id](#)

##### Data Fields

|              |           |  |
|--------------|-----------|--|
| unsigned     | cpu_id    |  |
| unsigned     | cuda_id   |  |
| cl_device_id | opencl_id |  |

##### 27.2.2.3 struct [starpu\\_conf](#)

This structure is passed to the [starpu\\_init\(\)](#) function in order to configure StarPU. It has to be initialized with [starpu\\_conf\\_init\(\)](#). When the default value is used, StarPU automatically selects the number of processing units and takes the default scheduling policy. The environment variables overwrite the equivalent parameters.



## Data Fields

|                                           |                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------------------------|-----------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int                                       | magic                                               | Will be initialized by <code>starpu_conf_init()</code> . Should not be set by hand.                                                                                                                                                                                                                                                                                                                                             |
| const char *                              | <code>sched_policy_name</code>                      | This is the name of the scheduling policy. This can also be specified with the environment variable <code>STARPU_SCHED</code> . (default = NULL).                                                                                                                                                                                                                                                                               |
| struct <code>starpu_sched_policy</code> * | <code>sched_policy</code>                           | This is the definition of the scheduling policy. This field is ignored if <code>starpu_conf::sched_policy_name</code> is set. (default = NULL)                                                                                                                                                                                                                                                                                  |
| int                                       | <code>ncpus</code>                                  | This is the number of CPU cores that StarPU can use. This can also be specified with the environment variable <code>STARPU_NCPU</code> . (default = -1)                                                                                                                                                                                                                                                                         |
| int                                       | <code>ncuda</code>                                  | This is the number of CUDA devices that StarPU can use. This can also be specified with the environment variable <code>STARPU_NCUDA</code> . (default = -1)                                                                                                                                                                                                                                                                     |
| int                                       | <code>nopencl</code>                                | This is the number of OpenCL devices that StarPU can use. This can also be specified with the environment variable <code>STARPU_NOPENCL</code> . (default = -1)                                                                                                                                                                                                                                                                 |
| int                                       | <code>nmic</code>                                   | This is the number of MIC devices that StarPU can use. This can also be specified with the environment variable <code>STARPU_NMIC</code> . (default = -1)                                                                                                                                                                                                                                                                       |
| int                                       | <code>nsccl</code>                                  | This is the number of SCC devices that StarPU can use. This can also be specified with the environment variable <code>STARPU_NSCC</code> . (default = -1)                                                                                                                                                                                                                                                                       |
| unsigned                                  | <code>use_explicit_workers_bindid</code>            | If this flag is set, the <code>starpu_conf::workers_bindid</code> array indicates where the different workers are bound, otherwise StarPU automatically selects where to bind the different workers. This can also be specified with the environment variable <code>STARPU_WORKERS_CPUID</code> . (default = 0)                                                                                                                 |
| unsigned                                  | <code>workers_bindid[STARPU_NMAXWORKERS]</code>     | If the <code>starpu_conf::use_explicit_workers_bindid</code> flag is set, this array indicates where to bind the different workers. The i-th entry of the <code>starpu_conf::workers_bindid</code> indicates the logical identifier of the processor which should execute the i-th worker. Note that the logical ordering of the CPUs is either determined by the OS, or provided by the hwloc library in case it is available. |
| unsigned                                  | <code>use_explicit_workers_cuda_gguid</code>        | If this flag is set, the CUDA workers will be attached to the CUDA devices specified in the <code>starpu_conf::workers_cuda_gguid</code> array. Otherwise, StarPU affects the CUDA devices in a round-robin fashion. This can also be specified with the environment variable <code>STARPU_WORKERS_CUDAID</code> . (default = 0)                                                                                                |
| unsigned                                  | <code>workers_cuda_gguid[STARPU_NMAXWORKERS]</code> | If the <code>starpu_conf::use_explicit_workers_cuda_gguid</code> flag is set, this array contains the logical identifiers of the CUDA devices (as used by <code>cudaGetDevice()</code> ).                                                                                                                                                                                                                                       |
| unsigned                                  | <code>use_explicit_workers_opencl_gguid</code>      | If this flag is set, the OpenCL workers will be attached to the OpenCL devices specified in the <code>starpu_conf::workers_opencl_gguid</code> array. Otherwise, StarPU affects the OpenCL devices in a round-robin fashion. This can also be specified with the environment variable <code>STARPU_WORKERS_OPENCLID</code> . (default = 0)                                                                                      |

|          |                                                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------|--------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| unsigned | <code>workers_↵<br/>opengl_↵<br/>gpuid[STAR↵<br/>RPU_NMAXW↵<br/>ORKERS]</code> | If the <code>starpu_conf::use_explicit_workers_opengl_gpuid</code> flag is set, this array contains the logical identifiers of the OpenCL devices to be used.                                                                                                                                                                                                                                                                                                                                                                                                                           |
| unsigned | <code>use_explicit_↵<br/>workers_mic_↵<br/>deviceid</code>                     | If this flag is set, the MIC workers will be attached to the MIC devices specified in the array <code>starpu_conf::workers_mic_deviceid</code> . Otherwise, StarPU affects the MIC devices in a round-robin fashion. This can also be specified with the environment variable <code>STARPU_WORKERS_MIC_↵<br/>ID</code> . (default = 0)                                                                                                                                                                                                                                                  |
| unsigned | <code>workers_mic_↵<br/>deviceid[STAR↵<br/>PU_NMAXWO↵<br/>RKERS]</code>        | If the flag <code>starpu_conf::use_explicit_workers_mic_deviceid</code> is set, the array contains the logical identifiers of the MIC devices to be used.                                                                                                                                                                                                                                                                                                                                                                                                                               |
| unsigned | <code>use_explicit_↵<br/>workers_scc_↵<br/>deviceid</code>                     | If this flag is set, the SCC workers will be attached to the SCC devices specified in the array <code>starpu_conf::workers_scc_deviceid</code> . (default = 0)                                                                                                                                                                                                                                                                                                                                                                                                                          |
| unsigned | <code>workers_scc_↵<br/>deviceid[STAR↵<br/>PU_NMAXWO↵<br/>RKERS]</code>        | If the flag <code>starpu_conf::use_explicit_workers_scc_deviceid</code> is set, the array contains the logical identifiers of the SCC devices to be used. Otherwise, StarPU affects the SCC devices in a round-robin fashion. This can also be specified with the environment variable <code>STARPU_WORKERS_↵<br/>_SCCID</code> .                                                                                                                                                                                                                                                       |
| int      | <code>bus_calibrate</code>                                                     | If this flag is set, StarPU will recalibrate the bus. If this value is equal to -1, the default value is used. This can also be specified with the environment variable <code>STARPU_BUS_CALIBRATE</code> . (default = 0)                                                                                                                                                                                                                                                                                                                                                               |
| int      | <code>calibrate</code>                                                         | If this flag is set, StarPU will calibrate the performance models when executing tasks. If this value is equal to -1, the default value is used. If the value is equal to 1, it will force continuing calibration. If the value is equal to 2, the existing performance models will be overwritten. This can also be specified with the environment variable <code>STARPU_CALIBRATE</code> . (default = 0)                                                                                                                                                                              |
| int      | <code>single_↵<br/>combined_↵<br/>worker</code>                                | By default, StarPU executes parallel tasks concurrently. Some parallel libraries (e.g. most OpenMP implementations) however do not support concurrent calls to parallel code. In such case, setting this flag makes StarPU only start one parallel task at a time (but other CPU and GPU tasks are not affected and can be run concurrently). The parallel task scheduler will however still try varying combined worker sizes to look for the most efficient ones. This can also be specified with the environment variable <code>STARPU_SINGLE_COMBINED_WORKER</code> . (default = 0) |
| char *   | <code>mic_sink_↵<br/>program_path</code>                                       | Path to the kernel to execute on the MIC device, compiled for MIC architecture. When set to NULL, StarPU automatically looks next to the host program location. (default = NULL)                                                                                                                                                                                                                                                                                                                                                                                                        |
| int      | <code>disable_↵<br/>asynchronous↵<br/>_copy</code>                             | This flag should be set to 1 to disable asynchronous copies between CPUs and all accelerators. This can also be specified with the environment variable <code>STARPU_DISABLE_ASYNCHRONOUS_COPY</code> . The A↵<br>MD implementation of OpenCL is known to fail when copying data asynchronously. When using this implementation, it is therefore necessary to disable asynchronous data transfers. This can also be specified at compilation time by giving to the configure script the option <code>--disable-<br/>asynchronous-copy</code> . (default = 0)                            |

|                                        |                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------------------------------|------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int                                    | <code>disable_↵<br/>asynchronous_↵<br/>_cuda_copy</code>   | This flag should be set to 1 to disable asynchronous copies between CPUs and CUDA accelerators. This can also be specified with the environment variable <code>STARPU_DISABLE_ASYNCHRONOUS_CUDA_C↵<br/>OPY</code> . This can also be specified at compilation time by giving to the configure script the option <code>--disable-asynchronous-cuda-copy</code> . (default = 0)                                                                                                                                                                                                  |
| int                                    | <code>disable_↵<br/>asynchronous_↵<br/>_opengl_copy</code> | This flag should be set to 1 to disable asynchronous copies between C↵<br>PUs and OpenGL accelerators. This can also be specified with the environment variable <code>STARPU_DISABLE_ASYNCHRONOUS_OPENGL_↵<br/>COPY</code> . The AMD implementation of OpenGL is known to fail when copying data asynchronously. When using this implementation, it is therefore necessary to disable asynchronous data transfers. This can also be specified at compilation time by giving to the configure script the option <code>--disable-asynchronous-opengl-copy</code> . (default = 0) |
| int                                    | <code>disable_↵<br/>asynchronous_↵<br/>_mic_copy</code>    | This flag should be set to 1 to disable asynchronous copies between CPUs and MIC accelerators. This can also be specified with the environment variable <code>STARPU_DISABLE_ASYNCHRONOUS_MIC_COPY</code> . This can also be specified at compilation time by giving to the configure script the option <code>--disable-asynchronous-mic-copy</code> . (default = 0).                                                                                                                                                                                                          |
| unsigned *                             | <code>cuda_opengl_↵<br/>interoperability</code>            | Enable CUDA/OpenGL interoperability on these CUDA devices. This can be set to an array of CUDA device identifiers for which <code>cudaGLSetGL↵<br/>Device()</code> should be called instead of <code>cudaSetDevice()</code> . Its size is specified by the <code>starpu_conf::n_cuda_opengl_interoperability</code> field below (default = NULL)                                                                                                                                                                                                                               |
| unsigned                               | <code>n_cuda_↵<br/>opengl_↵<br/>interoperability</code>    | todo                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| struct<br><code>starpu_driver *</code> | <code>not_launched_↵<br/>drivers</code>                    | Array of drivers that should not be launched by StarPU. The application will run in one of its own threads. (default = NULL)                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| unsigned                               | <code>n_not_↵<br/>launched_↵<br/>drivers</code>            | The number of StarPU drivers that should not be launched by StarPU. (default = 0)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| unsigned                               | <code>trace_buffer_↵<br/>size</code>                       | Specifies the buffer size used for FxT tracing. Starting from FxT version 0.2.12, the buffer will automatically be flushed when it fills in, but it may still be interesting to specify a bigger value to avoid any flushing (which would disturb the trace).                                                                                                                                                                                                                                                                                                                  |

## 27.2.3 Function Documentation

### 27.2.3.1 `int starpu_init ( struct starpu_conf * conf )`

This is StarPU initialization method, which must be called prior to any other StarPU call. It is possible to specify StarPU's configuration (e.g. scheduling policy, number of cores, ...) by passing a non-null argument. Default configuration is used if the passed argument is NULL. Upon successful completion, this function returns 0. Otherwise, -ENODEV indicates that no worker was available (so that StarPU was not initialized).

### 27.2.3.2 `int starpu_initialize ( struct starpu_conf * user_conf, int * argc, char *** argv )`

This is the same as `starpu_init()`, but also takes the `argc` and `argv` as defined by the application. This is needed for SCC execution to initialize the communication library. Do not call `starpu_init()` and `starpu_initialize()` in the same program.

### 27.2.3.3 `int starpu_conf_init ( struct starpu_conf * conf )`

This function initializes the `conf` structure passed as argument with the default values. In case some configuration parameters are already specified through environment variables, `starpu_conf_init()` initializes the fields of the struc-

ture according to the environment variables. For instance if `STARPU_CALIBRATE` is set, its value is put in the field `starpu_conf::calibrate` of the structure passed as argument. Upon successful completion, this function returns 0. Otherwise, `-EINVAL` indicates that the argument was NULL.

#### 27.2.3.4 `void starpu_shutdown ( void )`

This is StarPU termination method. It must be called at the end of the application: statistics and other post-mortem debugging information are not guaranteed to be available until this method has been called.

#### 27.2.3.5 `void starpu_pause ( void )`

This call is used to suspend the processing of new tasks by workers. It can be used in a program where StarPU is used during only a part of the execution. Without this call, the workers continue to poll for new tasks in a tight loop, wasting CPU time. The symmetric call to `starpu_resume()` should be used to unfreeze the workers.

#### 27.2.3.6 `void starpu_resume ( void )`

This is the symmetrical call to `starpu_pause()`, used to resume the workers polling for new tasks.

#### 27.2.3.7 `int starpu_asynchronous_copy_disabled ( void )`

Return 1 if asynchronous data transfers between CPU and accelerators are disabled.

#### 27.2.3.8 `int starpu_asynchronous_cuda_copy_disabled ( void )`

Return 1 if asynchronous data transfers between CPU and CUDA accelerators are disabled.

#### 27.2.3.9 `int starpu_asynchronous_opengl_copy_disabled ( void )`

Return 1 if asynchronous data transfers between CPU and OpenCL accelerators are disabled.

#### 27.2.3.10 `void starpu_topology_print ( FILE * f )`

Prints a description of the topology on f.

## 27.3 Standard Memory Library

### Macros

- `#define starpu_data_malloc_pinned_if_possible`
- `#define starpu_data_free_pinned_if_possible`
- `#define STARPU_MALLOC_PINNED`
- `#define STARPU_MALLOC_COUNT`
- `#define STARPU_MALLOC_NORECLAIM`

### Functions

- `int starpu_malloc_flags (void **A, size_t dim, int flags)`
- `void starpu_malloc_set_align (size_t align)`
- `int starpu_malloc (void **A, size_t dim)`

- int [starpu\\_free](#) (void \*A)
- int [starpu\\_free\\_flags](#) (void \*A, size\_t dim, int flags)
- starpu\_ssize\_t [starpu\\_memory\\_get\\_total](#) (unsigned node)
- starpu\_ssize\_t [starpu\\_memory\\_get\\_available](#) (unsigned node)
- int [starpu\\_memory\\_allocate](#) (unsigned node, size\_t size, int flags)
- void [starpu\\_memory\\_deallocate](#) (unsigned node, size\_t size)

### 27.3.1 Detailed Description

### 27.3.2 Macro Definition Documentation

#### 27.3.2.1 #define starpu\_data\_malloc\_pinned\_if\_possible

**Deprecated** Equivalent to [starpu\\_malloc\(\)](#). This macro is provided to avoid breaking old codes.

#### 27.3.2.2 #define starpu\_data\_free\_pinned\_if\_possible

**Deprecated** Equivalent to [starpu\\_free\(\)](#). This macro is provided to avoid breaking old codes.

#### 27.3.2.3 #define STARPU\_MALLOC\_PINNED

Value passed to the function [starpu\\_malloc\\_flags\(\)](#) to indicate the memory allocation should be pinned.

#### 27.3.2.4 #define STARPU\_MALLOC\_COUNT

Value passed to the function [starpu\\_malloc\\_flags\(\)](#) to indicate the memory allocation should be in the limit defined by the environment variables [STARPU\\_LIMIT\\_CUDA\\_devid\\_MEM](#), [STARPU\\_LIMIT\\_CUDA\\_MEM](#), [STARPU\\_LIMIT\\_OPENCL\\_devid\\_MEM](#), [STARPU\\_LIMIT\\_OPENCL\\_MEM](#) and [STARPU\\_LIMIT\\_CPU\\_MEM](#) (see Section [How to limit memory per node](#)). If no memory is available, it tries to reclaim memory from StarPU. Memory allocated this way needs to be freed by calling the function [starpu\\_free\\_flags\(\)](#) with the same flag.

#### 27.3.2.5 #define STARPU\_MALLOC\_NORECLAIM

Value passed to the function [starpu\\_malloc\\_flags\(\)](#) along [STARPU\\_MALLOC\\_COUNT](#) to indicate that while the memory allocation should be kept in the limits defined for [STARPU\\_MALLOC\\_COUNT](#), no reclaiming should be performed by [starpu\\_malloc\\_flags](#) itself, thus potentially overflowing the memory node a bit. StarPU will reclaim memory after next task termination, according to the [STARPU\\_MINIMUM\\_AVAILABLE\\_MEM](#) and [STARPU\\_TARGET\\_AVAILABLE\\_MEM](#) environment variables.

### 27.3.3 Function Documentation

#### 27.3.3.1 int starpu\_malloc\_flags ( void \*\* A, size\_t dim, int flags )

Performs a memory allocation based on the constraints defined by the given flag.

#### 27.3.3.2 void starpu\_malloc\_set\_align ( size\_t align )

This function sets an alignment constraints for [starpu\\_malloc\(\)](#) allocations. align must be a power of two. This is for instance called automatically by the OpenCL driver to specify its own alignment constraints.

### 27.3.3.3 `int starpu_malloc ( void ** A, size_t dim )`

This function allocates data of the given size in main memory. It will also try to pin it in CUDA or OpenCL, so that data transfers from this buffer can be asynchronous, and thus permit data transfer and computation overlapping. The allocated buffer must be freed thanks to the [starpu\\_free\(\)](#) function.

### 27.3.3.4 `int starpu_free ( void * A )`

This function frees memory which has previously been allocated with [starpu\\_malloc\(\)](#).

### 27.3.3.5 `int starpu_free_flags ( void * A, size_t dim, int flags )`

This function frees memory by specifying its size. The given flags should be consistent with the ones given to [starpu\\_malloc\\_flags\(\)](#) when allocating the memory.

### 27.3.3.6 `ssize_t starpu_memory_get_total ( unsigned node )`

If a memory limit is defined on the given node (see Section [How to limit memory per node](#)), return the amount of total memory on the node. Otherwise return -1.

### 27.3.3.7 `ssize_t starpu_memory_get_available ( unsigned node )`

If a memory limit is defined on the given node (see Section [How to limit memory per node](#)), return the amount of available memory on the node. Otherwise return -1.

### 27.3.3.8 `int starpu_memory_allocate ( unsigned node, size_t size, int flags )`

Try to allocate memory on the given node

#### Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>size</i> | amount of memory to allocate             |
| <i>node</i> | node where the memory is to be allocated |

#### Returns

1 if the given amount of memory was allocated on the given node

If a memory limit is defined on the given node (see Section [How to limit memory per node](#)), try to allocate some of it. This does not actually allocate memory, but only accounts for it. This can be useful when the application allocates data another way, but want StarPU to be aware of the allocation size e.g. for memory reclaiming. By default, the function returns -ENOMEM if there is not enough room on the given node. *flags* can be either STARPU\_MEMORY\_MANAGER\_WAIT or STARPU\_MEMORY\_MANAGER\_OVERFLOW to change this.

### 27.3.3.9 `int starpu_memory_deallocate ( unsigned node, size_t size )`

Indicates the given amount of memory is going to be deallocated from the given node

#### Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>size</i> | amount of memory to be deallocated |
|-------------|------------------------------------|

|             |                                                  |
|-------------|--------------------------------------------------|
| <i>node</i> | node where the memory is going to be deallocated |
|-------------|--------------------------------------------------|

If a memory limit is defined on the given node (see Section [How to limit memory per node](#)), free some of it. This does not actually free memory, but only accounts for it, like `starpu_memory_allocate()`. The amount does not have to be exactly the same as what was passed to `starpu_memory_allocate()`, only the eventual amount needs to be the same, i.e. one call to `starpu_memory_allocate()` can be followed by several calls to `starpu_memory_deallocate()` to declare the deallocation piece by piece.

## 27.4 Toolbox

The following macros allow to make GCC extensions portable, and to have a code which can be compiled with any C compiler.

### Macros

- `#define STARPU_GNUC_PREREQ(maj, min)`
- `#define STARPU_UNLIKELY(expr)`
- `#define STARPU_LIKELY(expr)`
- `#define STARPU_ATTRIBUTE_UNUSED`
- `#define STARPU_ATTRIBUTE_INTERNAL`
- `#define STARPU_ATTRIBUTE_MALLOC`
- `#define STARPU_ATTRIBUTE_WARN_UNUSED_RESULT`
- `#define STARPU_ATTRIBUTE_PURE`
- `#define STARPU_ATTRIBUTE_ALIGNED(size)`
- `#define STARPU_WARN_UNUSED_RESULT`
- `#define STARPU_POISON_PTR`
- `#define STARPU_MIN(a, b)`
- `#define STARPU_MAX(a, b)`
- `#define STARPU_ASSERT(x)`
- `#define STARPU_ASSERT_MSG(x, msg,...)`
- `#define STARPU_ABORT()`
- `#define STARPU_ABORT_MSG(msg,...)`
- `#define STARPU_CHECK_RETURN_VALUE(err, message,...)`
- `#define STARPU_CHECK_RETURN_VALUE_IS(err, value, message,...)`
- `#define STARPU_RMB()`
- `#define STARPU_WMB()`

### Functions

- `static __starpu_inline int starpu_get_env_number (const char *str)`

#### 27.4.1 Detailed Description

The following macros allow to make GCC extensions portable, and to have a code which can be compiled with any C compiler.

#### 27.4.2 Macro Definition Documentation

##### 27.4.2.1 `#define STARPU_GNUC_PREREQ( maj, min )`

Return true (non-zero) if GCC version MAJ.MIN or later is being used (macro taken from glibc.)

#### 27.4.2.2 `#define STARPU_UNLIKELY( expr )`

When building with a GNU C Compiler, this macro allows programmers to mark an expression as unlikely.

#### 27.4.2.3 `#define STARPU_LIKELY( expr )`

When building with a GNU C Compiler, this macro allows programmers to mark an expression as likely.

#### 27.4.2.4 `#define STARPU_ATTRIBUTE_UNUSED`

When building with a GNU C Compiler, this macro is defined to `__attribute__((unused))`

#### 27.4.2.5 `#define STARPU_ATTRIBUTE_INTERNAL`

When building with a GNU C Compiler, this macro is defined to `__attribute__((visibility ("internal")))`

#### 27.4.2.6 `#define STARPU_ATTRIBUTE_MALLOC`

When building with a GNU C Compiler, this macro is defined to `__attribute__((malloc))`

#### 27.4.2.7 `#define STARPU_ATTRIBUTE_WARN_UNUSED_RESULT`

When building with a GNU C Compiler, this macro is defined to `__attribute__((warn_unused_result))`

#### 27.4.2.8 `#define STARPU_ATTRIBUTE_PURE`

When building with a GNU C Compiler, this macro is defined to `__attribute__((pure))`

#### 27.4.2.9 `#define STARPU_ATTRIBUTE_ALIGNED( size )`

When building with a GNU C Compiler, this macro is defined to `__attribute__((aligned(size)))`

#### 27.4.2.10 `#define STARPU_WARN_UNUSED_RESULT`

When building with a GNU C Compiler, this macro is defined to `__attribute__((__warn_unused_result__))`

#### 27.4.2.11 `#define STARPU_POISON_PTR`

This macro defines a value which can be used to mark pointers as invalid values.

#### 27.4.2.12 `#define STARPU_MIN( a, b )`

This macro returns the min of the two parameters.

#### 27.4.2.13 `#define STARPU_MAX( a, b )`

This macro returns the max of the two parameters.



#### 27.4.2.14 `#define STARPU_ASSERT( x )`

Unless StarPU has been configured with the option `--enable-fast`, this macro will abort if the expression is false.

#### 27.4.2.15 `#define STARPU_ASSERT_MSG( x, msg, ... )`

Unless StarPU has been configured with the option `--enable-fast`, this macro will abort if the expression is false. The given message will be displayed.

#### 27.4.2.16 `#define STARPU_ABORT( )`

This macro aborts the program.

#### 27.4.2.17 `#define STARPU_ABORT_MSG( msg, ... )`

This macro aborts the program, and displays the given message.

#### 27.4.2.18 `#define STARPU_CHECK_RETURN_VALUE( err, message, ... )`

If `err` has a value which is not 0, the given message is displayed before aborting.

#### 27.4.2.19 `#define STARPU_CHECK_RETURN_VALUE_IS( err, value, message, ... )`

If `err` has a value which is not `value`, the given message is displayed before aborting.

#### 27.4.2.20 `#define STARPU_RMB( )`

This macro can be used to do a synchronization.

#### 27.4.2.21 `#define STARPU_WMB( )`

This macro can be used to do a synchronization.

### 27.4.3 Function Documentation

#### 27.4.3.1 `int starpu_get_env_number( const char * str ) [static]`

If `str` is the name of a existing environment variable which is defined to an integer, the function returns the value of the integer. It returns 0 otherwise.

## 27.5 Threads

This section describes the thread facilities provided by StarPU. The thread function are either implemented on top of the pthread library or the Simgrid library when the simulated performance mode is enabled ([SimGrid Support](#)).

### Macros

- `#define STARPU_PTHREAD_CREATE_ON(name, thread, attr, routine, arg, where)`
- `#define STARPU_PTHREAD_CREATE(thread, attr, routine, arg)`

- `#define STARPU_PTHREAD_MUTEX_INIT(mutex, attr)`
- `#define STARPU_PTHREAD_MUTEX_DESTROY(mutex)`
- `#define STARPU_PTHREAD_MUTEX_LOCK(mutex)`
- `#define STARPU_PTHREAD_MUTEX_UNLOCK(mutex)`
- `#define STARPU_PTHREAD_KEY_CREATE(key, destr)`
- `#define STARPU_PTHREAD_KEY_DELETE(key)`
- `#define STARPU_PTHREAD_SETSPECIFIC(key, ptr)`
- `#define STARPU_PTHREAD_GETSPECIFIC(key)`
- `#define STARPU_PTHREAD_RWLOCK_INIT(rwlock, attr)`
- `#define STARPU_PTHREAD_RWLOCK_RDLOCK(rwlock)`
- `#define STARPU_PTHREAD_RWLOCK_WRLOCK(rwlock)`
- `#define STARPU_PTHREAD_RWLOCK_UNLOCK(rwlock)`
- `#define STARPU_PTHREAD_RWLOCK_DESTROY(rwlock)`
- `#define STARPU_PTHREAD_COND_INIT(cond, attr)`
- `#define STARPU_PTHREAD_COND_DESTROY(cond)`
- `#define STARPU_PTHREAD_COND_SIGNAL(cond)`
- `#define STARPU_PTHREAD_COND_BROADCAST(cond)`
- `#define STARPU_PTHREAD_COND_WAIT(cond, mutex)`
- `#define STARPU_PTHREAD_BARRIER_INIT(barrier, attr, count)`
- `#define STARPU_PTHREAD_BARRIER_DESTROY(barrier)`
- `#define STARPU_PTHREAD_BARRIER_WAIT(barrier)`
- `#define STARPU_PTHREAD_MUTEX_INITIALIZER`
- `#define STARPU_PTHREAD_COND_INITIALIZER`

## Functions

- `int starpu_thread_create_on (char *name, starpu_thread_t *thread, const starpu_thread_attr_t *attr, void *(*start_routine)(void *), void *arg, int where)`
- `int starpu_thread_create (starpu_thread_t *thread, const starpu_thread_attr_t *attr, void *(*start_routine)(void *), void *arg)`
- `int starpu_thread_join (starpu_thread_t thread, void **retval)`
- `int starpu_thread_exit (void *retval)`
- `int starpu_thread_attr_init (starpu_thread_attr_t *attr)`
- `int starpu_thread_attr_destroy (starpu_thread_attr_t *attr)`
- `int starpu_thread_attr_setdetachstate (starpu_thread_attr_t *attr, int detachstate)`
- `int starpu_thread_mutex_init (starpu_thread_mutex_t *mutex, const starpu_thread_mutexattr_t *mutexattr)`
- `int starpu_thread_mutex_destroy (starpu_thread_mutex_t *mutex)`
- `int starpu_thread_mutex_lock (starpu_thread_mutex_t *mutex)`
- `int starpu_thread_mutex_unlock (starpu_thread_mutex_t *mutex)`
- `int starpu_thread_mutex_trylock (starpu_thread_mutex_t *mutex)`
- `int starpu_thread_mutexattr_gettype (const starpu_thread_mutexattr_t *attr, int *type)`
- `int starpu_thread_mutexattr_settype (starpu_thread_mutexattr_t *attr, int type)`
- `int starpu_thread_mutexattr_destroy (starpu_thread_mutexattr_t *attr)`
- `int starpu_thread_mutexattr_init (starpu_thread_mutexattr_t *attr)`
- `int starpu_thread_key_create (starpu_thread_key_t *key, void(*destr_function)(void *))`
- `int starpu_thread_key_delete (starpu_thread_key_t key)`
- `int starpu_thread_setspecific (starpu_thread_key_t key, const void *pointer)`
- `void * starpu_thread_getspecific (starpu_thread_key_t key)`
- `int starpu_thread_cond_init (starpu_thread_cond_t *cond, starpu_thread_condattr_t *cond_attr)`
- `int starpu_thread_cond_signal (starpu_thread_cond_t *cond)`
- `int starpu_thread_cond_broadcast (starpu_thread_cond_t *cond)`
- `int starpu_thread_cond_wait (starpu_thread_cond_t *cond, starpu_thread_mutex_t *mutex)`

- `int starpu_pthread_cond_timedwait` (`starpu_pthread_cond_t *cond`, `starpu_pthread_mutex_t *mutex`, `const struct timespec *abstime`)
- `int starpu_pthread_cond_destroy` (`starpu_pthread_cond_t *cond`)
- `int starpu_pthread_rwlock_init` (`starpu_pthread_rwlock_t *rwlock`, `const starpu_pthread_rwlockattr_t *attr`)
- `int starpu_pthread_rwlock_destroy` (`starpu_pthread_rwlock_t *rwlock`)
- `int starpu_pthread_rwlock_rdlock` (`starpu_pthread_rwlock_t *rwlock`)
- `int starpu_pthread_rwlock_tryrdlock` (`starpu_pthread_rwlock_t *rwlock`)
- `int starpu_pthread_rwlock_wrlock` (`starpu_pthread_rwlock_t *rwlock`)
- `int starpu_pthread_rwlock_trywrlock` (`starpu_pthread_rwlock_t *rwlock`)
- `int starpu_pthread_rwlock_unlock` (`starpu_pthread_rwlock_t *rwlock`)
- `int starpu_pthread_barrier_init` (`starpu_pthread_barrier_t *barrier`, `const starpu_pthread_barrierattr_t *attr`, `unsigned count`)
- `int starpu_pthread_barrier_destroy` (`starpu_pthread_barrier_t *barrier`)
- `int starpu_pthread_barrier_wait` (`starpu_pthread_barrier_t *barrier`)
- `int starpu_pthread_spin_init` (`starpu_pthread_spinlock_t *lock`, `int pshared`)
- `int starpu_pthread_spin_destroy` (`starpu_pthread_spinlock_t *lock`)
- `int starpu_pthread_spin_lock` (`starpu_pthread_spinlock_t *lock`)
- `int starpu_pthread_spin_trylock` (`starpu_pthread_spinlock_t *lock`)
- `int starpu_pthread_spin_unlock` (`starpu_pthread_spinlock_t *lock`)

### 27.5.1 Detailed Description

This section describes the thread facilities provided by StarPU. The thread function are either implemented on top of the pthread library or the Simgrid library when the simulated performance mode is enabled ([SimGrid Support](#)).

### 27.5.2 Macro Definition Documentation

#### 27.5.2.1 `#define STARPU_PTHREAD_CREATE_ON( name, thread, attr, routine, arg, where )`

This macro calls the function `starpu_pthread_create_on()` and aborts on error.

#### 27.5.2.2 `#define STARPU_PTHREAD_CREATE( thread, attr, routine, arg )`

This macro calls the function `starpu_pthread_create()` and aborts on error.

#### 27.5.2.3 `#define STARPU_PTHREAD_MUTEX_INIT( mutex, attr )`

This macro calls the function `starpu_pthread_mutex_init()` and aborts on error.

#### 27.5.2.4 `#define STARPU_PTHREAD_MUTEX_DESTROY( mutex )`

This macro calls the function `starpu_pthread_mutex_destroy()` and aborts on error.

#### 27.5.2.5 `#define STARPU_PTHREAD_MUTEX_LOCK( mutex )`

This macro calls the function `starpu_pthread_mutex_lock()` and aborts on error.

#### 27.5.2.6 `#define STARPU_PTHREAD_MUTEX_UNLOCK( mutex )`

This macro calls the function `starpu_pthread_mutex_unlock()` and aborts on error.

**27.5.2.7 #define STARPU\_PTHREAD\_KEY\_CREATE( *key*, *destr* )**

This macro calls the function [starpu\\_thread\\_key\\_create\(\)](#) and aborts on error.

**27.5.2.8 #define STARPU\_PTHREAD\_KEY\_DELETE( *key* )**

This macro calls the function [starpu\\_thread\\_key\\_delete\(\)](#) and aborts on error.

**27.5.2.9 #define STARPU\_PTHREAD\_SETSPECIFIC( *key*, *ptr* )**

This macro calls the function [starpu\\_thread\\_setspecific\(\)](#) and aborts on error.

**27.5.2.10 #define STARPU\_PTHREAD\_GETSPECIFIC( *key* )**

This macro calls the function [starpu\\_thread\\_getspecific\(\)](#) and aborts on error.

**27.5.2.11 #define STARPU\_PTHREAD\_RWLOCK\_INIT( *rwlock*, *attr* )**

This macro calls the function [starpu\\_thread\\_rwlock\\_init\(\)](#) and aborts on error.

**27.5.2.12 #define STARPU\_PTHREAD\_RWLOCK\_RDLOCK( *rwlock* )**

This macro calls the function [starpu\\_thread\\_rwlock\\_rdlock\(\)](#) and aborts on error.

**27.5.2.13 #define STARPU\_PTHREAD\_RWLOCK\_WRLOCK( *rwlock* )**

This macro calls the function [starpu\\_thread\\_rwlock\\_wrlock\(\)](#) and aborts on error.

**27.5.2.14 #define STARPU\_PTHREAD\_RWLOCK\_UNLOCK( *rwlock* )**

This macro calls the function [starpu\\_thread\\_rwlock\\_unlock\(\)](#) and aborts on error.

**27.5.2.15 #define STARPU\_PTHREAD\_RWLOCK\_DESTROY( *rwlock* )**

This macro calls the function [starpu\\_thread\\_rwlock\\_destroy\(\)](#) and aborts on error.

**27.5.2.16 #define STARPU\_PTHREAD\_COND\_INIT( *cond*, *attr* )**

This macro calls the function [starpu\\_thread\\_cond\\_init\(\)](#) and aborts on error.

**27.5.2.17 #define STARPU\_PTHREAD\_COND\_DESTROY( *cond* )**

This macro calls the function [starpu\\_thread\\_cond\\_destroy\(\)](#) and aborts on error.

**27.5.2.18 #define STARPU\_PTHREAD\_COND\_SIGNAL( *cond* )**

This macro calls the function [starpu\\_thread\\_cond\\_signal\(\)](#) and aborts on error.

#### 27.5.2.19 `#define STARPU_PTHREAD_COND_BROADCAST( cond )`

This macro calls the function `starpu_pthread_cond_broadcast()` and aborts on error.

#### 27.5.2.20 `#define STARPU_PTHREAD_COND_WAIT( cond, mutex )`

This macro calls the function `starpu_pthread_cond_wait()` and aborts on error.

#### 27.5.2.21 `#define STARPU_PTHREAD_BARRIER_INIT( barrier, attr, count )`

This macro calls the function `starpu_pthread_barrier_init()` and aborts on error.

#### 27.5.2.22 `#define STARPU_PTHREAD_BARRIER_DESTROY( barrier )`

This macro calls the function `starpu_pthread_barrier_destroy()` and aborts on error.

#### 27.5.2.23 `#define STARPU_PTHREAD_BARRIER_WAIT( barrier )`

This macro calls the function `starpu_pthread_barrier_wait()` and aborts on error.

#### 27.5.2.24 `STARPU_PTHREAD_MUTEX_INITIALIZER`

This macro initializes the mutex given in parameter.

#### 27.5.2.25 `STARPU_PTHREAD_COND_INITIALIZER`

This macro initializes the condition variable given in parameter.

### 27.5.3 Function Documentation

#### 27.5.3.1 `int starpu_pthread_create ( starpu_pthread_t * thread, const starpu_pthread_attr_t * attr, void (*)(void *) start_routine, void * arg )`

This function starts a new thread in the calling process. The new thread starts execution by invoking `start_routine`; `arg` is passed as the sole argument of `start_routine`.

#### 27.5.3.2 `int starpu_pthread_join ( starpu_pthread_t thread, void ** retval )`

This function waits for the thread specified by `thread` to terminate. If that thread has already terminated, then the function returns immediately. The thread specified by `thread` must be joinable.

#### 27.5.3.3 `int starpu_pthread_exit ( void * retval )`

This function terminates the calling thread and returns a value via `retval` that (if the thread is joinable) is available to another thread in the same process that calls `starpu_pthread_join()`.

#### 27.5.3.4 `int starpu_pthread_attr_init ( starpu_pthread_attr_t * attr )`

This function initializes the thread attributes object pointed to by `attr` with default attribute values.

It does not do anything when the simulated performance mode is enabled ([SimGrid Support](#)).

### 27.5.3.5 `int starpu_pthread_attr_destroy ( starpu_pthread_attr_t * attr )`

This function destroys a thread attributes object which is no longer required. Destroying a thread attributes object has no effect on threads that were created using that object.

It does not do anything when the simulated performance mode is enabled ([SimGrid Support](#)).

### 27.5.3.6 `int starpu_pthread_attr_setdetachstate ( starpu_pthread_attr_t * attr, int detachstate )`

This function sets the detach state attribute of the thread attributes object referred to by `attr` to the value specified in `detachstate`. The detach state attribute determines whether a thread created using the thread attributes object `attr` will be created in a joinable or a detached state.

It does not do anything when the simulated performance mode is enabled ([SimGrid Support](#)).

### 27.5.3.7 `int starpu_pthread_mutex_init ( starpu_pthread_mutex_t * mutex, const starpu_pthread_mutexattr_t * mutexattr )`

This function initializes the mutex object pointed to by `mutex` according to the mutex attributes specified in `mutexattr`. If `mutexattr` is NULL, default attributes are used instead.

### 27.5.3.8 `int starpu_pthread_mutex_destroy ( starpu_pthread_mutex_t * mutex )`

This function destroys a mutex object, freeing the resources it might hold. The mutex must be unlocked on entrance.

### 27.5.3.9 `int starpu_pthread_mutex_lock ( starpu_pthread_mutex_t * mutex )`

This function locks the given mutex. If the mutex is currently unlocked, it becomes locked and owned by the calling thread, and the function returns immediately. If the mutex is already locked by another thread, the function suspends the calling thread until the mutex is unlocked.

This function also produces trace when the configure option `--enable-fxt-lock` is enabled.

### 27.5.3.10 `int starpu_pthread_mutex_unlock ( starpu_pthread_mutex_t * mutex )`

This function unlocks the given mutex. The mutex is assumed to be locked and owned by the calling thread on entrance to `starpu_pthread_mutex_unlock()`.

This function also produces trace when the configure option `--enable-fxt-lock` is enabled.

### 27.5.3.11 `int starpu_pthread_mutex_trylock ( starpu_pthread_mutex_t * mutex )`

This function behaves identically to `starpu_pthread_mutex_lock()`, except that it does not block the calling thread if the mutex is already locked by another thread (or by the calling thread in the case of a “fast” mutex). Instead, the function returns immediately with the error code EBUSY.

This function also produces trace when the configure option `--enable-fxt-lock` is enabled.

### 27.5.3.12 `int starpu_pthread_mutexattr_gettype ( const starpu_pthread_mutexattr_t * attr, int * type )`

todo

### 27.5.3.13 `int starpu_pthread_mutexattr_settype ( starpu_pthread_mutexattr_t * attr, int type )`

todo

27.5.3.14 `int starpu_thread_mutexattr_destroy ( starpu_thread_mutexattr_t * attr )`

todo

27.5.3.15 `int starpu_thread_mutexattr_init ( starpu_thread_mutexattr_t * attr )`

todo

27.5.3.16 `int starpu_thread_key_create ( starpu_thread_key_t * key, void(*) (void *) destr_function )`

This function allocates a new TSD key. The key is stored in the location pointed to by `key`.

27.5.3.17 `int starpu_thread_key_delete ( starpu_thread_key_t key )`

This function deallocates a TSD key. It does not check whether non-NULL values are associated with that key in the currently executing threads, nor call the destructor function associated with the key.

27.5.3.18 `int starpu_thread_setspecific ( starpu_thread_key_t key, const void * pointer )`

This function changes the value associated with `key` in the calling thread, storing the given `pointer` instead.

27.5.3.19 `* starpu_thread_getspecific ( starpu_thread_key_t key )`

This function returns the value associated with `key` on success, and NULL on error.

27.5.3.20 `starpu_thread_cond_init ( starpu_thread_cond_t * cond, starpu_thread_condattr_t * cond_attr )`

This function initializes the condition variable `cond`, using the condition attributes specified in `cond_attr`, or default attributes if `cond_attr` is NULL.

27.5.3.21 `starpu_thread_cond_signal ( starpu_thread_cond_t * cond )`

This function restarts one of the threads that are waiting on the condition variable `cond`. If no threads are waiting on `cond`, nothing happens. If several threads are waiting on `cond`, exactly one is restarted, but it not specified which.

27.5.3.22 `starpu_thread_cond_broadcast ( starpu_thread_cond_t * cond )`

This function restarts all the threads that are waiting on the condition variable `cond`. Nothing happens if no threads are waiting on `cond`.

27.5.3.23 `starpu_thread_cond_wait ( starpu_thread_cond_t * cond, starpu_thread_mutex_t * mutex )`

This function atomically unlocks the mutex (as per [starpu\\_thread\\_mutex\\_unlock\(\)](#)) and waits for the condition variable `cond` to be signaled. The thread execution is suspended and does not consume any CPU time until the condition variable is signaled. The mutex must be locked by the calling thread on entrance to [starpu\\_thread\\_cond\\_wait\(\)](#). Before returning to the calling thread, the function re-acquires mutex (as per [starpu\\_thread\\_mutex\\_lock\(\)](#)).

This function also produces trace when the configure option `--enable-fxt-lock` is enabled.

**27.5.3.24** `starpu_thread_cond_timedwait ( starpu_thread_cond_t * cond, starpu_thread_mutex_t * mutex, const struct timespec * abstime )`

This function atomically unlocks `mutex` and waits on `cond`, as `starpu_thread_cond_wait()` does, but it also bounds the duration of the wait.

**27.5.3.25** `starpu_thread_cond_destroy ( starpu_thread_cond_t * cond )`

This function destroys a condition variable, freeing the resources it might hold. No threads must be waiting on the condition variable on entrance to the function.

**27.5.3.26** `starpu_thread_rwlock_init ( starpu_thread_rwlock_t * rwlock, const starpu_thread_rwlockattr_t * attr )`

This function is the same as `starpu_thread_mutex_init()`.

**27.5.3.27** `starpu_thread_rwlock_destroy ( starpu_thread_rwlock_t * rwlock )`

This function is the same as `starpu_thread_mutex_destroy()`.

**27.5.3.28** `starpu_thread_rwlock_rdlock ( starpu_thread_rwlock_t * rwlock )`

This function is the same as `starpu_thread_mutex_lock()`.

**27.5.3.29** `int starpu_thread_rwlock_tryrdlock ( starpu_thread_rwlock_t * rwlock )`

todo

**27.5.3.30** `starpu_thread_rwlock_wrlock ( starpu_thread_rwlock_t * rwlock )`

This function is the same as `starpu_thread_mutex_lock()`.

**27.5.3.31** `int starpu_thread_rwlock_trywrlock ( starpu_thread_rwlock_t * rwlock )`

todo

**27.5.3.32** `starpu_thread_rwlock_unlock ( starpu_thread_rwlock_t * rwlock )`

This function is the same as `starpu_thread_mutex_unlock()`.

**27.5.3.33** `int starpu_thread_barrier_init ( starpu_thread_barrier_t * barrier, const starpu_thread_barrierattr_t * attr, unsigned count )`

todo

**27.5.3.34** `int starpu_thread_barrier_destroy ( starpu_thread_barrier_t * barrier )`

todo



27.5.3.35 `int starpu_thread_barrier_wait ( starpu_thread_barrier_t * barrier )`

todo

27.5.3.36 `int starpu_thread_spin_init ( starpu_thread_spinlock_t * lock, int pshared )`

todo

27.5.3.37 `int starpu_thread_spin_destroy ( starpu_thread_spinlock_t * lock )`

todo

27.5.3.38 `int starpu_thread_spin_lock ( starpu_thread_spinlock_t * lock )`

todo

27.5.3.39 `int starpu_thread_spin_trylock ( starpu_thread_spinlock_t * lock )`

todo

27.5.3.40 `int starpu_thread_spin_unlock ( starpu_thread_spinlock_t * lock )`

todo

## 27.6 Bitmap

This section describes the bitmap facilities provided by StarPU.

### Functions

- `struct starpu_bitmap * starpu_bitmap_create` (void)
- `void starpu_bitmap_destroy` (struct starpu\_bitmap \*b)
- `void starpu_bitmap_set` (struct starpu\_bitmap \*b, int e)
- `void starpu_bitmap_unset` (struct starpu\_bitmap \*b, int e)
- `void starpu_bitmap_unset_all` (struct starpu\_bitmap \*b)
- `int starpu_bitmap_get` (struct starpu\_bitmap \*b, int e)
- `void starpu_bitmap_unset_and` (struct starpu\_bitmap \*a, struct starpu\_bitmap \*b, struct starpu\_bitmap \*c)
- `void starpu_bitmap_or` (struct starpu\_bitmap \*a, struct starpu\_bitmap \*b)
- `int starpu_bitmap_and_get` (struct starpu\_bitmap \*b1, struct starpu\_bitmap \*b2, int e)
- `int starpu_bitmap_cardinal` (struct starpu\_bitmap \*b)
- `int starpu_bitmap_first` (struct starpu\_bitmap \*b)
- `int starpu_bitmap_last` (struct starpu\_bitmap \*b)
- `int starpu_bitmap_next` (struct starpu\_bitmap \*b, int e)
- `int starpu_bitmap_has_next` (struct starpu\_bitmap \*b, int e)

### 27.6.1 Detailed Description

This section describes the bitmap facilities provided by StarPU.

## 27.6.2 Function Documentation

### 27.6.2.1 `struct starpu_bitmap * starpu_bitmap_create ( void )`

create a empty `starpu_bitmap`

### 27.6.2.2 `void starpu_bitmap_destroy ( struct starpu_bitmap * b )`

free a `starpu_bitmap`

### 27.6.2.3 `void starpu_bitmap_set ( struct starpu_bitmap * b, int e )`

set bit *e* in *b*

### 27.6.2.4 `void starpu_bitmap_unset ( struct starpu_bitmap * b, int e )`

unset bit *e* in *b*

### 27.6.2.5 `void starpu_bitmap_unset_all ( struct starpu_bitmap * b )`

unset all bits in *b*

### 27.6.2.6 `int starpu_bitmap_get ( struct starpu_bitmap * b, int e )`

return true iff bit *e* is set in *b*

### 27.6.2.7 `void starpu_bitmap_unset_and ( struct starpu_bitmap * a, struct starpu_bitmap * b, struct starpu_bitmap * c )`

Basically compute `starpu_bitmap_unset_all(a)` ;  $a = b \& c$ ;

### 27.6.2.8 `void starpu_bitmap_or ( struct starpu_bitmap * a, struct starpu_bitmap * b )`

Basically compute  $a \mid= b$

### 27.6.2.9 `int starpu_bitmap_and_get ( struct starpu_bitmap * b1, struct starpu_bitmap * b2, int e )`

return 1 iff *e* set in *b1* AND *e* set in *b2*

### 27.6.2.10 `int starpu_bitmap_cardinal ( struct starpu_bitmap * b )`

return the number of set bits in *b*

### 27.6.2.11 `int starpu_bitmap_first ( struct starpu_bitmap * b )`

return the index of first set bit of *b*, -1 if none

### 27.6.2.12 `int starpu_bitmap_last ( struct starpu_bitmap * b )`

return the position of the last set bit of *b*, -1 if none

27.6.2.13 `int starpu_bitmap_next ( struct starpu_bitmap * b, int e )`

return the position of set bit right after `e` in `b`, -1 if none

27.6.2.14 `int starpu_bitmap_has_next ( struct starpu_bitmap * b, int e )`

todo

## 27.7 Workers' Properties

### Data Structures

- struct [starpu\\_worker\\_collection](#)
- struct [starpu\\_sched\\_ctx\\_iterator](#)

### Macros

- `#define STARPU_NMAXWORKERS`

### Enumerations

- enum [starpu\\_node\\_kind](#) {  
[STARPU\\_UNUSED](#), [STARPU\\_CPU\\_RAM](#), [STARPU\\_CUDA\\_RAM](#), [STARPU\\_OPENCL\\_RAM](#),  
[STARPU\\_DISK\\_RAM](#), [STARPU\\_MIC\\_RAM](#), [STARPU\\_SCC\\_RAM](#), [STARPU\\_SCC\\_SHM](#) }
- enum [starpu\\_worker\\_archtype](#) {  
[STARPU\\_CPU\\_WORKER](#), [STARPU\\_CUDA\\_WORKER](#), [STARPU\\_OPENCL\\_WORKER](#), [STARPU\\_MIC\\_↵  
WORKER](#),  
[STARPU\\_SCC\\_WORKER](#), [STARPU\\_ANY\\_WORKER](#) }
- enum [starpu\\_worker\\_collection\\_type](#) { [STARPU\\_WORKER\\_TREE](#), [STARPU\\_WORKER\\_LIST](#) }

### Functions

- unsigned [starpu\\_worker\\_get\\_count](#) (void)
- int [starpu\\_worker\\_get\\_count\\_by\\_type](#) (enum [starpu\\_worker\\_archtype](#) type)
- unsigned [starpu\\_cpu\\_worker\\_get\\_count](#) (void)
- unsigned [starpu\\_cuda\\_worker\\_get\\_count](#) (void)
- unsigned [starpu\\_mic\\_worker\\_get\\_count](#) (void)
- unsigned [starpu\\_mic\\_device\\_get\\_count](#) (void)
- unsigned [starpu\\_scc\\_worker\\_get\\_count](#) (void)
- unsigned [starpu\\_opencl\\_worker\\_get\\_count](#) (void)
- int [starpu\\_worker\\_get\\_id](#) (void)
- int [starpu\\_worker\\_get\\_ids\\_by\\_type](#) (enum [starpu\\_worker\\_archtype](#) type, int \*workerids, int maxsize)
- int [starpu\\_worker\\_get\\_by\\_type](#) (enum [starpu\\_worker\\_archtype](#) type, int num)
- int [starpu\\_worker\\_get\\_by\\_devid](#) (enum [starpu\\_worker\\_archtype](#) type, int devid)
- int [starpu\\_worker\\_get\\_devid](#) (int id)
- enum [starpu\\_worker\\_archtype](#) [starpu\\_worker\\_get\\_type](#) (int id)
- void [starpu\\_worker\\_get\\_name](#) (int id, char \*dst, size\_t maxlen)
- unsigned [starpu\\_worker\\_get\\_memory\\_node](#) (unsigned workerid)
- enum [starpu\\_node\\_kind](#) [starpu\\_node\\_get\\_kind](#) (unsigned node)
- char \* [starpu\\_worker\\_get\\_type\\_as\\_string](#) (enum [starpu\\_worker\\_archtype](#) type)

### 27.7.1 Detailed Description

### 27.7.2 Data Structure Documentation

#### 27.7.2.1 struct starpu\_worker\_collection

A scheduling context manages a collection of workers that can be memorized using different data structures. Thus, a generic structure is available in order to simplify the choice of its type. Only the list data structure is available but further data structures (like tree) implementations are foreseen.

#### Data Fields

- void \* [workerids](#)
- unsigned [nworkers](#)
- void \* **masters**
- unsigned **nmasters**
- int **present** [[STARPU\\_NMAXWORKERS](#)]
- int **is\_master** [[STARPU\\_NMAXWORKERS](#)]
- enum [starpu\\_worker\\_collection\\_type](#) type
- unsigned(\* [has\\_next](#) )(struct [starpu\\_worker\\_collection](#) \*workers, struct [starpu\\_sched\\_ctx\\_iterator](#) \*it)
- int(\* [get\\_next](#) )(struct [starpu\\_worker\\_collection](#) \*workers, struct [starpu\\_sched\\_ctx\\_iterator](#) \*it)
- unsigned(\* [has\\_next\\_master](#) )(struct [starpu\\_worker\\_collection](#) \*workers, struct [starpu\\_sched\\_ctx\\_iterator](#) \*it)
- int(\* [get\\_next\\_master](#) )(struct [starpu\\_worker\\_collection](#) \*workers, struct [starpu\\_sched\\_ctx\\_iterator](#) \*it)
- int(\* [add](#) )(struct [starpu\\_worker\\_collection](#) \*workers, int worker)
- int(\* [remove](#) )(struct [starpu\\_worker\\_collection](#) \*workers, int worker)
- void(\* [init](#) )(struct [starpu\\_worker\\_collection](#) \*workers)
- void(\* [deinit](#) )(struct [starpu\\_worker\\_collection](#) \*workers)
- void(\* [init\\_iterator](#) )(struct [starpu\\_worker\\_collection](#) \*workers, struct [starpu\\_sched\\_ctx\\_iterator](#) \*it)

#### 27.7.2.1.1 Field Documentation

##### 27.7.2.1.1.1 void \* starpu\_worker\_collection::workerids

The workerids managed by the collection

##### 27.7.2.1.1.2 unsigned starpu\_worker\_collection::nworkers

The number of workers in the collection

##### 27.7.2.1.1.3 enum starpu\_worker\_collection\_type starpu\_worker\_collection::type

The type of structure (currently [STARPU\\_WORKER\\_LIST](#) is the only one available)

##### 27.7.2.1.1.4 unsigned(\* starpu\_worker\_collection::has\_next)(struct starpu\_worker\_collection \*workers, struct starpu\_sched\_ctx\_iterator \*it)

Checks if there is another element in collection

##### 27.7.2.1.1.5 int(\* starpu\_worker\_collection::get\_next)(struct starpu\_worker\_collection \*workers, struct starpu\_sched\_ctx\_iterator \*it)

return the next element in the collection

##### 27.7.2.1.1.6 int(\* starpu\_worker\_collection::add)(struct starpu\_worker\_collection \*workers, int worker)

add a new element in the collection

27.7.2.1.1.7 `int(* starpu_worker_collection::remove)(struct starpu_worker_collection *workers, int worker)`

remove an element from the collection

27.7.2.1.1.8 `void(* starpu_worker_collection::init)(struct starpu_worker_collection *workers)`

Initialize the collection

27.7.2.1.1.9 `void(* starpu_worker_collection::deinit)(struct starpu_worker_collection *workers)`

Deinitialize the collection

27.7.2.1.1.10 `void(* starpu_worker_collection::init_iterator)(struct starpu_worker_collection *workers, struct starpu_sched_ctx_iterator *it)`

Initialize the cursor if there is one

27.7.2.2 `struct starpu_sched_ctx_iterator`

Structure needed to iterate on the collection

Data Fields

|        |                                                            |                                                                                             |
|--------|------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| int    | cursor                                                     | The index of the current worker in the collection, needed when iterating on the collection. |
| void * | value                                                      |                                                                                             |
| void * | possible_value                                             |                                                                                             |
| int    | visited[ <a href="#">STARPU↔<br/>U_NMAXWOR↔<br/>KERS</a> ] |                                                                                             |

## 27.7.3 Macro Definition Documentation

27.7.3.1 `#define STARPU_NMAXWORKERS`

Define the maximum number of workers managed by StarPU.

## 27.7.4 Enumeration Type Documentation

27.7.4.1 `enum starpu_node_kind`

TODO

Enumerator

**`STARPU_UNUSED`** TODO

**`STARPU_CPU_RAM`** TODO

**`STARPU_CUDA_RAM`** TODO

**`STARPU_OPENCL_RAM`** TODO

**`STARPU_MIC_RAM`** TODO

**`STARPU_SCC_RAM`** This node kind is not used anymore, but implementations in interfaces will be useful for MPI.

**`STARPU_SCC_SHM`** TODO

#### 27.7.4.2 enum starpu\_worker\_archtype

Worker Architecture Type

Enumerator

**STARPU\_CPU\_WORKER** CPU core  
**STARPU\_CUDA\_WORKER** NVIDIA CUDA device  
**STARPU\_OPENCL\_WORKER** OpenCL device  
**STARPU\_MIC\_WORKER** Intel MIC device  
**STARPU\_SCC\_WORKER** Intel SCC device  
**STARPU\_ANY\_WORKER** any worker, used in the hypervisor

#### 27.7.4.3 enum starpu\_worker\_collection\_type

Types of structures the worker collection can implement

Enumerator

**STARPU\_WORKER\_LIST** The collection is an array

### 27.7.5 Function Documentation

#### 27.7.5.1 unsigned starpu\_worker\_get\_count ( void )

This function returns the number of workers (i.e. processing units executing StarPU tasks). The returned value should be at most [STARPU\\_NMAXWORKERS](#).

#### 27.7.5.2 int starpu\_worker\_get\_count\_by\_type ( enum starpu\_worker\_archtype type )

Returns the number of workers of the given type. A positive (or NULL) value is returned in case of success, -EINVAL indicates that the type is not valid otherwise.

#### 27.7.5.3 unsigned starpu\_cpu\_worker\_get\_count ( void )

This function returns the number of CPUs controlled by StarPU. The returned value should be at most [STARPU\\_MAXCPUS](#).

#### 27.7.5.4 unsigned starpu\_cuda\_worker\_get\_count ( void )

This function returns the number of CUDA devices controlled by StarPU. The returned value should be at most [STARPU\\_MAXCUDADEVES](#).

#### 27.7.5.5 unsigned starpu\_mic\_worker\_get\_count ( void )

This function returns the number of MIC workers controlled by StarPU.

#### 27.7.5.6 unsigned starpu\_mic\_device\_get\_count ( void )

This function returns the number of MIC devices controlled by StarPU. The returned value should be at most [STARPU\\_MAXMICDEVES](#).

**27.7.5.7 unsigned starpu\_scc\_worker\_get\_count ( void )**

This function returns the number of SCC devices controlled by StarPU. The returned value should be at most [STARPU\\_MAXSCCDEVS](#).

**27.7.5.8 unsigned starpu\_opengl\_worker\_get\_count ( void )**

This function returns the number of OpenGL devices controlled by StarPU. The returned value should be at most [STARPU\\_MAXOPENCLDEVS](#).

**27.7.5.9 int starpu\_worker\_get\_id ( void )**

This function returns the identifier of the current worker, i.e the one associated to the calling thread. The returned value is either -1 if the current context is not a StarPU worker (i.e. when called from the application outside a task or a callback), or an integer between 0 and [starpu\\_worker\\_get\\_count\(\)](#) - 1.

**27.7.5.10 int starpu\_worker\_get\_ids\_by\_type ( enum starpu\_worker\_archtype type, int \* workerids, int maxsize )**

This function gets the list of identifiers of workers with the given type. It fills the array `workerids` with the identifiers of the workers that have the type indicated in the first argument. The argument `maxsize` indicates the size of the array `workerids`. The returned value gives the number of identifiers that were put in the array. -ERANGE is returned if `maxsize` is lower than the number of workers with the appropriate type: in that case, the array is filled with the `maxsize` first elements. To avoid such overflows, the value of `maxsize` can be chosen by the means of the function [starpu\\_worker\\_get\\_count\\_by\\_type\(\)](#), or by passing a value greater or equal to [STARPU\\_NMAXWORKERS](#).

**27.7.5.11 int starpu\_worker\_get\_by\_type ( enum starpu\_worker\_archtype type, int num )**

This returns the identifier of the num-th worker that has the specified type type. If there are no such worker, -1 is returned.

**27.7.5.12 int starpu\_worker\_get\_by\_devid ( enum starpu\_worker\_archtype type, int devid )**

This returns the identifier of the worker that has the specified type `type` and device id `devid` (which may not be the n-th, if some devices are skipped for instance). If there are no such worker, -1 is returned.

**27.7.5.13 int starpu\_worker\_get\_devid ( int id )**

This function returns the device id of the given worker. The worker should be identified with the value returned by the [starpu\\_worker\\_get\\_id\(\)](#) function. In the case of a CUDA worker, this device identifier is the logical device identifier exposed by CUDA (used by the function `cudaGetDevice()` for instance). The device identifier of a CPU worker is the logical identifier of the core on which the worker was bound; this identifier is either provided by the OS or by the library `hwloc` in case it is available.

**27.7.5.14 enum starpu\_worker\_archtype starpu\_worker\_get\_type ( int id )**

This function returns the type of processing unit associated to a worker. The worker identifier is a value returned by the function [starpu\\_worker\\_get\\_id\(\)](#). The returned value indicates the architecture of the worker: [STARPU\\_CPU\\_WORKER](#) for a CPU core, [STARPU\\_CUDA\\_WORKER](#) for a CUDA device, and [STARPU\\_OPENGL\\_WORKER](#) for an OpenGL device. The value returned for an invalid identifier is unspecified.

**27.7.5.15** `void starpu_worker_get_name ( int id, char * dst, size_t maxlen )`

This function allows to get the name of a given worker. StarPU associates a unique human readable string to each processing unit. This function copies at most the maxlen first bytes of the unique string associated to a worker identified by its identifier id into the dst buffer. The caller is responsible for ensuring that dst is a valid pointer to a buffer of maxlen bytes at least. Calling this function on an invalid identifier results in an unspecified behaviour.

**27.7.5.16** `unsigned starpu_worker_get_memory_node ( unsigned workerid )`

This function returns the identifier of the memory node associated to the worker identified by workerid.

**27.7.5.17** `enum starpu_node_kind starpu_node_get_kind ( unsigned node )`

Returns the type of the given node as defined by [starpu\\_node\\_kind](#). For example, when defining a new data interface, this function should be used in the allocation function to determine on which device the memory needs to be allocated.

**27.7.5.18** `char * starpu_worker_get_type_as_string ( enum starpu_worker_archtype type )`

Returns the given worker type as a string.

## 27.8 Data Management

This section describes the data management facilities provided by StarPU. We show how to use existing data interfaces in [Data Interfaces](#), but developers can design their own data interfaces if required.

### Typedefs

- typedef struct \_starpu\_data\_state \* [starpu\\_data\\_handle\\_t](#)

### Enumerations

- enum [starpu\\_data\\_access\\_mode](#) {  
[STARPU\\_NONE](#), [STARPU\\_R](#), [STARPU\\_W](#), [STARPU\\_RW](#),  
[STARPU\\_SCRATCH](#), [STARPU\\_REDUX](#), [STARPU\\_COMMUTE](#), [STARPU\\_SSEND](#) }

### Basic Data Management API

Data management is done at a high-level in StarPU: rather than accessing a mere list of contiguous buffers, the tasks may manipulate data that are described by a high-level construct which we call data interface.

An example of data interface is the "vector" interface which describes a contiguous data array on a specific memory node. This interface is a simple structure containing the number of elements in the array, the size of the elements, and the address of the array in the appropriate address space (this address may be invalid if there is no valid copy of the array in the memory node). More informations on the data interfaces provided by StarPU are given in [Data Interfaces](#).

When a piece of data managed by StarPU is used by a task, the task implementation is given a pointer to an interface describing a valid copy of the data that is accessible from the current processing unit.

Every worker is associated to a memory node which is a logical abstraction of the address space from which the processing unit gets its data. For instance, the memory node associated to the different CPU workers represents main memory (RAM), the memory node associated to a GPU is DRAM embedded on the device. Every memory



node is identified by a logical index which is accessible from the function `starpu_worker_get_memory_node()`. When registering a piece of data to StarPU, the specified memory node indicates where the piece of data initially resides (we also call this memory node the home node of a piece of data).

- void `starpu_data_register` (`starpu_data_handle_t` \*handleptr, unsigned home\_node, void \*data\_interface, struct `starpu_data_interface_ops` \*ops)
- void `starpu_data_ptr_register` (`starpu_data_handle_t` handle, unsigned node)
- void `starpu_data_register_same` (`starpu_data_handle_t` \*handledst, `starpu_data_handle_t` handlesrc)
- void `starpu_data_unregister` (`starpu_data_handle_t` handle)
- void `starpu_data_unregister_no_coherency` (`starpu_data_handle_t` handle)
- void `starpu_data_unregister_submit` (`starpu_data_handle_t` handle)
- void `starpu_data_invalidate` (`starpu_data_handle_t` handle)
- void `starpu_data_invalidate_submit` (`starpu_data_handle_t` handle)
- void `starpu_data_set_wt_mask` (`starpu_data_handle_t` handle, uint32\_t wt\_mask)
- int `starpu_data_prefetch_on_node` (`starpu_data_handle_t` handle, unsigned node, unsigned async)
- `starpu_data_handle_t` `starpu_data_lookup` (const void \*ptr)
- int `starpu_data_request_allocation` (`starpu_data_handle_t` handle, unsigned node)
- void `starpu_data_query_status` (`starpu_data_handle_t` handle, int memory\_node, int \*is\_allocated, int \*is\_↵ valid, int \*is\_requested)
- void `starpu_data_advise_as_important` (`starpu_data_handle_t` handle, unsigned is\_important)
- void `starpu_data_set_reduction_methods` (`starpu_data_handle_t` handle, struct `starpu_codelet` \*redux\_cl, struct `starpu_codelet` \*init\_cl)
- struct `starpu_data_interface_ops` \* `starpu_data_get_interface_ops` (`starpu_data_handle_t` handle)

### Access registered data from the application

- #define `STARPU_DATA_ACQUIRE_CB`(handle, mode, code)
- int `starpu_data_acquire` (`starpu_data_handle_t` handle, enum `starpu_data_access_mode` mode)
- int `starpu_data_acquire_cb` (`starpu_data_handle_t` handle, enum `starpu_data_access_mode` mode, void(\*callback)(void \*), void \*arg)
- int `starpu_data_acquire_cb_sequential_consistency` (`starpu_data_handle_t` handle, enum `starpu_data_↵ access_mode` mode, void(\*callback)(void \*), void \*arg, int sequential\_consistency)
- int `starpu_data_acquire_on_node` (`starpu_data_handle_t` handle, int node, enum `starpu_data_access_mode` mode)
- int `starpu_data_acquire_on_node_cb` (`starpu_data_handle_t` handle, int node, enum `starpu_data_access_↵ mode` mode, void(\*callback)(void \*), void \*arg)
- int `starpu_data_acquire_on_node_cb_sequential_consistency` (`starpu_data_handle_t` handle, int node, enum `starpu_data_access_mode` mode, void(\*callback)(void \*), void \*arg, int sequential\_consistency)
- void `starpu_data_release` (`starpu_data_handle_t` handle)
- void `starpu_data_release_on_node` (`starpu_data_handle_t` handle, int node)

### 27.8.1 Detailed Description

This section describes the data management facilities provided by StarPU. We show how to use existing data interfaces in [Data Interfaces](#), but developers can design their own data interfaces if required.

### 27.8.2 Macro Definition Documentation

#### 27.8.2.1 #define STARPU\_DATA\_ACQUIRE\_CB( handle, mode, code )

`STARPU_DATA_ACQUIRE_CB()` is the same as `starpu_data_acquire_cb()`, except that the code to be executed in a callback is directly provided as a macro parameter, and the data `handle` is automatically released after it. This permits to easily execute code which depends on the value of some registered data. This is non-blocking too and may be called from task callbacks.

### 27.8.3 Typedef Documentation

#### 27.8.3.1 `starpu_data_handle_t`

StarPU uses `starpu_data_handle_t` as an opaque handle to manage a piece of data. Once a piece of data has been registered to StarPU, it is associated to a `starpu_data_handle_t` which keeps track of the state of the piece of data over the entire machine, so that we can maintain data consistency and locate data replicates for instance.

### 27.8.4 Enumeration Type Documentation

#### 27.8.4.1 `enum starpu_data_access_mode`

This datatype describes a data access mode.

Enumerator

**`STARPU_NONE`** TODO

**`STARPU_R`** read-only mode.

**`STARPU_W`** write-only mode.

**`STARPU_RW`** read-write mode. This is equivalent to `STARPU_R|STARPU_W`

**`STARPU_SCRATCH`** A temporary buffer is allocated for the task, but StarPU does not enforce data consistency—i.e. each device has its own buffer, independently from each other (even for CPUs), and no data transfer is ever performed. This is useful for temporary variables to avoid allocating/freeing buffers inside each task. Currently, no behavior is defined concerning the relation with the `STARPU_R` and `STARPU_W` modes and the value provided at registration — i.e., the value of the scratch buffer is undefined at entry of the codelet function. It is being considered for future extensions at least to define the initial value. For now, data to be used in `STARPU_SCRATCH` mode should be registered with node `-1` and a `NULL` pointer, since the value of the provided buffer is simply ignored for now.

**`STARPU_REDUX`** todo

**`STARPU_COMMUTE`** In addition to that, `STARPU_COMMUTE` can be passed along `STARPU_W` or `STARPU_RW` to express that StarPU can let tasks commute, which is useful e.g. when bringing a contribution into some data, which can be done in any order (but still require sequential consistency against reads or non-commutative writes).

**`STARPU_SSEND`** used in `starpu_mpi_insert_task()` to specify the data has to be sent using a synchronous and non-blocking mode (see `starpu_mpi_issend()`)

### 27.8.5 Function Documentation

#### 27.8.5.1 `void starpu_data_register ( starpu_data_handle_t * handleptr, unsigned home_node, void * data_interface, struct starpu_data_interface_ops * ops )`

Register a piece of data into the handle located at the `handleptr` address. The `data_interface` buffer contains the initial description of the data in the `home_node`. The `ops` argument is a pointer to a structure describing the different methods used to manipulate this type of interface. See `starpu_data_interface_ops` for more details on this structure. If `home_node` is `-1`, StarPU will automatically allocate the memory when it is used for the first time in write-only mode. Once such data handle has been automatically allocated, it is possible to access it using any access mode. Note that StarPU supplies a set of predefined types of interface (e.g. vector or matrix) which can be registered by the means of helper functions (e.g. `starpu_vector_data_register()` or `starpu_matrix_data_register()`).

#### 27.8.5.2 `void starpu_data_ptr_register ( starpu_data_handle_t handle, unsigned node )`

Register that a buffer for `handle` on `node` will be set. This is typically used by `starpu_*_ptr_register` helpers before setting the interface pointers for this node, to tell the core that that is now allocated.

**27.8.5.3** `void starpu_data_register_same ( starpu_data_handle_t * handledst, starpu_data_handle_t handlesrc )`

Register a new piece of data into the handle `handledst` with the same interface as the handle `handlesrc`.

**27.8.5.4** `void starpu_data_unregister ( starpu_data_handle_t handle )`

This function unregisters a data handle from StarPU. If the data was automatically allocated by StarPU because the home node was -1, all automatically allocated buffers are freed. Otherwise, a valid copy of the data is put back into the home node in the buffer that was initially registered. Using a data handle that has been unregistered from StarPU results in an undefined behaviour. In case we do not need to update the value of the data in the home node, we can use the function [starpu\\_data\\_unregister\\_no\\_coherency\(\)](#) instead.

**27.8.5.5** `void starpu_data_unregister_no_coherency ( starpu_data_handle_t handle )`

This is the same as [starpu\\_data\\_unregister\(\)](#), except that StarPU does not put back a valid copy into the home node, in the buffer that was initially registered.

**27.8.5.6** `void starpu_data_unregister_submit ( starpu_data_handle_t handle )`

Destroy the data handle once it is not needed anymore by any submitted task. No coherency is assumed.

**27.8.5.7** `void starpu_data_invalidate ( starpu_data_handle_t handle )`

Destroy all replicates of the data handle immediately. After data invalidation, the first access to the handle must be performed in write-only mode. Accessing an invalidated data in read-mode results in undefined behaviour.

**27.8.5.8** `void starpu_data_invalidate_submit ( starpu_data_handle_t handle )`

Submits invalidation of the data handle after completion of previously submitted tasks.

**27.8.5.9** `void starpu_data_set_wt_mask ( starpu_data_handle_t handle, uint32_t wt_mask )`

This function sets the write-through mask of a given data (and its children), i.e. a bitmask of nodes where the data should be always replicated after modification. It also prevents the data from being evicted from these nodes when memory gets scarce. When the data is modified, it is automatically transferred into those memory node. For instance a  $1 << 0$  write-through mask means that the CUDA workers will commit their changes in main memory (node 0).

**27.8.5.10** `int starpu_data_prefetch_on_node ( starpu_data_handle_t handle, unsigned node, unsigned async )`

Issue a prefetch request for a given data to a given node, i.e. requests that the data be replicated to the given node, so that it is available there for tasks. If the `async` parameter is 0, the call will block until the transfer is achieved, else the call will return immediately, after having just queued the request. In the latter case, the request will asynchronously wait for the completion of any task writing on the data.

**27.8.5.11** `starpu_data_handle_t starpu_data_lookup ( const void * ptr )`

Return the handle corresponding to the data pointed to by the `ptr` host pointer.

**27.8.5.12** `int starpu_data_request_allocation ( starpu_data_handle_t handle, unsigned node )`

Explicitly ask StarPU to allocate room for a piece of data on the specified memory node.

**27.8.5.13** `void starpu_data_query_status ( starpu_data_handle_t handle, int memory_node, int * is_allocated, int * is_valid, int * is_requested )`

Query the status of `handle` on the specified `memory_node`.

**27.8.5.14** `void starpu_data_advise_as_important ( starpu_data_handle_t handle, unsigned is_important )`

This function allows to specify that a piece of data can be discarded without impacting the application.

**27.8.5.15** `void starpu_data_set_reduction_methods ( starpu_data_handle_t handle, struct starpu_codelet * redux_cl, struct starpu_codelet * init_cl )`

This sets the codelets to be used for `handle` when it is accessed in the mode `STARPU_REDUX`. Per-worker buffers will be initialized with the codelet `init_cl`, and reduction between per-worker buffers will be done with the codelet `redux_cl`.

**27.8.5.16** `struct starpu_data_interface_ops * starpu_data_get_interface_ops ( starpu_data_handle_t handle )`

todo

**27.8.5.17** `int starpu_data_acquire ( starpu_data_handle_t handle, enum starpu_data_access_mode mode )`

The application must call this function prior to accessing registered data from main memory outside tasks. StarPU ensures that the application will get an up-to-date copy of the data in main memory located where the data was originally registered, and that all concurrent accesses (e.g. from tasks) will be consistent with the access mode specified in the mode argument. `starpu_data_release()` must be called once the application does not need to access the piece of data anymore. Note that implicit data dependencies are also enforced by `starpu_data_acquire()`, i.e. `starpu_data_acquire()` will wait for all tasks scheduled to work on the data, unless they have been disabled explicitly by calling `starpu_data_set_default_sequential_consistency_flag()` or `starpu_data_set_sequential_consistency_flag()`. `starpu_data_acquire()` is a blocking call, so that it cannot be called from tasks or from their callbacks (in that case, `starpu_data_acquire()` returns `-EDEADLK`). Upon successful completion, this function returns 0.

**27.8.5.18** `int starpu_data_acquire_cb ( starpu_data_handle_t handle, enum starpu_data_access_mode mode, void(*) (void *) callback, void * arg )`

Asynchronous equivalent of `starpu_data_acquire()`. When the data specified in `handle` is available in the appropriate access mode, the `callback` function is executed. The application may access the requested data during the execution of this `callback`. The `callback` function must call `starpu_data_release()` once the application does not need to access the piece of data anymore. Note that implicit data dependencies are also enforced by `starpu_data_acquire_cb()` in case they are not disabled. Contrary to `starpu_data_acquire()`, this function is non-blocking and may be called from task callbacks. Upon successful completion, this function returns 0.

**27.8.5.19** `int starpu_data_acquire_cb_sequential_consistency ( starpu_data_handle_t handle, enum starpu_data_access_mode mode, void(*) (void *) callback, void * arg, int sequential_consistency )`

Equivalent of `starpu_data_acquire_cb()` with the possibility of enabling or disabling data dependencies. When the data specified in `handle` is available in the appropriate access mode, the `callback` function is executed. The application may access the requested data during the execution of this `callback`. The `callback` function must call `starpu_data_release()` once the application does not need to access the piece of data anymore. Note that implicit data dependencies are also enforced by `starpu_data_acquire_cb_sequential_consistency()` in case they are not disabled specifically for the given `handle` or by the parameter `sequential_consistency`. Similarly to `starpu_data_acquire_cb()`, this function is non-blocking and may be called from task callbacks. Upon successful completion, this function returns 0.

