

# GNU Offloading and Multi Processing Runtime Library

---

The GNU OpenMP and OpenACC Implementation

---

Published by the Free Software Foundation  
51 Franklin Street, Fifth Floor  
Boston, MA 02110-1301, USA

Copyright © 2006-2025 Free Software Foundation, Inc.

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 the Invariant Sections being “Funding Free Software”, the Front-Cover texts being (a) (see below), and with the Back-Cover Texts being (b) (see below). A copy of the license is included in the section entitled “GNU Free Documentation License”.

(a) The FSF’s Front-Cover Text is:

A GNU Manual

(b) The FSF’s Back-Cover Text is:

You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development.

## Short Contents

1	Enabling OpenMP.....	1
2	OpenMP Implementation Status .....	3
3	OpenMP Runtime Library Routines .....	15
4	OpenMP Environment Variables .....	59
5	Enabling OpenACC.....	71
6	OpenACC Runtime Library Routines .....	73
7	OpenACC Environment Variables .....	93
8	CUDA Streams Usage.....	95
9	OpenACC Library Interoperability .....	97
10	OpenACC Profiling Interface .....	101
11	OpenMP-Implementation Specifics.....	107
12	Offload-Target Specifics.....	111
13	The libgomp ABI.....	117
14	Reporting Bugs .....	123
	GNU General Public License .....	125
	GNU Free Documentation License.....	137
	Funding Free Software .....	145
	Library Index .....	147



# Table of Contents

<b>1</b>	<b>Enabling OpenMP .....</b>	<b>1</b>
<b>2</b>	<b>OpenMP Implementation Status .....</b>	<b>3</b>
2.1	OpenMP 4.5 .....	3
2.2	OpenMP 5.0 .....	3
	New features listed in Appendix B of the OpenMP specification ...	3
	Other new OpenMP 5.0 features .....	5
2.3	OpenMP 5.1 .....	5
	New features listed in Appendix B of the OpenMP specification ...	5
	Other new OpenMP 5.1 features .....	6
2.4	OpenMP 5.2 .....	7
	New features listed in Appendix B of the OpenMP specification ...	7
	Other new OpenMP 5.2 features .....	8
2.5	OpenMP 6.0 .....	9
	New features listed in Appendix B of the OpenMP specification ...	9
	Deprecated features, unless listed above .....	12
	Other new OpenMP 6.0 features .....	12
<b>3</b>	<b>OpenMP Runtime Library Routines .....</b>	<b>15</b>
3.1	Thread Team Routines .....	15
3.1.1	omp_set_num_threads – Set upper team size limit .....	15
3.1.2	omp_get_num_threads – Size of the active team .....	15
3.1.3	omp_get_max_threads – Maximum number of threads of parallel region .....	16
3.1.4	omp_get_thread_num – Current thread ID .....	16
3.1.5	omp_in_parallel – Whether a parallel region is active ...	16
3.1.6	omp_set_dynamic – Enable/disable dynamic teams .....	17
3.1.7	omp_get_dynamic – Dynamic teams setting .....	17
3.1.8	omp_get_cancellation – Whether cancellation support is enabled .....	17
3.1.9	omp_set_nested – Enable/disable nested parallel regions ..	18
3.1.10	omp_get_nested – Nested parallel regions .....	18
3.1.11	omp_set_schedule – Set the runtime scheduling method ..	19
3.1.12	omp_get_schedule – Obtain the runtime scheduling method ..	19
3.1.13	omp_get_teams_thread_limit – Maximum number of threads imposed by teams .....	20
3.1.14	omp_get_supported_active_levels – Maximum number of active regions supported .....	20
3.1.15	omp_set_max_active_levels – Limits the number of active parallel regions .....	20
3.1.16	omp_get_max_active_levels – Current maximum number of active regions .....	21

3.1.17	<code>omp_get_level</code> – Obtain the current nesting level.....	21
3.1.18	<code>omp_get_ancestor_thread_num</code> – Ancestor thread ID ...	21
3.1.19	<code>omp_get_team_size</code> – Number of threads in a team.....	22
3.1.20	<code>omp_get_active_level</code> – Number of parallel regions.....	22
3.2	Thread Affinity Routines .....	22
3.2.1	<code>omp_get_proc_bind</code> – Whether threads may be moved between CPUs .....	22
3.3	Teams Region Routines.....	23
3.3.1	<code>omp_get_num_teams</code> – Number of teams .....	23
3.3.2	<code>omp_get_team_num</code> – Get team number .....	23
3.3.3	<code>omp_set_num_teams</code> – Set upper teams limit for teams construct.....	23
3.3.4	<code>omp_get_max_teams</code> – Maximum number of teams of teams region .....	24
3.3.5	<code>omp_set_teams_thread_limit</code> – Set upper thread limit for teams construct .....	24
3.3.6	<code>omp_get_thread_limit</code> – Maximum number of threads ...	24
3.4	Tasking Routines.....	25
3.4.1	<code>omp_get_max_task_priority</code> – Maximum priority value ..	25
3.4.2	<code>omp_in_explicit_task</code> – Whether a given task is an explicit task.....	25
3.4.3	<code>omp_in_final</code> – Whether in final or included task region..	25
3.5	Resource Relinquishing Routines.....	26
3.5.1	<code>omp_pause_resource</code> – Release OpenMP resources on a device .....	26
3.5.2	<code>omp_pause_resource_all</code> – Release OpenMP resources on all devices .....	26
3.6	Device Information Routines.....	27
3.6.1	<code>omp_get_num_procs</code> – Number of processors online.....	27
3.6.2	<code>omp_set_default_device</code> – Set the default device for target regions .....	27
3.6.3	<code>omp_get_default_device</code> – Get the default device for target regions .....	27
3.6.4	<code>omp_get_num_devices</code> – Number of target devices .....	28
3.6.5	<code>omp_get_device_num</code> – Return device number of current device .....	28
3.6.6	<code>omp_get_device_from_uid</code> – Obtain the device number to a unique id.....	28
3.6.7	<code>omp_get_uid_from_device</code> – Obtain the unique id of a device.....	29
3.6.8	<code>omp_is_initial_device</code> – Whether executing on the host device.....	29
3.6.9	<code>omp_get_initial_device</code> – Return device number of initial device.....	30
3.7	Device Memory Routines.....	30
3.7.1	<code>omp_target_alloc</code> – Allocate device memory .....	30
3.7.2	<code>omp_target_free</code> – Free device memory .....	31

3.7.3	<code>omp_target_is_present</code> – Check whether storage is mapped ..	31
3.7.4	<code>omp_target_is_accessible</code> – Check whether memory is device accessible .....	32
3.7.5	<code>omp_target_memcpy</code> – Copy data between devices .....	33
3.7.6	<code>omp_target_memcpy_async</code> – Copy data between devices asynchronously .....	34
3.7.7	<code>omp_target_memcpy_rect</code> – Copy a subvolume of data between devices .....	35
3.7.8	<code>omp_target_memcpy_rect_async</code> – Copy a subvolume of data between devices asynchronously .....	36
3.7.9	<code>omp_target_memset</code> – Set bytes in device memory .....	37
3.7.10	<code>omp_target_memset</code> – Set bytes in device memory asynchronously .....	38
3.7.11	<code>omp_target_associate_ptr</code> – Associate a device pointer with a host pointer .....	39
3.7.12	<code>omp_target_disassociate_ptr</code> – Remove device–host pointer association .....	40
3.7.13	<code>omp_get_mapped_ptr</code> – Return device pointer to a host pointer .....	40
3.8	Lock Routines .....	41
3.8.1	<code>omp_init_lock</code> – Initialize simple lock .....	41
3.8.2	<code>omp_init_nest_lock</code> – Initialize nested lock .....	41
3.8.3	<code>omp_destroy_lock</code> – Destroy simple lock .....	42
3.8.4	<code>omp_destroy_nest_lock</code> – Destroy nested lock .....	42
3.8.5	<code>omp_set_lock</code> – Wait for and set simple lock .....	42
3.8.6	<code>omp_set_nest_lock</code> – Wait for and set nested lock .....	43
3.8.7	<code>omp_unset_lock</code> – Unset simple lock .....	43
3.8.8	<code>omp_unset_nest_lock</code> – Unset nested lock .....	43
3.8.9	<code>omp_test_lock</code> – Test and set simple lock if available .....	44
3.8.10	<code>omp_test_nest_lock</code> – Test and set nested lock if available ..	44
3.9	Timing Routines .....	44
3.9.1	<code>omp_get_wtick</code> – Get timer precision .....	45
3.9.2	<code>omp_get_wtime</code> – Elapsed wall clock time .....	45
3.10	Event Routine .....	45
3.10.1	<code>omp_fulfill_event</code> – Fulfill and destroy an OpenMP event ..	45
3.11	Interoperability Routines .....	46
3.11.1	<code>omp_get_num_interop_properties</code> – Get the number of implementation-specific properties .....	46
3.11.2	<code>omp_get_interop_int</code> – Obtain integer-valued interoperability property .....	46
3.11.3	<code>omp_get_interop_ptr</code> – Obtain pointer-valued interoperability property .....	47
3.11.4	<code>omp_get_interop_str</code> – Obtain string-valued interoperability property .....	48
3.11.5	<code>omp_get_interop_name</code> – Obtain the name of an interop_property value as string .....	48

3.11.6	<code>omp_get_interop_type_desc</code> – Obtain type and description to an <code>interop_property</code> .....	49
3.11.7	<code>omp_get_interop_rc_desc</code> – Obtain error string to an <code>interop_rc</code> error code .....	49
3.12	Memory Management Routines .....	50
3.12.1	<code>omp_init_allocator</code> – Create an allocator .....	50
3.12.2	<code>omp_destroy_allocator</code> – Destroy an allocator .....	51
3.12.3	<code>omp_set_default_allocator</code> – Set the default allocator ..	51
3.12.4	<code>omp_get_default_allocator</code> – Get the default allocator ..	52
3.12.5	<code>omp_alloc</code> – Memory allocation with an allocator .....	52
3.12.6	<code>omp_aligned_alloc</code> – Memory allocation with an allocator and alignment .....	53
3.12.7	<code>omp_free</code> – Freeing memory allocated with OpenMP routines .....	54
3.12.8	<code>omp_calloc</code> – Allocate nullified memory with an allocator ..	54
3.12.9	<code>omp_aligned_calloc</code> – Allocate aligned nullified memory with an allocator .....	55
3.12.10	<code>omp_realloc</code> – Reallocate memory allocated with OpenMP routines .....	56
3.13	Environment Display Routine .....	57
3.13.1	<code>omp_display_env</code> – print the initial ICV values .....	57

## 4 OpenMP Environment Variables ..... 59

4.1	<code>OMP_ALLOCATOR</code> – Set the default allocator .....	59
4.2	<code>OMP_AFFINITY_FORMAT</code> – Set the format string used for affinity display .....	60
4.3	<code>OMP_CANCELLATION</code> – Set whether cancellation is activated .....	61
4.4	<code>OMP_DISPLAY_AFFINITY</code> – Display thread affinity information ..	61
4.5	<code>OMP_DISPLAY_ENV</code> – Show OpenMP version and environment variables .....	61
4.6	<code>OMP_DEFAULT_DEVICE</code> – Set the device used in target regions ...	61
4.7	<code>OMP_DYNAMIC</code> – Dynamic adjustment of threads .....	62
4.8	<code>OMP_MAX_ACTIVE_LEVELS</code> – Set the maximum number of nested parallel regions .....	62
4.9	<code>OMP_MAX_TASK_PRIORITY</code> – Set the maximum priority .....	62
4.10	<code>OMP_NESTED</code> – Nested parallel regions .....	63
4.11	<code>OMP_NUM_TEAMS</code> – Specifies the number of teams to use by teams region .....	63
4.12	<code>OMP_NUM_THREADS</code> – Specifies the number of threads to use ....	63
4.13	<code>OMP_PROC_BIND</code> – Whether threads may be moved between CPUs ..	64
4.14	<code>OMP_PLACES</code> – Specifies on which CPUs the threads should be placed .....	64
4.15	<code>OMP_STACKSIZE</code> – Set default thread stack size .....	65
4.16	<code>OMP_SCHEDULE</code> – How threads are scheduled .....	65
4.17	<code>OMP_TARGET_OFFLOAD</code> – Controls offloading behavior .....	66