27.8.5.20 `int starpu_data_acquire_on_node ( starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode )`

This is the same as [starpu\\_data\\_acquire\(\)](#), except that the data will be available on the given memory node instead of main memory.

27.8.5.21 `int starpu_data_acquire_on_node_cb ( starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode, void (*)(void *) callback, void * arg )`

This is the same as [starpu\\_data\\_acquire\\_cb\(\)](#), except that the data will be available on the given memory node instead of main memory.

27.8.5.22 `int starpu_data_acquire_on_node_cb_sequential_consistency ( starpu_data_handle_t handle, int node, enum starpu_data_access_mode mode, void (*)(void *) callback, void * arg, int sequential_consistency )`

This is the same as [starpu\\_data\\_acquire\\_cb\\_sequential\\_consistency\(\)](#), except that the data will be available on the given memory node instead of main memory.

27.8.5.23 `void starpu_data_release ( starpu_data_handle_t handle )`

This function releases the piece of data acquired by the application either by [starpu\\_data\\_acquire\(\)](#) or by [starpu\\_data\\_acquire\\_cb\(\)](#).

27.8.5.24 `void starpu_data_release_on_node ( starpu_data_handle_t handle, int node )`

This is the same as [starpu\\_data\\_release\(\)](#), except that the data will be available on the given memory node instead of main memory.

## 27.9 Data Interfaces

### Data Structures

- struct [starpu\\_data\\_interface\\_ops](#)
- struct [starpu\\_data\\_copy\\_methods](#)
- struct [starpu\\_variable\\_interface](#)
- struct [starpu\\_vector\\_interface](#)
- struct [starpu\\_matrix\\_interface](#)
- struct [starpu\\_block\\_interface](#)
- struct [starpu\\_bcsr\\_interface](#)
- struct [starpu\\_csr\\_interface](#)
- struct [starpu\\_coo\\_interface](#)

### Enumerations

- enum [starpu\\_data\\_interface\\_id](#) {  
[STARPU\\_UNKNOWN\\_INTERFACE\\_ID](#), [STARPU\\_MATRIX\\_INTERFACE\\_ID](#), [STARPU\\_BLOCK\\_INTERFACE\\_ID](#), [STARPU\\_VECTOR\\_INTERFACE\\_ID](#),  
[STARPU\\_CSR\\_INTERFACE\\_ID](#), [STARPU\\_BCSR\\_INTERFACE\\_ID](#), [STARPU\\_VARIABLE\\_INTERFACE\\_ID](#),  
[STARPU\\_VOID\\_INTERFACE\\_ID](#),  
[STARPU\\_MULTIFORMAT\\_INTERFACE\\_ID](#), [STARPU\\_COO\\_INTERFACE\\_ID](#), [STARPU\\_MAX\\_INTERFACE\\_ID](#) }

## Registering Data

There are several ways to register a memory region so that it can be managed by StarPU. The functions below allow the registration of vectors, 2D matrices, 3D matrices as well as BCSR and CSR sparse matrices.

- void [starpu\\_void\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle)
- void [starpu\\_variable\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, uintptr\_t ptr, size\_t size)
- void [starpu\\_variable\\_ptr\\_register](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned node, uintptr\_t ptr, uintptr\_t dev↔\_handle, size\_t offset)
- void [starpu\\_vector\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, uintptr\_t ptr, uint32\_t nx, size\_t elemsize)
- void [starpu\\_vector\\_ptr\\_register](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned node, uintptr\_t ptr, uintptr\_t dev↔\_handle, size\_t offset)
- void [starpu\\_matrix\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, uintptr\_t ptr, uint32\_t ld, uint32\_t nx, uint32\_t ny, size\_t elemsize)
- void [starpu\\_matrix\\_ptr\\_register](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned node, uintptr\_t ptr, uintptr\_t dev↔\_handle, size\_t offset, uint32\_t ld)
- void [starpu\\_block\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, uintptr\_t ptr, uint32↔\_t ld, uint32\_t ldz, uint32\_t nx, uint32\_t ny, uint32\_t nz, size\_t elemsize)
- void [starpu\\_block\\_ptr\\_register](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned node, uintptr\_t ptr, uintptr\_t dev↔\_handle, size\_t offset, uint32\_t ld, uint32\_t ldz)
- void [starpu\\_bcsr\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, uint32\_t nnz, uint32↔\_t nrow, uintptr\_t nzval, uint32\_t \*colind, uint32\_t \*rowptr, uint32\_t firstentry, uint32\_t r, uint32\_t c, size\_t elemsize)
- void [starpu\\_csr\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, uint32\_t nnz, uint32\_t nrow, uintptr\_t nzval, uint32\_t \*colind, uint32\_t \*rowptr, uint32\_t firstentry, size\_t elemsize)
- void [starpu\\_coo\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handleptr, unsigned home\_node, uint32\_t nx, uint32\_t ny, uint32\_t n\_values, uint32\_t \*columns, uint32\_t \*rows, uintptr\_t values, size\_t elemsize)
- void \* [starpu\\_data\\_get\\_interface\\_on\\_node](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned memory\_node)

## Accessing Data Interfaces

Each data interface is provided with a set of field access functions. The ones using a void \* parameter aimed to be used in codelet implementations (see for example the code in [Vector Scaling Using StarPU's API](#)).

- void \* [starpu\\_data\\_handle\\_to\\_pointer](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned node)
- void \* [starpu\\_data\\_get\\_local\\_ptr](#) ([starpu\\_data\\_handle\\_t](#) handle)
- enum [starpu\\_data\\_interface\\_id](#) [starpu\\_data\\_get\\_interface\\_id](#) ([starpu\\_data\\_handle\\_t](#) handle)
- size\_t [starpu\\_data\\_get\\_size](#) ([starpu\\_data\\_handle\\_t](#) handle)
- int [starpu\\_data\\_pack](#) ([starpu\\_data\\_handle\\_t](#) handle, void \*\*ptr, starpu\_ssize\_t \*count)
- int [starpu\\_data\\_unpack](#) ([starpu\\_data\\_handle\\_t](#) handle, void \*ptr, size\_t count)

## Accessing Variable Data Interfaces

- #define [STARPU\\_VARIABLE\\_GET\\_PTR](#)(interface)
- #define [STARPU\\_VARIABLE\\_GET\\_ELEMSIZE](#)(interface)
- #define [STARPU\\_VARIABLE\\_GET\\_DEV\\_HANDLE](#)(interface)
- #define [STARPU\\_VARIABLE\\_GET\\_OFFSET](#)(interface)
- size\_t [starpu\\_variable\\_get\\_elemsize](#) ([starpu\\_data\\_handle\\_t](#) handle)
- uintptr\_t [starpu\\_variable\\_get\\_local\\_ptr](#) ([starpu\\_data\\_handle\\_t](#) handle)

### Accessing Vector Data Interfaces

- `#define STARPU_VECTOR_GET_PTR(interface)`
- `#define STARPU_VECTOR_GET_DEV_HANDLE(interface)`
- `#define STARPU_VECTOR_GET_OFFSET(interface)`
- `#define STARPU_VECTOR_GET_NX(interface)`
- `#define STARPU_VECTOR_GET_ELEMSIZE(interface)`
- `#define STARPU_VECTOR_GET_SLICE_BASE(interface)`
- `uint32_t starpu_vector_get_nx (starpu_data_handle_t handle)`
- `size_t starpu_vector_get_elemsize (starpu_data_handle_t handle)`
- `uintptr_t starpu_vector_get_local_ptr (starpu_data_handle_t handle)`

### Accessing Matrix Data Interfaces

- `#define STARPU_MATRIX_GET_PTR(interface)`
- `#define STARPU_MATRIX_GET_DEV_HANDLE(interface)`
- `#define STARPU_MATRIX_GET_OFFSET(interface)`
- `#define STARPU_MATRIX_GET_NX(interface)`
- `#define STARPU_MATRIX_GET_NY(interface)`
- `#define STARPU_MATRIX_GET_LD(interface)`
- `#define STARPU_MATRIX_GET_ELEMSIZE(interface)`
- `uint32_t starpu_matrix_get_nx (starpu_data_handle_t handle)`
- `uint32_t starpu_matrix_get_ny (starpu_data_handle_t handle)`
- `uint32_t starpu_matrix_get_local_ld (starpu_data_handle_t handle)`
- `uintptr_t starpu_matrix_get_local_ptr (starpu_data_handle_t handle)`
- `size_t starpu_matrix_get_elemsize (starpu_data_handle_t handle)`

### Accessing Block Data Interfaces

- `#define STARPU_BLOCK_GET_PTR(interface)`
- `#define STARPU_BLOCK_GET_DEV_HANDLE(interface)`
- `#define STARPU_BLOCK_GET_OFFSET(interface)`
- `#define STARPU_BLOCK_GET_NX(interface)`
- `#define STARPU_BLOCK_GET_NY(interface)`
- `#define STARPU_BLOCK_GET_NZ(interface)`
- `#define STARPU_BLOCK_GET_LDY(interface)`
- `#define STARPU_BLOCK_GET_LDZ(interface)`
- `#define STARPU_BLOCK_GET_ELEMSIZE(interface)`
- `uint32_t starpu_block_get_nx (starpu_data_handle_t handle)`
- `uint32_t starpu_block_get_ny (starpu_data_handle_t handle)`
- `uint32_t starpu_block_get_nz (starpu_data_handle_t handle)`
- `uint32_t starpu_block_get_local_ldy (starpu_data_handle_t handle)`
- `uint32_t starpu_block_get_local_ldz (starpu_data_handle_t handle)`
- `uintptr_t starpu_block_get_local_ptr (starpu_data_handle_t handle)`
- `size_t starpu_block_get_elemsize (starpu_data_handle_t handle)`



### Accessing BCSR Data Interfaces

- `#define STARPU_BCSR_GET_NNZ(interface)`
- `#define STARPU_BCSR_GET_NZVAL(interface)`
- `#define STARPU_BCSR_GET_NZVAL_DEV_HANDLE(interface)`
- `#define STARPU_BCSR_GET_COLIND(interface)`
- `#define STARPU_BCSR_GET_COLIND_DEV_HANDLE(interface)`
- `#define STARPU_BCSR_GET_ROWPTR(interface)`
- `#define STARPU_BCSR_GET_ROWPTR_DEV_HANDLE(interface)`
- `#define STARPU_BCSR_GET_OFFSET`
- `uint32_t starpu_bcsr_get_nnz (starpu_data_handle_t handle)`
- `uint32_t starpu_bcsr_get_nrow (starpu_data_handle_t handle)`
- `uint32_t starpu_bcsr_get_firstentry (starpu_data_handle_t handle)`
- `uintptr_t starpu_bcsr_get_local_nzval (starpu_data_handle_t handle)`
- `uint32_t * starpu_bcsr_get_local_colind (starpu_data_handle_t handle)`
- `uint32_t * starpu_bcsr_get_local_rowptr (starpu_data_handle_t handle)`
- `uint32_t starpu_bcsr_get_r (starpu_data_handle_t handle)`
- `uint32_t starpu_bcsr_get_c (starpu_data_handle_t handle)`
- `size_t starpu_bcsr_get_elemsize (starpu_data_handle_t handle)`

### Accessing CSR Data Interfaces

- `#define STARPU_CSR_GET_NNZ(interface)`
- `#define STARPU_CSR_GET_NROW(interface)`
- `#define STARPU_CSR_GET_NZVAL(interface)`
- `#define STARPU_CSR_GET_NZVAL_DEV_HANDLE(interface)`
- `#define STARPU_CSR_GET_COLIND(interface)`
- `#define STARPU_CSR_GET_COLIND_DEV_HANDLE(interface)`
- `#define STARPU_CSR_GET_ROWPTR(interface)`
- `#define STARPU_CSR_GET_ROWPTR_DEV_HANDLE(interface)`
- `#define STARPU_CSR_GET_OFFSET`
- `#define STARPU_CSR_GET_FIRSTENTRY(interface)`
- `#define STARPU_CSR_GET_ELEMSIZE(interface)`
- `uint32_t starpu_csr_get_nnz (starpu_data_handle_t handle)`
- `uint32_t starpu_csr_get_nrow (starpu_data_handle_t handle)`
- `uint32_t starpu_csr_get_firstentry (starpu_data_handle_t handle)`
- `uintptr_t starpu_csr_get_local_nzval (starpu_data_handle_t handle)`
- `uint32_t * starpu_csr_get_local_colind (starpu_data_handle_t handle)`
- `uint32_t * starpu_csr_get_local_rowptr (starpu_data_handle_t handle)`
- `size_t starpu_csr_get_elemsize (starpu_data_handle_t handle)`

### Accessing COO Data Interfaces

- `#define STARPU_COO_GET_COLUMNS(interface)`
- `#define STARPU_COO_GET_COLUMNS_DEV_HANDLE(interface)`
- `#define STARPU_COO_GET_ROWS(interface)`
- `#define STARPU_COO_GET_ROWS_DEV_HANDLE(interface)`
- `#define STARPU_COO_GET_VALUES(interface)`
- `#define STARPU_COO_GET_VALUES_DEV_HANDLE(interface)`
- `#define STARPU_COO_GET_OFFSET`
- `#define STARPU_COO_GET_NX(interface)`
- `#define STARPU_COO_GET_NY(interface)`
- `#define STARPU_COO_GET_NVALUES(interface)`
- `#define STARPU_COO_GET_ELEMSIZE(interface)`



## Defining Interface

Applications can provide their own interface as shown in [Defining A New Data Interface](#).

- `uintptr_t starpu_malloc_on_node` (unsigned dst\_node, size\_t size)
- `void starpu_free_on_node` (unsigned dst\_node, uintptr\_t addr, size\_t size)
- `int starpu_interface_copy` (uintptr\_t src, size\_t src\_offset, unsigned src\_node, uintptr\_t dst, size\_t dst\_offset, unsigned dst\_node, size\_t size, void \*async\_data)
- `uint32_t starpu_hash_crc32c_be_n` (const void \*input, size\_t n, uint32\_t inputcrc)
- `uint32_t starpu_hash_crc32c_be` (uint32\_t input, uint32\_t inputcrc)
- `uint32_t starpu_hash_crc32c_string` (const char \*str, uint32\_t inputcrc)
- `int starpu_data_interface_get_next_id` (void)

### 27.9.1 Detailed Description

### 27.9.2 Data Structure Documentation

#### 27.9.2.1 struct starpu\_data\_interface\_ops

Per-interface data transfer methods.

#### Data Fields

- `void(* register_data_handle)(starpu_data_handle_t handle, unsigned home_node, void *data_interface)`
- `starpu_ssize_t(* allocate_data_on_node)(void *data_interface, unsigned node)`
- `void(* free_data_on_node)(void *data_interface, unsigned node)`
- `const struct`  
`starpu_data_copy_methods * copy_methods`
- `void(* handle_to_pointer)(starpu_data_handle_t handle, unsigned node)`
- `size_t(* get_size)(starpu_data_handle_t handle)`
- `uint32_t(* footprint)(starpu_data_handle_t handle)`
- `int(* compare)(void *data_interface_a, void *data_interface_b)`
- `void(* display)(starpu_data_handle_t handle, FILE *f)`
- `starpu_ssize_t(* describe)(void *data_interface, char *buf, size_t size)`
- `enum starpu_data_interface_id interfaceid`
- `size_t interface_size`
- `int is_multiformat`
- `struct`  
`starpu_multiformat_data_interface_ops>(* get_mf_ops)(void *data_interface)`
- `int(* pack_data)(starpu_data_handle_t handle, unsigned node, void **ptr, starpu_ssize_t *count)`
- `int(* unpack_data)(starpu_data_handle_t handle, unsigned node, void *ptr, size_t count)`

#### 27.9.2.1.1 Field Documentation

**27.9.2.1.1.1** `void(* starpu_data_interface_ops::register_data_handle)(starpu_data_handle_t handle, unsigned home_node, void *data_interface)`

Register an existing interface into a data handle.

**27.9.2.1.1.2** `starpu_ssize_t(* starpu_data_interface_ops::allocate_data_on_node)(void *data_interface, unsigned node)`

Allocate data for the interface on a given node.

**27.9.2.1.1.3** `void(* starpu_data_interface_ops::free_data_on_node)(void *data_interface, unsigned node)`

Free data of the interface on a given node.

27.9.2.1.1.4 `const struct starpu_data_copy_methods * starpu_data_interface_ops::copy_methods`

ram/cuda/opencl synchronous and asynchronous transfer methods.

27.9.2.1.1.5 `void *(* starpu_data_interface_ops::handle_to_pointer)(starpu_data_handle_t handle, unsigned node)`

Return the current pointer (if any) for the handle on the given node.

27.9.2.1.1.6 `size_t(* starpu_data_interface_ops::get_size)(starpu_data_handle_t handle)`

Return an estimation of the size of data, for performance models.

27.9.2.1.1.7 `uint32_t(* starpu_data_interface_ops::footprint)(starpu_data_handle_t handle)`

Return a 32bit footprint which characterizes the data size.

27.9.2.1.1.8 `int(* starpu_data_interface_ops::compare)(void *data_interface_a, void *data_interface_b)`

Compare the data size of two interfaces.

27.9.2.1.1.9 `void(* starpu_data_interface_ops::display)(starpu_data_handle_t handle, FILE *f)`

Dump the sizes of a handle to a file.

27.9.2.1.1.10 `starpu_ssize_t(* starpu_data_interface_ops::describe)(void *data_interface, char *buf, size_t size)`

Describe the data into a string.

27.9.2.1.1.11 `enum starpu_data_interface_id starpu_data_interface_ops::interfaceid`

An identifier that is unique to each interface.

27.9.2.1.1.12 `size_t starpu_data_interface_ops::interface_size`

The size of the interface data descriptor.

27.9.2.1.1.13 `int starpu_data_interface_ops::is_multiformat`

todo

27.9.2.1.1.14 `struct starpu_multiformat_data_interface_ops *(* starpu_data_interface_ops::get_mf_ops)(void *data_interface)`

todo

27.9.2.1.1.15 `int(* starpu_data_interface_ops::pack_data)(starpu_data_handle_t handle, unsigned node, void **ptr, starpu_ssize_t *count)`

Pack the data handle into a contiguous buffer at the address allocated with `starpu_malloc_flags(ptr, size, 0)` (and thus returned in `ptr`) and set the size of the newly created buffer in `count`. If `ptr` is `NULL`, the function should not copy the data in the buffer but just set `count` to the size of the buffer which would have been allocated. The special value `-1` indicates the size is yet unknown.

27.9.2.1.1.16 `int(* starpu_data_interface_ops::unpack_data)(starpu_data_handle_t handle, unsigned node, void *ptr, size_t count)`

Unpack the data handle from the contiguous buffer at the address `ptr` of size `count`

## 27.9.2.2 struct starpu\_data\_copy\_methods

Defines the per-interface methods. If the `any_to_any` method is provided, it will be used by default if no more specific method is provided. It can still be useful to provide more specific method in case of e.g. available particular CUDA or OpenCL support.

## Data Fields

- `int(* can_copy)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, unsigned handling_node)`
- `int(* ram_to_ram)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* ram_to_cuda)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* ram_to_opencil)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* ram_to_mic)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* cuda_to_ram)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* cuda_to_cuda)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* cuda_to_opencil)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* opencil_to_ram)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* opencil_to_cuda)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* opencil_to_opencil)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* mic_to_ram)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* scc_src_to_sink)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* scc_sink_to_src)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* scc_sink_to_sink)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* ram_to_cuda_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, starpu_cudaStream_t stream)`
- `int(* cuda_to_ram_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, starpu_cudaStream_t stream)`
- `int(* cuda_to_cuda_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, starpu_cudaStream_t stream)`
- `int(* ram_to_opencil_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, cl_event *event)`
- `int(* opencil_to_ram_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, cl_event *event)`
- `int(* opencil_to_opencil_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, cl_event *event)`
- `int(* ram_to_mic_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* mic_to_ram_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`
- `int(* any_to_any)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, void *async_data)`

## 27.9.2.2.1 Field Documentation

**27.9.2.2.1.1** `int(* starpu_data_copy_methods::can_copy)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, unsigned handling_node)`

If defined, allows the interface to declare whether it supports transferring from `src_interface` on node `src_node` to `dst_interface` on node `dst_node`, run from node `handling_node`. If not defined, it is assumed that the interface supports all transfers.

**27.9.2.2.1.2** `int(* starpu_data_copy_methods::ram_to_ram)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` CPU node. Return 0 on success.

**27.9.2.2.1.3** `int(* starpu_data_copy_methods::ram_to_cuda)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` CUDA node. Return 0 on success.

**27.9.2.2.1.4** `int(* starpu_data_copy_methods::ram_to_opengl)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` OpenCL node. Return 0 on success.

**27.9.2.2.1.5** `int(* starpu_data_copy_methods::ram_to_mic)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` MIC node. Return 0 on success.

**27.9.2.2.1.6** `int(* starpu_data_copy_methods::cuda_to_ram)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CUDA node to the `dst_interface` interface on the `dst_node` CPU node. Return 0 on success.

**27.9.2.2.1.7** `int(* starpu_data_copy_methods::cuda_to_cuda)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CUDA node to the `dst_interface` interface on the `dst_node` CUDA node. Return 0 on success.

**27.9.2.2.1.8** `int(* starpu_data_copy_methods::cuda_to_opengl)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CUDA node to the `dst_interface` interface on the `dst_node` OpenCL node. Return 0 on success.

**27.9.2.2.1.9** `int(* starpu_data_copy_methods::opengl_to_ram)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` OpenCL node to the `dst_interface` interface on the `dst_node` CPU node. Return 0 on success.

**27.9.2.2.1.10** `int(* starpu_data_copy_methods::opengl_to_cuda)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` OpenCL node to the `dst_interface` interface on the `dst_node` CUDA node. Return 0 on success.

**27.9.2.2.1.11** `int(* starpu_data_copy_methods::opengl_to_opengl)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` OpenCL node to the `dst_interface` interface on the `dst_node` OpenCL node. Return 0 on success.

**27.9.2.2.1.12** `int(* starpu_data_copy_methods::mic_to_ram)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` MIC node to the `dst_interface` interface on the `dst_node` CPU node. Return 0 on success.

**27.9.2.2.1.13** `int(* starpu_data_copy_methods::scc_src_to_sink)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` node to the `dst_interface` interface on the `dst_node` node. Must return 0 if the transfer was actually completed completely synchronously, or -EAGAIN if at least some transfers are still ongoing and should be awaited for by the core.

**27.9.2.2.1.14** `int(* starpu_data_copy_methods::scc_sink_to_src)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` node to the `dst_interface` interface on the `dst_node` node. Must return 0 if the transfer was actually completed completely synchronously, or -EAGAIN if at least some transfers are still ongoing and should be awaited for by the core.

**27.9.2.2.1.15** `int(* starpu_data_copy_methods::scc_sink_to_sink)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` node to the `dst_interface` interface on the `dst_node` node. Must return 0 if the transfer was actually completed completely synchronously, or -EAGAIN if at least some transfers are still ongoing and should be awaited for by the core.

**27.9.2.2.1.16** `int(* starpu_data_copy_methods::ram_to_cuda_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, cudaStream_t stream)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` CUDA node, using the given stream. Must return 0 if the transfer was actually completed completely synchronously, or -EAGAIN if at least some transfers are still ongoing and should be awaited for by the core.

**27.9.2.2.1.17** `int(* starpu_data_copy_methods::cuda_to_ram_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, cudaStream_t stream)`

Define how to copy data from the `src_interface` interface on the `src_node` CUDA node to the `dst_interface` interface on the `dst_node` CPU node, using the given stream. Must return 0 if the transfer was actually completed completely synchronously, or -EAGAIN if at least some transfers are still ongoing and should be awaited for by the core.

**27.9.2.2.1.18** `int(* starpu_data_copy_methods::cuda_to_cuda_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, cudaStream_t stream)`

Define how to copy data from the `src_interface` interface on the `src_node` CUDA node to the `dst_interface` interface on the `dst_node` CUDA node, using the given stream. Must return 0 if the transfer was actually completed completely synchronously, or -EAGAIN if at least some transfers are still ongoing and should be awaited for by the core.

**27.9.2.2.1.19** `int(* starpu_data_copy_methods::ram_to_opengl_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, cl_event *event)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` OpenCL node, by recording in event, a pointer to a `cl_event`, the event of the last submitted transfer. Must return 0 if the transfer was actually completed completely synchronously, or -EAGAIN if at least some transfers are still ongoing and should be awaited for by the core.

**27.9.2.2.1.20** `int(* starpu_data_copy_methods::opengl_to_ram_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, cl_event *event)`

Define how to copy data from the `src_interface` interface on the `src_node` OpenCL node to the `dst_interface` interface on the `dst_node` CPU node, by recording in event, a pointer to a `cl_event`, the event of the last submitted transfer. Must return 0 if the transfer was actually completed completely synchronously, or -EAGAIN if at least some transfers are still ongoing and should be awaited for by the core.

**27.9.2.2.1.21** `int(* starpu_data_copy_methods::opengl_to_opengl_async)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, cl_event *event)`

Define how to copy data from the `src_interface` interface on the `src_node` OpenCL node to the `dst_interface` interface on the `dst_node` OpenCL node, by recording in event, a pointer to a `cl_event`, the event of the last submitted transfer. Must return 0 if the transfer was actually completed completely synchronously, or -EAGAIN if at least some transfers are still ongoing and should be awaited for by the core.

**27.9.2.2.1.22** `int(* starpu_data_copy_methods::ram_to_mic_async)(void *src_intreface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` CPU node to the `dst_interface` interface on the `dst_node` MIC node. Must return 0 if the transfer was actually completed completely synchronously, or -EAGAIN if at least some transfers are still ongoing and should be awaited for by the core.

**27.9.2.2.1.23** `int(* starpu_data_copy_methods::mic_to_ram_async)(void *src_intreface, unsigned src_node, void *dst_interface, unsigned dst_node)`

Define how to copy data from the `src_interface` interface on the `src_node` MIC node to the `dst_interface` interface on the `dst_node` CPU node. Must return 0 if the transfer was actually completed completely synchronously, or -EAGAIN if at least some transfers are still ongoing and should be awaited for by the core.

**27.9.2.2.1.24** `int(* starpu_data_copy_methods::any_to_any)(void *src_interface, unsigned src_node, void *dst_interface, unsigned dst_node, void *async_data)`

Define how to copy data from the `src_interface` interface on the `src_node` node to the `dst_interface` interface on the `dst_node` node. This is meant to be implemented through the `starpu_interface_copy()` helper, to which `async_data` should be passed as such, and will be used to manage asynchronicity. This must return -EAGAIN if any of the `starpu_interface_copy()` calls has returned -EAGAIN (i.e. at least some transfer is still ongoing), and return 0 otherwise.

### 27.9.2.3 struct starpu\_variable\_interface

Variable interface for a single data (not a vector, a matrix, a list, ...)

#### Data Fields

|                                                     |            |                                |
|-----------------------------------------------------|------------|--------------------------------|
| enum<br><code>starpu_data_↔<br/>interface_id</code> | id         | Identifier of the interface    |
| uintptr_t                                           | ptr        | local pointer of the variable  |
| uintptr_t                                           | dev_handle | device handle of the variable. |
| size_t                                              | offset     | offset in the variable         |
| size_t                                              | elemsize   | size of the variable           |

### 27.9.2.4 struct starpu\_vector\_interface

Vector interface

#### Data Fields

|                                                     |    |                             |
|-----------------------------------------------------|----|-----------------------------|
| enum<br><code>starpu_data_↔<br/>interface_id</code> | id | Identifier of the interface |
|-----------------------------------------------------|----|-----------------------------|

|           |            |                                                              |
|-----------|------------|--------------------------------------------------------------|
| uintptr_t | ptr        | local pointer of the vector                                  |
| uintptr_t | dev_handle | device handle of the vector.                                 |
| size_t    | offset     | offset in the vector                                         |
| uint32_t  | nx         | number of elements on the x-axis of the vector               |
| size_t    | elemsize   | size of the elements of the vector                           |
| uint32_t  | slice_base | vector slice base, used by the StarPU OpenMP runtime support |

#### 27.9.2.5 struct starpu\_matrix\_interface

Matrix interface for dense matrices

Data Fields

|                                                        |            |                                                                                                                                            |
|--------------------------------------------------------|------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| enum<br><a href="#">starpu_data_↔<br/>interface_id</a> | id         | Identifier of the interface                                                                                                                |
| uintptr_t                                              | ptr        | local pointer of the matrix                                                                                                                |
| uintptr_t                                              | dev_handle | device handle of the matrix.                                                                                                               |
| size_t                                                 | offset     | offset in the matrix                                                                                                                       |
| uint32_t                                               | nx         | number of elements on the x-axis of the matrix                                                                                             |
| uint32_t                                               | ny         | number of elements on the y-axis of the matrix                                                                                             |
| uint32_t                                               | ld         | number of elements between each row of the matrix. Maybe be equal to <a href="#">starpu_matrix_interface::nx</a> when there is no padding. |
| size_t                                                 | elemsize   | size of the elements of the matrix                                                                                                         |

#### 27.9.2.6 struct starpu\_block\_interface

Block interface for 3D dense blocks

Data Fields

|                                                        |            |                                                |
|--------------------------------------------------------|------------|------------------------------------------------|
| enum<br><a href="#">starpu_data_↔<br/>interface_id</a> | id         | identifier of the interface                    |
| uintptr_t                                              | ptr        | local pointer of the block                     |
| uintptr_t                                              | dev_handle | device handle of the block.                    |
| size_t                                                 | offset     | offset in the block.                           |
| uint32_t                                               | nx         | number of elements on the x-axis of the block. |
| uint32_t                                               | ny         | number of elements on the y-axis of the block. |
| uint32_t                                               | nz         | number of elements on the z-axis of the block. |
| uint32_t                                               | ldy        | number of elements between two lines           |
| uint32_t                                               | ldz        | number of elements between two planes          |
| size_t                                                 | elemsize   | size of the elements of the block.             |

#### 27.9.2.7 struct starpu\_bcsr\_interface

BCSR interface for sparse matrices (blocked compressed sparse row representation)

Data Fields

|                                                        |    |                             |
|--------------------------------------------------------|----|-----------------------------|
| enum<br><a href="#">starpu_data_↔<br/>interface_id</a> | id | Identifier of the interface |
|--------------------------------------------------------|----|-----------------------------|

|            |            |                                                                                     |
|------------|------------|-------------------------------------------------------------------------------------|
| uint32_t   | nnz        | number of non-zero BLOCKS                                                           |
| uint32_t   | nrow       | number of rows (in terms of BLOCKS)                                                 |
| uintptr_t  | nzval      | non-zero values                                                                     |
| uint32_t * | colind     | position of non-zero entried on the row                                             |
| uint32_t * | rowptr     | index (in nzval) of the first entry of the row                                      |
| uint32_t   | firstentry | k for k-based indexing (0 or 1 usually). Also useful when partitionning the matrix. |
| uint32_t   | r          | size of the blocks                                                                  |
| uint32_t   | c          | size of the blocks                                                                  |
| size_t     | elemsize   | size of the elements of the matrix                                                  |

#### 27.9.2.8 struct starpu\_csr\_interface

CSR interface for sparse matrices (compressed sparse row representation)

##### Data Fields

|                                       |            |                                                                                     |
|---------------------------------------|------------|-------------------------------------------------------------------------------------|
| enum<br>starpu_data_↔<br>interface_id | id         | Identifier of the interface                                                         |
| uint32_t                              | nnz        | number of non-zero entries                                                          |
| uint32_t                              | nrow       | number of rows                                                                      |
| uintptr_t                             | nzval      | non-zero values                                                                     |
| uint32_t *                            | colind     | position of non-zero entries on the row                                             |
| uint32_t *                            | rowptr     | index (in nzval) of the first entry of the row                                      |
| uint32_t                              | firstentry | k for k-based indexing (0 or 1 usually). also useful when partitionning the matrix. |
| size_t                                | elemsize   | size of the elements of the matrix                                                  |

#### 27.9.2.9 struct starpu\_coo\_interface

COO Matrices

##### Data Fields

|                                       |          |                                                |
|---------------------------------------|----------|------------------------------------------------|
| enum<br>starpu_data_↔<br>interface_id | id       | identifier of the interface                    |
| uint32_t *                            | columns  | column array of the matrix                     |
| uint32_t *                            | rows     | row array of the matrix                        |
| uintptr_t                             | values   | values of the matrix                           |
| uint32_t                              | nx       | number of elements on the x-axis of the matrix |
| uint32_t                              | ny       | number of elements on the y-axis of the matrix |
| uint32_t                              | n_values | number of values registered in the matrix      |
| size_t                                | elemsize | size of the elements of the matrix             |

### 27.9.3 Macro Definition Documentation

#### 27.9.3.1 #define STARPU\_VARIABLE\_GET\_PTR( *interface* )

Return a pointer to the variable designated by *interface*.

#### 27.9.3.2 #define STARPU\_VARIABLE\_GET\_ELEMSIZE( *interface* )

Return the size of the variable designated by *interface*.



**27.9.3.3 #define STARPU\_VARIABLE\_GET\_DEV\_HANDLE( *interface* )**

Return a device handle for the variable designated by *interface*, to be used on OpenCL. The offset documented below has to be used in addition to this.

**27.9.3.4 #define STARPU\_VARIABLE\_GET\_OFFSET( *interface* )**

Return the offset in the variable designated by *interface*, to be used with the device handle.

**27.9.3.5 #define STARPU\_VECTOR\_GET\_PTR( *interface* )**

Return a pointer to the array designated by *interface*, valid on CPUs and CUDA only. For OpenCL, the device handle and offset need to be used instead.

**27.9.3.6 #define STARPU\_VECTOR\_GET\_DEV\_HANDLE( *interface* )**

Return a device handle for the array designated by *interface*, to be used on OpenCL. the offset documented below has to be used in addition to this.

**27.9.3.7 #define STARPU\_VECTOR\_GET\_OFFSET( *interface* )**

Return the offset in the array designated by *interface*, to be used with the device handle.

**27.9.3.8 #define STARPU\_VECTOR\_GET\_NX( *interface* )**

Return the number of elements registered into the array designated by *interface*.

**27.9.3.9 #define STARPU\_VECTOR\_GET\_ELEMSIZE( *interface* )**

Return the size of each element of the array designated by *interface*.

**27.9.3.10 #define STARPU\_VECTOR\_GET\_SLICE\_BASE( *interface* )**

Return the OpenMP slice base annotation of each element of the array designated by *interface*.

See also

[starpu\\_omp\\_vector\\_annotate](#)

**27.9.3.11 #define STARPU\_MATRIX\_GET\_PTR( *interface* )**

Return a pointer to the matrix designated by *interface*, valid on CPUs and CUDA devices only. For OpenCL devices, the device handle and offset need to be used instead.

**27.9.3.12 #define STARPU\_MATRIX\_GET\_DEV\_HANDLE( *interface* )**

Return a device handle for the matrix designated by *interface*, to be used on OpenCL. The offset documented below has to be used in addition to this.

**27.9.3.13 #define STARPU\_MATRIX\_GET\_OFFSET( *interface* )**

Return the offset in the matrix designated by *interface*, to be used with the device handle.

**27.9.3.14 #define STARPU\_MATRIX\_GET\_NX( *interface* )**

Return the number of elements on the x-axis of the matrix designated by *interface*.

**27.9.3.15 #define STARPU\_MATRIX\_GET\_NY( *interface* )**

Return the number of elements on the y-axis of the matrix designated by *interface*.

**27.9.3.16 #define STARPU\_MATRIX\_GET\_LD( *interface* )**

Return the number of elements between each row of the matrix designated by *interface*. May be equal to nx when there is no padding.

**27.9.3.17 #define STARPU\_MATRIX\_GET\_ELEMSIZE( *interface* )**

Return the size of the elements registered into the matrix designated by *interface*.

**27.9.3.18 #define STARPU\_BLOCK\_GET\_PTR( *interface* )**

Return a pointer to the block designated by *interface*.

**27.9.3.19 #define STARPU\_BLOCK\_GET\_DEV\_HANDLE( *interface* )**

Return a device handle for the block designated by *interface*, to be used on OpenCL. The offset document below has to be used in addition to this.

**27.9.3.20 #define STARPU\_BLOCK\_GET\_OFFSET( *interface* )**

Return the offset in the block designated by *interface*, to be used with the device handle.

**27.9.3.21 #define STARPU\_BLOCK\_GET\_NX( *interface* )**

Return the number of elements on the x-axis of the block designated by *interface*.

**27.9.3.22 #define STARPU\_BLOCK\_GET\_NY( *interface* )**

Return the number of elements on the y-axis of the block designated by *interface*.

**27.9.3.23 #define STARPU\_BLOCK\_GET\_NZ( *interface* )**

Return the number of elements on the z-axis of the block designated by *interface*.

**27.9.3.24 #define STARPU\_BLOCK\_GET\_LDY( *interface* )**

Return the number of elements between each row of the block designated by *interface*. May be equal to nx when there is no padding.

**27.9.3.25 #define STARPU\_BLOCK\_GET\_LDZ( *interface* )**

Return the number of elements between each z plane of the block designated by *interface*. May be equal to  $nx \times ny$  when there is no padding.

**27.9.3.26 #define STARPU\_BLOCK\_GET\_ELEMSIZE( *interface* )**

Return the size of the elements of the block designated by *interface*.

**27.9.3.27 #define STARPU\_BCSR\_GET\_NNZ( *interface* )**

Return the number of non-zero values in the matrix designated by *interface*.

**27.9.3.28 #define STARPU\_BCSR\_GET\_NZVAL( *interface* )**

Return a pointer to the non-zero values of the matrix designated by *interface*.

**27.9.3.29 #define STARPU\_BCSR\_GET\_NZVAL\_DEV\_HANDLE( *interface* )**

Return a device handle for the array of non-zero values in the matrix designated by *interface*. The offset documented below has to be used in addition to this.

**27.9.3.30 #define STARPU\_BCSR\_GET\_COLIND( *interface* )**

Return a pointer to the column index of the matrix designated by *interface*.

**27.9.3.31 #define STARPU\_BCSR\_GET\_COLIND\_DEV\_HANDLE( *interface* )**

Return a device handle for the column index of the matrix designated by *interface*. The offset documented below has to be used in addition to this.

**27.9.3.32 #define STARPU\_BCSR\_GET\_ROWPTR( *interface* )**

Return a pointer to the row pointer array of the matrix designated by *interface*.

**27.9.3.33 #define STARPU\_BCSR\_GET\_ROWPTR\_DEV\_HANDLE( *interface* )**

Return a device handle for the row pointer array of the matrix designated by *interface*. The offset documented below has to be used in addition to this.

**27.9.3.34 #define STARPU\_BCSR\_GET\_OFFSET**

Return the offset in the arrays (colind, rowptr, nzval) of the matrix designated by *interface*, to be used with the device handles.

**27.9.3.35 #define STARPU\_CSR\_GET\_NNZ( *interface* )**

Return the number of non-zero values in the matrix designated by *interface*.

**27.9.3.36 #define STARPU\_CSR\_GET\_NROW( *interface* )**

Return the size of the row pointer array of the matrix designated by *interface*.

**27.9.3.37 #define STARPU\_CSR\_GET\_NZVAL( *interface* )**

Return a pointer to the non-zero values of the matrix designated by *interface*.

**27.9.3.38 #define STARPU\_CSR\_GET\_NZVAL\_DEV\_HANDLE( *interface* )**

Return a device handle for the array of non-zero values in the matrix designated by *interface*. The offset documented below has to be used in addition to this.

**27.9.3.39 #define STARPU\_CSR\_GET\_COLIND( *interface* )**

Return a pointer to the column index of the matrix designated by *interface*.

**27.9.3.40 #define STARPU\_CSR\_GET\_COLIND\_DEV\_HANDLE( *interface* )**

Return a device handle for the column index of the matrix designated by *interface*. The offset documented below has to be used in addition to this.

**27.9.3.41 #define STARPU\_CSR\_GET\_ROWPTR( *interface* )**

Return a pointer to the row pointer array of the matrix designated by *interface*.

**27.9.3.42 #define STARPU\_CSR\_GET\_ROWPTR\_DEV\_HANDLE( *interface* )**

Return a device handle for the row pointer array of the matrix designated by *interface*. The offset documented below has to be used in addition to this.

**27.9.3.43 #define STARPU\_CSR\_GET\_OFFSET**

Return the offset in the arrays (colind, rowptr, nzval) of the matrix designated by *interface*, to be used with the device handles.

**27.9.3.44 #define STARPU\_CSR\_GET\_FIRSTENTRY( *interface* )**

Return the index at which all arrays (the column indexes, the row pointers...) of the *interface* start.

**27.9.3.45 #define STARPU\_CSR\_GET\_ELEMSIZE( *interface* )**

Return the size of the elements registered into the matrix designated by *interface*.

**27.9.3.46 #define STARPU\_COO\_GET\_COLUMNS( *interface* )**

Return a pointer to the column array of the matrix designated by *interface*.

**27.9.3.47 #define STARPU\_COO\_GET\_COLUMNS\_DEV\_HANDLE( *interface* )**

Return a device handle for the column array of the matrix designated by *interface*, to be used on OpenCL. The offset documented below has to be used in addition to this.

**27.9.3.48 #define STARPU\_COO\_GET\_ROWS( *interface* )**

Return a pointer to the rows array of the matrix designated by *interface*.

**27.9.3.49 #define STARPU\_COO\_GET\_ROWS\_DEV\_HANDLE( *interface* )**

Return a device handle for the row array of the matrix designated by *interface*, to be used on OpenCL. The offset documented below has to be used in addition to this.

**27.9.3.50 #define STARPU\_COO\_GET\_VALUES( *interface* )**

Return a pointer to the values array of the matrix designated by *interface*.

**27.9.3.51 #define STARPU\_COO\_GET\_VALUES\_DEV\_HANDLE( *interface* )**

Return a device handle for the value array of the matrix designated by *interface*, to be used on OpenCL. The offset documented below has to be used in addition to this.

**27.9.3.52 #define STARPU\_COO\_GET\_OFFSET**

Return the offset in the arrays of the COO matrix designated by *interface*.

**27.9.3.53 #define STARPU\_COO\_GET\_NX( *interface* )**

Return the number of elements on the x-axis of the matrix designated by *interface*.

**27.9.3.54 #define STARPU\_COO\_GET\_NY( *interface* )**

Return the number of elements on the y-axis of the matrix designated by *interface*.

**27.9.3.55 #define STARPU\_COO\_GET\_NVALUES( *interface* )**

Return the number of values registered in the matrix designated by *interface*.

**27.9.3.56 #define STARPU\_COO\_GET\_ELEMSIZE( *interface* )**

Return the size of the elements registered into the matrix designated by *interface*.

**27.9.4 Enumeration Type Documentation****27.9.4.1 enum starpu\_data\_interface\_id**

Identifier for all predefined StarPU data interfaces

## Enumerator

**STARPU\_UNKNOWN\_INTERFACE\_ID** Unknown interface

**STARPU\_MATRIX\_INTERFACE\_ID** Identifier for the matrix data interface

**STARPU\_BLOCK\_INTERFACE\_ID** Identifier for block data interface

**STARPU\_VECTOR\_INTERFACE\_ID** Identifier for the vector data interface

**STARPU\_CSR\_INTERFACE\_ID** Identifier for the csr data interface

**STARPU\_BCSR\_INTERFACE\_ID** Identifier for the bcsr data interface

**STARPU\_VARIABLE\_INTERFACE\_ID** Identifier for the variable data interface

**STARPU\_VOID\_INTERFACE\_ID** Identifier for the void data interface

**STARPU\_MULTIFORMAT\_INTERFACE\_ID** Identifier for the multiformat data interface

**STARPU\_COO\_INTERFACE\_ID** Identifier for the coo data interface

**STARPU\_MAX\_INTERFACE\_ID** Maximum number of data interfaces

## 27.9.5 Function Documentation

## 27.9.5.1 void starpu\_void\_data\_register ( starpu\_data\_handle\_t \* handle )

Register a void interface. There is no data really associated to that interface, but it may be used as a synchronization mechanism. It also permits to express an abstract piece of data that is managed by the application internally: this makes it possible to forbid the concurrent execution of different tasks accessing the same `void` data in read-write concurrently.

## 27.9.5.2 void starpu\_variable\_data\_register ( starpu\_data\_handle\_t \* handle, unsigned home\_node, uintptr\_t ptr, size\_t size )

Register the `size` byte element pointed to by `ptr`, which is typically a scalar, and initialize `handle` to represent this data item.

Here an example of how to use the function.

```
float var;
starpu_data_handle_t var_handle;
starpu_variable_data_register(&var_handle,
 STARPU_MAIN_RAM, (uintptr_t)&var, sizeof(var));
```

## 27.9.5.3 void starpu\_variable\_ptr\_register ( starpu\_data\_handle\_t handle, unsigned node, uintptr\_t ptr, uintptr\_t dev\_handle, size\_t offset )

Register into the `handle` that to store data on `node` it should use the buffer located at `ptr`, or device handle `dev_handle` and offset `offset` (for OpenCL, notably)

## 27.9.5.4 void starpu\_vector\_data\_register ( starpu\_data\_handle\_t \* handle, unsigned home\_node, uintptr\_t ptr, uint32\_t nx, size\_t elemsize )

Register the `nx` `elemsize`-byte elements pointed to by `ptr` and initialize `handle` to represent it.

Here an example of how to use the function.

```
float vector[NX];
starpu_data_handle_t vector_handle;
starpu_vector_data_register(&vector_handle,
 STARPU_MAIN_RAM, (uintptr_t)vector, NX, sizeof(vector[0]));
```

**27.9.5.5** `void starpu_vector_ptr_register ( starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_handle, size_t offset )`

Register into the `handle` that to store data on node `node` it should use the buffer located at `ptr`, or device handle `dev_handle` and offset `offset` (for OpenCL, notably)

**27.9.5.6** `void starpu_matrix_data_register ( starpu_data_handle_t * handle, unsigned home_node, uintptr_t ptr, uint32_t ld, uint32_t nx, uint32_t ny, size_t elemsize )`

Register the  $n_x \times n_y$  2D matrix of `elemsize`-byte elements pointed by `ptr` and initialize `handle` to represent it. `ld` specifies the number of elements between rows. a value greater than `nx` adds padding, which can be useful for alignment purposes.

Here an example of how to use the function.

```
float *matrix;
starpu_data_handle_t matrix_handle;
matrix = (float*)malloc(width * height * sizeof(float));
starpu_matrix_data_register(&matrix_handle,
 STARPU_MAIN_RAM, (uintptr_t)matrix, width, width, height, sizeof(float));
```

**27.9.5.7** `void starpu_matrix_ptr_register ( starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_handle, size_t offset, uint32_t ld )`

Register into the `handle` that to store data on node `node` it should use the buffer located at `ptr`, or device handle `dev_handle` and offset `offset` (for OpenCL, notably), with `ld` elements between rows.

**27.9.5.8** `void starpu_block_data_register ( starpu_data_handle_t * handle, unsigned home_node, uintptr_t ptr, uint32_t ld, uint32_t ldz, uint32_t nx, uint32_t ny, uint32_t nz, size_t elemsize )`

Register the  $n_x \times n_y \times n_z$  3D matrix of `elemsize` byte elements pointed by `ptr` and initialize `handle` to represent it. Again, `ldy` and `ldz` specify the number of elements between rows and between z planes.

Here an example of how to use the function.

```
float *block;
starpu_data_handle_t block_handle;
block = (float*)malloc(nx*ny*nz*sizeof(float));
starpu_block_data_register(&block_handle,
 STARPU_MAIN_RAM, (uintptr_t)block, nx, nx*ny, nx, ny, nz, sizeof(float));
```

**27.9.5.9** `void starpu_block_ptr_register ( starpu_data_handle_t handle, unsigned node, uintptr_t ptr, uintptr_t dev_handle, size_t offset, uint32_t ldy, uint32_t ldz )`

Register into the `handle` that to store data on node `node` it should use the buffer located at `ptr`, or device handle `dev_handle` and offset `offset` (for OpenCL, notably), with `ldy` elements between rows and `ldz` elements between z planes.

**27.9.5.10** `void starpu_bcsr_data_register ( starpu_data_handle_t * handle, unsigned home_node, uint32_t nnz, uint32_t nrow, uintptr_t nzval, uint32_t * colind, uint32_t * rowptr, uint32_t firstentry, uint32_t r, uint32_t c, size_t elemsize )`

This variant of `starpu_data_register()` uses the BCSR (Blocked Compressed Sparse Row Representation) sparse matrix interface. Register the sparse matrix made of `nnz` non-zero blocks of elements of size `elemsize` stored in `nzval` and initializes `handle` to represent it. Blocks have size `r * c`. `nrow` is the number of rows (in terms of blocks), `colind[i]` is the block-column index for block `i` in `nzval`, `rowptr[i]` is the block-index (in `nzval`) of the first block of row `i`. `firstentry` is the index of the first entry of the given arrays (usually 0 or 1).

27.9.5.11 `void starpu_csr_data_register ( starpu_data_handle_t * handle, unsigned home_node, uint32_t nnz, uint32_t nrow, uintptr_t nzval, uint32_t * colind, uint32_t * rowptr, uint32_t firstentry, size_t elemsize )`

This variant of [starpu\\_data\\_register\(\)](#) uses the CSR (Compressed Sparse Row Representation) sparse matrix interface. TODO

27.9.5.12 `void starpu_coo_data_register ( starpu_data_handle_t * handleptr, unsigned home_node, uint32_t nx, uint32_t ny, uint32_t n_values, uint32_t * columns, uint32_t * rows, uintptr_t values, size_t elemsize )`

Register the  $n_x \times n_y$  2D matrix given in the COO format, using the `columns`, `rows`, `values` arrays, which must have `n_values` elements of size `elemsize`. Initialize `handleptr`.

27.9.5.13 `void * starpu_data_get_interface_on_node ( starpu_data_handle_t handle, unsigned memory_node )`

Return the interface associated with `handle` on `memory_node`.

27.9.5.14 `void * starpu_data_handle_to_pointer ( starpu_data_handle_t handle, unsigned node )`

Return the pointer associated with `handle` on node `node` or NULL if `handle`'s interface does not support this operation or data for this `handle` is not allocated on that `node`.

27.9.5.15 `void * starpu_data_get_local_ptr ( starpu_data_handle_t handle )`

Return the local pointer associated with `handle` or NULL if `handle`'s interface does not have data allocated locally

27.9.5.16 `enum starpu_data_interface_id starpu_data_get_interface_id ( starpu_data_handle_t handle )`

Return the unique identifier of the interface associated with the given `handle`.

27.9.5.17 `size_t starpu_data_get_size ( starpu_data_handle_t handle )`

Return the size of the data associated with `handle`.

27.9.5.18 `int starpu_data_pack ( starpu_data_handle_t handle, void ** ptr, starpu_ssize_t * count )`

Execute the packing operation of the interface of the data registered at `handle` (see [starpu\\_data\\_interface\\_ops](#)). This packing operation must allocate a buffer large enough at `ptr` and copy into the newly allocated buffer the data associated to `handle`. `count` will be set to the size of the allocated buffer. If `ptr` is NULL, the function should not copy the data in the buffer but just set `count` to the size of the buffer which would have been allocated. The special value -1 indicates the size is yet unknown.

27.9.5.19 `int starpu_data_unpack ( starpu_data_handle_t handle, void * ptr, size_t count )`

Unpack in `handle` the data located at `ptr` of size `count` as described by the interface of the data. The interface registered at `handle` must define a unpacking operation (see [starpu\\_data\\_interface\\_ops](#)). The memory at the address `ptr` is freed after calling the data unpacking operation.

27.9.5.20 `size_t starpu_variable_get_elemsize ( starpu_data_handle_t handle )`

Return the size of the variable designated by `handle`.



27.9.5.21 `uintptr_t starpu_variable_get_local_ptr ( starpu_data_handle_t handle )`

Return a pointer to the variable designated by `handle`.

27.9.5.22 `uint32_t starpu_vector_get_nx ( starpu_data_handle_t handle )`

Return the number of elements registered into the array designated by `handle`.

27.9.5.23 `size_t starpu_vector_get_elemsize ( starpu_data_handle_t handle )`

Return the size of each element of the array designated by `handle`.

27.9.5.24 `uintptr_t starpu_vector_get_local_ptr ( starpu_data_handle_t handle )`

Return the local pointer associated with `handle`.

27.9.5.25 `uint32_t starpu_matrix_get_nx ( starpu_data_handle_t handle )`

Return the number of elements on the x-axis of the matrix designated by `handle`.

27.9.5.26 `uint32_t starpu_matrix_get_ny ( starpu_data_handle_t handle )`

Return the number of elements on the y-axis of the matrix designated by `handle`.

27.9.5.27 `uint32_t starpu_matrix_get_local_ld ( starpu_data_handle_t handle )`

Return the number of elements between each row of the matrix designated by `handle`. Maybe be equal to `nx` when there is no padding.

27.9.5.28 `uintptr_t starpu_matrix_get_local_ptr ( starpu_data_handle_t handle )`

Return the local pointer associated with `handle`.

27.9.5.29 `size_t starpu_matrix_get_elemsize ( starpu_data_handle_t handle )`

Return the size of the elements registered into the matrix designated by `handle`.

27.9.5.30 `uint32_t starpu_block_get_nx ( starpu_data_handle_t handle )`

Return the number of elements on the x-axis of the block designated by `handle`.

27.9.5.31 `uint32_t starpu_block_get_ny ( starpu_data_handle_t handle )`

Return the number of elements on the y-axis of the block designated by `handle`.

27.9.5.32 `uint32_t starpu_block_get_nz ( starpu_data_handle_t handle )`

Return the number of elements on the z-axis of the block designated by `handle`.

27.9.5.33 `uint32_t starpu_block_get_local_ldy ( starpu_data_handle_t handle )`

Return the number of elements between each row of the block designated by `handle`, in the format of the current memory node.

27.9.5.34 `uint32_t starpu_block_get_local_ldz ( starpu_data_handle_t handle )`

Return the number of elements between each z plane of the block designated by `handle`, in the format of the current memory node.

27.9.5.35 `uintptr_t starpu_block_get_local_ptr ( starpu_data_handle_t handle )`

Return the local pointer associated with `handle`.

27.9.5.36 `size_t starpu_block_get_elemsize ( starpu_data_handle_t handle )`

Return the size of the elements of the block designated by `handle`.

27.9.5.37 `uint32_t starpu_bcsr_get_nnz ( starpu_data_handle_t handle )`

Return the number of non-zero elements in the matrix designated by `handle`.

27.9.5.38 `uint32_t starpu_bcsr_get_nrow ( starpu_data_handle_t handle )`

Return the number of rows (in terms of blocks of size  $r \times c$ ) in the matrix designated by `handle`.

27.9.5.39 `uint32_t starpu_bcsr_get_firstentry ( starpu_data_handle_t handle )`

Return the index at which all arrays (the column indexes, the row pointers...) of the matrix designated by `handle`.

27.9.5.40 `uintptr_t starpu_bcsr_get_local_nzval ( starpu_data_handle_t handle )`

Return a pointer to the non-zero values of the matrix designated by `handle`.

27.9.5.41 `uint32_t * starpu_bcsr_get_local_colind ( starpu_data_handle_t handle )`

Return a pointer to the column index, which holds the positions of the non-zero entries in the matrix designated by `handle`.

27.9.5.42 `uint32_t * starpu_bcsr_get_local_rowptr ( starpu_data_handle_t handle )`

Return the row pointer array of the matrix designated by `handle`.

27.9.5.43 `uint32_t starpu_bcsr_get_r ( starpu_data_handle_t handle )`

Return the number of rows in a block.

27.9.5.44 `uint32_t starpu_bcsr_get_c ( starpu_data_handle_t handle )`

Return the number of columns in a block.

27.9.5.45 `size_t starpu_bcsr_get_elemsize ( starpu_data_handle_t handle )`

Return the size of the elements in the matrix designated by `handle`.

27.9.5.46 `uint32_t starpu_csr_get_nnz ( starpu_data_handle_t handle )`

Return the number of non-zero values in the matrix designated by `handle`.

27.9.5.47 `uint32_t starpu_csr_get_nrow ( starpu_data_handle_t handle )`

Return the size of the row pointer array of the matrix designated by `handle`.

27.9.5.48 `uint32_t starpu_csr_get_firstentry ( starpu_data_handle_t handle )`

Return the index at which all arrays (the column indexes, the row pointers...) of the matrix designated by `handle`.

27.9.5.49 `uintptr_t starpu_csr_get_local_nzval ( starpu_data_handle_t handle )`

Return a local pointer to the non-zero values of the matrix designated by `handle`.

27.9.5.50 `uint32_t * starpu_csr_get_local_colind ( starpu_data_handle_t handle )`

Return a local pointer to the column index of the matrix designated by `handle`.

27.9.5.51 `uint32_t * starpu_csr_get_local_rowptr ( starpu_data_handle_t handle )`

Return a local pointer to the row pointer array of the matrix designated by `handle`.

27.9.5.52 `size_t starpu_csr_get_elemsize ( starpu_data_handle_t handle )`

Return the size of the elements registered into the matrix designated by `handle`.

27.9.5.53 `uintptr_t starpu_malloc_on_node ( unsigned dst_node, size_t size )`

Allocate `size` bytes on node `dst_node`. This returns 0 if allocation failed, the allocation method should then return `-ENOMEM` as allocated size. Deallocation must be done with `starpu_free_on_node`.

27.9.5.54 `void starpu_free_on_node ( unsigned dst_node, uintptr_t addr, size_t size )`

Free `addr` of `size` bytes on node `dst_node` which was previously allocated with `starpu_malloc_on_node`.

27.9.5.55 `int starpu_interface_copy ( uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t size, void * async_data )`

Copy `size` bytes from byte offset `src_offset` of `src` on `src_node` to byte offset `dst_offset` of `dst` on `dst_node`. This is to be used in the `any_to_any()` copy method, which is provided with the `async_data` to be passed to `starpu_interface_copy()`. this returns `-EAGAIN` if the transfer is still ongoing, or 0 if the transfer is already completed.

27.9.5.56 `uint32_t starpu_hash_crc32c_be_n ( const void * input, size_t n, uint32_t inputcrc )`

Compute the CRC of a byte buffer seeded by the `inputcrc` *current state*. The return value should be considered as the new *current state* for future CRC computation. This is used for computing data size footprint.

27.9.5.57 `uint32_t starpu_hash_crc32c_be ( uint32_t input, uint32_t inputcrc )`

Compute the CRC of a 32bit number seeded by the `inputcrc` *current state*. The return value should be considered as the new *current state* for future CRC computation. This is used for computing data size footprint.

27.9.5.58 `uint32_t starpu_hash_crc32c_string ( const char * str, uint32_t inputcrc )`

Compute the CRC of a string seeded by the `inputcrc` *current state*. The return value should be considered as the new *current state* for future CRC computation. This is used for computing data size footprint.

27.9.5.59 `int starpu_data_interface_get_next_id ( void )`

Return the next available id for a newly created data interface ([Defining A New Data Interface](#)).

## 27.10 Data Partition

### Data Structures

- struct [starpu\\_data\\_filter](#)

### Basic API

- void [starpu\\_data\\_partition](#) ([starpu\\_data\\_handle\\_t](#) initial\_handle, struct [starpu\\_data\\_filter](#) \*f)
- void [starpu\\_data\\_unpartition](#) ([starpu\\_data\\_handle\\_t](#) root\_data, unsigned gathering\_node)
- int [starpu\\_data\\_get\\_nb\\_children](#) ([starpu\\_data\\_handle\\_t](#) handle)
- [starpu\\_data\\_handle\\_t](#) [starpu\\_data\\_get\\_child](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned i)
- [starpu\\_data\\_handle\\_t](#) [starpu\\_data\\_get\\_sub\\_data](#) ([starpu\\_data\\_handle\\_t](#) root\_data, unsigned depth,...)
- [starpu\\_data\\_handle\\_t](#) [starpu\\_data\\_vget\\_sub\\_data](#) ([starpu\\_data\\_handle\\_t](#) root\_data, unsigned depth, va\_list pa)
- void [starpu\\_data\\_map\\_filters](#) ([starpu\\_data\\_handle\\_t](#) root\_data, unsigned nfilters,...)
- void [starpu\\_data\\_vmap\\_filters](#) ([starpu\\_data\\_handle\\_t](#) root\_data, unsigned nfilters, va\_list pa)

### Predefined Vector Filter Functions

This section gives a partial list of the predefined partitioning functions for vector data. Examples on how to use them are shown in [Partitioning Data](#). The complete list can be found in the file [starpu\\_data\\_filters.h](#).

- void [starpu\\_vector\\_filter\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_vector\\_filter\\_block\\_shadow](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_vector\\_filter\\_list](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_vector\\_filter\\_divide\\_in\\_2](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)

## Predefined Matrix Filter Functions

This section gives a partial list of the predefined partitioning functions for matrix data. Examples on how to use them are shown in [Partitioning Data](#). The complete list can be found in the file [starpu\\_data\\_filters.h](#).

- void [starpu\\_matrix\\_filter\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_matrix\\_filter\\_block\\_shadow](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_matrix\\_filter\\_vertical\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_matrix\\_filter\\_vertical\\_block\\_shadow](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)

## Predefined Block Filter Functions

This section gives a partial list of the predefined partitioning functions for block data. Examples on how to use them are shown in [Partitioning Data](#). The complete list can be found in the file [starpu\\_data\\_filters.h](#). A usage example is available in [examples/filters/shadow3d.c](#)

- void [starpu\\_block\\_filter\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_block\\_filter\\_block\\_shadow](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_block\\_filter\\_vertical\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_block\\_filter\\_vertical\\_block\\_shadow](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_block\\_filter\\_depth\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_block\\_filter\\_depth\\_block\\_shadow](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)

## Predefined BCSR Filter Functions

This section gives a partial list of the predefined partitioning functions for BCSR data. Examples on how to use them are shown in [Partitioning Data](#). The complete list can be found in the file [starpu\\_data\\_filters.h](#).

- void [starpu\\_bcsr\\_filter\\_canonical\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_csr\\_filter\\_vertical\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)

### 27.10.1 Detailed Description

### 27.10.2 Data Structure Documentation

#### 27.10.2.1 struct starpu\_data\_filter

The filter structure describes a data partitioning operation, to be given to the [starpu\\_data\\_partition\(\)](#) function.

## Data Fields

- void(\* [filter\\_func](#) )(void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*, unsigned id, unsigned nparts)
- unsigned [nchildren](#)
- unsigned(\* [get\\_nchildren](#) )(struct [starpu\\_data\\_filter](#) \*, [starpu\\_data\\_handle\\_t](#) initial\_handle)
- struct  
    [starpu\\_data\\_interface\\_ops](#) \*(\* [get\\_child\\_ops](#) )(struct [starpu\\_data\\_filter](#) \*, unsigned id)
- unsigned [filter\\_arg](#)
- void \* [filter\\_arg\\_ptr](#)

### 27.10.2.1.1 Field Documentation

**27.10.2.1.1.1** void(\* [starpu\\_data\\_filter::filter\\_func](#))(void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*, unsigned id, unsigned nparts)

This function fills the `child_interface` structure with interface information for the `id`-th child of the parent `father_↔` interface (among `nparts`).

**27.10.2.1.1.2** unsigned [starpu\\_data\\_filter::nchildren](#)

This is the number of parts to partition the data into.

**27.10.2.1.1.3** unsigned(\* [starpu\\_data\\_filter::get\\_nchildren](#))(struct [starpu\\_data\\_filter](#) \*, [starpu\\_data\\_handle\\_t](#) initial\_handle)

This returns the number of children. This can be used instead of `nchildren` when the number of children depends on the actual data (e.g. the number of blocks in a sparse matrix).

**27.10.2.1.1.4** struct [starpu\\_data\\_interface\\_ops](#) \*(\* [starpu\\_data\\_filter::get\\_child\\_ops](#))(struct [starpu\\_data\\_filter](#) \*, unsigned id)

In case the resulting children use a different data interface, this function returns which interface is used by child number `id`.

**27.10.2.1.1.5** unsigned [starpu\\_data\\_filter::filter\\_arg](#)

Allow to define an additional parameter for the filter function.

**27.10.2.1.1.6** void \* [starpu\\_data\\_filter::filter\\_arg\\_ptr](#)

Allow to define an additional pointer parameter for the filter function, such as the sizes of the different parts.

## 27.10.3 Function Documentation

**27.10.3.1** void [starpu\\_data\\_partition](#) ( [starpu\\_data\\_handle\\_t](#) *initial\_handle*, struct [starpu\\_data\\_filter](#) \* *f* )

This requests partitioning one StarPU data `initial_handle` into several subdata according to the filter `f`.

Here an example of how to use the function.

```
struct starpu_data_filter f = {
 .filter_func = starpu_matrix_filter_block,
 .nchildren = nslicesx,
 .get_nchildren = NULL,
 .get_child_ops = NULL
};
starpu_data_partition(A_handle, &f);
```

### 27.10.3.2 void starpu\_data\_unpartition ( starpu\_data\_handle\_t root\_data, unsigned gathering\_node )

This unapplies one filter, thus unpartitioning the data. The pieces of data are collected back into one big piece in the `gathering_node` (usually `STARPU_MAIN_RAM`). Tasks working on the partitioned data must be already finished when calling `starpu_data_unpartition()`.

Here an example of how to use the function.

```
starpu_data_unpartition(A_handle, STARPU_MAIN_RAM);
```

### 27.10.3.3 int starpu\_data\_get\_nb\_children ( starpu\_data\_handle\_t handle )

This function returns the number of children.

### 27.10.3.4 starpu\_data\_handle\_t starpu\_data\_get\_child ( starpu\_data\_handle\_t handle, unsigned i )

Return the *i*th child of the given `handle`, which must have been partitionned beforehand.

### 27.10.3.5 starpu\_data\_handle\_t starpu\_data\_get\_sub\_data ( starpu\_data\_handle\_t root\_data, unsigned depth, ... )

After partitioning a StarPU data by applying a filter, `starpu_data_get_sub_data()` can be used to get handles for each of the data portions. `root_data` is the parent data that was partitioned. `depth` is the number of filters to traverse (in case several filters have been applied, to e.g. partition in row blocks, and then in column blocks), and the subsequent parameters are the indexes. The function returns a handle to the subdata.

Here an example of how to use the function.

```
h = starpu_data_get_sub_data(A_handle, 1, taskx);
```

### 27.10.3.6 starpu\_data\_handle\_t starpu\_data\_vget\_sub\_data ( starpu\_data\_handle\_t root\_data, unsigned depth, va\_list pa )

This function is similar to `starpu_data_get_sub_data()` but uses a `va_list` for the parameter list.

### 27.10.3.7 void starpu\_data\_map\_filters ( starpu\_data\_handle\_t root\_data, unsigned nfilters, ... )

Applies `nfilters` filters to the handle designated by `root_handle` recursively. `nfilters` pointers to variables of the type `starpu_data_filter` should be given.

### 27.10.3.8 void starpu\_data\_vmap\_filters ( starpu\_data\_handle\_t root\_data, unsigned nfilters, va\_list pa )

Applies `nfilters` filters to the handle designated by `root_handle` recursively. It uses a `va_list` of pointers to variables of the type `starpu_data_filter`.

### 27.10.3.9 void starpu\_vector\_filter\_block ( void \* father\_interface, void \* child\_interface, struct starpu\_data\_filter \* f, unsigned id, unsigned nparts )

Return in `child_interface` the `id` th element of the vector represented by `father_interface` once partitioned in `nparts` chunks of equal size.

27.10.3.10 `void starpu_vector_filter_block_shadow ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

Return in `child_interface` the `id` th element of the vector represented by `father_interface` once partitioned in `nparts` chunks of equal size with a shadow border `filter_arg_ptr`, thus getting a vector of size  $(n-2*\text{shadow})/\text{nparts}+2*\text{shadow}$ . The `filter_arg_ptr` field of `f` must be the shadow size casted into `void*`. **IMPORTANT:** This can only be used for read-only access, as no coherency is enforced for the shadowed parts. An usage example is available in `examples/filters/shadow.c`

27.10.3.11 `void starpu_vector_filter_list ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

Return in `child_interface` the `id` th element of the vector represented by `father_interface` once partitioned into `nparts` chunks according to the `filter_arg_ptr` field of `f`. The `filter_arg_ptr` field must point to an array of `nparts` `uint32_t` elements, each of which specifies the number of elements in each chunk of the partition.

27.10.3.12 `void starpu_vector_filter_divide_in_2 ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

Return in `child_interface` the `id` th element of the vector represented by `father_interface` once partitioned in 2 chunks of equal size, ignoring `nparts`. Thus, `id` must be 0 or 1.

27.10.3.13 `void starpu_matrix_filter_block ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

This partitions a dense Matrix along the x dimension, thus getting  $(x/\text{nparts}, y)$  matrices. If `nparts` does not divide `x`, the last submatrix contains the remainder.

27.10.3.14 `void starpu_matrix_filter_block_shadow ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

This partitions a dense Matrix along the x dimension, with a shadow border `filter_arg_ptr`, thus getting  $((x-2*\text{shadow})/\text{nparts}+2*\text{shadow}, y)$  matrices. If `nparts` does not divide  $x-2*\text{shadow}$ , the last submatrix contains the remainder. **IMPORTANT:** This can only be used for read-only access, as no coherency is enforced for the shadowed parts. A usage example is available in `examples/filters/shadow2d.c`

27.10.3.15 `void starpu_matrix_filter_vertical_block ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

This partitions a dense Matrix along the y dimension, thus getting  $(x, y/\text{nparts})$  matrices. If `nparts` does not divide `y`, the last submatrix contains the remainder.

27.10.3.16 `void starpu_matrix_filter_vertical_block_shadow ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

This partitions a dense Matrix along the y dimension, with a shadow border `filter_arg_ptr`, thus getting  $(x, (y-2*\text{shadow})/\text{nparts}+2*\text{shadow})$  matrices. If `nparts` does not divide  $y-2*\text{shadow}$ , the last submatrix contains the remainder. **IMPORTANT:** This can only be used for read-only access, as no coherency is enforced for the shadowed parts. A usage example is available in `examples/filters/shadow2d.c`



27.10.3.17 `void starpu_block_filter_block ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

This partitions a block along the X dimension, thus getting  $(x/nparts, y, z)$  3D matrices. If `nparts` does not divide `x`, the last submatrix contains the remainder.

27.10.3.18 `void starpu_block_filter_block_shadow ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

This partitions a block along the X dimension, with a shadow border `filter_arg_ptr`, thus getting  $((x-2*shadow)/nparts + 2*shadow, y, z)$  blocks. If `nparts` does not divide `x`, the last submatrix contains the remainder. **IMPORTANT:** This can only be used for read-only access, as no coherency is enforced for the shadowed parts.

27.10.3.19 `void starpu_block_filter_vertical_block ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

This partitions a block along the Y dimension, thus getting  $(x, y/nparts, z)$  blocks. If `nparts` does not divide `y`, the last submatrix contains the remainder.

27.10.3.20 `void starpu_block_filter_vertical_block_shadow ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

This partitions a block along the Y dimension, with a shadow border `filter_arg_ptr`, thus getting  $(x, (y-2*shadow)/nparts + 2*shadow, z)$  3D matrices. If `nparts` does not divide `y`, the last submatrix contains the remainder. **IMPORTANT:** This can only be used for read-only access, as no coherency is enforced for the shadowed parts.

27.10.3.21 `void starpu_block_filter_depth_block ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

This partitions a block along the Z dimension, thus getting  $(x, y, z/nparts)$  blocks. If `nparts` does not divide `z`, the last submatrix contains the remainder.

27.10.3.22 `void starpu_block_filter_depth_block_shadow ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

This partitions a block along the Z dimension, with a shadow border `filter_arg_ptr`, thus getting  $(x, y, (z-2*shadow)/nparts + 2*shadow)$  blocks. If `nparts` does not divide `z`, the last submatrix contains the remainder. **IMPORTANT:** This can only be used for read-only access, as no coherency is enforced for the shadowed parts.

27.10.3.23 `void starpu_bcsr_filter_canonical_block ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

This partitions a block-sparse matrix into dense matrices.

27.10.3.24 `void starpu_csr_filter_vertical_block ( void * father_interface, void * child_interface, struct starpu_data_filter * f, unsigned id, unsigned nparts )`

This partitions a block-sparse matrix into vertical block-sparse matrices.

## 27.11 Out Of Core

### Data Structures

- struct [starpu\\_disk\\_ops](#)

### Functions

- void \* [starpu\\_disk\\_open](#) (unsigned node, void \*pos, size\_t size)
- void [starpu\\_disk\\_close](#) (unsigned node, void \*obj, size\_t size)

### Variables

- struct [starpu\\_disk\\_ops](#) [starpu\\_disk\\_stdio\\_ops](#)
- struct [starpu\\_disk\\_ops](#) [starpu\\_disk\\_unistd\\_ops](#)
- struct [starpu\\_disk\\_ops](#) [starpu\\_disk\\_unistd\\_o\\_direct\\_ops](#)
- struct [starpu\\_disk\\_ops](#) [starpu\\_disk\\_leveldb\\_ops](#)

#### 27.11.1 Detailed Description

#### 27.11.2 Data Structure Documentation

##### 27.11.2.1 struct starpu\_disk\_ops

This is a set of functions to manipulate datas on disk.

#### Data Fields

- void \*(\* [plug](#) )(void \*parameter, starpu\_ssize\_t size)
- void(\* [unplug](#) )(void \*base)
- int(\* [bandwidth](#) )(unsigned node)
- void \*(\* [alloc](#) )(void \*base, size\_t size)
- void(\* [free](#) )(void \*base, void \*obj, size\_t size)
- void \*(\* [open](#) )(void \*base, void \*pos, size\_t size)
- void(\* [close](#) )(void \*base, void \*obj, size\_t size)
- int(\* [read](#) )(void \*base, void \*obj, void \*buf, off\_t offset, size\_t size)
- int(\* [write](#) )(void \*base, void \*obj, const void \*buf, off\_t offset, size\_t size)
- int(\* [full\\_read](#) )(void \*base, void \*obj, void \*\*ptr, size\_t \*size)
- int(\* [full\\_write](#) )(void \*base, void \*obj, void \*ptr, size\_t size)
- void \*(\* [async\\_write](#) )(void \*base, void \*obj, void \*buf, off\_t offset, size\_t size)
- void \*(\* [async\\_read](#) )(void \*base, void \*obj, void \*buf, off\_t offset, size\_t size)
- void \*(\* [async\\_full\\_read](#) )(void \*base, void \*obj, void \*\*ptr, size\_t \*size)
- void \*(\* [async\\_full\\_write](#) )(void \*base, void \*obj, void \*ptr, size\_t size)
- void \*(\* [copy](#) )(void \*base\_src, void \*obj\_src, off\_t offset\_src, void \*base\_dst, void \*obj\_dst, off\_t offset\_dst, size\_t size)
- void(\* [wait\\_request](#) )(void \*async\_channel)
- int(\* [test\\_request](#) )(void \*async\_channel)
- void(\* [free\\_request](#) )(void \*async\_channel)

## 27.11.2.1.1 Field Documentation

27.11.2.1.1.1 `void>(* starpu_disk_ops::plug)(void *parameters, size_t size)`

Connect a disk memory at location `parameter` with size `size`, and return a base as `void*`, which will be passed by StarPU to all other methods.

27.11.2.1.1.2 `void(* starpu_disk_ops::unplug)(void *base)`

Disconnect a disk memory `base`.

27.11.2.1.1.3 `int(* starpu_disk_ops::bandwidth)(unsigned node)`

Measure the bandwidth and the latency for the disk `node` and save it. Returns 1 if it could measure it.

27.11.2.1.1.4 `void(* starpu_disk_ops::alloc)(void *base, size_t size)`

Create a new location for datas of size `size`. This returns an opaque object pointer.

27.11.2.1.1.5 `void(* starpu_disk_ops::free)(void *base, void *obj, size_t size)`

Free a data `obj` previously allocated with `alloc`.

27.11.2.1.1.6 `void(* starpu_disk_ops::open)(void *base, void *pos, size_t size)`

Open an existing location of datas, at a specific position `pos` dependent on the backend.

27.11.2.1.1.7 `void(* starpu_disk_ops::close)(void *base, void *obj, size_t size)`

Close, without deleting it, a location of datas `obj`.

27.11.2.1.1.8 `int(* starpu_disk_ops::read)(void *base, void *obj, void *buf, off_t offset, size_t size)`

Read `size` bytes of data from `obj` in `base`, at offset `offset`, and put into `buf`. Returns the actual number of read bytes.

27.11.2.1.1.9 `int(* starpu_disk_ops::write)(void *base, void *obj, const void *buf, off_t offset, size_t size)`

Write `size` bytes of data to `obj` in `base`, at offset `offset`, from `buf`. Returns 0 on success.

27.11.2.1.1.10 `int(* starpu_disk_ops::full_read)(void *base, void *obj, void **ptr, size_t size)`

Read all data from `obj` of `base`, from offset 0. Returns it in an allocated buffer `ptr`, of size `size`

27.11.2.1.1.11 `int(* starpu_disk_ops::full_write)(void *base, void *obj, void *ptr, size_t size)`

Write data in `ptr` to `obj` of `base`, from offset 0, and truncate `obj` to `size`, so that a `full_read` will get it.

27.11.2.1.1.12 `void(* starpu_disk_ops::async_write)(void *base, void *obj, const void *buf, off_t offset, size_t size)`

Asynchronously write `size` bytes of data to `obj` in `base`, at offset `offset`, from `buf`. Returns a `void*` pointer that StarPU will pass to `_request` methods for testing for the completion.

27.11.2.1.1.13 `void(* starpu_disk_ops::async_read)(void *base, void *obj, void *buf, off_t offset, size_t size)`

Asynchronously read `size` bytes of data from `obj` in `base`, at offset `offset`, and put into `buf`. Returns a `void*` pointer that StarPU will pass to `_request` methods for testing for the completion.

27.11.2.1.1.14 `void(* starpu_disk_ops::async_full_read)(void *base, void *obj, void **ptr, size_t size)`

Read all data from `obj` of `base`, from offset 0. Returns it in an allocated buffer `ptr`, of size `size`

27.11.2.1.1.15 `void>(* starpu_disk_ops::async_full_write)(void *base, void *obj, void *ptr, size_t size)`

Write data in `ptr` to `obj` of `base`, from offset 0, and truncate `obj` to `size`, so that a `full_read` will get it.

27.11.2.1.1.16 `void>(* starpu_disk_ops::copy)(void *base_src, void *obj_src, off_t offset_src, void *base_dst, void *obj_dst, off_t offset_dst, size_t size)`

Copy from offset `offset_src` of disk object `obj_src` in `base_src` to offset `offset_dst` of disk object `obj_dst` in `base_dst`. Returns a `void*` pointer that StarPU will pass to `*_request` methods for testing for the completion.

27.11.2.1.1.17 `void(* starpu_disk_ops::wait_request)(void *async_channel)`

Wait for completion of request `async_channel` returned by a previous asynchronous read, write or copy.

27.11.2.1.1.18 `void(* starpu_disk_ops::test_request)(void *async_channel)`

Test for completion of request `async_channel` returned by a previous asynchronous read, write or copy. Returns 1 on completion, 0 otherwise.

27.11.2.1.1.19 `void(* starpu_disk_ops::free_request)(void *async_channel)`

Free the request allocated by a previous asynchronous read, write or copy.

### 27.11.3 Function Documentation

27.11.3.1 `void * starpu_disk_open ( unsigned node, void * pos, size_t size )`

Open an existing file memory in a disk node. `size`: this is a size of your file. `pos` is specific position dependent on the backend, given to the `open` method of the disk operations. This returns an opaque object pointer.

27.11.3.2 `void starpu_disk_close ( unsigned node, void * obj, size_t size )`

Close an existing data opened with `starpu_disk_open`.

### 27.11.4 Variable Documentation

27.11.4.1 `starpu_disk_stdio_ops`

This set uses the `stdio` library (`fwrite`, `fread`...) to read/write on disk.

**Warning: It creates one file per allocation !**

It doesn't support asynchronous transfers.

27.11.4.2 `starpu_disk_unistd_ops`

This set uses the `unistd` library (`write`, `read`...) to read/write on disk.

**Warning: It creates one file per allocation !**

27.11.4.3 `starpu_disk_unistd_o_direct_ops`

This set uses the `unistd` library (`write`, `read`...) to read/write on disk with the `O_DIRECT` flag.

**Warning: It creates one file per allocation !**

Only available on Linux systems.

## 27.11.4.4 starpu\_disk\_leveldb\_ops

This set uses the leveldb created by Google

Show here: <https://code.google.com/p/leveldb/>

It doesn't support asynchronous transfers.

## 27.12 Multiformat Data Interface

### Data Structures

- struct [starpu\\_multiformat\\_data\\_interface\\_ops](#)
- struct [starpu\\_multiformat\\_interface](#)

### Macros

- #define [STARPU\\_MULTIFORMAT\\_GET\\_CPU\\_PTR](#)(interface)
- #define [STARPU\\_MULTIFORMAT\\_GET\\_CUDA\\_PTR](#)(interface)
- #define [STARPU\\_MULTIFORMAT\\_GET\\_OPENCL\\_PTR](#)(interface)
- #define [STARPU\\_MULTIFORMAT\\_GET\\_MIC\\_PTR](#)(interface)
- #define [STARPU\\_MULTIFORMAT\\_GET\\_NX](#)(interface)

### Functions

- void [starpu\\_multiformat\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, void \*ptr, uint32\_t nobjects, struct [starpu\\_multiformat\\_data\\_interface\\_ops](#) \*format\_ops)

#### 27.12.1 Detailed Description

#### 27.12.2 Data Structure Documentation

##### 27.12.2.1 struct starpu\_multiformat\_data\_interface\_ops

The different fields are:

##### Data Fields

|                                         |                       |                                                        |
|-----------------------------------------|-----------------------|--------------------------------------------------------|
| size_t                                  | cpu_elemsize          | the size of each element on CPUs                       |
| size_t                                  | opencl_elemsize       | the size of each element on OpenCL devices             |
| struct <a href="#">starpu_codelet</a> * | cpu_to_opencl↔<br>_cl | pointer to a codelet which converts from CPU to OpenCL |
| struct <a href="#">starpu_codelet</a> * | opencl_to_cpu↔<br>_cl | pointer to a codelet which converts from OpenCL to CPU |
| size_t                                  | cuda_elemsize         | the size of each element on CUDA devices               |
| struct <a href="#">starpu_codelet</a> * | cpu_to_cuda_cl        | pointer to a codelet which converts from CPU to CUDA   |
| struct <a href="#">starpu_codelet</a> * | cuda_to_cpu_cl        | pointer to a codelet which converts from CUDA to CPU   |
| size_t                                  | mic_elemsize          | the size of each element on MIC devices                |
| struct <a href="#">starpu_codelet</a> * | cpu_to_mic_cl         | pointer to a codelet which converts from CPU to MIC    |

|                                            |               |                                                     |
|--------------------------------------------|---------------|-----------------------------------------------------|
| struct<br><a href="#">starpu_codelet</a> * | mic_to_cpu_cl | pointer to a codelet which converts from MIC to CPU |
|--------------------------------------------|---------------|-----------------------------------------------------|

### 27.12.2.2 struct starpu\_multiformat\_interface

todo

#### Data Fields

|                                                                                        |            |  |
|----------------------------------------------------------------------------------------|------------|--|
| enum<br><a href="#">starpu_data_↔<br/>interface_id</a>                                 | id         |  |
| void *                                                                                 | cpu_ptr    |  |
| void *                                                                                 | cuda_ptr   |  |
| void *                                                                                 | opencl_ptr |  |
| void *                                                                                 | mic_ptr    |  |
| uint32_t                                                                               | nx         |  |
| struct<br><a href="#">starpu_↔<br/>multiformat_↔<br/>data_interface↔<br/>_ops</a><br>* | ops        |  |

## 27.12.3 Macro Definition Documentation

### 27.12.3.1 #define STARPU\_MULTIFORMAT\_GET\_CPU\_PTR( *interface* )

returns the local pointer to the data with CPU format.

### 27.12.3.2 #define STARPU\_MULTIFORMAT\_GET\_CUDA\_PTR( *interface* )

returns the local pointer to the data with CUDA format.

### 27.12.3.3 #define STARPU\_MULTIFORMAT\_GET\_OPENCL\_PTR( *interface* )

returns the local pointer to the data with OpenCL format.

### 27.12.3.4 #define STARPU\_MULTIFORMAT\_GET\_MIC\_PTR( *interface* )

returns the local pointer to the data with MIC format.

### 27.12.3.5 #define STARPU\_MULTIFORMAT\_GET\_NX( *interface* )

returns the number of elements in the data.

## 27.12.4 Function Documentation

27.12.4.1 void starpu\_multiformat\_data\_register ( starpu\_data\_handle\_t \* handle, unsigned home\_node, void \* ptr, uint32\_t nobjects, struct starpu\_multiformat\_data\_interface\_ops \* format\_ops )

Register a piece of data that can be represented in different ways, depending upon the processing unit that manipulates it. It allows the programmer, for instance, to use an array of structures when working on a CPU, and a structure of arrays when working on a GPU. `nobjects` is the number of elements in the data. `format_ops` describes the format.

## 27.13 Codelet And Tasks

This section describes the interface to manipulate codelets and tasks.

### Data Structures

- struct [starpu\\_codelet](#)
- struct [starpu\\_data\\_descr](#)
- struct [starpu\\_task](#)

### Macros

- #define [STARPU\\_CPU](#)
- #define [STARPU\\_CUDA](#)
- #define [STARPU\\_OPENCL](#)
- #define [STARPU\\_MIC](#)
- #define [STARPU\\_SCC](#)
- #define [STARPU\\_MAIN\\_RAM](#)
- #define [STARPU\\_MULTIPLE\\_CPU\\_IMPLEMENTATIONS](#)
- #define [STARPU\\_MULTIPLE\\_CUDA\\_IMPLEMENTATIONS](#)
- #define [STARPU\\_MULTIPLE\\_OPENCL\\_IMPLEMENTATIONS](#)
- #define [STARPU\\_NMAXBUFS](#)
- #define [STARPU\\_VARIABLE\\_NBUFFERS](#)
- #define [STARPU\\_TASK\\_INITIALIZER](#)
- #define [STARPU\\_TASK\\_GET\\_NBUFFERS](#)(task)
- #define [STARPU\\_TASK\\_GET\\_HANDLE](#)(task, i)
- #define [STARPU\\_TASK\\_SET\\_HANDLE](#)(task, handle, i)
- #define [STARPU\\_CODELET\\_GET\\_MODE](#)(codelet, i)
- #define [STARPU\\_CODELET\\_SET\\_MODE](#)(codelet, mode, i)
- #define [STARPU\\_TASK\\_GET\\_MODE](#)(task, i)
- #define [STARPU\\_TASK\\_SET\\_MODE](#)(task, mode, i)
- #define [STARPU\\_TASK\\_INVALID](#)

### Typedefs

- typedef void(\* [starpu\\_cpu\\_func\\_t](#) )(void \*\*, void \*)
- typedef void(\* [starpu\\_cuda\\_func\\_t](#) )(void \*\*, void \*)
- typedef void(\* [starpu\\_opengl\\_func\\_t](#) )(void \*\*, void \*)
- typedef [starpu\\_mic\\_kernel\\_t](#)(\* [starpu\\_mic\\_func\\_t](#) )(void)
- typedef [starpu\\_scc\\_kernel\\_t](#)(\* [starpu\\_scc\\_func\\_t](#) )(void)
- typedef void(\* [starpu\\_mic\\_kernel\\_t](#) )(void \*\*, void \*)
- typedef void(\* [starpu\\_scc\\_kernel\\_t](#) )(void \*\*, void \*)

## Enumerations

- enum `starpu_codelet_type` { `STARPU_SEQ`, `STARPU_SPMD`, `STARPU_FORKJOIN` }
- enum `starpu_task_status` {  
`STARPU_TASK_INVALID`, `STARPU_TASK_INVALID`, `STARPU_TASK_BLOCKED`, `STARPU_TASK_READY`,  
`STARPU_TASK_RUNNING`, `STARPU_TASK_FINISHED`, `STARPU_TASK_BLOCKED_ON_TAG`, `STARPU_TASK_BLOCKED_ON_TASK`,  
`STARPU_TASK_BLOCKED_ON_DATA`, `STARPU_TASK_STOPPED` }

## Functions

- void `starpu_codelet_init` (struct `starpu_codelet` \*cl)
- void `starpu_task_init` (struct `starpu_task` \*task)
- struct `starpu_task` \* `starpu_task_create` (void)
- struct `starpu_task` \* `starpu_task_dup` (struct `starpu_task` \*task)
- void `starpu_task_clean` (struct `starpu_task` \*task)
- void `starpu_task_destroy` (struct `starpu_task` \*task)
- int `starpu_task_wait` (struct `starpu_task` \*task) `STARPU_WARN_UNUSED_RESULT`
- int `starpu_task_submit` (struct `starpu_task` \*task) `STARPU_WARN_UNUSED_RESULT`
- int `starpu_task_submit_to_ctx` (struct `starpu_task` \*task, unsigned sched\_ctx\_id)
- int `starpu_task_wait_for_all` (void)
- int `starpu_task_wait_for_all_in_ctx` (unsigned sched\_ctx\_id)
- int `starpu_task_nready` (void)
- int `starpu_task_nsubmitted` (void)
- struct `starpu_task` \* `starpu_task_get_current` (void)
- void `starpu_codelet_display_stats` (struct `starpu_codelet` \*cl)
- int `starpu_task_wait_for_no_ready` (void)
- void `starpu_task_set_implementation` (struct `starpu_task` \*task, unsigned impl)
- unsigned `starpu_task_get_implementation` (struct `starpu_task` \*task)
- void `starpu_create_sync_task` (starpu\_tag\_t sync\_tag, unsigned ndeps, starpu\_tag\_t \*deps, void(\*callback)(void \*), void \*callback\_arg)

### 27.13.1 Detailed Description

This section describes the interface to manipulate codelets and tasks.

### 27.13.2 Data Structure Documentation

#### 27.13.2.1 struct `starpu_codelet`

The codelet structure describes a kernel that is possibly implemented on various targets. For compatibility, make sure to initialize the whole structure to zero, either by using explicit `memset`, or the function `starpu_codelet_init()`, or by letting the compiler implicitly do it in e.g. static storage case.

#### Data Fields

- uint32\_t `where`
- int(\* `can_execute` )(unsigned workerid, struct `starpu_task` \*task, unsigned nimpl)
- enum `starpu_codelet_type` `type`
- int `max_parallelism`
- `starpu_cpu_func_t` `cpu_func`
- `starpu_cuda_func_t` `cuda_func`



- `starpu_opencil_func_t opencil_func`
- `starpu_cpu_func_t cpu_funcs` [STARPU\_MAXIMPLEMENTATIONS]
- `starpu_cuda_func_t cuda_funcs` [STARPU\_MAXIMPLEMENTATIONS]
- `char cuda_flags` [STARPU\_MAXIMPLEMENTATIONS]
- `starpu_opencil_func_t opencil_funcs` [STARPU\_MAXIMPLEMENTATIONS]
- `char opencil_flags` [STARPU\_MAXIMPLEMENTATIONS]
- `starpu_mic_func_t mic_funcs` [STARPU\_MAXIMPLEMENTATIONS]
- `starpu_scc_func_t scc_funcs` [STARPU\_MAXIMPLEMENTATIONS]
- `char * cpu_funcs_name` [STARPU\_MAXIMPLEMENTATIONS]
- `int nbuffers`
- `enum starpu_data_access_mode modes` [STARPU\_NMAXBUFS]
- `enum starpu_data_access_mode * dyn_modes`
- `unsigned specific_nodes`
- `int nodes` [STARPU\_NMAXBUFS]
- `int * dyn_nodes`
- `struct starpu_perfmodel * model`
- `struct starpu_perfmodel * power_model`
- `unsigned long per_worker_stats` [STARPU\_NMAXWORKERS]
- `const char * name`

#### 27.13.2.1.1 Field Documentation

##### 27.13.2.1.1.1 `uint32_t starpu_codelet::where`

Optional field to indicate which types of processing units are able to execute the codelet. The different values `STARPU_CPU`, `STARPU_CUDA`, `STARPU_OPENCL` can be combined to specify on which types of processing units the codelet can be executed. `STARPU_CPU|STARPU_CUDA` for instance indicates that the codelet is implemented for both CPU cores and CUDA devices while `STARPU_OPENCL` indicates that it is only available on OpenCL devices. If the field is unset, its value will be automatically set based on the availability of the `XXX_funcs` fields defined below.

##### 27.13.2.1.1.2 `int(* starpu_codelet::can_execute)(unsigned workerid, struct starpu_task *task, unsigned nimpi)`

Define a function which should return 1 if the worker designated by `workerid` can execute the `nimplth` implementation of the given task, 0 otherwise.

##### 27.13.2.1.1.3 `enum starpu_codelet_type starpu_codelet::type`

Optional field to specify the type of the codelet. The default is `STARPU_SEQ`, i.e. usual sequential implementation. Other values (`STARPU_SPMD` or `STARPU_FORKJOIN`) declare that a parallel implementation is also available. See [Parallel Tasks](#) for details.

##### 27.13.2.1.1.4 `int starpu_codelet::max_parallelism`

Optional field. If a parallel implementation is available, this denotes the maximum combined worker size that StarPU will use to execute parallel tasks for this codelet.

##### 27.13.2.1.1.5 `starpu_cpu_func_t starpu_codelet::cpu_func`

**Deprecated** Optional field which has been made deprecated. One should use instead the field `starpu_codelet::cpu_funcs`.

##### 27.13.2.1.1.6 `starpu_cuda_func_t starpu_codelet::cuda_func`

**Deprecated** Optional field which has been made deprecated. One should use instead the `starpu_codelet::cuda_funcs` field.

#### 27.13.2.1.1.7 `starpu_opencil_func_t` `starpu_codelet::opencil_func`

**Deprecated** Optional field which has been made deprecated. One should use instead the `starpu_codelet::opencil_funcs` field.

#### 27.13.2.1.1.8 `starpu_cpu_func_t` `starpu_codelet::cpu_funcs[STARPU_MAXIMPLEMENTATIONS]`

Optional array of function pointers to the CPU implementations of the codelet. The functions prototype must be:

```
void cpu_func(void *buffers[], void *cl_arg)
```

The first argument being the array of data managed by the data management library, and the second argument is a pointer to the argument passed from the field `starpu_task::cl_arg`. If the field `starpu_codelet::where` is set, then the field `starpu_codelet::cpu_funcs` is ignored if `STARPU_CPU` does not appear in the field `starpu_codelet::where`, it must be non-null otherwise.

#### 27.13.2.1.1.9 `starpu_cuda_func_t` `starpu_codelet::cuda_funcs[STARPU_MAXIMPLEMENTATIONS]`

Optional array of function pointers to the CUDA implementations of the codelet. The functions must be host-functions written in the CUDA runtime API. Their prototype must be:

```
void cuda_func(void *buffers[], void *cl_arg)
```

If the field `starpu_codelet::where` is set, then the field `starpu_codelet::cuda_funcs` is ignored if `STARPU_CUDA` does not appear in the field `starpu_codelet::where`, it must be non-null otherwise.

#### 27.13.2.1.1.10 `char` `starpu_codelet::cuda_flags[STARPU_MAXIMPLEMENTATIONS]`

Optional array of flags for CUDA execution. They specify some semantic details about CUDA kernel execution, such as asynchronous execution.

#### 27.13.2.1.1.11 `starpu_opencil_func_t` `starpu_codelet::opencil_funcs[STARPU_MAXIMPLEMENTATIONS]`

Optional array of function pointers to the OpenCL implementations of the codelet. The functions prototype must be:

```
void opencil_func(void *buffers[], void *cl_arg)
```

If the field `starpu_codelet::where` field is set, then the field `starpu_codelet::opencil_funcs` is ignored if `STARPU_OPENCL` does not appear in the field `starpu_codelet::where`, it must be non-null otherwise.

#### 27.13.2.1.1.12 `char` `starpu_codelet::opencil_flags[STARPU_MAXIMPLEMENTATIONS]`

Optional array of flags for OpenCL execution. They specify some semantic details about OpenCL kernel execution, such as asynchronous execution.

#### 27.13.2.1.1.13 `starpu_mic_func_t` `starpu_codelet::mic_funcs[STARPU_MAXIMPLEMENTATIONS]`

Optional array of function pointers to a function which returns the MIC implementation of the codelet. The functions prototype must be:

```
starpu_mic_kernel_t mic_func(struct starpu_codelet *cl, unsigned nimpl)
```

If the field `starpu_codelet::where` is set, then the field `starpu_codelet::mic_funcs` is ignored if `STARPU_MIC` does not appear in the field `starpu_codelet::where`. It can be null if `starpu_codelet::cpu_funcs_name` is non-NULL, in which case StarPU will simply make a symbol lookup to get the implementation.

#### 27.13.2.1.1.14 `starpu_scc_func_t` `starpu_codelet::scc_funcs[STARPU_MAXIMPLEMENTATIONS]`

Optional array of function pointers to a function which returns the SCC implementation of the codelet. The functions prototype must be:

```
starpu_scc_kernel_t scc_func(struct starpu_codelet *cl, unsigned nimpl)
```

If the field `starpu_codelet::where` is set, then the field `starpu_codelet::scc_funcs` is ignored if `STARPU_SCC` does not appear in the field `starpu_codelet::where`. It can be null if `starpu_codelet::cpu_funcs_name` is non-NULL, in which case StarPU will simply make a symbol lookup to get the implementation.

**27.13.2.1.1.15** `char * starpu_codelet::cpu_funcs_name[STARPU_MAXIMPLEMENTATIONS]`

Optional array of strings which provide the name of the CPU functions referenced in the array `starpu_codelet::cpu_funcs`. This can be used when running on MIC devices or the SCC platform, for StarPU to simply look up the MIC function implementation through its name.

**27.13.2.1.1.16** `int starpu_codelet::nbuffers`

Specify the number of arguments taken by the codelet. These arguments are managed by the DSM and are accessed from the `void *buffers[]` array. The constant argument passed with the field `starpu_task::cl_arg` is not counted in this number. This value should not be above `STARPU_NMAXBUFS`. It may be set to `STARPU_VARIABLE_NBUFFERS` to specify that the number of buffers and their access modes will be set in `starpu_task::nbuffers` and `starpu_task::modes` or `starpu_task::dyn_modes`, which thus permits to define codelets with a varying number of data.

**27.13.2.1.1.17** `enum starpu_data_access_mode starpu_codelet::modes[STARPU_NMAXBUFS]`

Is an array of `starpu_data_access_mode`. It describes the required access modes to the data needed by the codelet (e.g. `STARPU_RW`). The number of entries in this array must be specified in the field `starpu_codelet::nbuffers`, and should not exceed `STARPU_NMAXBUFS`. If insufficient, this value can be set with the configure option `--enable-maxbuffers`.

**27.13.2.1.1.18** `enum starpu_data_access_mode * starpu_codelet::dyn_modes`

Is an array of `starpu_data_access_mode`. It describes the required access modes to the data needed by the codelet (e.g. `STARPU_RW`). The number of entries in this array must be specified in the field `starpu_codelet::nbuffers`. This field should be used for codelets having a number of datas greater than `STARPU_NMAXBUFS` (see [Setting The Data Handles For A Task](#)). When defining a codelet, one should either define this field or the field `starpu_codelet::modes` defined above.

**27.13.2.1.1.19** `unsigned starpu_codelet::specific_nodes`

Default value is 0. If this flag is set, StarPU will not systematically send all data to the memory node where the task will be executing, it will read the `starpu_codelet::nodes` or `starpu_codelet::dyn_nodes` array to determine, for each data, whether to send it on the memory node where the task will be executing (-1), or on a specific node (!= -1).

**27.13.2.1.1.20** `int starpu_codelet::nodes[STARPU_NMAXBUFS]`

Optional field. When `starpu_codelet::specific_nodes` is 1, this specifies the memory nodes where each data should be sent to for task execution. The number of entries in this array is `starpu_codelet::nbuffers`, and should not exceed `STARPU_NMAXBUFS`.

**27.13.2.1.1.21** `int * starpu_codelet::dyn_nodes`

Optional field. When `starpu_codelet::specific_nodes` is 1, this specifies the memory nodes where each data should be sent to for task execution. The number of entries in this array is `starpu_codelet::nbuffers`. This field should be used for codelets having a number of datas greater than `STARPU_NMAXBUFS` (see [Setting The Data Handles For A Task](#)). When defining a codelet, one should either define this field or the field `starpu_codelet::nodes` defined above.

**27.13.2.1.1.22** `struct starpu_perfmodel * starpu_codelet::model`

Optional pointer to the task duration performance model associated to this codelet. This optional field is ignored when set to `NULL` or when its field `starpu_perfmodel::symbol` is not set.

### 27.13.2.1.1.23 struct starpu\_perfmodel \* starpu\_codelet::power\_model

Optional pointer to the task power consumption performance model associated to this codelet. This optional field is ignored when set to `NULL` or when its field `starpu_perfmodel::field` is not set. In the case of parallel codelets, this has to account for all processing units involved in the parallel execution.

### 27.13.2.1.1.24 unsigned long starpu\_codelet::per\_worker\_stats[STARPU\_NMAXWORKERS]

Optional array for statistics collected at runtime: this is filled by StarPU and should not be accessed directly, but for example by calling the function `starpu_codelet_display_stats()` (See `starpu_codelet_display_stats()` for details).

### 27.13.2.1.1.25 const char \* starpu\_codelet::name

Optional name of the codelet. This can be useful for debugging purposes.

## 27.13.2.2 struct starpu\_data\_descr

This type is used to describe a data handle along with an access mode.

### Data Fields

|                                                        |        |                           |
|--------------------------------------------------------|--------|---------------------------|
| <a href="#">starpu_data_↔<br/>handle_t</a>             | handle | describes a data          |
| <a href="#">enum<br/>starpu_data_↔<br/>access_mode</a> | mode   | describes its access mode |

## 27.13.2.3 struct starpu\_task

The structure describes a task that can be offloaded on the various processing units managed by StarPU. It instantiates a codelet. It can either be allocated dynamically with the function `starpu_task_create()`, or declared statically. In the latter case, the programmer has to zero the structure `starpu_task` and to fill the different fields properly. The indicated default values correspond to the configuration of a task allocated with `starpu_task_create()`.

### Data Fields

- const char \* [name](#)
- struct [starpu\\_codelet](#) \* [cl](#)
- int [nbuffers](#)
- [starpu\\_data\\_handle\\_t](#) [handles](#) [STARPU\_NMAXBUFS]
- void \* [interfaces](#) [STARPU\_NMAXBUFS]
- enum [starpu\\_data\\_access\\_mode](#) [modes](#) [STARPU\_NMAXBUFS]
- [starpu\\_data\\_handle\\_t](#) \* [dyn\\_handles](#)
- void \*\* [dyn\\_interfaces](#)
- enum [starpu\\_data\\_access\\_mode](#) \* [dyn\\_modes](#)
- void \* [cl\\_arg](#)
- size\_t [cl\\_arg\\_size](#)
- void(\* [callback\\_func](#) )(void \*)
- void \* [callback\\_arg](#)
- void(\* [prologue\\_callback\\_func](#) )(void \*)
- void \* [prologue\\_callback\\_arg](#)
- void(\* [prologue\\_callback\\_pop\\_func](#) )(void \*)
- void \* [prologue\\_callback\\_pop\\_arg](#)
- [starpu\\_tag\\_t](#) [tag\\_id](#)
- unsigned [cl\\_arg\\_free](#):1
- unsigned [callback\\_arg\\_free](#):1

- unsigned `prologue_callback_arg_free`:1
- unsigned `prologue_callback_pop_arg_free`:1
- unsigned `use_tag`:1
- unsigned `sequential_consistency`:1
- unsigned `synchronous`:1
- unsigned `execute_on_a_specific_worker`:1
- unsigned `detach`:1
- unsigned `destroy`:1
- unsigned `regenerate`:1
- unsigned `workerid`
- unsigned `workerorder`
- unsigned `scheduled`:1
- unsigned int `mf_skip`:1
- int `priority`
- enum `starpu_task_status` status
- int `magic`
- unsigned `sched_ctx`
- int `hypervisor_tag`
- unsigned `possibly_parallel`
- `starpu_task_bundle_t` bundle
- struct `starpu_profiling_task_info` \* `profiling_info`
- double `flops`
- double `predicted`
- double `predicted_transfer`
- struct `starpu_task` \* `prev`
- struct `starpu_task` \* `next`
- void \* `starpu_private`
- unsigned `prefetched`
- struct `starpu_omp_task` \* `omp_task`

#### 27.13.2.3.1 Field Documentation

##### 27.13.2.3.1.1 `const char * starpu_task::name`

Optional name of the task. This can be useful for debugging purposes.

##### 27.13.2.3.1.2 `struct starpu_codelet * starpu_task::cl`

Is a pointer to the corresponding structure `starpu_codelet`. This describes where the kernel should be executed, and supplies the appropriate implementations. When set to NULL, no code is executed during the tasks, such empty tasks can be useful for synchronization purposes. This field has been made deprecated. One should use instead the field `starpu_task::handles` to specify the data handles accessed by the task. The access modes are now defined in the field `starpu_codelet::modes`.

##### 27.13.2.3.1.3 `int starpu_task::nbuffers`

Specifies the number of buffers. This is only used when `starpu_codelet::nbuffers` is `STARPU_VARIABLE_NBUFFERS`.

##### 27.13.2.3.1.4 `starpu_data_handle_t starpu_task::handles[STARPU_NMAXBUFS]`

Is an array of `starpu_data_handle_t`. It specifies the handles to the different pieces of data accessed by the task. The number of entries in this array must be specified in the field `starpu_codelet::nbuffers`, and should not exceed `STARPU_NMAXBUFS`. If insufficient, this value can be set with the configure option `--enable-maxbuffers`.

##### 27.13.2.3.1.5 `void * starpu_task::interfaces[STARPU_NMAXBUFS]`

The actual data pointers to the memory node where execution will happen, managed by the DSM.

#### 27.13.2.3.1.6 `enum starpu_data_access_mode starpu_task::modes[STARPU_NMAXBUFS]`

Is used only when `starpu_codelet::nbuffers` is `STARPU_VARIABLE_NBUFFERS`. It is an array of `starpu_data_access_mode`. It describes the required access modes to the data needed by the codelet (e.g. `STARPU_RW`). The number of entries in this array must be specified in the field `starpu_task::nbuffers`, and should not exceed `STARPU_NMAXBUFS`. If insufficient, this value can be set with the configure option `--enable-maxbuffers`.

#### 27.13.2.3.1.7 `starpu_data_handle_t * starpu_task::dyn_handles`

Is an array of `starpu_data_handle_t`. It specifies the handles to the different pieces of data accessed by the task. The number of entries in this array must be specified in the field `starpu_codelet::nbuffers`. This field should be used for tasks having a number of datas greater than `STARPU_NMAXBUFS` (see [Setting The Data Handles For A Task](#)). When defining a task, one should either define this field or the field `starpu_task::handles` defined above.

#### 27.13.2.3.1.8 `void ** starpu_task::dyn_interfaces`

The actual data pointers to the memory node where execution will happen, managed by the DSM. Is used when the field `starpu_task::dyn_handles` is defined.

#### 27.13.2.3.1.9 `enum starpu_data_access_mode * starpu_task::dyn_modes`

Is used only when `starpu_codelet::nbuffers` is `STARPU_VARIABLE_NBUFFERS`. It is an array of `starpu_data_access_mode`. It describes the required access modes to the data needed by the codelet (e.g. `STARPU_RW`). The number of entries in this array must be specified in the field `starpu_codelet::nbuffers`. This field should be used for codelets having a number of datas greater than `STARPU_NMAXBUFS` (see [Setting The Data Handles For A Task](#)). When defining a codelet, one should either define this field or the field `starpu_task::modes` defined above.

#### 27.13.2.3.1.10 `void * starpu_task::cl_arg`

Optional pointer which is passed to the codelet through the second argument of the codelet implementation (e.g. `starpu_codelet::cpu_func` or `starpu_codelet::cuda_func`). The default value is `NULL`.

#### 27.13.2.3.1.11 `size_t starpu_task::cl_arg_size`

Optional field. For some specific drivers, the pointer `starpu_task::cl_arg` cannot not be directly given to the driver function. A buffer of size `starpu_task::cl_arg_size` needs to be allocated on the driver. This buffer is then filled with the `starpu_task::cl_arg_size` bytes starting at address `starpu_task::cl_arg`. In this case, the argument given to the codelet is therefore not the `starpu_task::cl_arg` pointer, but the address of the buffer in local store (LS) instead. This field is ignored for CPU, CUDA and OpenCL codelets, where the `starpu_task::cl_arg` pointer is given as such.

#### 27.13.2.3.1.12 `void(* starpu_task::callback_func)(void *)`

Optional field, the default value is `NULL`. This is a function pointer of prototype `void (*f)(void *)` which specifies a possible callback. If this pointer is non-null, the callback function is executed on the host after the execution of the task. Tasks which depend on it might already be executing. The callback is passed the value contained in the `starpu_task::callback_arg` field. No callback is executed if the field is set to `NULL`.

#### 27.13.2.3.1.13 `void * starpu_task::callback_arg`

Optional field, the default value is `NULL`. This is the pointer passed to the callback function. This field is ignored if the field `starpu_task::callback_func` is set to `NULL`.

#### 27.13.2.3.1.14 `void(* starpu_task::prologue_callback_func)(void *)`

Optional field, the default value is `NULL`. This is a function pointer of prototype `void (*f)(void *)` which specifies a possible callback. If this pointer is non-null, the callback function is executed on the host when the task becomes ready for execution, before getting scheduled. The callback is passed the value contained in the `starpu_task::prologue_callback_arg` field. No callback is executed if the field is set to `NULL`.

**27.13.2.3.1.15 void \* starpu\_task::prologue\_callback\_arg**

Optional field, the default value is `NULL`. This is the pointer passed to the prologue callback function. This field is ignored if the field `starpu_task::prologue_callback_func` is set to `NULL`.

**27.13.2.3.1.16 starpu\_tag\_t starpu\_task::tag\_id**

This optional field contains the tag associated to the task if the field `starpu_task::use_tag` is set, it is ignored otherwise.

**27.13.2.3.1.17 unsigned starpu\_task::cl\_arg\_free**

Optional field. In case `starpu_task::cl_arg` was allocated by the application through `malloc()`, setting `starpu_task::cl_arg_free` to 1 makes StarPU automatically call `free(cl_arg)` when destroying the task. This saves the user from defining a callback just for that. This is mostly useful when targeting MIC or SCC, where the codelet does not execute in the same memory space as the main thread.

**27.13.2.3.1.18 unsigned starpu\_task::callback\_arg\_free**

Optional field. In case `starpu_task::callback_arg` was allocated by the application through `malloc()`, setting `starpu_task::callback_arg_free` to 1 makes StarPU automatically call `free(callback_arg)` when destroying the task.

**27.13.2.3.1.19 unsigned starpu\_task::prologue\_callback\_arg\_free**

Optional field. In case `starpu_task::prologue_callback_arg` was allocated by the application through `malloc()`, setting `starpu_task::prologue_callback_arg_free` to 1 makes StarPU automatically call `free(prologue_callback_arg)` when destroying the task.

**27.13.2.3.1.20 unsigned starpu\_task::use\_tag**

Optional field, the default value is 0. If set, this flag indicates that the task should be associated with the tag contained in the `starpu_task::tag_id` field. Tag allow the application to synchronize with the task and to express task dependencies easily.

**27.13.2.3.1.21 unsigned starpu\_task::sequential\_consistency**

If this flag is set (which is the default), sequential consistency is enforced for the data parameters of this task for which sequential consistency is enabled. Clearing this flag permits to disable sequential consistency for this task, even if data have it enabled.

**27.13.2.3.1.22 unsigned starpu\_task::synchronous**

If this flag is set, the function `starpu_task_submit()` is blocking and returns only when the task has been executed (or if no worker is able to process the task). Otherwise, `starpu_task_submit()` returns immediately.

**27.13.2.3.1.23 unsigned starpu\_task::execute\_on\_a\_specific\_worker**

Default value is 0. If this flag is set, StarPU will bypass the scheduler and directly affect this task to the worker specified by the field `starpu_task::workerid`.

**27.13.2.3.1.24 unsigned starpu\_task::detach**

Optional field, default value is 1. If this flag is set, it is not possible to synchronize with the task by the means of `starpu_task_wait()` later on. Internal data structures are only guaranteed to be freed once `starpu_task_wait()` is called if the flag is not set.

**27.13.2.3.1.25 unsigned starpu\_task::destroy**

Optional value. Default value is 0 for `starpu_task_init()`, and 1 for `starpu_task_create()`. If this flag is set, the task structure will automatically be freed, either after the execution of the callback if the task is detached, or dur-



ing [starpu\\_task\\_wait\(\)](#) otherwise. If this flag is not set, dynamically allocated data structures will not be freed until [starpu\\_task\\_destroy\(\)](#) is called explicitly. Setting this flag for a statically allocated task structure will result in undefined behaviour. The flag is set to 1 when the task is created by calling [starpu\\_task\\_create\(\)](#). Note that [starpu\\_task\\_wait\\_for\\_all\(\)](#) will not free any task.

#### 27.13.2.3.1.26 unsigned starpu\_task::regenerate

Optional field. If this flag is set, the task will be re-submitted to StarPU once it has been executed. This flag must not be set if the flag [starpu\\_task::destroy](#) is set. This flag must be set before making another task depend on this one.

#### 27.13.2.3.1.27 unsigned starpu\_task::workerid

Optional field. If the field [starpu\\_task::execute\\_on\\_a\\_specific\\_worker](#) is set, this field indicates the identifier of the worker that should process this task (as returned by [starpu\\_worker\\_get\\_id\(\)](#)). This field is ignored if the field [starpu\\_task::execute\\_on\\_a\\_specific\\_worker](#) is set to 0.

#### 27.13.2.3.1.28 unsigned starpu\_task::workerorder

Optional field. If the field [starpu\\_task::execute\\_on\\_a\\_specific\\_worker](#) is set, this field indicates the per-worker consecutive order in which tasks should be executed on the worker. Tasks will be executed in consecutive [starpu\\_task::workerorder](#) values, thus ignoring the availability order or task priority. See [Static Scheduling](#) for more details. This field is ignored if the field [starpu\\_task::execute\\_on\\_a\\_specific\\_worker](#) is set to 0.

#### 27.13.2.3.1.29 unsigned starpu\_task::scheduled

Whether the scheduler has pushed the task on some queue

#### 27.13.2.3.1.30 unsigned int starpu\_task::mf\_skip

This is only used for tasks that use multifunction handle. This should only be used by StarPU.

#### 27.13.2.3.1.31 int starpu\_task::priority

Optional field, the default value is [STARPU\\_DEFAULT\\_PRIO](#). This field indicates a level of priority for the task. This is an integer value that must be set between the return values of the function [starpu\\_sched\\_get\\_min\\_priority\(\)](#) for the least important tasks, and that of the function [starpu\\_sched\\_get\\_max\\_priority\(\)](#) for the most important tasks (included). The [STARPU\\_MIN\\_PRIO](#) and [STARPU\\_MAX\\_PRIO](#) macros are provided for convenience and respectively returns the value of [starpu\\_sched\\_get\\_min\\_priority\(\)](#) and [starpu\\_sched\\_get\\_max\\_priority\(\)](#). Default priority is [STARPU\\_DEFAULT\\_PRIO](#), which is always defined as 0 in order to allow static task initialization. Scheduling strategies that take priorities into account can use this parameter to take better scheduling decisions, but the scheduling policy may also ignore it.

#### 27.13.2.3.1.32 enum starpu\_task\_status starpu\_task::status

Optional field. Current state of the task.

#### 27.13.2.3.1.33 int starpu\_task::magic

This field is set when initializing a task. The function [starpu\\_task\\_submit\(\)](#) will fail if the field does not have the right value. This will hence avoid submitting tasks which have not been properly initialised.

#### 27.13.2.3.1.34 unsigned starpu\_task::sched\_ctx

Scheduling context.

#### 27.13.2.3.1.35 int starpu\_task::hypervisor\_tag

Helps the hypervisor monitor the execution of this task.



27.13.2.3.1.36 `starpu_task_bundle_t starpu_task::bundle`

Optional field. The bundle that includes this task. If no bundle is used, this should be NULL.

27.13.2.3.1.37 `struct starpu_profiling_task_info * starpu_task::profiling_info`

Optional field. Profiling information for the task.

27.13.2.3.1.38 `double starpu_task::flops`

This can be set to the number of floating points operations that the task will have to achieve. This is useful for easily getting GFlops curves from the tool `starpu_perfmodel_plot`, and for the hypervisor load balancing.

27.13.2.3.1.39 `double starpu_task::predicted`

Output field. Predicted duration of the task. This field is only set if the scheduling strategy uses performance models.

27.13.2.3.1.40 `double starpu_task::predicted_transfer`

Optional field. Predicted data transfer duration for the task in microseconds. This field is only valid if the scheduling strategy uses performance models.

27.13.2.3.1.41 `struct starpu_task * starpu_task::prev`

A pointer to the previous task. This should only be used by StarPU.

27.13.2.3.1.42 `struct starpu_task * starpu_task::next`

A pointer to the next task. This should only be used by StarPU.

27.13.2.3.1.43 `void * starpu_task::starpu_private`

This is private to StarPU, do not modify. If the task is allocated by hand (without `starpu_task_create()`), this field should be set to NULL.

## 27.13.3 Macro Definition Documentation

27.13.3.1 `#define STARPU_CPU`

This macro is used when setting the field `starpu_codelet::where` to specify the codelet may be executed on a CPU processing unit.

27.13.3.2 `#define STARPU_CUDA`

This macro is used when setting the field `starpu_codelet::where` to specify the codelet may be executed on a CUDA processing unit.

27.13.3.3 `#define STARPU_OPENCL`

This macro is used when setting the field `starpu_codelet::where` to specify the codelet may be executed on a OpenCL processing unit.

27.13.3.4 `#define STARPU_MIC`

This macro is used when setting the field `starpu_codelet::where` to specify the codelet may be executed on a MIC processing unit.

#### 27.13.3.5 `#define STARPU_SCC`

This macro is used when setting the field `starpu_codelet::where` to specify the codelet may be executed on an SCC processing unit.

#### 27.13.3.6 `#define STARPU_MAIN_RAM`

This macro is used when the RAM memory node is specified.

#### 27.13.3.7 `#define STARPU_MULTIPLE_CPU_IMPLEMENTATIONS`

**Deprecated** Setting the field `starpu_codelet::cpu_func` with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field `starpu_codelet::cpu_funcs`.

#### 27.13.3.8 `#define STARPU_MULTIPLE_CUDA_IMPLEMENTATIONS`

**Deprecated** Setting the field `starpu_codelet::cuda_func` with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field `starpu_codelet::cuda_funcs`.

#### 27.13.3.9 `#define STARPU_MULTIPLE_OPENCL_IMPLEMENTATIONS`

**Deprecated** Setting the field `starpu_codelet::opencl_func` with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field `starpu_codelet::opencl_funcs`.

#### 27.13.3.10 `#define STARPU_NMAXBUFS`

Defines the maximum number of buffers that tasks will be able to take as parameters. The default value is 8, it can be changed by using the configure option `--enable-maxbuffers`.

#### 27.13.3.11 `#define STARPU_VARIABLE_NBUFFERS`

Value to set in `starpu_codelet::nbuffers` to specify that the codelet can accept a variable number of buffers, specified in `starpu_task::nbuffers`.

#### 27.13.3.12 `#define STARPU_TASK_INITIALIZER`

It is possible to initialize statically allocated tasks with this value. This is equivalent to initializing a structure `starpu_task` with the function `starpu_task_init()` function.

#### 27.13.3.13 `#define STARPU_TASK_GET_NBUFFERS( task )`

Return the number of buffers for this task, i.e. `starpu_codelet::nbuffers`, or `starpu_task::nbuffers` if the former is `STARPU_VARIABLE_BUFFERS`.

#### 27.13.3.14 `#define STARPU_TASK_GET_HANDLE( task, i )`

Return the *i* th data handle of the given task. If the task is defined with a static or dynamic number of handles, will either return the *i* th element of the field `starpu_task::handles` or the *i* th element of the field `starpu_task::dyn_handles` (see [Setting The Data Handles For A Task](#))

#### 27.13.3.15 `#define STARPU_TASK_SET_HANDLE( task, handle, i )`

Set the *i* th data handle of the given task with the given data handle. If the task is defined with a static or dynamic number of handles, will either set the *i* th element of the field `starpu_task::handles` or the *i* th element of the field `starpu_task::dyn_handles` (see [Setting The Data Handles For A Task](#))

#### 27.13.3.16 `#define STARPU_CODELET_GET_MODE( codelet, i )`

Return the access mode of the *i* th data handle of the given codelet. If the codelet is defined with a static or dynamic number of handles, will either return the *i* th element of the field `starpu_codelet::modes` or the *i* th element of the field `starpu_codelet::dyn_modes` (see [Setting The Data Handles For A Task](#))

#### 27.13.3.17 `#define STARPU_CODELET_SET_MODE( codelet, mode, i )`

Set the access mode of the *i* th data handle of the given codelet. If the codelet is defined with a static or dynamic number of handles, will either set the *i* th element of the field `starpu_codelet::modes` or the *i* th element of the field `starpu_codelet::dyn_modes` (see [Setting The Data Handles For A Task](#))

#### 27.13.3.18 `#define STARPU_TASK_GET_MODE( task, i )`

Return the access mode of the *i* th data handle of the given task. If the task is defined with a static or dynamic number of handles, will either return the *i* th element of the field `starpu_task::modes` or the *i* th element of the field `starpu_task::dyn_modes` (see [Setting The Data Handles For A Task](#))

#### 27.13.3.19 `#define STARPU_TASK_SET_MODE( task, mode, i )`

Set the access mode of the *i* th data handle of the given task. If the task is defined with a static or dynamic number of handles, will either set the *i* th element of the field `starpu_task::modes` or the *i* th element of the field `starpu_task::dyn_modes` (see [Setting The Data Handles For A Task](#))

#### 27.13.3.20 `starpu_task_status::STARPU_TASK_INVALID`

The task has just been initialized.

### 27.13.4 Typedef Documentation

#### 27.13.4.1 `starpu_cpu_func_t`

CPU implementation of a codelet.

#### 27.13.4.2 `starpu_cuda_func_t`

CUDA implementation of a codelet.

#### 27.13.4.3 `starpu_openccl_func_t`

OpenCL implementation of a codelet.

#### 27.13.4.4 `starpu_mic_func_t`

MIC implementation of a codelet.

#### 27.13.4.5 `starpu_scc_func_t`

SCC implementation of a codelet.

#### 27.13.4.6 `starpu_mic_kernel_t`

MIC kernel for a codelet

#### 27.13.4.7 \* `starpu_scc_kernel_t`

SCC kernel for a codelet

### 27.13.5 Enumeration Type Documentation

#### 27.13.5.1 `enum starpu_codelet_type`

Describes the type of parallel task. See [Parallel Tasks](#) for details.

Enumerator

**STARPU\_SEQ** (default) for classical sequential tasks.

**STARPU\_SPMD** for a parallel task whose threads are handled by StarPU, the code has to use [starpu\\_combined\\_worker\\_get\\_size\(\)](#) and [starpu\\_combined\\_worker\\_get\\_rank\(\)](#) to distribute the work.

**STARPU\_FORKJOIN** for a parallel task whose threads are started by the codelet function, which has to use [starpu\\_combined\\_worker\\_get\\_size\(\)](#) to determine how many threads should be started.

#### 27.13.5.2 `enum starpu_task_status`

Task status

Enumerator

**STARPU\_TASK\_BLOCKED** The task has just been submitted, and its dependencies has not been checked yet.

**STARPU\_TASK\_READY** The task is ready for execution.

**STARPU\_TASK\_RUNNING** The task is running on some worker.

**STARPU\_TASK\_FINISHED** The task is finished executing.

**STARPU\_TASK\_BLOCKED\_ON\_TAG** The task is waiting for a tag.

**STARPU\_TASK\_BLOCKED\_ON\_TASK** The task is waiting for a task.

**STARPU\_TASK\_BLOCKED\_ON\_DATA** The task is waiting for some data.

### 27.13.6 Function Documentation

#### 27.13.6.1 void starpu\_codelet\_init ( struct starpu\_codelet \* *cl* )

Initialize *cl* with default values. Codelets should preferably be initialized statically as shown in [Defining A Codelet](#). However such a initialisation is not always possible, e.g. when using C++.

#### 27.13.6.2 void starpu\_task\_init ( struct starpu\_task \* *task* )

Initialize task with default values. This function is implicitly called by [starpu\\_task\\_create\(\)](#). By default, tasks initialized with [starpu\\_task\\_init\(\)](#) must be deinitialized explicitly with [starpu\\_task\\_clean\(\)](#). Tasks can also be initialized statically, using [STARPU\\_TASK\\_INITIALIZER](#).

#### 27.13.6.3 struct starpu\_task \* starpu\_task\_create ( void )

Allocate a task structure and initialize it with default values. Tasks allocated dynamically with [starpu\\_task\\_create\(\)](#) are automatically freed when the task is terminated. This means that the task pointer can not be used any more once the task is submitted, since it can be executed at any time (unless dependencies make it wait) and thus freed at any time. If the field [starpu\\_task::destroy](#) is explicitly unset, the resources used by the task have to be freed by calling [starpu\\_task\\_destroy\(\)](#).

#### 27.13.6.4 struct starpu\_task \* starpu\_task\_dup ( struct starpu\_task \* *task* )

Allocate a task structure which is the exact duplicate of the given task.

#### 27.13.6.5 void starpu\_task\_clean ( struct starpu\_task \* *task* )

Release all the structures automatically allocated to execute task, but not the task structure itself and values set by the user remain unchanged. It is thus useful for statically allocated tasks for instance. It is also useful when users want to execute the same operation several times with as least overhead as possible. It is called automatically by [starpu\\_task\\_destroy\(\)](#). It has to be called only after explicitly waiting for the task or after [starpu\\_shutdown\(\)](#) (waiting for the callback is not enough, since StarPU still manipulates the task after calling the callback).

#### 27.13.6.6 void starpu\_task\_destroy ( struct starpu\_task \* *task* )

Free the resource allocated during [starpu\\_task\\_create\(\)](#) and associated with task. This function is already called automatically after the execution of a task when the field [starpu\\_task::destroy](#) is set, which is the default for tasks created by [starpu\\_task\\_create\(\)](#). Calling this function on a statically allocated task results in an undefined behaviour.

#### 27.13.6.7 int starpu\_task\_wait ( struct starpu\_task \* *task* )

This function blocks until *task* has been executed. It is not possible to synchronize with a task more than once. It is not possible to wait for synchronous or detached tasks. Upon successful completion, this function returns 0. Otherwise, `-EINVAL` indicates that the specified task was either synchronous or detached.

#### 27.13.6.8 int starpu\_task\_submit ( struct starpu\_task \* *task* )

This function submits task to StarPU. Calling this function does not mean that the task will be executed immediately as there can be data or task (tag) dependencies that are not fulfilled yet: StarPU will take care of scheduling this task with respect to such dependencies. This function returns immediately if the field [starpu\\_task::synchronous](#) is set to 0, and block until the termination of the task otherwise. It is also possible to synchronize the application with asynchronous tasks by the means of tags, using the function [starpu\\_tag\\_wait\(\)](#) function for instance. In case of

success, this function returns 0, a return value of `-ENODEV` means that there is no worker able to process this task (e.g. there is no GPU available and this task is only implemented for CUDA devices). `starpu_task_submit()` can be called from anywhere, including codelet functions and callbacks, provided that the field `starpu_task::synchronous` is set to 0.

**27.13.6.9** `int starpu_task_submit_to_ctx ( struct starpu_task * task, unsigned sched_ctx_id )`

This function submits a task to StarPU to the context `sched_ctx_id` . By default `starpu_task_submit` submits the task to a global context that is created automatically by StarPU.

**27.13.6.10** `int starpu_task_wait_for_all ( void )`

This function blocks until all the tasks that were submitted (to the current context or the global one if there aren't any) are terminated. It does not destroy these tasks.

**27.13.6.11** `int starpu_task_wait_for_all_in_ctx ( unsigned sched_ctx_id )`

This function waits until all the tasks that were already submitted to the context `sched_ctx_id` have been executed

**27.13.6.12** `int starpu_task_nready ( void )`

TODO

Return the number of submitted tasks which are ready for execution or already executing. It thus does not include tasks waiting for dependencies.

**27.13.6.13** `int starpu_task_nsubmitted ( void )`

Return the number of submitted tasks which have not completed yet.

**27.13.6.14** `struct starpu_task * starpu_task_get_current ( void )`

This function returns the task currently executed by the worker, or `NULL` if it is called either from a thread that is not a task or simply because there is no task being executed at the moment.

**27.13.6.15** `void starpu_codelet_display_stats ( struct starpu_codelet * cl )`

Output on `stderr` some statistics on the codelet `cl`.

**27.13.6.16** `int starpu_task_wait_for_no_ready ( void )`

This function waits until there is no more ready task.

**27.13.6.17** `void starpu_task_set_implementation ( struct starpu_task * task, unsigned impl )`

This function should be called by schedulers to specify the codelet implementation to be executed when executing the task.

27.13.6.18 unsigned starpu\_task\_get\_implementation ( struct starpu\_task \* task )

This function return the codelet implementation to be executed when executing the task.

27.13.6.19 void starpu\_create\_sync\_task ( starpu\_tag\_t sync\_tag, unsigned ndeps, starpu\_tag\_t \* deps, void(\*) (void \*) callback, void \* callback\_arg )

This creates (and submits) an empty task that unlocks a tag once all its dependencies are fulfilled.

## 27.14 Insert\_Task

### Macros

- #define STARPU\_VALUE
- #define STARPU\_CALLBACK
- #define STARPU\_CALLBACK\_WITH\_ARG
- #define STARPU\_CALLBACK\_ARG
- #define STARPU\_PRIORITY
- #define STARPU\_DATA\_ARRAY
- #define STARPU\_DATA\_MODE\_ARRAY
- #define STARPU\_EXECUTE\_ON\_WORKER
- #define STARPU\_WORKER\_ORDER
- #define STARPU\_TAG
- #define STARPU\_TAG\_ONLY
- #define STARPU\_FLOPS
- #define STARPU\_SCHED\_CTX

### Functions

- int starpu\_insert\_task (struct starpu\_codelet \*cl,...)
- int starpu\_task\_insert (struct starpu\_codelet \*cl,...)
- void starpu\_codelet\_pack\_args (void \*\*arg\_buffer, size\_t \*arg\_buffer\_size,...)
- void starpu\_codelet\_unpack\_args (void \*cl\_arg,...)
- struct starpu\_task \* starpu\_task\_build (struct starpu\_codelet \*cl,...)

#### 27.14.1 Detailed Description

#### 27.14.2 Macro Definition Documentation

##### 27.14.2.1 #define STARPU\_VALUE

this macro is used when calling [starpu\\_task\\_insert\(\)](#), and must be followed by a pointer to a constant value and the size of the constant

##### 27.14.2.2 #define STARPU\_CALLBACK

this macro is used when calling [starpu\\_task\\_insert\(\)](#), and must be followed by a pointer to a callback function

#### 27.14.2.3 `#define STARPU_CALLBACK_WITH_ARG`

this macro is used when calling `starpu_task_insert()`, and must be followed by two pointers: one to a callback function, and the other to be given as an argument to the callback function; this is equivalent to using both `STARPU_CALLBACK` and `STARPU_CALLBACK_WITH_ARG`.

#### 27.14.2.4 `#define STARPU_CALLBACK_ARG`

this macro is used when calling `starpu_task_insert()`, and must be followed by a pointer to be given as an argument to the callback function

#### 27.14.2.5 `#define STARPU_PRIORITY`

this macro is used when calling `starpu_task_insert()`, and must be followed by a integer defining a priority level

#### 27.14.2.6 `#define STARPU_DATA_ARRAY`

TODO

#### 27.14.2.7 `#define STARPU_DATA_MODE_ARRAY`

TODO

#### 27.14.2.8 `#define STARPU_EXECUTE_ON_WORKER`

this macro is used when calling `starpu_task_insert()`, and must be followed by an integer value specifying the worker on which to execute the task (as specified by `starpu_task::execute_on_a_specific_worker`)

#### 27.14.2.9 `#define STARPU_WORKER_ORDER`

this macro is used when calling `starpu_task_insert()`, and must be followed by an integer value specifying the worker order in which to execute the tasks (as specified by `starpu_task::workerorder`)

#### 27.14.2.10 `#define STARPU_TAG`

this macro is used when calling `starpu_task_insert()`, and must be followed by a tag.

#### 27.14.2.11 `#define STARPU_TAG_ONLY`

this macro is used when calling `starpu_task_insert()`, and must be followed by a tag. It sets `starpu_task::tag_id`, but leaves `starpu_task::use_tag` as 0.

#### 27.14.2.12 `#define STARPU_FLOPS`

this macro is used when calling `starpu_task_insert()`, and must be followed by an amount of floating point operations, as a double. Users **MUST** explicitly cast into double, otherwise parameter passing will not work.

#### 27.14.2.13 `#define STARPU_SCHED_CTX`

this macro is used when calling `starpu_task_insert()`, and must be followed by the id of the scheduling context to which we want to submit the task.



### 27.14.3 Function Documentation

#### 27.14.3.1 `starpu_insert_task ( struct starpu_codelet * cl, ... )`

This function does the same as the function `starpu_task_insert()`. It has been kept to avoid breaking old codes.

#### 27.14.3.2 `int starpu_task_insert ( struct starpu_codelet * cl, ... )`

Create and submit a task corresponding to `cl` with the following arguments. The argument list must be zero-terminated.

The arguments following the codelet can be of the following types:

- `STARPU_R`, `STARPU_W`, `STARPU_RW`, `STARPU_SCRATCH`, `STARPU_REDUX` an access mode followed by a data handle;
- `STARPU_DATA_ARRAY` followed by an array of data handles and its number of elements;
- `STARPU_DATA_MODE_ARRAY` followed by an array of struct `starpu_data_descr`, i.e data handles with their associated access modes, and its number of elements;
- `STARPU_EXECUTE_ON_WORKER`, `STARPU_WORKER_ORDER` followed by an integer value specifying the worker on which to execute the task (as specified by `starpu_task::execute_on_a_specific_worker`)
- the specific values `STARPU_VALUE`, `STARPU_CALLBACK`, `STARPU_CALLBACK_ARG`, `STARPU_CALLBACK_WITH_ARG`, `STARPU_PRIORITY`, `STARPU_TAG`, `STARPU_TAG_ONLY`, `STARPU_FLOPS`, `STARPU_SCHED_CTX` followed by the appropriated objects as defined elsewhere.

When using `STARPU_DATA_ARRAY`, the access mode of the data handles is not defined. One should use `STARPU_DATA_MODE_ARRAY` to define the data handles along with the access modes.

Parameters to be passed to the codelet implementation are defined through the type `STARPU_VALUE`. The function `starpu_codelet_unpack_args()` must be called within the codelet implementation to retrieve them.

#### 27.14.3.3 `void starpu_codelet_pack_args ( void ** arg_buffer, size_t * arg_buffer_size, ... )`

Pack arguments of type `STARPU_VALUE` into a buffer which can be given to a codelet and later unpacked with the function `starpu_codelet_unpack_args()`.

#### 27.14.3.4 `void starpu_codelet_unpack_args ( void * cl_arg, ... )`

Retrieve the arguments of type `STARPU_VALUE` associated to a task automatically created using the function `starpu_task_insert()`.

#### 27.14.3.5 `struct starpu_task * starpu_task_build ( struct starpu_codelet * cl, ... )`

Create a task corresponding to `cl` with the following arguments. The argument list must be zero-terminated. The arguments following the codelet are the same as the ones for the function `starpu_task_insert()`.

## 27.15 Explicit Dependencies

### Typedefs

- `typedef uint64_t starpu_tag_t`

## Functions

- void `starpu_task_declare_deps_array` (struct `starpu_task` \*task, unsigned ndeps, struct `starpu_task` \*task←\_array[])
- void `starpu_tag_declare_deps` (starpu\_tag\_t id, unsigned ndeps,...)
- void `starpu_tag_declare_deps_array` (starpu\_tag\_t id, unsigned ndeps, starpu\_tag\_t \*array)
- int `starpu_tag_wait` (starpu\_tag\_t id)
- int `starpu_tag_wait_array` (unsigned ntags, starpu\_tag\_t \*id)
- void `starpu_tag_restart` (starpu\_tag\_t id)
- void `starpu_tag_remove` (starpu\_tag\_t id)
- void `starpu_tag_notify_from_apps` (starpu\_tag\_t id)

### 27.15.1 Detailed Description

### 27.15.2 Typedef Documentation

#### 27.15.2.1 starpu\_tag\_t

This type defines a task logical identifier. It is possible to associate a task with a unique *tag* chosen by the application, and to express dependencies between tasks by the means of those tags. To do so, fill the field `starpu_task::tag_id` with a tag number (can be arbitrary) and set the field `starpu_task::use_tag` to 1. If `starpu_tag_declare_deps()` is called with this tag number, the task will not be started until the tasks which holds the declared dependency tags are completed.

### 27.15.3 Function Documentation

#### 27.15.3.1 void starpu\_task\_declare\_deps\_array ( struct starpu\_task \* task, unsigned ndeps, struct starpu\_task \* task\_array[] )

Declare task dependencies between a `task` and an array of tasks of length `ndeps`. This function must be called prior to the submission of the task, but it may be called after the submission or the execution of the tasks in the array, provided the tasks are still valid (i.e. they were not automatically destroyed). Calling this function on a task that was already submitted or with an entry of `task_array` that is no longer a valid task results in an undefined behaviour. If `ndeps` is 0, no dependency is added. It is possible to call `starpu_task_declare_deps_array()` several times on the same task, in this case, the dependencies are added. It is possible to have redundancy in the task dependencies.

#### 27.15.3.2 void starpu\_tag\_declare\_deps ( starpu\_tag\_t id, unsigned ndeps, ... )

Specify the dependencies of the task identified by tag `id`. The first argument specifies the tag which is configured, the second argument gives the number of tag(s) on which `id` depends. The following arguments are the tags which have to be terminated to unlock the task. This function must be called before the associated task is submitted to StarPU with `starpu_task_submit()`.

**WARNING! Use with caution.** Because of the variable arity of `starpu_tag_declare_deps()`, note that the last arguments must be of type `starpu_tag_t`: constant values typically need to be explicitly casted. Otherwise, due to integer sizes and argument passing on the stack, the C compiler might consider the tag `0x200000003` instead of `0x2` and `0x3` when calling `starpu_tag_declare_deps(0x1, 2, 0x2, 0x3)`. Using the `starpu_tag←_declare_deps_array()` function avoids this hazard.

```
/* Tag 0x1 depends on tags 0x32 and 0x52 */
starpu_tag_declare_deps((starpu_tag_t)0x1, 2, (
 starpu_tag_t)0x32, (starpu_tag_t)0x52);
```

### 27.15.3.3 void starpu\_tag\_declare\_deps\_array ( starpu\_tag\_t id, unsigned ndeps, starpu\_tag\_t \* array )

This function is similar to [starpu\\_tag\\_declare\\_deps\(\)](#), except that it does not take a variable number of arguments but an array of tags of size `ndeps`.

```
/* Tag 0x1 depends on tags 0x32 and 0x52 */
starpu_tag_t tag_array[2] = {0x32, 0x52};
starpu_tag_declare_deps_array((starpu_tag_t)0x1, 2, tag_array);
```

### 27.15.3.4 int starpu\_tag\_wait ( starpu\_tag\_t id )

This function blocks until the task associated to tag `id` has been executed. This is a blocking call which must therefore not be called within tasks or callbacks, but only from the application directly. It is possible to synchronize with the same tag multiple times, as long as the [starpu\\_tag\\_remove\(\)](#) function is not called. Note that it is still possible to synchronize with a tag associated to a task for which the structure [starpu\\_task](#) was freed (e.g. if the field [starpu\\_task::destroy](#) was enabled).

### 27.15.3.5 int starpu\_tag\_wait\_array ( unsigned ntags, starpu\_tag\_t \* id )

This function is similar to [starpu\\_tag\\_wait\(\)](#) except that it blocks until all the `ntags` tags contained in the array `id` are terminated.

### 27.15.3.6 void starpu\_tag\_restart ( starpu\_tag\_t id )

This function can be used to clear the *already notified* status of a tag which is not associated with a task. Before that, calling [starpu\\_tag\\_notify\\_from\\_apps\(\)](#) again will not notify the successors. After that, the next call to [starpu\\_tag\\_notify\\_from\\_apps\(\)](#) will notify the successors.

### 27.15.3.7 void starpu\_tag\_remove ( starpu\_tag\_t id )

This function releases the resources associated to tag `id`. It can be called once the corresponding task has been executed and when there is no other tag that depend on this tag anymore.

### 27.15.3.8 void starpu\_tag\_notify\_from\_apps ( starpu\_tag\_t id )

This function explicitly unlocks tag `id`. It may be useful in the case of applications which execute part of their computation outside StarPU tasks (e.g. third-party libraries). It is also provided as a convenient tool for the programmer, for instance to entirely construct the task DAG before actually giving StarPU the opportunity to execute the tasks. When called several times on the same tag, notification will be done only on first call, thus implementing "OR" dependencies, until the tag is restarted using [starpu\\_tag\\_restart\(\)](#).

## 27.16 Implicit Data Dependencies

In this section, we describe how StarPU makes it possible to insert implicit task dependencies in order to enforce sequential data consistency. When this data consistency is enabled on a specific data handle, any data access will appear as sequentially consistent from the application. For instance, if the application submits two tasks that access the same piece of data in read-only mode, and then a third task that access it in write mode, dependencies will be added between the two first tasks and the third one. Implicit data dependencies are also inserted in the case of data accesses from the application.

## Functions

- void [starpu\\_data\\_set\\_default\\_sequential\\_consistency\\_flag](#) (unsigned flag)
- unsigned [starpu\\_data\\_get\\_default\\_sequential\\_consistency\\_flag](#) (void)
- void [starpu\\_data\\_set\\_sequential\\_consistency\\_flag](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned flag)
- unsigned [starpu\\_data\\_get\\_sequential\\_consistency\\_flag](#) ([starpu\\_data\\_handle\\_t](#) handle)

### 27.16.1 Detailed Description

In this section, we describe how StarPU makes it possible to insert implicit task dependencies in order to enforce sequential data consistency. When this data consistency is enabled on a specific data handle, any data access will appear as sequentially consistent from the application. For instance, if the application submits two tasks that access the same piece of data in read-only mode, and then a third task that access it in write mode, dependencies will be added between the two first tasks and the third one. Implicit data dependencies are also inserted in the case of data accesses from the application.

### 27.16.2 Function Documentation

#### 27.16.2.1 [starpu\\_data\\_set\\_default\\_sequential\\_consistency\\_flag](#) ( unsigned *flag* )

Set the default sequential consistency flag. If a non-zero value is passed, a sequential data consistency will be enforced for all handles registered after this function call, otherwise it is disabled. By default, StarPU enables sequential data consistency. It is also possible to select the data consistency mode of a specific data handle with the function [starpu\\_data\\_set\\_sequential\\_consistency\\_flag](#)().

#### 27.16.2.2 unsigned [starpu\\_data\\_get\\_default\\_sequential\\_consistency\\_flag](#) ( void )

Return the default sequential consistency flag

#### 27.16.2.3 void [starpu\\_data\\_set\\_sequential\\_consistency\\_flag](#) ( [starpu\\_data\\_handle\\_t](#) *handle*, unsigned *flag* )

Set the data consistency mode associated to a data handle. The consistency mode set using this function has the priority over the default mode which can be set with [starpu\\_data\\_set\\_default\\_sequential\\_consistency\\_flag](#)().

#### 27.16.2.4 unsigned [starpu\\_data\\_get\\_sequential\\_consistency\\_flag](#) ( [starpu\\_data\\_handle\\_t](#) *handle* )

Get the data consistency mode associated to the data handle *handle*

## 27.17 Performance Model

### Data Structures

- struct [starpu\\_perfmodel](#)
- struct [starpu\\_perfmodel\\_regression\\_model](#)
- struct [starpu\\_perfmodel\\_per\\_arch](#)
- struct [starpu\\_perfmodel\\_history\\_list](#)
- struct [starpu\\_perfmodel\\_history\\_entry](#)

## Enumerations

- enum `starpu_perfmodel_type` {  
**STARPU\_PERFMODEL\_INVALID**, **STARPU\_PER\_ARCH**, **STARPU\_COMMON**, **STARPU\_HISTORY\_B←**  
**ASED**,  
**STARPU\_REGRESSION\_BASED**, **STARPU\_NL\_REGRESSION\_BASED** }

## Functions

- void `starpu_perfmodel_init` (FILE \*f, struct `starpu_perfmodel` \*model)
- void `starpu_perfmodel_free_sampling_directories` (void)
- int `starpu_perfmodel_load_symbol` (const char \*symbol, struct `starpu_perfmodel` \*model)
- int `starpu_perfmodel_unload_model` (struct `starpu_perfmodel` \*model)
- void `starpu_perfmodel_debugfilepath` (struct `starpu_perfmodel` \*model, struct `starpu_perfmodel_arch` \*arch, char \*path, size\_t maxlen, unsigned nimpl)
- char \* `starpu_perfmodel_get_archtype_name` (enum `starpu_worker_archtype` archtype)
- void `starpu_perfmodel_get_arch_name` (struct `starpu_perfmodel_arch` \*arch, char \*archname, size\_t maxlen, unsigned nimpl)
- struct `starpu_perfmodel_arch` \* `starpu_worker_get_perf_archtype` (int workerid, unsigned sched\_ctx\_id)
- int `starpu_perfmodel_list` (FILE \*output)
- void `starpu_perfmodel_directory` (FILE \*output)
- void `starpu_perfmodel_print` (struct `starpu_perfmodel` \*model, struct `starpu_perfmodel_arch` \*arch, unsigned nimpl, char \*parameter, uint32\_t \*footprint, FILE \*output)
- int `starpu_perfmodel_print_all` (struct `starpu_perfmodel` \*model, char \*arch, char \*parameter, uint32\_t \*footprint, FILE \*output)
- void `starpu_bus_print_bandwidth` (FILE \*f)
- void `starpu_bus_print_affinity` (FILE \*f)
- void `starpu_perfmodel_update_history` (struct `starpu_perfmodel` \*model, struct `starpu_task` \*task, struct `starpu_perfmodel_arch` \*arch, unsigned cpuid, unsigned nimpl, double measured)
- double `starpu_transfer_bandwidth` (unsigned src\_node, unsigned dst\_node)
- double `starpu_transfer_latency` (unsigned src\_node, unsigned dst\_node)
- double `starpu_transfer_predict` (unsigned src\_node, unsigned dst\_node, size\_t size)
- double `starpu_perfmodel_history_based_expected_perf` (struct `starpu_perfmodel` \*model, struct `starpu_perfmodel_arch` \*arch, uint32\_t footprint)

### 27.17.1 Detailed Description

### 27.17.2 Data Structure Documentation

#### 27.17.2.1 struct `starpu_perfmodel`

Contains all information about a performance model. At least the type and symbol fields have to be filled when defining a performance model for a codelet. For compatibility, make sure to initialize the whole structure to zero, either by using explicit memset, or by letting the compiler implicitly do it in e.g. static storage case. If not provided, other fields have to be zero.

#### Data Fields

- enum `starpu_perfmodel_type` type
- double(\* `cost_function`) (struct `starpu_task` \*, unsigned nimpl)
- size\_t(\* `size_base`) (struct `starpu_task` \*, unsigned nimpl)
- uint32\_t(\* `footprint`) (struct `starpu_task` \*)
- const char \* `symbol`
- unsigned `is_loaded`

- unsigned **benchmarking**
- unsigned [is\\_init](#)
- starpu\_perfmodel\_state\_t **state**

#### 27.17.2.1.1 Field Documentation

##### 27.17.2.1.1.1 enum starpu\_perfmodel\_type starpu\_perfmodel::type

is the type of performance model

- [STARPU\\_HISTORY\\_BASED](#), [STARPU\\_REGRESSION\\_BASED](#), [STARPU\\_NL\\_REGRESSION\\_BASED](#)↵: No other fields needs to be provided, this is purely history-based.
- [STARPU\\_PER\\_ARCH](#): field `starpu_perfmodel::per_arch` has to be filled with functions which return the cost in micro-seconds.
- [STARPU\\_COMMON](#): field `starpu_perfmodel::cost_function` has to be filled with a function that returns the cost in micro-seconds on a CPU, timing on other archs will be determined by multiplying by an arch-specific factor.

##### 27.17.2.1.1.2 double(\* starpu\_perfmodel::cost\_function)(struct starpu\_task \*, unsigned nimpl)

Used by [STARPU\\_COMMON](#): takes a task and implementation number, and must return a task duration estimation in micro-seconds.

##### 27.17.2.1.1.3 size\_t(\* starpu\_perfmodel::size\_base)(struct starpu\_task \*, unsigned nimpl)

Used by [STARPU\\_HISTORY\\_BASED](#), [STARPU\\_REGRESSION\\_BASED](#) and [STARPU\\_NL\\_REGRESSION\\_BASED](#)↵. If not NULL, takes a task and implementation number, and returns the size to be used as index to distinguish histories and as a base for regressions.

##### 27.17.2.1.1.4 uint32\_t(\* starpu\_perfmodel::footprint)(struct starpu\_task \*)

Used by [STARPU\\_HISTORY\\_BASED](#). If not NULL, takes a task and returns the footprint to be used as index to distinguish histories. The default is to use the `starpu_task_data_footprint` function.

##### 27.17.2.1.1.5 const char \* starpu\_perfmodel::symbol

is the symbol name for the performance model, which will be used as file name to store the model. It must be set otherwise the model will be ignored.

##### 27.17.2.1.1.6 unsigned starpu\_perfmodel::is\_loaded

Whether the performance model is already loaded from the disk.

##### 27.17.2.1.1.7 unsigned starpu\_perfmodel::is\_init

todo

#### 27.17.2.2 struct starpu\_perfmodel\_regression\_model

...

##### Data Fields

|        |        |                     |
|--------|--------|---------------------|
| double | sumlny | sum of ln(measured) |
|--------|--------|---------------------|

|               |           |                                                                         |
|---------------|-----------|-------------------------------------------------------------------------|
| double        | sumlnx    | sum of $\ln(\text{size})$                                               |
| double        | sumlnx2   | sum of $\ln(\text{size})^2$                                             |
| unsigned long | minx      | minimum size                                                            |
| unsigned long | maxx      | maximum size                                                            |
| double        | sumlnxlny | sum of $\ln(\text{size}) * \ln(\text{measured})$                        |
| double        | alpha     | estimated = $\alpha * \text{size}^\beta$                                |
| double        | beta      | estimated = $\alpha * \text{size}^\beta$                                |
| unsigned      | valid     | whether the linear regression model is valid (i.e. enough measures)     |
| double        | a         | estimated = $a * \text{size}^b + c$                                     |
| double        | b         | estimated = $a * \text{size}^b + c$                                     |
| double        | c         | estimated = $a * \text{size}^b + c$                                     |
| unsigned      | nl_valid  | whether the non-linear regression model is valid (i.e. enough measures) |
| unsigned      | nsample   | number of sample values for non-linear regression                       |

### 27.17.2.3 struct starpu\_perfmodel\_per\_arch

contains information about the performance model of a given arch.

#### Data Fields

|                                          |                 |                                                                                                                                                                                                                                                                 |
|------------------------------------------|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| starpu_perfmodel_per_arch_cost_function  | cost_function   | Used by <a href="#">STARPU_PER_ARCH</a> , must point to functions which take a task, the target arch and implementation number (as mere conveniency, since the array is already indexed by these), and must return a task duration estimation in micro-seconds. |
| starpu_perfmodel_per_arch_size_base      | size_base       | Same as in structure <a href="#">starpu_perfmodel</a> , but per-arch, in case it depends on the architecture-specific implementation.                                                                                                                           |
| struct starpu_perfmodel_history_table *  | history         | The history of performance measurements.                                                                                                                                                                                                                        |
| struct starpu_perfmodel_history_list *   | list            | Used by <a href="#">STARPU_HISTORY_BASED</a> and <a href="#">STARPU_NL_REGRESSION_BASED</a> , records all execution history measures.                                                                                                                           |
| struct starpu_perfmodel_regression_model | regression      | Used by <a href="#">STARPU_REGRESSION_BASED</a> and <a href="#">STARPU_NL_REGRESSION_BASED</a> , contains the estimated factors of the regression.                                                                                                              |
| char                                     | debug_path[256] |                                                                                                                                                                                                                                                                 |

### 27.17.2.4 struct starpu\_perfmodel\_history\_list

todo

## Data Fields

|                                                         |       |      |
|---------------------------------------------------------|-------|------|
| struct<br>starpu_↔<br>perfmodel_↔<br>history_list<br>*  | next  | todo |
| struct<br>starpu_↔<br>perfmodel_↔<br>history_entry<br>* | entry | todo |

## 27.17.2.5 struct starpu\_perfmodel\_history\_entry

todo

## Data Fields

|          |           |                              |
|----------|-----------|------------------------------|
| double   | mean      | mean_n = 1/n sum             |
| double   | deviation | n dev_n = sum2 - 1/n (sum)^2 |
| double   | sum       | sum of samples (in $\mu$ s)  |
| double   | sum2      | sum of samples^2             |
| unsigned | nsample   | number of samples            |
| unsigned | nerror    |                              |
| uint32_t | footprint | data footprint               |
| size_t   | size      | in bytes                     |
| double   | flops     | Provided by the application  |

## 27.17.3 Enumeration Type Documentation

## 27.17.3.1 enum starpu\_perfmodel\_type

TODO

## Enumerator

**STARPU\_PER\_ARCH** Application-provided per-arch cost model function**STARPU\_COMMON** Application-provided common cost model function, with per-arch factor**STARPU\_HISTORY\_BASED** Automatic history-based cost model**STARPU\_REGRESSION\_BASED** Automatic linear regression-based cost model ( $\alpha * \text{size}^\beta$ )**STARPU\_NL\_REGRESSION\_BASED** Automatic non-linear regression-based cost model ( $a * \text{size}^\beta + b + c$ )

## 27.17.4 Function Documentation

## 27.17.4.1 void starpu\_perfmodel\_init ( FILE \* f, struct starpu\_perfmodel \* model )

todo

## 27.17.4.2 void starpu\_perfmodel\_free\_sampling\_directories ( void )

this function frees internal memory used for sampling directory management. It should only be called by an application which is not calling starpu\_shutdown as this function already calls it. See for example `tools/starpu_↔perfmodel_display.c`.