4.18	OMP_TEAMS_THREAD_LIMIT – Set the maximum number of threads imposed by teams .....	66
4.19	OMP_THREAD_LIMIT – Set the maximum number of threads ....	67
4.20	OMP_WAIT_POLICY – How waiting threads are handled .....	67
4.21	GOMP_CPU_AFFINITY – Bind threads to specific CPUs .....	67
4.22	GOMP_DEBUG – Enable debugging output .....	68
4.23	GOMP_STACKSIZE – Set default thread stack size .....	68
4.24	GOMP_SPINCOUNT – Set the busy-wait spin count .....	68
4.25	GOMP_RTEMS_THREAD_POOLS – Set the RTEMS specific thread pools .....	68
<b>5</b>	<b>Enabling OpenACC .....</b>	<b>71</b>
<b>6</b>	<b>OpenACC Runtime Library Routines .....</b>	<b>73</b>
6.1	acc_get_num_devices – Get number of devices for given device type .....	73
6.2	acc_set_device_type – Set type of device accelerator to use ...	73
6.3	acc_get_device_type – Get type of device accelerator to be used .....	73
6.4	acc_set_device_num – Set device number to use .....	74
6.5	acc_get_device_num – Get device number to be used .....	74
6.6	acc_get_property – Get device property .....	74
6.7	acc_async_test – Test for completion of a specific asynchronous operation .....	75
6.8	acc_async_test_all – Tests for completion of all asynchronous operations .....	76
6.9	acc_wait – Wait for completion of a specific asynchronous operation .....	76
6.10	acc_wait_all – Waits for completion of all asynchronous operations .....	76
6.11	acc_wait_all_async – Wait for completion of all asynchronous operations .....	77
6.12	acc_wait_async – Wait for completion of asynchronous operations .....	77
6.13	acc_init – Initialize runtime for a specific device type .....	77
6.14	acc_shutdown – Shuts down the runtime for a specific device type .....	78
6.15	acc_on_device – Whether executing on a particular device ...	78
6.16	acc_malloc – Allocate device memory .....	78
6.17	acc_free – Free device memory .....	79
6.18	acc_copyin – Allocate device memory and copy host memory to it .....	79
6.19	acc_present_or_copyin – If the data is not present on the device, allocate device memory and copy from host memory .....	80
6.20	acc_create – Allocate device memory and map it to host memory .....	80

6.21	<code>acc_present_or_create</code> – If the data is not present on the device, allocate device memory and map it to host memory.....	81
6.22	<code>acc_copyout</code> – Copy device memory to host memory.....	82
6.23	<code>acc_delete</code> – Free device memory.....	83
6.24	<code>acc_update_device</code> – Update device memory from mapped host memory.....	84
6.25	<code>acc_update_self</code> – Update host memory from mapped device memory.....	84
6.26	<code>acc_map_data</code> – Map previously allocated device memory to host memory.....	85
6.27	<code>acc_unmap_data</code> – Unmap device memory from host memory...	85
6.28	<code>acc_deviceptr</code> – Get device pointer associated with specific host address.....	86
6.29	<code>acc_hostptr</code> – Get host pointer associated with specific device address.....	86
6.30	<code>acc_is_present</code> – Indicate whether host variable / array is present on device.....	86
6.31	<code>acc_memcpy_to_device</code> – Copy host memory to device memory...	87
6.32	<code>acc_memcpy_from_device</code> – Copy device memory to host memory.....	87
6.33	<code>acc_memcpy_device</code> – Copy memory within a device.....	88
6.34	<code>acc_attach</code> – Let device pointer point to device-pointer target...	89
6.35	<code>acc_detach</code> – Let device pointer point to host-pointer target...	89
6.36	<code>acc_get_current_cuda_device</code> – Get CUDA device handle...	90
6.37	<code>acc_get_current_cuda_context</code> – Get CUDA context handle...	90
6.38	<code>acc_get_cuda_stream</code> – Get CUDA stream handle.....	90
6.39	<code>acc_set_cuda_stream</code> – Set CUDA stream handle.....	90
6.40	<code>acc_prof_register</code> – Register callbacks.....	91
6.41	<code>acc_prof_unregister</code> – Unregister callbacks.....	91
6.42	<code>acc_prof_lookup</code> – Obtain inquiry functions.....	91
6.43	<code>acc_register_library</code> – Library registration.....	91
<b>7</b>	<b>OpenACC Environment Variables .....</b>	<b>93</b>
7.1	<code>ACC_DEVICE_TYPE</code> .....	93
7.2	<code>ACC_DEVICE_NUM</code> .....	93
7.3	<code>ACC_PROFLIB</code> .....	93
<b>8</b>	<b>CUDA Streams Usage .....</b>	<b>95</b>
<b>9</b>	<b>OpenACC Library Interoperability .....</b>	<b>97</b>
9.1	Introduction.....	97
9.2	First invocation: NVIDIA CUBLAS library API.....	97
9.3	First invocation: OpenACC library API .....	98
9.4	OpenACC library and environment variables.....	99

<b>10</b>	<b>OpenACC Profiling Interface</b>	<b>101</b>
10.1	Implementation Status and Implementation-Defined Behavior	101
<b>11</b>	<b>OpenMP-Implementation Specifics</b>	<b>107</b>
11.1	Implementation-defined ICV Initialization	107
11.2	OpenMP Context Selectors	107
11.3	Memory allocation	107
<b>12</b>	<b>Offload-Target Specifics</b>	<b>111</b>
12.1	AMD Radeon (GCN)	111
12.1.1	OpenMP <code>interop</code> – Foreign-Runtime Support for AMD GPUs	112
12.2	nvptx	113
12.2.1	OpenMP <code>interop</code> – Foreign-Runtime Support for Nvidia GPUs	115
<b>13</b>	<b>The libgomp ABI</b>	<b>117</b>
13.1	Implementing MASTER construct	117
13.2	Implementing CRITICAL construct	117
13.3	Implementing ATOMIC construct	117
13.4	Implementing FLUSH construct	117
13.5	Implementing BARRIER construct	117
13.6	Implementing THREADPRIVATE construct	117
13.7	Implementing PRIVATE clause	118
13.8	Implementing FIRSTPRIVATE LASTPRIVATE COPYIN and COPYPRIVATE clauses	118
13.9	Implementing REDUCTION clause	118
13.10	Implementing PARALLEL construct	118
13.11	Implementing FOR construct	119
13.12	Implementing ORDERED construct	120
13.13	Implementing SECTIONS construct	120
13.14	Implementing SINGLE construct	120
13.15	Implementing OpenACC's PARALLEL construct	121
<b>14</b>	<b>Reporting Bugs</b>	<b>123</b>
	<b>GNU General Public License</b>	<b>125</b>
	<b>GNU Free Documentation License</b>	<b>137</b>
	ADDENDUM: How to use this License for your documents	144
	<b>Funding Free Software</b>	<b>145</b>
	<b>Library Index</b>	<b>147</b>



# 1 Enabling OpenMP

To activate the OpenMP extensions for C/C++ and Fortran, the compile-time flag `-fopenmp` must be specified. For C and C++, this enables the handling of the OpenMP directives using `#pragma omp` and the `[[omp::directive(...)]], [[omp::sequence(...)]]` and `[[omp::decl(...)]]` attributes. For Fortran, it enables for free source form the `!$omp` sentinel for directives and the `!$` conditional compilation sentinel and for fixed source form the `c$omp`, `*$omp` and `!$omp` sentinels for directives and the `c$`, `*$` and `!$` conditional compilation sentinels. The flag also arranges for automatic linking of the OpenMP runtime library (Chapter 3 [Runtime Library Routines], page 15).

The `-fopenmp-simd` flag can be used to enable a subset of OpenMP directives that do not require the linking of either the OpenMP runtime library or the POSIX threads library.

A complete description of all OpenMP directives may be found in the OpenMP Application Program Interface (<https://www.openmp.org>) manuals. See also Chapter 2 [OpenMP Implementation Status], page 3.



## 2 OpenMP Implementation Status

The `_OPENMP` preprocessor macro and Fortran's `openmp_version` parameter, provided by `omp_lib.h` and the `omp_lib` module, have the value 201511 (i.e. OpenMP 4.5).

### 2.1 OpenMP 4.5

The OpenMP 4.5 specification is fully supported.

### 2.2 OpenMP 5.0

#### New features listed in Appendix B of the OpenMP specification

Description	Status	Comments
Array shaping	N	
Array sections with non-unit strides in C and C++	N	
Iterators	Y	
<code>metadirective</code> directive	Y	
<code>declare variant</code> directive	Y	
<code>target-offload-var</code> ICV and <code>OMP_TARGET_OFFLOAD</code> env variable	Y	
Nested-parallel changes to <code>max-active-levels-var</code> ICV	Y	
<code>requires</code> directive	Y	See also Chapter 12 [Offload-Target Specifics], page 111,
<code>teams</code> construct outside an enclosing target region	Y	
Non-rectangular loop nests	P	Full support for C/C++, partial for Fortran (PR110735 ( <a href="https://gcc.gnu.org/PR110735">https://gcc.gnu.org/PR110735</a> ))
<code>!=</code> as relational-op in canonical loop form for C/C++	Y	
<code>nonmonotonic</code> as default loop schedule modifier for worksharing-loop constructs	Y	
Collapse of associated loops that are imperfectly nested loops	Y	
Clauses <code>if</code> , <code>nontemporal</code> and <code>order(concurrent)</code> in <code>simd</code> construct	Y	
<code>atomic</code> constructs in <code>simd</code>	Y	
<code>loop</code> construct	Y	
<code>order(concurrent)</code> clause	Y	
<code>scan</code> directive and <code>in_scan</code> modifier for the <code>reduction</code> clause	Y	
<code>in_reduction</code> clause on <code>task</code> constructs	Y	
<code>in_reduction</code> clause on <code>target</code> constructs	P	<code>nowait</code> only stub
<code>task_reduction</code> clause with <code>taskgroup</code>	Y	
<code>task</code> modifier to <code>reduction</code> clause	Y	

<code>affinity</code> clause to <code>task</code> construct	Y	Stub only
<code>detach</code> clause to <code>task</code> construct	Y	
<code>omp_fulfill_event</code> runtime routine	Y	
<code>reduction</code> and <code>in_reduction</code> clauses on <code>taskloop</code> and <code>taskloop simd</code> constructs	Y	
<code>taskloop</code> construct cancelable by <code>cancel</code> construct	Y	
<code>mutexinoutset</code> <i>dependence-type</i> for <code>depend</code> clause	Y	
Predefined memory spaces, memory allocators, allocator traits	Y	See also Section 11.3 [Memory allocation], page 107,
Memory management routines	Y	
<code>allocate</code> directive	P	C++ unsupported; see also Section 11.3 [Memory allocation], page 107,
<code>allocate</code> clause	P	Clause has no effect on <code>target</code> (PR113436 ( <a href="https://gcc.gnu.org/PR113436">https://gcc.gnu.org/PR113436</a> ))
<code>use_device_addr</code> clause on <code>target</code> data	Y	
<code>ancestor</code> modifier on <code>device</code> clause	Y	
Implicit declare <code>target</code> directive	Y	
Discontiguous array section with <code>target update</code> construct	N	
C/C++'s lvalue expressions in <code>to</code> , <code>from</code> and <code>map</code> clauses	Y	
C/C++'s lvalue expressions in <code>depend</code> clauses	Y	
Nested <code>declare target</code> directive	Y	
Combined <code>master</code> constructs	Y	
<code>depend</code> clause on <code>taskwait</code>	Y	
Weak memory ordering clauses on <code>atomic</code> and <code>flush</code> construct	Y	
<code>hint</code> clause on the <code>atomic</code> construct	Y	Stub only
<code>depobj</code> construct and depend objects	Y	
Lock hints were renamed to synchronization hints	Y	
<code>conditional</code> modifier to <code>lastprivate</code> clause	Y	
Map-order clarifications	P	
<code>close map-type-modifier</code>	Y	
Mapping C/C++ pointer variables and to assign the address of device memory mapped by an array section	P	
Mapping of Fortran pointer and allocatable variables, including pointer and allocatable components of variables	Y	
<code>defaultmap</code> extensions	Y	

<code>declare mapper</code> directive	N
<code>omp_get_supported_active_levels</code> routine	Y
Runtime routines and environment variables to display runtime thread affinity information	Y
<code>omp_pause_resource</code> and <code>omp_pause_resource_all</code> runtime routines	Y
<code>omp_get_device_num</code> runtime routine	Y
OMPT interface	N
OMPD interface	N

## Other new OpenMP 5.0 features

Description	Status	Comments
Supporting C++'s range-based for loop	Y	

## 2.3 OpenMP 5.1

### New features listed in Appendix B of the OpenMP specification

Description	Status	Comments
OpenMP directive as C++ attribute specifiers	Y	
<code>omp_all_memory</code> reserved locator	Y	
<i>target_device_trait</i> in OpenMP Context	Y	
<code>target_device</code> selector set in context selectors	Y	
C/C++'s <code>declare variant</code> directive: elision support of preprocessed code	N	
<code>declare variant</code> : new clauses <code>adjust_args</code> and <code>append_args</code>	Y	
<code>dispatch</code> construct	Y	
device-specific ICV settings with environment variables	Y	
<code>assume</code> and <code>assumes</code> directives	Y	
<code>nothing</code> directive	Y	
<code>error</code> directive	Y	
<code>masked</code> construct	Y	
<code>scope</code> directive	Y	
Loop transformation constructs	Y	
<code>strict</code> modifier in the <code>grainsize</code> and <code>num_tasks</code> clauses of the <code>taskloop</code> construct	Y	
<code>align</code> clause in <code>allocate</code> directive	P	Only C and Fortran
<code>align</code> modifier in <code>allocate</code> clause	Y	
<code>thread_limit</code> clause to <code>target</code> construct	Y	
<code>has_device_addr</code> clause to <code>target</code> construct	Y	
Iterators in <code>target update</code> motion clauses and <code>map</code> clauses	N	
Indirect calls to the device version of a procedure or function in <code>target</code> regions	Y	