**27.17.4.3** `int starpu_perfmmodel_load_symbol ( const char * symbol, struct starpu_perfmmodel * model )`

loads a given performance model. The model structure has to be completely zero, and will be filled with the information saved in `$STARPU_HOME/.starpu`. The function is intended to be used by external tools that should read the performance model files.

**27.17.4.4** `int starpu_perfmmodel_unload_model ( struct starpu_perfmmodel * model )`

unloads the given model which has been previously loaded through the function [starpu\\_perfmmodel\\_load\\_symbol\(\)](#)

**27.17.4.5** `void starpu_perfmmodel_debugfilepath ( struct starpu_perfmmodel * model, struct starpu_perfmmodel_arch * arch, char * path, size_t maxlen, unsigned nimpl )`

returns the path to the debugging information for the performance model.

**27.17.4.6** `char * starpu_perfmmodel_get_archtype_name ( enum starpu_worker_archtype archtype )`

todo

**27.17.4.7** `void starpu_perfmmodel_get_arch_name ( struct starpu_perfmmodel_arch * arch, char * archname, size_t maxlen, unsigned nimpl )`

returns the architecture name for `arch`

**27.17.4.8** `struct starpu_perfmmodel_arch * starpu_worker_get_perf_archtype ( int workerid, unsigned sched_ctx_id )`

returns the architecture type of a given worker.

**27.17.4.9** `int starpu_perfmmodel_list ( FILE * output )`

prints a list of all performance models on `output`

**27.17.4.10** `void starpu_perfmmodel_directory ( FILE * output )`

prints the directory name storing performance models on `output`

**27.17.4.11** `void starpu_perfmmodel_print ( struct starpu_perfmmodel * model, struct starpu_perfmmodel_arch * arch, unsigned nimpl, char * parameter, uint32_t * footprint, FILE * output )`

todo

**27.17.4.12** `int starpu_perfmmodel_print_all ( struct starpu_perfmmodel * model, char * arch, char * parameter, uint32_t * footprint, FILE * output )`

todo

**27.17.4.13** `void starpu_bus_print_bandwidth ( FILE * f )`

prints a matrix of bus bandwidths on `f`.

27.17.4.14 `void starpu_bus_print_affinity ( FILE * f )`

prints the affinity devices on `f`.

27.17.4.15 `void starpu_perfmodel_update_history ( struct starpu_perfmodel * model, struct starpu_task * task, struct starpu_perfmodel_arch * arch, unsigned cpuid, unsigned nimpl, double measured )`

This feeds the performance model `model` with an explicit measurement `measured` (in  $\mu$ s), in addition to measurements done by StarPU itself. This can be useful when the application already has an existing set of measurements done in good conditions, that StarPU could benefit from instead of doing on-line measurements. And example of use can be seen in [Performance Model Example](#).

27.17.4.16 `double starpu_transfer_bandwidth ( unsigned src_node, unsigned dst_node )`

Return the bandwidth of data transfer between two memory nodes

27.17.4.17 `double starpu_transfer_latency ( unsigned src_node, unsigned dst_node )`

Return the latency of data transfer between two memory nodes

27.17.4.18 `double starpu_transfer_predict ( unsigned src_node, unsigned dst_node, size_t size )`

Return the estimated time to transfer a given size between two memory nodes.

27.17.4.19 `double starpu_permodel_history_based_expected_perf ( struct starpu_perfmodel * model, struct starpu_perfmodel_arch * arch, uint32_t footprint )`

todo

## 27.18 Profiling

### Data Structures

- struct [starpu\\_profiling\\_task\\_info](#)
- struct [starpu\\_profiling\\_worker\\_info](#)
- struct [starpu\\_profiling\\_bus\\_info](#)

### Macros

- `#define` [STARPU\\_PROFILING\\_DISABLE](#)
- `#define` [STARPU\\_PROFILING\\_ENABLE](#)

### Functions

- int [starpu\\_profiling\\_status\\_set](#) (int status)
- int [starpu\\_profiling\\_status\\_get](#) (void)
- void [starpu\\_profiling\\_init](#) (void)
- void [starpu\\_profiling\\_set\\_id](#) (int new\_id)
- int [starpu\\_profiling\\_worker\\_get\\_info](#) (int workerid, struct [starpu\\_profiling\\_worker\\_info](#) \*worker\_info)
- int [starpu\\_bus\\_get\\_profiling\\_info](#) (int busid, struct [starpu\\_profiling\\_bus\\_info](#) \*bus\_info)

- int [starpu\\_bus\\_get\\_count](#) (void)
- int [starpu\\_bus\\_get\\_id](#) (int src, int dst)
- int [starpu\\_bus\\_get\\_src](#) (int busid)
- int [starpu\\_bus\\_get\\_dst](#) (int busid)
- double [starpu\\_timing\\_timespec\\_delay\\_us](#) (struct timespec \*start, struct timespec \*end)
- double [starpu\\_timing\\_timespec\\_to\\_us](#) (struct timespec \*ts)
- void [starpu\\_profiling\\_bus\\_helper\\_display\\_summary](#) (void)
- void [starpu\\_profiling\\_worker\\_helper\\_display\\_summary](#) (void)
- void [starpu\\_data\\_display\\_memory\\_stats](#) ()

### 27.18.1 Detailed Description

### 27.18.2 Data Structure Documentation

#### 27.18.2.1 struct starpu\_profiling\_task\_info

This structure contains information about the execution of a task. It is accessible from the field [starpu\\_task->::profiling\\_info](#) if profiling was enabled.

##### Data Fields

|                 |                              |                                                                                            |
|-----------------|------------------------------|--------------------------------------------------------------------------------------------|
| struct timespec | submit_time                  | Date of task submission (relative to the initialization of StarPU).                        |
| struct timespec | push_start_time              | Time when the task was submitted to the scheduler.                                         |
| struct timespec | push_end_time                | Time when the scheduler finished with the task submission.                                 |
| struct timespec | pop_start_time               | Time when the scheduler started to be requested for a task, and eventually gave that task. |
| struct timespec | pop_end_time                 | Time when the scheduler finished providing the task for execution.                         |
| struct timespec | acquire_data_↔<br>start_time | Time when the worker started fetching input data.                                          |
| struct timespec | acquire_data_↔<br>end_time   | Time when the worker finished fetching input data.                                         |
| struct timespec | start_time                   | Date of task execution beginning (relative to the initialization of StarPU).               |
| struct timespec | end_time                     | Date of task execution termination (relative to the initialization of StarPU).             |
| struct timespec | release_data_↔<br>start_time | Time when the worker started releasing data.                                               |
| struct timespec | release_data_↔<br>end_time   | Time when the worker finished releasing data.                                              |
| struct timespec | callback_start_↔<br>time     | Time when the worker started the application callback for the task.                        |
| struct timespec | callback_end_↔<br>time       | Time when the worker finished the application callback for the task.                       |
| int             | workerid                     | Identifier of the worker which has executed the task.                                      |
| uint64_t        | used_cycles                  | Number of cycles used by the task, only available in the Movisim                           |
| uint64_t        | stall_cycles                 | Number of cycles stalled within the task, only available in the Movisim                    |
| double          | power_↔<br>consumed          | Power consumed by the task, only available in the Movisim                                  |

#### 27.18.2.2 struct starpu\_profiling\_worker\_info

This structure contains the profiling information associated to a worker. The timing is provided since the previous call to [starpu\\_profiling\\_worker\\_get\\_info\(\)](#)

## Data Fields

|                 |                     |                                                                                      |
|-----------------|---------------------|--------------------------------------------------------------------------------------|
| struct timespec | start_time          | Starting date for the reported profiling measurements.                               |
| struct timespec | total_time          | Duration of the profiling measurement interval.                                      |
| struct timespec | executing_time      | Time spent by the worker to execute tasks during the profiling measurement interval. |
| struct timespec | sleeping_time       | Time spent idling by the worker during the profiling measurement interval.           |
| int             | executed_tasks      | Number of tasks executed by the worker during the profiling measurement interval.    |
| uint64_t        | used_cycles         | Number of cycles used by the worker, only available in the MoviSim                   |
| uint64_t        | stall_cycles        | Number of cycles stalled within the worker, only available in the MoviSim            |
| double          | power_↔<br>consumed | Power consumed by the worker, only available in the MoviSim                          |

## 27.18.2.3 struct starpu\_profiling\_bus\_info

todo

## Data Fields

|                 |                        |                                               |
|-----------------|------------------------|-----------------------------------------------|
| struct timespec | start_time             | Time of bus profiling startup.                |
| struct timespec | total_time             | Total time of bus profiling.                  |
| int long long   | transferred_↔<br>bytes | Number of bytes transferred during profiling. |
| int             | transfer_count         | Number of transfers during profiling.         |

## 27.18.3 Macro Definition Documentation

## 27.18.3.1 STARPU\_PROFILING\_DISABLE

This value is used when calling the function [starpu\\_profiling\\_status\\_set\(\)](#) to disable profiling.

## 27.18.3.2 STARPU\_PROFILING\_ENABLE

This value is used when calling the function [starpu\\_profiling\\_status\\_set\(\)](#) to enable profiling.

## 27.18.4 Function Documentation

## 27.18.4.1 int starpu\_profiling\_status\_set ( int status )

This function sets the profiling status. Profiling is activated by passing [STARPU\\_PROFILING\\_ENABLE](#) in status. Passing [STARPU\\_PROFILING\\_DISABLE](#) disables profiling. Calling this function resets all profiling measurements. When profiling is enabled, the field [starpu\\_task::profiling\\_info](#) points to a valid structure [starpu\\_profiling\\_task\\_↔info](#) containing information about the execution of the task. Negative return values indicate an error, otherwise the previous status is returned.

## 27.18.4.2 int starpu\_profiling\_status\_get ( void )

Return the current profiling status or a negative value in case there was an error.

## 27.18.4.3 void starpu\_profiling\_init ( void )

This function resets performance counters and enable profiling if the environment variable [STARPU\\_PROFILING](#) is set to a positive value.

27.18.4.4 `void starpu_profiling_set_id ( int new_id )`

This function sets the ID used for profiling trace filename. It needs to be called before [starpu\\_init\(\)](#).

27.18.4.5 `int starpu_profiling_worker_get_info ( int workerid, struct starpu_profiling_worker_info * worker_info )`

Get the profiling info associated to the worker identified by `workerid`, and reset the profiling measurements. If the argument `worker_info` is NULL, only reset the counters associated to worker `workerid`. Upon successful completion, this function returns 0. Otherwise, a negative value is returned.

27.18.4.6 `int starpu_bus_get_profiling_info ( int busid, struct starpu_profiling_bus_info * bus_info )`

todo

27.18.4.7 `int starpu_bus_get_count ( void )`

Return the number of buses in the machine

27.18.4.8 `int starpu_bus_get_id ( int src, int dst )`

Return the identifier of the bus between `src` and `dst`

27.18.4.9 `int starpu_bus_get_src ( int busid )`

Return the source point of bus `busid`

27.18.4.10 `int starpu_bus_get_dst ( int busid )`

Return the destination point of bus `busid`

27.18.4.11 `double starpu_timing_timespec_delay_us ( struct timespec * start, struct timespec * end )`

Returns the time elapsed between `start` and `end` in microseconds.

27.18.4.12 `double starpu_timing_timespec_to_us ( struct timespec * ts )`

Converts the given timespec `ts` into microseconds

27.18.4.13 `void starpu_profiling_bus_helper_display_summary ( void )`

Displays statistics about the bus on stderr. if the environment variable [STARPU\\_BUS\\_STATS](#) is defined. The function is called automatically by [starpu\\_shutdown\(\)](#).

27.18.4.14 `void starpu_profiling_worker_helper_display_summary ( void )`

Displays statistics about the workers on stderr if the environment variable [STARPU\\_WORKER\\_STATS](#) is defined. The function is called automatically by [starpu\\_shutdown\(\)](#).

#### 27.18.4.15 void starpu\_data\_display\_memory\_stats ( )

Display statistics about the current data handles registered within StarPU. StarPU must have been configured with the configure option `--enable-memory-stats` (see [Memory Feedback](#)).

## 27.19 Theoretical Lower Bound on Execution Time

Compute theoretical upper computation efficiency bound corresponding to some actual execution.

### Functions

- void [starpu\\_bound\\_start](#) (int *deps*, int *prio*)
- void [starpu\\_bound\\_stop](#) (void)
- void [starpu\\_bound\\_print\\_dot](#) (FILE \**output*)
- void [starpu\\_bound\\_compute](#) (double \**res*, double \**integer\_res*, int *integer*)
- void [starpu\\_bound\\_print\\_lp](#) (FILE \**output*)
- void [starpu\\_bound\\_print\\_mps](#) (FILE \**output*)
- void [starpu\\_bound\\_print](#) (FILE \**output*, int *integer*)

#### 27.19.1 Detailed Description

Compute theoretical upper computation efficiency bound corresponding to some actual execution.

#### 27.19.2 Function Documentation

##### 27.19.2.1 void starpu\_bound\_start ( int *deps*, int *prio* )

Start recording tasks (resets stats). *deps* tells whether dependencies should be recorded too (this is quite expensive)

##### 27.19.2.2 void starpu\_bound\_stop ( void )

Stop recording tasks

##### 27.19.2.3 void starpu\_bound\_print\_dot ( FILE \* *output* )

Print the DAG that was recorded

##### 27.19.2.4 void starpu\_bound\_compute ( double \* *res*, double \* *integer\_res*, int *integer* )

Get theoretical upper bound (in ms) (needs glpk support detected by configure script). It returns 0 if some performance models are not calibrated.

##### 27.19.2.5 void starpu\_bound\_print\_lp ( FILE \* *output* )

Emit the Linear Programming system on *output* for the recorded tasks, in the lp format

##### 27.19.2.6 void starpu\_bound\_print\_mps ( FILE \* *output* )

Emit the Linear Programming system on *output* for the recorded tasks, in the mps format

27.19.2.7 `void starpu_bound_print ( FILE * output, int integer )`

Emit statistics of actual execution vs theoretical upper bound. `integer` permits to choose between integer solving (which takes a long time but is correct), and relaxed solving (which provides an approximate solution).

## 27.20 CUDA Extensions

### Macros

- `#define STARPU_USE_CUDA`
- `#define STARPU_MAXCUDADEVs`
- `#define STARPU_CUDA_REPORT_ERROR(status)`
- `#define STARPU_CUBLAS_REPORT_ERROR(status)`

### Functions

- `cudaStream_t starpu_cuda_get_local_stream (void)`
- `const struct cudaDeviceProp * starpu_cuda_get_device_properties (unsigned workerid)`
- `void starpu_cuda_report_error (const char *func, const char *file, int line, cudaError_t status)`
- `int starpu_cuda_copy_async_sync (void *src_ptr, unsigned src_node, void *dst_ptr, unsigned dst_node, size_t ssize, cudaStream_t stream, enum cudaMemcpyKind kind)`
- `void starpu_cuda_set_device (unsigned devid)`
- `void starpu_cublas_init (void)`
- `void starpu_cublas_shutdown (void)`

### 27.20.1 Detailed Description

### 27.20.2 Macro Definition Documentation

#### 27.20.2.1 `#define STARPU_USE_CUDA`

This macro is defined when StarPU has been installed with CUDA support. It should be used in your code to detect the availability of CUDA as shown in [Full source code for the 'Scaling a Vector' example](#).

#### 27.20.2.2 `#define STARPU_MAXCUDADEVs`

This macro defines the maximum number of CUDA devices that are supported by StarPU.

#### 27.20.2.3 `#define STARPU_CUDA_REPORT_ERROR( status )`

Calls `starpu_cuda_report_error()`, passing the current function, file and line position.

#### 27.20.2.4 `#define STARPU_CUBLAS_REPORT_ERROR( status )`

Calls `starpu_cublas_report_error()`, passing the current function, file and line position.

### 27.20.3 Function Documentation

#### 27.20.3.1 `cudaStream_t starpu_cuda_get_local_stream ( void )`

This function gets the current worker's CUDA stream. StarPU provides a stream for every CUDA device controlled by StarPU. This function is only provided for convenience so that programmers can easily use asynchronous operations within codelets without having to create a stream by hand. Note that the application is not forced to use the stream provided by `starpu_cuda_get_local_stream()` and may also create its own streams. Synchronizing with `cudaThreadSynchronize()` is allowed, but will reduce the likelihood of having all transfers overlapped.

#### 27.20.3.2 `const struct cudaDeviceProp * starpu_cuda_get_device_properties ( unsigned workerid )`

This function returns a pointer to device properties for worker `workerid` (assumed to be a CUDA worker).

#### 27.20.3.3 `void starpu_cuda_report_error ( const char * func, const char * file, int line, cudaError_t status )`

Report a CUDA error.

#### 27.20.3.4 `int starpu_cuda_copy_async_sync ( void * src_ptr, unsigned src_node, void * dst_ptr, unsigned dst_node, size_t ssize, cudaStream_t stream, enum cudaMemcpyKind kind )`

Copy `ssize` bytes from the pointer `src_ptr` on `src_node` to the pointer `dst_ptr` on `dst_node`. The function first tries to copy the data asynchronously (unless `stream` is `NULL`). If the asynchronous copy fails or if `stream` is `NULL`, it copies the data synchronously. The function returns `-EAGAIN` if the asynchronous launch was successful. It returns 0 if the synchronous copy was successful, or fails otherwise.

#### 27.20.3.5 `void starpu_cuda_set_device ( unsigned devid )`

Calls `cudaSetDevice(devid)` or `cudaGLSetGLDevice(devid)`, according to whether `devid` is among the field `starpu_conf::cuda_opengl_interoperability`.

#### 27.20.3.6 `void starpu_cublas_init ( void )`

This function initializes CUBLAS on every CUDA device. The CUBLAS library must be initialized prior to any CUBLAS call. Calling `starpu_cublas_init()` will initialize CUBLAS on every CUDA device controlled by StarPU. This call blocks until CUBLAS has been properly initialized on every device.

#### 27.20.3.7 `void starpu_cublas_shutdown ( void )`

This function synchronously deinitializes the CUBLAS library on every CUDA device.

## 27.21 OpenCL Extensions

### Data Structures

- struct `starpu_opengl_program`

### Macros

- `#define STARPU_USE_OPENCL`
- `#define STARPU_MAXOPENCLDEVS`



- `#define STARPU_OPENCL_DATADIR`

## Writing OpenCL kernels

- void `starpu_opengl_get_context` (int devid, cl\_context \*context)
- void `starpu_opengl_get_device` (int devid, cl\_device\_id \*device)
- void `starpu_opengl_get_queue` (int devid, cl\_command\_queue \*queue)
- void `starpu_opengl_get_current_context` (cl\_context \*context)
- void `starpu_opengl_get_current_queue` (cl\_command\_queue \*queue)
- int `starpu_opengl_set_kernel_args` (cl\_int \*err, cl\_kernel \*kernel,...)

## Compiling OpenCL kernels

Source codes for OpenCL kernels can be stored in a file or in a string. StarPU provides functions to build the program executable for each available OpenCL device as a `cl_program` object. This program executable can then be loaded within a specific queue as explained in the next section. These are only helpers, Applications can also fill a `starpu_opengl_program` array by hand for more advanced use (e.g. different programs on the different OpenCL devices, for relocation purpose for instance).

- int `starpu_opengl_load_opengl_from_file` (const char \*source\_file\_name, struct `starpu_opengl_program` \*opengl\_programs, const char \*build\_options)
- int `starpu_opengl_load_opengl_from_string` (const char \*opengl\_program\_source, struct `starpu_opengl_program` \*opengl\_programs, const char \*build\_options)
- int `starpu_opengl_unload_opengl` (struct `starpu_opengl_program` \*opengl\_programs)
- void `starpu_opengl_load_program_source` (const char \*source\_file\_name, char \*located\_file\_name, char \*located\_dir\_name, char \*opengl\_program\_source)
- int `starpu_opengl_compile_opengl_from_file` (const char \*source\_file\_name, const char \*build\_options)
- int `starpu_opengl_compile_opengl_from_string` (const char \*opengl\_program\_source, const char \*file\_name, const char \*build\_options)
- int `starpu_opengl_load_binary_opengl` (const char \*kernel\_id, struct `starpu_opengl_program` \*opengl\_programs)

## Loading OpenCL kernels

- int `starpu_opengl_load_kernel` (cl\_kernel \*kernel, cl\_command\_queue \*queue, struct `starpu_opengl_program` \*opengl\_programs, const char \*kernel\_name, int devid)
- int `starpu_opengl_release_kernel` (cl\_kernel kernel)

## OpenCL statistics

- int `starpu_opengl_collect_stats` (cl\_event event)

## OpenCL utilities

- `#define STARPU_OPENCL_DISPLAY_ERROR(status)`
- `#define STARPU_OPENCL_REPORT_ERROR(status)`
- `#define STARPU_OPENCL_REPORT_ERROR_WITH_MSG(msg, status)`
- const char \* `starpu_opengl_error_string` (cl\_int status)
- void `starpu_opengl_display_error` (const char \*func, const char \*file, int line, const char \*msg, cl\_int status)
- static \_\_starpu\_inline void `starpu_opengl_report_error` (const char \*func, const char \*file, int line, const char \*msg, cl\_int status)
- cl\_int `starpu_opengl_allocate_memory` (int devid, cl\_mem \*addr, size\_t size, cl\_mem\_flags flags)

- `cl_int starpu_opencil_copy_ram_to_opencil` (`void *ptr`, `unsigned src_node`, `cl_mem buffer`, `unsigned dst_node`, `size_t size`, `size_t offset`, `cl_event *event`, `int *ret`)
- `cl_int starpu_opencil_copy_opencil_to_ram` (`cl_mem buffer`, `unsigned src_node`, `void *ptr`, `unsigned dst_node`, `size_t size`, `size_t offset`, `cl_event *event`, `int *ret`)
- `cl_int starpu_opencil_copy_opencil_to_opencil` (`cl_mem src`, `unsigned src_node`, `size_t src_offset`, `cl_mem dst`, `unsigned dst_node`, `size_t dst_offset`, `size_t size`, `cl_event *event`, `int *ret`)
- `cl_int starpu_opencil_copy_async_sync` (`uintptr_t src`, `size_t src_offset`, `unsigned src_node`, `uintptr_t dst`, `size_t dst_offset`, `unsigned dst_node`, `size_t size`, `cl_event *event`)

### 27.21.1 Detailed Description

### 27.21.2 Data Structure Documentation

#### 27.21.2.1 `struct starpu_opencil_program`

Stores the OpenCL programs as compiled for the different OpenCL devices.

##### Data Fields

|                         |                                             |                                             |
|-------------------------|---------------------------------------------|---------------------------------------------|
| <code>cl_program</code> | <code>programs[STARPU_MAXOPENCLDEVS]</code> | Stores each program for each OpenCL device. |
|-------------------------|---------------------------------------------|---------------------------------------------|

### 27.21.3 Macro Definition Documentation

#### 27.21.3.1 `#define STARPU_USE_OPENCL`

This macro is defined when StarPU has been installed with OpenCL support. It should be used in your code to detect the availability of OpenCL as shown in [Full source code for the 'Scaling a Vector' example](#).

#### 27.21.3.2 `#define STARPU_MAXOPENCLDEVS`

This macro defines the maximum number of OpenCL devices that are supported by StarPU.

#### 27.21.3.3 `#define STARPU_OPENCL_DATADIR`

This macro defines the directory in which the OpenCL codelets of the applications provided with StarPU have been installed.

#### 27.21.3.4 `#define STARPU_OPENCL_DISPLAY_ERROR( status )`

Call the function `starpu_opencil_display_error()` with the given error `status`, the current function name, current file and line number, and a empty message.

#### 27.21.3.5 `#define STARPU_OPENCL_REPORT_ERROR( status )`

Call the function `starpu_opencil_report_error()` with the given error `status`, with the current function name, current file and line number, and a empty message.

#### 27.21.3.6 `#define STARPU_OPENCL_REPORT_ERROR_WITH_MSG( msg, status )`

Call the function `starpu_opencil_report_error()` with the given `msg` and the given error `status`, with the current function name, current file and line number.

### 27.21.4 Function Documentation

#### 27.21.4.1 void starpu\_opengl\_get\_context ( int *devid*, cl\_context \* *context* )

Places the OpenCL context of the device designated by *devid* into *context*.

#### 27.21.4.2 void starpu\_opengl\_get\_device ( int *devid*, cl\_device\_id \* *device* )

Places the *cl\_device\_id* corresponding to *devid* in *device*.

#### 27.21.4.3 void starpu\_opengl\_get\_queue ( int *devid*, cl\_command\_queue \* *queue* )

Places the command queue of the device designated by *devid* into *queue*.

#### 27.21.4.4 void starpu\_opengl\_get\_current\_context ( cl\_context \* *context* )

Return the context of the current worker.

#### 27.21.4.5 void starpu\_opengl\_get\_current\_queue ( cl\_command\_queue \* *queue* )

Return the computation kernel command queue of the current worker.

#### 27.21.4.6 int starpu\_opengl\_set\_kernel\_args ( cl\_int \* *err*, cl\_kernel \* *kernel*, ... )

Sets the arguments of a given kernel. The list of arguments must be given as (*size\_t size\_of\_the\_argument*, *cl\_mem \* pointer\_to\_the\_argument*). The last argument must be 0. Returns the number of arguments that were successfully set. In case of failure, returns the id of the argument that could not be set and *err* is set to the error returned by OpenCL. Otherwise, returns the number of arguments that were set.

Here an example:

```
int n;
cl_int err;
cl_kernel kernel;
n = starpu_opengl_set_kernel_args(&err, 2, &kernel,
 sizeof(foo), &foo,
 sizeof(bar), &bar,
 0);
if (n != 2)
 fprintf(stderr, "Error : %d\n", err);
```

#### 27.21.4.7 int starpu\_opengl\_load\_opengl\_from\_file ( const char \* *source\_file\_name*, struct starpu\_opengl\_program \* *opengl\_programs*, const char \* *build\_options* )

This function compiles an OpenCL source code stored in a file.

#### 27.21.4.8 int starpu\_opengl\_load\_opengl\_from\_string ( const char \* *opengl\_program\_source*, struct starpu\_opengl\_program \* *opengl\_programs*, const char \* *build\_options* )

This function compiles an OpenCL source code stored in a string.

#### 27.21.4.9 int starpu\_opengl\_unload\_opengl ( struct starpu\_opengl\_program \* *opengl\_programs* )

This function unloads an OpenCL compiled code.

**27.21.4.10** `void starpu_opengl_load_program_source ( const char * source_file_name, char * located_file_name, char * located_dir_name, char * opengl_program_source )`

Store the contents of the file `source_file_name` in the buffer `opengl_program_source`. The file `source_file_name` can be located in the current directory, or in the directory specified by the environment variable `STARPU_OPENGL_PROGRAM_DIR`, or in the directory `share/starpu/opengl` of the installation directory of StarPU, or in the source directory of StarPU. When the file is found, `located_file_name` is the full name of the file as it has been located on the system, `located_dir_name` the directory where it has been located. Otherwise, they are both set to the empty string.

**27.21.4.11** `int starpu_opengl_compile_opengl_from_file ( const char * source_file_name, const char * build_options )`

Compile the OpenCL kernel stored in the file `source_file_name` with the given options `build_options` and stores the result in the directory `$STARPU_HOME/.starpu/opengl` with the same filename as `source_file_name`. The compilation is done for every OpenCL device, and the filename is suffixed with the vendor id and the device id of the OpenCL device.

**27.21.4.12** `int starpu_opengl_compile_opengl_from_string ( const char * opengl_program_source, const char * file_name, const char * build_options )`

Compile the OpenCL kernel in the string `opengl_program_source` with the given options `build_options` and stores the result in the directory `$STARPU_HOME/.starpu/opengl` with the filename `file_name`. The compilation is done for every OpenCL device, and the filename is suffixed with the vendor id and the device id of the OpenCL device.

**27.21.4.13** `int starpu_opengl_load_binary_opengl ( const char * kernel_id, struct starpu_opengl_program * opengl_programs )`

Compile the binary OpenCL kernel identified with `kernel_id`. For every OpenCL device, the binary OpenCL kernel will be loaded from the file `$STARPU_HOME/.starpu/opengl/<kernel_id>.<device_type>.<vendor_id>.<device_id>`.

**27.21.4.14** `int starpu_opengl_load_kernel ( cl_kernel * kernel, cl_command_queue * queue, struct starpu_opengl_program * opengl_programs, const char * kernel_name, int devid )`

Create a kernel `kernel` for device `devid`, on its computation command queue returned in `queue`, using program `opengl_programs` and name `kernel_name`.

**27.21.4.15** `int starpu_opengl_release_kernel ( cl_kernel kernel )`

Release the given `kernel`, to be called after kernel execution.

**27.21.4.16** `int starpu_opengl_collect_stats ( cl_event event )`

This function allows to collect statistics on a kernel execution. After termination of the kernels, the OpenCL codelet should call this function to pass it the even returned by `clEnqueueNDRangeKernel`, to let StarPU collect statistics about the kernel execution (used cycles, consumed power).

**27.21.4.17** `const char * starpu_opengl_error_string ( cl_int status )`

Return the error message in English corresponding to `status`, an OpenCL error code.

**27.21.4.18** `void starpu_opengl_display_error ( const char * func, const char * file, int line, const char * msg, cl_int status )`

Given a valid error status, prints the corresponding error message on stdout, along with the given function name *func*, the given filename *file*, the given line number *line* and the given message *msg*.

**27.21.4.19** `void starpu_opengl_report_error ( const char * func, const char * file, int line, const char * msg, cl_int status )  
[static]`

Call the function `starpu_opengl_display_error()` and abort.

**27.21.4.20** `cl_int starpu_opengl_allocate_memory ( int devid, cl_mem * addr, size_t size, cl_mem_flags flags )`

Allocate *size* bytes of memory, stored in *addr*. *flags* must be a valid combination of `cl_mem_flags` values.

**27.21.4.21** `cl_int starpu_opengl_copy_ram_to_opengl ( void * ptr, unsigned src_node, cl_mem buffer, unsigned dst_node,  
size_t size, size_t offset, cl_event * event, int * ret )`

Copy *size* bytes from the given *ptr* on RAM *src\_node* to the given *buffer* on OpenCL *dst\_node*. *offset* is the offset, in bytes, in *buffer*. If *event* is NULL, the copy is synchronous, i.e. the queue is synchronised before returning. If not NULL, *event* can be used after the call to wait for this particular copy to complete. This function returns `CL_SUCCESS` if the copy was successful, or a valid OpenCL error code otherwise. The integer pointed to by *ret* is set to `-EAGAIN` if the asynchronous launch was successful, or to 0 if *event* was NULL.

**27.21.4.22** `cl_int starpu_opengl_copy_opengl_to_ram ( cl_mem buffer, unsigned src_node, void * ptr, unsigned dst_node,  
size_t size, size_t offset, cl_event * event, int * ret )`

Copy *size* bytes asynchronously from the given *buffer* on OpenCL *src\_node* to the given *ptr* on RAM *dst\_node*. *offset* is the offset, in bytes, in *buffer*. If *event* is NULL, the copy is synchronous, i.e. the queue is synchronised before returning. If not NULL, *event* can be used after the call to wait for this particular copy to complete. This function returns `CL_SUCCESS` if the copy was successful, or a valid OpenCL error code otherwise. The integer pointed to by *ret* is set to `-EAGAIN` if the asynchronous launch was successful, or to 0 if *event* was NULL.

**27.21.4.23** `cl_int starpu_opengl_copy_opengl_to_opengl ( cl_mem src, unsigned src_node, size_t src_offset, cl_mem dst,  
unsigned dst_node, size_t dst_offset, size_t size, cl_event * event, int * ret )`

Copy *size* bytes asynchronously from byte offset *src\_offset* of *src* on OpenCL *src\_node* to byte offset *dst\_offset* of *dst* on OpenCL *dst\_node*. If *event* is NULL, the copy is synchronous, i.e. the queue is synchronised before returning. If not NULL, *event* can be used after the call to wait for this particular copy to complete. This function returns `CL_SUCCESS` if the copy was successful, or a valid OpenCL error code otherwise. The integer pointed to by *ret* is set to `-EAGAIN` if the asynchronous launch was successful, or to 0 if *event* was NULL.

**27.21.4.24** `cl_int starpu_opengl_copy_async_sync ( uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t  
dst_offset, unsigned dst_node, size_t size, cl_event * event )`

Copy *size* bytes from byte offset *src\_offset* of *src* on *src\_node* to byte offset *dst\_offset* of *dst* on *dst\_node*. If *event* is NULL, the copy is synchronous, i.e. the queue is synchronised before returning. If not NULL, *event* can be used after the call to wait for this particular copy to complete. The function returns `-EAGAIN` if the asynchronous launch was successful. It returns 0 if the synchronous copy was successful, or fails otherwise.

## 27.22 OpenMP Runtime Support

This section describes the interface provided for implementing OpenMP runtimes on top of StarPU.

### Data Structures

- struct [starpu\\_omp\\_lock\\_t](#)
- struct [starpu\\_omp\\_nest\\_lock\\_t](#)
- struct [starpu\\_omp\\_parallel\\_region\\_attr](#)
- struct [starpu\\_omp\\_task\\_region\\_attr](#)

### Enumerations

- enum [starpu\\_omp\\_sched\\_value](#) {  
[starpu\\_omp\\_sched\\_undefined](#), [starpu\\_omp\\_sched\\_static](#), [starpu\\_omp\\_sched\\_dynamic](#), [starpu\\_omp\\_sched\\_guided](#),  
[starpu\\_omp\\_sched\\_auto](#), [starpu\\_omp\\_sched\\_runtime](#) }
- enum [starpu\\_omp\\_proc\\_bind\\_value](#) {  
[starpu\\_omp\\_proc\\_bind\\_undefined](#), [starpu\\_omp\\_proc\\_bind\\_false](#), [starpu\\_omp\\_proc\\_bind\\_true](#), [starpu\\_omp\\_proc\\_bind\\_master](#),  
[starpu\\_omp\\_proc\\_bind\\_close](#), [starpu\\_omp\\_proc\\_bind\\_spread](#) }

### Initialisation

- #define [STARPU\\_OPENMP](#)
- int [starpu\\_omp\\_init](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_shutdown](#) (void) \_\_STARPU\_OMP\_NOTHROW

### Parallel

- void [starpu\\_omp\\_parallel\\_region](#) (const struct [starpu\\_omp\\_parallel\\_region\\_attr](#) \*attr) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_master](#) (void(\*f)(void \*arg), void \*arg) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_master\\_inline](#) (void) \_\_STARPU\_OMP\_NOTHROW

### Synchronization

- void [starpu\\_omp\\_barrier](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_critical](#) (void(\*f)(void \*arg), void \*arg, const char \*name) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_critical\\_inline\\_begin](#) (const char \*name) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_critical\\_inline\\_end](#) (const char \*name) \_\_STARPU\_OMP\_NOTHROW

### Worksharing

- void [starpu\\_omp\\_single](#) (void(\*f)(void \*arg), void \*arg, int nowait) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_single\\_inline](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_single\\_copyprivate](#) (void(\*f)(void \*arg, void \*data, unsigned long long data\_size), void \*arg, void \*data, unsigned long long data\_size) \_\_STARPU\_OMP\_NOTHROW
- void \* [starpu\\_omp\\_single\\_copyprivate\\_inline\\_begin](#) (void \*data) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_single\\_copyprivate\\_inline\\_end](#) (void) \_\_STARPU\_OMP\_NOTHROW

- void [starpu\\_omp\\_for](#) (void(\*f)(unsigned long long \_first\_i, unsigned long long \_nb\_i, void \*arg), void \*arg, unsigned long long nb\_iterations, unsigned long long chunk, int schedule, int ordered, int nowait) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_for\\_inline\\_first](#) (unsigned long long nb\_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long \*\_first\_i, unsigned long long \*\_nb\_i) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_for\\_inline\\_next](#) (unsigned long long nb\_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long \*\_first\_i, unsigned long long \*\_nb\_i) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_for\\_alt](#) (void(\*f)(unsigned long long \_begin\_i, unsigned long long \_end\_i, void \*arg), void \*arg, unsigned long long nb\_iterations, unsigned long long chunk, int schedule, int ordered, int nowait) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_for\\_inline\\_first\\_alt](#) (unsigned long long nb\_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long \*\_begin\_i, unsigned long long \*\_end\_i) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_for\\_inline\\_next\\_alt](#) (unsigned long long nb\_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long \*\_begin\_i, unsigned long long \*\_end\_i) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_ordered](#) (void(\*f)(void \*arg), void \*arg) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_ordered\\_inline\\_begin](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_ordered\\_inline\\_end](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_sections](#) (unsigned long long nb\_sections, void(\*\*section\_f)(void \*arg), void \*\*section\_arg, int nowait) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_sections\\_combined](#) (unsigned long long nb\_sections, void(\*section\_f)(unsigned long long section\_num, void \*arg), void \*section\_arg, int nowait) \_\_STARPU\_OMP\_NOTHROW

## Task

- void [starpu\\_omp\\_task\\_region](#) (const struct [starpu\\_omp\\_task\\_region\\_attr](#) \*attr) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_taskwait](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_taskgroup](#) (void(\*f)(void \*arg), void \*arg) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_taskgroup\\_inline\\_begin](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_taskgroup\\_inline\\_end](#) (void) \_\_STARPU\_OMP\_NOTHROW

## API

- void [starpu\\_omp\\_set\\_num\\_threads](#) (int threads) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_num\\_threads](#) () \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_thread\\_num](#) () \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_max\\_threads](#) () \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_num\\_procs](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_in\\_parallel](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_dynamic](#) (int dynamic\_threads) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_dynamic](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_nested](#) (int nested) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_nested](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_cancellation](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_schedule](#) (enum [starpu\\_omp\\_sched\\_value](#) kind, int modifier) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_get\\_schedule](#) (enum [starpu\\_omp\\_sched\\_value](#) \*kind, int \*modifier) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_thread\\_limit](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_max\\_active\\_levels](#) (int max\_levels) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_max\\_active\\_levels](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_level](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_ancestor\\_thread\\_num](#) (int level) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_team\\_size](#) (int level) \_\_STARPU\_OMP\_NOTHROW

- int [starpu\\_omp\\_get\\_active\\_level](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_in\\_final](#) (void) \_\_STARPU\_OMP\_NOTHROW
- enum [starpu\\_omp\\_proc\\_bind\\_value](#) [starpu\\_omp\\_get\\_proc\\_bind](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_default\\_device](#) (int device\_num) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_default\\_device](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_num\\_devices](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_num\\_teams](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_team\\_num](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_is\\_initial\\_device](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_init\\_lock](#) ([starpu\\_omp\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_destroy\\_lock](#) ([starpu\\_omp\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_lock](#) ([starpu\\_omp\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_unset\\_lock](#) ([starpu\\_omp\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_test\\_lock](#) ([starpu\\_omp\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_init\\_nest\\_lock](#) ([starpu\\_omp\\_nest\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_destroy\\_nest\\_lock](#) ([starpu\\_omp\\_nest\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_nest\\_lock](#) ([starpu\\_omp\\_nest\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_unset\\_nest\\_lock](#) ([starpu\\_omp\\_nest\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_test\\_nest\\_lock](#) ([starpu\\_omp\\_nest\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_atomic\\_fallback\\_inline\\_begin](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_atomic\\_fallback\\_inline\\_end](#) (void) \_\_STARPU\_OMP\_NOTHROW
- double [starpu\\_omp\\_get\\_wtime](#) (void) \_\_STARPU\_OMP\_NOTHROW
- double [starpu\\_omp\\_get\\_wtick](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_vector\\_annotate](#) ([starpu\\_data\\_handle\\_t](#) handle, uint32\_t slice\_base) \_\_STARPU\_OMP\_NOTHROW

### 27.22.1 Detailed Description

This section describes the interface provided for implementing OpenMP runtimes on top of StarPU.

### 27.22.2 Data Structure Documentation

#### 27.22.2.1 struct [starpu\\_omp\\_lock\\_t](#)

Opaque Simple Lock object ([Simple Locks](#)) for inter-task synchronization operations.

See also

[starpu\\_omp\\_init\\_lock\(\)](#)  
[starpu\\_omp\\_destroy\\_lock\(\)](#)  
[starpu\\_omp\\_set\\_lock\(\)](#)  
[starpu\\_omp\\_unset\\_lock\(\)](#)  
[starpu\\_omp\\_test\\_lock\(\)](#)

#### Data Fields

|        |          |                                        |
|--------|----------|----------------------------------------|
| void * | internal | Is an opaque pointer for internal use. |
|--------|----------|----------------------------------------|

#### 27.22.2.2 struct [starpu\\_omp\\_nest\\_lock\\_t](#)

Opaque Nestable Lock object ([Nestable Locks](#)) for inter-task synchronization operations.



See also

[starpu\\_omp\\_init\\_nest\\_lock\(\)](#)  
[starpu\\_omp\\_destroy\\_nest\\_lock\(\)](#)  
[starpu\\_omp\\_set\\_nest\\_lock\(\)](#)  
[starpu\\_omp\\_unset\\_nest\\_lock\(\)](#)  
[starpu\\_omp\\_test\\_nest\\_lock\(\)](#)

Data Fields

|        |          |                                        |
|--------|----------|----------------------------------------|
| void * | internal | Is an opaque pointer for internal use. |
|--------|----------|----------------------------------------|

### 27.22.2.3 struct starpu\_omp\_parallel\_region\_attr

Set of attributes used for creating a new parallel region.

See also

[starpu\\_omp\\_parallel\\_region\(\)](#)

Data Fields

|                                          |             |                                                                                                                                                                                                                                                                                                                                 |
|------------------------------------------|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| struct<br><a href="#">starpu_codelet</a> | cl          | Is a <a href="#">starpu_codelet</a> (Codelet And Tasks) to use for the parallel region implicit tasks. The codelet must provide a CPU implementation function.                                                                                                                                                                  |
| <a href="#">starpu_data_handle_t</a> *   | handles     | Is an array of zero or more <a href="#">starpu_data_handle_t</a> data handle to be passed to the parallel region implicit tasks.                                                                                                                                                                                                |
| void *                                   | cl_arg      | Is an optional pointer to an inline argument to be passed to the region implicit tasks.                                                                                                                                                                                                                                         |
| size_t                                   | cl_arg_size | Is the size of the optional inline argument to be passed to the region implicit tasks, or 0 if unused.                                                                                                                                                                                                                          |
| unsigned                                 | cl_arg_free | Is a boolean indicating whether the optional inline argument should be automatically freed (true), or not (false).                                                                                                                                                                                                              |
| int                                      | if_clause   | Is a boolean indicating whether the <code>if</code> clause of the corresponding <code>pragma omp parallel</code> is true or false.                                                                                                                                                                                              |
| int                                      | num_threads | Is an integer indicating the requested number of threads in the team of the newly created parallel region, or 0 to let the runtime choose the number of threads alone. This attribute may be ignored by the runtime system if the requested number of threads is higher than the number of threads that the runtime can create. |

### 27.22.2.4 struct starpu\_omp\_task\_region\_attr

Set of attributes used for creating a new task region.

See also

[starpu\\_omp\\_task\\_region\(\)](#)

Data Fields

|                                          |    |                                                                                                                                                                                                                         |
|------------------------------------------|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| struct<br><a href="#">starpu_codelet</a> | cl | Is a <a href="#">starpu_codelet</a> (Codelet And Tasks) to use for the task region explicit task. The codelet must provide a CPU implementation function or an accelerator implementation for offloaded target regions. |
|------------------------------------------|----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                        |                  |                                                                                                                                 |
|----------------------------------------|------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">starpu_data_handle_t</a> * | handles          | Is an array of zero or more <a href="#">starpu_data_handle_t</a> data handle to be passed to the task region explicit tasks.    |
| void *                                 | cl_arg           | Is an optional pointer to an inline argument to be passed to the region implicit tasks.                                         |
| size_t                                 | cl_arg_size      | Is the size of the optional inline argument to be passed to the region implicit tasks, or 0 if unused.                          |
| unsigned                               | cl_arg_free      | Is a boolean indicating whether the optional inline argument should be automatically freed (true), or not (false).              |
| int                                    | if_clause        | Is a boolean indicating whether the <b>if</b> clause of the corresponding <code>pragma omp task</code> is true or false.        |
| int                                    | final_clause     | Is a boolean indicating whether the <b>final</b> clause of the corresponding <code>pragma omp task</code> is true or false.     |
| int                                    | untied_clause    | Is a boolean indicating whether the <b>untied</b> clause of the corresponding <code>pragma omp task</code> is true or false.    |
| int                                    | mergeable_clause | Is a boolean indicating whether the <b>mergeable</b> clause of the corresponding <code>pragma omp task</code> is true or false. |

### 27.22.3 Macro Definition Documentation

#### 27.22.3.1 #define STARPU\_OPENMP

This macro is defined when StarPU has been installed with OpenMP Runtime support. It should be used in your code to detect the availability of the runtime support for OpenMP.

### 27.22.4 Enumeration Type Documentation

#### 27.22.4.1 enum starpu\_omp\_sched\_value

Set of constants for selecting the for loop iteration scheduling algorithm ([Parallel For](#)) as defined by the OpenMP specification.

Enumerator

***starpu\_omp\_sched\_undefined*** Undefined iteration scheduling algorithm.

***starpu\_omp\_sched\_static*** Static iteration scheduling algorithm.

***starpu\_omp\_sched\_dynamic*** Dynamic iteration scheduling algorithm.

***starpu\_omp\_sched\_guided*** Guided iteration scheduling algorithm.

***starpu\_omp\_sched\_auto*** Automatically chosen iteration scheduling algorithm.

***starpu\_omp\_sched\_runtime*** Choice of iteration scheduling algorithm deferred at **runtime**.

See also

```

starpu_omp_for\(\)
starpu_omp_for_inline_first\(\)
starpu_omp_for_inline_next\(\)
starpu_omp_for_alt\(\)
starpu_omp_for_inline_first_alt\(\)
starpu_omp_for_inline_next_alt\(\)

```

#### 27.22.4.2 enum starpu\_omp\_proc\_bind\_value

Set of constants for selecting the processor binding method, as defined in the OpenMP specification.

## Enumerator

**starpu\_omp\_proc\_bind\_undefined** Undefined processor binding method.

**starpu\_omp\_proc\_bind\_false** Team threads may be moved between places at any time.

**starpu\_omp\_proc\_bind\_true** Team threads may not be moved between places.

**starpu\_omp\_proc\_bind\_master** Assign every thread in the team to the same place as the **master** thread.

**starpu\_omp\_proc\_bind\_close** Assign every thread in the team to a place **close** to the parent thread.

**starpu\_omp\_proc\_bind\_spread** Assign team threads as a sparse distribution over the selected places.

See also

[starpu\\_omp\\_get\\_proc\\_bind\(\)](#)

## 27.22.5 Function Documentation

27.22.5.1 `int starpu_omp_init ( void )`

Initializes StarPU and its OpenMP Runtime support.

27.22.5.2 `void starpu_omp_shutdown ( void )`

Shutdown StarPU and its OpenMP Runtime support.

27.22.5.3 `void starpu_omp_parallel_region ( const struct starpu_omp_parallel_region_attr * attr )`

Generates and launch an OpenMP parallel region and return after its completion. `attr` specifies the attributes for the generated parallel region. If this function is called from inside another, generating, parallel region, the generated parallel region is nested within the generating parallel region.

This function can be used to implement `#pragma omp parallel`.

27.22.5.4 `void starpu_omp_master ( void(*)(void *arg) f, void * arg )`

Executes a function only on the master thread of the OpenMP parallel region it is called from. When called from a thread that is not the master of the parallel region it is called from, this function does nothing. `f` is the function to be called. `arg` is an argument passed to function `f`.

This function can be used to implement `#pragma omp master`.

27.22.5.5 `int starpu_omp_master_inline ( void )`

Determines whether the calling thread is the master of the OpenMP parallel region it is called from or not.

This function can be used to implement `#pragma omp master` without code outlining.

## Returns

! 0 if called by the region's master thread.

0 if not called by the region's master thread.

27.22.5.6 `void starpu_omp_barrier ( void )`

Waits until each participating thread of the innermost OpenMP parallel region has reached the barrier and each explicit OpenMP task bound to this region has completed its execution.

This function can be used to implement `#pragma omp barrier`.

#### 27.22.5.7 void starpu\_omp\_critical ( void(\*)(void \*arg) f, void \* arg, const char \* name )

Waits until no other thread is executing within the context of the selected critical section, then proceeds to the exclusive execution of a function within the critical section. `f` is the function to be executed in the critical section. `arg` is an argument passed to function `f`. `name` is the name of the selected critical section. If `name == NULL`, the selected critical section is the unique anonymous critical section.

This function can be used to implement `#pragma omp critical`.

#### 27.22.5.8 void starpu\_omp\_critical\_inline\_begin ( const char \* name )

Waits until execution can proceed exclusively within the context of the selected critical section. `name` is the name of the selected critical section. If `name == NULL`, the selected critical section is the unique anonymous critical section.

This function together with [starpu\\_omp\\_critical\\_inline\\_end](#) can be used to implement `#pragma omp critical` without code outlining.

#### 27.22.5.9 void starpu\_omp\_critical\_inline\_end ( const char \* name )

Ends the exclusive execution within the context of the selected critical section. `name` is the name of the selected critical section. If `name==NULL`, the selected critical section is the unique anonymous critical section.

This function together with [starpu\\_omp\\_critical\\_inline\\_begin](#) can be used to implement `#pragma omp critical` without code outlining.

#### 27.22.5.10 void starpu\_omp\_single ( void(\*)(void \*arg) f, void \* arg, int nowait )

Ensures that a single participating thread of the innermost OpenMP parallel region executes a function. `f` is the function to be executed by a single thread. `arg` is an argument passed to function `f`. `nowait` is a flag indicating whether an implicit barrier is requested after the single section (`nowait==0`) or not (`nowait==!0`).

This function can be used to implement `#pragma omp single`.

#### 27.22.5.11 int starpu\_omp\_single\_inline ( void )

Decides whether the current thread is elected to run the following single section among the participating threads of the innermost OpenMP parallel region.

This function can be used to implement `#pragma omp single` without code outlining.

#### Returns

- !0 if the calling thread has won the election.
- 0 if the calling thread has lost the election.

#### 27.22.5.12 void starpu\_omp\_single\_copyprivate ( void(\*)(void \*arg, void \*data, unsigned long long data\_size) f, void \* arg, void \* data, unsigned long long data\_size )

This function executes `f` on a single task of the current parallel region task, and then broadcast the contents of the memory block pointed by the copyprivate pointer `data` and of size `data_size` to the corresponding `data` pointed memory blocks of all the other participating region tasks. This function can be used to implement `#pragma omp single` with a copyprivate clause.

See also

[starpu\\_omp\\_single\\_copyprivate\\_inline](#)  
[starpu\\_omp\\_single\\_copyprivate\\_inline\\_begin](#)  
[starpu\\_omp\\_single\\_copyprivate\\_inline\\_end](#)

**27.22.5.13** `void * starpu_omp_single_copyprivate_inline_begin ( void * data )`

This function elects one task among the tasks of the current parallel region task to execute the following single section, and then broadcast the copyprivate pointer `data` to all the other participating region tasks. This function can be used to implement `#pragma omp single` with a copyprivate clause without code outlining.

See also

[starpu\\_omp\\_single\\_copyprivate\\_inline](#)  
[starpu\\_omp\\_single\\_copyprivate\\_inline\\_end](#)

**27.22.5.14** `void starpu_omp_single_copyprivate_inline_end ( void )`

This function completes the execution of a single section and returns the broadcasted copyprivate pointer for tasks that lost the election and NULL for the task that won the election. This function can be used to implement `#pragma omp single` with a copyprivate clause without code outlining.

Returns

the copyprivate pointer for tasks that lost the election and therefore did not execute the code of the single section.

NULL for the task that won the election and executed the code of the single section.

See also

[starpu\\_omp\\_single\\_copyprivate\\_inline](#)  
[starpu\\_omp\\_single\\_copyprivate\\_inline\\_begin](#)

**27.22.5.15** `void starpu_omp_for ( void(*) (unsigned long long _first_i, unsigned long long _nb_i, void *arg) f, void * arg, unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, int nowait )`

Executes a parallel loop together with the other threads participating to the innermost parallel region. `f` is the function to be executed iteratively. `arg` is an argument passed to function `f`. `nb_iterations` is the number of iterations to be performed by the parallel loop. `chunk` is the number of consecutive iterations that should be affected to the same thread when scheduling the loop workshares, it follows the semantics of the `modifier` argument in OpenMP `#pragma omp for` specification. `schedule` is the scheduling mode according to the OpenMP specification. `ordered` is a flag indicating whether the loop region may contain an ordered section (`ordered==!0`) or not (`ordered==0`). `nowait` is a flag indicating whether an implicit barrier is requested after the for section (`nowait==0`) or not (`nowait==!0`).

The function `f` will be called with arguments `_first_i`, the first iteration to perform, `_nb_i`, the number of consecutive iterations to perform before returning, `arg`, the free `arg` argument.

This function can be used to implement `#pragma omp for`.

**27.22.5.16** `int starpu_omp_for_inline_first ( unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long * _first_i, unsigned long long * _nb_i )`

Decides whether the current thread should start to execute a parallel loop section. See [starpu\\_omp\\_for](#) for the argument description.

This function together with [starpu\\_omp\\_for\\_inline\\_next](#) can be used to implement `#pragma omp for` without code outlining.

#### Returns

!0 if the calling thread participates to the loop region and should execute a first chunk of iterations. In that case, `*_first_i` will be set to the first iteration of the chunk to perform and `*_nb_i` will be set to the number of iterations of the chunk to perform.

0 if the calling thread does not participate to the loop region because all the available iterations have been affected to the other threads of the parallel region.

#### See also

[starpu\\_omp\\_for](#)

**27.22.5.17** `int starpu_omp_for_inline_next ( unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long * _first_i, unsigned long long * _nb_i )`

Decides whether the current thread should continue to execute a parallel loop section. See [starpu\\_omp\\_for](#) for the argument description.

This function together with [starpu\\_omp\\_for\\_inline\\_first](#) can be used to implement `#pragma omp for` without code outlining.

#### Returns

!0 if the calling thread should execute a next chunk of iterations. In that case, `*_first_i` will be set to the first iteration of the chunk to perform and `*_nb_i` will be set to the number of iterations of the chunk to perform.

0 if the calling thread does not participate anymore to the loop region because all the available iterations have been affected to the other threads of the parallel region.

#### See also

[starpu\\_omp\\_for](#)

**27.22.5.18** `void starpu_omp_for_alt ( void(*) (unsigned long long _begin_i, unsigned long long _end_i, void *arg) f, void * arg, unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, int nowait )`

Alternative implementation of a parallel loop. This function differs from [starpu\\_omp\\_for](#) in the expected arguments of the loop function `f`.

The function `f` will be called with arguments `_begin_i`, the first iteration to perform, `_end_i`, the first iteration not to perform before returning, `arg`, the free `arg` argument.

This function can be used to implement `#pragma omp for`.

#### See also

[starpu\\_omp\\_for](#)

**27.22.5.19** `int starpu_omp_for_inline_first_alt ( unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long * _begin_i, unsigned long long * _end_i )`

Inline version of the alternative implementation of a parallel loop.

This function together with [starpu\\_omp\\_for\\_inline\\_next\\_alt](#) can be used to implement `#pragma omp for` without code outlining.

See also

[starpu\\_omp\\_for](#)  
[starpu\\_omp\\_for\\_alt](#)  
[starpu\\_omp\\_for\\_inline\\_first](#)

**27.22.5.20** `int starpu_omp_for_inline_next_alt ( unsigned long long nb_iterations, unsigned long long chunk, int schedule, int ordered, unsigned long long * _begin_i, unsigned long long * _end_i )`

Inline version of the alternative implementation of a parallel loop.

This function together with [starpu\\_omp\\_for\\_inline\\_first\\_alt](#) can be used to implement `#pragma omp for` without code outlining.

See also

[starpu\\_omp\\_for](#)  
[starpu\\_omp\\_for\\_alt](#)  
[starpu\\_omp\\_for\\_inline\\_next](#)

**27.22.5.21** `void starpu_omp_ordered ( void(*) (void *arg) f, void * arg )`

Ensures that a function is sequentially executed once for each iteration in order within a parallel loop, by the thread that own the iteration. *f* is the function to be executed by the thread that own the current iteration. *arg* is an argument passed to function *f*.

This function can be used to implement `#pragma omp ordered`.

**27.22.5.22** `void starpu_omp_ordered_inline_begin ( void )`

Waits until all the iterations of a parallel loop below the iteration owned by the current thread have been executed.

This function together with [starpu\\_omp\\_ordered\\_inline\\_end](#) can be used to implement `#pragma omp ordered` without code code outlining.

**27.22.5.23** `void starpu_omp_ordered_inline_end ( void )`

Notifies that the ordered section for the current iteration has been completed.

This function together with [starpu\\_omp\\_ordered\\_inline\\_begin](#) can be used to implement `#pragma omp ordered` without code code outlining.

**27.22.5.24** `void starpu_omp_sections ( unsigned long long nb_sections, void(**) (void *arg) section_f, void ** section_arg, int nowait )`

Ensures that each function of a given array of functions is executed by one and only one thread. *nb\_sections* is the number of functions in the array *section\_f*. *section\_f* is the array of functions to be executed as sections. *section\_arg* is an array of arguments to be passed to the corresponding function. *nowait* is a flag indicating whether an implicit barrier is requested after the execution of all the sections (*nowait*==0) or not (*nowait*==!0).

This function can be used to implement `#pragma omp sections` and `#pragma omp section`.

**27.22.5.25** `void starpu_omp_sections_combined ( unsigned long long nb_sections, void(*)(unsigned long long section_num, void *arg) section_f, void * section_arg, int nowait )`

Alternative implementation of sections. This function differs from [starpu\\_omp\\_sections](#) in that all the sections are combined within a single function in this version. *section\_f* is the function implementing the combined sections.

The function *section\_f* will be called with arguments *section\_num*, the section number to be executed, *arg*, the entry of *section\_arg* corresponding to this section.

This function can be used to implement `#pragma omp sections` and `#pragma omp section`.

See also

[starpu\\_omp\\_sections](#)

**27.22.5.26** `void starpu_omp_task_region ( const struct starpu_omp_task_region_attr * attr )`

Generates an explicit child task. The execution of the generated task is asynchronous with respect to the calling code unless specified otherwise. *attr* specifies the attributes for the generated task region.

This function can be used to implement `#pragma omp task`.

**27.22.5.27** `void starpu_omp_taskwait ( void )`

Waits for the completion of the tasks generated by the current task. This function does not wait for the descendants of the tasks generated by the current task.

This function can be used to implement `#pragma omp taskwait`.

**27.22.5.28** `void starpu_omp_taskgroup ( void(*)(void *arg) f, void * arg )`

Launches a function and wait for the completion of every descendant task generated during the execution of the function.

This function can be used to implement `#pragma omp taskgroup`.

See also

[starpu\\_omp\\_taskgroup\\_inline\\_begin](#)  
[starpu\\_omp\\_taskgroup\\_inline\\_end](#)

**27.22.5.29** `void starpu_omp_taskgroup_inline_begin ( void )`

Launches a function and gets ready to wait for the completion of every descendant task generated during the dynamic scope of the taskgroup.

This function can be used to implement `#pragma omp taskgroup` without code outlining.

See also

[starpu\\_omp\\_taskgroup](#)  
[starpu\\_omp\\_taskgroup\\_inline\\_end](#)

**27.22.5.30** `void starpu_omp_taskgroup_inline_end ( void )`

Waits for the completion of every descendant task generated during the dynamic scope of the taskgroup.

This function can be used to implement `#pragma omp taskgroup` without code outlining.



See also

[starpu\\_omp\\_taskgroup](#)  
[starpu\\_omp\\_taskgroup\\_inline\\_begin](#)

#### 27.22.5.31 void starpu\_omp\_set\_num\_threads ( int *threads* )

This function sets ICVS `nthreads_var` for the parallel regions to be created with the current region.

Note: The StarPU OpenMP runtime support currently ignores this setting for nested parallel regions.

See also

[starpu\\_omp\\_get\\_num\\_threads](#)  
[starpu\\_omp\\_get\\_thread\\_num](#)  
[starpu\\_omp\\_get\\_max\\_threads](#)  
[starpu\\_omp\\_get\\_num\\_procs](#)

#### 27.22.5.32 int starpu\_omp\_get\_num\_threads ( )

This function returns the number of threads of the current region.

Returns

the number of threads of the current region.

See also

[starpu\\_omp\\_set\\_num\\_threads](#)  
[starpu\\_omp\\_get\\_thread\\_num](#)  
[starpu\\_omp\\_get\\_max\\_threads](#)  
[starpu\\_omp\\_get\\_num\\_procs](#)

#### 27.22.5.33 int starpu\_omp\_get\_thread\_num ( )

This function returns the rank of the current thread among the threads of the current region.

Returns

the rank of the current thread in the current region.

See also

[starpu\\_omp\\_set\\_num\\_threads](#)  
[starpu\\_omp\\_get\\_num\\_threads](#)  
[starpu\\_omp\\_get\\_max\\_threads](#)  
[starpu\\_omp\\_get\\_num\\_procs](#)

#### 27.22.5.34 int starpu\_omp\_get\_max\_threads ( )

This function returns the maximum number of threads that can be used to create a region from the current region.

Returns

the maximum number of threads that can be used to create a region from the current region.

**See also**

[starpu\\_omp\\_set\\_num\\_threads](#)  
[starpu\\_omp\\_get\\_num\\_threads](#)  
[starpu\\_omp\\_get\\_thread\\_num](#)  
[starpu\\_omp\\_get\\_num\\_procs](#)

**27.22.5.35** `int starpu_omp_get_num_procs ( void )`

This function returns the number of StarPU CPU workers.

**Returns**

the number of StarPU CPU workers.

**See also**

[starpu\\_omp\\_set\\_num\\_threads](#)  
[starpu\\_omp\\_get\\_num\\_threads](#)  
[starpu\\_omp\\_get\\_thread\\_num](#)  
[starpu\\_omp\\_get\\_max\\_threads](#)

**27.22.5.36** `int starpu_omp_in_parallel ( void )`

This function returns whether it is called from the scope of a parallel region or not.

**Returns**

! 0 if called from a parallel region scope.  
0 otherwise.

**27.22.5.37** `void starpu_omp_set_dynamic ( int dynamic_threads )`

This function enables (1) or disables (0) dynamically adjusting the number of parallel threads.

Note: The StarPU OpenMP runtime support currently ignores the argument of this function.

**See also**

[starpu\\_omp\\_get\\_dynamic](#)

**27.22.5.38** `int starpu_omp_get_dynamic ( void )`

This function returns the state of dynamic thread number adjustment.

**Returns**

! 0 if dynamic thread number adjustment is enabled.  
0 otherwise.

**See also**

[starpu\\_omp\\_set\\_dynamic](#)

**27.22.5.39 void starpu\_omp\_set\_nested ( int *nested* )**

This function enables (1) or disables (0) nested parallel regions.

Note: The StarPU OpenMP runtime support currently ignores the argument of this function.

See also

[starpu\\_omp\\_get\\_nested](#)  
[starpu\\_omp\\_get\\_max\\_active\\_levels](#)  
[starpu\\_omp\\_set\\_max\\_active\\_levels](#)  
[starpu\\_omp\\_get\\_level](#)  
[starpu\\_omp\\_get\\_active\\_level](#)

**27.22.5.40 int starpu\_omp\_get\_nested ( void )**

This function returns whether nested parallel sections are enabled or not.

Returns

! 0 if nested parallel sections are enabled.  
0 otherwise.

See also

[starpu\\_omp\\_set\\_nested](#)  
[starpu\\_omp\\_get\\_max\\_active\\_levels](#)  
[starpu\\_omp\\_set\\_max\\_active\\_levels](#)  
[starpu\\_omp\\_get\\_level](#)  
[starpu\\_omp\\_get\\_active\\_level](#)

**27.22.5.41 int starpu\_omp\_get\_cancellation ( void )**

This function returns the state of the cancel ICVS var.

**27.22.5.42 void starpu\_omp\_set\_schedule ( enum starpu\_omp\_sched\_value *kind*, int *modifier* )**

This function sets the default scheduling kind for upcoming loops within the current parallel section. *kind* is the scheduler kind, *modifier* complements the scheduler kind with informations such as the chunk size, in accordance with the OpenMP specification.

See also

[starpu\\_omp\\_get\\_schedule](#)

**27.22.5.43 void starpu\_omp\_get\_schedule ( enum starpu\_omp\_sched\_value \* *kind*, int \* *modifier* )**

This function returns the current selected default loop scheduler.

Returns

the kind and the modifier of the current default loop scheduler.

See also

[starpu\\_omp\\_set\\_schedule](#)

#### 27.22.5.44 `int starpu_omp_get_thread_limit ( void )`

This function returns the number of StarPU CPU workers.

##### Returns

the number of StarPU CPU workers.

#### 27.22.5.45 `void starpu_omp_set_max_active_levels ( int max_levels )`

This function sets the maximum number of allowed active parallel section levels.

Note: The StarPU OpenMP runtime support currently ignores the argument of this function and assume `max_levels` equals 1 instead.

##### See also

[starpu\\_omp\\_set\\_nested](#)  
[starpu\\_omp\\_get\\_nested](#)  
[starpu\\_omp\\_get\\_max\\_active\\_levels](#)  
[starpu\\_omp\\_get\\_level](#)  
[starpu\\_omp\\_get\\_active\\_level](#)

#### 27.22.5.46 `int starpu_omp_get_max_active_levels ( void )`

This function returns the current maximum number of allowed active parallel section levels

##### Returns

the current maximum number of allowed active parallel section levels.

##### See also

[starpu\\_omp\\_set\\_nested](#)  
[starpu\\_omp\\_get\\_nested](#)  
[starpu\\_omp\\_set\\_max\\_active\\_levels](#)  
[starpu\\_omp\\_get\\_level](#)  
[starpu\\_omp\\_get\\_active\\_level](#)

#### 27.22.5.47 `int starpu_omp_get_level ( void )`

This function returns the nesting level of the current parallel section.

##### Returns

the nesting level of the current parallel section.

##### See also

[starpu\\_omp\\_set\\_nested](#)  
[starpu\\_omp\\_get\\_nested](#)  
[starpu\\_omp\\_get\\_max\\_active\\_levels](#)  
[starpu\\_omp\\_set\\_max\\_active\\_levels](#)  
[starpu\\_omp\\_get\\_active\\_level](#)

**27.22.5.48** `int starpu_omp_get_ancestor_thread_num ( int level )`

This function returns the number of the ancestor of the current parallel section.

**Returns**

the number of the ancestor of the current parallel section.

**27.22.5.49** `int starpu_omp_get_team_size ( int level )`

This function returns the size of the team of the current parallel section.

**Returns**

the size of the team of the current parallel section.

**27.22.5.50** `int starpu_omp_get_active_level ( void )`

This function returns the nestinglevel of the current innermost active parallel section.

**Returns**

the nestinglevel of the current innermost active parallel section.

**See also**

[starpu\\_omp\\_set\\_nested](#)  
[starpu\\_omp\\_get\\_nested](#)  
[starpu\\_omp\\_get\\_max\\_active\\_levels](#)  
[starpu\\_omp\\_set\\_max\\_active\\_levels](#)  
[starpu\\_omp\\_get\\_level](#)

**27.22.5.51** `int starpu_omp_in_final ( void )`

This function checks whether the current task is final or not.

**Returns**

! 0 if called from a final task.  
0 otherwise.

**27.22.5.52** `enum starpu_omp_proc_bind_value starpu_omp_get_proc_bind ( void )`

This function returns the `proc_bind` setting of the current parallel region.

**Returns**

the `proc_bind` setting of the current parallel region.

27.22.5.53 `void starpu_omp_set_default_device ( int device_num )`

This function sets the number of the device to use as default.

Note: The StarPU OpenMP runtime support currently ignores the argument of this function.

See also

[starpu\\_omp\\_get\\_default\\_device](#)  
[starpu\\_omp\\_is\\_initial\\_device](#)

27.22.5.54 `int starpu_omp_get_default_device ( void )`

This function returns the number of the device used as default.

Returns

the number of the device used as default.

See also

[starpu\\_omp\\_set\\_default\\_device](#)  
[starpu\\_omp\\_is\\_initial\\_device](#)

27.22.5.55 `int starpu_omp_get_num_devices ( void )`

This function returns the number of the devices.

Returns

the number of the devices.

27.22.5.56 `int starpu_omp_get_num_teams ( void )`

This function returns the number of teams in the current teams region.

Returns

the number of teams in the current teams region.

See also

[starpu\\_omp\\_get\\_num\\_teams](#)

27.22.5.57 `int starpu_omp_get_team_num ( void )`

This function returns the team number of the calling thread.

Returns

the team number of the calling thread.

See also

[starpu\\_omp\\_get\\_num\\_teams](#)

27.22.5.58 `int starpu_omp_is_initial_device ( void )`

This function checks whether the current device is the initial device or not.

#### Returns

! 0 if called from the host device.  
0 otherwise.

#### See also

[starpu\\_omp\\_set\\_default\\_device](#)  
[starpu\\_omp\\_get\\_default\\_device](#)

27.22.5.59 `void starpu_omp_init_lock ( starpu_omp_lock_t * lock )`

This function initializes an opaque lock object.

#### See also

[starpu\\_omp\\_destroy\\_lock](#)  
[starpu\\_omp\\_set\\_lock](#)  
[starpu\\_omp\\_unset\\_lock](#)  
[starpu\\_omp\\_test\\_lock](#)

27.22.5.60 `void starpu_omp_destroy_lock ( starpu_omp_lock_t * lock )`

This function destroys an opaque lock object.

#### See also

[starpu\\_omp\\_init\\_lock](#)  
[starpu\\_omp\\_set\\_lock](#)  
[starpu\\_omp\\_unset\\_lock](#)  
[starpu\\_omp\\_test\\_lock](#)

27.22.5.61 `void starpu_omp_set_lock ( starpu_omp_lock_t * lock )`

This function locks an opaque lock object. If the lock is already locked, the function will block until it succeeds in exclusively acquiring the lock.

#### See also

[starpu\\_omp\\_init\\_lock](#)  
[starpu\\_omp\\_destroy\\_lock](#)  
[starpu\\_omp\\_unset\\_lock](#)  
[starpu\\_omp\\_test\\_lock](#)

27.22.5.62 `void starpu_omp_unset_lock ( starpu_omp_lock_t * lock )`

This function unlocks a previously locked lock object. The behaviour of this function is unspecified if it is called on an unlocked lock object.

See also

[starpu\\_omp\\_init\\_lock](#)  
[starpu\\_omp\\_destroy\\_lock](#)  
[starpu\\_omp\\_set\\_lock](#)  
[starpu\\_omp\\_test\\_lock](#)

**27.22.5.63** `int starpu_omp_test_lock ( starpu_omp_lock_t * lock )`

This function unblockingly attempts to lock a lock object and returns whether it succeeded or not.

Returns

! 0 if the function succeeded in acquiring the lock.  
0 if the lock was already locked.

See also

[starpu\\_omp\\_init\\_lock](#)  
[starpu\\_omp\\_destroy\\_lock](#)  
[starpu\\_omp\\_set\\_lock](#)  
[starpu\\_omp\\_unset\\_lock](#)

**27.22.5.64** `void starpu_omp_init_nest_lock ( starpu_omp_nest_lock_t * lock )`

This function initializes an opaque lock object supporting nested locking operations.

See also

[starpu\\_omp\\_destroy\\_nest\\_lock](#)  
[starpu\\_omp\\_set\\_nest\\_lock](#)  
[starpu\\_omp\\_unset\\_nest\\_lock](#)  
[starpu\\_omp\\_test\\_nest\\_lock](#)

**27.22.5.65** `void starpu_omp_destroy_nest_lock ( starpu_omp_nest_lock_t * lock )`

This function destroys an opaque lock object supporting nested locking operations.

See also

[starpu\\_omp\\_init\\_nest\\_lock](#)  
[starpu\\_omp\\_set\\_nest\\_lock](#)  
[starpu\\_omp\\_unset\\_nest\\_lock](#)  
[starpu\\_omp\\_test\\_nest\\_lock](#)

**27.22.5.66** `void starpu_omp_set_nest_lock ( starpu_omp_nest_lock_t * lock )`

This function locks an opaque lock object supporting nested locking operations. If the lock is already locked by another task, the function will block until it succeeds in exclusively acquiring the lock. If the lock is already taken by the current task, the function will increase the nested locking level of the lock object.

See also

[starpu\\_omp\\_init\\_nest\\_lock](#)  
[starpu\\_omp\\_destroy\\_nest\\_lock](#)  
[starpu\\_omp\\_unset\\_nest\\_lock](#)  
[starpu\\_omp\\_test\\_nest\\_lock](#)



**27.22.5.67 void starpu\_omp\_unset\_nest\_lock ( starpu\_omp\_nest\_lock\_t \* lock )**

This function unlocks a previously locked lock object supporting nested locking operations. If the lock has been locked multiple times in nested fashion, the nested locking level is decreased and the lock remains locked. Otherwise, if the lock has only been locked once, it becomes unlocked. The behaviour of this function is unspecified if it is called on an unlocked lock object. The behaviour of this function is unspecified if it is called from a different task than the one that locked the lock object.

See also

[starpu\\_omp\\_init\\_nest\\_lock](#)  
[starpu\\_omp\\_destroy\\_nest\\_lock](#)  
[starpu\\_omp\\_set\\_nest\\_lock](#)  
[starpu\\_omp\\_test\\_nest\\_lock](#)

**27.22.5.68 int starpu\_omp\_test\_nest\_lock ( starpu\_omp\_nest\_lock\_t \* lock )**

This function unblocking attempts to lock an opaque lock object supporting nested locking operations and returns whether it succeeded or not. If the lock is already locked by another task, the function will return without having acquired the lock. If the lock is already taken by the current task, the function will increase the nested locking level of the lock object.

Returns

! 0 if the function succeeded in acquiring the lock.  
0 if the lock was already locked.

See also

[starpu\\_omp\\_init\\_nest\\_lock](#)  
[starpu\\_omp\\_destroy\\_nest\\_lock](#)  
[starpu\\_omp\\_set\\_nest\\_lock](#)  
[starpu\\_omp\\_unset\\_nest\\_lock](#)

**27.22.5.69 void starpu\_omp\_atomic\_fallback\_inline\_begin ( void )**

This function implements the entry point of a fallback global atomic region. It blocks until it succeeds in acquiring exclusive access to the global atomic region.

See also

[starpu\\_omp\\_atomic\\_fallback\\_inline\\_end](#)

**27.22.5.70 void starpu\_omp\_atomic\_fallback\_inline\_end ( void )**

This function implements the exit point of a fallback global atomic region. It release the exclusive access to the global atomic region.

See also

[starpu\\_omp\\_atomic\\_fallback\\_inline\\_begin](#)

27.22.5.71 `double starpu_omp_get_wtime ( void )`

This function returns the elapsed wallclock time in seconds.

#### Returns

the elapsed wallclock time in seconds.

#### See also

[starpu\\_omp\\_get\\_wtick](#)

27.22.5.72 `double starpu_omp_get_wtick ( void )`

This function returns the precision of the time used by `starpu_omp_get_wtime`.

#### Returns

the precision of the time used by `starpu_omp_get_wtime`.

#### See also

[starpu\\_omp\\_get\\_wtime](#)

27.22.5.73 `void starpu_omp_vector_annotate ( starpu_data_handle_t handle, uint32_t slice_base )`

This function enables setting additional vector metadata needed by the OpenMP Runtime Support.

`handle` is vector data handle. `slice_base` is the base of an array slice, expressed in number of vector elements from the array base.

#### See also

[STARPU\\_VECTOR\\_GET\\_SLICE\\_BASE](#)

## 27.23 MIC Extensions

### Macros

- `#define STARPU_USE_MIC`
- `#define STARPU_MAXMICDEVS`

### Typedefs

- `typedef void * starpu_mic_func_symbol_t`

### Functions

- `int starpu_mic_register_kernel (starpu_mic_func_symbol_t *symbol, const char *func_name)`
- `starpu_mic_kernel_t starpu_mic_get_kernel (starpu_mic_func_symbol_t symbol)`

### 27.23.1 Detailed Description

### 27.23.2 Macro Definition Documentation

#### 27.23.2.1 `#define STARPU_USE_MIC`

This macro is defined when StarPU has been installed with MIC support. It should be used in your code to detect the availability of MIC.

#### 27.23.2.2 `#define STARPU_MAXMICDEVS`

This macro defines the maximum number of MIC devices that are supported by StarPU.

### 27.23.3 Typedef Documentation

#### 27.23.3.1 `starpu_mic_func_symbol_t`

Type for MIC function symbols

### 27.23.4 Function Documentation

#### 27.23.4.1 `int starpu_mic_register_kernel ( starpu_mic_func_symbol_t * symbol, const char * func_name )`

Initiate a lookup on each MIC device to find the adress of the function named `func_name`, store them in the global array `kernels` and return the index in the array through `symbol`.

#### 27.23.4.2 `starpu_mic_kernel_t starpu_mic_get_kernel ( starpu_mic_func_symbol_t symbol )`

If success, return the pointer to the function defined by `symbol` on the device linked to the called device. This can for instance be used in a `starpu_mic_func_t` implementation.

## 27.24 SCC Extensions

### Macros

- `#define STARPU_USE_SCC`
- `#define STARPU_MAXSCCDEVS`

### Typedefs

- `typedef void * starpu_scc_func_symbol_t`

### Functions

- `int starpu_scc_register_kernel (starpu_scc_func_symbol_t *symbol, const char *func_name)`
- `starpu_scc_kernel_t starpu_scc_get_kernel (starpu_scc_func_symbol_t symbol)`

### 27.24.1 Detailed Description

### 27.24.2 Macro Definition Documentation

#### 27.24.2.1 `#define STARPU_USE_SCC`

This macro is defined when StarPU has been installed with SCC support. It should be used in your code to detect the availability of SCC.

#### 27.24.2.2 `#define STARPU_MAXSCCDEVS`

This macro defines the maximum number of SCC devices that are supported by StarPU.

### 27.24.3 Typedef Documentation

#### 27.24.3.1 `starpu_scc_func_symbol_t`

Type for SCC function symbols

### 27.24.4 Function Documentation

#### 27.24.4.1 `int starpu_scc_register_kernel ( starpu_scc_func_symbol_t * symbol, const char * func_name )`

Initiate a lookup on each SCC device to find the adress of the function named `func_name`, store them in the global array `kernels` and return the index in the array through `symbol`.

#### 27.24.4.2 `starpu_scc_kernel_t starpu_scc_get_kernel ( starpu_scc_func_symbol_t symbol )`

If success, return the pointer to the function defined by `symbol` on the device linked to the called device. This can for instance be used in a `starpu_scc_func_t` implementation.

## 27.25 Miscellaneous Helpers

### Functions

- `int starpu_data_cpy ( starpu_data_handle_t dst_handle, starpu_data_handle_t src_handle, int asynchronous, void(*callback_func)(void *), void *callback_arg )`
- `void starpu_execute_on_each_worker ( void(*func)(void *), void *arg, uint32_t where )`
- `void starpu_execute_on_each_worker_ex ( void(*func)(void *), void *arg, uint32_t where, const char *name )`
- `void starpu_execute_on_specific_workers ( void(*func)(void *), void *arg, unsigned num_workers, unsigned *workers, const char *name )`
- `double starpu_timing_now ( void )`

### 27.25.1 Detailed Description

### 27.25.2 Function Documentation

#### 27.25.2.1 `int starpu_data_cpy ( starpu_data_handle_t dst_handle, starpu_data_handle_t src_handle, int asynchronous, void(*)(void *) callback_func, void * callback_arg )`

Copy the content of `src_handle` into `dst_handle`. The parameter `asynchronous` indicates whether the function should block or not. In the case of an asynchronous call, it is possible to synchronize with the termination

of this operation either by the means of implicit dependencies (if enabled) or by calling [starpu\\_task\\_wait\\_for\\_all\(\)](#). If `callback_func` is not NULL, this callback function is executed after the handle has been copied, and it is given the pointer `callback_arg` as argument.

**27.25.2.2** `void starpu_execute_on_each_worker ( void(*) (void *) func, void * arg, uint32_t where )`

This function executes the given function on a subset of workers. When calling this method, the offloaded function `func` is executed by every StarPU worker that may execute the function. The argument `arg` is passed to the offloaded function. The argument `where` specifies on which types of processing units the function should be executed. Similarly to the field [starpu\\_codelet::where](#), it is possible to specify that the function should be executed on every CUDA device and every CPU by passing [STARPU\\_CPU|STARPU\\_CUDA](#). This function blocks until the function has been executed on every appropriate processing units, so that it may not be called from a callback function for instance.

**27.25.2.3** `void starpu_execute_on_each_worker_ex ( void(*) (void *) func, void * arg, uint32_t where, const char * name )`

Same as [starpu\\_execute\\_on\\_each\\_worker\(\)](#), except that the task name is specified in the argument `name`.

**27.25.2.4** `void starpu_execute_on_specific_workers ( void(*) (void *) func, void * arg, unsigned num_workers, unsigned * workers, const char * name )`

Call `func(arg)` on every worker in the `workers` array. `num_workers` indicates the number of workers in this array. This function is synchronous, but the different workers may execute the function in parallel.

**27.25.2.5** `double starpu_timing_now ( void )`

Return the current date in micro-seconds.

## 27.26 FxT Support

### Data Structures

- struct [starpu\\_fxt\\_codelet\\_event](#)
- struct [starpu\\_fxt\\_options](#)

### Functions

- void [starpu\\_fxt\\_options\\_init](#) (struct [starpu\\_fxt\\_options](#) \*options)
- void [starpu\\_fxt\\_generate\\_trace](#) (struct [starpu\\_fxt\\_options](#) \*options)
- void [starpu\\_fxt\\_start\\_profiling](#) (void)
- void [starpu\\_fxt\\_stop\\_profiling](#) (void)
- void [starpu\\_fxt\\_write\\_data\\_trace](#) (char \*filename\_in)
- void [starpu\\_fxt\\_trace\\_user\\_event](#) (unsigned long code)

#### 27.26.1 Detailed Description

#### 27.26.2 Data Structure Documentation

##### 27.26.2.1 struct starpu\_fxt\_codelet\_event

todo

## Data Fields

|          |                              |                     |
|----------|------------------------------|---------------------|
| char     | symbol[256]                  | name of the codelet |
| int      | workerid                     |                     |
| char     | perfmodel_↔<br>archname[256] |                     |
| uint32_t | hash                         |                     |
| size_t   | size                         |                     |
| float    | time                         |                     |

## 27.26.2.2 struct starpu\_fxt\_options

todo

## Data Fields

|                                               |                                                      |                                                                                                                                      |
|-----------------------------------------------|------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| unsigned                                      | per_task_colour                                      |                                                                                                                                      |
| unsigned                                      | no_counter                                           |                                                                                                                                      |
| unsigned                                      | no_bus                                               |                                                                                                                                      |
| unsigned                                      | ninputfiles                                          |                                                                                                                                      |
| char *                                        | filenames[STA↔<br>RPU_FXT_MA↔<br>X_FILES]            |                                                                                                                                      |
| char *                                        | out_paje_path                                        |                                                                                                                                      |
| char *                                        | distrib_time_↔<br>path                               |                                                                                                                                      |
| char *                                        | activity_path                                        |                                                                                                                                      |
| char *                                        | dag_path                                             |                                                                                                                                      |
| char *                                        | file_prefix                                          | In case we are going to gather multiple traces (e.g in the case of MPI processes), we may need to prefix the name of the containers. |
| uint64_t                                      | file_offset                                          | In case we are going to gather multiple traces (e.g in the case of MPI processes), we may need to prefix the name of the containers. |
| int                                           | file_rank                                            | In case we are going to gather multiple traces (e.g in the case of MPI processes), we may need to prefix the name of the containers. |
| char                                          | worker_↔<br>names[STARP↔<br>U_NMAXWOR↔<br>KERS][256] | Output parameters                                                                                                                    |
| struct starpu_↔<br>perfmodel_arch             | worker_↔<br>archtypes[STA↔<br>RPU_NMAXW↔<br>ORKERS]  | Output parameters                                                                                                                    |
| int                                           | nworkers                                             | Output parameters                                                                                                                    |
| struct<br>starpu_fxt_↔<br>codelet_event<br>** | dumped_↔<br>codelets                                 | In case we want to dump the list of codelets to an external tool                                                                     |
| long                                          | dumped_↔<br>codelets_count                           | In case we want to dump the list of codelets to an external tool                                                                     |

## 27.26.3 Function Documentation

## 27.26.3.1 void starpu\_fxt\_options\_init ( struct starpu\_fxt\_options \* options )

todo

27.26.3.2 void `starpu_fxt_generate_trace` ( struct `starpu_fxt_options` \* *options* )

todo

27.26.3.3 void `starpu_fxt_start_profiling` ( void )

Start recording the trace. The trace is by default started from `starpu_init()` call, but can be paused by using `starpu_fxt_stop_profiling()`, in which case `starpu_fxt_start_profiling()` should be called to resume recording events.

27.26.3.4 void `starpu_fxt_stop_profiling` ( void )

Stop recording the trace. The trace is by default stopped when calling `starpu_shutdown()`. `starpu_fxt_stop_profiling()` can however be used to stop it earlier. `starpu_fxt_start_profiling()` can then be called to start recording it again, etc.

27.26.3.5 void `starpu_fxt_write_data_trace` ( char \* *filename\_in* )

todo

27.26.3.6 void `starpu_fxt_trace_user_event` ( unsigned long *code* )

Add an event in the execution trace if FxT is enabled.

## 27.27 FFT Support

### Functions

- void \* `starpufft_malloc` (size\_t *n*)
- void `starpufft_free` (void \**p*)
- starpufft\_plan `starpufft_plan_dft_1d` (int *n*, int *sign*, unsigned *flags*)
- starpufft\_plan `starpufft_plan_dft_2d` (int *n*, int *m*, int *sign*, unsigned *flags*)
- struct `starpu_task` \* `starpufft_start` (starpufft\_plan *p*, void \**in*, void \**out*)
- struct `starpu_task` \* `starpufft_start_handle` (starpufft\_plan *p*, `starpu_data_handle_t` *in*, `starpu_data_handle_t` *out*)
- int `starpufft_execute` (starpufft\_plan *p*, void \**in*, void \**out*)
- int `starpufft_execute_handle` (starpufft\_plan *p*, `starpu_data_handle_t` *in*, `starpu_data_handle_t` *out*)
- void `starpufft_cleanup` (starpufft\_plan *p*)
- void `starpufft_destroy_plan` (starpufft\_plan *p*)

### 27.27.1 Detailed Description

### 27.27.2 Function Documentation

27.27.2.1 void \* `starpufft_malloc` ( size\_t *n* )

Allocates memory for *n* bytes. This is preferred over `malloc()`, since it allocates pinned memory, which allows overlapped transfers.

27.27.2.2 void \* `starpufft_free` ( void \* *p* )

Release memory previously allocated.

**27.27.2.3** `struct starpufft_plan * starpufft_plan_dft_1d ( int n, int sign, unsigned flags )`

Initializes a plan for 1D FFT of size *n*. *sign* can be STARPUFFT\_FORWARD or STARPUFFT\_INVERSE. *flags* must be 0.

**27.27.2.4** `struct starpufft_plan * starpufft_plan_dft_2d ( int n, int m, int sign, unsigned flags )`

Initializes a plan for 2D FFT of size (*n*, *m*). *sign* can be STARPUFFT\_FORWARD or STARPUFFT\_INVERSE. *flags* must be 0.

**27.27.2.5** `struct starpu_task * starpufft_start ( starpufft_plan p, void * in, void * out )`

Start an FFT previously planned as *p*, using *in* and *out* as input and output. This only submits the task and does not wait for it. The application should call [starpufft\\_cleanup\(\)](#) to unregister the

**27.27.2.6** `struct starpu_task * starpufft_start_handle ( starpufft_plan p, starpu_data_handle_t in, starpu_data_handle_t out )`

Start an FFT previously planned as *p*, using data handles *in* and *out* as input and output (assumed to be vectors of elements of the expected types). This only submits the task and does not wait for it.

**27.27.2.7** `void starpufft_execute ( starpufft_plan p, void * in, void * out )`

Execute an FFT previously planned as *p*, using *in* and *out* as input and output. This submits and waits for the task.

**27.27.2.8** `void starpufft_execute_handle ( starpufft_plan p, starpu_data_handle_t in, starpu_data_handle_t out )`

Execute an FFT previously planned as *p*, using data handles *in* and *out* as input and output (assumed to be vectors of elements of the expected types). This submits and waits for the task.

**27.27.2.9** `void starpufft_cleanup ( starpufft_plan p )`

Releases data for plan *p*, in the [starpufft\\_start\(\)](#) case.

**27.27.2.10** `void starpufft_destroy_plan ( starpufft_plan p )`

Destroys plan *p*, i.e. release all CPU (fftw) and GPU (cufft) resources.

## 27.28 MPI Support

### Initialisation

- `#define STARPU_USE_MPI`
- `int starpu_mpi_init (int *argc, char ***argv, int initialize_mpi)`
- `int starpu_mpi_initialize (void)`
- `int starpu_mpi_initialize_extended (int *rank, int *world_size)`
- `int starpu_mpi_shutdown (void)`
- `void starpu_mpi_comm_amounts_retrieve (size_t *comm_amounts)`



## Communication

- int [starpu\\_mpi\\_send](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int dest, int mpi\_tag, MPI\_Comm comm)
- int [starpu\\_mpi\\_recv](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int source, int mpi\_tag, MPI\_Comm comm, MPI\_Status \*status)
- int [starpu\\_mpi\\_isend](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, [starpu\\_mpi\\_req](#) \*req, int dest, int mpi\_tag, MPI\_Comm comm)
- int [starpu\\_mpi\\_irecv](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, [starpu\\_mpi\\_req](#) \*req, int source, int mpi\_tag, MPI\_Comm comm)
- int [starpu\\_mpi\\_isend\\_detached](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int dest, int mpi\_tag, MPI\_Comm comm, void(\*callback)(void \*), void \*arg)
- int [starpu\\_mpi\\_irecv\\_detached](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int source, int mpi\_tag, MPI\_Comm comm, void(\*callback)(void \*), void \*arg)
- int [starpu\\_mpi\\_irecv\\_detached\\_sequential\\_consistency](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int source, int mpi\_tag, MPI\_Comm comm, void(\*callback)(void \*), void \*arg, int sequential\_consistency)
- int [starpu\\_mpi\\_issend](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, [starpu\\_mpi\\_req](#) \*req, int dest, int mpi\_tag, MPI\_Comm comm)
- int [starpu\\_mpi\\_wait](#) ([starpu\\_mpi\\_req](#) \*req, MPI\_Status \*status)
- int [starpu\\_mpi\\_test](#) ([starpu\\_mpi\\_req](#) \*req, int \*flag, MPI\_Status \*status)
- int [starpu\\_mpi\\_barrier](#) (MPI\_Comm comm)
- int [starpu\\_mpi\\_isend\\_detached\\_unlock\\_tag](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int dest, int mpi\_tag, MPI\_Comm comm, [starpu\\_tag\\_t](#) tag)
- int [starpu\\_mpi\\_irecv\\_detached\\_unlock\\_tag](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int source, int mpi\_tag, MPI\_Comm comm, [starpu\\_tag\\_t](#) tag)
- int [starpu\\_mpi\\_isend\\_array\\_detached\\_unlock\\_tag](#) (unsigned array\_size, [starpu\\_data\\_handle\\_t](#) \*data\_handle, int \*dest, int \*mpi\_tag, MPI\_Comm \*comm, [starpu\\_tag\\_t](#) tag)
- int [starpu\\_mpi\\_irecv\\_array\\_detached\\_unlock\\_tag](#) (unsigned array\_size, [starpu\\_data\\_handle\\_t](#) \*data\_handle, int \*source, int \*mpi\_tag, MPI\_Comm \*comm, [starpu\\_tag\\_t](#) tag)
- int [starpu\\_mpi\\_get\\_communication\\_tag](#) (void)
- void [starpu\\_mpi\\_set\\_communication\\_tag](#) (int tag)

## Communication Cache

- void [starpu\\_mpi\\_cache\\_flush](#) (MPI\_Comm comm, [starpu\\_data\\_handle\\_t](#) data\_handle)
- void [starpu\\_mpi\\_cache\\_flush\\_all\\_data](#) (MPI\_Comm comm)

## MPI Insert Task

- #define [STARPU\\_EXECUTE\\_ON\\_NODE](#)
- #define [STARPU\\_EXECUTE\\_ON\\_DATA](#)
- int [starpu\\_data\\_set\\_tag](#) ([starpu\\_data\\_handle\\_t](#) handle, int tag)
- int [starpu\\_data\\_get\\_tag](#) ([starpu\\_data\\_handle\\_t](#) handle)
- void [starpu\\_mpi\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int tag, int rank)
- int [starpu\\_data\\_set\\_rank](#) ([starpu\\_data\\_handle\\_t](#) handle, int rank)
- int [starpu\\_data\\_get\\_rank](#) ([starpu\\_data\\_handle\\_t](#) handle)
- int [starpu\\_mpi\\_insert\\_task](#) (MPI\_Comm comm, struct [starpu\\_codelet](#) \*codelet,...)
- int [starpu\\_mpi\\_task\\_insert](#) (MPI\_Comm comm, struct [starpu\\_codelet](#) \*codelet,...)
- struct [starpu\\_task](#) \* [starpu\\_mpi\\_task\\_build](#) (MPI\_Comm comm, struct [starpu\\_codelet](#) \*codelet,...)
- int [starpu\\_mpi\\_task\\_post\\_build](#) (MPI\_Comm comm, struct [starpu\\_codelet](#) \*codelet,...)
- void [starpu\\_mpi\\_get\\_data\\_on\\_node](#) (MPI\_Comm comm, [starpu\\_data\\_handle\\_t](#) data\_handle, int node)
- void [starpu\\_mpi\\_get\\_data\\_on\\_node\\_detached](#) (MPI\_Comm comm, [starpu\\_data\\_handle\\_t](#) data\_handle, int node, void(\*callback)(void \*), void \*arg)
- char \* [starpu\\_mpi\\_node\\_selection\\_get\\_default\\_policy](#) ()
- int [starpu\\_mpi\\_node\\_selection\\_set\\_default\\_policy](#) (char \*policy)

## Collective Operations

- void [starpupmpi\\_redux\\_data](#) (MPI\_Comm comm, [starpupmpi\\_data\\_handle\\_t](#) data\_handle)
- int [starpupmpi\\_scatter\\_detached](#) ([starpupmpi\\_data\\_handle\\_t](#) \*data\_handles, int count, int root, MPI\_Comm comm, void(\*scallback)(void \*), void \*sarg, void(\*rcallback)(void \*), void \*rarg)
- int [starpupmpi\\_gather\\_detached](#) ([starpupmpi\\_data\\_handle\\_t](#) \*data\_handles, int count, int root, MPI\_Comm comm, void(\*scallback)(void \*), void \*sarg, void(\*rcallback)(void \*), void \*rarg)

### 27.28.1 Detailed Description

### 27.28.2 Macro Definition Documentation

#### 27.28.2.1 #define STARPU\_USE\_MPI

This macro is defined when StarPU has been installed with MPI support. It should be used in your code to detect the availability of MPI.

#### 27.28.2.2 #define STARPU\_EXECUTE\_ON\_NODE

this macro is used when calling [starpupmpi\\_task\\_insert\(\)](#), and must be followed by a integer value which specified the node on which to execute the codelet.

#### 27.28.2.3 #define STARPU\_EXECUTE\_ON\_DATA

this macro is used when calling [starpupmpi\\_task\\_insert\(\)](#), and must be followed by a data handle to specify that the node owning the given data will execute the codelet.

### 27.28.3 Function Documentation

#### 27.28.3.1 int starpupmpi\_init ( int \* argc, char \*\*\* argv, int initialize\_mpi )

Initializes the starpupmpi library. `initialize_mpi` indicates if MPI should be initialized or not by StarPU. If the value is not 0, MPI will be initialized by calling `MPI_Init_Thread(argc, argv, MPI_THREAD_SERIALIZED, ...)`.

#### 27.28.3.2 int starpupmpi\_initialize ( void )

**Deprecated** This function has been made deprecated. One should use instead the function [starpupmpi\\_init\(\)](#). This function does not call `MPI_Init()`, it should be called beforehand.

#### 27.28.3.3 int starpupmpi\_initialize\_extended ( int \* rank, int \* world\_size )

**Deprecated** This function has been made deprecated. One should use instead the function [starpupmpi\\_init\(\)](#). MPI will be initialized by starpupmpi by calling `MPI_Init_Thread(argc, argv, MPI_THREAD_SERIALIZED, ...)`.

#### 27.28.3.4 int starpupmpi\_shutdown ( void )

Cleans the starpupmpi library. This must be called between calling starpupmpi functions and [starpupmpi\\_shutdown\(\)](#). `MPI_Finalize()` will be called if StarPU-MPI has been initialized by [starpupmpi\\_init\(\)](#).

27.28.3.5 `void starpu_mpi_comm_amounts_retrieve ( size_t * comm_amounts )`

Retrieve the current amount of communications from the current node in the array `comm_amounts` which must have a size greater or equal to the world size. Communications statistics must be enabled (see [STARPU\\_COMM↔\\_STATS](#)).

27.28.3.6 `int starpu_mpi_send ( starpu_data_handle_t data_handle, int dest, int mpi_tag, MPI_Comm comm )`

Performs a standard-mode, blocking send of `data_handle` to the node `dest` using the message tag `mpi_tag` within the communicator `comm`.

27.28.3.7 `int starpu_mpi_recv ( starpu_data_handle_t data_handle, int source, int mpi_tag, MPI_Comm comm, MPI_Status * status )`

Performs a standard-mode, blocking receive in `data_handle` from the node `source` using the message tag `mpi_tag` within the communicator `comm`.

27.28.3.8 `int starpu_mpi_isend ( starpu_data_handle_t data_handle, starpu_mpi_req * req, int dest, int mpi_tag, MPI_Comm comm )`

Posts a standard-mode, non blocking send of `data_handle` to the node `dest` using the message tag `mpi_tag` within the communicator `comm`. After the call, the pointer to the request `req` can be used to test or to wait for the completion of the communication.

27.28.3.9 `int starpu_mpi_irecv ( starpu_data_handle_t data_handle, starpu_mpi_req * req, int source, int mpi_tag, MPI_Comm comm )`

Posts a nonblocking receive in `data_handle` from the node `source` using the message tag `mpi_tag` within the communicator `comm`. After the call, the pointer to the request `req` can be used to test or to wait for the completion of the communication.

27.28.3.10 `int starpu_mpi_isend_detached ( starpu_data_handle_t data_handle, int dest, int mpi_tag, MPI_Comm comm, void(*) (void *) callback, void * arg )`

Posts a standard-mode, non blocking send of `data_handle` to the node `dest` using the message tag `mpi_tag` within the communicator `comm`. On completion, the `callback` function is called with the argument `arg`. Similarly to the pthread detached functionality, when a detached communication completes, its resources are automatically released back to the system, there is no need to test or to wait for the completion of the request.

27.28.3.11 `int starpu_mpi_irecv_detached ( starpu_data_handle_t data_handle, int source, int mpi_tag, MPI_Comm comm, void(*) (void *) callback, void * arg )`

Posts a nonblocking receive in `data_handle` from the node `source` using the message tag `mpi_tag` within the communicator `comm`. On completion, the `callback` function is called with the argument `arg`. Similarly to the pthread detached functionality, when a detached communication completes, its resources are automatically released back to the system, there is no need to test or to wait for the completion of the request.

27.28.3.12 `int starpu_mpi_irecv_detached_sequential_consistency ( starpu_data_handle_t data_handle, int source, int mpi_tag, MPI_Comm comm, void(*) (void *) callback, void * arg, int sequential_consistency )`

Posts a nonblocking receive in `data_handle` from the node `source` using the message tag `mpi_tag` within the communicator `comm`. On completion, the `callback` function is called with the argument `arg`. The parameter

`sequential_consistency` allows to enable or disable the sequential consistency for data handle (sequential consistency will be enabled or disabled based on the value of the parameter `sequential_consistency` and the value of the sequential consistency defined for `data_handle`). Similarly to the pthread detached functionality, when a detached communication completes, its resources are automatically released back to the system, there is no need to test or to wait for the completion of the request.

**27.28.3.13** `int starpu_mpi_issend ( starpu_data_handle_t data_handle, starpu_mpi_req * req, int dest, int mpi_tag, MPI_Comm comm )`

Performs a synchronous-mode, non-blocking send of `data_handle` to the node `dest` using the message tag `mpi_tag` within the communicator `comm`.

`int starpu_mpi_issend_detached(starpu_data_handle_t data_handle, starpu_mpi_req *req, int dest, int mpi_tag, MPI_Comm comm, void (*callback)(void *), void *arg)`

Performs a synchronous-mode, non-blocking send of `data_handle` to the node `dest` using the message tag `mpi_tag` within the communicator `comm`. On completion, the `callback` function is called with the argument `arg`. Similarly to the pthread detached functionality, when a detached communication completes, its resources are automatically released back to the system, there is no need to test or to wait for the completion of the request.

**27.28.3.14** `int starpu_mpi_wait ( starpu_mpi_req * req, MPI_Status * status )`

Returns when the operation identified by request `req` is complete.

**27.28.3.15** `int starpu_mpi_test ( starpu_mpi_req * req, int * flag, MPI_Status * status )`

If the operation identified by `req` is complete, set `flag` to 1. The `status` object is set to contain information on the completed operation.

**27.28.3.16** `int starpu_mpi_barrier ( MPI_Comm comm )`

Blocks the caller until all group members of the communicator `comm` have called it.

**27.28.3.17** `int starpu_mpi_isend_detached_unlock_tag ( starpu_data_handle_t data_handle, int dest, int mpi_tag, MPI_Comm comm, starpu_tag_t tag )`

Posts a standard-mode, non blocking send of `data_handle` to the node `dest` using the message tag `mpi_tag` within the communicator `comm`. On completion, `tag` is unlocked.

**27.28.3.18** `int starpu_mpi_irecv_detached_unlock_tag ( starpu_data_handle_t data_handle, int source, int mpi_tag, MPI_Comm comm, starpu_tag_t tag )`

Posts a nonblocking receive in `data_handle` from the node `source` using the message tag `mpi_tag` within the communicator `comm`. On completion, `tag` is unlocked.

**27.28.3.19** `int starpu_mpi_isend_array_detached_unlock_tag ( unsigned array_size, starpu_data_handle_t * data_handle, int * dest, int * mpi_tag, MPI_Comm * comm, starpu_tag_t tag )`

Posts `array_size` standard-mode, non blocking send. Each post sends the `n`-th data of the array `data_handle` to the `n`-th node of the array `dest` using the `n`-th message tag of the array `mpi_tag` within the `n`-th communicator of the array `comm`. On completion of the all the requests, `tag` is unlocked.

**27.28.3.20** `int starpu_mpi_irecv_array_detached_unlock_tag ( unsigned array_size, starpu_data_handle_t * data_handle, int * source, int * mpi_tag, MPI_Comm * comm, starpu_tag_t tag )`

Posts `array_size` nonblocking receive. Each post receives in the `n`-th data of the array `data_handle` from the `n`-th node of the array `source` using the `n`-th message tag of the array `mpi_tag` within the `n`-th communicator of the array `comm`. On completion of the all the requests, `tag` is unlocked.

**27.28.3.21** `int starpu_mpi_get_communication_tag ( void )`

todo

**27.28.3.22** `void starpu_mpi_set_communication_tag ( int tag )`

todo

**27.28.3.23** `void starpu_mpi_cache_flush ( MPI_Comm comm, starpu_data_handle_t data_handle )`

Clear the send and receive communication cache for the data `data_handle` and invalidate the value. The function has to be called synchronously by all the MPI nodes. The function does nothing if the cache mechanism is disabled (see [STARPU\\_MPI\\_CACHE](#)).

**27.28.3.24** `void starpu_mpi_cache_flush_all_data ( MPI_Comm comm )`

Clear the send and receive communication cache for all data and invalidate their values. The function has to be called synchronously by all the MPI nodes. The function does nothing if the cache mechanism is disabled (see [STARPU\\_MPI\\_CACHE](#)).

**27.28.3.25** `int starpu_data_set_tag ( starpu_data_handle_t handle, int tag )`

Tell StarPU-MPI which MPI tag to use when exchanging the data.

**27.28.3.26** `int starpu_data_get_tag ( starpu_data_handle_t handle )`

Returns the MPI tag to be used when exchanging the data.

**27.28.3.27** `void starpu_mpi_data_register ( starpu_data_handle_t data_handle, int tag, int rank )`

Calling this function should be preferred to calling both [starpu\\_data\\_set\\_rank\(\)](#) and [starpu\\_data\\_set\\_tag\(\)](#) as it also allows to automatically clear the MPI communication cache when unregistering the data.

**27.28.3.28** `int starpu_data_set_rank ( starpu_data_handle_t handle, int rank )`

Tell StarPU-MPI which MPI node "owns" a given data, that is, the node which will always keep an up-to-date value, and will by default execute tasks which write to it.

**27.28.3.29** `int starpu_data_get_rank ( starpu_data_handle_t handle )`

Returns the last value set by [starpu\\_data\\_set\\_rank\(\)](#).

**27.28.3.30** `int starpu_mpi_insert_task ( MPI_Comm comm, struct starpu_codelet * codelet, ... )`

This function does the same as the function [starpu\\_mpi\\_task\\_insert\(\)](#). It has been kept to avoid breaking old codes.

**27.28.3.31** `int starpu_mpi_task_insert ( MPI_Comm comm, struct starpu_codelet * codelet, ... )`

Create and submit a task corresponding to codelet with the following arguments. The argument list must be zero-terminated.

The arguments following the codelet are the same types as for the function [starpu\\_task\\_insert\(\)](#). Access modes for data can also be set with [STARPU\\_SSEND](#) to specify the data has to be sent using a synchronous and non-blocking mode (see [starpu\\_mpi\\_issend\(\)](#)). The extra argument [STARPU\\_EXECUTE\\_ON\\_NODE](#) followed by an integer allows to specify the MPI node to execute the codelet. It is also possible to specify that the node owning a specific data will execute the codelet, by using [STARPU\\_EXECUTE\\_ON\\_DATA](#) followed by a data handle.

The internal algorithm is as follows:

1. Find out which MPI node is going to execute the codelet.
  - If there is only one node owning data in [STARPU\\_W](#) mode, it will be selected;
  - If there is several nodes owning data in [STARPU\\_W](#) mode, a node will be selected according to a given node selection policy (see [::STARPU\\_NODE\\_SELECTION\\_POLICY](#) or [starpu\\_mpi\\_node\\_selection\\_set\\_default\\_policy\(\)](#));
  - The argument [STARPU\\_EXECUTE\\_ON\\_NODE](#) followed by an integer can be used to specify the node;
  - The argument [STARPU\\_EXECUTE\\_ON\\_DATA](#) followed by a data handle can be used to specify that the node owning the given data will execute the codelet.
2. Send and receive data as requested. Nodes owning data which need to be read by the task are sending them to the MPI node which will execute it. The latter receives them.
3. Execute the codelet. This is done by the MPI node selected in the 1st step of the algorithm.
4. If several MPI nodes own data to be written to, send written data back to their owners.

The algorithm also includes a communication cache mechanism that allows not to send data twice to the same MPI node, unless the data has been modified. The cache can be disabled (see [STARPU\\_MPI\\_CACHE](#)).

**27.28.3.32** `struct starpu_task * starpu_mpi_task_build ( MPI_Comm comm, struct starpu_codelet * codelet, ... )`

Create a task corresponding to codelet with the following arguments. The argument list must be zero-terminated. The function performs the first two steps of the function [starpu\\_mpi\\_task\\_insert\(\)](#). Only the MPI node selected in the first step of the algorithm will return a valid task structure which can then be submitted. The function [starpu\\_mpi\\_task\\_post\\_build\(\)](#) MUST be called after the submission of the task, with the SAME list of arguments.

**27.28.3.33** `int starpu_mpi_task_post_build ( MPI_Comm comm, struct starpu_codelet * codelet, ... )`

This function MUST be called after a call to [starpu\\_mpi\\_task\\_build\(\)](#), with the SAME list of arguments. It performs the fourth – last – step of the algorithm described in [starpu\\_mpi\\_task\\_insert\(\)](#).

**27.28.3.34** `void starpu_mpi_get_data_on_node ( MPI_Comm comm, starpu_data_handle_t data_handle, int node )`

Transfer data `data_handle` to MPI node `node`, sending it from its owner if needed. At least the target node and the owner have to call the function.

**27.28.3.35** `void starpu_mpi_get_data_on_node_detached ( MPI_Comm comm, starpu_data_handle_t data_handle, int node, void(*) (void *) callback, void * arg )`

Transfer data `data_handle` to MPI node `node`, sending it from its owner if needed. At least the target node and the owner have to call the function. On reception, the `callback` function is called with the argument `arg`.

**27.28.3.36** `char * starpu_mpi_node_selection_get_default_policy ( )`

Return the current default policy used to select the node which will execute the codelet

**27.28.3.37** `int starpu_mpi_node_selection_set_default_policy ( char * policy )`

Set the current default policy used to select the node which will execute the codelet. The policy "node\_with\_most\_R\_data" selects the node having the most data in R mode so as to minimize the amount of data to be transferred.

**27.28.3.38** `void starpu_mpi_redux_data ( MPI_Comm comm, starpu_data_handle_t data_handle )`

Perform a reduction on the given data. All nodes send the data to its owner node which will perform a reduction.

**27.28.3.39** `int starpu_mpi_scatter_detached ( starpu_data_handle_t * data_handles, int count, int root, MPI_Comm comm, void(*) (void *) scallback, void * sarg, void(*) (void *) rcallback, void * rarg )`

Scatter data among processes of the communicator based on the ownership of the data. For each data of the array `data_handles`, the process `root` sends the data to the process owning this data. Processes receiving data must have valid data handles to receive them. On completion of the collective communication, the `scallback` function is called with the argument `sarg` on the process `root`, the `rcallback` function is called with the argument `rarg` on any other process.

**27.28.3.40** `int starpu_mpi_gather_detached ( starpu_data_handle_t * data_handles, int count, int root, MPI_Comm comm, void(*) (void *) scallback, void * sarg, void(*) (void *) rcallback, void * rarg )`

Gather data from the different processes of the communicator onto the process `root`. Each process owning data handle in the array `data_handles` will send them to the process `root`. The process `root` must have valid data handles to receive the data. On completion of the collective communication, the `rcallback` function is called with the argument `rarg` on the process `root`, the `scallback` function is called with the argument `sarg` on any other process.

## 27.29 Task Bundles

### Typedefs

- typedef struct  
\_starpu\_task\_bundle \* [starpu\\_task\\_bundle\\_t](#)

### Functions

- void [starpu\\_task\\_bundle\\_create](#) ([starpu\\_task\\_bundle\\_t](#) \*bundle)
- int [starpu\\_task\\_bundle\\_insert](#) ([starpu\\_task\\_bundle\\_t](#) bundle, struct [starpu\\_task](#) \*task)
- int [starpu\\_task\\_bundle\\_remove](#) ([starpu\\_task\\_bundle\\_t](#) bundle, struct [starpu\\_task](#) \*task)
- void [starpu\\_task\\_bundle\\_close](#) ([starpu\\_task\\_bundle\\_t](#) bundle)

- double `starpu_task_bundle_expected_length` (`starpu_task_bundle_t` bundle, struct `starpu_perfmodel_arch` \*arch, unsigned nimpl)
- double `starpu_task_bundle_expected_power` (`starpu_task_bundle_t` bundle, struct `starpu_perfmodel_arch` \*arch, unsigned nimpl)
- double `starpu_task_bundle_expected_data_transfer_time` (`starpu_task_bundle_t` bundle, unsigned memory\_node)

### 27.29.1 Detailed Description

### 27.29.2 Typedef Documentation

#### 27.29.2.1 `starpu_task_bundle_t`

Opaque structure describing a list of tasks that should be scheduled on the same worker whenever it's possible. It must be considered as a hint given to the scheduler as there is no guarantee that they will be executed on the same worker.

### 27.29.3 Function Documentation

#### 27.29.3.1 `void starpu_task_bundle_create ( starpu_task_bundle_t * bundle )`

Factory function creating and initializing `bundle`, when the call returns, memory needed is allocated and `bundle` is ready to use.

#### 27.29.3.2 `int starpu_task_bundle_insert ( starpu_task_bundle_t bundle, struct starpu_task * task )`

Insert `task` in `bundle`. Until `task` is removed from `bundle` its expected length and data transfer time will be considered along those of the other tasks of `bundle`. This function must not be called if `bundle` is already closed and/or `task` is already submitted. On success, it returns 0. There are two cases of error : if `bundle` is already closed it returns `-EPERM`, if `task` was already submitted it returns `-EINVAL`.

#### 27.29.3.3 `int starpu_task_bundle_remove ( starpu_task_bundle_t bundle, struct starpu_task * task )`

Remove `task` from `bundle`. Of course `task` must have been previously inserted in `bundle`. This function must not be called if `bundle` is already closed and/or `task` is already submitted. Doing so would result in undefined behaviour. On success, it returns 0. If `bundle` is already closed it returns `-ENOENT`.

#### 27.29.3.4 `void starpu_task_bundle_close ( starpu_task_bundle_t bundle )`

Inform the runtime that the user will not modify `bundle` anymore, it means no more inserting or removing task. Thus the runtime can destroy it when possible.

#### 27.29.3.5 `double starpu_task_bundle_expected_length ( starpu_task_bundle_t bundle, struct starpu_perfmodel_arch * arch, unsigned nimpl )`

Return the expected duration of `bundle` in micro-seconds.

#### 27.29.3.6 `double starpu_task_bundle_expected_power ( starpu_task_bundle_t bundle, struct starpu_perfmodel_arch * arch, unsigned nimpl )`

Return the expected power consumption of `bundle` in J.



27.29.3.7 `double starpu_task_bundle_expected_data_transfer_time ( starpu_task_bundle_t bundle, unsigned memory_node )`

Return the time (in micro-seconds) expected to transfer all data used within `bundle`.

## 27.30 Task Lists

### Data Structures

- struct [starpu\\_task\\_list](#)

### Functions

- static STARPU\_INLINE void [starpu\\_task\\_list\\_init](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE void [starpu\\_task\\_list\\_push\\_front](#) (struct [starpu\\_task\\_list](#) \*list, struct [starpu\\_task](#) \*task)
- static STARPU\_INLINE void [starpu\\_task\\_list\\_push\\_back](#) (struct [starpu\\_task\\_list](#) \*list, struct [starpu\\_task](#) \*task)
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_front](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_back](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE int [starpu\\_task\\_list\\_empty](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE void [starpu\\_task\\_list\\_erase](#) (struct [starpu\\_task\\_list](#) \*list, struct [starpu\\_task](#) \*task)
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_pop\\_front](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_pop\\_back](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_begin](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_next](#) (struct [starpu\\_task](#) \*task)

### 27.30.1 Detailed Description

### 27.30.2 Data Structure Documentation

#### 27.30.2.1 struct [starpu\\_task\\_list](#)

Stores a double-chained list of tasks

#### Data Fields

|                                      |      |                  |
|--------------------------------------|------|------------------|
| struct <a href="#">starpu_task</a> * | head | head of the list |
| struct <a href="#">starpu_task</a> * | tail | tail of the list |

### 27.30.3 Function Documentation

27.30.3.1 `void starpu_task_list_init ( struct starpu\_task\_list * list )` `[static]`

Initialize a list structure

**27.30.3.2** `void starpu_task_list_push_front ( struct starpu_task_list * list, struct starpu_task * task )` `[static]`

Push `task` at the front of `list`

**27.30.3.3** `void starpu_task_list_push_back ( struct starpu_task_list * list, struct starpu_task * task )` `[static]`

Push `task` at the back of `list`

**27.30.3.4** `struct starpu_task * starpu_task_list_front ( struct starpu_task_list * list )` `[static]`

Get the front of `list` (without removing it)

**27.30.3.5** `struct starpu_task * starpu_task_list_back ( struct starpu_task_list * list )` `[static]`

Get the back of `list` (without removing it)

**27.30.3.6** `int starpu_task_list_empty ( struct starpu_task_list * list )` `[static]`

Test if `list` is empty

**27.30.3.7** `void starpu_task_list_erase ( struct starpu_task_list * list, struct starpu_task * task )` `[static]`

Remove `task` from `list`

**27.30.3.8** `struct starpu_task * starpu_task_list_pop_front ( struct starpu_task_list * list )` `[static]`

Remove the element at the front of `list`

**27.30.3.9** `struct starpu_task * starpu_task_list_pop_back ( struct starpu_task_list * list )` `[static]`

Remove the element at the back of `list`

**27.30.3.10** `struct starpu_task * starpu_task_list_begin ( struct starpu_task_list * list )` `[static]`

Get the first task of `list`.

**27.30.3.11** `struct starpu_task * starpu_task_list_next ( struct starpu_task * task )` `[static]`

Get the next task of `list`. This is not erase-safe.

## 27.31 Parallel Tasks

### Functions

- `int starpu_combined_worker_get_size` (void)
- `int starpu_combined_worker_get_rank` (void)
- `unsigned starpu_combined_worker_get_count` (void)
- `int starpu_combined_worker_get_id` (void)
- `int starpu_combined_worker_assign_workerid` (int nworkers, int workerid\_array[])

- int [starpu\\_combined\\_worker\\_get\\_description](#) (int workerid, int \*worker\_size, int \*\*combined\_workerid)
- int [starpu\\_combined\\_worker\\_can\\_execute\\_task](#) (unsigned workerid, struct [starpu\\_task](#) \*task, unsigned nimpl)
- void [starpu\\_parallel\\_task\\_barrier\\_init](#) (struct [starpu\\_task](#) \*task, int workerid)
- void [starpu\\_parallel\\_task\\_barrier\\_init\\_n](#) (struct [starpu\\_task](#) \*task, int worker\_size)

### 27.31.1 Detailed Description

### 27.31.2 Function Documentation

#### 27.31.2.1 int starpu\_combined\_worker\_get\_size ( void )

Return the size of the current combined worker, i.e. the total number of cpus running the same task in the case of [STARPU\\_SPMD](#) parallel tasks, or the total number of threads that the task is allowed to start in the case of [STARPU\\_FORKJOIN](#) parallel tasks.

#### 27.31.2.2 int starpu\_combined\_worker\_get\_rank ( void )

Return the rank of the current thread within the combined worker. Can only be used in [STARPU\\_FORKJOIN](#) parallel tasks, to know which part of the task to work on.

#### 27.31.2.3 unsigned starpu\_combined\_worker\_get\_count ( void )

Return the number of different combined workers.

#### 27.31.2.4 int starpu\_combined\_worker\_get\_id ( void )

Return the identifier of the current combined worker.

#### 27.31.2.5 int starpu\_combined\_worker\_assign\_workerid ( int nworkers, int workerid\_array[] )

Register a new combined worker and get its identifier

#### 27.31.2.6 int starpu\_combined\_worker\_get\_description ( int workerid, int \* worker\_size, int \*\* combined\_workerid )

Get the description of a combined worker

#### 27.31.2.7 int starpu\_combined\_worker\_can\_execute\_task ( unsigned workerid, struct starpu\_task \* task, unsigned nimpl )

Variant of [starpu\\_worker\\_can\\_execute\\_task\(\)](#) compatible with combined workers

#### 27.31.2.8 void starpu\_parallel\_task\_barrier\_init ( struct starpu\_task \* task, int workerid )

Initialise the barrier for the parallel task, and dispatch the task between the different workers of the given combined worker.

#### 27.31.2.9 void starpu\_parallel\_task\_barrier\_init\_n ( struct starpu\_task \* task, int worker\_size )

Initialise the barrier for the parallel task, to be pushed to *worker\_size* workers (without having to explicit a given combined worker).

## 27.32 Running Drivers

### Functions

- int [starpu\\_driver\\_run](#) (struct [starpu\\_driver](#) \*d)
- int [starpu\\_driver\\_init](#) (struct [starpu\\_driver](#) \*d)
- int [starpu\\_driver\\_run\\_once](#) (struct [starpu\\_driver](#) \*d)
- int [starpu\\_driver\\_deinit](#) (struct [starpu\\_driver](#) \*d)
- void [starpu\\_drivers\\_request\\_termination](#) (void)

### 27.32.1 Detailed Description

#### 27.32.2 Function Documentation

##### 27.32.2.1 int [starpu\\_driver\\_run](#) ( struct [starpu\\_driver](#) \* d )

Initialize the given driver, run it until it receives a request to terminate, deinitialize it and return 0 on success. It returns `-EINVAL` if `d->type` is not a valid StarPU device type ([STARPU\\_CPU\\_WORKER](#), [STARPU\\_CUDA\\_WORKER](#) or [STARPU\\_OPENCL\\_WORKER](#)). This is the same as using the following functions: calling [starpu\\_driver\\_init\(\)](#), then calling [starpu\\_driver\\_run\\_once\(\)](#) in a loop, and eventually [starpu\\_driver\\_deinit\(\)](#).

##### 27.32.2.2 int [starpu\\_driver\\_init](#) ( struct [starpu\\_driver](#) \* d )

Initialize the given driver. Returns 0 on success, `-EINVAL` if `d->type` is not a valid [starpu\\_worker\\_archtype](#).

##### 27.32.2.3 int [starpu\\_driver\\_run\\_once](#) ( struct [starpu\\_driver](#) \* d )

Run the driver once, then returns 0 on success, `-EINVAL` if `d->type` is not a valid [starpu\\_worker\\_archtype](#).

##### 27.32.2.4 int [starpu\\_driver\\_deinit](#) ( struct [starpu\\_driver](#) \* d )

Deinitialize the given driver. Returns 0 on success, `-EINVAL` if `d->type` is not a valid [starpu\\_worker\\_archtype](#).

##### 27.32.2.5 void [starpu\\_drivers\\_request\\_termination](#) ( void )

Notify all running drivers they should terminate.

## 27.33 Expert Mode

### Functions

- void [starpu\\_wake\\_all\\_blocked\\_workers](#) (void)
- int [starpu\\_progression\\_hook\\_register](#) (unsigned(\*func)(void \*arg), void \*arg)
- void [starpu\\_progression\\_hook\\_deregister](#) (int hook\_id)

### 27.33.1 Detailed Description

#### 27.33.2 Function Documentation

27.33.2.1 `void starpu_wake_all_blocked_workers ( void )`

Wake all the workers, so they can inspect data requests and task submissions again.

27.33.2.2 `int starpu_progression_hook_register ( unsigned(*) (void *arg) func, void * arg )`

Register a progression hook, to be called when workers are idle.

27.33.2.3 `void starpu_progression_hook_deregister ( int hook_id )`

Unregister a given progression hook.

## 27.34 StarPU-Top Interface

### Data Structures

- struct [starpu\\_top\\_data](#)
- struct [starpu\\_top\\_param](#)

### Enumerations

- enum [starpu\\_top\\_data\\_type](#) { STARPU\_TOP\_DATA\_BOOLEAN, STARPU\_TOP\_DATA\_INTEGER, STARPU\_TOP\_DATA\_FLOAT }
- enum [starpu\\_top\\_param\\_type](#) { STARPU\_TOP\_PARAM\_BOOLEAN, STARPU\_TOP\_PARAM\_INTEGER, STARPU\_TOP\_PARAM\_FLOAT, STARPU\_TOP\_PARAM\_ENUM }
- enum [starpu\\_top\\_message\\_type](#) { TOP\_TYPE\_GO, TOP\_TYPE\_SET, TOP\_TYPE\_CONTINUE, TOP\_TYPE\_ENABLE, TOP\_TYPE\_DISABLE, TOP\_TYPE\_DEBUG, TOP\_TYPE\_UNKNOW }

### Functions to call before the initialisation

- struct [starpu\\_top\\_data](#) \* [starpu\\_top\\_add\\_data\\_boolean](#) (const char \*data\_name, int active)
- struct [starpu\\_top\\_data](#) \* [starpu\\_top\\_add\\_data\\_integer](#) (const char \*data\_name, int minimum\_value, int maximum\_value, int active)
- struct [starpu\\_top\\_data](#) \* [starpu\\_top\\_add\\_data\\_float](#) (const char \*data\_name, double minimum\_value, double maximum\_value, int active)
- struct [starpu\\_top\\_param](#) \* [starpu\\_top\\_register\\_parameter\\_boolean](#) (const char \*param\_name, int \*parameter\_field, void(\*callback)(struct [starpu\\_top\\_param](#) \*))
- struct [starpu\\_top\\_param](#) \* [starpu\\_top\\_register\\_parameter\\_float](#) (const char \*param\_name, double \*parameter\_field, double minimum\_value, double maximum\_value, void(\*callback)(struct [starpu\\_top\\_param](#) \*))
- struct [starpu\\_top\\_param](#) \* [starpu\\_top\\_register\\_parameter\\_integer](#) (const char \*param\_name, int \*parameter\_field, int minimum\_value, int maximum\_value, void(\*callback)(struct [starpu\\_top\\_param](#) \*))
- struct [starpu\\_top\\_param](#) \* [starpu\\_top\\_register\\_parameter\\_enum](#) (const char \*param\_name, int \*parameter\_field, char \*\*values, int nb\_values, void(\*callback)(struct [starpu\\_top\\_param](#) \*))

### Initialisation

- void [starpu\\_top\\_init\\_and\\_wait](#) (const char \*server\_name)

### To call after initialisation

- void [starpu\\_top\\_update\\_parameter](#) (const struct [starpu\\_top\\_param](#) \*param)
- void [starpu\\_top\\_update\\_data\\_boolean](#) (const struct [starpu\\_top\\_data](#) \*data, int value)
- void [starpu\\_top\\_update\\_data\\_integer](#) (const struct [starpu\\_top\\_data](#) \*data, int value)
- void [starpu\\_top\\_update\\_data\\_float](#) (const struct [starpu\\_top\\_data](#) \*data, double value)
- void [starpu\\_top\\_task\\_prevision](#) (struct [starpu\\_task](#) \*task, int devid, unsigned long long start, unsigned long long end)
- void [starpu\\_top\\_debug\\_log](#) (const char \*message)
- void [starpu\\_top\\_debug\\_lock](#) (const char \*message)

## 27.34.1 Detailed Description

## 27.34.2 Data Structure Documentation

### 27.34.2.1 struct starpu\_top\_data

todo

#### Data Fields

|                                                 |                       |      |
|-------------------------------------------------|-----------------------|------|
| unsigned int                                    | id                    | todo |
| const char *                                    | name                  | todo |
| int                                             | int_min_value         | todo |
| int                                             | int_max_value         | todo |
| double                                          | double_min_↔<br>value | todo |
| double                                          | double_max_↔<br>value | todo |
| int                                             | active                | todo |
| enum <a href="#">starpu_↔<br/>top_data_type</a> | type                  | todo |
| struct <a href="#">starpu_top_data</a><br>*     | next                  | todo |

### 27.34.2.2 struct starpu\_top\_param

todo

#### Data Fields

- unsigned int [id](#)
- const char \* [name](#)
- enum [starpu\\_top\\_param\\_type](#) type
- void \* [value](#)
- char \*\* [enum\\_values](#)
- int [nb\\_values](#)
- void(\* [callback](#) )(struct [starpu\\_top\\_param](#) \*)
- int [int\\_min\\_value](#)
- int [int\\_max\\_value](#)
- double [double\\_min\\_value](#)
- double [double\\_max\\_value](#)
- struct [starpu\\_top\\_param](#) \* [next](#)

## 27.34.2.2.1 Field Documentation

27.34.2.2.1.1 unsigned int starpu\_top\_param::id

todo

27.34.2.2.1.2 const char \* starpu\_top\_param::name

todo

27.34.2.2.1.3 enum starpu\_top\_param\_type starpu\_top\_param::type

todo

27.34.2.2.1.4 void \* starpu\_top\_param::value

todo

27.34.2.2.1.5 char \*\* starpu\_top\_param::enum\_values

only for enum type can be NULL

27.34.2.2.1.6 int starpu\_top\_param::nb\_values

todo

27.34.2.2.1.7 void(\* starpu\_top\_param::callback)(struct starpu\_top\_param \*)

todo

27.34.2.2.1.8 int starpu\_top\_param::int\_min\_value

only for integer type

27.34.2.2.1.9 int starpu\_top\_param::int\_max\_value

todo

27.34.2.2.1.10 double starpu\_top\_param::double\_min\_value

only for double type

27.34.2.2.1.11 double starpu\_top\_param::double\_max\_value

todo

27.34.2.2.1.12 struct starpu\_top\_param \* starpu\_top\_param::next

todo

## 27.34.3 Enumeration Type Documentation

27.34.3.1 enum starpu\_top\_data\_type

StarPU-Top Data type

Enumerator

**STARPU\_TOP\_DATA\_BOOLEAN** todo**STARPU\_TOP\_DATA\_INTEGER** todo**STARPU\_TOP\_DATA\_FLOAT** todo

### 27.34.3.2 enum starpu\_top\_param\_type

StarPU-Top Parameter type

Enumerator

```
STARPU_TOP_PARAM_BOOLEAN todo
STARPU_TOP_PARAM_INTEGER todo
STARPU_TOP_PARAM_FLOAT todo
STARPU_TOP_PARAM_ENUM todo
```

### 27.34.3.3 enum starpu\_top\_message\_type

StarPU-Top Message type

Enumerator

```
TOP_TYPE_GO todo
TOP_TYPE_SET todo
TOP_TYPE_CONTINUE todo
TOP_TYPE_ENABLE todo
TOP_TYPE_DISABLE todo
TOP_TYPE_DEBUG todo
TOP_TYPE_UNKNOW todo
```

## 27.34.4 Function Documentation

### 27.34.4.1 struct starpu\_top\_data \* starpu\_top\_add\_data\_boolean ( const char \* *data\_name*, int *active* )

This fonction register a data named *data\_name* of type boolean. If *active*=0, the value will NOT be displayed to user by default. Any other value will make the value displayed by default.

### 27.34.4.2 struct starpu\_top\_data \* starpu\_top\_add\_data\_integer ( const char \* *data\_name*, int *minimum\_value*, int *maximum\_value*, int *active* )

This fonction register a data named *data\_name* of type integer. The minimum and maximum value will be usefull to define the scale in UI. If *active*=0, the value will NOT be displayed to user by default. Any other value will make the value displayed by default.

### 27.34.4.3 struct starpu\_top\_data \* starpu\_top\_add\_data\_float ( const char \* *data\_name*, double *minimum\_value*, double *maximum\_value*, int *active* )

This fonction register a data named *data\_name* of type float. The minimum and maximum value will be usefull to define the scale in UI. If *active*=0, the value will NOT be displayed to user by default. Any other value will make the value displayed by default.

### 27.34.4.4 struct starpu\_top\_param \* starpu\_top\_register\_parameter\_boolean ( const char \* *param\_name*, int \* *parameter\_field*, void(\*) (struct starpu\_top\_param \*) *callback* )

This fonction register a parameter named *parameter\_name*, of type boolean. The *callback* fonction will be called when the parameter is modified by UI, and can be null.



**27.34.4.5** `struct starpu_top_param * starpu_top_register_parameter_float ( const char * param_name, double * parameter_field, double minimum_value, double maximum_value, void(*) (struct starpu_top_param *) callback )`

his fonction register a parameter named `param_name`, of type integer. Minimum and maximum value will be used to prevent user seting incorrect value. The `callback` fonction will be called when the parameter is modified by UI, and can be null.

**27.34.4.6** `struct starpu_top_param * starpu_top_register_parameter_integer ( const char * param_name, int * parameter_field, int minimum_value, int maximum_value, void(*) (struct starpu_top_param *) callback )`

This fonction register a parameter named `param_name`, of type float. Minimum and maximum value will be used to prevent user seting incorrect value. The `callback` fonction will be called when the parameter is modified by UI, and can be null.

**27.34.4.7** `struct starpu_top_param * starpu_top_register_parameter_enum ( const char * param_name, int * parameter_field, char ** values, int nb_values, void(*) (struct starpu_top_param *) callback )`

This fonction register a parameter named `param_name`, of type enum. Minimum and maximum value will be used to prevent user seting incorrect value. The `callback` fonction will be called when the parameter is modified by UI, and can be null.

**27.34.4.8** `void starpu_top_init_and_wait ( const char * server_name )`

This function must be called when all parameters and data have been registered AND initialised (for parameters). This function will wait for a TOP to connect, send initialisation sentences, and wait for the GO message.

**27.34.4.9** `void starpu_top_update_parameter ( const struct starpu_top_param * param )`

This function should be called after every modification of a parameter from something other than `starpu_top`. This fonction notice UI that the configuration changed.

**27.34.4.10** `void starpu_top_update_data_boolean ( const struct starpu_top_data * data, int value )`

This function updates the value of the [starpu\\_top\\_data](#) on UI.

**27.34.4.11** `void starpu_top_update_data_integer ( const struct starpu_top_data * data, int value )`

This function updates the value of the [starpu\\_top\\_data](#) on UI.

**27.34.4.12** `void starpu_top_update_data_float ( const struct starpu_top_data * data, double value )`

This function updates the value of the [starpu\\_top\\_data](#) on UI.

**27.34.4.13** `void starpu_top_task_prevision ( struct starpu_task * task, int devid, unsigned long long start, unsigned long long end )`

This function notifies UI than the task have been planed to run from start to end, on computation-core.

**27.34.4.14** `void starpu_top_debug_log ( const char * message )`

This function is useful in debug mode. The starpu developper doesn't need to check if the debug mode is active. This is checked by `starpu_top` itsefl. It just send a message to display by UI.

27.34.4.15 `void starpu_top_debug_lock ( const char * message )`

This function is useful in debug mode. The starpu developer doesn't need to check if the debug mode is active. This is checked by `starpu_top` itself. It send a message and wait for a continue message from UI to return. The lock (which create a stop-point) should be called only by the main thread. Calling it from more than one thread is not supported.

## 27.35 Scheduling Contexts

StarPU permits on one hand grouping workers in combined workers in order to execute a parallel task and on the other hand grouping tasks in bundles that will be executed by a single specified worker. In contrast when we group workers in scheduling contexts we submit starpu tasks to them and we schedule them with the policy assigned to the context. Scheduling contexts can be created, deleted and modified dynamically.

### Data Structures

- struct [starpu\\_sched\\_ctx\\_performance\\_counters](#)

### Scheduling Contexts Basic API

- `#define STARPU_SCHED_CTX_POLICY_NAME`
- `#define STARPU_SCHED_CTX_POLICY_STRUCT`
- `#define STARPU_SCHED_CTX_POLICY_MIN_PRIO`
- `#define STARPU_SCHED_CTX_POLICY_MAX_PRIO`
- `unsigned starpu_sched_ctx_create (int *workerids_ctx, int nworkers_ctx, const char *sched_ctx_name,...)`
- `unsigned starpu_sched_ctx_create_inside_interval (const char *policy_name, const char *sched_ctx_name, int min_ncpus, int max_ncpus, int min_ngpus, int max_ngpus, unsigned allow_overlap)`
- `void starpu_sched_ctx_register_close_callback (unsigned sched_ctx_id, void(*close_callback)(unsigned sched_ctx_id, void *args), void *args)`
- `void starpu_sched_ctx_add_workers (int *workerids_ctx, int nworkers_ctx, unsigned sched_ctx_id)`
- `void starpu_sched_ctx_remove_workers (int *workerids_ctx, int nworkers_ctx, unsigned sched_ctx_id)`
- `void starpu_sched_ctx_display_workers (unsigned sched_ctx_id, FILE *f)`
- `void starpu_sched_ctx_delete (unsigned sched_ctx_id)`
- `void starpu_sched_ctx_set_inheritor (unsigned sched_ctx_id, unsigned inheritor)`
- `void starpu_sched_ctx_set_context (unsigned *sched_ctx_id)`
- `unsigned starpu_sched_ctx_get_context (void)`
- `void starpu_sched_ctx_stop_task_submission (void)`
- `void starpu_sched_ctx_finished_submit (unsigned sched_ctx_id)`
- `unsigned starpu_sched_ctx_get_workers_list (unsigned sched_ctx_id, int **workerids)`
- `unsigned starpu_sched_ctx_get_nworkers (unsigned sched_ctx_id)`
- `unsigned starpu_sched_ctx_get_nshared_workers (unsigned sched_ctx_id, unsigned sched_ctx_id2)`
- `unsigned starpu_sched_ctx_contains_worker (int workerid, unsigned sched_ctx_id)`
- `unsigned starpu_sched_ctx_worker_get_id (unsigned sched_ctx_id)`
- `unsigned starpu_sched_ctx_overlapping_ctxs_on_worker (int workerid)`

### Scheduling Context Priorities

- `#define STARPU_MIN_PRIO`
- `#define STARPU_MAX_PRIO`
- `#define STARPU_DEFAULT_PRIO`
- `int starpu_sched_ctx_set_min_priority (unsigned sched_ctx_id, int min_prio)`
- `int starpu_sched_ctx_set_max_priority (unsigned sched_ctx_id, int max_prio)`

- int [starpu\\_sched\\_ctx\\_get\\_min\\_priority](#) (unsigned sched\_ctx\_id)
- int [starpu\\_sched\\_ctx\\_get\\_max\\_priority](#) (unsigned sched\_ctx\_id)
- int [starpu\\_sched\\_ctx\\_min\\_priority\\_is\\_set](#) (unsigned sched\_ctx\_id)
- int [starpu\\_sched\\_ctx\\_max\\_priority\\_is\\_set](#) (unsigned sched\_ctx\_id)

### Scheduling Context Worker Collection

- struct [starpu\\_worker\\_collection](#) \* [starpu\\_sched\\_ctx\\_create\\_worker\\_collection](#) (unsigned sched\_ctx\_id, enum [starpu\\_worker\\_collection\\_type](#) type)
- void [starpu\\_sched\\_ctx\\_delete\\_worker\\_collection](#) (unsigned sched\_ctx\_id)
- struct [starpu\\_worker\\_collection](#) \* [starpu\\_sched\\_ctx\\_get\\_worker\\_collection](#) (unsigned sched\_ctx\_id)

### Scheduling Context Link with Hypervisor

- void [starpu\\_sched\\_ctx\\_set\\_perf\\_counters](#) (unsigned sched\_ctx\_id, void \*perf\_counters)
- void [starpu\\_sched\\_ctx\\_call\\_pushed\\_task\\_cb](#) (int workerid, unsigned sched\_ctx\_id)
- void [starpu\\_sched\\_ctx\\_notify\\_hypervisor\\_exists](#) (void)
- unsigned [starpu\\_sched\\_ctx\\_check\\_if\\_hypervisor\\_exists](#) (void)
- void [starpu\\_sched\\_ctx\\_set\\_policy\\_data](#) (unsigned sched\_ctx\_id, void \*policy\_data)
- void \* [starpu\\_sched\\_ctx\\_get\\_policy\\_data](#) (unsigned sched\_ctx\_id)
- void \* [starpu\\_sched\\_ctx\\_exec\\_parallel\\_code](#) (void \*(\*func)(void \*), void \*param, unsigned sched\_ctx\_id)
- int [starpu\\_sched\\_ctx\\_get\\_nready\\_tasks](#) (unsigned sched\_ctx\_id)
- double [starpu\\_sched\\_ctx\\_get\\_nready\\_flops](#) (unsigned sched\_ctx\_id)

#### 27.35.1 Detailed Description

StarPU permits on one hand grouping workers in combined workers in order to execute a parallel task and on the other hand grouping tasks in bundles that will be executed by a single specified worker. In contrast when we group workers in scheduling contexts we submit starpu tasks to them and we schedule them with the policy assigned to the context. Scheduling contexts can be created, deleted and modified dynamically.

#### 27.35.2 Data Structure Documentation

##### 27.35.2.1 struct starpu\_sched\_ctx\_performance\_counters

Performance counters used by the starpu to indicate the hypervisor how the application and the resources are executing.

##### Data Fields

- void(\* [notify\\_idle\\_cycle](#) )(unsigned sched\_ctx\_id, int worker, double idle\_time)
- void(\* [notify\\_poped\\_task](#) )(unsigned sched\_ctx\_id, int worker)
- void(\* [notify\\_pushed\\_task](#) )(unsigned sched\_ctx\_id, int worker)
- void(\* [notify\\_post\\_exec\\_task](#) )(struct [starpu\\_task](#) \*task, size\_t data\_size, uint32\_t footprint, int hypervisor\_tag, double flops)
- void(\* [notify\\_submitted\\_job](#) )(struct [starpu\\_task](#) \*task, uint32\_t footprint, size\_t data\_size)
- void(\* [notify\\_empty\\_ctx](#) )(unsigned sched\_ctx\_id, struct [starpu\\_task](#) \*task)
- void(\* [notify\\_delete\\_context](#) )(unsigned sched\_ctx)

### 27.35.2.1.1 Field Documentation

**27.35.2.1.1.1** `void(* starpu_sched_ctx_performance_counters::notify_idle_cycle)(unsigned sched_ctx_id, int worker, double idle_time)`

Informes the hypervisor for how long a worker has been idle in the specified context

**27.35.2.1.1.2** `void(* starpu_sched_ctx_performance_counters::notify_poped_task)(unsigned sched_ctx_id, int worker)`

Informes the hypervisor that a task executing a specified number of instructions has been popped from the worker

**27.35.2.1.1.3** `void(* starpu_sched_ctx_performance_counters::notify_pushed_task)(unsigned sched_ctx_id, int worker)`

Notifies the hypervisor that a task has been scheduled on the queue of the worker corresponding to the specified context

**27.35.2.1.1.4** `void(* starpu_sched_ctx_performance_counters::notify_post_exec_task)(struct starpu_task *task, size_t data_size, uint32_t footprint, int hypervisor_tag, double flops)`

Notifies the hypervisor that a task has just been executed

**27.35.2.1.1.5** `void(* starpu_sched_ctx_performance_counters::notify_submitted_job)(struct starpu_task *task, uint32_t footprint, size_t data_size)`

Notifies the hypervisor that a task has just been submitted

**27.35.2.1.1.6** `void(* starpu_sched_ctx_performance_counters::notify_delete_context)(unsigned sched_ctx)`

Notifies the hypervisor that the context was deleted

### 27.35.3 Macro Definition Documentation

**27.35.3.1** `#define STARPU_SCHED_CTX_POLICY_NAME`

This macro is used when calling [starpu\\_sched\\_ctx\\_create\(\)](#) to specify a name for a scheduling policy

**27.35.3.2** `#define STARPU_SCHED_CTX_POLICY_STRUCT`

This macro is used when calling [starpu\\_sched\\_ctx\\_create\(\)](#) to specify a pointer to a scheduling policy

**27.35.3.3** `#define STARPU_SCHED_CTX_POLICY_MIN_PRIO`

This macro is used when calling [starpu\\_sched\\_ctx\\_create\(\)](#) to specify a minimum scheduler priority value.

**27.35.3.4** `#define STARPU_SCHED_CTX_POLICY_MAX_PRIO`

This macro is used when calling [starpu\\_sched\\_ctx\\_create\(\)](#) to specify a maximum scheduler priority value.

**27.35.3.5** `#define STARPU_MIN_PRIO`

Provided for legacy reasons.

**27.35.3.6** `#define STARPU_MAX_PRIO`

Provided for legacy reasons.

27.35.3.7 `#define STARPU_DEFAULT_PRIO`

By convention, the default priority level should be 0 so that we can statically allocate tasks with a default priority.

## 27.35.4 Function Documentation

27.35.4.1 `unsigned starpu_sched_ctx_create ( int * workerids_ctx, int nworkers_ctx, const char * sched_ctx_name, ... )`

This function creates a scheduling context with the given parameters (see below) and assigns the workers in `workerids_ctx` to execute the tasks submitted to it. The return value represents the identifier of the context that has just been created. It will be further used to indicate the context the tasks will be submitted to. The return value should be at most `STARPU_NMAX_SCHED_CTXS`.

The arguments following the name of the scheduling context can be of the following types:

- `STARPU_SCHED_CTX_POLICY_NAME`, followed by the name of a predefined scheduling policy
- `STARPU_SCHED_CTX_POLICY_STRUCT`, followed by a pointer to a custom scheduling policy (struct `starpu_sched_policy *`)
- `STARPU_SCHED_CTX_POLICY_MIN_PRIO`, followed by a integer representing the minimum priority value to be defined for the scheduling policy.
- `STARPU_SCHED_CTX_POLICY_MAX_PRIO`, followed by a integer representing the maximum priority value to be defined for the scheduling policy.

27.35.4.2 `unsigned starpu_sched_ctx_create_inside_interval ( const char * policy_name, const char * sched_ctx_name, int min_ncpus, int max_ncpus, int min_ngpus, int max_ngpus, unsigned allow_overlap )`

Create a context indicating an approximate interval of resources

27.35.4.3 `void starpu_sched_ctx_register_close_callback ( unsigned sched_ctx_id, void (*)(unsigned sched_ctx_id, void *args) close_callback, void * args )`

Execute the callback whenever the last task of the context finished executing, it is called with the parameters: `sched_ctx_id` and any other parameter needed by the application (packed in a `void *`)

27.35.4.4 `void starpu_sched_ctx_add_workers ( int * workerids_ctx, int nworkers_ctx, unsigned sched_ctx_id )`

This function adds dynamically the workers in `workerids_ctx` to the context `sched_ctx_id`. The last argument cannot be greater than `STARPU_NMAX_SCHED_CTXS`.

27.35.4.5 `void starpu_sched_ctx_remove_workers ( int * workerids_ctx, int nworkers_ctx, unsigned sched_ctx_id )`

This function removes the workers in `workerids_ctx` from the context `sched_ctx_id`. The last argument cannot be greater than `STARPU_NMAX_SCHED_CTXS`.

27.35.4.6 `void starpu_sched_ctx_display_workers ( unsigned sched_ctx_id, FILE * f )`

This function prints on the file `f` the worker names belonging to the context `sched_ctx_id`

27.35.4.7 `void starpu_sched_ctx_delete ( unsigned sched_ctx_id )`

Delete scheduling context `sched_ctx_id` and transfer remaining workers to the inheritor scheduling context.

27.35.4.8 `void starpu_sched_ctx_set_inheritor ( unsigned sched_ctx_id, unsigned inheritor )`

Indicate which context will inherit the resources of this context when he will be deleted.

27.35.4.9 `void starpu_sched_ctx_set_context ( unsigned * sched_ctx_id )`

Set the scheduling context the subsequent tasks will be submitted to

27.35.4.10 `unsigned starpu_sched_ctx_get_context ( void )`

Return the scheduling context the tasks are currently submitted to, or `::STARPU_NMAX_SCHED_CTXS` if no default context has been defined by calling the function [starpu\\_sched\\_ctx\\_set\\_context\(\)](#).

27.35.4.11 `void starpu_sched_ctx_stop_task_submission ( void )`

Stop submitting tasks from the empty context list until the next time the context has time to check the empty context list

27.35.4.12 `void starpu_sched_ctx_finished_submit ( unsigned sched_ctx_id )`

Indicate starpu that the application finished submitting to this context in order to move the workers to the inheritor as soon as possible.

27.35.4.13 `unsigned starpu_sched_ctx_get_workers_list ( unsigned sched_ctx_id, int ** workerids )`

Returns the list of workers in the array `workerids`, the returned value is the number of workers. The user should free the `workerids` table after finishing using it (it is allocated inside the function with the proper size)

27.35.4.14 `unsigned starpu_sched_ctx_get_nworkers ( unsigned sched_ctx_id )`

Return the number of workers managed by the specified contexts (Usually needed to verify if it manages any workers or if it should be blocked)

27.35.4.15 `unsigned starpu_sched_ctx_get_nshared_workers ( unsigned sched_ctx_id, unsigned sched_ctx_id2 )`

Return the number of workers shared by two contexts.

27.35.4.16 `unsigned starpu_sched_ctx_contains_worker ( int workerid, unsigned sched_ctx_id )`

Return 1 if the worker belongs to the context and 0 otherwise

27.35.4.17 `unsigned starpu_sched_ctx_worker_get_id ( unsigned sched_ctx_id )`

Return the workerid if the worker belongs to the context and -1 otherwise. If the thread calling this function is not a worker the function returns -1 as it calls the function [starpu\\_worker\\_get\\_id\(\)](#)

27.35.4.18 `unsigned starpu_sched_ctx_overlapping_ctxs_on_worker ( int workerid )`

Check if a worker is shared between several contexts

**27.35.4.19** `int starpu_sched_ctx_set_min_priority ( unsigned sched_ctx_id, int min_prio )`

Defines the minimum task priority level supported by the scheduling policy of the given scheduler context. The default minimum priority level is the same as the default priority level which is 0 by convention. The application may access that value by calling the function `starpu_sched_ctx_get_min_priority()`. This function should only be called from the initialization method of the scheduling policy, and should not be used directly from the application.

**27.35.4.20** `int starpu_sched_ctx_set_max_priority ( unsigned sched_ctx_id, int max_prio )`

Defines the maximum priority level supported by the scheduling policy of the given scheduler context. The default maximum priority level is 1. The application may access that value by calling the `starpu_sched_ctx_get_max_priority` function. This function should only be called from the initialization method of the scheduling policy, and should not be used directly from the application.

**27.35.4.21** `int starpu_sched_ctx_get_min_priority ( unsigned sched_ctx_id )`

Returns the current minimum priority level supported by the scheduling policy of the given scheduler context.

**27.35.4.22** `int starpu_sched_ctx_get_max_priority ( unsigned sched_ctx_id )`

Returns the current maximum priority level supported by the scheduling policy of the given scheduler context.

**27.35.4.23** `int starpu_sched_ctx_min_priority_is_set ( unsigned sched_ctx_id )`

todo

**27.35.4.24** `int starpu_sched_ctx_max_priority_is_set ( unsigned sched_ctx_id )`

todo

**27.35.4.25** `struct starpu_worker_collection * starpu_sched_ctx_create_worker_collection ( unsigned sched_ctx_id, enum starpu_worker_collection_type type )`

Create a worker collection of the type indicated by the last parameter for the context specified through the first parameter.

**27.35.4.26** `void starpu_sched_ctx_delete_worker_collection ( unsigned sched_ctx_id )`

Delete the worker collection of the specified scheduling context

**27.35.4.27** `struct starpu_worker_collection * starpu_sched_ctx_get_worker_collection ( unsigned sched_ctx_id )`

Return the worker collection managed by the indicated context

**27.35.4.28** `void starpu_sched_ctx_set_perf_counters ( unsigned sched_ctx_id, void * perf_counters )`

Indicates to starpu the pointer to the performance counter

**27.35.4.29** `void starpu_sched_ctx_call_pushed_task_cb ( int workerid, unsigned sched_ctx_id )`

Callback that lets the scheduling policy tell the hypervisor that a task was pushed on a worker

27.35.4.30 `void starpu_sched_ctx_notify_hypervisor_exists ( void )`

Allow the hypervisor to let starpu know he's initialised

27.35.4.31 `unsigned starpu_sched_ctx_check_if_hypervisor_exists ( void )`

Ask starpu if he is informed if the hypervisor is initialised

27.35.4.32 `void starpu_sched_ctx_set_policy_data ( unsigned sched_ctx_id, void * policy_data )`

Allocate the scheduling policy data (private information of the scheduler like queues, variables, additional condition variables) the context

27.35.4.33 `void * starpu_sched_ctx_get_policy_data ( unsigned sched_ctx_id )`

Return the scheduling policy data (private information of the scheduler) of the contexts previously assigned to.

27.35.4.34 `void * starpu_sched_ctx_exec_parallel_code ( void (*)(void *) func, void * param, unsigned sched_ctx_id )`

execute any parallel code on the workers of the sched\_ctx (workers are blocked)

27.35.4.35 `int starpu_sched_ctx_get_nready_tasks ( unsigned sched_ctx_id )`

todo

27.35.4.36 `double starpu_sched_ctx_get_nready_flops ( unsigned sched_ctx_id )`

todo

## 27.36 Scheduling Policy

TODO. While StarPU comes with a variety of scheduling policies (see [Task Scheduling Policy](#)), it may sometimes be desirable to implement custom policies to address specific problems. The API described below allows users to write their own scheduling policy.