<code>interop</code> directive	Y	Cf. Chapter 12 [Offload-Target Specifics], page 111,
<code>omp_interop_t</code> object support in runtime routines	Y	
<code>nowait</code> clause in <code>taskwait</code> directive	Y	
Extensions to the <code>atomic</code> directive	Y	
<code>seq_cst</code> clause on a <code>flush</code> construct	Y	
<code>inoutset</code> argument to the <code>depend</code> clause	Y	
<code>private</code> and <code>firstprivate</code> argument to <code>default</code> clause in C and C++	Y	
<code>present</code> argument to <code>defaultmap</code> clause	Y	
<code>omp_set_num_teams</code> , <code>omp_set_teams_thread_limit</code> , <code>omp_get_max_teams</code> , <code>omp_get_teams_thread_limit</code> runtime routines	Y	
<code>omp_target_is_accessible</code> runtime routine	Y	
<code>omp_target_memcpy_async</code> and <code>omp_target_memcpy_rect_async</code> runtime routines	Y	
<code>omp_get_mapped_ptr</code> runtime routine	Y	
<code>omp_calloc</code> , <code>omp_realloc</code> , <code>omp_aligned_alloc</code> and <code>omp_aligned_calloc</code> runtime routines	Y	
<code>omp_alloctrail_key_t</code> enum: <code>omp_atv_serialized</code> added, <code>omp_atv_default</code> changed	Y	
<code>omp_display_env</code> runtime routine	Y	
<code>ompt_scope_endpoint_t</code> enum: <code>ompt_scope_beginend</code>	N	
<code>ompt_sync_region_t</code> enum additions	N	
<code>ompt_state_t</code> enum: <code>ompt_state_wait_barrier_implementation</code> and <code>ompt_state_wait_barrier_teams</code>	N	
<code>ompt_callback_target_data_op_emi_t</code> , <code>ompt_callback_target_emi_t</code> , <code>ompt_callback_target_map_emi_t</code> and <code>ompt_callback_target_submit_emi_t</code>	N	
<code>ompt_callback_error_t</code> type	N	
<code>OMP_PLACES</code> syntax extensions	Y	
<code>OMP_NUM_TEAMS</code> and <code>OMP_TEAMS_THREAD_LIMIT</code> environment variables	Y	

## Other new OpenMP 5.1 features

Description	Status	Comments
Support of strictly structured blocks in Fortran	Y	
Support of structured block sequences in C/C++	Y	
<code>unconstrained</code> and <code>reproducible</code> modifiers on <code>order</code> clause	Y	
Support <code>begin/end declare target</code> syntax in C/C++	Y	

Pointer predetermined firstprivate getting initialized to address of matching mapped list item per 5.1, Sect. 2.21.7.2	N	
For Fortran, diagnose placing declarative before/between <code>USE</code> , <code>IMPORT</code> , and <code>IMPLICIT</code> as invalid	N	
Optional comma between directive and clause in the <code>#pragma</code> form	Y	
<code>indirect</code> clause in <code>declare target</code>	Y	
<code>device_type(nohost)/device_type(host)</code> for variables	N	
<code>present</code> modifier to the <code>map</code> , <code>to</code> and <code>from</code> clauses	Y	
Changed interaction between <code>declare target</code> and OpenMP context	Y	
Dynamic selector support in <code>metadirective</code>	Y	
Dynamic selector support in <code>declare variant</code>	P	Fortran rejects non-constant expressions in dynamic selectors; C/C++ reject expressions using argument variables. (PR113904 ( <a href="https://gcc.gnu.org/PR113904">https://gcc.gnu.org/PR113904</a> ))

## 2.4 OpenMP 5.2

### New features listed in Appendix B of the OpenMP specification

Description	Status	Comments
<code>omp_in_explicit_task</code> routine and <i>explicit-task-var</i> ICV	Y	
<code>omp/ompx/omx</code> sentinels and <code>omp_/ompx_</code> namespaces	N/A	warning for <code>ompx/omx</code> sentinels <sup>1</sup>
Clauses on <code>end</code> directive can be on directive	Y	
<code>destroy</code> clause with <code>destroy-var</code> argument on <code>depobj</code>	Y	
Deprecation of no-argument <code>destroy</code> clause on <code>depobj</code>	N/A	undeprecated in OpenMP 6
<code>linear</code> clause syntax changes and <code>step</code> modifier	Y	
Deprecation of minus operator for reductions	N	
Deprecation of separating <code>map</code> modifiers without comma	N	

<sup>1</sup> The `ompx` sentinel as C/C++ pragma and C++ attributes are warned for with `-Wunknown-pragmas` (implied by `-Wall`) and `-Wattributes` (enabled by default), respectively; for Fortran free-source code, there is a warning enabled by default and, for fixed-source code, the `omx` sentinel is warned for with `-Wsurprising` (enabled by `-Wall`). Unknown clauses are always rejected with an error.

<code>declare mapper</code> with iterator and <code>present</code> modifiers	N	
If a matching mapped list item is not found in the data environment, the pointer retains its original value	Y	
New <code>enter</code> clause as alias for <code>to</code> on declare target directive	Y	
Deprecation of <code>to</code> clause on declare target directive	N	
Extended list of directives permitted in Fortran pure procedures	Y	
New <code>allocators</code> directive for Fortran	Y	
Deprecation of <code>allocate</code> directive for Fortran allocatables/pointers	N	
Optional paired <code>end</code> directive with <code>dispatch</code>	Y	
New <code>memspace</code> and <code>traits</code> modifiers for <code>uses_allocators</code>	N	
Deprecation of <code>traits</code> array following the <code>allocator_handle</code> expression in <code>uses_allocators</code>	N	
New <code>otherwise</code> clause as alias for <code>default</code> on metadirectives	Y	
Deprecation of <code>default</code> clause on metadirectives	N	Both <code>otherwise</code> and <code>default</code> are accepted without diagnostics.
Deprecation of delimited form of <code>declare target</code>	N	
Reproducible semantics changed for <code>order(concurrent)</code>	N	
<code>allocate</code> and <code>firstprivate</code> clauses on <code>scope</code>	Y	
<code>ompt_callback_work</code>	N	
Default map-type for the <code>map</code> clause in <code>target enter/exit data</code>	Y	
New <code>doacross</code> clause as alias for <code>depend</code> with <code>source/sink</code> modifier	Y	
Deprecation of <code>depend</code> with <code>source/sink</code> modifier	N	
<code>omp_cur_iteration</code> keyword	Y	

## Other new OpenMP 5.2 features

Description	Status	Comments
For Fortran, optional comma between directive and clause	N	
Conforming device numbers and <code>omp_initial_device</code> and <code>omp_invalid_device</code> enum/PARAMETER	Y	
Initial value of <i>default-device-var</i> ICV with <code>OMP_TARGET_OFFLOAD=mandatory</code>	Y	
<code>all</code> as <i>implicit-behavior</i> for <code>defaultmap</code>	Y	
<i>interop-types</i> in any position of the modifier list for the <code>init</code> clause of the <code>interop</code> construct	Y	

Invoke virtual member functions of C++ objects created on the host device on other devices	N
<code>mapper</code> as map-type modifier in <code>declare mapper</code>	N

## 2.5 OpenMP 6.0

### New features listed in Appendix B of the OpenMP specification

Features deprecated in versions 5.0, 5.1 and 5.2 were removed	N/A	Backward compatibility
Full support for C23 was added	P	
Full support for C++23 was added	P	
Full support for Fortran 2023 was added	P	
<code>_ALL</code> suffix to the device-scope environment variables	P	Host device number wrongly accepted
<code>num_threads</code> clause now accepts a list	N	
Abstract names added for <code>OMP_NUM_THREADS</code> , <code>OMP_THREAD_LIMIT</code> and <code>OMP_TEAMS_THREAD_LIMIT</code>	N	
Supporting increments with abstract names in <code>OMP_PLACES</code>	N	
Extension of <code>OMP_DEFAULT_DEVICE</code> and new <code>OMP_AVAILABLE_DEVICES</code> environment vars	N	
New <code>uid</code> trait for target devices and for <code>OMP_AVAILABLE_DEVICES</code> and <code>OMP_DEFAULT_DEVICE</code>	N	
New <code>OMP_THREADS_RESERVE</code> environment variable	N	
The <code>decl</code> attribute was added to the C++ attribute syntax	Y	
The OpenMP directive syntax was extended to include C23 attribute specifiers	Y	
Support for pure directives in Fortran's <code>do concurrent</code>	N	
All inarguable clauses take now an optional Boolean argument	N	
The <code>adjust_args</code> clause was extended to specify the argument by position and supports variadic arguments	N	
For Fortran, <i>locator list</i> can be also function reference with data pointer result	N	
Concept of <i>assumed-size arrays</i> in C and C++	N	
<i>directive-name-modifier</i> accepted in all clauses	N	
Extension of <code>interop</code> operation of <code>append_args</code> , allowing all modifiers of the <code>init</code> clause	Y	
New argument-free version of <code>depobj</code> with repeatable clauses and the <code>init</code> clause	N	
Undeprecate omitting the argument to the <code>depend</code> clause of the argument version of the <code>depend</code> construct	Y	
For Fortran, atomic with <code>BLOCK</code> construct and, for C/C++, with unlimited curly braces supported	N	

For Fortran, atomic with pointer comparison	N		
For Fortran, atomic with enum and enumeration types	N		
For Fortran, atomic compare with storing the comparison result	N		
Canonical loop sequences and new <code>looprange</code> clause	N		
For Fortran, handling polymorphic types in data-sharing-attribute clauses	P	<code>private</code>	not supported
For Fortran, rejecting polymorphic types in data-mapping clauses	N	not	diagnosed (and mostly unsupported)
New <code>taskgraph</code> construct including <code>saved</code> modifier and <code>replayable</code> clause	N		
<code>default</code> clause on the <code>target</code> directive and accepting variable categories	N		
Semantic change regarding the reference count update with <code>use_device_ptr</code> and <code>use_device_addr</code>	N		
Support for inductions	N		
Reduction over private variables with <code>reduction</code> clause	N		
Implicit reduction identifiers of C++ classes	N		
New <code>init_complete</code> clause to the <code>scan</code> directive	N		
<code>ref</code> modifier to the <code>map</code> clause	N		
New <code>storage</code> map-type modifier; context-dependent <code>alloc</code> and <code>release</code> are aliases	N		
Change of the <i>map-type</i> property from <i>ultimate</i> to <i>default</i>	N		
<code>self</code> modifier to <code>map</code> and <code>self</code> as <code>defaultmap</code> argument	N		
Mapping of <i>assumed-size arrays</i> in C, C++ and Fortran	N		
<code>delete</code> as delete-modifier not as map type	N		
For Fortran, the <code>automap</code> modifier to the <code>enter</code> clause of <code>declare_target</code>	N		
<code>groupprivate</code> directive	N		
<code>local</code> clause to <code>declare_target</code> directive	N		
<code>part_size</code> allocator trait for <code>interleaved</code> allocator partitions	N		
<code>pin_device</code> , <code>preferred_device</code> and <code>target_access</code> allocator traits	N		
<code>access</code> allocator trait changes	N		
New <code>partitioner</code> value to <code>partition</code> allocator trait	N		
Semicolon-separated list to <code>uses_allocators</code>	N		
New <code>need_device_addr</code> modifier to <code>adjust_args</code> clause	N		
<code>interop</code> clause to <code>dispatch</code>	Y		
Scope requirement changes for <code>declare_target</code>	N		
<code>message</code> and <code>severity</code> clauses to <code>parallel</code> directive	N		

<code>self_maps</code> clause to <code>requires</code> directive	Y
<code>no_openmp_constructs</code> assumptions clause	N
Restriction for <code>ordered</code> regarding loop-transforming directives	N
<code>apply</code> clause to loop-transforming constructs	N
Non-constant values in the <code>sizes</code> clause	N
<code>fuse</code> loop-transformation construct	N
<code>interchange</code> loop-transformation construct	N
<code>reverse</code> loop-transformation construct	N
<code>split</code> loop-transformation construct	N
<code>stripe</code> loop-transformation construct	N
<code>tile</code> permitting association of grid and inter-tile loops	N
<code>strict</code> modifier keyword to <code>num_threads</code>	N
<code>safesync</code> clause to the <code>parallel</code> construct	N
<code>omp_curr_progress_width</code> identifier	N
<code>omp_get_max_progress_width</code> runtime routine	N
Lifted restrictions on <code>order(concurrent)</code> and, hence, the <code>loop</code> construct	N
<code>atomic</code> permitted in a construct with <code>order(concurrent)</code>	N
Lifted restrictions on not-strictly-nested regions with <code>order(concurrent)</code>	N
<code>workdistribute</code> directive for Fortran	N
Fortran DO CONCURRENT as associated loop in a <code>loop</code> construct	N
New <code>task_iteration</code> directive inside <code>taskloop</code>	N
<code>threadset</code> clause in task-generating constructs	N
New <code>priority</code> clause to <code>target</code> , <code>target_enter_data</code> , <code>target_data</code> , <code>target_exit_data</code> and <code>target_update</code>	N
New <code>device_type</code> clause to the <code>target</code> directive	N
<code>target_data</code> as composite construct	N
<code>nowait</code> clause with reverse-offload <code>target</code> directives	N
Extended <i>prefer-type</i> modifier to <code>init</code> clause	Y
Boolean argument to <code>nowait</code> and <code>nogroup</code> may be non constant	N
<code>memscope</code> clause to <code>atomic</code> and <code>flush</code>	N
New <code>transparent</code> clause for multi-generational task-dependence graphs	N
The <code>cancel</code> construct now completes tasks with unfulfilled events	N
<code>omp_fulfill_event</code> routine was restricted regarding fulfillment of event variables	N
Added rule for compound-directive names, permitting many more combinations	N
<code>omp_is_free_agent</code> and <code>omp_ancestor_is_free_agent</code> routines	N

omp_get_device_from_uid and omp_get_uid_from_device routines	Y
omp_get_device_num_teams, omp_set_device_num_teams, omp_get_device_teams_thread_limit, and omp_set_device_teams_thread_limit routines	N
omp_target_memset and omp_target_memset_async routines	Y
Fortran version of the interop runtime routines	Y
Routines for obtaining memory spaces/allocators for shared/device memory	N
omp_get_memspace_num_resources routine	N
omp_get_memspace_pagesize routine	N
omp_get_submemspace routine	N
omp_init_mempartitioner, omp_destroy_mempartitioner, omp_init_mempartition, omp_destroy_mempartition, omp_mempartition_set_part, omp_mempartition_get_user_data routines	N
Deprecation of the target_data_op, target, target_map and target_submit callbacks and as values that set_callback must return	N
ompt_target_data_transfer and ompt_target_data_transfer_async values in ompt_target_data_op_t enum	N
The values ompt_target_data_transfer_to_device, ompt_target_data_transfer_from_device, ompt_target_data_transfer_to_device_async and ompt_target_data_transfer_from_device_async of the target_data_op OMPT type were deprecated	N
ompt_get_buffer_limits OMPT routine	N

### Deprecated features, unless listed above

Deprecation of omitting the optional white space to separate adjacent keywords in the directive-name in Fortran (fixed and free source form)	N
Deprecation of the combiner expression in the declare_reduction argument	N
Deprecation of the Fortran include file omp_lib.h	N

### Other new OpenMP 6.0 features

Multi-word directives now use underscore by default	N
Relaxed Fortran restrictions to the aligned clause	N
Mapping lambda captures	N
New omp_pause_stop_tool constant for omp_pause_resource	N

In Fortran (fixed and free source form), spaces between N  
directive names are mandatory  
Update of the map-type decay for mapping and N  
`declare_mapper`



## 3 OpenMP Runtime Library Routines

The runtime routines described here are defined by Section 18 of the OpenMP specification in version 5.2.

### 3.1 Thread Team Routines

Routines controlling threads in the current contention group. They have C linkage and do not throw exceptions.

#### 3.1.1 `omp_set_num_threads` – Set upper team size limit

*Description:*

Specifies the number of threads used by default in subsequent parallel sections, if those do not specify a `num_threads` clause. The argument of `omp_set_num_threads` shall be a positive integer.

*C/C++:*

*Prototype:*            `void omp_set_num_threads(int num_threads);`

*Fortran:*

*Interface:*            `subroutine omp_set_num_threads(num_threads)  
integer, intent(in) :: num_threads`

*See also:* Section 4.12 [OMP\_NUM\_THREADS], page 63, Section 3.1.2 [omp\_get\_num\_threads], page 15, Section 3.1.3 [omp\_get\_max\_threads], page 16,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.1.

#### 3.1.2 `omp_get_num_threads` – Size of the active team

*Description:*

Returns the number of threads in the current team. In a sequential section of the program `omp_get_num_threads` returns 1.

The default team size may be initialized at startup by the `OMP_NUM_THREADS` environment variable. At runtime, the size of the current team may be set either by the `NUM_THREADS` clause or by `omp_set_num_threads`. If none of the above were used to define a specific value and `OMP_DYNAMIC` is disabled, one thread per CPU online is used.

*C/C++:*

*Prototype:*            `int omp_get_num_threads(void);`

*Fortran:*

*Interface:*            `integer function omp_get_num_threads()`

*See also:* Section 3.1.3 [omp\_get\_max\_threads], page 16, Section 3.1.1 [omp\_set\_num\_threads], page 15, Section 4.12 [OMP\_NUM\_THREADS], page 63,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.2.

### 3.1.3 `omp_get_max_threads` – Maximum number of threads of parallel region

*Description:*

Return the maximum number of threads used for the current parallel region that does not use the clause `num_threads`.

*C/C++:*

*Prototype:*        `int omp_get_max_threads(void);`

*Fortran:*

*Interface:*        `integer function omp_get_max_threads()`

*See also:*    Section 3.1.1 [`omp_set_num_threads`], page 15, Section 3.1.6 [`omp_set_dynamic`], page 17, Section 3.3.6 [`omp_get_thread_limit`], page 24,

*Reference:*    OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.3.

### 3.1.4 `omp_get_thread_num` – Current thread ID

*Description:*

Returns a unique thread identification number within the current team. In a sequential parts of the program, `omp_get_thread_num` always returns 0. In parallel regions the return value varies from 0 to `omp_get_num_threads-1` inclusive. The return value of the primary thread of a team is always 0.

*C/C++:*

*Prototype:*        `int omp_get_thread_num(void);`

*Fortran:*

*Interface:*        `integer function omp_get_thread_num()`

*See also:*    Section 3.1.2 [`omp_get_num_threads`], page 15, Section 3.1.18 [`omp_get_ancestor_thread_num`], page 21,

*Reference:*    OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.4.

### 3.1.5 `omp_in_parallel` – Whether a parallel region is active

*Description:*

This function returns `true` if currently running in parallel, `false` otherwise. Here, `true` and `false` represent their language-specific counterparts.

*C/C++:*

*Prototype:*        `int omp_in_parallel(void);`

*Fortran:*

*Interface:*        `logical function omp_in_parallel()`

*Reference:*    OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.6.

### 3.1.6 `omp_set_dynamic` – Enable/disable dynamic teams

*Description:*

Enable or disable the dynamic adjustment of the number of threads within a team. The function takes the language-specific equivalent of **true** and **false**, where **true** enables dynamic adjustment of team sizes and **false** disables it.

*C/C++:*

*Prototype:*            `void omp_set_dynamic(int dynamic_threads);`

*Fortran:*

*Interface:*           `subroutine omp_set_dynamic(dynamic_threads)`  
                          `logical, intent(in) :: dynamic_threads`

*See also:*    Section 4.7 [OMP\_DYNAMIC], page 62, Section 3.1.7 [omp\_get\_dynamic], page 17,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.7.

### 3.1.7 `omp_get_dynamic` – Dynamic teams setting

*Description:*

This function returns **true** if enabled, **false** otherwise. Here, **true** and **false** represent their language-specific counterparts.

The dynamic team setting may be initialized at startup by the `OMP_DYNAMIC` environment variable or at runtime using `omp_set_dynamic`. If undefined, dynamic adjustment is disabled by default.

*C/C++:*

*Prototype:*           `int omp_get_dynamic(void);`

*Fortran:*

*Interface:*           `logical function omp_get_dynamic()`

*See also:*    Section 3.1.6 [omp\_set\_dynamic], page 17, Section 4.7 [OMP\_DYNAMIC], page 62,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.8.

### 3.1.8 `omp_get_cancellation` – Whether cancellation support is enabled

*Description:*

This function returns **true** if cancellation is activated, **false** otherwise. Here, **true** and **false** represent their language-specific counterparts. Unless `OMP_CANCELLATION` is set true, cancellations are deactivated.

*C/C++:*

*Prototype:*           `int omp_get_cancellation(void);`

*Fortran:*

*Interface:*           `logical function omp_get_cancellation()`

*See also:*    Section 4.3 [OMP\_CANCELLATION], page 61,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.9.

### 3.1.9 `omp_set_nested` – Enable/disable nested parallel regions

*Description:*

Enable or disable nested parallel regions, i.e., whether team members are allowed to create new teams. The function takes the language-specific equivalent of `true` and `false`, where `true` enables dynamic adjustment of team sizes and `false` disables it.

Enabling nested parallel regions also sets the maximum number of active nested regions to the maximum supported. Disabling nested parallel regions sets the maximum number of active nested regions to one.

Note that the `omp_set_nested` API routine was deprecated in the OpenMP specification 5.0 in favor of `omp_set_max_active_levels`.

*C/C++:*

*Prototype:*        `void omp_set_nested(int nested);`

*Fortran:*

*Interface:*        `subroutine omp_set_nested(nested)`  
                      `logical, intent(in) :: nested`

*See also:*    Section 3.1.10 [`omp_get_nested`], page 18, Section 3.1.15 [`omp_set_max_active_levels`], page 20, Section 4.8 [`OMP_MAX_ACTIVE_LEVELS`], page 62, Section 4.10 [`OMP_NESTED`], page 63,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.10.

### 3.1.10 `omp_get_nested` – Nested parallel regions

*Description:*

This function returns `true` if nested parallel regions are enabled, `false` otherwise. Here, `true` and `false` represent their language-specific counterparts.

The state of nested parallel regions at startup depends on several environment variables. If `OMP_MAX_ACTIVE_LEVELS` is defined and is set to greater than one, then nested parallel regions will be enabled. If not defined, then the value of the `OMP_NESTED` environment variable will be followed if defined. If neither are defined, then if either `OMP_NUM_THREADS` or `OMP_PROC_BIND` are defined with a list of more than one value, then nested parallel regions are enabled. If none of these are defined, then nested parallel regions are disabled by default.

Nested parallel regions can be enabled or disabled at runtime using `omp_set_nested`, or by setting the maximum number of nested regions with `omp_set_max_active_levels` to one to disable, or above one to enable.

Note that the `omp_get_nested` API routine was deprecated in the OpenMP specification 5.0 in favor of `omp_get_max_active_levels`.

*C/C++:*

*Prototype:*        `int omp_get_nested(void);`

*Fortran:*

*Interface:*        `logical function omp_get_nested()`

*See also:* Section 3.1.16 [omp-get-max-active-levels], page 21, Section 3.1.9 [omp-set-nested], page 18, Section 4.8 [OMP\_MAX\_ACTIVE\_LEVELS], page 62, Section 4.10 [OMP\_NESTED], page 63,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.11.

### 3.1.11 omp\_set\_schedule – Set the runtime scheduling method

*Description:*

Sets the runtime scheduling method. The *kind* argument can have the value `omp_sched_static`, `omp_sched_dynamic`, `omp_sched_guided` or `omp_sched_auto`. Except for `omp_sched_auto`, the chunk size is set to the value of *chunk\_size* if positive, or to the default value if zero or negative. For `omp_sched_auto` the *chunk\_size* argument is ignored.

*C/C++*

*Prototype:*        `void omp_set_schedule(omp_sched_t kind, int chunk_size);`

*Fortran:*

*Interface:*        `subroutine omp_set_schedule(kind, chunk_size)`  
                      `integer(kind=omp_sched_kind) kind`  
                      `integer chunk_size`

*See also:* Section 3.1.12 [omp-get-schedule], page 19, Section 4.16 [OMP\_SCHEDULE], page 65,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.12.

### 3.1.12 omp\_get\_schedule – Obtain the runtime scheduling method

*Description:*

Obtain the runtime scheduling method. The *kind* argument is set to `omp_sched_static`, `omp_sched_dynamic`, `omp_sched_guided` or `omp_sched_auto`. The second argument, *chunk\_size*, is set to the chunk size.

*C/C++*

*Prototype:*        `void omp_get_schedule(omp_sched_t *kind, int *chunk_size);`

*Fortran:*

*Interface:*        `subroutine omp_get_schedule(kind, chunk_size)`  
                      `integer(kind=omp_sched_kind) kind`  
                      `integer chunk_size`

*See also:* Section 3.1.11 [omp-set-schedule], page 19, Section 4.16 [OMP\_SCHEDULE], page 65,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.13.

### 3.1.13 `omp_get_teams_thread_limit` – Maximum number of threads imposed by teams

*Description:*

Return the maximum number of threads that are able to participate in each team created by a teams construct.

*C/C++:*

*Prototype:*        `int omp_get_teams_thread_limit(void);`

*Fortran:*

*Interface:*        `integer function omp_get_teams_thread_limit()`

*See also:*    Section 3.3.5 [`omp_set_teams_thread_limit`], page 24, Section 4.18 [`OMP_TEAMS_THREAD_LIMIT`], page 66,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.4.6.

### 3.1.14 `omp_get_supported_active_levels` – Maximum number of active regions supported

*Description:*

This function returns the maximum number of nested, active parallel regions supported by this implementation.

*C/C++:*

*Prototype:*        `int omp_get_supported_active_levels(void);`

*Fortran:*

*Interface:*        `integer function omp_get_supported_active_levels()`

*See also:*    Section 3.1.16 [`omp_get_max_active_levels`], page 21, Section 3.1.15 [`omp_set_max_active_levels`], page 20,

*Reference:* OpenMP specification v5.0 (<https://www.openmp.org>), Section 3.2.15.

### 3.1.15 `omp_set_max_active_levels` – Limits the number of active parallel regions

*Description:*

This function limits the maximum allowed number of nested, active parallel regions. *max\_levels* must be less or equal to the value returned by `omp_get_supported_active_levels`.

*C/C++:*

*Prototype:*        `void omp_set_max_active_levels(int max_levels);`

*Fortran:*

*Interface:*        `subroutine omp_set_max_active_levels(max_levels)  
                      integer max_levels`

*See also:*    Section 3.1.16 [`omp_get_max_active_levels`], page 21, Section 3.1.20 [`omp_get_active_level`], page 22, Section 3.1.14 [`omp_get_supported_active_levels`], page 20,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.15.

### 3.1.16 `omp_get_max_active_levels` – Current maximum number of active regions

*Description:*

This function obtains the maximum allowed number of nested, active parallel regions.