### Data Structures

- struct [starpu\\_sched\\_policy](#)

### Functions

- struct [starpu\\_sched\\_policy](#) \*\* [starpu\\_sched\\_get\\_predefined\\_policies](#) ()
- void [starpu\\_worker\\_get\\_sched\\_condition](#) (int workerid, starpu\_pthread\_mutex\_t \*\*sched\_mutex, starpu\_↔ pthread\_cond\_t \*\*sched\_cond)
- int [starpu\\_sched\\_set\\_min\\_priority](#) (int min\_prio)
- int [starpu\\_sched\\_set\\_max\\_priority](#) (int max\_prio)
- int [starpu\\_sched\\_get\\_min\\_priority](#) (void)
- int [starpu\\_sched\\_get\\_max\\_priority](#) (void)
- int [starpu\\_push\\_local\\_task](#) (int workerid, struct [starpu\\_task](#) \*task, int back)
- int [starpu\\_push\\_task\\_end](#) (struct [starpu\\_task](#) \*task)



- int [starpu\\_worker\\_can\\_execute\\_task](#) (unsigned workerid, struct [starpu\\_task](#) \*task, unsigned nimpl)
- int [starpu\\_worker\\_can\\_execute\\_task\\_impl](#) (unsigned workerid, struct [starpu\\_task](#) \*task, unsigned \*impl\_↔ mask)
- int [starpu\\_worker\\_can\\_execute\\_task\\_first\\_impl](#) (unsigned workerid, struct [starpu\\_task](#) \*task, unsigned \*nimpl)
- uint32\_t [starpu\\_task\\_footprint](#) (struct [starpu\\_perfmodel](#) \*model, struct [starpu\\_task](#) \*task, struct [starpu\\_↔ perfmodel\\_arch](#) \*arch, unsigned nimpl)
- uint32\_t [starpu\\_task\\_data\\_footprint](#) (struct [starpu\\_task](#) \*task)
- double [starpu\\_task\\_expected\\_length](#) (struct [starpu\\_task](#) \*task, struct [starpu\\_perfmodel\\_arch](#) \*arch, unsigned nimpl)
- double [starpu\\_worker\\_get\\_relative\\_speedup](#) (struct [starpu\\_perfmodel\\_arch](#) \*perf\_arch)
- double [starpu\\_task\\_expected\\_data\\_transfer\\_time](#) (unsigned memory\_node, struct [starpu\\_task](#) \*task)
- double [starpu\\_data\\_expected\\_transfer\\_time](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned memory\_node, enum [starpu\\_data\\_access\\_mode](#) mode)
- double [starpu\\_task\\_expected\\_power](#) (struct [starpu\\_task](#) \*task, struct [starpu\\_perfmodel\\_arch](#) \*arch, unsigned nimpl)
- double [starpu\\_task\\_expected\\_conversion\\_time](#) (struct [starpu\\_task](#) \*task, struct [starpu\\_perfmodel\\_arch](#) \*arch, unsigned nimpl)
- int [starpu\\_get\\_prefetch\\_flag](#) (void)
- int [starpu\\_prefetch\\_task\\_input\\_on\\_node](#) (struct [starpu\\_task](#) \*task, unsigned node)
- void [starpu\\_sched\\_ctx\\_worker\\_shares\\_tasks\\_lists](#) (int workerid, int sched\_ctx\_id)

### 27.36.1 Detailed Description

TODO. While StarPU comes with a variety of scheduling policies (see [Task Scheduling Policy](#)), it may sometimes be desirable to implement custom policies to address specific problems. The API described below allows users to write their own scheduling policy.

### 27.36.2 Data Structure Documentation

#### 27.36.2.1 struct [starpu\\_sched\\_policy](#)

This structure contains all the methods that implement a scheduling policy. An application may specify which scheduling strategy in the field [starpu\\_conf::sched\\_policy](#) passed to the function [starpu\\_init\(\)](#).

#### Data Fields

- void(\* [init\\_sched](#) )(unsigned sched\_ctx\_id)
- void(\* [deinit\\_sched](#) )(unsigned sched\_ctx\_id)
- int(\* [push\\_task](#) )(struct [starpu\\_task](#) \*)
- void(\* [push\\_task\\_notify](#) )(struct [starpu\\_task](#) \*, int workerid, int perf\_workerid, unsigned sched\_ctx\_id)
- struct [starpu\\_task](#) \*(\* [pop\\_task](#) )(unsigned sched\_ctx\_id)
- struct [starpu\\_task](#) \*(\* [pop\\_every\\_task](#) )(unsigned sched\_ctx\_id)
- void(\* [pre\\_exec\\_hook](#) )(struct [starpu\\_task](#) \*)
- void(\* [post\\_exec\\_hook](#) )(struct [starpu\\_task](#) \*)
- void(\* [add\\_workers](#) )(unsigned sched\_ctx\_id, int \*workerids, unsigned nworkers)
- void(\* [remove\\_workers](#) )(unsigned sched\_ctx\_id, int \*workerids, unsigned nworkers)
- const char \* [policy\\_name](#)
- const char \* [policy\\_description](#)

#### 27.36.2.1.1 Field Documentation

##### 27.36.2.1.1.1 void(\* [starpu\\_sched\\_policy::init\\_sched](#))(unsigned sched\_ctx\_id)

Initialize the scheduling policy.

**27.36.2.1.1.2** `void(* starpu_sched_policy::deinit_sched)(unsigned sched_ctx_id)`

Cleanup the scheduling policy.

**27.36.2.1.1.3** `int(* starpu_sched_policy::push_task)(struct starpu_task *)`

Insert a task into the scheduler.

**27.36.2.1.1.4** `void(* starpu_sched_policy::push_task_notify)(struct starpu_task *, int workerid, int perf_workerid, unsigned sched_ctx_id)`

Notify the scheduler that a task was pushed on a given worker. This method is called when a task that was explicitly assigned to a worker becomes ready and is about to be executed by the worker. This method therefore permits to keep the state of the scheduler coherent even when StarPU bypasses the scheduling strategy.

**27.36.2.1.1.5** `struct starpu_task *(* starpu_sched_policy::pop_task)(unsigned sched_ctx_id)`

Get a task from the scheduler. The mutex associated to the worker is already taken when this method is called. If this method is defined as NULL, the worker will only execute tasks from its local queue. In this case, the `push_task` method should use the `starpu_push_local_task` method to assign tasks to the different workers.

**27.36.2.1.1.6** `struct starpu_task *(* starpu_sched_policy::pop_every_task)(unsigned sched_ctx_id)`

Remove all available tasks from the scheduler (tasks are chained by the means of the field `starpu_task::prev` and `starpu_task::next`). The mutex associated to the worker is already taken when this method is called. This is currently not used.

**27.36.2.1.1.7** `void(* starpu_sched_policy::pre_exec_hook)(struct starpu_task *)`

Optional field. This method is called every time a task is starting.

**27.36.2.1.1.8** `void(* starpu_sched_policy::post_exec_hook)(struct starpu_task *)`

Optional field. This method is called every time a task has been executed.

**27.36.2.1.1.9** `void(* starpu_sched_policy::add_workers)(unsigned sched_ctx_id, int *workerids, unsigned nworkers)`

Initialize scheduling structures corresponding to each worker used by the policy.

**27.36.2.1.1.10** `void(* starpu_sched_policy::remove_workers)(unsigned sched_ctx_id, int *workerids, unsigned nworkers)`

Deinitialize scheduling structures corresponding to each worker used by the policy.

**27.36.2.1.1.11** `const char * starpu_sched_policy::policy_name`

Optional field. Name of the policy.

**27.36.2.1.1.12** `const char * starpu_sched_policy::policy_description`

Optional field. Human readable description of the policy.

## 27.36.3 Function Documentation

**27.36.3.1** `struct starpu_sched_policy ** starpu_sched_get_predefined_policies ( )`

Return an NULL-terminated array of all the predefined scheduling policies.

27.36.3.2 void starpu\_worker\_get\_sched\_condition ( int *workerid*, starpu\_pthread\_mutex\_t \*\* *sched\_mutex*, starpu\_pthread\_cond\_t \*\* *sched\_cond* )

When there is no available task for a worker, StarPU blocks this worker on a condition variable. This function specifies which condition variable (and the associated mutex) should be used to block (and to wake up) a worker. Note that multiple workers may use the same condition variable. For instance, in the case of a scheduling strategy with a single task queue, the same condition variable would be used to block and wake up all workers.

27.36.3.3 int starpu\_sched\_set\_min\_priority ( int *min\_prio* )

TODO: check if this is correct Defines the minimum task priority level supported by the scheduling policy. The default minimum priority level is the same as the default priority level which is 0 by convention. The application may access that value by calling the function [starpu\\_sched\\_get\\_min\\_priority\(\)](#). This function should only be called from the initialization method of the scheduling policy, and should not be used directly from the application.

27.36.3.4 int starpu\_sched\_set\_max\_priority ( int *max\_prio* )

TODO: check if this is correct Defines the maximum priority level supported by the scheduling policy. The default maximum priority level is 1. The application may access that value by calling the function [starpu\\_sched\\_get\\_max\\_priority\(\)](#). This function should only be called from the initialization method of the scheduling policy, and should not be used directly from the application.

27.36.3.5 int starpu\_sched\_get\_min\_priority ( void )

TODO: check if this is correct Returns the current minimum priority level supported by the scheduling policy

27.36.3.6 int starpu\_sched\_get\_max\_priority ( void )

TODO: check if this is correct Returns the current maximum priority level supported by the scheduling policy

27.36.3.7 int starpu\_push\_local\_task ( int *workerid*, struct starpu\_task \* *task*, int *back* )

The scheduling policy may put tasks directly into a worker's local queue so that it is not always necessary to create its own queue when the local queue is sufficient. If *back* is not 0, *task* is put at the back of the queue where the worker will pop tasks first. Setting *back* to 0 therefore ensures a FIFO ordering.

27.36.3.8 int starpu\_push\_task\_end ( struct starpu\_task \* *task* )

This function must be called by a scheduler to notify that the given task has just been pushed.

27.36.3.9 int starpu\_worker\_can\_execute\_task ( unsigned *workerid*, struct starpu\_task \* *task*, unsigned *nimpl* )

Check if the worker specified by *workerid* can execute the codelet. Schedulers need to call it before assigning a task to a worker, otherwise the task may fail to execute.

27.36.3.10 int starpu\_worker\_can\_execute\_task\_impl ( unsigned *workerid*, struct starpu\_task \* *task*, unsigned \* *impl\_mask* )

Check if the worker specified by *workerid* can execute the codelet and returns which implementation numbers can be used. Schedulers need to call it before assigning a task to a worker, otherwise the task may fail to execute. This should be preferred rather than calling `starpu_worker_can_execute_task` for each and every implementation. It can also be used with `impl_mask == NULL` to check for at least one implementation without determining which.

27.36.3.11 `int starpu_worker_can_execute_task_first_impl ( unsigned workerid, struct starpu_task * task, unsigned * nimpl )`

Check if the worker specified by *workerid* can execute the codelet and returns the first implementation which can be used. Schedulers need to call it before assigning a task to a worker, otherwise the task may fail to execute. This should be preferred rather than calling `starpu_worker_can_execute_task` for each and every implementation. It can also be used with `impl_mask == NULL` to check for at least one implementation without determining which.

27.36.3.12 `uint32_t starpu_task_footprint ( struct starpu_perfmodel * model, struct starpu_task * task, struct starpu_perfmodel_arch * arch, unsigned nimpl )`

Returns the footprint for a given task, taking into account user-provided perfmodel footprint or `size_base` functions.

27.36.3.13 `uint32_t starpu_task_data_footprint ( struct starpu_task * task )`

Returns the raw footprint for the data of a given task (without taking into account user-provided functions).

27.36.3.14 `double starpu_task_expected_length ( struct starpu_task * task, struct starpu_perfmodel_arch * arch, unsigned nimpl )`

Returns expected task duration in micro-seconds.

27.36.3.15 `double starpu_worker_get_relative_speedup ( struct starpu_perfmodel_arch * perf_arch )`

Returns an estimated speedup factor relative to CPU speed

27.36.3.16 `double starpu_task_expected_data_transfer_time ( unsigned memory_node, struct starpu_task * task )`

Returns expected data transfer time in micro-seconds.

27.36.3.17 `double starpu_data_expected_transfer_time ( starpu_data_handle_t handle, unsigned memory_node, enum starpu_data_access_mode mode )`

Predict the transfer time (in micro-seconds) to move *handle* to a memory node

27.36.3.18 `double starpu_task_expected_power ( struct starpu_task * task, struct starpu_perfmodel_arch * arch, unsigned nimpl )`

Returns expected power consumption in J

27.36.3.19 `double starpu_task_expected_conversion_time ( struct starpu_task * task, struct starpu_perfmodel_arch * arch, unsigned nimpl )`

Returns expected conversion time in ms (multiformat interface only)

27.36.3.20 `int starpu_get_prefetch_flag ( void )`

Whether [STARPU\\_PREFETCH](#) was set

27.36.3.21 `int starpu_prefetch_task_input_on_node ( struct starpu_task * task, unsigned node )`

Prefetch data for a given task on a given node

27.36.3.22 `void starpu_sched_ctx_worker_shares_tasks_lists ( int workerid, int sched_ctx_id )`

The scheduling policies indicates if the worker may pop tasks from the list of other workers or if there is a central list with task for all the workers

## 27.37 Tree

This section describes the tree facilities provided by StarPU.

### Data Structures

- struct [starpu\\_tree](#)

### Functions

- void [starpu\\_tree\\_reset\\_visited](#) (struct [starpu\\_tree](#) \*tree, int \*visited)
- void [starpu\\_tree\\_insert](#) (struct [starpu\\_tree](#) \*tree, int id, int level, int is\_pu, int arity, struct [starpu\\_tree](#) \*father)
- struct [starpu\\_tree](#) \* [starpu\\_tree\\_get](#) (struct [starpu\\_tree](#) \*tree, int id)
- struct [starpu\\_tree](#) \* [starpu\\_tree\\_get\\_neighbour](#) (struct [starpu\\_tree](#) \*tree, struct [starpu\\_tree](#) \*node, int \*visited, int \*present)
- void [starpu\\_tree\\_free](#) (struct [starpu\\_tree](#) \*tree)

### 27.37.1 Detailed Description

This section describes the tree facilities provided by StarPU.

### 27.37.2 Data Structure Documentation

27.37.2.1 `struct starpu_tree`

#### Data Fields

|                                       |        |      |
|---------------------------------------|--------|------|
| struct <a href="#">starpu_tree</a> ** | nodes  | todo |
| struct <a href="#">starpu_tree</a> *  | father | todo |
| int                                   | arity  | todo |
| int                                   | id     | todo |
| int                                   | level  | todo |
| int                                   | is_pu  | todo |

### 27.37.3 Function Documentation

27.37.3.1 `void starpu_tree_reset_visited ( struct starpu_tree * tree, int * visited )`

todo

27.37.3.2 void starpu\_tree\_insert ( struct starpu\_tree \* tree, int id, int level, int is\_pu, int arity, struct starpu\_tree \* father )

todo

27.37.3.3 struct starpu\_tree \* starpu\_tree\_get ( struct starpu\_tree \* tree, int id )

todo

27.37.3.4 struct starpu\_tree \* starpu\_tree\_get\_neighbour ( struct starpu\_tree \* tree, struct starpu\_tree \* node, int \* visited, int \* present )

todo

27.37.3.5 void starpu\_tree\_free ( struct starpu\_tree \* tree )

todo

## 27.38 Scheduling Context Hypervisor - Regular usage

### Macros

- #define SC\_HYPERVISOR\_MAX\_IDLE
- #define SC\_HYPERVISOR\_PRIORITY
- #define SC\_HYPERVISOR\_MIN\_WORKERS
- #define SC\_HYPERVISOR\_MAX\_WORKERS
- #define SC\_HYPERVISOR\_GRANULARITY
- #define SC\_HYPERVISOR\_FIXED\_WORKERS
- #define SC\_HYPERVISOR\_MIN\_TASKS
- #define SC\_HYPERVISOR\_NEW\_WORKERS\_MAX\_IDLE
- #define SC\_HYPERVISOR\_TIME\_TO\_APPLY
- #define SC\_HYPERVISOR\_ISPEED\_W\_SAMPLE
- #define SC\_HYPERVISOR\_ISPEED\_CTX\_SAMPLE
- #define SC\_HYPERVISOR\_NULL

### Functions

- void \* sc\_hypervisor\_init (struct sc\_hypervisor\_policy \*policy)
- void sc\_hypervisor\_shutdown (void)
- void sc\_hypervisor\_register\_ctx (unsigned sched\_ctx, double total\_flops)
- void sc\_hypervisor\_unregister\_ctx (unsigned sched\_ctx)
- void sc\_hypervisor\_resize\_ctxs (unsigned \*sched\_ctxs, int nsched\_ctxs, int \*workers, int nworkers)
- void sc\_hypervisor\_stop\_resize (unsigned sched\_ctx)
- void sc\_hypervisor\_start\_resize (unsigned sched\_ctx)
- const char \* sc\_hypervisor\_get\_policy ()
- void sc\_hypervisor\_add\_workers\_to\_sched\_ctx (int \*workers\_to\_add, unsigned nworkers\_to\_add, unsigned sched\_ctx)
- void sc\_hypervisor\_remove\_workers\_from\_sched\_ctx (int \*workers\_to\_remove, unsigned nworkers\_to\_remove, unsigned sched\_ctx, unsigned now)
- void sc\_hypervisor\_move\_workers (unsigned sender\_sched\_ctx, unsigned receiver\_sched\_ctx, int \*workers\_to\_move, unsigned nworkers\_to\_move, unsigned now)
- void sc\_hypervisor\_size\_ctxs (unsigned \*sched\_ctxs, int nsched\_ctxs, int \*workers, int nworkers)

- void [sc\\_hypervisor\\_set\\_type\\_of\\_task](#) (struct [starpu\\_codelet](#) \*cl, unsigned sched\_ctx, uint32\_t footprint, size\_t data\_size)
- void [sc\\_hypervisor\\_update\\_diff\\_total\\_flops](#) (unsigned sched\_ctx, double diff\_total\_flops)
- void [sc\\_hypervisor\\_update\\_diff\\_elapsed\\_flops](#) (unsigned sched\_ctx, double diff\_task\_flops)
- void [sc\\_hypervisor\\_ctl](#) (unsigned sched\_ctx,...)

### 27.38.1 Detailed Description

### 27.38.2 Macro Definition Documentation

#### 27.38.2.1 #define SC\_HYPERVISOR\_MAX\_IDLE

This macro is used when calling [sc\\_hypervisor\\_ctl\(\)](#) and must be followed by 3 arguments: an array of int for the workerids to apply the condition, an int to indicate the size of the array, and a double value indicating the maximum idle time allowed for a worker before the resizing process should be triggered

#### 27.38.2.2 #define SC\_HYPERVISOR\_PRIORITY

This macro is used when calling [sc\\_hypervisor\\_ctl\(\)](#) and must be followed by 3 arguments: an array of int for the workerids to apply the condition, an int to indicate the size of the array, and an int value indicating the priority of the workers previously mentioned. The workers with the smallest priority are moved the first.

#### 27.38.2.3 #define SC\_HYPERVISOR\_MIN\_WORKERS

This macro is used when calling [sc\\_hypervisor\\_ctl\(\)](#) and must be followed by 1 argument(int) indicating the minimum number of workers a context should have, underneath this limit the context cannot execute.

#### 27.38.2.4 #define SC\_HYPERVISOR\_MAX\_WORKERS

This macro is used when calling [sc\\_hypervisor\\_ctl\(\)](#) and must be followed by 1 argument(int) indicating the maximum number of workers a context should have, above this limit the context would not be able to scale

#### 27.38.2.5 #define SC\_HYPERVISOR\_GRANULARITY

This macro is used when calling [sc\\_hypervisor\\_ctl\(\)](#) and must be followed by 1 argument(int) indicating the granularity of the resizing process (the number of workers should be moved from the context once it is resized) This parameter is ignore for the Gflops rate based strategy (see [Resizing Strategies](#)), the number of workers that have to be moved is calculated by the strategy.

#### 27.38.2.6 #define SC\_HYPERVISOR\_FIXED\_WORKERS

This macro is used when calling [sc\\_hypervisor\\_ctl\(\)](#) and must be followed by 2 arguments: an array of int for the workerids to apply the condition and an int to indicate the size of the array. These workers are not allowed to be moved from the context.

#### 27.38.2.7 #define SC\_HYPERVISOR\_MIN\_TASKS

This macro is used when calling [sc\\_hypervisor\\_ctl\(\)](#) and must be followed by 1 argument (int) that indicated the minimum number of tasks that have to be executed before the context could be resized. This parameter is ignored for the Application Driven strategy (see [Resizing Strategies](#)) where the user indicates exactly when the resize should be done.

**27.38.2.8 #define SC\_HYPERVISOR\_NEW\_WORKERS\_MAX\_IDLE**

This macro is used when calling [sc\\_hypervisor\\_ctl\(\)](#) and must be followed by 1 argument, a double value indicating the maximum idle time allowed for workers that have just been moved from other contexts in the current context.

**27.38.2.9 #define SC\_HYPERVISOR\_TIME\_TO\_APPLY**

This macro is used when calling [sc\\_hypervisor\\_ctl\(\)](#) and must be followed by 1 argument (int) indicating the tag an executed task should have such that this configuration should be taken into account.

**27.38.2.10 #define SC\_HYPERVISOR\_ISPEED\_W\_SAMPLE**

This macro is used when calling [sc\\_hypervisor\\_ctl\(\)](#) and must be followed by 1 argument, a double, that indicates the number of flops needed to be executed before computing the speed of a worker

**27.38.2.11 #define SC\_HYPERVISOR\_ISPEED\_CTX\_SAMPLE**

This macro is used when calling [sc\\_hypervisor\\_ctl\(\)](#) and must be followed by 1 argument, a double, that indicates the number of flops needed to be executed before computing the speed of a context

**27.38.2.12 #define SC\_HYPERVISOR\_NULL**

This macro is used when calling [sc\\_hypervisor\\_ctl\(\)](#) and must be followed by 1 arguments

**27.38.3 Function Documentation****27.38.3.1 void \* sc\_hypervisor\_init ( struct sc\_hypervisor\_policy \* policy )**

There is a single hypervisor that is in charge of resizing contexts and the resizing strategy is chosen at the initialization of the hypervisor. A single resize can be done at a time.

The Scheduling Context Hypervisor Plugin provides a series of performance counters to StarPU. By incrementing them, StarPU can help the hypervisor in the resizing decision making process.

This function initializes the hypervisor to use the strategy provided as parameter and creates the performance counters (see [starpu\\_sched\\_ctx\\_performance\\_counters](#)). These performance counters represent actually some callbacks that will be used by the contexts to notify the information needed by the hypervisor.

Note: The Hypervisor is actually a worker that takes this role once certain conditions trigger the resizing process (there is no additional thread assigned to the hypervisor).

**27.38.3.2 void sc\_hypervisor\_shutdown ( void )**

The hypervisor and all information concerning it is cleaned. There is no synchronization between this function and [starpu\\_shutdown\(\)](#). Thus, this should be called after [starpu\\_shutdown\(\)](#), because the performance counters will still need allocated callback functions.

**27.38.3.3 void sc\_hypervisor\_register\_ctx ( unsigned sched\_ctx, double total\_flops )**

Scheduling Contexts that have to be resized by the hypervisor must be first registered to the hypervisor. This function registers the context to the hypervisor, and indicate the number of flops the context will execute (used for Gflops rate based strategy or any other custom strategy needing it, for the others we can pass 0.0)



27.38.3.4 void `sc_hypervisor_unregister_ctx` ( unsigned *sched\_ctx* )

Whenever we want to exclude contexts from the resizing process we have to unregister them from the hypervisor.

27.38.3.5 void `sc_hypervisor_resize_ctxs` ( unsigned \* *sched\_ctxs*, int *nsched\_ctxs*, int \* *workers*, int *nworkers* )

Requires reconsidering the distribution of ressources over the indicated scheduling contexts

27.38.3.6 void `sc_hypervisor_stop_resize` ( unsigned *sched\_ctx* )

The user can totally forbid the resizing of a certain context or can then change his mind and allow it (in this case the resizing is managed by the hypervisor, that can forbid it or allow it)

27.38.3.7 void `sc_hypervisor_start_resize` ( unsigned *sched\_ctx* )

Allow resizing of a context. The user can then provide information to the hypervisor concerning the conditions of resizing.

27.38.3.8 char \* `sc_hypervisor_get_policy` ( )

Returns the name of the resizing policy the hypervisor uses

27.38.3.9 void `sc_hypervisor_add_workers_to_sched_ctx` ( int \* *workers\_to\_add*, unsigned *nworkers\_to\_add*, unsigned *sched\_ctx* )

Ask the hypervisor to add workers to a *sched\_ctx*

27.38.3.10 void `sc_hypervisor_remove_workers_from_sched_ctx` ( int \* *workers\_to\_remove*, unsigned *nworkers\_to\_remove*, unsigned *sched\_ctx*, unsigned *now* )

Ask the hypervisor to remove workers from a *sched\_ctx*

27.38.3.11 void `sc_hypervisor_move_workers` ( unsigned *sender\_sched\_ctx*, unsigned *receiver\_sched\_ctx*, int \* *workers\_to\_move*, unsigned *nworkers\_to\_move*, unsigned *now* )

Moves workers from one context to another

27.38.3.12 void `sc_hypervisor_size_ctxs` ( unsigned \* *sched\_ctxs*, int *nsched\_ctxs*, int \* *workers*, int *nworkers* )

Ask the hypervisor to chose a distribution of workers in the required contexts

27.38.3.13 void `sc_hypervisor_set_type_of_task` ( struct *starpup\_codelet* \* *cl*, unsigned *sched\_ctx*, uint32\_t *footprint*, size\_t *data\_size* )

Indicate the types of tasks a context will execute in order to better decide the sizing of *ctxs*

27.38.3.14 void `sc_hypervisor_update_diff_total_flops` ( unsigned *sched\_ctx*, double *diff\_total\_flops* )

Change dynamically the total number of flops of a context, move the deadline of the finishing time of the context

27.38.3.15 void `sc_hypervisor_update_diff_elapsed_flops` ( unsigned *sched\_ctx*, double *diff\_task\_flops* )

Change dynamically the number of the elapsed flops in a context, modify the past in order to better compute the speed

27.38.3.16 void `sc_hypervisor_ctl` ( unsigned *sched\_ctx*, ... )

Inputs conditions to the context `sched_ctx` with the following arguments. The argument list must be zero-terminated.

## 27.39 Scheduling Context Hypervisor - Building a new resizing policy

### Data Structures

- struct `sc_hypervisor_policy`
- struct `sc_hypervisor_policy_config`
- struct `sc_hypervisor_wrapper`
- struct `sc_hypervisor_resize_ack`
- struct `sc_hypervisor_policy_task_pool`

### Functions

- void `sc_hypervisor_post_resize_request` (unsigned *sched\_ctx*, int *task\_tag*)
- unsigned `sc_hypervisor_get_size_req` (unsigned \*\**sched\_ctxs*, int \**nsched\_ctxs*, int \*\**workers*, int \**nworkers*)
- void `sc_hypervisor_save_size_req` (unsigned \**sched\_ctxs*, int *nsched\_ctxs*, int \**workers*, int *nworkers*)
- void `sc_hypervisor_free_size_req` (void)
- unsigned `sc_hypervisor_can_resize` (unsigned *sched\_ctx*)
- struct `sc_hypervisor_policy_config` \* `sc_hypervisor_get_config` (unsigned *sched\_ctx*)
- void `sc_hypervisor_set_config` (unsigned *sched\_ctx*, void \**config*)
- unsigned \* `sc_hypervisor_get_sched_ctxs` ()
- int `sc_hypervisor_get_nsched_ctxs` ()
- struct `sc_hypervisor_wrapper` \* `sc_hypervisor_get_wrapper` (unsigned *sched\_ctx*)
- double `sc_hypervisor_get_elapsed_flops_per_sched_ctx` (struct `sc_hypervisor_wrapper` \**sc\_w*)

### 27.39.1 Detailed Description

### 27.39.2 Data Structure Documentation

#### 27.39.2.1 struct `sc_hypervisor_policy`

This structure contains all the methods that implement a hypervisor resizing policy.

#### Data Fields

- const char \* `name`
- unsigned `custom`
- void(\* `size_ctxs` )(unsigned \**sched\_ctxs*, int *nsched\_ctxs*, int \**workers*, int *nworkers*)
- void(\* `resize_ctxs` )(unsigned \**sched\_ctxs*, int *nsched\_ctxs*, int \**workers*, int *nworkers*)
- void(\* `handle_idle_cycle` )(unsigned *sched\_ctx*, int *worker*)
- void(\* `handle_pushed_task` )(unsigned *sched\_ctx*, int *worker*)

- void(\* [handle\\_poped\\_task](#) )(unsigned sched\_ctx, int worker, struct [starpu\\_task](#) \*task, uint32\_t footprint)
- void(\* [handle\\_idle\\_end](#) )(unsigned sched\_ctx, int worker)
- void(\* [handle\\_post\\_exec\\_hook](#) )(unsigned sched\_ctx, int task\_tag)
- void(\* [handle\\_submitted\\_job](#) )(struct [starpu\\_codelet](#) \*cl, unsigned sched\_ctx, uint32\_t footprint, size\_t data\_size)
- void(\* [end\\_ctx](#) )(unsigned sched\_ctx)

#### 27.39.2.1.1 Field Documentation

##### 27.39.2.1.1.1 `sc_hypervisor_policy::name`

Indicates the name of the policy, if there is not a custom policy, the policy corresponding to this name will be used by the hypervisor

##### 27.39.2.1.1.2 `sc_hypervisor_policy::custom`

Indicates whether the policy is custom or not

##### 27.39.2.1.1.3 `sc_hypervisor_policy::size_ctxs`

Distribute workers to contexts even at the beginning of the program

##### 27.39.2.1.1.4 `sc_hypervisor_policy::resize_ctxs`

Require explicit resizing

##### 27.39.2.1.1.5 `sc_hypervisor_policy::handle_idle_cycle`

It is called whenever the indicated worker executes another idle cycle in sched\_ctx

##### 27.39.2.1.1.6 `sc_hypervisor_policy::handle_pushed_task`

It is called whenever a task is pushed on the worker's queue corresponding to the context sched\_ctx

##### 27.39.2.1.1.7 `sc_hypervisor_policy::handle_poped_task`

It is called whenever a task is popped from the worker's queue corresponding to the context sched\_ctx

The hypervisor takes a decision when another task was popped from this worker in this ctx

##### 27.39.2.1.1.8 `sc_hypervisor_policy::handle_idle_end`

It is called whenever a task is executed on the indicated worker and context after a long period of idle time

##### 27.39.2.1.1.9 `sc_hypervisor_policy::handle_post_exec_hook`

It is called whenever a tag task has just been executed. The table of resize requests is provided as well as the tag

##### 27.39.2.1.1.10 `sc_hypervisor_policy::handle_submitted_job`

The hypervisor takes a decision when a job was submitted in this ctx

##### 27.39.2.1.1.11 `sc_hypervisor_policy::end_ctx`

The hypervisor takes a decision when a certain ctx was deleted

#### 27.39.2.2 `struct sc_hypervisor_policy_config`

This structure contains all configuration information of a context. It contains configuration information for each context, which can be used to construct new resize strategies.

## Data Fields

|        |                                                       |                                                                                                                            |
|--------|-------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| int    | min_nworkers                                          | Indicates the minimum number of workers needed by the context                                                              |
| int    | max_nworkers                                          | Indicates the maximum number of workers needed by the context                                                              |
| int    | granularity                                           | Indicates the workers granularity of the context                                                                           |
| int    | priority[ <a href="#">STARPU_NMAXWORKERS</a> ]        | Indicates the priority of each worker in the context                                                                       |
| double | max_idle[ <a href="#">STARPU_NMAXWORKERS</a> ]        | Indicates the maximum idle time accepted before a resize is triggered                                                      |
| double | min_working[ <a href="#">STARPU_NMAXWORKERS</a> ]     | Indicates that underneath this limit the priority of the worker is reduced                                                 |
| int    | fixed_workers[ <a href="#">STARPU_NMAXWORKERS</a> ]   | Indicates which workers can be moved and which ones are fixed                                                              |
| double | new_workers_max_idle                                  | Indicates the maximum idle time accepted before a resize is triggered for the workers that just arrived in the new context |
| double | ispeed_w_sample[ <a href="#">STARPU_NMAXWORKERS</a> ] | Indicates the sample used to compute the instant speed per worker                                                          |
| double | ispeed_ctx_sample                                     | Indicates the sample used to compute the instant speed per ctxs                                                            |
| double | time_sample                                           | todo                                                                                                                       |

## 27.39.2.3 struct sc\_hypervisor\_wrapper

This structure is a wrapper of the contexts available in StarPU and contains all information about a context obtained by incrementing the performance counters.

## Data Fields

|                                                         |                                                         |                                                     |
|---------------------------------------------------------|---------------------------------------------------------|-----------------------------------------------------|
| unsigned                                                | sched_ctx                                               | The context wrapped                                 |
| struct<br><a href="#">sc_hypervisor_policy_config</a> * | config                                                  | The corresponding resize configuration              |
| double                                                  | start_time_w[ <a href="#">STARPU_NMAXWORKERS</a> ]      |                                                     |
| double                                                  | current_idle_time[ <a href="#">STARPU_NMAXWORKERS</a> ] | The idle time counter of each worker of the context |
| double                                                  | idle_time[ <a href="#">STARPU_NMAXWORKERS</a> ]         | The time the workers were idle from the last resize |
| double                                                  | idle_start_time[ <a href="#">STARPU_NMAXWORKERS</a> ]   | The moment when the workers started being idle      |

|                                    |                                          |                                                                                                        |
|------------------------------------|------------------------------------------|--------------------------------------------------------------------------------------------------------|
| double                             | exec_time[STARPU_NMAXWORKERS]            |                                                                                                        |
| double                             | exec_start_time[STARPU_NMAXWORKERS]      |                                                                                                        |
| int                                | worker_to_be_removed[STARPU_NMAXWORKERS] | The list of workers that will leave this contexts (lazy resizing process)                              |
| int                                | pushed_tasks[STARPU_NMAXWORKERS]         | The number of pushed tasks of each worker of the context                                               |
| int                                | popped_tasks[STARPU_NMAXWORKERS]         | The number of popped tasks of each worker of the context                                               |
| double                             | total_flops                              | The total number of flops to execute by the context                                                    |
| double                             | total_elapsed_flops[STARPU_NMAXWORKERS]  | The number of flops executed by each workers of the context                                            |
| double                             | elapsed_flops[STARPU_NMAXWORKERS]        | The number of flops executed by each worker of the context from last resize                            |
| size_t                             | elapsed_data[STARPU_NMAXWORKERS]         | The quantity of data (in bytes) used to execute tasks on each worker in this ctx                       |
| int                                | elapsed_tasks[STARPU_NMAXWORKERS]        | The nr of tasks executed on each worker in this ctx                                                    |
| double                             | ref_speed[2]                             | The average speed of the workers (type of workers) when they belonged to this context 0 - cuda 1 - cpu |
| double                             | submitted_flops                          | The number of flops submitted to this ctx                                                              |
| double                             | remaining_flops                          | The number of flops that still have to be executed by the workers in the context                       |
| double                             | start_time                               | The time when he started executed                                                                      |
| double                             | real_start_time                          | The first time a task was pushed to this context                                                       |
| double                             | hyp_react_start_time                     |                                                                                                        |
| struct<br>sc_hypervisor_resize_ack | resize_ack                               | The structure confirming the last resize finished and a new one can be done                            |
| starpu_thread_mutex_t              | mutex                                    | The mutex needed to synchronize the acknowledgment of the workers into the receiver context            |

|          |                                                         |                                                                                                               |
|----------|---------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| unsigned | total_flops_↔<br>available                              | A boolean indicating if the hypervisor can use the flops corresponding to the entire execution of the context |
| unsigned | to_be_sized                                             |                                                                                                               |
| unsigned | compute_↔<br>idle[STARPU_↔<br>NMAXWORKE↔<br>RS]         |                                                                                                               |
| unsigned | compute_↔<br>partial_idle[ST↔<br>ARPU_NMAX↔<br>WORKERS] |                                                                                                               |
| unsigned | consider_max                                            |                                                                                                               |

#### 27.39.2.4 struct sc\_hypervisor\_resize\_ack

This structures checks if the workers moved to another context are actually taken into account in that context.

##### Data Fields

|       |                         |                                                                                                             |
|-------|-------------------------|-------------------------------------------------------------------------------------------------------------|
| int   | receiver_↔<br>sched_ctx | The context receiving the new workers                                                                       |
| int * | moved_workers           | The workers moved to the receiver context                                                                   |
| int   | nmoved_workers          | The number of workers moved                                                                                 |
| int * | acked_workers           | If the value corresponding to a worker is 1, this one is taken into account in the new context if 0 not yet |

#### 27.39.2.5 struct sc\_hypervisor\_policy\_task\_pool

task wrapper linked list

##### Data Fields

|                                                          |              |                                                              |
|----------------------------------------------------------|--------------|--------------------------------------------------------------|
| struct<br>starpu_codelet *                               | cl           | Which codelet has been executed                              |
| uint32_t                                                 | footprint    | Task footprint key                                           |
| unsigned                                                 | sched_ctx_id | Context the task belongs to                                  |
| unsigned long                                            | n            | Number of tasks of this kind                                 |
| size_t                                                   | data_size    | The quantity of data(in bytes) needed by the task to execute |
| struct<br>sc_hypervisor_↔<br>_policy_task_↔<br>pool<br>* | next         | Other task kinds                                             |

### 27.39.3 Function Documentation

#### 27.39.3.1 void sc\_hypervisor\_post\_resize\_request ( unsigned sched\_ctx, int task\_tag )

Requires resizing the context `sched_ctx` whenever a task tagged with the id `task_tag` finished executing

#### 27.39.3.2 unsigned sc\_hypervisor\_get\_size\_req ( unsigned \*\* sched\_ctxs, int \* nsched\_ctxs, int \*\* workers, int \* nworkers )

Check if there are pending demands of resizing

27.39.3.3 `void sc_hypervisor_save_size_req ( unsigned * sched_ctxs, int nsched_ctxs, int * workers, int nworkers )`

Save a demand of resizing

27.39.3.4 `void sc_hypervisor_free_size_req ( void )`

Clear the list of pending demands of resizing

27.39.3.5 `unsigned sc_hypervisor_can_resize ( unsigned sched_ctx )`

Check out if a context can be resized

27.39.3.6 `struct sc_hypervisor_policy_config * sc_hypervisor_get_config ( unsigned sched_ctx )`

Returns the configuration structure of a context

27.39.3.7 `void sc_hypervisor_set_config ( unsigned sched_ctx, void * config )`

Set a certain configuration to a contexts

27.39.3.8 `unsigned * sc_hypervisor_get_sched_ctxs ( )`

Gets the contexts managed by the hypervisor

27.39.3.9 `int sc_hypervisor_get_nsched_ctxs ( )`

Gets the number of contexts managed by the hypervisor

27.39.3.10 `struct sc_hypervisor_wrapper * sc_hypervisor_get_wrapper ( unsigned sched_ctx )`

Returns the wrapper corresponding the context `sched_ctx`

27.39.3.11 `double sc_hypervisor_get_elapsed_flops_per_sched_ctx ( struct sc_hypervisor_wrapper * sc_w )`

Returns the flops of a context elapsed from the last resize

## 27.40 Modularized Scheduler Interface

### Data Structures

- struct [starpu\\_sched\\_component](#)
- struct [starpu\\_sched\\_tree](#)
- struct [starpu\\_sched\\_component\\_composed\\_recipe](#)
- struct [starpu\\_sched\\_component\\_specs](#)

### Macros

- `#define STARPU_SCHED_COMPONENT_IS_HOMOGENEOUS(component)`
- `#define STARPU_SCHED_COMPONENT_IS_SINGLE_MEMORY_NODE(component)`

## Enumerations

- enum `starpu_sched_component_properties` { `STARPU_SCHED_COMPONENT_HOMOGENEOUS`, `STARPU_SCHED_COMPONENT_SINGLE_MEMORY_NODE` }

## Scheduling Tree API

- struct `starpu_sched_tree` \* `starpu_sched_tree_create` (unsigned sched\_ctx\_id)
- void `starpu_sched_tree_destroy` (struct `starpu_sched_tree` \*tree)
- void `starpu_sched_tree_update_workers` (struct `starpu_sched_tree` \*t)
- void `starpu_sched_tree_update_workers_in_ctx` (struct `starpu_sched_tree` \*t)
- int `starpu_sched_tree_push_task` (struct `starpu_task` \*task)
- struct `starpu_task` \* `starpu_sched_tree_pop_task` ()
- void `starpu_sched_tree_add_workers` (unsigned sched\_ctx\_id, int \*workerids, unsigned nworkers)
- void `starpu_sched_tree_remove_workers` (unsigned sched\_ctx\_id, int \*workerids, unsigned nworkers)

## Generic Scheduling Component API

- struct `starpu_sched_component` \* `starpu_sched_component_create` (struct `starpu_sched_tree` \*tree)
- void `starpu_sched_component_destroy` (struct `starpu_sched_component` \*component)
- void `starpu_sched_component_destroy_rec` (struct `starpu_sched_component` \*component)
- int `starpu_sched_component_can_execute_task` (struct `starpu_sched_component` \*component, struct `starpu_task` \*task)
- int `STARPU_WARN_UNUSED_RESULT` `starpu_sched_component_execute_preds` (struct `starpu_sched_component` \*component, struct `starpu_task` \*task, double \*length)
- double `starpu_sched_component_transfer_length` (struct `starpu_sched_component` \*component, struct `starpu_task` \*task)

## Worker Component API

- struct `starpu_sched_component` \* `starpu_sched_component_worker_get` (unsigned sched\_ctx, int workerid)
- int `starpu_sched_component_worker_get_workerid` (struct `starpu_sched_component` \*worker\_component)
- int `starpu_sched_component_is_worker` (struct `starpu_sched_component` \*component)
- int `starpu_sched_component_is_simple_worker` (struct `starpu_sched_component` \*component)
- int `starpu_sched_component_is_combined_worker` (struct `starpu_sched_component` \*component)
- void `starpu_sched_component_worker_pre_exec_hook` (struct `starpu_task` \*task)
- void `starpu_sched_component_worker_post_exec_hook` (struct `starpu_task` \*task)

## Flow-control Fifo Component API

- struct `starpu_sched_component` \* `starpu_sched_component_fifo_create` (struct `starpu_sched_tree` \*tree, struct `starpu_sched_component_fifo_data` \*fifo\_data)
- int `starpu_sched_component_is_fifo` (struct `starpu_sched_component` \*component)

## Flow-control Prio Component API

- struct `starpu_sched_component` \* `starpu_sched_component_prio_create` (struct `starpu_sched_tree` \*tree, struct `starpu_sched_component_prio_data` \*prio\_data)
- int `starpu_sched_component_is_prio` (struct `starpu_sched_component` \*component)



**Resource-mapping Work-Stealing Component API**

- int [starpu\\_sched\\_tree\\_work\\_stealing\\_push\\_task](#) (struct [starpu\\_task](#) \*task)
- int [starpu\\_sched\\_component\\_is\\_work\\_stealing](#) (struct [starpu\\_sched\\_component](#) \*component)

**Resource-mapping Random Component API**

- int [starpu\\_sched\\_component\\_is\\_random](#) (struct [starpu\\_sched\\_component](#) \*)

**Resource-mapping Eager Component API**

- int [starpu\\_sched\\_component\\_is\\_eager](#) (struct [starpu\\_sched\\_component](#) \*)

**Resource-mapping Eager-Calibration Component API**

- int [starpu\\_sched\\_component\\_is\\_eager\\_calibration](#) (struct [starpu\\_sched\\_component](#) \*)

**Resource-mapping MCT Component API**

- struct [starpu\\_sched\\_component](#) \* [starpu\\_sched\\_component\\_mct\\_create](#) (struct [starpu\\_sched\\_tree](#) \*tree, struct [starpu\\_sched\\_component\\_mct\\_data](#) \*mct\_data)
- int [starpu\\_sched\\_component\\_is\\_mct](#) (struct [starpu\\_sched\\_component](#) \*component)

**Resource-mapping Heft Component API**

- struct [starpu\\_sched\\_component](#) \* [starpu\\_sched\\_component\\_heft\\_create](#) (struct [starpu\\_sched\\_tree](#) \*tree, struct [starpu\\_sched\\_component\\_mct\\_data](#) \*mct\_data)
- int [starpu\\_sched\\_component\\_is\\_heft](#) (struct [starpu\\_sched\\_component](#) \*component)

**Special-purpose Perfmodel\_Select Component API**

- struct [starpu\\_sched\\_component](#) \* [starpu\\_sched\\_component\\_perfmodel\\_select\\_create](#) (struct [starpu\\_sched\\_tree](#) \*tree, struct [starpu\\_sched\\_component\\_perfmodel\\_select\\_data](#) \*perfmodel\_select\_data)
- int [starpu\\_sched\\_component\\_is\\_perfmodel\\_select](#) (struct [starpu\\_sched\\_component](#) \*component)

**Recipe Component API**

- struct [starpu\\_sched\\_component\\_composed\\_recipe](#) \* [starpu\\_sched\\_component\\_composed\\_recipe\\_create](#) (void)
- struct [starpu\\_sched\\_component\\_composed\\_recipe](#) \* [starpu\\_sched\\_component\\_composed\\_recipe\\_create\\_singleton](#) (struct [starpu\\_sched\\_component](#) \*(\*create\_component)(struct [starpu\\_sched\\_tree](#) \*tree, void \*arg), void \*arg)
- void [starpu\\_sched\\_component\\_composed\\_recipe\\_add](#) (struct [starpu\\_sched\\_component\\_composed\\_recipe](#) \*recipe, struct [starpu\\_sched\\_component](#) \*(\*create\_component)(struct [starpu\\_sched\\_tree](#) \*tree, void \*arg), void \*arg)
- void [starpu\\_sched\\_component\\_composed\\_recipe\\_destroy](#) (struct [starpu\\_sched\\_component\\_composed\\_recipe](#) \*)
- struct [starpu\\_sched\\_component](#) \* [starpu\\_sched\\_component\\_composed\\_component\\_create](#) (struct [starpu\\_sched\\_tree](#) \*tree, struct [starpu\\_sched\\_component\\_composed\\_recipe](#) \*recipe)
- struct [starpu\\_sched\\_tree](#) \* [starpu\\_sched\\_component\\_make\\_scheduler](#) (unsigned sched\_ctx\_id, struct [starpu\\_sched\\_component\\_specs](#) s)

### 27.40.1 Detailed Description

### 27.40.2 Data Structure Documentation

#### 27.40.2.1 struct starpu\_sched\_component

This structure represent a scheduler module. A scheduler is a tree-like structure of them, some parts of scheduler can be shared by several contexes to perform some local optimisations, so, for all components, a list of parent is defined indexed by sched\_ctx\_id. They embed there specialised method in a pseudo object-style, so calls are like component->push\_task(component,task)

#### Data Fields

- struct [starpu\\_sched\\_tree](#) \* [tree](#)
- struct starpu\_bitmap \* [workers](#)
- struct starpu\_bitmap \* [workers\\_in\\_ctx](#)
- void \* [data](#)
- int [nchildren](#)
- struct [starpu\\_sched\\_component](#) \*\* [children](#)
- int [nparents](#)
- struct [starpu\\_sched\\_component](#) \*\* [parents](#)
- void(\* [add\\_child](#) )(struct [starpu\\_sched\\_component](#) \*component, struct [starpu\\_sched\\_component](#) \*child)
- void(\* [remove\\_child](#) )(struct [starpu\\_sched\\_component](#) \*component, struct [starpu\\_sched\\_component](#) \*child)
- void(\* [add\\_parent](#) )(struct [starpu\\_sched\\_component](#) \*component, struct [starpu\\_sched\\_component](#) \*parent)
- void(\* [remove\\_parent](#) )(struct [starpu\\_sched\\_component](#) \*component, struct [starpu\\_sched\\_component](#) \*parent)
- int(\* [push\\_task](#) )(struct [starpu\\_sched\\_component](#) \*, struct [starpu\\_task](#) \*)
- struct [starpu\\_task](#) \*(\* [pull\\_task](#) )(struct [starpu\\_sched\\_component](#) \*)
- int(\* [can\\_push](#) )(struct [starpu\\_sched\\_component](#) \*component)
- void(\* [can\\_pull](#) )(struct [starpu\\_sched\\_component](#) \*component)
- double(\* [estimated\\_load](#) )(struct [starpu\\_sched\\_component](#) \*component)
- double(\* [estimated\\_end](#) )(struct [starpu\\_sched\\_component](#) \*component)
- void(\* [deinit\\_data](#) )(struct [starpu\\_sched\\_component](#) \*component)
- void(\* [notify\\_change\\_workers](#) )(struct [starpu\\_sched\\_component](#) \*component)
- int [properties](#)
- hwloc\_obj\_t [obj](#)

#### 27.40.2.1.1 Field Documentation

##### 27.40.2.1.1.1 struct starpu\_sched\_tree \* starpu\_sched\_component::tree

The tree containing the component

##### 27.40.2.1.1.2 struct starpu\_bitmap \* starpu\_sched\_component::workers

this member contain the set of underlying workers

##### 27.40.2.1.1.3 starpu\_sched\_component::workers\_in\_ctx

this member contain the subset of [starpu\\_sched\\_component::workers](#) that is currently available in the context The push method should take this member into account. this member is set with : component->workers UNION tree->workers UNION component->child[i]->workers\_in\_ctx iff exist x such as component->children[i]->parents[x] == component

##### 27.40.2.1.1.4 void \* starpu\_sched\_component::data

private data

27.40.2.1.1.5 `int starpu_sched_component::nchildren`

the number of components's children

27.40.2.1.1.6 `struct starpu_sched_component ** starpu_sched_component::children`

the vector of component's children

27.40.2.1.1.7 `int starpu_sched_component::nparents`

the numbers of component's parents

27.40.2.1.1.8 `struct starpu_sched_component ** starpu_sched_component::parents`

the vector of component's parents

27.40.2.1.1.9 `void(* starpu_sched_component::add_child)(struct starpu_sched_component *component, struct starpu_sched_component *child)`

add a child to component

27.40.2.1.1.10 `void(* starpu_sched_component::remove_child)(struct starpu_sched_component *component, struct starpu_sched_component *child)`

remove a child from component

27.40.2.1.1.11 `void(* starpu_sched_component::add_parent)(struct starpu_sched_component *component, struct starpu_sched_component *parent)`

todo

27.40.2.1.1.12 `void(* starpu_sched_component::remove_parent)(struct starpu_sched_component *component, struct starpu_sched_component *parent)`

todo

27.40.2.1.1.13 `int(* starpu_sched_component::push_task)(struct starpu_sched_component *, struct starpu_task *)`

push a task in the scheduler module. this function is called to push a task on component subtree, this can either perform a recursive call on a child or store the task in the component, then it will be returned by a further pull\_task call. the caller must ensure that component is able to execute task.

27.40.2.1.1.14 `struct starpu_task *(* starpu_sched_component::pull_task)(struct starpu_sched_component *)`

pop a task from the scheduler module. this function is called by workers to get a task from their parents. this function should first return a locally stored task or perform a recursive call on the parents. the task returned by this function is executable by the caller

27.40.2.1.1.15 `int(* starpu_sched_component::can_push)(struct starpu_sched_component *component)`

This function is called by a component which implements a queue, allowing it to signify to its parents that an empty slot is available in its queue. The basic implementation of this function is a recursive call to its parents, the user have to specify a personally-made function to catch those calls.

27.40.2.1.1.16 `void(* starpu_sched_component::can_pull)(struct starpu_sched_component *component)`

This function allow a component to wake up a worker. It is currently called by component which implements a queue, to signify to its children that a task have been pushed in its local queue, and is available to be popped by a worker, for example. The basic implementation of this function is a recursive call to its children, until at least one worker have been woken up.

**27.40.2.1.1.17** `double(* starpu_sched_component::estimated_load)(struct starpu_sched_component *component)`

is an heuristic to compute load of scheduler module. Basically the number of tasks divided by the sum of relatives speedup of workers available in context. `estimated_load(component) = sum(estimated_load(component_children)) + nb_local_tasks / average(relative_speedup(underlying_worker))`

**27.40.2.1.1.18** `starpu_sched_component::estimated_end`

return the time when a worker will enter in starvation. This function is relevant only if the task->predicted member has been set.

**27.40.2.1.1.19** `void(* starpu_sched_component::deinit_data)(struct starpu_sched_component *component)`

called by `starpu_sched_component_destroy`. Should free data allocated during creation

**27.40.2.1.1.20** `void(* starpu_sched_component::notify_change_workers)(struct starpu_sched_component *component)`

this function is called for each component when workers are added or removed from a context

**27.40.2.1.1.21** `int starpu_sched_component::properties`

todo

**27.40.2.1.1.22** `hwloc_obj_t starpu_sched_component::obj`

the hwloc object associated to scheduler module. points to the part of topology that is binded to this component, eg: a numa node for a ws component that would balance load between underlying sockets

**27.40.2.2** `struct starpu_sched_tree`

The actual scheduler

Data Fields

|                                              |                                                      |                                                                                                                                      |
|----------------------------------------------|------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <code>struct starpu_sched_component *</code> | <code>root</code>                                    | this is the entry module of the scheduler                                                                                            |
| <code>struct starpu_bitmap *</code>          | <code>workers</code>                                 | this is the set of workers available in this context, this value is used to mask workers in modules                                  |
| <code>unsigned</code>                        | <code>sched_ctx_id</code>                            | the context id of the scheduler                                                                                                      |
| <code>struct starpu_sched_component *</code> | <code>worker_components[S_TARPU_NMA_XWORKERS]</code> | worker components                                                                                                                    |
| <code>starpu_thread_mutex_t</code>           | <code>lock</code>                                    | this lock is used to protect the scheduler, it is taken in read mode pushing a task and in write mode for adding or removing workers |

**27.40.2.3** `struct starpu_sched_component_composed_recipe`

parameters for `starpu_sched_component_composed_component_create`

**27.40.2.4** `struct starpu_sched_component_specs`

Define how build a scheduler according to topology. Each level (except for `hwloc_machine_composed_scheduler_component`) can be NULL, then the level is just skipped. Bugs everywhere, do not rely on.

## Data Fields

- struct [starpu\\_sched\\_component\\_composed\\_recipe](#) \* [hwloc\\_machine\\_composed\\_sched\\_component](#)
- struct [starpu\\_sched\\_component\\_composed\\_recipe](#) \* [hwloc\\_component\\_composed\\_sched\\_component](#)
- struct [starpu\\_sched\\_component\\_composed\\_recipe](#) \* [hwloc\\_socket\\_composed\\_sched\\_component](#)
- struct [starpu\\_sched\\_component\\_composed\\_recipe](#) \* [hwloc\\_cache\\_composed\\_sched\\_component](#)
- struct [starpu\\_sched\\_component\\_composed\\_recipe](#) \* (\* [worker\\_composed\\_sched\\_component](#) )(enum [starpu\\_↵\\_worker\\_archtype](#) archtype)
- int **mix\_heterogeneous\_workers**

## 27.40.3 Macro Definition Documentation

27.40.3.1 **#define** STARPU\_SCHED\_COMPONENT\_IS\_HOMOGENEOUS( *component* )

indicate if component is homogeneous

27.40.3.2 **#define** STARPU\_SCHED\_COMPONENT\_IS\_SINGLE\_MEMORY\_NODE( *component* )

indicate if all workers have the same memory component

## 27.40.4 Enumeration Type Documentation

27.40.4.1 **enum** [starpu\\_sched\\_component\\_properties](#)

flags for [starpu\\_sched\\_component::properties](#)

## Enumerator

**STARPU\_SCHED\_COMPONENT\_HOMOGENEOUS** indicate that all workers have the same [starpu\\_↵\\_worker\\_archtype](#)

**STARPU\_SCHED\_COMPONENT\_SINGLE\_MEMORY\_NODE** indicate that all workers have the same memory component

## 27.40.5 Function Documentation

27.40.5.1 **struct** [starpu\\_sched\\_tree](#) \* [starpu\\_sched\\_tree\\_create](#) ( unsigned *sched\_ctx\_id* )

create a empty initialized [starpu\\_sched\\_tree](#)

27.40.5.2 **void** [starpu\\_sched\\_tree\\_destroy](#) ( **struct** [starpu\\_sched\\_tree](#) \* *tree* )

destroy tree and free all non shared component in it.

27.40.5.3 **void** [starpu\\_sched\\_tree\\_update\\_workers](#) ( **struct** [starpu\\_sched\\_tree](#) \* *t* )

recursively set all [starpu\\_sched\\_component::workers](#), do not take into account shared parts (except workers).

27.40.5.4 void starpu\_sched\_tree\_update\_workers\_in\_ctx ( struct starpu\_sched\_tree \* *t* )

recursively set all [starpu\\_sched\\_component::workers\\_in\\_ctx](#), do not take into account shared parts (except workers)

27.40.5.5 int starpu\_sched\_tree\_push\_task ( struct starpu\_task \* *task* )

compatibility with [starpu\\_sched\\_policy](#) interface

27.40.5.6 struct starpu\_task \* starpu\_sched\_tree\_pop\_task ( )

compatibility with [starpu\\_sched\\_policy](#) interface

27.40.5.7 void starpu\_sched\_tree\_add\_workers ( unsigned *sched\_ctx\_id*, int \* *workerids*, unsigned *nworkers* )

compatibility with [starpu\\_sched\\_policy](#) interface

27.40.5.8 void starpu\_sched\_tree\_remove\_workers ( unsigned *sched\_ctx\_id*, int \* *workerids*, unsigned *nworkers* )

compatibility with [starpu\\_sched\\_policy](#) interface

27.40.5.9 struct starpu\_sched\_component \* starpu\_sched\_component\_create ( struct starpu\_sched\_tree \* *tree* )

allocate and initialize component field with defaults values : `.pop_task` make recursive call on father `.estimated_load` compute relative speedup and tasks in sub tree `.estimated_end` return the average of recursive call on children `.add_child` is `starpu_sched_component_add_child` `.remove_child` is `starpu_sched_component_remove_child` `.notify_change_workers` does nothing `.deinit_data` does nothing

27.40.5.10 void starpu\_sched\_component\_destroy ( struct starpu\_sched\_component \* *component* )

free data allocated by `starpu_sched_component_create` and call `component->deinit_data(component)` set to null the member `starpu_sched_component::fathers[sched_ctx_id]` of all child if its equal to `component`

27.40.5.11 void starpu\_sched\_component\_destroy\_rec ( struct starpu\_sched\_component \* *component* )

recursively destroy non shared parts of a `component` 's tree

27.40.5.12 int starpu\_sched\_component\_can\_execute\_task ( struct starpu\_sched\_component \* *component*, struct starpu\_task \* *task* )

return true iff `component` can execute `task`, this function take into account the workers available in the scheduling context

27.40.5.13 int starpu\_sched\_component\_execute\_preds ( struct starpu\_sched\_component \* *component*, struct starpu\_task \* *task*, double \* *length* )

return a non null value if `component` can execute `task`. write the execution prediction length for the best implementation of the best worker available and write this at `length` address. this result is more relevant if `starpu_sched_component::is_homogeneous` is non null. if a worker need to be calibrated for an implementation, nan is set to `length`.

27.40.5.14 `double starpu_sched_component_transfer_length ( struct starpu_sched_component * component, struct starpu_task * task )`

return the average time to transfer `task` data to underlying `component` workers.

27.40.5.15 `struct starpu_sched_component * starpu_sched_component_worker_get ( unsigned sched_ctx, int workerid )`

return the struct `starpu_sched_component` corresponding to `workerid`. Undefined if `workerid` is not a valid `workerid`

27.40.5.16 `int starpu_sched_component_worker_get_workerid ( struct starpu_sched_component * worker_component )`

return the `workerid` of `worker_component`, undefined if `starpu_sched_component_is_worker(worker_component) == 0`

27.40.5.17 `int starpu_sched_component_is_worker ( struct starpu_sched_component * component )`

return true iff `component` is a worker component

27.40.5.18 `int starpu_sched_component_is_simple_worker ( struct starpu_sched_component * component )`

return true iff `component` is a simple worker component

27.40.5.19 `int starpu_sched_component_is_combined_worker ( struct starpu_sched_component * component )`

return true iff `component` is a combined worker component

27.40.5.20 `void starpu_sched_component_worker_pre_exec_hook ( struct starpu_task * task )`

compatibility with `starpu_sched_policy` interface update predictions for workers

27.40.5.21 `void starpu_sched_component_worker_post_exec_hook ( struct starpu_task * task )`

compatibility with `starpu_sched_policy` interface

27.40.5.22 `struct starpu_sched_component * starpu_sched_component_fifo_create ( struct starpu_sched_tree * tree, struct starpu_sched_component_fifo_data * fifo_data )`

Return a struct `starpu_sched_component` with a fifo. A stable sort is performed according to tasks priorities. A `push_task` call on this component does not perform recursive calls, underlying components will have to call `pop_task` to get it. `starpu_sched_component::estimated_end` function compute the estimated length by dividing the sequential length by the number of underlying workers. Do not take into account tasks that are currently executed.

27.40.5.23 `int starpu_sched_component_is_fifo ( struct starpu_sched_component * component )`

return true iff `component` is a fifo component

27.40.5.24 `struct starpu_sched_component * starpu_sched_component_prio_create ( struct starpu_sched_tree * tree, struct starpu_sched_component_prio_data * prio_data )`

todo

27.40.5.25 `int starpu_sched_component_is_prio ( struct starpu_sched_component * component )`

todo

27.40.5.26 `int starpu_sched_tree_work_stealing_push_task ( struct starpu_task * task )`

undefined if there is no work stealing component in the scheduler. If any, `task` is pushed in a default way if the caller is the application, and in the caller's fifo if its a worker.

27.40.5.27 `int starpu_sched_component_is_work_stealing ( struct starpu_sched_component * component )`

return true iff `component` is a work stealing component

27.40.5.28 `int starpu_sched_component_is_random ( struct starpu_sched_component * )`

return true iff `component` is a random component

27.40.5.29 `int starpu_sched_component_is_eager ( struct starpu_sched_component * )`

todo

27.40.5.30 `int starpu_sched_component_is_eager_calibration ( struct starpu_sched_component * )`

todo

27.40.5.31 `struct starpu_sched_component * starpu_sched_component_mct_create ( struct starpu_sched_tree * tree, struct starpu_sched_component_mct_data * mct_data )`

create a component with `mct_data` paremeters. the mct component doesnt do anything but pushing tasks on `no_↔` `perf_model_component` and `calibrating_component`

27.40.5.32 `int starpu_sched_component_is_mct ( struct starpu_sched_component * component )`

todo

27.40.5.33 `struct starpu_sched_component * starpu_sched_component_heft_create ( struct starpu_sched_tree * tree, struct starpu_sched_component_mct_data * mct_data )`

this component perform a heft scheduling

27.40.5.34 `int starpu_sched_component_is_heft ( struct starpu_sched_component * component )`

return true iff `component` is a heft component

27.40.5.35 `struct starpu_sched_component * starpu_sched_component_perfmodel_select_create ( struct starpu_sched_tree * tree, struct starpu_sched_component_perfmodel_select_data * perfmodel_select_data )`

todo



27.40.5.36 `int starpu_sched_component_is_perfmodel_select ( struct starpu_sched_component * component )`

todo

27.40.5.37 `struct starpu_sched_component_composed_recipe * starpu_sched_component_composed_recipe_create ( void )`

return an empty recipe for a composed component, it should not be used without modification

27.40.5.38 `struct starpu_sched_component_composed_recipe * starpu_sched_component_composed_recipe_create_singleton ( struct starpu_sched_component (*)(struct starpu_sched_tree *tree, void *arg) create_component, void * arg )`

return a recipe to build a composed component with a `create_component`

27.40.5.39 `void starpu_sched_component_composed_recipe_add ( struct starpu_sched_component_composed_recipe * recipe, struct starpu_sched_component (*)(struct starpu_sched_tree *tree, void *arg) create_component, void * arg )`

add `create_component` under all previous components in recipe

27.40.5.40 `void starpu_sched_component_composed_recipe_destroy ( struct starpu_sched_component_composed_recipe * )`

destroy `composed_sched_component`, this should be done after `starpu_sched_component_composed_recipe_create` was called

27.40.5.41 `struct starpu_sched_component * starpu_sched_component_composed_component_create ( struct starpu_sched_tree * tree, struct starpu_sched_component_composed_recipe * recipe )`

create a component that behave as all component of recipe where linked. Except that you cant use `starpu_sched_component_is_foo` function if recipe contain a single `create_foo arg_foo` pair, `create_foo(arg_foo)` is returned instead of a composed component

27.40.5.42 `struct starpu_sched_tree * starpu_sched_component_make_scheduler ( unsigned sched_ctx_id, struct starpu_sched_component_specs s )`

this function build a scheduler for `sched_ctx_id` according to `s` and the hwloc topology of the machine.



## Chapter 28

# File Index

### 28.1 File List

Here is a list of all documented files with brief descriptions:

|                                 |           |
|---------------------------------|-----------|
| sc_hypervisor.h                 | 370       |
| sc_hypervisor_config.h          | 371       |
| sc_hypervisor_lp.h              | 372       |
| sc_hypervisor_monitoring.h      | 373       |
| sc_hypervisor_policy.h          | 373       |
| starpu.h                        | 335       |
| starpu_bitmap.h                 | 336       |
| starpu_bound.h                  | 336       |
| starpu_config.h                 | 337       |
| starpu_cublas.h                 | 338       |
| starpu_cuda.h                   | 338       |
| starpu_data.h                   | 339       |
| starpu_data_filters.h           | 341       |
| starpu_data_interfaces.h        | 342       |
| starpu_deprecated_api.h         | 346       |
| starpu_disk.h                   | 346       |
| starpu_driver.h                 | 347       |
| starpu_expert.h                 | 347       |
| starpu_fxt.h                    | 347       |
| starpu_hash.h                   | 348       |
| starpu_mic.h                    | 348       |
| starpu_mpi.h                    | 369       |
| starpu_opencl.h                 | 349       |
| starpu_openmp.h                 | 350       |
| starpu_perfmodel.h              | 352       |
| starpu_profiling.h              | 354       |
| starpu_rand.h                   | 355       |
| starpu_scc.h                    | 355       |
| <b>starpu_sched_component.h</b> | <b>??</b> |
| starpu_sched_ctx.h              | 356       |
| starpu_sched_ctx_hypervisor.h   | 357       |
| starpu_scheduler.h              | 358       |
| starpu_sink.h                   | 358       |
| starpu_stdlib.h                 | 359       |
| starpu_task.h                   | 359       |
| starpu_task_bundle.h            | 361       |
| starpu_task_list.h              | 361       |
| starpu_task_util.h              | 362       |

|                                |     |
|--------------------------------|-----|
| starpu_thread.h . . . . .      | 363 |
| starpu_thread_util.h . . . . . | 365 |
| starpu_top.h . . . . .         | 366 |
| starpu_tree.h . . . . .        | 367 |
| starpu_util.h . . . . .        | 367 |
| starpu_worker.h . . . . .      | 368 |
| starpufft.h . . . . .          | ??  |

## Chapter 29

# File Documentation

### 29.1 starpu.h File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <starpu_config.h>
#include <starpu_opengl.h>
#include <starpu_thread.h>
#include <starpu_thread_util.h>
#include <starpu_util.h>
#include <starpu_data.h>
#include <starpu_disk.h>
#include <starpu_data_interfaces.h>
#include <starpu_data_filters.h>
#include <starpu_stdlib.h>
#include <starpu_perfmodel.h>
#include <starpu_worker.h>
#include <starpu_task.h>
#include <starpu_task_list.h>
#include <starpu_task_util.h>
#include <starpu_sched_ctx.h>
#include <starpu_expert.h>
#include <starpu_rand.h>
#include <starpu_cuda.h>
#include <starpu_cublas.h>
#include <starpu_bound.h>
#include <starpu_hash.h>
#include <starpu_profiling.h>
#include <starpu_top.h>
#include <starpu_fxt.h>
#include <starpu_driver.h>
#include <starpu_tree.h>
#include <starpu_openmp.h>
#include <starpu_simgrid_wrap.h>
#include <starpu_bitmap.h>
#include "starpu_deprecated_api.h"
```

#### Data Structures

- struct [starpu\\_conf](#)

## Functions

- int [starpu\\_conf\\_init](#) (struct [starpu\\_conf](#) \*conf)
- int [starpu\\_init](#) (struct [starpu\\_conf](#) \*conf) [STARPU\\_WARN\\_UNUSED\\_RESULT](#)
- int [starpu\\_initialize](#) (struct [starpu\\_conf](#) \*user\_conf, int \*argc, char \*\*\*argv)
- void [starpu\\_pause](#) (void)
- void [starpu\\_resume](#) (void)
- void [starpu\\_shutdown](#) (void)
- void [starpu\\_topology\\_print](#) (FILE \*f)
- int [starpu\\_asynchronous\\_copy\\_disabled](#) (void)
- int [starpu\\_asynchronous\\_cuda\\_copy\\_disabled](#) (void)
- int [starpu\\_asynchronous\\_opengl\\_copy\\_disabled](#) (void)
- int [starpu\\_asynchronous\\_mic\\_copy\\_disabled](#) (void)
- void [starpu\\_display\\_stats](#) ()
- void [starpu\\_get\\_version](#) (int \*major, int \*minor, int \*release)

## 29.2 starpu\_bitmap.h File Reference

### Functions

- struct [starpu\\_bitmap](#) \* [starpu\\_bitmap\\_create](#) (void)
- void [starpu\\_bitmap\\_destroy](#) (struct [starpu\\_bitmap](#) \*b)
- void [starpu\\_bitmap\\_set](#) (struct [starpu\\_bitmap](#) \*b, int e)
- void [starpu\\_bitmap\\_unset](#) (struct [starpu\\_bitmap](#) \*b, int e)
- void [starpu\\_bitmap\\_unset\\_all](#) (struct [starpu\\_bitmap](#) \*b)
- int [starpu\\_bitmap\\_get](#) (struct [starpu\\_bitmap](#) \*b, int e)
- void [starpu\\_bitmap\\_unset\\_and](#) (struct [starpu\\_bitmap](#) \*a, struct [starpu\\_bitmap](#) \*b, struct [starpu\\_bitmap](#) \*c)
- void [starpu\\_bitmap\\_or](#) (struct [starpu\\_bitmap](#) \*a, struct [starpu\\_bitmap](#) \*b)
- int [starpu\\_bitmap\\_and\\_get](#) (struct [starpu\\_bitmap](#) \*b1, struct [starpu\\_bitmap](#) \*b2, int e)
- int [starpu\\_bitmap\\_cardinal](#) (struct [starpu\\_bitmap](#) \*b)
- int [starpu\\_bitmap\\_first](#) (struct [starpu\\_bitmap](#) \*b)
- int [starpu\\_bitmap\\_last](#) (struct [starpu\\_bitmap](#) \*b)
- int [starpu\\_bitmap\\_next](#) (struct [starpu\\_bitmap](#) \*b, int e)
- int [starpu\\_bitmap\\_has\\_next](#) (struct [starpu\\_bitmap](#) \*b, int e)

## 29.3 starpu\_bound.h File Reference

```
#include <stdio.h>
```

### Functions

- void [starpu\\_bound\\_start](#) (int deps, int prio)
- void [starpu\\_bound\\_stop](#) (void)
- void [starpu\\_bound\\_print\\_dot](#) (FILE \*output)
- void [starpu\\_bound\\_compute](#) (double \*res, double \*integer\_res, int integer)
- void [starpu\\_bound\\_print\\_lp](#) (FILE \*output)
- void [starpu\\_bound\\_print\\_mps](#) (FILE \*output)
- void [starpu\\_bound\\_print](#) (FILE \*output, int integer)

## 29.4 starpu\_config.h File Reference

```
#include <sys/types.h>
```

### Macros

- #define [STARPU\\_MAJOR\\_VERSION](#)
- #define [STARPU\\_MINOR\\_VERSION](#)
- #define [STARPU\\_RELEASE\\_VERSION](#)
- #define **STARPU\_USE\_CPU**
- #define [STARPU\\_USE\\_CUDA](#)
- #define [STARPU\\_USE\\_OPENCL](#)
- #define [STARPU\\_USE\\_MIC](#)
- #define [STARPU\\_USE\\_SCC](#)
- #define **STARPU\_SIMGRID**
- #define **STARPU\_SIMGRID\_HAVE\_XBT\_BARRIER\_INIT**
- #define **STARPU\_HAVE\_ICC**
- #define **STARPU\_ATLAS**
- #define **STARPU\_GOTO**
- #define **STARPU\_MKL**
- #define **STARPU\_SYSTEM\_BLAS**
- #define **STARPU\_BUILD\_DIR**
- #define [STARPU\\_OPENCL\\_DATADIR](#)
- #define **STARPU\_HAVE\_MAGMA**
- #define **STARPU\_OPENGL\_RENDER**
- #define **STARPU\_USE\_GTK**
- #define **STARPU\_HAVE\_X11**
- #define **STARPU\_HAVE\_POSIX\_MEMALIGN**
- #define **STARPU\_HAVE\_MEMALIGN**
- #define **STARPU\_HAVE\_MALLOC\_H**
- #define **STARPU\_HAVE\_SYNC\_BOOL\_COMPARE\_AND\_SWAP**
- #define **STARPU\_HAVE\_SYNC\_VAL\_COMPARE\_AND\_SWAP**
- #define **STARPU\_HAVE\_SYNC\_FETCH\_AND\_ADD**
- #define **STARPU\_HAVE\_SYNC\_FETCH\_AND\_OR**
- #define **STARPU\_HAVE\_SYNC\_LOCK\_TEST\_AND\_SET**
- #define **STARPU\_HAVE\_SYNC\_SYNCHRONIZE**
- #define **STARPU\_MODEL\_DEBUG**
- #define **STARPU\_NO\_ASSERT**
- #define **STARPU\_HAVE\_FFTW**
- #define **STARPU\_HAVE\_FFTWF**
- #define **STARPU\_HAVE\_FFTWL**
- #define **STARPU\_HAVE\_CURAND**
- #define **STARPU\_MAXNODES**
- #define [STARPU\\_NMAXBUFS](#)
- #define **STARPU\_MAXCPUS**
- #define [STARPU\\_MAXCUDADEVs](#)
- #define [STARPU\\_MAXOPENCLDEVs](#)
- #define [STARPU\\_MAXMICDEVs](#)
- #define [STARPU\\_MAXSCCDEVs](#)
- #define [STARPU\\_NMAXWORKERS](#)
- #define **STARPU\_NMAX\_SCHED\_CTXs**
- #define **STARPU\_MAXIMPLEMENTATIONS**
- #define **STARPU\_MAXMPKERNELS**

- `#define STARPU_USE_SC_HYPERVISOR`
- `#define STARPU_SC_HYPERVISOR_DEBUG`
- `#define STARPU_HAVE_GLPK_H`
- `#define STARPU_HAVE_LIBNUMA`
- `#define STARPU_HAVE_WINDOWS`
- `#define STARPU_LINUX_SYS`
- `#define STARPU_HAVE_UNSETENV`
- `#define __starpu_func__`
- `#define __starpu_inline`
- `#define STARPU_QUICK_CHECK`
- `#define STARPU_USE_DRAND48`
- `#define STARPU_USE_ERAND48_R`
- `#define STARPU_HAVE_NEARBYINTF`
- `#define STARPU_HAVE_RINTF`
- `#define STARPU_USE_TOP`
- `#define STARPU_HAVE_HWLOC`
- `#define STARPU_HAVE_PTHREAD_SPIN_LOCK`
- `#define STARPU_HAVE_PTHREAD_BARRIER`

#### Initialisation

- `#define STARPU_OPENMP`

#### Initialisation

- `#define STARPU_USE_MPI`

#### Typedefs

- `typedef ssize_t starpu_ssize_t`

## 29.5 starpu\_cublas.h File Reference

#### Functions

- void `starpu_cublas_init` (void)
- void `starpu_cublas_shutdown` (void)

## 29.6 starpu\_cuda.h File Reference

```
#include <starpu_config.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <cuda_runtime_api.h>
```

#### Macros

- `#define STARPU_CUBLAS_REPORT_ERROR(status)`
- `#define STARPU_CUDA_REPORT_ERROR(status)`



## Functions

- void **starpu\_cublas\_report\_error** (const char \*func, const char \*file, int line, int status)
- void **starpu\_cuda\_report\_error** (const char \*func, const char \*file, int line, cudaError\_t status)
- cudaStream\_t **starpu\_cuda\_get\_local\_stream** (void)
- const struct cudaDeviceProp \* **starpu\_cuda\_get\_device\_properties** (unsigned workerid)
- int **starpu\_cuda\_copy\_async\_sync** (void \*src\_ptr, unsigned src\_node, void \*dst\_ptr, unsigned dst\_node, size\_t ssize, cudaStream\_t stream, enum cudaMemcpyKind kind)
- void **starpu\_cuda\_set\_device** (unsigned devid)

## 29.7 starpu\_data.h File Reference