*C/C++*

*Prototype:*            `int omp_get_max_active_levels(void);`

*Fortran:*

*Interface:*            `integer function omp_get_max_active_levels()`

*See also:*    Section 3.1.15 [`omp_set_max_active_levels`], page 20, Section 3.1.20 [`omp_get_active_level`], page 22,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.16.

### 3.1.17 `omp_get_level` – Obtain the current nesting level

*Description:*

This function returns the nesting level for the parallel blocks, which enclose the calling call.

*C/C++*

*Prototype:*            `int omp_get_level(void);`

*Fortran:*

*Interface:*            `integer function omp_level()`

*See also:*    Section 3.1.20 [`omp_get_active_level`], page 22,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.17.

### 3.1.18 `omp_get_ancestor_thread_num` – Ancestor thread ID

*Description:*

This function returns the thread identification number for the given nesting level of the current thread. For values of *level* outside zero to `omp_get_level` -1 is returned; if *level* is `omp_get_level` the result is identical to `omp_get_thread_num`.

*C/C++*

*Prototype:*            `int omp_get_ancestor_thread_num(int level);`

*Fortran:*

*Interface:*            `integer function omp_get_ancestor_thread_num(level)`  
                          `integer level`

*See also:*    Section 3.1.17 [`omp_get_level`], page 21, Section 3.1.4 [`omp_get_thread_num`], page 16, Section 3.1.19 [`omp_get_team_size`], page 22,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.18.

### 3.1.19 `omp_get_team_size` – Number of threads in a team

*Description:*

This function returns the number of threads in a thread team to which either the current thread or its ancestor belongs. For values of *level* outside zero to `omp_get_level`, -1 is returned; if *level* is zero, 1 is returned, and for `omp_get_level`, the result is identical to `omp_get_num_threads`.

*C/C++:*

*Prototype:*            `int omp_get_team_size(int level);`

*Fortran:*

*Interface:*           `integer function omp_get_team_size(level)`  
                         `integer level`

*See also:*    Section 3.1.2 [`omp_get_num_threads`], page 15, Section 3.1.17 [`omp_get_level`], page 21, Section 3.1.18 [`omp_get_ancestor_thread_num`], page 21,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.19.

### 3.1.20 `omp_get_active_level` – Number of parallel regions

*Description:*

This function returns the nesting level for the active parallel blocks, which enclose the calling call.

*C/C++:*

*Prototype:*           `int omp_get_active_level(void);`

*Fortran:*

*Interface:*           `integer function omp_get_active_level()`

*See also:*    Section 3.1.17 [`omp_get_level`], page 21, Section 3.1.16 [`omp_get_max_active_levels`], page 21, Section 3.1.15 [`omp_set_max_active_levels`], page 20,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.20.

## 3.2 Thread Affinity Routines

Routines controlling and accessing thread-affinity policies. They have C linkage and do not throw exceptions.

### 3.2.1 `omp_get_proc_bind` – Whether threads may be moved between CPUs

*Description:*

This functions returns the currently active thread affinity policy, which is set via `OMP_PROC_BIND`. Possible values are `omp_proc_bind_false`, `omp_proc_bind_true`, `omp_proc_bind_primary`, `omp_proc_bind_master`, `omp_proc_bind_close` and `omp_proc_bind_spread`, where `omp_proc_bind_master` is an alias for `omp_proc_bind_primary`.

*C/C++:*

*Prototype:*           `omp_proc_bind_t omp_get_proc_bind(void);`

*Fortran:*

*Interface:*            `integer(kind=omp_proc_bind_kind) function  
                         omp_get_proc_bind()`

*See also:*    Section 4.13 [OMP\_PROC\_BIND], page 64, Section 4.14 [OMP\_PLACES],  
                 page 64, Section 4.21 [GOMP\_CPU\_AFFINITY], page 67,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.22.

### 3.3 Teams Region Routines

Routines controlling the league of teams that are executed in a `teams` region. They have C linkage and do not throw exceptions.

#### 3.3.1 `omp_get_num_teams` – Number of teams

*Description:*

Returns the number of teams in the current team region.

*C/C++:*

*Prototype:*            `int omp_get_num_teams(void);`

*Fortran:*

*Interface:*            `integer function omp_get_num_teams()`

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.32.

#### 3.3.2 `omp_get_team_num` – Get team number

*Description:*

Returns the team number of the calling thread.

*C/C++:*

*Prototype:*            `int omp_get_team_num(void);`

*Fortran:*

*Interface:*            `integer function omp_get_team_num()`

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.33.

#### 3.3.3 `omp_set_num_teams` – Set upper teams limit for teams construct

*Description:*

Specifies the upper bound for number of teams created by the teams construct which does not specify a `num_teams` clause. The argument of `omp_set_num_teams` shall be a positive integer.

*C/C++:*

*Prototype:*            `void omp_set_num_teams(int num_teams);`

*Fortran:*

*Interface:*            `subroutine omp_set_num_teams(num_teams)  
                         integer, intent(in) :: num_teams`

*See also:* Section 4.11 [OMP\_NUM\_TEAMS], page 63, Section 3.3.1 [omp\_get\_num\_teams], page 23, Section 3.3.4 [omp\_get\_max\_teams], page 24,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.4.3.

### 3.3.4 omp\_get\_max\_teams – Maximum number of teams of teams region

*Description:*

Return the maximum number of teams used for the teams region that does not use the clause `num_teams`.

*C/C++:*

*Prototype:*        `int omp_get_max_teams(void);`

*Fortran:*

*Interface:*        `integer function omp_get_max_teams()`

*See also:* Section 3.3.3 [omp\_set\_num\_teams], page 23, Section 3.3.1 [omp\_get\_num\_teams], page 23,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.4.4.

### 3.3.5 omp\_set\_teams\_thread\_limit – Set upper thread limit for teams construct

*Description:*

Specifies the upper bound for number of threads that are available for each team created by the teams construct which does not specify a `thread_limit` clause. The argument of `omp_set_teams_thread_limit` shall be a positive integer.

*C/C++:*

*Prototype:*        `void omp_set_teams_thread_limit(int thread_limit);`

*Fortran:*

*Interface:*        `subroutine omp_set_teams_thread_limit(thread_limit)`  
                       `integer, intent(in) :: thread_limit`

*See also:* Section 4.18 [OMP\_TEAMS\_THREAD\_LIMIT], page 66, Section 3.1.13 [omp\_get\_teams\_thread\_limit], page 20, Section 3.3.6 [omp\_get\_thread\_limit], page 24,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.4.5.

### 3.3.6 omp\_get\_thread\_limit – Maximum number of threads

*Description:*

Return the maximum number of threads of the program.

*C/C++:*

*Prototype:*        `int omp_get_thread_limit(void);`

*Fortran:*

*Interface:*            `integer function omp_get_thread_limit()`

*See also:* Section 3.1.3 [omp\_get\_max\_threads], page 16, Section 4.19 [OMP\_THREAD\_LIMIT], page 67,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.14.

## 3.4 Tasking Routines

Routines relating to explicit tasks. They have C linkage and do not throw exceptions.

### 3.4.1 omp\_get\_max\_task\_priority – Maximum priority value

that can be set for tasks.

*Description:*

This function obtains the maximum allowed priority number for tasks.

*C/C++*

*Prototype:*            `int omp_get_max_task_priority(void);`

*Fortran:*

*Interface:*            `integer function omp_get_max_task_priority()`

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.29.

### 3.4.2 omp\_in\_explicit\_task – Whether a given task is an explicit task

*Description:*

The function returns the *explicit-task-var* ICV; it returns true when the encountering task was generated by a task-generating construct such as `target`, `task` or `taskloop`. Otherwise, the encountering task is in an implicit task region such as generated by the implicit or explicit `parallel` region and `omp_in_explicit_task` returns false.

*C/C++*

*Prototype:*            `int omp_in_explicit_task(void);`

*Fortran:*

*Interface:*            `logical function omp_in_explicit_task()`

*Reference:* OpenMP specification v5.2 (<https://www.openmp.org>), Section 18.5.2.

### 3.4.3 omp\_in\_final – Whether in final or included task region

*Description:*

This function returns `true` if currently running in a final or included task region, `false` otherwise. Here, `true` and `false` represent their language-specific counterparts.

*C/C++:*

*Prototype:*            `int omp_in_final(void);`

*Fortran:*

*Interface:*           logical function omp\_in\_final()

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.21.

## 3.5 Resource Relinquishing Routines

Routines releasing resources used by the OpenMP runtime. They have C linkage and do not throw exceptions.

### 3.5.1 omp\_pause\_resource – Release OpenMP resources on a device

*Description:*

Free resources used by the OpenMP program and the runtime library on and for the device specified by *device\_num*; on success, zero is returned and non-zero otherwise.

The value of *device\_num* must be a conforming device number. The routine may not be called from within any explicit region and all explicit threads that do not bind to the implicit parallel region have finalized execution.

*C/C++:*

*Prototype:*           int omp\_pause\_resource(omp\_pause\_resource\_t kind,  
                          int device\_num);

*Fortran:*

*Interface:*           integer function omp\_pause\_resource(kind,  
                          device\_num)  
                          integer (kind=omp\_pause\_resource\_kind) kind  
                          integer device\_num

*Reference:* OpenMP specification v5.0 (<https://www.openmp.org>), Section 3.2.43.

### 3.5.2 omp\_pause\_resource\_all – Release OpenMP resources on all devices

*Description:*

Free resources used by the OpenMP program and the runtime library on all devices, including the host. On success, zero is returned and non-zero otherwise.

The routine may not be called from within any explicit region and all explicit threads that do not bind to the implicit parallel region have finalized execution.

*C/C++:*

*Prototype:*           int omp\_pause\_resource(omp\_pause\_resource\_t kind);

*Fortran:*

*Interface:*           integer function omp\_pause\_resource(kind)  
                          integer (kind=omp\_pause\_resource\_kind) kind

*See also:*   Section 3.5.1 [omp\_pause\_resource], page 26,

*Reference:* OpenMP specification v5.0 (<https://www.openmp.org>), Section 3.2.44.

## 3.6 Device Information Routines

Routines related to devices available to an OpenMP program. They have C linkage and do not throw exceptions.

### 3.6.1 `omp_get_num_procs` – Number of processors online

*Description:*

Returns the number of processors online on that device.

*C/C++:*

*Prototype:*            `int omp_get_num_procs(void);`

*Fortran:*

*Interface:*           `integer function omp_get_num_procs()`

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.5.

### 3.6.2 `omp_set_default_device` – Set the default device for target regions

*Description:*

Get the value of the *default-device-var* ICV, which is used for target regions without a device clause. The argument shall be a nonnegative device number, `omp_initial_device`, or `omp_invalid_device`.

The effect of running this routine in a *target* region is unspecified.

*C/C++:*

*Prototype:*           `void omp_set_default_device(int device_num);`

*Fortran:*

*Interface:*           `subroutine omp_set_default_device(device_num)`  
                         `integer device_num`

*See also:* Section 4.6 [OMP\_DEFAULT\_DEVICE], page 61, Section 3.6.3 [omp\_get\_default\_device], page 27,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.29.

### 3.6.3 `omp_get_default_device` – Get the default device for target regions

*Description:*

Get the value of the *default-device-var* ICV, which is used for target regions without a device clause. The value is either a nonnegative device number, `omp_initial_device` or `omp_invalid_device`. Note that for the host, the ICV can have two values: either the value of the named constant `omp_initial_device` or the value returned by the `omp_get_num_devices` routine.

The effect of running this routine in a *target* region is unspecified.

*C/C++:*

*Prototype:*           `int omp_get_default_device(void);`

*Fortran:*

*Interface:*            `integer function omp_get_default_device()`

*See also:*    Section 4.6 [OMP\_DEFAULT\_DEVICE], page 61, Section 3.6.2 [omp\_set\_default\_device], page 27, Section 3.6.9 [omp\_get\_initial\_device], page 30,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.30.

### 3.6.4 `omp_get_num_devices` – Number of target devices

*Description:*

Returns the number of available non-host devices.

The effect of running this routine in a `target` region is unspecified.

Note that in GCC the function is marked pure, i.e. as returning always the same number. When GCC was not configured to support offloading, it is replaced by zero; compile with `-fno-builtin-omp_get_num_devices` if a run-time function is desired.

*C/C++:*

*Prototype:*            `int omp_get_num_devices(void);`

*Fortran:*

*Interface:*            `integer function omp_get_num_devices()`

*See also:*    Section 3.6.9 [omp\_get\_initial\_device], page 30,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.31.

### 3.6.5 `omp_get_device_num` – Return device number of current device

*Description:*

This function returns a device number that represents the device that the current thread is executing on. When called on the host, it returns the same value as returned by the `omp_get_initial_device` function as required since OpenMP 5.0.

*C/C++*

*Prototype:*            `int omp_get_device_num(void);`

*Fortran:*

*Interface:*            `integer function omp_get_device_num()`

*See also:*    Section 3.6.9 [omp\_get\_initial\_device], page 30,

*Reference:*   OpenMP specification v5.0 (<https://www.openmp.org>), Section 3.2.37.

### 3.6.6 `omp_get_device_from_uid` – Obtain the device number to a unique id

*Description:*

This function returns the device number associated with the passed unique-identifier (UID) string. If no device with this UID is available, the value `omp_`

`invalid_device` is returned. The effect of running this routine in a `target` region is unspecified.

GCC treats the UID string case sensitive; for the initial device, GCC currently only accepts the value `OMP_INITIAL_DEVICE` and returns for it the value of `omp_initial_device`.

*C/C++:*

*Prototype:*        `int omp_get_device_from_uid(const char *uid);`

*Fortran:*

*Interface:*        `integer function omp_get_device_from_uid(uid)`  
                       `character(len=*), intent(in) :: uid`

*See also:*    Section 3.6.7 [`omp_get_uid_from_device`], page 29, Chapter 12 [Offload-Target Specifics], page 111,

*Reference:*   OpenMP specification v6.0 (<https://www.openmp.org>), Section 24.7

### 3.6.7 `omp_get_uid_from_device` – Obtain the unique id of a device

*Description:*

This function returns a pointer to a string that represents a unique identifier (UID) for the device specified by `device_num`. It returns a NULL (C/C++) or a disassociated pointer (Fortran) for `omp_invalid_device`. The effect of running this routine in a `target` region is unspecified.

GCC currently returns for initial device the value `OMP_INITIAL_DEVICE`.

*C/C++:*

*Prototype:*        `const char *omp_get_uid_from_device(int device_num);`

*Fortran:*

*Interface:*        `character(:) function omp_get_uid_from_device(device_num)`

*Interface:*        `pointer :: omp_get_uid_from_device`  
                       `integer, intent(in) :: device_num`

*See also:*    Section 3.6.7 [`omp_get_uid_from_device`], page 29, Chapter 12 [Offload-Target Specifics], page 111,

*Reference:*   OpenMP specification v6.0 (<https://www.openmp.org>), Section 24.8

### 3.6.8 `omp_is_initial_device` – Whether executing on the host device

*Description:*

This function returns `true` if currently running on the host device, `false` otherwise. Here, `true` and `false` represent their language-specific counterparts.

Note that in GCC this function call is already folded to a constant in the compiler; compile with `-fno-builtin-omp_is_initial_device` if a run-time function is desired.

*C/C++:*

*Prototype:*        `int omp_is_initial_device(void);`

*Fortran:*

*Interface:*        `logical function omp_is_initial_device()`

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.34.

### 3.6.9 `omp_get_initial_device` – Return device number of initial device

*Description:*

This function returns a device number that represents the host device. Since OpenMP 5.1, this is equal to the value returned by the `omp_get_num_devices` function; since OpenMP 6.0 it may also return the value of `omp_initial_device`.

The effect of running this routine in a `target` region is unspecified.

Note that GCC inlines this function unless you compile with `-fno-builtin-omp_get_initial_device`. If GCC was not configured to support offloading, it expands to constant zero; in non-host code it expands to `omp_initial_device`; and otherwise it is replaced with a call to `omp_get_num_devices`.

*C/C++:*

*Prototype:*        `int omp_get_initial_device(void);`

*Fortran:*

*Interface:*        `integer function omp_get_initial_device()`

*See also:* Section 3.6.4 [`omp_get_num_devices`], page 28,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.2.35.

## 3.7 Device Memory Routines

Routines related to memory allocation and managing corresponding pointers on devices. They have C linkage and do not throw exceptions.

### 3.7.1 `omp_target_alloc` – Allocate device memory

*Description:*

This routine allocates *size* bytes of memory in the device environment associated with the device number *device\_num*. If successful, a device pointer is returned, otherwise a null pointer.

In GCC, when the device is the host or the device shares memory with the host, the memory is allocated on the host; in that case, when *size* is zero, either NULL or a unique pointer value that can later be successfully passed to `omp_target_free` is returned. When the allocation is not performed on the host, a null pointer is returned when *size* is zero; in that case, additionally a diagnostic might be printed to standard error (stderr).

Running this routine in a `target` region except on the initial device is not supported.

*C/C++*

*Prototype:*        `void *omp_target_alloc(size_t size, int device_num)`

*Fortran:*

*Interface:*        `type(c_ptr) function omp_target_alloc(size,  
device_num) bind(C)  
use, intrinsic :: iso_c_binding, only: c_ptr, c_int,  
c_size_t  
integer(c_size_t), value :: size  
integer(c_int), value :: device_num`

*See also:*    Section 3.7.2 [omp\_target\_free], page 31, Section 3.7.11 [omp\_target\_associate\_ptr],  
page 39,

*Reference:*    OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.8.1

### 3.7.2 omp\_target\_free – Free device memory

*Description:*

This routine frees memory allocated by the `omp_target_alloc` routine. The `device_ptr` argument must be either a null pointer or a device pointer returned by `omp_target_alloc` for the specified `device_num`. The device number `device_num` must be a conforming device number.

Running this routine in a `target` region except on the initial device is not supported.

*C/C++*

*Prototype:*        `void omp_target_free(void *device_ptr, int  
device_num)`

*Fortran:*

*Interface:*        `subroutine omp_target_free(device_ptr, device_num)  
bind(C)  
use, intrinsic :: iso_c_binding, only: c_ptr, c_int  
type(c_ptr), value :: device_ptr  
integer(c_int), value :: device_num`

*See also:*    Section 3.7.1 [omp\_target\_alloc], page 30, Section 3.7.12 [omp\_target\_disassociate\_ptr],  
page 40,

*Reference:*    OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.8.2

### 3.7.3 omp\_target\_is\_present – Check whether storage is mapped

*Description:*

This routine tests whether storage, identified by the host pointer `ptr` is mapped to the device specified by `device_num`. If so, it returns a nonzero value and otherwise zero.

In GCC, this includes self mapping such that `omp_target_is_present` returns `true` when `device_num` specifies the host or when the host and the device share

Running this routine in a **target** region except on the initial device is not supported.

```

Prototype:      int omp_target_is_present(const void *ptr,
                                     int device_num)

```

```

Interface:      integer(c_int) function omp_target_is_present(ptr,
                  &
                  device_num) bind(C)
                  use, intrinsic :: iso_c_binding, only: c_ptr, c_int
                  type(c_ptr), value :: ptr
                  integer(c_int), value :: device_num

```

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.8.3

*Description:*

Running this routine in a **target** region except on the initial device is not supported.

```

Prototype:      int omp_target_is_accessible(const void *ptr,
                    size_t size,
                    int device_num)

```

```
Interface:  integer(c_int) function omp_target_is_
            accessible(ptr, &
```

```

        size, device_num) bind(C)
    use, intrinsic :: iso_c_binding, only: c_ptr,
    c_size_t, c_int
    type(c_ptr), value :: ptr
    integer(c_size_t), value :: size
    integer(c_int), value :: device_num

```

*See also:* Section 3.7.11 [omp\_target\_associate\_ptr], page 39,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.8.4

### 3.7.5 omp\_target\_memcpy – Copy data between devices

*Description:*

This routine copies *length* of bytes of data from the device identified by device number *src\_device\_num* to device *dst\_device\_num*. The data is copied from the source device from the address provided by *src*, shifted by the offset of *src\_offset* bytes, to the destination device's *dst* address shifted by *dst\_offset*. The routine returns zero on success and non-zero otherwise.

Running this routine in a **target** region except on the initial device is not supported.

*C/C++*

*Prototype:*

```

int omp_target_memcpy(void *dst,
    const void *src,
    size_t length,
    size_t dst_offset,
    size_t src_offset,
    int dst_device_num,
    int src_device_num)

```

*Fortran:*

*Interface:*

```

integer(c_int) function omp_target_memcpy( &
    dst, src, length, dst_offset, src_offset, &
    dst_device_num, src_device_num) bind(C)
    use, intrinsic :: iso_c_binding, only: c_ptr,
    c_size_t, c_int
    type(c_ptr), value :: dst, src
    integer(c_size_t), value :: length, dst_offset,
    src_offset
    integer(c_int), value :: dst_device_num, src_
    device_num

```

*See also:* Section 3.7.6 [omp\_target\_memcpy\_async], page 34, Section 3.7.7 [omp\_target\_memcpy\_rect], page 35,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.8.5

### 3.7.6 `omp_target_memcpy_async` – Copy data between devices asynchronously

*Description:*

This routine copies asynchronously *length* of bytes of data from the device identified by device number *src\_device\_num* to device *dst\_device\_num*. The data is copied from the source device from the address provided by *src*, shifted by the offset of *src\_offset* bytes, to the destination device's *dst* address shifted by *dst\_offset*. Task dependence is expressed by passing an array of depend objects to *depobj\_list*, where the number of array elements is passed as *depobj\_count*; if the count is zero, the *depobj\_list* argument is ignored. In C++ and Fortran, the *depobj\_list* argument can also be omitted in that case. The routine returns zero if the copying process has successfully been started and non-zero otherwise.

Running this routine in a `target` region except on the initial device is not supported.

*C/C++*

*Prototype:*

```
int omp_target_memcpy_async(void *dst,
    const void *src,
    size_t length,
    size_t dst_offset,
    size_t src_offset,
    int dst_device_num,
    int src_device_num,
    int depobj_count,
    omp_depend_t *depobj_list)
```

*Fortran:*

*Interface:*

```
integer(c_int) function omp_target_memcpy_async( &
    dst, src, length, dst_offset, src_offset, &
    dst_device_num, src_device_num, &
    depobj_count, depobj_list) bind(C)
use, intrinsic :: iso_c_binding, only: c_ptr,
c_size_t, c_int
type(c_ptr), value :: dst, src
integer(c_size_t), value :: length, dst_offset,
src_offset
integer(c_int), value :: dst_device_num, src_
device_num, depobj_count
integer(omp_depend_kind), optional :: depobj_
list(*)
```

*See also:* Section 3.7.5 [`omp_target_memcpy`], page 33, Section 3.7.8 [`omp_target_memcpy_rect_async`], page 36,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.8.7

### 3.7.7 `omp_target_memcpy_rect` – Copy a subvolume of data between devices

*Description:*

This routine copies a subvolume of data from the device identified by device number *src\_device\_num* to device *dst\_device\_num*. The array has *num\_dims* dimensions and each array element has a size of *element\_size* bytes. The *volume* array specifies how many elements per dimension are copied. The full sizes of the destination and source arrays are given by the *dst\_dimensions* and *src\_dimensions* arguments, respectively. The offset per dimension to the first element to be copied is given by the *dst\_offset* and *src\_offset* arguments. The routine returns zero on success and non-zero otherwise.

The OpenMP specification only requires that *num\_dims* up to three is supported. In order to find implementation-specific maximally supported number of dimensions, the routine returns this value when invoked with a null pointer to both the *dst* and *src* arguments. As GCC supports arbitrary dimensions, it returns `INT_MAX`.

The device-number arguments must be conforming device numbers, the *src* and *dst* must be either both null pointers or all of the following must be fulfilled: *element\_size* and *num\_dims* must be positive and the *volume*, offset and dimension arrays must have at least *num\_dims* dimensions.

Running this routine in a `target` region is not supported except on the initial device.

*C/C++*

*Prototype:*

```
int omp_target_memcpy_rect(void *dst,
    const void *src,
    size_t element_size,
    int num_dims,
    const size_t *volume,
    const size_t *dst_offset,
    const size_t *src_offset,
    const size_t *dst_dimensions,
    const size_t *src_dimensions,
    int dst_device_num,
    int src_device_num)
```

*Fortran:*

*Interface:*

```
integer(c_int) function omp_target_memcpy_rect( &
    dst, src, element_size, num_dims, volume, &
    dst_offset, src_offset, dst_dimensions, &
    src_dimensions, dst_device_num, src_device_num)
bind(C)
use, intrinsic :: iso_c_binding, only: c_ptr,
    c_size_t, c_int
type(c_ptr), value :: dst, src
integer(c_size_t), value :: element_size,
    dst_offset, src_offset
```

```
integer(c_size_t), value :: volume, dst_dimensions,
src_dimensions
integer(c_int), value :: num_dims, dst_device_num,
src_device_num
```

*See also:* Section 3.7.8 [omp\_target\_memcpy\_rect\_async], page 36, Section 3.7.5 [omp\_target\_memcpy], page 33, Chapter 12 [Offload-Target Specifics], page 111,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.8.6

### 3.7.8 omp\_target\_memcpy\_rect\_async – Copy a subvolume of data between devices asynchronously

*Description:*

This routine copies asynchronously a subvolume of data from the device identified by device number *src\_device\_num* to device *dst\_device\_num*. The array has *num\_dims* dimensions and each array element has a size of *element\_size* bytes. The *volume* array specifies how many elements per dimension are copied. The full sizes of the destination and source arrays are given by the *dst\_dimensions* and *src\_dimensions* arguments, respectively. The offset per dimension to the first element to be copied is given by the *dst\_offset* and *src\_offset* arguments. Task dependence is expressed by passing an array of depend objects to *depobj\_list*, where the number of array elements is passed as *depobj\_count*; if the count is zero, the *depobj\_list* argument is ignored. In C++ and Fortran, the *depobj\_list* argument can also be omitted in that case. The routine returns zero on success and non-zero otherwise.

The OpenMP specification only requires that *num\_dims* up to three is supported. In order to find implementation-specific maximally supported number of dimensions, the routine returns this value when invoked with a null pointer to both the *dst* and *src* arguments. As GCC supports arbitrary dimensions, it returns INT\_MAX.

The device-number arguments must be conforming device numbers, the *src* and *dst* must be either both null pointers or all of the following must be fulfilled: *element\_size* and *num\_dims* must be positive and the *volume*, offset and dimension arrays must have at least *num\_dims* dimensions.