```
#include <starpu.h>
```

## Data Structures

- struct **starpu\_data\_descr**

## Macros

- #define **starpu\_data\_malloc\_pinned\_if\_possible**
- #define **starpu\_data\_free\_pinned\_if\_possible**
- #define **STARPU\_MAIN\_RAM**

## Typedefs

- typedef struct \_starpu\_data\_state \* **starpu\_data\_handle\_t**

## Enumerations

- enum **starpu\_data\_access\_mode** {  
STARPU\_NONE, STARPU\_R, STARPU\_W, STARPU\_RW,  
STARPU\_SCRATCH, STARPU\_REDUX, STARPU\_COMMUTE, STARPU\_SSEND }
- enum **starpu\_node\_kind** {  
STARPU\_UNUSED, STARPU\_CPU\_RAM, STARPU\_CUDA\_RAM, STARPU\_OPENCL\_RAM,  
STARPU\_DISK\_RAM, STARPU\_MIC\_RAM, STARPU\_SCC\_RAM, STARPU\_SCC\_SHM }

## Functions

- void **starpu\_data\_display\_memory\_stats** ()
- unsigned **starpu\_worker\_get\_memory\_node** (unsigned workerid)
- unsigned **starpu\_memory\_nodes\_get\_count** (void)
- enum **starpu\_node\_kind** **starpu\_node\_get\_kind** (unsigned node)
- void **starpu\_data\_set\_sequential\_consistency\_flag** (starpu\_data\_handle\_t handle, unsigned flag)
- unsigned **starpu\_data\_get\_sequential\_consistency\_flag** (starpu\_data\_handle\_t handle)
- unsigned **starpu\_data\_get\_default\_sequential\_consistency\_flag** (void)
- void **starpu\_data\_set\_default\_sequential\_consistency\_flag** (unsigned flag)
- unsigned **starpu\_data\_test\_if\_allocated\_on\_node** (starpu\_data\_handle\_t handle, unsigned memory\_node)
- void **starpu\_memchunk\_tidy** (unsigned memory\_node)

## Basic Data Management API

Data management is done at a high-level in StarPU: rather than accessing a mere list of contiguous buffers, the tasks may manipulate data that are described by a high-level construct which we call data interface.

An example of data interface is the "vector" interface which describes a contiguous data array on a specific memory node. This interface is a simple structure containing the number of elements in the array, the size of the elements, and the address of the array in the appropriate address space (this address may be invalid if there is no valid copy of the array in the memory node). More informations on the data interfaces provided by StarPU are given in [Data Interfaces](#).

When a piece of data managed by StarPU is used by a task, the task implementation is given a pointer to an interface describing a valid copy of the data that is accessible from the current processing unit.

Every worker is associated to a memory node which is a logical abstraction of the address space from which the processing unit gets its data. For instance, the memory node associated to the different CPU workers represents main memory (RAM), the memory node associated to a GPU is DRAM embedded on the device. Every memory node is identified by a logical index which is accessible from the function [starpu\\_worker\\_get\\_memory\\_node\(\)](#). When registering a piece of data to StarPU, the specified memory node indicates where the piece of data initially resides (we also call this memory node the home node of a piece of data).

- void [starpu\\_data\\_unregister](#) (starpu\_data\_handle\_t handle)
- void [starpu\\_data\\_unregister\\_no\\_coherency](#) (starpu\_data\_handle\_t handle)
- void [starpu\\_data\\_unregister\\_submit](#) (starpu\_data\_handle\_t handle)
- void [starpu\\_data\\_invalidate](#) (starpu\_data\_handle\_t handle)
- void [starpu\\_data\\_invalidate\\_submit](#) (starpu\_data\_handle\_t handle)
- void [starpu\\_data\\_advise\\_as\\_important](#) (starpu\_data\_handle\_t handle, unsigned is\_important)
- int [starpu\\_data\\_request\\_allocation](#) (starpu\_data\_handle\_t handle, unsigned node)
- int [starpu\\_data\\_prefetch\\_on\\_node](#) (starpu\_data\_handle\_t handle, unsigned node, unsigned async)
- void [starpu\\_data\\_set\\_wt\\_mask](#) (starpu\_data\_handle\_t handle, uint32\_t wt\_mask)
- void [starpu\\_data\\_query\\_status](#) (starpu\_data\_handle\_t handle, int memory\_node, int \*is\_allocated, int \*is\_valid, int \*is\_requested)
- void [starpu\\_data\\_set\\_reduction\\_methods](#) (starpu\_data\_handle\_t handle, struct [starpu\\_codelet](#) \*reduc\_cl, struct [starpu\\_codelet](#) \*init\_cl)
- struct [starpu\\_data\\_interface\\_ops](#) \* [starpu\\_data\\_get\\_interface\\_ops](#) (starpu\_data\_handle\_t handle)

## MPI Insert Task

- int [starpu\\_data\\_set\\_rank](#) (starpu\_data\_handle\_t handle, int rank)
- int [starpu\\_data\\_get\\_rank](#) (starpu\_data\_handle\_t handle)
- int [starpu\\_data\\_set\\_tag](#) (starpu\_data\_handle\_t handle, int tag)
- int [starpu\\_data\\_get\\_tag](#) (starpu\_data\_handle\_t handle)

## Access registered data from the application

- #define [STARPU\\_DATA\\_ACQUIRE\\_CB](#)(handle, mode, code)
- int [starpu\\_data\\_acquire](#) (starpu\_data\_handle\_t handle, enum [starpu\\_data\\_access\\_mode](#) mode)
- int [starpu\\_data\\_acquire\\_on\\_node](#) (starpu\_data\_handle\_t handle, int node, enum [starpu\\_data\\_access\\_mode](#) mode)
- int [starpu\\_data\\_acquire\\_cb](#) (starpu\_data\_handle\_t handle, enum [starpu\\_data\\_access\\_mode](#) mode, void(\*callback)(void \*), void \*arg)
- int [starpu\\_data\\_acquire\\_on\\_node\\_cb](#) (starpu\_data\_handle\_t handle, int node, enum [starpu\\_data\\_access\\_mode](#) mode, void(\*callback)(void \*), void \*arg)
- int [starpu\\_data\\_acquire\\_cb\\_sequential\\_consistency](#) (starpu\_data\_handle\_t handle, enum [starpu\\_data\\_access\\_mode](#) mode, void(\*callback)(void \*), void \*arg, int sequential\_consistency)
- int [starpu\\_data\\_acquire\\_on\\_node\\_cb\\_sequential\\_consistency](#) (starpu\_data\_handle\_t handle, int node, enum [starpu\\_data\\_access\\_mode](#) mode, void(\*callback)(void \*), void \*arg, int sequential\_consistency)
- void [starpu\\_data\\_release](#) (starpu\_data\_handle\_t handle)
- void [starpu\\_data\\_release\\_on\\_node](#) (starpu\_data\_handle\_t handle, int node)

## 29.8 starpu\_data\_filters.h File Reference

```
#include <starpu.h>
#include <stdarg.h>
```

### Data Structures

- struct [starpu\\_data\\_filter](#)

### Functions

#### Basic API

- void [starpu\\_data\\_partition](#) ([starpu\\_data\\_handle\\_t](#) initial\_handle, struct [starpu\\_data\\_filter](#) \*f)
- void [starpu\\_data\\_unpartition](#) ([starpu\\_data\\_handle\\_t](#) root\_data, unsigned gathering\_node)
- int [starpu\\_data\\_get\\_nb\\_children](#) ([starpu\\_data\\_handle\\_t](#) handle)
- [starpu\\_data\\_handle\\_t](#) [starpu\\_data\\_get\\_child](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned i)
- [starpu\\_data\\_handle\\_t](#) [starpu\\_data\\_get\\_sub\\_data](#) ([starpu\\_data\\_handle\\_t](#) root\_data, unsigned depth,...)
- [starpu\\_data\\_handle\\_t](#) [starpu\\_data\\_vget\\_sub\\_data](#) ([starpu\\_data\\_handle\\_t](#) root\_data, unsigned depth, va\_list pa)
- void [starpu\\_data\\_map\\_filters](#) ([starpu\\_data\\_handle\\_t](#) root\_data, unsigned nfilters,...)
- void [starpu\\_data\\_vmap\\_filters](#) ([starpu\\_data\\_handle\\_t](#) root\_data, unsigned nfilters, va\_list pa)

#### Predefined BCSR Filter Functions

This section gives a partial list of the predefined partitioning functions for BCSR data. Examples on how to use them are shown in [Partitioning Data](#). The complete list can be found in the file [starpu\\_data\\_filters.h](#).

- void [starpu\\_bcsr\\_filter\\_canonical\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_csr\\_filter\\_vertical\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)

#### Predefined Matrix Filter Functions

This section gives a partial list of the predefined partitioning functions for matrix data. Examples on how to use them are shown in [Partitioning Data](#). The complete list can be found in the file [starpu\\_data\\_filters.h](#).

- void [starpu\\_matrix\\_filter\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_matrix\\_filter\\_block\\_shadow](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_matrix\\_filter\\_vertical\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_matrix\\_filter\\_vertical\\_block\\_shadow](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)

#### Predefined Vector Filter Functions

This section gives a partial list of the predefined partitioning functions for vector data. Examples on how to use them are shown in [Partitioning Data](#). The complete list can be found in the file [starpu\\_data\\_filters.h](#).

- void [starpu\\_vector\\_filter\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_vector\\_filter\\_block\\_shadow](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_vector\\_filter\\_list](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_vector\\_filter\\_divide\\_in\\_2](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)

### Predefined Block Filter Functions

This section gives a partial list of the predefined partitioning functions for block data. Examples on how to use them are shown in [Partitioning Data](#). The complete list can be found in the file [starpu\\_data\\_filters.h](#). A usage example is available in [examples/filters/shadow3d.c](#)

- void [starpu\\_block\\_filter\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_block\\_filter\\_block\\_shadow](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_block\\_filter\\_vertical\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_block\\_filter\\_vertical\\_block\\_shadow](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_block\\_filter\\_depth\\_block](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)
- void [starpu\\_block\\_filter\\_depth\\_block\\_shadow](#) (void \*father\_interface, void \*child\_interface, struct [starpu\\_data\\_filter](#) \*f, unsigned id, unsigned nparts)

## 29.9 starpu\_data\_interfaces.h File Reference

```
#include <starpu.h>
#include <cuda_runtime.h>
```

### Data Structures

- struct [starpu\\_data\\_copy\\_methods](#)
- struct [starpu\\_data\\_interface\\_ops](#)
- struct [starpu\\_matrix\\_interface](#)
- struct [starpu\\_coo\\_interface](#)
- struct [starpu\\_block\\_interface](#)
- struct [starpu\\_vector\\_interface](#)
- struct [starpu\\_variable\\_interface](#)
- struct [starpu\\_csr\\_interface](#)
- struct [starpu\\_bcsr\\_interface](#)
- struct [starpu\\_multiformat\\_data\\_interface\\_ops](#)
- struct [starpu\\_multiformat\\_interface](#)

### Macros

- #define [STARPU\\_MULTIFORMAT\\_GET\\_CPU\\_PTR](#)(interface)
- #define [STARPU\\_MULTIFORMAT\\_GET\\_CUDA\\_PTR](#)(interface)
- #define [STARPU\\_MULTIFORMAT\\_GET\\_OPENCL\\_PTR](#)(interface)
- #define [STARPU\\_MULTIFORMAT\\_GET\\_MIC\\_PTR](#)(interface)
- #define [STARPU\\_MULTIFORMAT\\_GET\\_NX](#)(interface)

### Accessing COO Data Interfaces

- #define [STARPU\\_COO\\_GET\\_COLUMNS](#)(interface)
- #define [STARPU\\_COO\\_GET\\_COLUMNS\\_DEV\\_HANDLE](#)(interface)
- #define [STARPU\\_COO\\_GET\\_ROWS](#)(interface)
- #define [STARPU\\_COO\\_GET\\_ROWS\\_DEV\\_HANDLE](#)(interface)
- #define [STARPU\\_COO\\_GET\\_VALUES](#)(interface)
- #define [STARPU\\_COO\\_GET\\_VALUES\\_DEV\\_HANDLE](#)(interface)
- #define [STARPU\\_COO\\_GET\\_OFFSET](#)
- #define [STARPU\\_COO\\_GET\\_NX](#)(interface)
- #define [STARPU\\_COO\\_GET\\_NY](#)(interface)
- #define [STARPU\\_COO\\_GET\\_NVALUES](#)(interface)
- #define [STARPU\\_COO\\_GET\\_ELEMSIZE](#)(interface)

## Typedefs

- typedef cudaStream\_t **starpu\_cudaStream\_t**

## Enumerations

- enum **starpu\_data\_interface\_id** {  
[STARPU\\_UNKNOWN\\_INTERFACE\\_ID](#), [STARPU\\_MATRIX\\_INTERFACE\\_ID](#), [STARPU\\_BLOCK\\_INTERFACE\\_ID](#), [STARPU\\_VECTOR\\_INTERFACE\\_ID](#),  
[STARPU\\_CSR\\_INTERFACE\\_ID](#), [STARPU\\_BCSR\\_INTERFACE\\_ID](#), [STARPU\\_VARIABLE\\_INTERFACE\\_ID](#), [STARPU\\_VOID\\_INTERFACE\\_ID](#),  
[STARPU\\_MULTIFORMAT\\_INTERFACE\\_ID](#), [STARPU\\_COO\\_INTERFACE\\_ID](#), [STARPU\\_MAX\\_INTERFACE\\_ID](#) }

## Functions

- void [starpu\\_multiformat\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, void \*ptr, uint32\_t nobjects, struct [starpu\\_multiformat\\_data\\_interface\\_ops](#) \*format\_ops)

### Defining Interface

Applications can provide their own interface as shown in [Defining A New Data Interface](#).

- int [starpu\\_interface\\_copy](#) (uintptr\_t src, size\_t src\_offset, unsigned src\_node, uintptr\_t dst, size\_t dst\_offset, unsigned dst\_node, size\_t size, void \*async\_data)
- uintptr\_t [starpu\\_malloc\\_on\\_node](#) (unsigned dst\_node, size\_t size)
- void [starpu\\_free\\_on\\_node](#) (unsigned dst\_node, uintptr\_t addr, size\_t size)
- int [starpu\\_data\\_interface\\_get\\_next\\_id](#) (void)

### Basic Data Management API

Data management is done at a high-level in StarPU: rather than accessing a mere list of contiguous buffers, the tasks may manipulate data that are described by a high-level construct which we call data interface.

An example of data interface is the "vector" interface which describes a contiguous data array on a specific memory node. This interface is a simple structure containing the number of elements in the array, the size of the elements, and the address of the array in the appropriate address space (this address may be invalid if there is no valid copy of the array in the memory node). More informations on the data interfaces provided by StarPU are given in [Data Interfaces](#).

When a piece of data managed by StarPU is used by a task, the task implementation is given a pointer to an interface describing a valid copy of the data that is accessible from the current processing unit.

Every worker is associated to a memory node which is a logical abstraction of the address space from which the processing unit gets its data. For instance, the memory node associated to the different CPU workers represents main memory (RAM), the memory node associated to a GPU is DRAM embedded on the device. Every memory node is identified by a logical index which is accessible from the function [starpu\\_worker\\_get\\_memory\\_node\(\)](#). When registering a piece of data to StarPU, the specified memory node indicates where the piece of data initially resides (we also call this memory node the home node of a piece of data).

- void [starpu\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handleptr, unsigned home\_node, void \*data\_interface, struct [starpu\\_data\\_interface\\_ops](#) \*ops)
- void [starpu\\_data\\_ptr\\_register](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned node)
- void [starpu\\_data\\_register\\_same](#) ([starpu\\_data\\_handle\\_t](#) \*handledst, [starpu\\_data\\_handle\\_t](#) handlesrc)
- [starpu\\_data\\_handle\\_t](#) [starpu\\_data\\_lookup](#) (const void \*ptr)

### Accessing Data Interfaces

Each data interface is provided with a set of field access functions. The ones using a void \* parameter aimed to be used in codelet implementations (see for example the code in [Vector Scaling Using StarPU's API](#)).

- void \* [starpu\\_data\\_handle\\_to\\_pointer](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned node)

- void \* [starpu\\_data\\_get\\_local\\_ptr](#) ([starpu\\_data\\_handle\\_t](#) handle)
- enum [starpu\\_data\\_interface\\_id](#) [starpu\\_data\\_get\\_interface\\_id](#) ([starpu\\_data\\_handle\\_t](#) handle)
- int [starpu\\_data\\_pack](#) ([starpu\\_data\\_handle\\_t](#) handle, void \*\*ptr, [starpu\\_ssize\\_t](#) \*count)
- int [starpu\\_data\\_unpack](#) ([starpu\\_data\\_handle\\_t](#) handle, void \*ptr, [size\\_t](#) count)
- [size\\_t](#) [starpu\\_data\\_get\\_size](#) ([starpu\\_data\\_handle\\_t](#) handle)

## Registering Data

There are several ways to register a memory region so that it can be managed by StarPU. The functions below allow the registration of vectors, 2D matrices, 3D matrices as well as BCSR and CSR sparse matrices.

- void \* [starpu\\_data\\_get\\_interface\\_on\\_node](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned memory\_node)
- void [starpu\\_matrix\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, [uintptr\\_t](#) ptr, [uint32\\_t](#) ld, [uint32\\_t](#) nx, [uint32\\_t](#) ny, [size\\_t](#) elemsize)
- void [starpu\\_matrix\\_ptr\\_register](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned node, [uintptr\\_t](#) ptr, [uintptr\\_t](#) dev↵\_handle, [size\\_t](#) offset, [uint32\\_t](#) ld)
- void [starpu\\_coo\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handleptr, unsigned home\_node, [uint32\\_t](#) nx, [uint32\\_t](#) ny, [uint32\\_t](#) n\_values, [uint32\\_t](#) \*columns, [uint32\\_t](#) \*rows, [uintptr\\_t](#) values, [size\\_t](#) elemsize)
- void [starpu\\_block\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, [uintptr\\_t](#) ptr, [uint32\\_t](#) ldy, [uint32\\_t](#) ldz, [uint32\\_t](#) nx, [uint32\\_t](#) ny, [uint32\\_t](#) nz, [size\\_t](#) elemsize)
- void [starpu\\_block\\_ptr\\_register](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned node, [uintptr\\_t](#) ptr, [uintptr\\_t](#) dev↵\_handle, [size\\_t](#) offset, [uint32\\_t](#) ldy, [uint32\\_t](#) ldz)
- void [starpu\\_vector\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, [uintptr\\_t](#) ptr, [uint32\\_t](#) nx, [size\\_t](#) elemsize)
- void [starpu\\_vector\\_ptr\\_register](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned node, [uintptr\\_t](#) ptr, [uintptr\\_t](#) dev↵\_handle, [size\\_t](#) offset)
- void [starpu\\_variable\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, [uintptr\\_t](#) ptr, [size\\_t](#) size)
- void [starpu\\_variable\\_ptr\\_register](#) ([starpu\\_data\\_handle\\_t](#) handle, unsigned node, [uintptr\\_t](#) ptr, [uintptr\\_t](#) ↵\_dev\_handle, [size\\_t](#) offset)
- void [starpu\\_void\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle)
- void [starpu\\_csr\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, [uint32\\_t](#) nnz, [uint32\\_t](#) nrow, [uintptr\\_t](#) nzval, [uint32\\_t](#) \*colind, [uint32\\_t](#) \*rowptr, [uint32\\_t](#) firstentry, [size\\_t](#) elemsize)
- void [starpu\\_bcsr\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) \*handle, unsigned home\_node, [uint32\\_t](#) nnz, [uint32\\_t](#) nrow, [uintptr\\_t](#) nzval, [uint32\\_t](#) \*colind, [uint32\\_t](#) \*rowptr, [uint32\\_t](#) firstentry, [uint32\\_t](#) r, [uint32\\_t](#) c, [size\\_t](#) elemsize)

## Variables

- struct [starpu\\_data\\_interface\\_ops](#) [starpu\\_interface\\_matrix\\_ops](#)
- struct [starpu\\_data\\_interface\\_ops](#) [starpu\\_interface\\_coo\\_ops](#)
- struct [starpu\\_data\\_interface\\_ops](#) [starpu\\_interface\\_block\\_ops](#)
- struct [starpu\\_data\\_interface\\_ops](#) [starpu\\_interface\\_vector\\_ops](#)
- struct [starpu\\_data\\_interface\\_ops](#) [starpu\\_interface\\_variable\\_ops](#)
- struct [starpu\\_data\\_interface\\_ops](#) [starpu\\_interface\\_void\\_ops](#)
- struct [starpu\\_data\\_interface\\_ops](#) [starpu\\_interface\\_csr\\_ops](#)
- struct [starpu\\_data\\_interface\\_ops](#) [starpu\\_interface\\_bcsr\\_ops](#)

## Accessing Variable Data Interfaces

- #define [STARPU\\_VARIABLE\\_GET\\_PTR](#)(interface)
- #define [STARPU\\_VARIABLE\\_GET\\_OFFSET](#)(interface)
- #define [STARPU\\_VARIABLE\\_GET\\_ELEMSIZE](#)(interface)
- #define [STARPU\\_VARIABLE\\_GET\\_DEV\\_HANDLE](#)(interface)
- [size\\_t](#) [starpu\\_variable\\_get\\_elemsize](#) ([starpu\\_data\\_handle\\_t](#) handle)
- [uintptr\\_t](#) [starpu\\_variable\\_get\\_local\\_ptr](#) ([starpu\\_data\\_handle\\_t](#) handle)

### Accessing Vector Data Interfaces

- `#define STARPU_VECTOR_GET_PTR(interface)`
- `#define STARPU_VECTOR_GET_DEV_HANDLE(interface)`
- `#define STARPU_VECTOR_GET_OFFSET(interface)`
- `#define STARPU_VECTOR_GET_NX(interface)`
- `#define STARPU_VECTOR_GET_ELEMSIZE(interface)`
- `#define STARPU_VECTOR_GET_SLICE_BASE(interface)`
- `uint32_t starpu_vector_get_nx (starpu_data_handle_t handle)`
- `size_t starpu_vector_get_elemsize (starpu_data_handle_t handle)`
- `uintptr_t starpu_vector_get_local_ptr (starpu_data_handle_t handle)`

### Accessing Matrix Data Interfaces

- `#define STARPU_MATRIX_GET_PTR(interface)`
- `#define STARPU_MATRIX_GET_DEV_HANDLE(interface)`
- `#define STARPU_MATRIX_GET_OFFSET(interface)`
- `#define STARPU_MATRIX_GET_NX(interface)`
- `#define STARPU_MATRIX_GET_NY(interface)`
- `#define STARPU_MATRIX_GET_LD(interface)`
- `#define STARPU_MATRIX_GET_ELEMSIZE(interface)`
- `uint32_t starpu_matrix_get_nx (starpu_data_handle_t handle)`
- `uint32_t starpu_matrix_get_ny (starpu_data_handle_t handle)`
- `uint32_t starpu_matrix_get_local_ld (starpu_data_handle_t handle)`
- `uintptr_t starpu_matrix_get_local_ptr (starpu_data_handle_t handle)`
- `size_t starpu_matrix_get_elemsize (starpu_data_handle_t handle)`

### Accessing Block Data Interfaces

- `#define STARPU_BLOCK_GET_PTR(interface)`
- `#define STARPU_BLOCK_GET_DEV_HANDLE(interface)`
- `#define STARPU_BLOCK_GET_OFFSET(interface)`
- `#define STARPU_BLOCK_GET_NX(interface)`
- `#define STARPU_BLOCK_GET_NY(interface)`
- `#define STARPU_BLOCK_GET_NZ(interface)`
- `#define STARPU_BLOCK_GET_LDY(interface)`
- `#define STARPU_BLOCK_GET_LDZ(interface)`
- `#define STARPU_BLOCK_GET_ELEMSIZE(interface)`
- `uint32_t starpu_block_get_nx (starpu_data_handle_t handle)`
- `uint32_t starpu_block_get_ny (starpu_data_handle_t handle)`
- `uint32_t starpu_block_get_nz (starpu_data_handle_t handle)`
- `uint32_t starpu_block_get_local_ldy (starpu_data_handle_t handle)`
- `uint32_t starpu_block_get_local_ldz (starpu_data_handle_t handle)`
- `uintptr_t starpu_block_get_local_ptr (starpu_data_handle_t handle)`
- `size_t starpu_block_get_elemsize (starpu_data_handle_t handle)`



## Accessing BCSR Data Interfaces

- `#define STARPU_BCSR_GET_NNZ(interface)`
- `#define STARPU_BCSR_GET_NZVAL(interface)`
- `#define STARPU_BCSR_GET_NZVAL_DEV_HANDLE(interface)`
- `#define STARPU_BCSR_GET_COLIND(interface)`
- `#define STARPU_BCSR_GET_COLIND_DEV_HANDLE(interface)`
- `#define STARPU_BCSR_GET_ROWPTR(interface)`
- `#define STARPU_BCSR_GET_ROWPTR_DEV_HANDLE(interface)`
- `#define STARPU_BCSR_GET_OFFSET`
- `uint32_t starpu_bcsr_get_nnz (starpu_data_handle_t handle)`
- `uint32_t starpu_bcsr_get_nrow (starpu_data_handle_t handle)`
- `uint32_t starpu_bcsr_get_firstentry (starpu_data_handle_t handle)`
- `uintptr_t starpu_bcsr_get_local_nzval (starpu_data_handle_t handle)`
- `uint32_t * starpu_bcsr_get_local_colind (starpu_data_handle_t handle)`
- `uint32_t * starpu_bcsr_get_local_rowptr (starpu_data_handle_t handle)`
- `uint32_t starpu_bcsr_get_r (starpu_data_handle_t handle)`
- `uint32_t starpu_bcsr_get_c (starpu_data_handle_t handle)`
- `size_t starpu_bcsr_get_elemsize (starpu_data_handle_t handle)`

## Accessing CSR Data Interfaces

- `#define STARPU_CSR_GET_NNZ(interface)`
- `#define STARPU_CSR_GET_NROW(interface)`
- `#define STARPU_CSR_GET_NZVAL(interface)`
- `#define STARPU_CSR_GET_NZVAL_DEV_HANDLE(interface)`
- `#define STARPU_CSR_GET_COLIND(interface)`
- `#define STARPU_CSR_GET_COLIND_DEV_HANDLE(interface)`
- `#define STARPU_CSR_GET_ROWPTR(interface)`
- `#define STARPU_CSR_GET_ROWPTR_DEV_HANDLE(interface)`
- `#define STARPU_CSR_GET_OFFSET`
- `#define STARPU_CSR_GET_FIRSTENTRY(interface)`
- `#define STARPU_CSR_GET_ELEMSIZE(interface)`
- `uint32_t starpu_csr_get_nnz (starpu_data_handle_t handle)`
- `uint32_t starpu_csr_get_nrow (starpu_data_handle_t handle)`
- `uint32_t starpu_csr_get_firstentry (starpu_data_handle_t handle)`
- `uintptr_t starpu_csr_get_local_nzval (starpu_data_handle_t handle)`
- `uint32_t * starpu_csr_get_local_colind (starpu_data_handle_t handle)`
- `uint32_t * starpu_csr_get_local_rowptr (starpu_data_handle_t handle)`
- `size_t starpu_csr_get_elemsize (starpu_data_handle_t handle)`

## 29.10 starpu\_deprecated\_api.h File Reference

## 29.11 starpu\_disk.h File Reference

```
#include <sys/types.h>
#include <starpu_config.h>
```

### Data Structures

- struct `starpu_disk_ops`



## Functions

- void [starpu\\_disk\\_close](#) (unsigned node, void \*obj, size\_t size)
- void \* [starpu\\_disk\\_open](#) (unsigned node, void \*pos, size\_t size)
- int **starpu\_disk\_register** (struct [starpu\\_disk\\_ops](#) \*func, void \*parameter, starpu\_ssize\_t size)

## Variables

- struct [starpu\\_disk\\_ops](#) [starpu\\_disk\\_stdio\\_ops](#)
- struct [starpu\\_disk\\_ops](#) [starpu\\_disk\\_unistd\\_ops](#)
- struct [starpu\\_disk\\_ops](#) [starpu\\_disk\\_unistd\\_o\\_direct\\_ops](#)
- struct [starpu\\_disk\\_ops](#) [starpu\\_disk\\_leveldb\\_ops](#)

## 29.12 starpu\_driver.h File Reference

```
#include <starpu_config.h>
#include <starpu_openc1.h>
```

## Data Structures

- struct [starpu\\_driver](#)
- union [starpu\\_driver.id](#)

## Functions

- int [starpu\\_driver\\_run](#) (struct [starpu\\_driver](#) \*d)
- void [starpu\\_drivers\\_request\\_termination](#) (void)
- int [starpu\\_driver\\_init](#) (struct [starpu\\_driver](#) \*d)
- int [starpu\\_driver\\_run\\_once](#) (struct [starpu\\_driver](#) \*d)
- int [starpu\\_driver\\_deinit](#) (struct [starpu\\_driver](#) \*d)

## 29.13 starpu\_expert.h File Reference

```
#include <starpu.h>
```

## Functions

- void [starpu\\_wake\\_all\\_blocked\\_workers](#) (void)
- int [starpu\\_progression\\_hook\\_register](#) (unsigned(\*func)(void \*arg), void \*arg)
- void [starpu\\_progression\\_hook\\_deregister](#) (int hook\_id)

## 29.14 starpu\_fxt.h File Reference

```
#include <starpu_perfmodel.h>
```

## Data Structures

- struct [starpu\\_fxt\\_codelet\\_event](#)
- struct [starpu\\_fxt\\_options](#)

## Macros

- `#define STARPU_FXT_MAX_FILES`

## Functions

- void [starpu\\_fxt\\_options\\_init](#) (struct [starpu\\_fxt\\_options](#) \*options)
- void [starpu\\_fxt\\_generate\\_trace](#) (struct [starpu\\_fxt\\_options](#) \*options)
- void [starpu\\_fxt\\_start\\_profiling](#) (void)
- void [starpu\\_fxt\\_stop\\_profiling](#) (void)
- void [starpu\\_fxt\\_write\\_data\\_trace](#) (char \*filename\_in)
- void [starpu\\_fxt\\_trace\\_user\\_event](#) (unsigned long code)

## 29.15 starpu\_hash.h File Reference

```
#include <stdint.h>
#include <stddef.h>
```

## Functions

### Defining Interface

*Applications can provide their own interface as shown in [Defining A New Data Interface](#).*

- uint32\_t [starpu\\_hash\\_crc32c\\_be\\_n](#) (const void \*input, size\_t n, uint32\_t inputcrc)
- uint32\_t [starpu\\_hash\\_crc32c\\_be](#) (uint32\_t input, uint32\_t inputcrc)
- uint32\_t [starpu\\_hash\\_crc32c\\_string](#) (const char \*str, uint32\_t inputcrc)

## 29.16 starpu\_mic.h File Reference

```
#include <starpu_config.h>
```

## Typedefs

- typedef void \* [starpu\\_mic\\_func\\_symbol\\_t](#)

## Functions

- int [starpu\\_mic\\_register\\_kernel](#) ([starpu\\_mic\\_func\\_symbol\\_t](#) \*symbol, const char \*func\_name)
- [starpu\\_mic\\_kernel\\_t](#) [starpu\\_mic\\_get\\_kernel](#) ([starpu\\_mic\\_func\\_symbol\\_t](#) symbol)

## 29.17 starpu\_opengl.h File Reference

```
#include <starpu_config.h>
#include <CL/cl.h>
#include <assert.h>
```

### Data Structures

- struct [starpu\\_opengl\\_program](#)

### Functions

#### Writing OpenCL kernels

- void [starpu\\_opengl\\_get\\_context](#) (int devid, cl\_context \*context)
- void [starpu\\_opengl\\_get\\_device](#) (int devid, cl\_device\_id \*device)
- void [starpu\\_opengl\\_get\\_queue](#) (int devid, cl\_command\_queue \*queue)
- void [starpu\\_opengl\\_get\\_current\\_context](#) (cl\_context \*context)
- void [starpu\\_opengl\\_get\\_current\\_queue](#) (cl\_command\_queue \*queue)
- int [starpu\\_opengl\\_set\\_kernel\\_args](#) (cl\_int \*err, cl\_kernel \*kernel,...)

#### Compiling OpenCL kernels

Source codes for OpenCL kernels can be stored in a file or in a string. StarPU provides functions to build the program executable for each available OpenCL device as a `cl_program` object. This program executable can then be loaded within a specific queue as explained in the next section. These are only helpers, Applications can also fill a [starpu\\_opengl\\_program](#) array by hand for more advanced use (e.g. different programs on the different OpenCL devices, for relocation purpose for instance).

- void [starpu\\_opengl\\_load\\_program\\_source](#) (const char \*source\_file\_name, char \*located\_file\_name, char \*located\_dir\_name, char \*opengl\_program\_source)
- int [starpu\\_opengl\\_compile\\_opengl\\_from\\_file](#) (const char \*source\_file\_name, const char \*build\_options)
- int [starpu\\_opengl\\_compile\\_opengl\\_from\\_string](#) (const char \*opengl\_program\_source, const char \*file\_name, const char \*build\_options)
- int [starpu\\_opengl\\_load\\_binary\\_opengl](#) (const char \*kernel\_id, struct [starpu\\_opengl\\_program](#) \*opengl\_programs)
- int [starpu\\_opengl\\_load\\_opengl\\_from\\_file](#) (const char \*source\_file\_name, struct [starpu\\_opengl\\_program](#) \*opengl\_programs, const char \*build\_options)
- int [starpu\\_opengl\\_load\\_opengl\\_from\\_string](#) (const char \*opengl\_program\_source, struct [starpu\\_opengl\\_program](#) \*opengl\_programs, const char \*build\_options)
- int [starpu\\_opengl\\_unload\\_opengl](#) (struct [starpu\\_opengl\\_program](#) \*opengl\_programs)

#### Loading OpenCL kernels

- int [starpu\\_opengl\\_load\\_kernel](#) (cl\_kernel \*kernel, cl\_command\_queue \*queue, struct [starpu\\_opengl\\_program](#) \*opengl\_programs, const char \*kernel\_name, int devid)
- int [starpu\\_opengl\\_release\\_kernel](#) (cl\_kernel kernel)

#### OpenCL statistics

- int [starpu\\_opengl\\_collect\\_stats](#) (cl\_event event)

## OpenCL utilities

- `#define STARPU_OPENCL_DISPLAY_ERROR(status)`
- `#define STARPU_OPENCL_REPORT_ERROR(status)`
- `#define STARPU_OPENCL_REPORT_ERROR_WITH_MSG(msg, status)`
- `const char * starpu_openccl_error_string (cl_int status)`
- `void starpu_openccl_display_error (const char *func, const char *file, int line, const char *msg, cl_int status)`
- `static __starpu_inline void starpu_openccl_report_error (const char *func, const char *file, int line, const char *msg, cl_int status)`
- `cl_int starpu_openccl_allocate_memory (int devid, cl_mem *addr, size_t size, cl_mem_flags flags)`
- `cl_int starpu_openccl_copy_ram_to_openccl (void *ptr, unsigned src_node, cl_mem buffer, unsigned dst_node, size_t size, size_t offset, cl_event *event, int *ret)`
- `cl_int starpu_openccl_copy_openccl_to_ram (cl_mem buffer, unsigned src_node, void *ptr, unsigned dst_node, size_t size, size_t offset, cl_event *event, int *ret)`
- `cl_int starpu_openccl_copy_openccl_to_openccl (cl_mem src, unsigned src_node, size_t src_offset, cl_mem dst, unsigned dst_node, size_t dst_offset, size_t size, cl_event *event, int *ret)`
- `cl_int starpu_openccl_copy_async_sync (uintptr_t src, size_t src_offset, unsigned src_node, uintptr_t dst, size_t dst_offset, unsigned dst_node, size_t size, cl_event *event)`

## 29.18 starpu\_openmp.h File Reference

```
#include <starpu_config.h>
```

### Data Structures

- struct `starpu_omp_lock_t`
- struct `starpu_omp_nest_lock_t`
- struct `starpu_omp_parallel_region_attr`
- struct `starpu_omp_task_region_attr`

### Macros

- `#define __STARPU_OMP_NOTHROW`

### Enumerations

- enum `starpu_omp_sched_value` {  
`starpu_omp_sched_undefined`, `starpu_omp_sched_static`, `starpu_omp_sched_dynamic`, `starpu_omp_↵`  
`sched_guided`,  
`starpu_omp_sched_auto`, `starpu_omp_sched_runtime` }
- enum `starpu_omp_proc_bind_value` {  
`starpu_omp_proc_bind_undefined`, `starpu_omp_proc_bind_false`, `starpu_omp_proc_bind_true`, `starpu_↵`  
`omp_proc_bind_master`,  
`starpu_omp_proc_bind_close`, `starpu_omp_proc_bind_spread` }

### Functions

#### Initialisation

- int `starpu_omp_init` (void) `__STARPU_OMP_NOTHROW`
- void `starpu_omp_shutdown` (void) `__STARPU_OMP_NOTHROW`

### Parallel

- void [starpu\\_omp\\_parallel\\_region](#) (const struct [starpu\\_omp\\_parallel\\_region\\_attr](#) \*attr) \_\_STARPU\_OMP\_N←  
\_NOTHROW
- void [starpu\\_omp\\_master](#) (void(\*f)(void \*arg), void \*arg) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_master\\_inline](#) (void) \_\_STARPU\_OMP\_NOTHROW

### Synchronization

- void [starpu\\_omp\\_barrier](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_critical](#) (void(\*f)(void \*arg), void \*arg, const char \*name) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_critical\\_inline\\_begin](#) (const char \*name) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_critical\\_inline\\_end](#) (const char \*name) \_\_STARPU\_OMP\_NOTHROW

### Worksharing

- void [starpu\\_omp\\_single](#) (void(\*f)(void \*arg), void \*arg, int nowait) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_single\\_inline](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_single\\_copyprivate](#) (void(\*f)(void \*arg, void \*data, unsigned long long data\_size), void  
\*arg, void \*data, unsigned long long data\_size) \_\_STARPU\_OMP\_NOTHROW
- void \* [starpu\\_omp\\_single\\_copyprivate\\_inline\\_begin](#) (void \*data) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_single\\_copyprivate\\_inline\\_end](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_for](#) (void(\*f)(unsigned long long \_first\_i, unsigned long long \_nb\_i, void \*arg), void \*arg,  
unsigned long long nb\_iterations, unsigned long long chunk, int schedule, int ordered, int nowait) \_\_STA←  
RPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_for\\_inline\\_first](#) (unsigned long long nb\_iterations, unsigned long long chunk, int schedule,  
int ordered, unsigned long long \*\_first\_i, unsigned long long \*\_nb\_i) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_for\\_inline\\_next](#) (unsigned long long nb\_iterations, unsigned long long chunk, int schedule,  
int ordered, unsigned long long \*\_first\_i, unsigned long long \*\_nb\_i) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_for\\_alt](#) (void(\*f)(unsigned long long \_begin\_i, unsigned long long \_end\_i, void \*arg), void  
\*arg, unsigned long long nb\_iterations, unsigned long long chunk, int schedule, int ordered, int nowait) ←  
\_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_for\\_inline\\_first\\_alt](#) (unsigned long long nb\_iterations, unsigned long long chunk, int sched-  
ule, int ordered, unsigned long long \*\_begin\_i, unsigned long long \*\_end\_i) \_\_STARPU\_OMP\_NOTHR←  
OW
- int [starpu\\_omp\\_for\\_inline\\_next\\_alt](#) (unsigned long long nb\_iterations, unsigned long long chunk, int sched-  
ule, int ordered, unsigned long long \*\_begin\_i, unsigned long long \*\_end\_i) \_\_STARPU\_OMP\_NOTHR←  
OW
- void [starpu\\_omp\\_ordered\\_inline\\_begin](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_ordered\\_inline\\_end](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_ordered](#) (void(\*f)(void \*arg), void \*arg) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_sections](#) (unsigned long long nb\_sections, void(\*\*section\_f)(void \*arg), void \*\*section←  
\_arg, int nowait) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_sections\\_combined](#) (unsigned long long nb\_sections, void(\*section\_f)(unsigned long  
long section\_num, void \*arg), void \*section\_arg, int nowait) \_\_STARPU\_OMP\_NOTHROW

### Task

- void [starpu\\_omp\\_task\\_region](#) (const struct [starpu\\_omp\\_task\\_region\\_attr](#) \*attr) \_\_STARPU\_OMP\_NOT←  
HROW
- void [starpu\\_omp\\_taskwait](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_taskgroup](#) (void(\*f)(void \*arg), void \*arg) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_taskgroup\\_inline\\_begin](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_taskgroup\\_inline\\_end](#) (void) \_\_STARPU\_OMP\_NOTHROW

### API

- void [starpu\\_omp\\_set\\_num\\_threads](#) (int threads) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_num\\_threads](#) () \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_thread\\_num](#) () \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_max\\_threads](#) () \_\_STARPU\_OMP\_NOTHROW

- int [starpu\\_omp\\_get\\_num\\_procs](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_in\\_parallel](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_dynamic](#) (int dynamic\_threads) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_dynamic](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_nested](#) (int nested) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_nested](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_cancellation](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_schedule](#) (enum [starpu\\_omp\\_sched\\_value](#) kind, int modifier) \_\_STARPU\_OMP\_↵  
NOTHROW
- void [starpu\\_omp\\_get\\_schedule](#) (enum [starpu\\_omp\\_sched\\_value](#) \*kind, int \*modifier) \_\_STARPU\_OM↵  
P\_NOTHROW
- int [starpu\\_omp\\_get\\_thread\\_limit](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_max\\_active\\_levels](#) (int max\_levels) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_max\\_active\\_levels](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_level](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_ancestor\\_thread\\_num](#) (int level) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_team\\_size](#) (int level) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_active\\_level](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_in\\_final](#) (void) \_\_STARPU\_OMP\_NOTHROW
- enum [starpu\\_omp\\_proc\\_bind\\_value](#) [starpu\\_omp\\_get\\_proc\\_bind](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_default\\_device](#) (int device\_num) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_default\\_device](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_num\\_devices](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_num\\_teams](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_get\\_team\\_num](#) (void) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_is\\_initial\\_device](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_init\\_lock](#) ([starpu\\_omp\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_destroy\\_lock](#) ([starpu\\_omp\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_lock](#) ([starpu\\_omp\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_unset\\_lock](#) ([starpu\\_omp\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_test\\_lock](#) ([starpu\\_omp\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_init\\_nest\\_lock](#) ([starpu\\_omp\\_nest\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_destroy\\_nest\\_lock](#) ([starpu\\_omp\\_nest\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_set\\_nest\\_lock](#) ([starpu\\_omp\\_nest\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_unset\\_nest\\_lock](#) ([starpu\\_omp\\_nest\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- int [starpu\\_omp\\_test\\_nest\\_lock](#) ([starpu\\_omp\\_nest\\_lock\\_t](#) \*lock) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_atomic\\_fallback\\_inline\\_begin](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_atomic\\_fallback\\_inline\\_end](#) (void) \_\_STARPU\_OMP\_NOTHROW
- double [starpu\\_omp\\_get\\_wtime](#) (void) \_\_STARPU\_OMP\_NOTHROW
- double [starpu\\_omp\\_get\\_wtick](#) (void) \_\_STARPU\_OMP\_NOTHROW
- void [starpu\\_omp\\_vector\\_annotate](#) ([starpu\\_data\\_handle\\_t](#) handle, uint32\_t slice\_base) \_\_STARPU\_OM↵  
P\_NOTHROW

## 29.19 [starpu\\_perfmodel.h](#) File Reference

```
#include <starpu.h>
#include <stdio.h>
#include <starpu_util.h>
#include <starpu_worker.h>
```

### Data Structures

- struct [starpu\\_perfmodel\\_device](#)
- struct [starpu\\_perfmodel\\_arch](#)
- struct [starpu\\_perfmodel\\_history\\_entry](#)
- struct [starpu\\_perfmodel\\_history\\_list](#)
- struct [starpu\\_perfmodel\\_regression\\_model](#)
- struct [starpu\\_perfmodel\\_per\\_arch](#)
- struct [starpu\\_perfmodel](#)

## Macros

- `#define STARPU_NARCH`
- `#define starpu_per_arch_perfmodel`

## Typedefs

- `typedef double(* starpu_perfmodel_per_arch_cost_function )(struct starpu_task *task, struct starpu_perfmodel_arch *arch, unsigned nimpl)`
- `typedef size_t(* starpu_perfmodel_per_arch_size_base )(struct starpu_task *task, struct starpu_perfmodel_arch *arch, unsigned nimpl)`
- `typedef struct _starpu_perfmodel_state * starpu_perfmodel_state_t`

## Enumerations

- `enum starpu_perfmodel_type {  
    STARPU_PERFMODEL_INVALID, STARPU_PER_ARCH, STARPU_COMMON, STARPU_HISTORY_BASED,  
    STARPU_REGRESSION_BASED, STARPU_NL_REGRESSION_BASED }`

## Functions

- `void starpu_perfmodel_init (FILE *f, struct starpu_perfmodel *model)`
- `void starpu_perfmodel_free_sampling_directories (void)`
- `struct starpu_perfmodel_arch * starpu_worker_get_perf_archtype (int workerid, unsigned sched_ctx_id)`
- `int starpu_perfmodel_load_symbol (const char *symbol, struct starpu_perfmodel *model)`
- `int starpu_perfmodel_unload_model (struct starpu_perfmodel *model)`
- `int starpu_get_narch_combs ()`
- `int starpu_perfmodel_arch_comb_add (int ndevices, struct starpu_perfmodel_device *devices)`
- `int starpu_perfmodel_arch_comb_get (int ndevices, struct starpu_perfmodel_device *devices)`
- `struct starpu_perfmodel_per_arch * starpu_perfmodel_get_model_per_arch (struct starpu_perfmodel *model, struct starpu_perfmodel_arch *arch, unsigned impl)`
- `struct starpu_perfmodel_per_arch * starpu_perfmodel_get_model_per_devices (struct starpu_perfmodel *model, int impl,...)`
- `int starpu_perfmodel_set_per_devices_cost_function (struct starpu_perfmodel *model, int impl, starpu_perfmodel_per_arch_cost_function func,...)`
- `int starpu_perfmodel_set_per_devices_size_base (struct starpu_perfmodel *model, int impl, starpu_perfmodel_per_arch_size_base func,...)`
- `void starpu_perfmodel_debugfilepath (struct starpu_perfmodel *model, struct starpu_perfmodel_arch *arch, char *path, size_t maxlen, unsigned nimpl)`
- `char * starpu_perfmodel_get_archtype_name (enum starpu_worker_archtype archtype)`
- `void starpu_perfmodel_get_arch_name (struct starpu_perfmodel_arch *arch, char *archname, size_t maxlen, unsigned nimpl)`
- `double starpu_perfmodel_history_based_expected_perf (struct starpu_perfmodel *model, struct starpu_perfmodel_arch *arch, uint32_t footprint)`
- `int starpu_perfmodel_list (FILE *output)`
- `void starpu_perfmodel_print (struct starpu_perfmodel *model, struct starpu_perfmodel_arch *arch, unsigned nimpl, char *parameter, uint32_t *footprint, FILE *output)`
- `int starpu_perfmodel_print_all (struct starpu_perfmodel *model, char *arch, char *parameter, uint32_t *footprint, FILE *output)`
- `int starpu_perfmodel_list_combs (FILE *output, struct starpu_perfmodel *model)`
- `void starpu_perfmodel_update_history (struct starpu_perfmodel *model, struct starpu_task *task, struct starpu_perfmodel_arch *arch, unsigned cpuid, unsigned nimpl, double measured)`

- void [starpu\\_perfmodel\\_directory](#) (FILE \*output)
- void [starpu\\_bus\\_print\\_bandwidth](#) (FILE \*f)
- void [starpu\\_bus\\_print\\_affinity](#) (FILE \*f)
- double [starpu\\_transfer\\_bandwidth](#) (unsigned src\_node, unsigned dst\_node)
- double [starpu\\_transfer\\_latency](#) (unsigned src\_node, unsigned dst\_node)
- double [starpu\\_transfer\\_predict](#) (unsigned src\_node, unsigned dst\_node, size\_t size)

## 29.19.1 Data Structure Documentation

### 29.19.1.1 struct starpu\_perfmodel\_device

#### Data Fields

|                                             |        |  |
|---------------------------------------------|--------|--|
| enum <a href="#">starpu_worker_archtype</a> | type   |  |
| int                                         | devid  |  |
| int                                         | ncores |  |

### 29.19.1.2 struct starpu\_perfmodel\_arch

#### Data Fields

|                                                  |          |  |
|--------------------------------------------------|----------|--|
| int                                              | ndevices |  |
| struct <a href="#">starpu_perfmodel_device</a> * | devices  |  |

## 29.20 starpu\_profiling.h File Reference

```
#include <starpu.h>
#include <errno.h>
#include <time.h>
#include <starpu_util.h>
```

### Data Structures

- struct [starpu\\_profiling\\_task\\_info](#)
- struct [starpu\\_profiling\\_worker\\_info](#)
- struct [starpu\\_profiling\\_bus\\_info](#)

### Macros

- #define [STARPU\\_PROFILING\\_DISABLE](#)
- #define [STARPU\\_PROFILING\\_ENABLE](#)
- #define [starpu\\_timespec\\_cmp](#)(a, b, CMP)



## Functions

- void [starpu\\_profiling\\_init](#) (void)
- void [starpu\\_profiling\\_set\\_id](#) (int new\_id)
- int [starpu\\_profiling\\_status\\_set](#) (int status)
- int [starpu\\_profiling\\_status\\_get](#) (void)
- int [starpu\\_profiling\\_worker\\_get\\_info](#) (int workerid, struct [starpu\\_profiling\\_worker\\_info](#) \*worker\_info)
- int [starpu\\_bus\\_get\\_count](#) (void)
- int [starpu\\_bus\\_get\\_id](#) (int src, int dst)
- int [starpu\\_bus\\_get\\_src](#) (int busid)
- int [starpu\\_bus\\_get\\_dst](#) (int busid)
- int [starpu\\_bus\\_get\\_profiling\\_info](#) (int busid, struct [starpu\\_profiling\\_bus\\_info](#) \*bus\_info)
- static \_\_starpu\_inline void **starpu\_timespec\_clear** (struct timespec \*tsp)
- static \_\_starpu\_inline void **starpu\_timespec\_add** (struct timespec \*a, struct timespec \*b, struct timespec \*result)
- static \_\_starpu\_inline void **starpu\_timespec\_accumulate** (struct timespec \*result, struct timespec \*a)
- static \_\_starpu\_inline void **starpu\_timespec\_sub** (const struct timespec \*a, const struct timespec \*b, struct timespec \*result)
- double [starpu\\_timing\\_timespec\\_delay\\_us](#) (struct timespec \*start, struct timespec \*end)
- double [starpu\\_timing\\_timespec\\_to\\_us](#) (struct timespec \*ts)
- void [starpu\\_profiling\\_bus\\_helper\\_display\\_summary](#) (void)
- void [starpu\\_profiling\\_worker\\_helper\\_display\\_summary](#) (void)

## 29.21 starpu\_rand.h File Reference

```
#include <stdlib.h>
#include <starpu_config.h>
```

## Macros

- #define **starpu\_srand48**(seed)
- #define **starpu\_drand48**()
- #define **starpu\_erand48**(xsubi)
- #define **starpu\_srand48\_r**(seed, buffer)
- #define **starpu\_erand48\_r**(xsubi, buffer, result)

## Typedefs

- typedef int **starpu\_drand48\_data**

## 29.22 starpu\_scc.h File Reference

```
#include <starpu_config.h>
```

## Typedefs

- typedef void \* [starpu\\_scc\\_func\\_symbol\\_t](#)

## Functions

- int [starpu\\_scc\\_register\\_kernel](#) ([starpu\\_scc\\_func\\_symbol\\_t](#) \*symbol, const char \*func\_name)
- [starpu\\_scc\\_kernel\\_t](#) [starpu\\_scc\\_get\\_kernel](#) ([starpu\\_scc\\_func\\_symbol\\_t](#) symbol)

## 29.23 starpu\_sched\_ctx.h File Reference

```
#include <starpu.h>
```

## Macros

- #define **STARPU\_SCHED\_CTX\_HIERARCHY\_LEVEL**
- #define **STARPU\_SCHED\_CTX\_NESTED**
- #define **STARPU\_SCHED\_CTX\_AWAKE\_WORKERS**

## Functions

- unsigned [starpu\\_sched\\_ctx\\_get\\_inheritor](#) (unsigned sched\_ctx\_id)
- unsigned [starpu\\_sched\\_ctx\\_get\\_hierarchy\\_level](#) (unsigned sched\_ctx\_id)
- unsigned [starpu\\_sched\\_ctx\\_contains\\_type\\_of\\_worker](#) (enum [starpu\\_worker\\_archtype](#) arch, unsigned sched\_ctx\_id)
- int [starpu\\_sched\\_get\\_min\\_priority](#) (void)
- int [starpu\\_sched\\_get\\_max\\_priority](#) (void)
- int [starpu\\_sched\\_set\\_min\\_priority](#) (int min\_prio)
- int [starpu\\_sched\\_set\\_max\\_priority](#) (int max\_prio)
- void [starpu\\_sched\\_ctx\\_set\\_priority](#) (int \*workers, int nworkers, unsigned sched\_ctx\_id, unsigned priority)
- void [starpu\\_sched\\_ctx\\_set\\_priority\\_on\\_level](#) (int \*workers\_to\_add, unsigned nworkers\_to\_add, unsigned sched\_ctx, unsigned priority)
- unsigned [starpu\\_sched\\_ctx\\_get\\_priority](#) (int worker, unsigned sched\_ctx\_id)
- void [starpu\\_sched\\_ctx\\_get\\_available\\_cpuids](#) (unsigned sched\_ctx\_id, int \*\*cpuids, int \*ncpuids)
- void [starpu\\_sched\\_ctx\\_bind\\_current\\_thread\\_to\\_cpuid](#) (unsigned cpuid)
- int [starpu\\_sched\\_ctx\\_book\\_workers\\_for\\_task](#) (unsigned sched\_ctx\_id, int \*workerids, int nworkers)
- void [starpu\\_sched\\_ctx\\_unbook\\_workers\\_for\\_task](#) (unsigned sched\_ctx\_id, int master)
- unsigned [starpu\\_sched\\_ctx\\_worker\\_is\\_master\\_for\\_child\\_ctx](#) (int workerid, unsigned sched\_ctx\_id)
- void [starpu\\_sched\\_ctx\\_revert\\_task\\_counters](#) (unsigned sched\_ctx\_id, double flops)
- void [starpu\\_sched\\_ctx\\_move\\_task\\_to\\_ctx](#) (struct [starpu\\_task](#) \*task, unsigned sched\_ctx)
- int [starpu\\_sched\\_ctx\\_get\\_worker\\_rank](#) (unsigned sched\_ctx\_id)

### Scheduling Context Worker Collection

- struct [starpu\\_worker\\_collection](#) \* [starpu\\_sched\\_ctx\\_create\\_worker\\_collection](#) (unsigned sched\_ctx\_id, enum [starpu\\_worker\\_collection\\_type](#) type)
- void [starpu\\_sched\\_ctx\\_delete\\_worker\\_collection](#) (unsigned sched\_ctx\_id)
- struct [starpu\\_worker\\_collection](#) \* [starpu\\_sched\\_ctx\\_get\\_worker\\_collection](#) (unsigned sched\_ctx\_id)

### Scheduling Context Link with Hypervisor

- void [starpu\\_sched\\_ctx\\_set\\_policy\\_data](#) (unsigned sched\_ctx\_id, void \*policy\_data)
- void \* [starpu\\_sched\\_ctx\\_get\\_policy\\_data](#) (unsigned sched\_ctx\_id)
- void \* [starpu\\_sched\\_ctx\\_exec\\_parallel\\_code](#) (void \*(\*func)(void \*), void \*param, unsigned sched\_ctx\_id)
- int [starpu\\_sched\\_ctx\\_get\\_nready\\_tasks](#) (unsigned sched\_ctx\_id)
- double [starpu\\_sched\\_ctx\\_get\\_nready\\_flops](#) (unsigned sched\_ctx\_id)
- void [starpu\\_sched\\_ctx\\_call\\_pushed\\_task\\_cb](#) (int workerid, unsigned sched\_ctx\_id)

## Scheduling Contexts Basic API

- `#define STARPU_SCHED_CTX_POLICY_NAME`
- `#define STARPU_SCHED_CTX_POLICY_STRUCT`
- `#define STARPU_SCHED_CTX_POLICY_MIN_PRIO`
- `#define STARPU_SCHED_CTX_POLICY_MAX_PRIO`
- `unsigned starpu_sched_ctx_create` (int \*workerids\_ctx, int nworkers\_ctx, const char \*sched\_ctx\_name,...)
- `unsigned starpu_sched_ctx_create_inside_interval` (const char \*policy\_name, const char \*sched\_ctx\_name, int min\_ncpus, int max\_ncpus, int min\_ngpus, int max\_ngpus, unsigned allow\_overlap)
- `void starpu_sched_ctx_register_close_callback` (unsigned sched\_ctx\_id, void(\*close\_callback)(unsigned sched\_ctx\_id, void \*args), void \*args)
- `void starpu_sched_ctx_add_workers` (int \*workerids\_ctx, int nworkers\_ctx, unsigned sched\_ctx\_id)
- `void starpu_sched_ctx_remove_workers` (int \*workerids\_ctx, int nworkers\_ctx, unsigned sched\_ctx\_id)
- `void starpu_sched_ctx_display_workers` (unsigned sched\_ctx\_id, FILE \*f)
- `void starpu_sched_ctx_delete` (unsigned sched\_ctx\_id)
- `void starpu_sched_ctx_set_inheritor` (unsigned sched\_ctx\_id, unsigned inheritor)
- `void starpu_sched_ctx_set_context` (unsigned \*sched\_ctx\_id)
- `unsigned starpu_sched_ctx_get_context` (void)
- `void starpu_sched_ctx_stop_task_submission` (void)
- `void starpu_sched_ctx_finished_submit` (unsigned sched\_ctx\_id)
- `unsigned starpu_sched_ctx_get_workers_list` (unsigned sched\_ctx\_id, int \*\*workerids)
- `unsigned starpu_sched_ctx_get_nworkers` (unsigned sched\_ctx\_id)
- `unsigned starpu_sched_ctx_get_nshared_workers` (unsigned sched\_ctx\_id, unsigned sched\_ctx\_id2)
- `unsigned starpu_sched_ctx_contains_worker` (int workerid, unsigned sched\_ctx\_id)
- `unsigned starpu_sched_ctx_worker_get_id` (unsigned sched\_ctx\_id)
- `unsigned starpu_sched_ctx_overlapping_ctxs_on_worker` (int workerid)

## Scheduling Context Priorities

- `#define STARPU_MIN_PRIO`
- `#define STARPU_MAX_PRIO`
- `#define STARPU_DEFAULT_PRIO`
- `int starpu_sched_ctx_get_min_priority` (unsigned sched\_ctx\_id)
- `int starpu_sched_ctx_get_max_priority` (unsigned sched\_ctx\_id)
- `int starpu_sched_ctx_set_min_priority` (unsigned sched\_ctx\_id, int min\_prio)
- `int starpu_sched_ctx_set_max_priority` (unsigned sched\_ctx\_id, int max\_prio)
- `int starpu_sched_ctx_min_priority_is_set` (unsigned sched\_ctx\_id)
- `int starpu_sched_ctx_max_priority_is_set` (unsigned sched\_ctx\_id)

## 29.24 starpu\_sched\_ctx\_hypervisor.h File Reference

### Data Structures

- `struct starpu_sched_ctx_performance_counters`

### Functions

#### Scheduling Context Link with Hypervisor

- `void starpu_sched_ctx_set_perf_counters` (unsigned sched\_ctx\_id, void \*perf\_counters)
- `void starpu_sched_ctx_notify_hypervisor_exists` (void)
- `unsigned starpu_sched_ctx_check_if_hypervisor_exists` (void)

## 29.25 starpu\_scheduler.h File Reference

```
#include <starpu.h>
```

### Data Structures

- struct [starpu\\_sched\\_policy](#)

### Functions

- struct [starpu\\_sched\\_policy](#) \*\* [starpu\\_sched\\_get\\_predefined\\_policies](#) ()
- void [starpu\\_worker\\_get\\_sched\\_condition](#) (int workerid, starpu\_pthread\_mutex\_t \*\*sched\_mutex, starpu\_pthread\_cond\_t \*\*sched\_cond)
- int [starpu\\_wake\\_worker](#) (int workerid)
- int [starpu\\_wakeup\\_worker](#) (int workerid, starpu\_pthread\_cond\_t \*cond, starpu\_pthread\_mutex\_t \*mutex)
- int [starpu\\_worker\\_can\\_execute\\_task](#) (unsigned workerid, struct [starpu\\_task](#) \*task, unsigned nimpl)
- int [starpu\\_worker\\_can\\_execute\\_task\\_impl](#) (unsigned workerid, struct [starpu\\_task](#) \*task, unsigned \*impl\_mask)
- int [starpu\\_worker\\_can\\_execute\\_task\\_first\\_impl](#) (unsigned workerid, struct [starpu\\_task](#) \*task, unsigned \*nimpl)
- int [starpu\\_push\\_local\\_task](#) (int workerid, struct [starpu\\_task](#) \*task, int back)
- int [starpu\\_push\\_task\\_end](#) (struct [starpu\\_task](#) \*task)
- int [starpu\\_combined\\_worker\\_assign\\_workerid](#) (int nworkers, int workerid\_array[])
- int [starpu\\_combined\\_worker\\_get\\_description](#) (int workerid, int \*worker\_size, int \*\*combined\_workerid)
- int [starpu\\_combined\\_worker\\_can\\_execute\\_task](#) (unsigned workerid, struct [starpu\\_task](#) \*task, unsigned nimpl)
- int [starpu\\_get\\_prefetch\\_flag](#) (void)
- int [starpu\\_prefetch\\_task\\_input\\_on\\_node](#) (struct [starpu\\_task](#) \*task, unsigned node)
- uint32\_t [starpu\\_task\\_footprint](#) (struct [starpu\\_perfmodel](#) \*model, struct [starpu\\_task](#) \*task, struct [starpu\\_perfmodel\\_arch](#) \*arch, unsigned nimpl)
- uint32\_t [starpu\\_task\\_data\\_footprint](#) (struct [starpu\\_task](#) \*task)
- double [starpu\\_task\\_expected\\_length](#) (struct [starpu\\_task](#) \*task, struct [starpu\\_perfmodel\\_arch](#) \*arch, unsigned nimpl)
- double [starpu\\_worker\\_get\\_relative\\_speedup](#) (struct [starpu\\_perfmodel\\_arch](#) \*perf\_arch)
- double [starpu\\_task\\_expected\\_data\\_transfer\\_time](#) (unsigned memory\_node, struct [starpu\\_task](#) \*task)
- double [starpu\\_data\\_expected\\_transfer\\_time](#) (starpu\_data\_handle\_t handle, unsigned memory\_node, enum [starpu\\_data\\_access\\_mode](#) mode)
- double [starpu\\_task\\_expected\\_power](#) (struct [starpu\\_task](#) \*task, struct [starpu\\_perfmodel\\_arch](#) \*arch, unsigned nimpl)
- double [starpu\\_task\\_expected\\_conversion\\_time](#) (struct [starpu\\_task](#) \*task, struct [starpu\\_perfmodel\\_arch](#) \*arch, unsigned nimpl)
- double [starpu\\_task\\_bundle\\_expected\\_length](#) (starpu\_task\_bundle\_t bundle, struct [starpu\\_perfmodel\\_arch](#) \*arch, unsigned nimpl)
- double [starpu\\_task\\_bundle\\_expected\\_data\\_transfer\\_time](#) (starpu\_task\_bundle\_t bundle, unsigned memory\_node)
- double [starpu\\_task\\_bundle\\_expected\\_power](#) (starpu\_task\_bundle\_t bundle, struct [starpu\\_perfmodel\\_arch](#) \*arch, unsigned nimpl)
- void [starpu\\_sched\\_ctx\\_worker\\_shares\\_tasks\\_lists](#) (int workerid, int sched\_ctx\_id)

## 29.26 starpu\_sink.h File Reference

### Functions

- void [starpu\\_sink\\_common\\_worker](#) (int argc, char \*\*argv)

## 29.27 starpu\_stdlib.h File Reference

```
#include <starpu.h>
```

### Macros

- `#define STARPU_MALLOC_PINNED`
- `#define STARPU_MALLOC_COUNT`
- `#define STARPU_MALLOC_NORECLAIM`
- `#define STARPU_MEMORY_WAIT`
- `#define STARPU_MEMORY_OVERFLOW`

### Functions

- void `starpu_malloc_set_align` (size\_t align)
- int `starpu_malloc` (void \*\*A, size\_t dim)
- int `starpu_free` (void \*A)
- int `starpu_malloc_flags` (void \*\*A, size\_t dim, int flags)
- int `starpu_free_flags` (void \*A, size\_t dim, int flags)
- starpu\_ssize\_t `starpu_memory_get_total` (unsigned node)
- starpu\_ssize\_t `starpu_memory_get_available` (unsigned node)
- int `starpu_memory_allocate` (unsigned node, size\_t size, int flags)
- void `starpu_memory_deallocate` (unsigned node, size\_t size)

## 29.28 starpu\_task.h File Reference

```
#include <starpu.h>
#include <starpu_data.h>
#include <starpu_util.h>
#include <starpu_task_bundle.h>
#include <errno.h>
#include <cuda.h>
```

### Data Structures

- struct `starpu_codelet`
- struct `starpu_task`

### Macros

- `#define STARPU_CPU`
- `#define STARPU_CUDA`
- `#define STARPU_OPENCL`
- `#define STARPU_MIC`
- `#define STARPU_SCC`
- `#define STARPU_CUDA_ASYNC`
- `#define STARPU_OPENCL_ASYNC`
- `#define STARPU_TASK_INVALID`
- `#define STARPU_MULTIPLE_CPU_IMPLEMENTATIONS`

- `#define STARPU_MULTIPLE_CUDA_IMPLEMENTATIONS`
- `#define STARPU_MULTIPLE_OPENCL_IMPLEMENTATIONS`
- `#define STARPU_VARIABLE_NBUFFERS`
- `#define STARPU_TASK_INITIALIZER`
- `#define STARPU_TASK_GET_NBUFFERS(task)`
- `#define STARPU_TASK_GET_HANDLE(task, i)`
- `#define STARPU_TASK_SET_HANDLE(task, handle, i)`
- `#define STARPU_CODELET_GET_MODE(codelet, i)`
- `#define STARPU_CODELET_SET_MODE(codelet, mode, i)`
- `#define STARPU_TASK_GET_MODE(task, i)`
- `#define STARPU_TASK_SET_MODE(task, mode, i)`
- `#define STARPU_CODELET_GET_NODE(codelet, i)`
- `#define STARPU_CODELET_SET_NODE(codelet, __node, i)`

## Typedefs

- `typedef uint64_t starpu_tag_t`
- `typedef void(* starpu_cpu_func_t)(void **, void *)`
- `typedef void(* starpu_cuda_func_t)(void **, void *)`
- `typedef void(* starpu_opencl_func_t)(void **, void *)`
- `typedef void(* starpu_mic_kernel_t)(void **, void *)`
- `typedef void(* starpu_scc_kernel_t)(void **, void *)`
- `typedef starpu_mic_kernel_t(* starpu_mic_func_t)(void)`
- `typedef starpu_scc_kernel_t(* starpu_scc_func_t)(void)`

## Enumerations

- `enum starpu_codelet_type { STARPU_SEQ, STARPU_SPMD, STARPU_FORKJOIN }`
- `enum starpu_task_status { STARPU_TASK_INVALID, STARPU_TASK_INVALID, STARPU_TASK_BLOCKED, STARPU_TASK_READY, STARPU_TASK_RUNNING, STARPU_TASK_FINISHED, STARPU_TASK_BLOCKED_ON_TAG, STARPU_TASK_BLOCKED_ON_TASK, STARPU_TASK_BLOCKED_ON_DATA, STARPU_TASK_STOPPED }`

## Functions

- `void starpu_tag_declare_deps (starpu_tag_t id, unsigned ndeps,...)`
- `void starpu_tag_declare_deps_array (starpu_tag_t id, unsigned ndeps, starpu_tag_t *array)`
- `void starpu_task_declare_deps_array (struct starpu_task *task, unsigned ndeps, struct starpu_task *task_array[])`
- `int starpu_tag_wait (starpu_tag_t id)`
- `int starpu_tag_wait_array (unsigned ntags, starpu_tag_t *id)`
- `void starpu_tag_notify_from_apps (starpu_tag_t id)`
- `void starpu_tag_restart (starpu_tag_t id)`
- `void starpu_tag_remove (starpu_tag_t id)`
- `void starpu_task_init (struct starpu_task *task)`
- `void starpu_task_clean (struct starpu_task *task)`
- `struct starpu_task * starpu_task_create (void)`
- `void starpu_task_destroy (struct starpu_task *task)`
- `int starpu_task_submit (struct starpu_task *task) STARPU_WARN_UNUSED_RESULT`
- `int starpu_task_submit_to_ctx (struct starpu_task *task, unsigned sched_ctx_id)`
- `int starpu_task_finished (struct starpu_task *task) STARPU_WARN_UNUSED_RESULT`

- int [starpu\\_task\\_wait](#) (struct [starpu\\_task](#) \*task) [STARPU\\_WARN\\_UNUSED\\_RESULT](#)
- int [starpu\\_task\\_wait\\_for\\_all](#) (void)
- int [starpu\\_task\\_wait\\_for\\_all\\_in\\_ctx](#) (unsigned sched\_ctx\_id)
- int [starpu\\_task\\_wait\\_for\\_no\\_ready](#) (void)
- int [starpu\\_task\\_nready](#) (void)
- int [starpu\\_task\\_nsubmitted](#) (void)
- void [starpu\\_codelet\\_init](#) (struct [starpu\\_codelet](#) \*cl)
- void [starpu\\_codelet\\_display\\_stats](#) (struct [starpu\\_codelet](#) \*cl)
- struct [starpu\\_task](#) \* [starpu\\_task\\_get\\_current](#) (void)
- void [starpu\\_parallel\\_task\\_barrier\\_init](#) (struct [starpu\\_task](#) \*task, int workerid)
- void [starpu\\_parallel\\_task\\_barrier\\_init\\_n](#) (struct [starpu\\_task](#) \*task, int worker\_size)
- struct [starpu\\_task](#) \* [starpu\\_task\\_dup](#) (struct [starpu\\_task](#) \*task)
- void [starpu\\_task\\_set\\_implementation](#) (struct [starpu\\_task](#) \*task, unsigned impl)
- unsigned [starpu\\_task\\_get\\_implementation](#) (struct [starpu\\_task](#) \*task)

## 29.28.1 Macro Definition Documentation

### 29.28.1.1 `#define STARPU_CUDA_ASYNC`

Value to be set in [starpu\\_codelet::cuda\\_flags](#) to allow asynchronous CUDA kernel execution.

### 29.28.1.2 `#define STARPU_OPENCL_ASYNC`

Value to be set in [starpu\\_codelet::opencl\\_flags](#) to allow asynchronous OpenCL kernel execution.

## 29.29 starpu\_task\_bundle.h File Reference

### Typedefs

- typedef struct  
    [\\_starpu\\_task\\_bundle](#) \* [starpu\\_task\\_bundle\\_t](#)

### Functions

- void [starpu\\_task\\_bundle\\_create](#) ([starpu\\_task\\_bundle\\_t](#) \*bundle)
- int [starpu\\_task\\_bundle\\_insert](#) ([starpu\\_task\\_bundle\\_t](#) bundle, struct [starpu\\_task](#) \*task)
- int [starpu\\_task\\_bundle\\_remove](#) ([starpu\\_task\\_bundle\\_t](#) bundle, struct [starpu\\_task](#) \*task)
- void [starpu\\_task\\_bundle\\_close](#) ([starpu\\_task\\_bundle\\_t](#) bundle)

## 29.30 starpu\_task\_list.h File Reference

```
#include <starpu_task.h>
#include <starpu_util.h>
```

### Data Structures

- struct [starpu\\_task\\_list](#)

## Functions

- static STARPU\_INLINE void [starpu\\_task\\_list\\_init](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE void [starpu\\_task\\_list\\_push\\_front](#) (struct [starpu\\_task\\_list](#) \*list, struct [starpu\\_task](#) \*task)
- static STARPU\_INLINE void [starpu\\_task\\_list\\_push\\_back](#) (struct [starpu\\_task\\_list](#) \*list, struct [starpu\\_task](#) \*task)
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_front](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_back](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE int [starpu\\_task\\_list\\_empty](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE void [starpu\\_task\\_list\\_erase](#) (struct [starpu\\_task\\_list](#) \*list, struct [starpu\\_task](#) \*task)
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_pop\\_front](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_pop\\_back](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_begin](#) (struct [starpu\\_task\\_list](#) \*list)
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_end](#) (struct [starpu\\_task\\_list](#) \*list [STARPU\\_ATTRIBUTE\\_UNUSED](#))
- static STARPU\_INLINE struct [starpu\\_task](#) \* [starpu\\_task\\_list\\_next](#) (struct [starpu\\_task](#) \*task)

## 29.31 starpu\_task\_util.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <starpu.h>
```

## Macros

- [#define STARPU\\_VALUE](#)
- [#define STARPU\\_CALLBACK](#)
- [#define STARPU\\_CALLBACK\\_WITH\\_ARG](#)
- [#define STARPU\\_CALLBACK\\_ARG](#)
- [#define STARPU\\_PRIORITY](#)
- [#define STARPU\\_DATA\\_ARRAY](#)
- [#define STARPU\\_DATA\\_MODE\\_ARRAY](#)
- [#define STARPU\\_TAG](#)
- [#define STARPU\\_HYPERVISOR\\_TAG](#)
- [#define STARPU\\_FLOPS](#)
- [#define STARPU\\_SCHED\\_CTX](#)
- [#define STARPU\\_PROLOGUE\\_CALLBACK](#)
- [#define STARPU\\_PROLOGUE\\_CALLBACK\\_ARG](#)
- [#define STARPU\\_PROLOGUE\\_CALLBACK\\_POP](#)
- [#define STARPU\\_PROLOGUE\\_CALLBACK\\_POP\\_ARG](#)
- [#define STARPU\\_EXECUTE\\_ON\\_WORKER](#)
- [#define STARPU\\_TAG\\_ONLY](#)
- [#define STARPU\\_POSSIBLY\\_PARALLEL](#)
- [#define STARPU\\_WORKER\\_ORDER](#)



- `#define STARPU_NODE_SELECTION_POLICY`

### MPI Insert Task

- `#define STARPU_EXECUTE_ON_NODE`
- `#define STARPU_EXECUTE_ON_DATA`

### Functions

- void `starpu_create_sync_task` (`starpu_tag_t` sync\_tag, unsigned ndeps, `starpu_tag_t` \*deps, void(\*callback)(void \*), void \*callback\_arg)
- struct `starpu_task` \* `starpu_task_build` (struct `starpu_codelet` \*cl,...)
- int `starpu_task_insert` (struct `starpu_codelet` \*cl,...)
- int `starpu_insert_task` (struct `starpu_codelet` \*cl,...)
- void `starpu_codelet_unpack_args` (void \*cl\_arg,...)
- void `starpu_codelet_pack_args` (void \*\*arg\_buffer, size\_t \*arg\_buffer\_size,...)

## 29.32 starpu\_thread.h File Reference

```
#include <starpu_config.h>
#include <starpu_util.h>
#include <xbt/synchro_core.h>
#include <msg/msg.h>
#include <stdint.h>
```

### Data Structures

- struct `starpu_pthread_barrier_t`
- struct `starpu_pthread_spinlock_t`

### Macros

- `#define STARPU_PTHREAD_MUTEX_INITIALIZER`
- `#define STARPU_PTHREAD_COND_INITIALIZER`
- `#define STARPU_PTHREAD_BARRIER_SERIAL_THREAD`

### Typedefs

- typedef msg\_process\_t `starpu_pthread_t`
- typedef int `starpu_pthread_attr_t`
- typedef xbt\_mutex\_t `starpu_pthread_mutex_t`
- typedef int `starpu_pthread_mutexattr_t`
- typedef int `starpu_pthread_key_t`
- typedef xbt\_cond\_t `starpu_pthread_cond_t`
- typedef int `starpu_pthread_condattr_t`
- typedef xbt\_mutex\_t `starpu_pthread_rwlock_t`
- typedef int `starpu_pthread_rwlockattr_t`
- typedef int `starpu_pthread_barrierattr_t`

## Functions

- int **starpu\_thread\_create\_on** (char \*name, starpu\_thread\_t \*thread, const starpu\_thread\_attr\_t \*attr, void \*(\*start\_routine)(void \*), void \*arg, int where)
- int **starpu\_thread\_create** (starpu\_thread\_t \*thread, const starpu\_thread\_attr\_t \*attr, void \*(\*start\_routine)(void \*), void \*arg)
- int **starpu\_thread\_join** (starpu\_thread\_t thread, void \*\*retval)
- int **starpu\_thread\_exit** (void \*retval)
- int **starpu\_thread\_attr\_init** (starpu\_thread\_attr\_t \*attr)
- int **starpu\_thread\_attr\_destroy** (starpu\_thread\_attr\_t \*attr)
- int **starpu\_thread\_attr\_setdetachstate** (starpu\_thread\_attr\_t \*attr, int detachstate)
- int **starpu\_thread\_mutex\_init** (starpu\_thread\_mutex\_t \*mutex, const starpu\_thread\_mutexattr\_t \*mutexattr)
- int **starpu\_thread\_mutex\_destroy** (starpu\_thread\_mutex\_t \*mutex)
- int **starpu\_thread\_mutex\_lock** (starpu\_thread\_mutex\_t \*mutex)
- int **starpu\_thread\_mutex\_unlock** (starpu\_thread\_mutex\_t \*mutex)
- int **starpu\_thread\_mutex\_trylock** (starpu\_thread\_mutex\_t \*mutex)
- int **starpu\_thread\_mutexattr\_gettype** (const starpu\_thread\_mutexattr\_t \*attr, int \*type)
- int **starpu\_thread\_mutexattr\_settype** (starpu\_thread\_mutexattr\_t \*attr, int type)
- int **starpu\_thread\_mutexattr\_destroy** (starpu\_thread\_mutexattr\_t \*attr)
- int **starpu\_thread\_mutexattr\_init** (starpu\_thread\_mutexattr\_t \*attr)
- int **starpu\_thread\_key\_create** (starpu\_thread\_key\_t \*key, void(\*destr\_function)(void \*))
- int **starpu\_thread\_key\_delete** (starpu\_thread\_key\_t key)
- int **starpu\_thread\_setspecific** (starpu\_thread\_key\_t key, const void \*pointer)
- void \* **starpu\_thread\_getspecific** (starpu\_thread\_key\_t key)
- int **starpu\_thread\_cond\_init** (starpu\_thread\_cond\_t \*cond, starpu\_thread\_condattr\_t \*cond\_attr)
- int **starpu\_thread\_cond\_signal** (starpu\_thread\_cond\_t \*cond)
- int **starpu\_thread\_cond\_broadcast** (starpu\_thread\_cond\_t \*cond)
- int **starpu\_thread\_cond\_wait** (starpu\_thread\_cond\_t \*cond, starpu\_thread\_mutex\_t \*mutex)
- int **starpu\_thread\_cond\_timedwait** (starpu\_thread\_cond\_t \*cond, starpu\_thread\_mutex\_t \*mutex, const struct timespec \*abstime)
- int **starpu\_thread\_cond\_destroy** (starpu\_thread\_cond\_t \*cond)
- int **starpu\_thread\_rwlock\_init** (starpu\_thread\_rwlock\_t \*rwlock, const starpu\_thread\_rwlockattr\_t \*attr)
- int **starpu\_thread\_rwlock\_destroy** (starpu\_thread\_rwlock\_t \*rwlock)
- int **starpu\_thread\_rwlock\_rdlock** (starpu\_thread\_rwlock\_t \*rwlock)
- int **starpu\_thread\_rwlock\_tryrdlock** (starpu\_thread\_rwlock\_t \*rwlock)
- int **starpu\_thread\_rwlock\_wrlock** (starpu\_thread\_rwlock\_t \*rwlock)
- int **starpu\_thread\_rwlock\_trywrlock** (starpu\_thread\_rwlock\_t \*rwlock)
- int **starpu\_thread\_rwlock\_unlock** (starpu\_thread\_rwlock\_t \*rwlock)
- int **starpu\_thread\_barrier\_init** (starpu\_thread\_barrier\_t \*barrier, const starpu\_thread\_barrierattr\_t \*attr, unsigned count)
- int **starpu\_thread\_barrier\_destroy** (starpu\_thread\_barrier\_t \*barrier)
- int **starpu\_thread\_barrier\_wait** (starpu\_thread\_barrier\_t \*barrier)
- int **starpu\_thread\_spin\_init** (starpu\_thread\_spinlock\_t \*lock, int pshared)
- int **starpu\_thread\_spin\_destroy** (starpu\_thread\_spinlock\_t \*lock)
- int **starpu\_thread\_spin\_lock** (starpu\_thread\_spinlock\_t \*lock)
- int **starpu\_thread\_spin\_trylock** (starpu\_thread\_spinlock\_t \*lock)
- int **starpu\_thread\_spin\_unlock** (starpu\_thread\_spinlock\_t \*lock)

## 29.32.1 Data Structure Documentation

### 29.32.1.1 struct starpu\_thread\_barrier\_t

## Data Fields

|                                  |       |  |
|----------------------------------|-------|--|
| starpu_↔<br>pthread_mutex↔<br>_t | mutex |  |
| starpu_↔<br>pthread_cond_t       | cond  |  |
| unsigned                         | count |  |
| unsigned                         | done  |  |

## 29.32.1.2 struct starpu\_thread\_spinlock\_t

## Data Fields

|     |       |  |
|-----|-------|--|
| int | taken |  |
|-----|-------|--|

## 29.33 starpu\_thread\_util.h File Reference

```
#include <starpu_util.h>
#include <errno.h>
```

## Macros

- #define [STARPU\\_PTHREAD\\_CREATE\\_ON](#)(name, thread, attr, routine, arg, where)
- #define [STARPU\\_PTHREAD\\_CREATE](#)(thread, attr, routine, arg)
- #define [STARPU\\_PTHREAD\\_MUTEX\\_INIT](#)(mutex, attr)
- #define [STARPU\\_PTHREAD\\_MUTEX\\_DESTROY](#)(mutex)
- #define [STARPU\\_PTHREAD\\_MUTEX\\_LOCK](#)(mutex)
- #define [STARPU\\_PTHREAD\\_MUTEX\\_TRYLOCK](#)(mutex)
- #define [STARPU\\_PTHREAD\\_MUTEX\\_UNLOCK](#)(mutex)
- #define [STARPU\\_PTHREAD\\_KEY\\_CREATE](#)(key, destr)
- #define [STARPU\\_PTHREAD\\_KEY\\_DELETE](#)(key)
- #define [STARPU\\_PTHREAD\\_SETSPECIFIC](#)(key, ptr)
- #define [STARPU\\_PTHREAD\\_GETSPECIFIC](#)(key)
- #define [STARPU\\_PTHREAD\\_RWLOCK\\_INIT](#)(rwlock, attr)
- #define [STARPU\\_PTHREAD\\_RWLOCK\\_RDLOCK](#)(rwlock)
- #define [STARPU\\_PTHREAD\\_RWLOCK\\_TRYRDLOCK](#)(rwlock)
- #define [STARPU\\_PTHREAD\\_RWLOCK\\_WRLOCK](#)(rwlock)
- #define [STARPU\\_PTHREAD\\_RWLOCK\\_TRYWRLOCK](#)(rwlock)
- #define [STARPU\\_PTHREAD\\_RWLOCK\\_UNLOCK](#)(rwlock)
- #define [STARPU\\_PTHREAD\\_RWLOCK\\_DESTROY](#)(rwlock)
- #define [STARPU\\_PTHREAD\\_COND\\_INIT](#)(cond, attr)
- #define [STARPU\\_PTHREAD\\_COND\\_DESTROY](#)(cond)
- #define [STARPU\\_PTHREAD\\_COND\\_SIGNAL](#)(cond)
- #define [STARPU\\_PTHREAD\\_COND\\_BROADCAST](#)(cond)
- #define [STARPU\\_PTHREAD\\_COND\\_WAIT](#)(cond, mutex)
- #define [STARPU\\_PTHREAD\\_BARRIER\\_INIT](#)(barrier, attr, count)
- #define [STARPU\\_PTHREAD\\_BARRIER\\_DESTROY](#)(barrier)
- #define [STARPU\\_PTHREAD\\_BARRIER\\_WAIT](#)(barrier)

## Functions

- static STARPU\_INLINE int **\_starpu\_thread\_mutex\_trylock** (starpu\_thread\_mutex\_t \*mutex, char \*file, int line)
- static STARPU\_INLINE int **\_starpu\_thread\_rwlock\_tryrdlock** (starpu\_thread\_rwlock\_t \*rwlock, char \*file, int line)
- static STARPU\_INLINE int **\_starpu\_thread\_rwlock\_trywrlock** (starpu\_thread\_rwlock\_t \*rwlock, char \*file, int line)

## 29.34 starpu\_top.h File Reference

```
#include <starpu.h>
#include <stdlib.h>
#include <time.h>
```

## Data Structures

- struct [starpu\\_top\\_data](#)
- struct [starpu\\_top\\_param](#)

## Enumerations

- enum [starpu\\_top\\_data\\_type](#) { STARPU\_TOP\_DATA\_BOOLEAN, STARPU\_TOP\_DATA\_INTEGER, STARPU\_TOP\_DATA\_FLOAT }
- enum [starpu\\_top\\_param\\_type](#) { STARPU\_TOP\_PARAM\_BOOLEAN, STARPU\_TOP\_PARAM\_INTEGER, STARPU\_TOP\_PARAM\_FLOAT, STARPU\_TOP\_PARAM\_ENUM }
- enum [starpu\\_top\\_message\\_type](#) { TOP\_TYPE\_GO, TOP\_TYPE\_SET, TOP\_TYPE\_CONTINUE, TOP\_TYPE\_ENABLE, TOP\_TYPE\_DISABLE, TOP\_TYPE\_DEBUG, TOP\_TYPE\_UNKNOW }

## Functions

### Functions to call before the initialisation

- struct [starpu\\_top\\_data](#) \* [starpu\\_top\\_add\\_data\\_boolean](#) (const char \*data\_name, int active)
- struct [starpu\\_top\\_data](#) \* [starpu\\_top\\_add\\_data\\_integer](#) (const char \*data\_name, int minimum\_value, int maximum\_value, int active)
- struct [starpu\\_top\\_data](#) \* [starpu\\_top\\_add\\_data\\_float](#) (const char \*data\_name, double minimum\_value, double maximum\_value, int active)
- struct [starpu\\_top\\_param](#) \* [starpu\\_top\\_register\\_parameter\\_boolean](#) (const char \*param\_name, int \*parameter\_field, void(\*callback)(struct [starpu\\_top\\_param](#) \*))
- struct [starpu\\_top\\_param](#) \* [starpu\\_top\\_register\\_parameter\\_integer](#) (const char \*param\_name, int \*parameter\_field, int minimum\_value, int maximum\_value, void(\*callback)(struct [starpu\\_top\\_param](#) \*))
- struct [starpu\\_top\\_param](#) \* [starpu\\_top\\_register\\_parameter\\_float](#) (const char \*param\_name, double \*parameter\_field, double minimum\_value, double maximum\_value, void(\*callback)(struct [starpu\\_top\\_param](#) \*))
- struct [starpu\\_top\\_param](#) \* [starpu\\_top\\_register\\_parameter\\_enum](#) (const char \*param\_name, int \*parameter\_field, char \*\*values, int nb\_values, void(\*callback)(struct [starpu\\_top\\_param](#) \*))

### Initialisation

- void [starpu\\_top\\_init\\_and\\_wait](#) (const char \*server\_name)

### To call after initialisation

- void [starpu\\_top\\_update\\_parameter](#) (const struct [starpu\\_top\\_param](#) \*param)
- void [starpu\\_top\\_update\\_data\\_boolean](#) (const struct [starpu\\_top\\_data](#) \*data, int value)
- void [starpu\\_top\\_update\\_data\\_integer](#) (const struct [starpu\\_top\\_data](#) \*data, int value)
- void [starpu\\_top\\_update\\_data\\_float](#) (const struct [starpu\\_top\\_data](#) \*data, double value)
- void [starpu\\_top\\_task\\_prevision](#) (struct [starpu\\_task](#) \*task, int devid, unsigned long long start, unsigned long long end)
- void [starpu\\_top\\_debug\\_log](#) (const char \*message)
- void [starpu\\_top\\_debug\\_lock](#) (const char \*message)

## 29.35 starpu\_tree.h File Reference

### Data Structures

- struct [starpu\\_tree](#)

### Functions

- void [starpu\\_tree\\_reset\\_visited](#) (struct [starpu\\_tree](#) \*tree, int \*visited)
- void [starpu\\_tree\\_insert](#) (struct [starpu\\_tree](#) \*tree, int id, int level, int is\_pu, int arity, struct [starpu\\_tree](#) \*father)
- struct [starpu\\_tree](#) \* [starpu\\_tree\\_get](#) (struct [starpu\\_tree](#) \*tree, int id)
- struct [starpu\\_tree](#) \* [starpu\\_tree\\_get\\_neighbour](#) (struct [starpu\\_tree](#) \*tree, struct [starpu\\_tree](#) \*node, int \*visited, int \*present)
- void [starpu\\_tree\\_free](#) (struct [starpu\\_tree](#) \*tree)

## 29.36 starpu\_util.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <starpu_config.h>
#include <starpu_task.h>
#include <sys/time.h>
```

### Macros

- #define [STARPU\\_GNUC\\_PREREQ](#)(maj, min)
- #define [STARPU\\_UNLIKELY](#)(expr)
- #define [STARPU\\_LIKELY](#)(expr)
- #define [STARPU\\_ATTRIBUTE\\_UNUSED](#)
- #define [STARPU\\_ATTRIBUTE\\_INTERNAL](#)
- #define [STARPU\\_ATTRIBUTE\\_MALLOC](#)
- #define [STARPU\\_ATTRIBUTE\\_WARN\\_UNUSED\\_RESULT](#)
- #define [STARPU\\_ATTRIBUTE\\_PURE](#)
- #define [STARPU\\_ATTRIBUTE\\_ALIGNED](#)(size)
- #define [STARPU\\_INLINE](#)
- #define [endif](#)
- #define [STARPU\\_WARN\\_UNUSED\\_RESULT](#)
- #define [STARPU\\_POISON\\_PTR](#)
- #define [STARPU\\_MIN](#)(a, b)
- #define [STARPU\\_MAX](#)(a, b)

- `#define STARPU_ASSERT(x)`
- `#define STARPU_ASSERT_MSG(x, msg,...)`
- `#define STARPU_ABORT()`
- `#define STARPU_ABORT_MSG(msg,...)`
- `#define STARPU_CHECK_RETURN_VALUE(err, message,...)`
- `#define STARPU_CHECK_RETURN_VALUE_IS(err, value, message,...)`
- `#define STARPU_ATOMIC_SOMETHING(name, expr)`
- `#define STARPU_RMB()`
- `#define STARPU_WMB()`

## Functions

- static `__starpu_inline` int `starpu_get_env_number` (const char \*str)
- static `__starpu_inline` int `starpu_get_env_number_default` (const char \*str, int defval)
- static `__starpu_inline` float `starpu_get_env_float_default` (const char \*str, float defval)
- void `starpu_execute_on_each_worker` (void(\*func)(void \*), void \*arg, uint32\_t where)
- void `starpu_execute_on_each_worker_ex` (void(\*func)(void \*), void \*arg, uint32\_t where, const char \*name)
- void `starpu_execute_on_specific_workers` (void(\*func)(void \*), void \*arg, unsigned num\_workers, unsigned \*workers, const char \*name)
- int `starpu_data_cpy` (`starpu_data_handle_t` dst\_handle, `starpu_data_handle_t` src\_handle, int asynchronous, void(\*callback\_func)(void \*), void \*callback\_arg)
- double `starpu_timing_now` (void)

## 29.37 starpu\_worker.h File Reference

```
#include <stdlib.h>
#include <starpu_config.h>
#include <starpu_thread.h>
```

## Data Structures

- struct `starpu_sched_ctx_iterator`
- struct `starpu_worker_collection`

## Enumerations

- enum `starpu_worker_archtype` {  
`STARPU_CPU_WORKER`, `STARPU_CUDA_WORKER`, `STARPU_OPENCL_WORKER`, `STARPU_MIC_↵  
WORKER`,  
`STARPU_SCC_WORKER`, `STARPU_ANY_WORKER` }
- enum `starpu_worker_collection_type` { `STARPU_WORKER_TREE`, `STARPU_WORKER_LIST` }

## Functions

- unsigned `starpu_worker_get_count` (void)
- unsigned `starpu_combined_worker_get_count` (void)
- unsigned `starpu_worker_is_combined_worker` (int id)
- unsigned `starpu_cpu_worker_get_count` (void)
- unsigned `starpu_cuda_worker_get_count` (void)
- unsigned `starpu_opencl_worker_get_count` (void)
- unsigned `starpu_mic_worker_get_count` (void)

- unsigned [starpu\\_scc\\_worker\\_get\\_count](#) (void)
- unsigned [starpu\\_mic\\_device\\_get\\_count](#) (void)
- int [starpu\\_worker\\_get\\_id](#) (void)
- int [starpu\\_worker\\_get\\_bindid](#) (int workerid)
- int [starpu\\_combined\\_worker\\_get\\_id](#) (void)
- int [starpu\\_combined\\_worker\\_get\\_size](#) (void)
- int [starpu\\_combined\\_worker\\_get\\_rank](#) (void)
- enum [starpu\\_worker\\_archtype](#) [starpu\\_worker\\_get\\_type](#) (int id)
- int [starpu\\_worker\\_get\\_count\\_by\\_type](#) (enum [starpu\\_worker\\_archtype](#) type)
- int [starpu\\_worker\\_get\\_ids\\_by\\_type](#) (enum [starpu\\_worker\\_archtype](#) type, int \*workerids, int maxsize)
- int [starpu\\_worker\\_get\\_by\\_type](#) (enum [starpu\\_worker\\_archtype](#) type, int num)
- int [starpu\\_worker\\_get\\_by\\_devid](#) (enum [starpu\\_worker\\_archtype](#) type, int devid)
- void [starpu\\_worker\\_get\\_name](#) (int id, char \*dst, size\_t maxlen)
- int [starpu\\_worker\\_get\\_devid](#) (int id)
- int [starpu\\_worker\\_get\\_mp\\_nodeid](#) (int id)
- struct [starpu\\_tree](#) \* [starpu\\_workers\\_get\\_tree](#) (void)
- unsigned [starpu\\_worker\\_get\\_sched\\_ctx\\_list](#) (int worker, unsigned \*\*sched\_ctx)
- unsigned [starpu\\_worker\\_is\\_slave](#) (int workerid)
- char \* [starpu\\_worker\\_get\\_type\\_as\\_string](#) (enum [starpu\\_worker\\_archtype](#) type)

## Variables

- struct [starpu\\_worker\\_collection](#) [worker\\_list](#)
- struct [starpu\\_worker\\_collection](#) [worker\\_tree](#)

## 29.38 starpu\_mpi.h File Reference

```
#include <starpu.h>
#include <mpi.h>
```

## Typedefs

- typedef void \* [starpu\\_mpi\\_req](#)

## Functions

- int [starpu\\_mpi\\_issend\\_detached](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int dest, int mpi\_tag, MPI\_Comm comm, void(\*callback)(void \*), void \*arg)
- int [starpu\\_mpi\\_world\\_rank](#) (void)

## Communication

- int [starpu\\_mpi\\_isend](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, [starpu\\_mpi\\_req](#) \*req, int dest, int mpi\_tag, MPI\_Comm comm)
- int [starpu\\_mpi\\_irecv](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, [starpu\\_mpi\\_req](#) \*req, int source, int mpi\_tag, MPI\_Comm comm)
- int [starpu\\_mpi\\_send](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int dest, int mpi\_tag, MPI\_Comm comm)
- int [starpu\\_mpi\\_recv](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int source, int mpi\_tag, MPI\_Comm comm, MPI\_Status \*status)
- int [starpu\\_mpi\\_isend\\_detached](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int dest, int mpi\_tag, MPI\_Comm comm, void(\*callback)(void \*), void \*arg)

- int [starpu\\_mpi\\_irecv\\_detached](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int source, int mpi\_tag, MPI\_Comm comm, void(\*callback)(void \*), void \*arg)
- int [starpu\\_mpi\\_issend](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, [starpu\\_mpi\\_req](#) \*req, int dest, int mpi\_tag, MPI\_Comm comm)
- int [starpu\\_mpi\\_wait](#) ([starpu\\_mpi\\_req](#) \*req, MPI\_Status \*status)
- int [starpu\\_mpi\\_test](#) ([starpu\\_mpi\\_req](#) \*req, int \*flag, MPI\_Status \*status)
- int [starpu\\_mpi\\_barrier](#) (MPI\_Comm comm)
- int [starpu\\_mpi\\_irecv\\_detached\\_sequential\\_consistency](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int source, int mpi\_tag, MPI\_Comm comm, void(\*callback)(void \*), void \*arg, int sequential\_consistency)
- int [starpu\\_mpi\\_isend\\_detached\\_unlock\\_tag](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int dest, int mpi\_tag, MPI\_Comm comm, [starpu\\_tag\\_t](#) tag)
- int [starpu\\_mpi\\_irecv\\_detached\\_unlock\\_tag](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int source, int mpi\_tag, MPI\_Comm comm, [starpu\\_tag\\_t](#) tag)
- int [starpu\\_mpi\\_isend\\_array\\_detached\\_unlock\\_tag](#) (unsigned array\_size, [starpu\\_data\\_handle\\_t](#) \*data\_handle, int \*dest, int \*mpi\_tag, MPI\_Comm \*comm, [starpu\\_tag\\_t](#) tag)
- int [starpu\\_mpi\\_irecv\\_array\\_detached\\_unlock\\_tag](#) (unsigned array\_size, [starpu\\_data\\_handle\\_t](#) \*data\_handle, int \*source, int \*mpi\_tag, MPI\_Comm \*comm, [starpu\\_tag\\_t](#) tag)
- int [starpu\\_mpi\\_get\\_communication\\_tag](#) (void)
- void [starpu\\_mpi\\_set\\_communication\\_tag](#) (int tag)

### Initialisation

- int [starpu\\_mpi\\_init](#) (int \*argc, char \*\*\*argv, int initialize\_mpi)
- int [starpu\\_mpi\\_initialize](#) (void)
- int [starpu\\_mpi\\_initialize\\_extended](#) (int \*rank, int \*world\_size)
- int [starpu\\_mpi\\_shutdown](#) (void)
- void [starpu\\_mpi\\_comm\\_amounts\\_retrieve](#) (size\_t \*comm\_amounts)

### MPI Insert Task

- struct [starpu\\_task](#) \* [starpu\\_mpi\\_task\\_build](#) (MPI\_Comm comm, struct [starpu\\_codelet](#) \*codelet,...)
- int [starpu\\_mpi\\_task\\_post\\_build](#) (MPI\_Comm comm, struct [starpu\\_codelet](#) \*codelet,...)
- int [starpu\\_mpi\\_task\\_insert](#) (MPI\_Comm comm, struct [starpu\\_codelet](#) \*codelet,...)
- int [starpu\\_mpi\\_insert\\_task](#) (MPI\_Comm comm, struct [starpu\\_codelet](#) \*codelet,...)
- void [starpu\\_mpi\\_get\\_data\\_on\\_node](#) (MPI\_Comm comm, [starpu\\_data\\_handle\\_t](#) data\_handle, int node)
- void [starpu\\_mpi\\_get\\_data\\_on\\_node\\_detached](#) (MPI\_Comm comm, [starpu\\_data\\_handle\\_t](#) data\_handle, int node, void(\*callback)(void \*), void \*arg)
- void [starpu\\_mpi\\_data\\_register](#) ([starpu\\_data\\_handle\\_t](#) data\_handle, int tag, int rank)
- char \* [starpu\\_mpi\\_node\\_selection\\_get\\_default\\_policy](#) ()
- int [starpu\\_mpi\\_node\\_selection\\_set\\_default\\_policy](#) (char \*policy)

### Collective Operations

- void [starpu\\_mpi\\_redux\\_data](#) (MPI\_Comm comm, [starpu\\_data\\_handle\\_t](#) data\_handle)
- int [starpu\\_mpi\\_scatter\\_detached](#) ([starpu\\_data\\_handle\\_t](#) \*data\_handles, int count, int root, MPI\_Comm comm, void(\*scallback)(void \*), void \*sarg, void(\*rcallback)(void \*), void \*rarg)
- int [starpu\\_mpi\\_gather\\_detached](#) ([starpu\\_data\\_handle\\_t](#) \*data\_handles, int count, int root, MPI\_Comm comm, void(\*scallback)(void \*), void \*sarg, void(\*rcallback)(void \*), void \*rarg)

### Communication Cache

- void [starpu\\_mpi\\_cache\\_flush](#) (MPI\_Comm comm, [starpu\\_data\\_handle\\_t](#) data\_handle)
- void [starpu\\_mpi\\_cache\\_flush\\_all\\_data](#) (MPI\_Comm comm)

## 29.39 [sc\\_hypervisor.h](#) File Reference

```
#include <starpu.h>
#include <starpu_sched_ctx_hypervisor.h>
#include <sc_hypervisor_config.h>
#include <sc_hypervisor_monitoring.h>
#include <math.h>
```



## Data Structures

- struct [sc\\_hypervisor\\_policy](#)

## Functions

- void \* [sc\\_hypervisor\\_init](#) (struct [sc\\_hypervisor\\_policy](#) \*policy)
- void [sc\\_hypervisor\\_shutdown](#) (void)
- void [sc\\_hypervisor\\_register\\_ctx](#) (unsigned sched\_ctx, double total\_flops)
- void [sc\\_hypervisor\\_unregister\\_ctx](#) (unsigned sched\_ctx)
- void [sc\\_hypervisor\\_post\\_resize\\_request](#) (unsigned sched\_ctx, int task\_tag)
- void [sc\\_hypervisor\\_resize\\_ctxs](#) (unsigned \*sched\_ctxs, int nsched\_ctxs, int \*workers, int nworkers)
- void [sc\\_hypervisor\\_stop\\_resize](#) (unsigned sched\_ctx)
- void [sc\\_hypervisor\\_start\\_resize](#) (unsigned sched\_ctx)
- const char \* [sc\\_hypervisor\\_get\\_policy](#) ()
- void [sc\\_hypervisor\\_add\\_workers\\_to\\_sched\\_ctx](#) (int \*workers\_to\_add, unsigned nworkers\_to\_add, unsigned sched\_ctx)
- void [sc\\_hypervisor\\_remove\\_workers\\_from\\_sched\\_ctx](#) (int \*workers\_to\_remove, unsigned nworkers\_to\_remove, unsigned sched\_ctx, unsigned now)
- void [sc\\_hypervisor\\_move\\_workers](#) (unsigned sender\_sched\_ctx, unsigned receiver\_sched\_ctx, int \*workers\_to\_move, unsigned nworkers\_to\_move, unsigned now)
- void [sc\\_hypervisor\\_size\\_ctxs](#) (unsigned \*sched\_ctxs, int nsched\_ctxs, int \*workers, int nworkers)
- unsigned [sc\\_hypervisor\\_get\\_size\\_req](#) (unsigned \*\*sched\_ctxs, int \*nsched\_ctxs, int \*\*workers, int \*nworkers)
- void [sc\\_hypervisor\\_save\\_size\\_req](#) (unsigned \*sched\_ctxs, int nsched\_ctxs, int \*workers, int nworkers)
- void [sc\\_hypervisor\\_free\\_size\\_req](#) (void)
- unsigned [sc\\_hypervisor\\_can\\_resize](#) (unsigned sched\_ctx)
- void [sc\\_hypervisor\\_set\\_type\\_of\\_task](#) (struct [starpu\\_codelet](#) \*cl, unsigned sched\_ctx, uint32\_t footprint, size\_t data\_size)
- void [sc\\_hypervisor\\_update\\_diff\\_total\\_flops](#) (unsigned sched\_ctx, double diff\_total\_flops)
- void [sc\\_hypervisor\\_update\\_diff\\_elapsed\\_flops](#) (unsigned sched\_ctx, double diff\_task\_flops)
- void [sc\\_hypervisor\\_update\\_resize\\_interval](#) (unsigned \*sched\_ctxs, int nsched\_ctxs, int max\_nworkers)
- void [sc\\_hypervisor\\_get\\_ctxs\\_on\\_level](#) (unsigned \*\*sched\_ctxs, int \*nsched\_ctxs, unsigned hierarchy\_level, unsigned father\_sched\_ctx\_id)
- unsigned [sc\\_hypervisor\\_get\\_nhierarchy\\_levels](#) (void)
- void [sc\\_hypervisor\\_get\\_leaves](#) (unsigned \*sched\_ctxs, int nsched\_ctxs, unsigned \*leaves, int \*nleaves)
- double [sc\\_hypervisor\\_get\\_nready\\_flops\\_of\\_all\\_sons\\_of\\_sched\\_ctx](#) (unsigned sched\_ctx)
- void [sc\\_hypervisor\\_print\\_overhead](#) ()

## Variables

- `starpu_pthread_mutex_t` [act\\_hypervisor\\_mutex](#)

## 29.40 `sc_hypervisor_config.h` File Reference

```
#include <sc_hypervisor.h>
```

## Data Structures

- struct [sc\\_hypervisor\\_policy\\_config](#)

## Macros

- `#define SC_HYPERVISOR_MAX_IDLE`
- `#define SC_HYPERVISOR_MIN_WORKING`
- `#define SC_HYPERVISOR_PRIORITY`
- `#define SC_HYPERVISOR_MIN_WORKERS`
- `#define SC_HYPERVISOR_MAX_WORKERS`
- `#define SC_HYPERVISOR_GRANULARITY`
- `#define SC_HYPERVISOR_FIXED_WORKERS`
- `#define SC_HYPERVISOR_MIN_TASKS`
- `#define SC_HYPERVISOR_NEW_WORKERS_MAX_IDLE`
- `#define SC_HYPERVISOR_TIME_TO_APPLY`
- `#define SC_HYPERVISOR_NULL`
- `#define SC_HYPERVISOR_ISPEED_W_SAMPLE`
- `#define SC_HYPERVISOR_ISPEED_CTX_SAMPLE`
- `#define SC_HYPERVISOR_TIME_SAMPLE`
- `#define MAX_IDLE_TIME`
- `#define MIN_WORKING_TIME`

## Functions

- void `sc_hypervisor_set_config` (unsigned sched\_ctx, void \*config)
- struct `sc_hypervisor_policy_config` \* `sc_hypervisor_get_config` (unsigned sched\_ctx)
- void `sc_hypervisor_ctl` (unsigned sched\_ctx,...)

## 29.41 sc\_hypervisor\_lp.h File Reference

```
#include <sc_hypervisor.h>
#include <starpu_config.h>
```

## Functions

- double `sc_hypervisor_lp_get_nworkers_per_ctx` (int nsched\_ctxs, int ntypes\_of\_workers, double res[nsched\_ctxs][ntypes\_of\_workers], int total\_nw[ntypes\_of\_workers], struct `types_of_workers` \*tw, unsigned \*in\_sched\_ctxs)
- double `sc_hypervisor_lp_get_tmax` (int nw, int \*workers)
- void `sc_hypervisor_lp_round_double_to_int` (int ns, int nw, double res[ns][nw], int res\_rounded[ns][nw])
- void `sc_hypervisor_lp_redistribute_resources_in_ctxs` (int ns, int nw, int res\_rounded[ns][nw], double res[ns][nw], unsigned \*sched\_ctxs, struct `types_of_workers` \*tw)
- void `sc_hypervisor_lp_distribute_resources_in_ctxs` (unsigned \*sched\_ctxs, int ns, int nw, int res\_rounded[ns][nw], double res[ns][nw], int \*workers, int nworkers, struct `types_of_workers` \*tw)
- void `sc_hypervisor_lp_distribute_floating_no_resources_in_ctxs` (unsigned \*sched\_ctxs, int ns, int nw, double res[ns][nw], int \*workers, int nworkers, struct `types_of_workers` \*tw)
- void `sc_hypervisor_lp_place_resources_in_ctx` (int ns, int nw, double w\_in\_s[ns][nw], unsigned \*sched\_ctxs, int \*workers, unsigned do\_size, struct `types_of_workers` \*tw)
- void `sc_hypervisor_lp_share_remaining_resources` (int ns, unsigned \*sched\_ctxs, int nworkers, int \*workers)
- double `sc_hypervisor_lp_find_tmax` (double t1, double t2)
- unsigned `sc_hypervisor_lp_execute_dichotomy` (int ns, int nw, double w\_in\_s[ns][nw], unsigned solve\_lp\_integer, void \*specific\_data, double tmin, double tmax, double smallest\_tmax, double(\*lp\_estimated\_distrib\_func)(int ns, int nw, double draft\_w\_in\_s[ns][nw], unsigned is\_integer, double tmax, void \*specific\_data))

## 29.42 `sc_hypervisor_monitoring.h` File Reference

```
#include <sc_hypervisor.h>
```

### Data Structures

- struct [sc\\_hypervisor\\_resize\\_ack](#)
- struct [sc\\_hypervisor\\_wrapper](#)

### Functions

- struct [sc\\_hypervisor\\_wrapper](#) \* [sc\\_hypervisor\\_get\\_wrapper](#) (unsigned sched\_ctx)
- unsigned \* [sc\\_hypervisor\\_get\\_sched\\_ctxs](#) ()
- int [sc\\_hypervisor\\_get\\_nsched\\_ctxs](#) ()
- int [sc\\_hypervisor\\_get\\_nworkers\\_ctx](#) (unsigned sched\_ctx, enum [starpu\\_worker\\_archtype](#) arch)
- double [sc\\_hypervisor\\_get\\_elapsed\\_flops\\_per\\_sched\\_ctx](#) (struct [sc\\_hypervisor\\_wrapper](#) \*sc\_w)
- double [sc\\_hypervisor\\_get\\_total\\_elapsed\\_flops\\_per\\_sched\\_ctx](#) (struct [sc\\_hypervisor\\_wrapper](#) \*sc\_w)
- double [sc\\_hypervisor\\_sc\\_hypervisor\\_get\\_speed\\_per\\_worker\\_type](#) (struct [sc\\_hypervisor\\_wrapper](#) \*sc\_w, enum [starpu\\_worker\\_archtype](#) arch)
- double [sc\\_hypervisor\\_get\\_speed](#) (struct [sc\\_hypervisor\\_wrapper](#) \*sc\_w, enum [starpu\\_worker\\_archtype](#) arch)

## 29.43 `sc_hypervisor_policy.h` File Reference

```
#include <sc_hypervisor.h>
```

### Data Structures

- struct [types\\_of\\_workers](#)
- struct [sc\\_hypervisor\\_policy\\_task\\_pool](#)

### Macros

- #define [HYPERVISOR\\_REDIM\\_SAMPLE](#)
- #define [HYPERVISOR\\_START\\_REDIM\\_SAMPLE](#)
- #define [SC\\_NOTHING](#)
- #define [SC\\_IDLE](#)
- #define [SC\\_SPEED](#)

### Functions

- void [sc\\_hypervisor\\_policy\\_add\\_task\\_to\\_pool](#) (struct [starpu\\_codelet](#) \*cl, unsigned sched\_ctx, uint32\_t footprint, struct [sc\\_hypervisor\\_policy\\_task\\_pool](#) \*\*task\_pools, size\_t data\_size)
- void [sc\\_hypervisor\\_policy\\_remove\\_task\\_from\\_pool](#) (struct [starpu\\_task](#) \*task, uint32\_t footprint, struct [sc\\_hypervisor\\_policy\\_task\\_pool](#) \*\*task\_pools)
- struct [sc\\_hypervisor\\_policy\\_task\\_pool](#) \* [sc\\_hypervisor\\_policy\\_clone\\_task\\_pool](#) (struct [sc\\_hypervisor\\_policy\\_task\\_pool](#) \*tp)

- void **sc\_hypervisor\_get\_tasks\_times** (int nw, int nt, double times[nw][nt], int \*workers, unsigned size\_ctxs, struct [sc\\_hypervisor\\_policy\\_task\\_pool](#) \*task\_pools)
- unsigned **sc\_hypervisor\_find\_lowest\_prio\_sched\_ctx** (unsigned req\_sched\_ctx, int nworkers\_to\_move)
- int \* **sc\_hypervisor\_get\_idlest\_workers** (unsigned sched\_ctx, int \*nworkers, enum [starpu\\_worker\\_archtype](#) arch)
- int \* **sc\_hypervisor\_get\_idlest\_workers\_in\_list** (int \*start, int \*workers, int nall\_workers, int \*nworkers, enum [starpu\\_worker\\_archtype](#) arch)
- int **sc\_hypervisor\_get\_movable\_nworkers** (struct [sc\\_hypervisor\\_policy\\_config](#) \*config, unsigned sched\_ctx, enum [starpu\\_worker\\_archtype](#) arch)
- int **sc\_hypervisor\_compute\_nworkers\_to\_move** (unsigned req\_sched\_ctx)
- unsigned **sc\_hypervisor\_policy\_resize** (unsigned sender\_sched\_ctx, unsigned receiver\_sched\_ctx, unsigned force\_resize, unsigned now)
- unsigned **sc\_hypervisor\_policy\_resize\_to\_unknown\_receiver** (unsigned sender\_sched\_ctx, unsigned now)
- double **sc\_hypervisor\_get\_ctx\_speed** (struct [sc\\_hypervisor\\_wrapper](#) \*sc\_w)
- double **sc\_hypervisor\_get\_slowest\_ctx\_exec\_time** (void)
- double **sc\_hypervisor\_get\_fastest\_ctx\_exec\_time** (void)
- double **sc\_hypervisor\_get\_speed\_per\_worker** (struct [sc\\_hypervisor\\_wrapper](#) \*sc\_w, unsigned worker)
- double **sc\_hypervisor\_get\_speed\_per\_worker\_type** (struct [sc\\_hypervisor\\_wrapper](#) \*sc\_w, enum [starpu\\_worker\\_archtype](#) arch)
- double **sc\_hypervisor\_get\_ref\_speed\_per\_worker\_type** (struct [sc\\_hypervisor\\_wrapper](#) \*sc\_w, enum [starpu\\_worker\\_archtype](#) arch)
- double **sc\_hypervisor\_get\_avg\_speed** (enum [starpu\\_worker\\_archtype](#) arch)
- void **sc\_hypervisor\_check\_if\_consider\_max** (struct [types\\_of\\_workers](#) \*tw)
- void **sc\_hypervisor\_group\_workers\_by\_type** (struct [types\\_of\\_workers](#) \*tw, int \*total\_nw)
- enum [starpu\\_worker\\_archtype](#) **sc\_hypervisor\_get\_arch\_for\_index** (unsigned w, struct [types\\_of\\_workers](#) \*tw)
- unsigned **sc\_hypervisor\_get\_index\_for\_arch** (enum [starpu\\_worker\\_archtype](#) arch, struct [types\\_of\\_workers](#) \*tw)
- unsigned **sc\_hypervisor\_criteria\_fulfilled** (unsigned sched\_ctx, int worker)
- unsigned **sc\_hypervisor\_check\_idle** (unsigned sched\_ctx, int worker)
- unsigned **sc\_hypervisor\_check\_speed\_gap\_bt看\_ctxs** (unsigned \*sched\_ctxs, int nsched\_ctxs, int \*workers, int nworkers)
- unsigned **sc\_hypervisor\_check\_speed\_gap\_bt看\_ctxs\_on\_level** (int level, int \*workers\_in, int nworkers\_in, unsigned father\_sched\_ctx\_id, unsigned \*\*sched\_ctxs, int \*nsched\_ctxs)
- unsigned **sc\_hypervisor\_get\_resize\_criteria** ()
- struct [types\\_of\\_workers](#) \* **sc\_hypervisor\_get\_types\_of\_workers** (int \*workers, unsigned nworkers)

## 29.43.1 Data Structure Documentation

### 29.43.1.1 struct types\_of\_workers

#### Data Fields

|          |       |  |
|----------|-------|--|
| unsigned | nopus |  |
| unsigned | ncuda |  |
| unsigned | nw    |  |

## Chapter 30

# Deprecated List

### Global `starpu_codelet::cpu_func`

Optional field which has been made deprecated. One should use instead the field `starpu_codelet::cpu_funcs`.

### Global `starpu_codelet::cuda_func`

Optional field which has been made deprecated. One should use instead the `starpu_codelet::cuda_funcs` field.

### Global `starpu_codelet::opencl_func`

Optional field which has been made deprecated. One should use instead the `starpu_codelet::opencl_funcs` field.

### Global `starpu_data_free_pinned_if_possible`

Equivalent to `starpu_free()`. This macro is provided to avoid breaking old codes.

### Global `starpu_data_malloc_pinned_if_possible`

Equivalent to `starpu_malloc()`. This macro is provided to avoid breaking old codes.

### Global `starpu_mpi_initialize (void)`

This function has been made deprecated. One should use instead the function `starpu_mpi_init()`. This function does not call `MPI_Init()`, it should be called beforehand.

### Global `starpu_mpi_initialize_extended (int *rank, int *world_size)`

This function has been made deprecated. One should use instead the function `starpu_mpi_init()`. MPI will be initialized by `starpumpi` by calling `MPI_Init_Thread(argc, argv, MPI_THREAD_SERIALIZED, ...)`.

### Global `STARPU_MULTIPLE_CPU_IMPLEMENTATIONS`

Setting the field `starpu_codelet::cpu_func` with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field `starpu_codelet::cpu_funcs`.

### Global `STARPU_MULTIPLE_CUDA_IMPLEMENTATIONS`

Setting the field `starpu_codelet::cuda_func` with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field `starpu_codelet::cuda_funcs`.

### Global `STARPU_MULTIPLE_OPENCL_IMPLEMENTATIONS`

Setting the field `starpu_codelet::opencl_func` with this macro indicates the codelet will have several implementations. The use of this macro is deprecated. One should always only define the field `starpu_codelet::opencl_funcs`.



## **Part VI**

# **Appendix**





## Chapter 31

# Full Source Code for the 'Scaling a Vector' Example

### 31.1 Main Application

```
/*
 * This example demonstrates how to use StarPU to scale an array by a factor.
 * It shows how to manipulate data with StarPU's data management library.
 * 1- how to declare a piece of data to StarPU (starpu_vector_data_register)
 * 2- how to describe which data are accessed by a task (task->handles[0])
 * 3- how a kernel can manipulate the data (buffers[0].vector.ptr)
 */
#include <starpu.h>

#define NX 2048

extern void scal_cpu_func(void *buffers[], void *_args);
extern void scal_sse_func(void *buffers[], void *_args);
extern void scal_cuda_func(void *buffers[], void *_args);
extern void scal_opcnl_func(void *buffers[], void *_args);

static struct starpu_codelet cl = {
 .where = STARPU_CPU | STARPU_CUDA | STARPU_OPENCL,
 /* CPU implementation of the codelet */
 .cpu_funcs = { scal_cpu_func, scal_sse_func },
 .cpu_funcs_name = { "scal_cpu_func", "scal_sse_func" },
#ifdef STARPU_USE_CUDA
 /* CUDA implementation of the codelet */
 .cuda_funcs = { scal_cuda_func },
#endif
#ifdef STARPU_USE_OPENCL
 /* OpenCL implementation of the codelet */
 .opcnl_funcs = { scal_opcnl_func },
#endif
 .nbuffers = 1,
 .modes = { STARPU_RW }
};

#ifdef STARPU_USE_OPENCL
struct starpu_opcnl_program programs;
#endif

int main(int argc, char **argv)
{
 /* We consider a vector of float that is initialized just as any of C
 * data */
 float vector[NX];
 unsigned i;
 for (i = 0; i < NX; i++)
 vector[i] = 1.0f;

 fprintf(stderr, "BEFORE: First element was %f\n", vector[0]);

 /* Initialize StarPU with default configuration */
 starpu_init(NULL);

#ifdef STARPU_USE_OPENCL
 starpu_opcnl_load_opcnl_from_file(
 "examples/basic_examples/vector_scal_opcnl_kernel.cl", &programs, NULL);
#endif

 /* Tell StaPU to associate the "vector" vector with the "vector_handle"
 * identifier. When a task needs to access a piece of data, it should
 * refer to the handle that is associated to it.
 */
}
```

```

 * In the case of the "vector" data interface:
 * - the first argument of the registration method is a pointer to the
 * handle that should describe the data
 * - the second argument is the memory node where the data (ie. "vector")
 * resides initially: STARPU_MAIN_RAM stands for an address in main memory, as
 * opposed to an address on a GPU for instance.
 * - the third argument is the address of the vector in RAM
 * - the fourth argument is the number of elements in the vector
 * - the fifth argument is the size of each element.
 */
 starpu_data_handle_t vector_handle;
 starpu_vector_data_register(&vector_handle,
 STARPU_MAIN_RAM, (uintptr_t)vector,
 NX, sizeof(vector[0]));

 float factor = 3.14;

 /* create a synchronous task: any call to starpu_task_submit will block
 * until it is terminated */
 struct starpu_task *task = starpu_task_create();
 task->synchronous = 1;

 task->cl = &cl;

 /* the codelet manipulates one buffer in RW mode */
 task->handles[0] = vector_handle;

 /* an argument is passed to the codelet, beware that this is a
 * READ-ONLY buffer and that the codelet may be given a pointer to a
 * COPY of the argument */
 task->cl_arg = &factor;
 task->cl_arg_size = sizeof(factor);

 /* execute the task on any eligible computational resource */
 starpu_task_submit(task);

 /* StarPU does not need to manipulate the array anymore so we can stop
 * monitoring it */
 starpu_data_unregister(vector_handle);

#ifdef STARPU_USE_OPENCL
 starpu_openccl_unload_openccl(&programs);
#endif

 /* terminate StarPU, no task can be submitted after */
 starpu_shutdown();

 fprintf(stderr, "AFTER First element is %f\n", vector[0]);

 return 0;
}

```

## 31.2 CPU Kernel

```

#include <starpu.h>
#include <xmmintrin.h>

/* This kernel takes a buffer and scales it by a constant factor */
void scal_cpu_func(void *buffers[], void *cl_arg)
{
 unsigned i;
 float *factor = cl_arg;

 /*
 * The "buffers" array matches the task->handles array: for instance
 * task->handles[0] is a handle that corresponds to a data with
 * vector "interface", so that the first entry of the array in the
 * codelet is a pointer to a structure describing such a vector (ie.
 * struct starpu_vector_interface *). Here, we therefore manipulate
 * the buffers[0] element as a vector: nx gives the number of elements
 * in the array, ptr gives the location of the array (that was possibly
 * migrated/replicated), and elemsize gives the size of each elements.
 */
 struct starpu_vector_interface *vector = buffers[0];

 /* length of the vector */
 unsigned n = STARPU_VECTOR_GET_NX(vector);

 /* get a pointer to the local copy of the vector: note that we have to
 * cast it in (float *) since a vector could contain any type of
 * elements so that the .ptr field is actually a uintptr_t */
 float *val = (float *)STARPU_VECTOR_GET_PTR(vector);

```

```

 /* scale the vector */
 for (i = 0; i < n; i++)
 val[i] *= *factor;
}

void scal_sse_func(void *buffers[], void *cl_arg)
{
 float *vector = (float *) STARPU_VECTOR_GET_PTR(buffers[0]);
 unsigned int n = STARPU_VECTOR_GET_NX(buffers[0]);
 unsigned int n_iterations = n/4;

 __m128 *VECTOR = (__m128*) vector;
 __m128 FACTOR = STARPU_ATTRIBUTE_ALIGNED(16);
 float factor = *(float *) cl_arg;
 FACTOR = _mm_set1_ps(factor);

 unsigned int i;
 for (i = 0; i < n_iterations; i++)
 VECTOR[i] = _mm_mul_ps(FACTOR, VECTOR[i]);

 unsigned int remainder = n%4;
 if (remainder != 0)
 {
 unsigned int start = 4 * n_iterations;
 for (i = start; i < start+remainder; ++i)
 {
 vector[i] = factor * vector[i];
 }
 }
}

```

## 31.3 CUDA Kernel

```

1 #include <starpu.h>
2
3 static __global__ void vector_mult_cuda(unsigned n, float *val,
4 float factor)
5 {
6 unsigned i = blockIdx.x*blockDim.x + threadIdx.x;
7 if (i < n)
8 val[i] *= factor;
9 }
10
11 extern "C" void scal_cuda_func(void *buffers[], void *_args)
12 {
13 float *factor = (float *)_args;
14
15 /* length of the vector */
16 unsigned n = STARPU_VECTOR_GET_NX(buffers[0]);
17 /* local copy of the vector pointer */
18 float *val = (float *)STARPU_VECTOR_GET_PTR(buffers[0]);
19 unsigned threads_per_block = 64;
20 unsigned nblocks = (n + threads_per_block-1) / threads_per_block;
21
22 vector_mult_cuda<<<nblocks, threads_per_block, 0, starpu_cuda_get_local_stream())>>>
23 (n, val, *factor);
24
25 cudaStreamSynchronize(starpu_cuda_get_local_stream());
26 }

```

## 31.4 OpenCL Kernel

### 31.4.1 Invoking the Kernel

```

#include <starpu.h>

extern struct starpu_opengl_program programs;

void scal_opengl_func(void *buffers[], void *_args)
{
 float *factor = _args;
 int id, devid, err;
 cl_kernel kernel;
 cl_command_queue queue;
 cl_event event;

 /* OpenCL specific code */
 /* OpenCL specific code */
 /* OpenCL specific code */
 /* OpenCL specific code */

 /* length of the vector */

```

```

unsigned n = STARPU_VECTOR_GET_NX(bufers[0]);
/* OpenCL copy of the vector pointer */
cl_mem val = (cl_mem)STARPU_VECTOR_GET_DEV_HANDLE(bufers[0]);

{ /* OpenCL specific code */
 id = starpu_worker_get_id();
 devid = starpu_worker_get_devid(id);

 err = starpu_opencil_load_kernel(&kernel, &queue,
 &programs,
 "vector_mult_opencil", /* Name of the codelet */
 devid);
 if (err != CL_SUCCESS) STARPU_OPENCIL_REPORT_ERROR(err);

 err = clSetKernelArg(kernel, 0, sizeof(n), &n);
 err |= clSetKernelArg(kernel, 1, sizeof(val), &val);
 err |= clSetKernelArg(kernel, 2, sizeof(*factor), factor);
 if (err) STARPU_OPENCIL_REPORT_ERROR(err);
}

{ /* OpenCL specific code */
 size_t global=n;
 size_t local;
 size_t s;
 cl_device_id device;

 starpu_opencil_get_device(devid, &device);
 err = clGetKernelWorkGroupInfo (kernel, device, CL_KERNEL_WORK_GROUP_SIZE,
 sizeof(local), &local, &s);
 if (err != CL_SUCCESS) STARPU_OPENCIL_REPORT_ERROR(err);
 if (local > global) local=global;

 err = clEnqueueNDRangeKernel(queue, kernel, 1, NULL, &global, &local, 0,
 NULL, &event);
 if (err != CL_SUCCESS) STARPU_OPENCIL_REPORT_ERROR(err);
}

{ /* OpenCL specific code */
 clFinish(queue);
 starpu_opencil_collect_stats(event);
 clReleaseEvent(event);

 starpu_opencil_release_kernel(kernel);
}
}

```

### 31.4.2 Source of the Kernel

```

1 __kernel void vector_mult_opencil(int nx, __global float* val, float factor)
2 {
3 const int i = get_global_id(0);
4 if (i < nx) {
5 val[i] *= factor;
6 }
7 }

```

## Chapter 32

# GNU Free Documentation License

Version 1.3, 3 November 2008

### Copyright

2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

### 1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

### 2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does

not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

### 3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

### 4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

## 5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- (a) Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- (b) List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- (c) State on the Title page the name of the publisher of the Modified Version, as the publisher.
- (d) Preserve all the copyright notices of the Document.
- (e) Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- (f) Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- (g) Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- (h) Include an unaltered copy of this License.
- (i) Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- (j) Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- (k) For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- (l) Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- (m) Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- (n) Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- (o) Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these

sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

## 7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.



If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

#### 10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

#### 11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

#### 12. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

## 32.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) *year your name*. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published

by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

with the Invariant Sections being *list their titles*, with the Front-Cover Texts being *list*, and with the Back-Cover Texts being *list*.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

## **Part VII**

## **Index**



# Index

## \_Configure Options

- \_\_configure\_\_ --disable-asynchronous-copy, [148](#)
- \_\_configure\_\_ --disable-asynchronous-cuda-copy, [148](#)
- \_\_configure\_\_ --disable-asynchronous-mic-copy, [148](#)
- \_\_configure\_\_ --disable-asynchronous-opencl-copy, [148](#)
- \_\_configure\_\_ --disable-build-doc, [147](#)
- \_\_configure\_\_ --disable-build-examples, [150](#)
- \_\_configure\_\_ --disable-cpu, [148](#)
- \_\_configure\_\_ --disable-cuda, [148](#)
- \_\_configure\_\_ --disable-cuda-memcpy-peer, [148](#)
- \_\_configure\_\_ --disable-gcc-extensions, [149](#)
- \_\_configure\_\_ --disable-opencl, [148](#)
- \_\_configure\_\_ --disable-socl, [149](#)
- \_\_configure\_\_ --disable-starpu-top, [149](#)
- \_\_configure\_\_ --disable-starpuftt, [149](#)
- \_\_configure\_\_ --enable-allocation-cache, [149](#)
- \_\_configure\_\_ --enable-blas-lib, [149](#)
- \_\_configure\_\_ --enable-calibration-heuristic, [150](#)
- \_\_configure\_\_ --enable-coverage, [147](#)
- \_\_configure\_\_ --enable-debug, [147](#)
- \_\_configure\_\_ --enable-fast, [147](#)
- \_\_configure\_\_ --enable-fxt-lock, [149](#)
- \_\_configure\_\_ --enable-long-check, [147](#)
- \_\_configure\_\_ --enable-max-sched-ctxs, [148](#)
- \_\_configure\_\_ --enable-maxbuffers, [149](#)
- \_\_configure\_\_ --enable-maxcpus, [148](#)
- \_\_configure\_\_ --enable-maxcudadev, [148](#)
- \_\_configure\_\_ --enable-maximplementations, [148](#)
- \_\_configure\_\_ --enable-maxmicthreads, [148](#)
- \_\_configure\_\_ --enable-maxopencldev, [148](#)
- \_\_configure\_\_ --enable-memory-stats, [150](#)
- \_\_configure\_\_ --enable-model-debug, [149](#)
- \_\_configure\_\_ --enable-mpi-progression-hook, [149](#)
- \_\_configure\_\_ --enable-new-check, [147](#)
- \_\_configure\_\_ --enable-opencl-simulator, [148](#)
- \_\_configure\_\_ --enable-opengl-render, [149](#)
- \_\_configure\_\_ --enable-openmp, [149](#)
- \_\_configure\_\_ --enable-paje-codelet-details, [149](#)
- \_\_configure\_\_ --enable-perf-debug, [149](#)
- \_\_configure\_\_ --enable-quick-check, [147](#)
- \_\_configure\_\_ --enable-sc-hypervisor, [150](#)
- \_\_configure\_\_ --enable-simgrid, [150](#)
- \_\_configure\_\_ --enable-spinlock-check, [147](#)
- \_\_configure\_\_ --enable-starpuftt-examples, [149](#)
- \_\_configure\_\_ --enable-stats, [149](#)
- \_\_configure\_\_ --enable-verbose, [147](#)

- \_\_configure\_\_ --mic-host, [149](#)
- \_\_configure\_\_ --with-atlas-dir, [149](#)
- \_\_configure\_\_ --with-coi-dir, [149](#)
- \_\_configure\_\_ --with-cuda-dir, [148](#)
- \_\_configure\_\_ --with-cuda-include-dir, [148](#)
- \_\_configure\_\_ --with-cuda-lib-dir, [148](#)
- \_\_configure\_\_ --with-fxt, [149](#)
- \_\_configure\_\_ --with-goto-dir, [149](#)
- \_\_configure\_\_ --with-hwloc, [147](#)
- \_\_configure\_\_ --with-magma, [149](#)
- \_\_configure\_\_ --with-mkl-cflags, [150](#)
- \_\_configure\_\_ --with-mkl-ldflags, [150](#)
- \_\_configure\_\_ --with-mpicc, [149](#)
- \_\_configure\_\_ --with-opencl-dir, [148](#)
- \_\_configure\_\_ --with-opencl-include-dir, [148](#)
- \_\_configure\_\_ --with-opencl-lib-dir, [148](#)
- \_\_configure\_\_ --with-perf-model-dir, [149](#)
- \_\_configure\_\_ --with-simgrid-dir, [150](#)
- \_\_configure\_\_ --with-simgrid-include-dir, [150](#)
- \_\_configure\_\_ --with-simgrid-lib-dir, [150](#)
- \_\_configure\_\_ --without-hwloc, [147](#)

## \_Environment Variables

- \_\_env\_\_ OCL\_ICD\_VENDORS, [144](#)
- \_\_env\_\_ SC\_HYPERVISOR\_LAZY\_RESIZE, [146](#)
- \_\_env\_\_ SC\_HYPERVISOR\_MAX\_SPEED\_GAP, [146](#)
- \_\_env\_\_ SC\_HYPERVISOR\_POLICY, [146](#)
- \_\_env\_\_ SC\_HYPERVISOR\_SAMPLE\_CRITERIA, [146](#)
- \_\_env\_\_ SC\_HYPERVISOR\_START\_RESIZE, [146](#)
- \_\_env\_\_ SC\_HYPERVISOR\_STOP\_PRINT, [146](#)
- \_\_env\_\_ SC\_HYPERVISOR\_TRIGGER\_RESIZE, [146](#)
- \_\_env\_\_ SOCL\_OCL\_LIB\_OPENCL, [144](#)
- \_\_env\_\_ STARPU\_BUS\_CALIBRATE, [143](#)
- \_\_env\_\_ STARPU\_BUS\_STATS, [145](#)
- \_\_env\_\_ STARPU\_CALIBRATE, [143](#)
- \_\_env\_\_ STARPU\_CALIBRATE\_MINIMUM, [143](#)
- \_\_env\_\_ STARPU\_COMM\_STATS, [144](#)
- \_\_env\_\_ STARPU\_CUDA\_PIPELINE, [141](#)
- \_\_env\_\_ STARPU\_DISABLE\_ASYNCHRONOUS\_COPY, [143](#)
- \_\_env\_\_ STARPU\_DISABLE\_ASYNCHRONOUS\_CUDA\_COPY, [143](#)
- \_\_env\_\_ STARPU\_DISABLE\_ASYNCHRONOUS\_MIC\_COPY, [143](#)
- \_\_env\_\_ STARPU\_DISABLE\_ASYNCHRONOUS\_OPENCL\_COPY, [143](#)
- \_\_env\_\_ STARPU\_DISABLE\_KERNELS, [145](#)

- `__env__STARPU_DISABLE_PINNING`, 143
  - `__env__STARPU_DISK_SWAP`, 145
  - `__env__STARPU_DISK_SWAP_BACKEND`, 145
  - `__env__STARPU_DISK_SWAP_SIZE`, 145
  - `__env__STARPU_ENABLE_CUDA_GPU_GPU_↔  
DIRECT`, 143
  - `__env__STARPU_FXT_PREFIX`, 144
  - `__env__STARPU_GENERATE_TRACE`, 145
  - `__env__STARPU_HISTORY_MAX_ERROR`, 145
  - `__env__STARPU_HOME`, 144
  - `__env__STARPU_HOSTNAME`, 144
  - `__env__STARPU_IDLE_POWER`, 143
  - `__env__STARPU_LIMIT_CPU_MEM`, 145
  - `__env__STARPU_LIMIT_CUDA_devid_MEM`, 144
  - `__env__STARPU_LIMIT_CUDA_MEM`, 144
  - `__env__STARPU_LIMIT_OPENCL_devid_MEM`,  
144
  - `__env__STARPU_LIMIT_OPENCL_MEM`, 145
  - `__env__STARPU_LOGFILENAME`, 144
  - `__env__STARPU_MAX_WORKERSIZE`, 142
  - `__env__STARPU_MEMORY_STATS`, 145
  - `__env__STARPU_MIN_WORKERSIZE`, 142
  - `__env__STARPU_MINIMUM_AVAILABLE_MEM`,  
145
  - `__env__STARPU_MPI_CACHE`, 144
  - `__env__STARPU_MPI_CACHE_STATS`, 144
  - `__env__STARPU_NCPU`, 141
  - `__env__STARPU_NCPUS`, 141
  - `__env__STARPU_NCUDA`, 141
  - `__env__STARPU_NMIC`, 141
  - `__env__STARPU_NMICDEVS`, 141
  - `__env__STARPU_NOPENCL`, 141
  - `__env__STARPU_NSCC`, 142
  - `__env__STARPU_NWORKER_PER_CUDA`, 141
  - `__env__STARPU_OPENCL_ON_CPUS`, 141
  - `__env__STARPU_OPENCL_ONLY_ON_CPUS`,  
141
  - `__env__STARPU_OPENCL_PIPELINE`, 141
  - `__env__STARPU_OPENCL_PROGRAM_DIR`,  
144
  - `__env__STARPU_PERF_MODEL_DIR`, 144
  - `__env__STARPU_PREFETCH`, 143
  - `__env__STARPU_PROFILING`, 143
  - `__env__STARPU_SCHED`, 143
  - `__env__STARPU_SCHED_ALPHA`, 143
  - `__env__STARPU_SCHED_BETA`, 143
  - `__env__STARPU_SCHED_GAMMA`, 143
  - `__env__STARPU_SILENT`, 144
  - `__env__STARPU_SINGLE_COMBINED_↔  
WORKER`, 142
  - `__env__STARPU_STATS`, 145
  - `__env__STARPU_SYNTHESIZE_ARITY_↔  
COMBINED_WORKER`, 142
  - `__env__STARPU_TARGET_AVAILABLE_MEM`,  
145
  - `__env__STARPU_TRACE_BUFFER_SIZE`, 145
  - `__env__STARPU_WATCHDOG_CRASH`, 145
  - `__env__STARPU_WATCHDOG_TIMEOUT`, 145
  - `__env__STARPU_WORKER_STATS`, 145
  - `__env__STARPU_WORKERS_CPUID`, 142
  - `__env__STARPU_WORKERS_CUDAID`, 142
  - `__env__STARPU_WORKERS_MICID`, 142
  - `__env__STARPU_WORKERS_NOBIND`, 142
  - `__env__STARPU_WORKERS_OPENCLID`, 142
  - `__env__STARPU_WORKERS_SCCID`, 142
- Bitmap, 171
- Codelet And Tasks, 217
- STARPU\_FORKJOIN, 230
  - STARPU\_SEQ, 230
  - STARPU\_SPMD, 230
  - STARPU\_TASK\_BLOCKED, 230
  - STARPU\_TASK\_BLOCKED\_ON\_DATA, 230
  - STARPU\_TASK\_BLOCKED\_ON\_TAG, 230
  - STARPU\_TASK\_BLOCKED\_ON\_TASK, 230
  - STARPU\_TASK\_FINISHED, 230
  - STARPU\_TASK\_READY, 230
  - STARPU\_TASK\_RUNNING, 230
- Data Interfaces, 183
- STARPU\_BCSR\_INTERFACE\_ID, 200
  - STARPU\_BLOCK\_INTERFACE\_ID, 200
  - STARPU\_COO\_INTERFACE\_ID, 200
  - STARPU\_CSR\_INTERFACE\_ID, 200
  - STARPU\_MATRIX\_INTERFACE\_ID, 200
  - STARPU\_MAX\_INTERFACE\_ID, 200
  - STARPU\_MULTIFORMAT\_INTERFACE\_ID, 200
  - STARPU\_UNKNOWN\_INTERFACE\_ID, 200
  - STARPU\_VARIABLE\_INTERFACE\_ID, 200
  - STARPU\_VECTOR\_INTERFACE\_ID, 200
  - STARPU\_VOID\_INTERFACE\_ID, 200
- Data Management, 178
- STARPU\_COMMUTE, 180
  - STARPU\_NONE, 180
  - STARPU\_R, 180
  - STARPU\_REDUX, 180
  - STARPU\_RW, 180
  - STARPU\_SCRATCH, 180
  - STARPU\_SSEND, 180
  - STARPU\_W, 180
- Data Partition, 206
- Expert Mode, 294
- Explicit Dependencies, 235
- Implicit Data Dependencies, 237
- Initialization and Termination, 154
- Miscellaneous Helpers, 278
- Modularized Scheduler Interface, 321
- STARPU\_SCHED\_COMPONENT\_HOMOGEN\_↔  
EOUS, 327
  - STARPU\_SCHED\_COMPONENT\_SINGLE\_ME\_↔  
MORY\_NODE, 327
- Multiformat Data Interface, 215
- OpenMP Runtime Support

- starpu\_omp\_proc\_bind\_close, [261](#)
  - starpu\_omp\_proc\_bind\_false, [261](#)
  - starpu\_omp\_proc\_bind\_master, [261](#)
  - starpu\_omp\_proc\_bind\_spread, [261](#)
  - starpu\_omp\_proc\_bind\_true, [261](#)
  - starpu\_omp\_proc\_bind\_undefined, [261](#)
  - starpu\_omp\_sched\_auto, [260](#)
  - starpu\_omp\_sched\_dynamic, [260](#)
  - starpu\_omp\_sched\_guided, [260](#)
  - starpu\_omp\_sched\_runtime, [260](#)
  - starpu\_omp\_sched\_static, [260](#)
  - starpu\_omp\_sched\_undefined, [260](#)
- Out Of Core, [212](#)
- Parallel Tasks, [292](#)
- Performance Model, [238](#)
  - STARPU\_COMMON, [242](#)
  - STARPU\_HISTORY\_BASED, [242](#)
  - STARPU\_NL\_REGRESSION\_BASED, [242](#)
  - STARPU\_PER\_ARCH, [242](#)
  - STARPU\_REGRESSION\_BASED, [242](#)
- Profiling, [244](#)
- Running Drivers, [294](#)
- STARPU\_ANY\_WORKER
  - Workers' Properties, [176](#)
- STARPU\_BCSR\_INTERFACE\_ID
  - Data Interfaces, [200](#)
- STARPU\_BLOCK\_INTERFACE\_ID
  - Data Interfaces, [200](#)
- STARPU\_COMMON
  - Performance Model, [242](#)
- STARPU\_COMMUTE
  - Data Management, [180](#)
- STARPU\_COO\_INTERFACE\_ID
  - Data Interfaces, [200](#)
- STARPU\_CPU\_RAM
  - Workers' Properties, [175](#)
- STARPU\_CPU\_WORKER
  - Workers' Properties, [176](#)
- STARPU\_CSR\_INTERFACE\_ID
  - Data Interfaces, [200](#)
- STARPU\_CUDA\_RAM
  - Workers' Properties, [175](#)
- STARPU\_CUDA\_WORKER
  - Workers' Properties, [176](#)
- STARPU\_FORKJOIN
  - Codelet And Tasks, [230](#)
- STARPU\_HISTORY\_BASED
  - Performance Model, [242](#)
- STARPU\_MATRIX\_INTERFACE\_ID
  - Data Interfaces, [200](#)
- STARPU\_MAX\_INTERFACE\_ID
  - Data Interfaces, [200](#)
- STARPU\_MIC\_RAM
  - Workers' Properties, [175](#)
- STARPU\_MIC\_WORKER
  - Workers' Properties, [176](#)
- STARPU\_MULTIFORMAT\_INTERFACE\_ID
  - Data Interfaces, [200](#)
- STARPU\_NL\_REGRESSION\_BASED
  - Performance Model, [242](#)
- STARPU\_NONE
  - Data Management, [180](#)
- STARPU\_OPENCL\_RAM
  - Workers' Properties, [175](#)
- STARPU\_OPENCL\_WORKER
  - Workers' Properties, [176](#)
- STARPU\_PER\_ARCH
  - Performance Model, [242](#)
- STARPU\_R
  - Data Management, [180](#)
- STARPU\_REDUX
  - Data Management, [180](#)
- STARPU\_REGRESSION\_BASED
  - Performance Model, [242](#)
- STARPU\_RW
  - Data Management, [180](#)
- STARPU\_SCC\_RAM
  - Workers' Properties, [175](#)
- STARPU\_SCC\_SHM
  - Workers' Properties, [175](#)
- STARPU\_SCC\_WORKER
  - Workers' Properties, [176](#)
- STARPU\_SCHED\_COMPONENT\_HOMOGENEOUS
  - Modularized Scheduler Interface, [327](#)
- STARPU\_SCHED\_COMPONENT\_SINGLE\_MEMORY\_NODE
  - Modularized Scheduler Interface, [327](#)
- STARPU\_SCRATCH
  - Data Management, [180](#)
- STARPU\_SEQ
  - Codelet And Tasks, [230](#)
- STARPU\_SPMD
  - Codelet And Tasks, [230](#)
- STARPU\_SSEND
  - Data Management, [180](#)
- STARPU\_TASK\_BLOCKED
  - Codelet And Tasks, [230](#)
- STARPU\_TASK\_BLOCKED\_ON\_DATA
  - Codelet And Tasks, [230](#)
- STARPU\_TASK\_BLOCKED\_ON\_TAG
  - Codelet And Tasks, [230](#)
- STARPU\_TASK\_BLOCKED\_ON\_TASK
  - Codelet And Tasks, [230](#)
- STARPU\_TASK\_FINISHED
  - Codelet And Tasks, [230](#)
- STARPU\_TASK\_READY
  - Codelet And Tasks, [230](#)
- STARPU\_TASK\_RUNNING
  - Codelet And Tasks, [230](#)
- STARPU\_TOP\_DATA\_BOOLEAN
  - StarPU-Top Interface, [297](#)
- STARPU\_TOP\_DATA\_FLOAT
  - StarPU-Top Interface, [297](#)
- STARPU\_TOP\_DATA\_INTEGER

- StarPU-Top Interface, [297](#)
- STARPU\_TOP\_PARAM\_BOOLEAN
  - StarPU-Top Interface, [298](#)
- STARPU\_TOP\_PARAM\_ENUM
  - StarPU-Top Interface, [298](#)
- STARPU\_TOP\_PARAM\_FLOAT
  - StarPU-Top Interface, [298](#)
- STARPU\_TOP\_PARAM\_INTEGER
  - StarPU-Top Interface, [298](#)
- STARPU\_UNKNOWN\_INTERFACE\_ID
  - Data Interfaces, [200](#)
- STARPU\_UNUSED
  - Workers' Properties, [175](#)
- STARPU\_VARIABLE\_INTERFACE\_ID
  - Data Interfaces, [200](#)
- STARPU\_VECTOR\_INTERFACE\_ID
  - Data Interfaces, [200](#)
- STARPU\_VOID\_INTERFACE\_ID
  - Data Interfaces, [200](#)
- STARPU\_W
  - Data Management, [180](#)
- STARPU\_WORKER\_LIST
  - Workers' Properties, [176](#)
- sc\_hypervisor\_policy, [316](#)
- sc\_hypervisor\_policy\_config, [317](#)
- sc\_hypervisor\_policy\_task\_pool, [320](#)
- sc\_hypervisor\_resize\_ack, [320](#)
- sc\_hypervisor\_wrapper, [318](#)
- Scheduling Context Hypervisor - Building a new resizing policy, [316](#)
- Scheduling Context Hypervisor - Regular usage, [312](#)
- Scheduling Contexts, [300](#)
- Scheduling Policy, [306](#)
- Standard Memory Library, [158](#)
- StarPU-Top Interface
  - STARPU\_TOP\_DATA\_BOOLEAN, [297](#)
  - STARPU\_TOP\_DATA\_FLOAT, [297](#)
  - STARPU\_TOP\_DATA\_INTEGER, [297](#)
  - STARPU\_TOP\_PARAM\_BOOLEAN, [298](#)
  - STARPU\_TOP\_PARAM\_ENUM, [298](#)
  - STARPU\_TOP\_PARAM\_FLOAT, [298](#)
  - STARPU\_TOP\_PARAM\_INTEGER, [298](#)
  - TOP\_TYPE\_CONTINUE, [298](#)
  - TOP\_TYPE\_DEBUG, [298](#)
  - TOP\_TYPE\_DISABLE, [298](#)
  - TOP\_TYPE\_ENABLE, [298](#)
  - TOP\_TYPE\_GO, [298](#)
  - TOP\_TYPE\_SET, [298](#)
  - TOP\_TYPE\_UNKNOW, [298](#)
- starpu\_bcsr\_interface, [193](#)
- starpu\_block\_interface, [193](#)
- starpu\_codelet, [218](#)
- starpu\_conf, [154](#)
- starpu\_coo\_interface, [194](#)
- starpu\_csr\_interface, [194](#)
- starpu\_data\_copy\_methods, [188](#)
- starpu\_data\_descr, [222](#)
- starpu\_data\_filter, [207](#)
- starpu\_data\_interface\_ops, [187](#)
- starpu\_disk\_ops, [212](#)
- starpu\_driver, [154](#)
- starpu\_driver.id, [154](#)
- starpu\_fxt\_codelet\_event, [279](#)
- starpu\_fxt\_options, [280](#)
- starpu\_matrix\_interface, [193](#)
- starpu\_multiformat\_data\_interface\_ops, [215](#)
- starpu\_multiformat\_interface, [216](#)
- starpu\_omp\_lock\_t, [258](#)
- starpu\_omp\_nest\_lock\_t, [258](#)
- starpu\_omp\_parallel\_region\_attr, [259](#)
- starpu\_omp\_proc\_bind\_close
  - OpenMP Runtime Support, [261](#)
- starpu\_omp\_proc\_bind\_false
  - OpenMP Runtime Support, [261](#)
- starpu\_omp\_proc\_bind\_master
  - OpenMP Runtime Support, [261](#)
- starpu\_omp\_proc\_bind\_spread
  - OpenMP Runtime Support, [261](#)
- starpu\_omp\_proc\_bind\_true
  - OpenMP Runtime Support, [261](#)
- starpu\_omp\_proc\_bind\_undefined
  - OpenMP Runtime Support, [261](#)
- starpu\_omp\_sched\_auto
  - OpenMP Runtime Support, [260](#)
- starpu\_omp\_sched\_dynamic
  - OpenMP Runtime Support, [260](#)
- starpu\_omp\_sched\_guided
  - OpenMP Runtime Support, [260](#)
- starpu\_omp\_sched\_runtime
  - OpenMP Runtime Support, [260](#)
- starpu\_omp\_sched\_static
  - OpenMP Runtime Support, [260](#)
- starpu\_omp\_sched\_undefined
  - OpenMP Runtime Support, [260](#)
- starpu\_omp\_task\_region\_attr, [259](#)
- starpu\_opencl\_program, [252](#)
- starpu\_perfmodel, [239](#)
- starpu\_perfmodel\_arch, [354](#)
- starpu\_perfmodel\_device, [354](#)
- starpu\_perfmodel\_history\_entry, [242](#)
- starpu\_perfmodel\_history\_list, [241](#)
- starpu\_perfmodel\_per\_arch, [241](#)
- starpu\_perfmodel\_regression\_model, [240](#)
- starpu\_profiling\_bus\_info, [246](#)
- starpu\_profiling\_task\_info, [245](#)
- starpu\_profiling\_worker\_info, [245](#)
- starpu\_pthread\_barrier\_t, [364](#)
- starpu\_pthread\_spinlock\_t, [365](#)
- starpu\_sched\_component, [324](#)
- starpu\_sched\_component\_composed\_recipe, [326](#)
- starpu\_sched\_component\_specs, [326](#)
- starpu\_sched\_ctx\_iterator, [175](#)
- starpu\_sched\_ctx\_performance\_counters, [301](#)
- starpu\_sched\_policy, [307](#)
- starpu\_sched\_tree, [326](#)
- starpu\_task, [222](#)



- starpu\_task\_list, [291](#)
- starpu\_top\_data, [296](#)
- starpu\_top\_param, [296](#)
- starpu\_tree, [311](#)
- starpu\_variable\_interface, [192](#)
- starpu\_vector\_interface, [192](#)
- starpu\_worker\_collection, [174](#)
  
- TOP\_TYPE\_CONTINUE
  - StarPU-Top Interface, [298](#)
- TOP\_TYPE\_DEBUG
  - StarPU-Top Interface, [298](#)
- TOP\_TYPE\_DISABLE
  - StarPU-Top Interface, [298](#)
- TOP\_TYPE\_ENABLE
  - StarPU-Top Interface, [298](#)
- TOP\_TYPE\_GO
  - StarPU-Top Interface, [298](#)
- TOP\_TYPE\_SET
  - StarPU-Top Interface, [298](#)
- TOP\_TYPE\_UNKNOW
  - StarPU-Top Interface, [298](#)
- Task Bundles, [289](#)
- Task Lists, [291](#)
- Theoretical Lower Bound on Execution Time, [248](#)
- Threads, [163](#)
- Toolbox, [161](#)
- Tree, [311](#)
- types\_of\_workers, [374](#)
  
- Versioning, [153](#)
  
- Workers' Properties, [173](#)
  - STARPU\_ANY\_WORKER, [176](#)
  - STARPU\_CPU\_RAM, [175](#)
  - STARPU\_CPU\_WORKER, [176](#)
  - STARPU\_CUDA\_RAM, [175](#)
  - STARPU\_CUDA\_WORKER, [176](#)
  - STARPU\_MIC\_RAM, [175](#)
  - STARPU\_MIC\_WORKER, [176](#)
  - STARPU\_OPENCL\_RAM, [175](#)
  - STARPU\_OPENCL\_WORKER, [176](#)
  - STARPU\_SCC\_RAM, [175](#)
  - STARPU\_SCC\_SHM, [175](#)
  - STARPU\_SCC\_WORKER, [176](#)
  - STARPU\_UNUSED, [175](#)
  - STARPU\_WORKER\_LIST, [176](#)