Running this routine in a *target* region is not supported except on the initial device.

*C/C++*

*Prototype:*

```
int omp_target_memcpy_rect_async(void *dst,
const void *src,
size_t element_size,
int num_dims,
const size_t *volume,
const size_t *dst_offset,
const size_t *src_offset,
const size_t *dst_dimensions,
```

```

const size_t *src_dimensions,
int dst_device_num,
int src_device_num,
int depobj_count,
omp_depend_t *depobj_list)

```

*Fortran:*

*Interface:*

```

integer(c_int) function omp_target_memcpy_rect_
  async( &
    dst, src, element_size, num_dims, volume, &
    dst_offset, src_offset, dst_dimensions, &
    src_dimensions, dst_device_num, src_device_num, &
    depobj_count, depobj_list) bind(C)
use, intrinsic :: iso_c_binding, only: c_ptr,
  c_size_t, c_int
type(c_ptr), value :: dst, src
integer(c_size_t), value :: element_size,
  dst_offset, src_offset
integer(c_size_t), value :: volume, dst_dimensions,
  src_dimensions
integer(c_int), value :: num_dims, dst_device_num,
  src_device_num
integer(c_int), value :: depobj_count
integer(omp_depend_kind), optional :: depobj_
  list(*)

```

*See also:* Section 3.7.7 [omp\_target\_memcpy\_rect], page 35, Section 3.7.6 [omp\_target\_memcpy\_async], page 34, Chapter 12 [Offload-Target Specifics], page 111,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.8.8

### 3.7.9 omp\_target\_memset – Set bytes in device memory

*Description:*

This routine fills memory on the device identified by device number *device\_num*. Starting from the device address *ptr*, the first *count* bytes are set to the value *val*, converted to `unsigned char`. If *count* is zero, the routine has no effect; if *ptr* is NULL, the behavior is unspecified. The function returns *ptr*.

The *device\_num* must be a conforming device number and *ptr* must be a valid device pointer for that device. Running this routine in a **target** region except on the initial device is not supported.

*C/C++*

*Prototype:*

```

void *omp_target_memcpy(void *ptr,
  int val,
  size_t count,
  int device_num)

```

*Fortran:*

*Interface:*

```

type(c_ptr) function omp_target_memset( &
    ptr, val, count, device_num) bind(C)
use, intrinsic :: iso_c_binding, only: c_ptr,
    c_size_t, c_int
type(c_ptr), value :: ptr
integer(c_size_t), value :: count
integer(c_int), value :: val, device_num

```

*See also:* Section 3.7.10 [omp\_target\_memset\_async], page 38,

*Reference:* OpenMP specification v6.0 (<https://www.openmp.org>), Section 25.8.1

### 3.7.10 omp\_target\_memset – Set bytes in device memory asynchronously

*Description:*

This routine fills memory on the device identified by device number *device\_num*. Starting from the device address *ptr*, the first *count* bytes are set to the value *val*, converted to `unsigned char`. If *count* is zero, the routine has no effect; if *ptr* is `NULL`, the behavior is unspecified. Task dependence is expressed by passing an array of depend objects to *depobj\_list*, where the number of array elements is passed as *depobj\_count*; if the count is zero, the *depobj\_list* argument is ignored. In C++ and Fortran, the *depobj\_list* argument can also be omitted in that case. The function returns *ptr*.

The *device\_num* must be a conforming device number and *ptr* must be a valid device pointer for that device. Running this routine in a `target` region except on the initial device is not supported.

*C/C++*

*Prototype:*

```

void *omp_target_memcpy_async(void *ptr,
    int val,
    size_t count,
    int device_num,
    int depobj_count,
    omp_depend_t *depobj_list)

```

*Fortran:*

*Interface:*

```

type(c_ptr) function omp_target_memset_async( &
    ptr, val, count, device_num, &
    depobj_count, depobj_list) bind(C)
use, intrinsic :: iso_c_binding, only: c_ptr,
    c_size_t, c_int
type(c_ptr), value :: ptr
integer(c_size_t), value :: count
integer(c_int), value :: val, device_num, depobj_
count
integer(omp_depend_kind), optional :: depobj_
list(*)

```

*See also:* Section 3.7.9 [omp\_target\_memset], page 37,

*Reference:* OpenMP specification v6.0 (<https://www.openmp.org>), Section 25.8.2

### 3.7.11 omp\_target\_associate\_ptr – Associate a device pointer with a host pointer

*Description:*

This routine associates storage on the host with storage on a device identified by *device\_num*. The device pointer is usually obtained by calling `omp_target_alloc` or by other means (but not by using the `map` clauses or the `declare target` directive). The host pointer should point to memory that has a storage size of at least *size*.

The *device\_offset* parameter specifies the offset into *device\_ptr* that is used as the base address for the device side of the mapping; the storage size should be at least *device\_offset* plus *size*.

After the association, the host pointer can be used in a `map` clause and in the `to` and `from` clauses of the `target update` directive to transfer data between the associated pointers. The reference count of such associated storage is infinite. The association can be removed by calling `omp_target_disassociate_ptr` which should be done before the lifetime of either storage ends.

The routine returns nonzero (EINVAL) when the *device\_num* is invalid, for when the initial device or the associated device shares memory with the host. `omp_target_associate_ptr` returns zero if *host\_ptr* points into already associated storage that is fully inside of a previously associated memory. Otherwise, if the association was successful zero is returned; if none of the cases above apply, nonzero (EINVAL) is returned.

The `omp_target_is_present` routine can be used to test whether associated storage for a device pointer exists.

Running this routine in a `target` region except on the initial device is not supported.

*C/C++*

*Prototype:*

```
int omp_target_associate_ptr(const void *host_ptr,
                             const void *device_ptr,
                             size_t size,
                             size_t device_offset,
                             int device_num)
```

*Fortran:*

*Interface:*

```
integer(c_int) function omp_target_associate_ptr(host_ptr, &
  device_ptr, size, device_offset, device_num)
  bind(C)
  use, intrinsic :: iso_c_binding, only: c_ptr, c_int,
  c_size_t
  type(c_ptr), value :: host_ptr, device_ptr
```

```
integer(c_size_t), value :: size, device_offset
integer(c_int), value :: device_num
```

*See also:* Section 3.7.12 [omp\_target\_disassociate\_ptr], page 40, Section 3.7.3 [omp\_target\_is\_present], page 31, Section 3.7.1 [omp\_target\_alloc], page 30,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.8.9

### 3.7.12 omp\_target\_disassociate\_ptr – Remove device–host pointer association

*Description:*

This routine removes the storage association established by calling `omp_target_associate_ptr` and sets the reference count to zero, even if `omp_target_associate_ptr` was invoked multiple times for host pointer `ptr`. If applicable, the device memory needs to be freed by the user.

If an associated device storage location for the `device_num` was found and has infinite reference count, the association is removed and zero is returned. In all other cases, nonzero (`EINVAL`) is returned and no other action is taken.

Note that passing a host pointer where the association to the device pointer was established with the `declare target` directive yields undefined behavior.

Running this routine in a `target` region except on the initial device is not supported.

*C/C++*

*Prototype:*

```
int omp_target_disassociate_ptr(const void *ptr,
                               int device_num)
```

*Fortran:*

*Interface:*

```
integer(c_int) function omp_target_disassociate_
ptr(ptr, &
    device_num) bind(C)
use, intrinsic :: iso_c_binding, only: c_ptr, c_int
type(c_ptr), value :: ptr
integer(c_int), value :: device_num
```

*See also:* Section 3.7.11 [omp\_target\_associate\_ptr], page 39,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.8.10

### 3.7.13 omp\_get\_mapped\_ptr – Return device pointer to a host pointer

*Description:*

If the device number refers to the initial device or to a device with memory accessible from the host (shared memory), the `omp_get_mapped_ptr` routine returns the value of the passed `ptr`. Otherwise, if associated storage to the passed host pointer `ptr` exists on device associated with `device_num`, it returns that pointer. In all other cases and in cases of an error, a null pointer is returned.

The association of storage location is established either via an explicit or implicit `map` clause, the `declare target` directive or the `omp_target_associate_ptr` routine.

Running this routine in a `target` region except on the initial device is not supported.

*C/C++*

*Prototype:*        `void *omp_get_mapped_ptr(const void *ptr, int device_num);`

*Fortran:*

*Interface:*        `type(c_ptr) function omp_get_mapped_ptr(ptr,  
device_num) bind(C)  
use, intrinsic :: iso_c_binding, only: c_ptr, c_int  
type(c_ptr), value :: ptr  
integer(c_int), value :: device_num`

*See also:*    Section 3.7.11 [`omp_target_associate_ptr`], page 39,

*Reference:*   OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.8.11

## 3.8 Lock Routines

Initialize, set, test, unset and destroy simple and nested locks. The routines have C linkage and do not throw exceptions.

### 3.8.1 `omp_init_lock` – Initialize simple lock

*Description:*

Initialize a simple lock. After initialization, the lock is in an unlocked state.

*C/C++:*

*Prototype:*        `void omp_init_lock(omp_lock_t *lock);`

*Fortran:*

*Interface:*        `subroutine omp_init_lock(svar)  
integer(omp_lock_kind), intent(out) :: svar`

*See also:*    Section 3.8.3 [`omp_destroy_lock`], page 42,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.3.1.

### 3.8.2 `omp_init_nest_lock` – Initialize nested lock

*Description:*

Initialize a nested lock. After initialization, the lock is in an unlocked state and the nesting count is set to zero.

*C/C++:*

*Prototype:*        `void omp_init_nest_lock(omp_nest_lock_t *lock);`

*Fortran:*

*Interface:*        `subroutine omp_init_nest_lock(nvar)  
integer(omp_nest_lock_kind), intent(out) :: nvar`

*See also:* Section 3.8.4 [omp\_destroy\_nest\_lock], page 42,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.3.1.

### 3.8.3 omp\_destroy\_lock – Destroy simple lock

*Description:*

Destroy a simple lock. In order to be destroyed, a simple lock must be in the unlocked state.

*C/C++:*

*Prototype:*        `void omp_destroy_lock(omp_lock_t *lock);`

*Fortran:*

*Interface:*        `subroutine omp_destroy_lock(svar)`  
                      `integer(omp_lock_kind), intent(inout) :: svar`

*See also:* Section 3.8.1 [omp\_init\_lock], page 41,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.3.3.

### 3.8.4 omp\_destroy\_nest\_lock – Destroy nested lock

*Description:*

Destroy a nested lock. In order to be destroyed, a nested lock must be in the unlocked state and its nesting count must equal zero.

*C/C++:*

*Prototype:*        `void omp_destroy_nest_lock(omp_nest_lock_t *);`

*Fortran:*

*Interface:*        `subroutine omp_destroy_nest_lock(nvar)`  
                      `integer(omp_nest_lock_kind), intent(inout) :: nvar`

*See also:* Section 3.8.1 [omp\_init\_lock], page 41,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.3.3.

### 3.8.5 omp\_set\_lock – Wait for and set simple lock

*Description:*

Before setting a simple lock, the lock variable must be initialized by `omp_init_lock`. The calling thread is blocked until the lock is available. If the lock is already held by the current thread, a deadlock occurs.

*C/C++:*

*Prototype:*        `void omp_set_lock(omp_lock_t *lock);`

*Fortran:*

*Interface:*        `subroutine omp_set_lock(svar)`  
                      `integer(omp_lock_kind), intent(inout) :: svar`

*See also:* Section 3.8.1 [omp\_init\_lock], page 41, Section 3.8.9 [omp\_test\_lock], page 44, Section 3.8.7 [omp\_unset\_lock], page 43,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.3.4.

### 3.8.6 `omp_set_nest_lock` – Wait for and set nested lock

*Description:*

Before setting a nested lock, the lock variable must be initialized by `omp_init_nest_lock`. The calling thread is blocked until the lock is available. If the lock is already held by the current thread, the nesting count for the lock is incremented.

*C/C++:*

*Prototype:*            `void omp_set_nest_lock(omp_nest_lock_t *lock);`

*Fortran:*

*Interface:*            `subroutine omp_set_nest_lock(nvar)  
                          integer(omp_nest_lock_kind), intent(inout) :: nvar`

*See also:*    Section 3.8.2 [`omp_init_nest_lock`], page 41, Section 3.8.8 [`omp_unset_nest_lock`], page 43,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.3.4.

### 3.8.7 `omp_unset_lock` – Unset simple lock

*Description:*

A simple lock about to be unset must have been locked by `omp_set_lock` or `omp_test_lock` before. In addition, the lock must be held by the thread calling `omp_unset_lock`. Then, the lock becomes unlocked. If one or more threads attempted to set the lock before, one of them is chosen to, again, set the lock to itself.

*C/C++:*

*Prototype:*            `void omp_unset_lock(omp_lock_t *lock);`

*Fortran:*

*Interface:*            `subroutine omp_unset_lock(svar)  
                          integer(omp_lock_kind), intent(inout) :: svar`

*See also:*    Section 3.8.5 [`omp_set_lock`], page 42, Section 3.8.9 [`omp_test_lock`], page 44,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.3.5.

### 3.8.8 `omp_unset_nest_lock` – Unset nested lock

*Description:*

A nested lock about to be unset must have been locked by `omp_set_nested_lock` or `omp_test_nested_lock` before. In addition, the lock must be held by the thread calling `omp_unset_nested_lock`. If the nesting count drops to zero, the lock becomes unlocked. If one ore more threads attempted to set the lock before, one of them is chosen to, again, set the lock to itself.

*C/C++:*

*Prototype:*            `void omp_unset_nest_lock(omp_nest_lock_t *lock);`

*Fortran:*

*Interface:*           subroutine omp\_unset\_nest\_lock(nvar)  
                          integer(omp\_nest\_lock\_kind), intent(inout) :: nvar

*See also:*   Section 3.8.6 [omp\_set\_nest\_lock], page 43,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.3.5.

### 3.8.9 omp\_test\_lock – Test and set simple lock if available

*Description:*

Before setting a simple lock, the lock variable must be initialized by `omp_init_lock`. Contrary to `omp_set_lock`, `omp_test_lock` does not block if the lock is not available. This function returns `true` upon success, `false` otherwise. Here, `true` and `false` represent their language-specific counterparts.

*C/C++:*

*Prototype:*           int omp\_test\_lock(omp\_lock\_t \*lock);

*Fortran:*

*Interface:*           logical function omp\_test\_lock(svar)  
                          integer(omp\_lock\_kind), intent(inout) :: svar

*See also:*   Section 3.8.1 [omp\_init\_lock], page 41, Section 3.8.5 [omp\_set\_lock], page 42,  
              Section 3.8.5 [omp\_set\_lock], page 42,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.3.6.

### 3.8.10 omp\_test\_nest\_lock – Test and set nested lock if available

*Description:*

Before setting a nested lock, the lock variable must be initialized by `omp_init_nest_lock`. Contrary to `omp_set_nest_lock`, `omp_test_nest_lock` does not block if the lock is not available. If the lock is already held by the current thread, the new nesting count is returned. Otherwise, the return value equals zero.

*C/C++:*

*Prototype:*           int omp\_test\_nest\_lock(omp\_nest\_lock\_t \*lock);

*Fortran:*

*Interface:*           logical function omp\_test\_nest\_lock(nvar)  
                          integer(omp\_nest\_lock\_kind), intent(inout) :: nvar

*See also:*   Section 3.8.1 [omp\_init\_lock], page 41, Section 3.8.5 [omp\_set\_lock], page 42,  
              Section 3.8.5 [omp\_set\_lock], page 42,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.3.6.

## 3.9 Timing Routines

Portable, thread-based, wall clock timer. The routines have C linkage and do not throw exceptions.

### 3.9.1 omp\_get\_wtick – Get timer precision

*Description:*

Gets the timer precision, i.e., the number of seconds between two successive clock ticks.

*C/C++:*

*Prototype:*        `double omp_get_wtick(void);`

*Fortran:*

*Interface:*        `double precision function omp_get_wtick()`

*See also:*    Section 3.9.2 [omp\_get\_wtime], page 45,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.4.2.

### 3.9.2 omp\_get\_wtime – Elapsed wall clock time

*Description:*

Elapsed wall clock time in seconds. The time is measured per thread, no guarantee can be made that two distinct threads measure the same time. Time is measured from some "time in the past", which is an arbitrary time guaranteed not to change during the execution of the program.

*C/C++:*

*Prototype:*        `double omp_get_wtime(void);`

*Fortran:*

*Interface:*        `double precision function omp_get_wtime()`

*See also:*    Section 3.9.1 [omp\_get\_wtick], page 45,

*Reference:*   OpenMP specification v4.5 (<https://www.openmp.org>), Section 3.4.1.

## 3.10 Event Routine

Support for event objects. The routine has C linkage and do not throw exceptions.

### 3.10.1 omp\_fulfill\_event – Fulfill and destroy an OpenMP event

*Description:*

Fulfill the event associated with the event handle argument. Currently, it is only used to fulfill events generated by detach clauses on task constructs - the effect of fulfilling the event is to allow the task to complete.

The result of calling `omp_fulfill_event` with an event handle other than that generated by a detach clause is undefined. Calling it with an event handle that has already been fulfilled is also undefined.

*C/C++:*

*Prototype:*        `void omp_fulfill_event(omp_event_handle_t event);`

*Fortran:*

*Interface:*        `subroutine omp_fulfill_event(event)  
                      integer (kind=omp_event_handle_kind) :: event`

*Reference:*   OpenMP specification v5.0 (<https://www.openmp.org>), Section 3.5.1.

## 3.11 Interoperability Routines

Routines to obtain properties from an object of OpenMP interop type. They have C linkage and do not throw exceptions.

### 3.11.1 `omp_get_num_interop_properties` – Get the number of implementation-specific properties

*Description:*

The `omp_get_num_interop_properties` function returns the number of implementation-defined interoperability properties available for the passed *interop*, extending the OpenMP-defined properties. The available OpenMP interop-property-type values range from `omp_ipr_first` to the value returned by `omp_get_num_interop_properties` minus one.

No implementation-defined properties are currently defined in GCC.

*C/C++:*

*Prototype:*            `int omp_get_num_interop_properties(const omp_interop_t interop)`

*Fortran:*

*Interface:*           `integer function omp_get_num_interop_properties(interop)  
integer(omp_interop_kind), intent(in) :: interop`

*See also:*    Section 3.11.5 [`omp_get_interop_name`], page 48, Section 3.11.6 [`omp_get_interop_type_desc`], page 49,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.12.1, OpenMP specification v6.0 (<https://www.openmp.org>), Section 26.1

### 3.11.2 `omp_get_interop_int` – Obtain integer-valued interoperability property

*Description:*

The `omp_get_interop_int` function returns the integer value associated with the *property\_id* interoperability property of the passed *interop* object. The *ret\_code* argument is optional, i.e. it can be omitted in C++ and Fortran or used with NULL as argument in C and C++. If successful, *ret\_code* (if present) is set to `omp_irc_success`.

In GCC, the effect of running this routine in a `target` region that is not the initial device is unspecified.

GCC implements the OpenMP 6.0 version of this function for C and C++, which is not compatible with its type signature in previous versions of the OpenMP specification. In older versions, the type `int*` was used for the *ret\_code* argument in place of a pointer to the enumerated type `omp_interop_rc_t`.

*C/C++:*

*Prototype:*           `omp_intptr_t omp_get_interop_int(const omp_interop_t interop, omp_interop_property_t property_id, omp_interop_rc_t *ret_code)`

*Fortran:*

*Interface:*

```
integer(c_intptr_t) function omp_get_interop_
int(interop, property_id, ret_code)
use, intrinsic :: iso_c_binding, only : c_intptr_t
integer(omp_interop_kind), intent(in) :: interop
integer(omp_interop_property_kind) property_id
integer(omp_interop_rc_kind), optional,
intent(out) :: ret_code
```

*See also:* Section 3.11.3 [omp\_get\_interop\_ptr], page 47, Section 3.11.4 [omp\_get\_interop\_str], page 48, Section 3.11.7 [omp\_get\_interop\_rc\_desc], page 49, Chapter 12 [Offload-Target Specifics], page 111,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.12.2, OpenMP specification v6.0 (<https://www.openmp.org>), Section 26.2

### 3.11.3 omp\_get\_interop\_ptr – Obtain pointer-valued interoperability property

*Description:*

The `omp_get_interop_int` function returns the pointer value associated with the *property\_id* interoperability property of the passed *interop* object. The *ret\_code* argument is optional, i.e. it can be omitted in C++ and Fortran or used with NULL as argument in C and C++. If successful, *ret\_code* (if present) is set to `omp_irc_success`.

In GCC, the effect of running this routine in a `target` region that is not the initial device is unspecified.

GCC implements the OpenMP 6.0 version of this function for C and C++, which is not compatible with its type signature in previous versions of the OpenMP specification. In older versions, the type `int*` was used for the *ret\_code* argument in place of a pointer to the enumerated type `omp_interop_rc_t`.

*C/C++:*

*Prototype:*

```
void *omp_get_interop_ptr(const omp_interop_t
interop, omp_interop_property_t property_id,
omp_interop_rc_t *ret_code)
```

*Fortran:*

*Interface:*

```
type(c_ptr) function omp_get_interop_int(interop,
property_id, ret_code)
use, intrinsic :: iso_c_binding, only : c_ptr
integer(omp_interop_kind), intent(in) :: interop
integer(omp_interop_property_kind) property_id
integer(omp_interop_rc_kind), optional,
intent(out) :: ret_code
```

*See also:* Section 3.11.2 [omp\_get\_interop\_int], page 46, Section 3.11.4 [omp\_get\_interop\_str], page 48, Section 3.11.7 [omp\_get\_interop\_rc\_desc], page 49, Chapter 12 [Offload-Target Specifics], page 111,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.12.3,  
OpenMP specification v6.0 (<https://www.openmp.org>), Section 26.3

### 3.11.4 `omp_get_interop_str` – Obtain string-valued interoperability property

*Description:*

The `omp_get_interop_str` function returns the string value associated with the *property\_id* interoperability property of the passed *interop* object. The *ret\_code* argument is optional, i.e. it can be omitted in C++ and Fortran or used with NULL as argument in C and C++. If successful, *ret\_code* (if present) is set to `omp_irc_success`.

In GCC, the effect of running this routine in a `target` region that is not the initial device is unspecified.

GCC implements the OpenMP 6.0 version of this function for C and C++, which is not compatible with its type signature in previous versions of the OpenMP specification. In older versions, the type `int*` was used for the *ret\_code* argument in place of a pointer to the enumerated type `omp_interop_rc_t`.

*C/C++:*

*Prototype:*        `const char *omp_get_interop_str(const omp_interop_t  
interop, omp_interop_property_t property_id,  
omp_interop_rc_t *ret_code)`

*Fortran:*

*Interface:*        `character(:) function omp_get_interop_str(interop,  
property_id, ret_code)  
pointer :: omp_get_interop_str  
integer(omp_interop_kind), intent(in) :: interop  
integer(omp_interop_property_kind) property_id  
integer(omp_interop_rc_kind), optional,  
intent(out) :: ret_code`

*See also:* Section 3.11.2 [`omp_get_interop_int`], page 46, Section 3.11.3 [`omp_get_interop_ptr`], page 47, Section 3.11.7 [`omp_get_interop_rc_desc`], page 49, Chapter 12 [Offload-Target Specifics], page 111,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.12.4,  
OpenMP specification v6.0 (<https://www.openmp.org>), Section 26.4

### 3.11.5 `omp_get_interop_name` – Obtain the name of an `interop_` property value as string

*Description:*

The `omp_get_interop_name` function returns the name of the property itself as string; for the properties specified by the OpenMP specification, the name matches the name of the named constant with the ‘`omp_ipr_`’ prefix removed.

*C/C++:*

*Prototype:*        `const char *omp_get_interop_name(const omp_interop_  
t interop, omp_interop_property_t property_id)`

*Fortran:*

*Interface:*        `character(:) function omp_get_interop_  
                         name(interop, property_id)  
                         pointer :: omp_get_interop_name  
                         integer(omp_interop_kind), intent(in) :: interop  
                         integer(omp_interop_property_kind) property_id`

*See also:*    Section 3.11.1 [omp\_get\_num\_interop\_properties], page 46, Section 3.11.6 [omp\_get\_interop\_type\_desc], page 49,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.12.5, OpenMP specification v6.0 (<https://www.openmp.org>), Section 26.5

### 3.11.6 omp\_get\_interop\_type\_desc – Obtain type and description to an interop\_property

*Description:*

The `omp_get_interop_type_desc` function returns a string that describes in human-readable form the data type associated with the *property\_id* interoperability property of the passed *interop* object.

In GCC, this function returns the name of the C/C++ data type for this property or ‘N/A’ if this property is not available for the given foreign runtime. If *interop* is `omp_interop_none` or for invalid property values, a null pointer is returned. The effect of running this routine in a `target` region that is not the initial device is unspecified.

*C/C++:*

*Prototype:*        `const char *omp_get_interop_type_desc(const  
                         omp_interop_t interop, omp_interop_property_t  
                         property_id)`

*Fortran:*

*Interface:*        `character(:) function omp_get_interop_type_  
                         desc(interop, property_id)  
                         pointer :: omp_get_interop_type_desc  
                         integer(omp_interop_kind), intent(in) :: interop  
                         integer(omp_interop_property_kind) property_id`

*See also:*    Section 3.11.1 [omp\_get\_num\_interop\_properties], page 46, Section 3.11.5 [omp\_get\_interop\_name], page 48, Chapter 12 [Offload-Target Specifics], page 111,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.12.6, OpenMP specification v6.0 (<https://www.openmp.org>), Section 26.6

### 3.11.7 omp\_get\_interop\_rc\_desc – Obtain error string to an interop\_rc error code

*Description:*

The `omp_get_interop_rc_desc` function returns a string value describing the *ret\_code* in human-readable form.

The behavior is unspecified if value of *ret\_code* was not set by an interoperability routine invoked for *interop*.

GCC implements the OpenMP 6.0 version of this function for C and C++, which is not compatible with its type signature in previous versions of the OpenMP specification. In older versions, the type `int` was used for the *ret\_code* argument in place of the enumerated type `omp_interop_rc_t`.

*C/C++:*

*Prototype:*        `const char *omp_get_interop_rc_desc(const  
omp_interop_t interop, omp_interop_rc_t ret_code)`

*Fortran:*

*Interface:*        `character(:) function omp_get_interop_rc_  
desc(interop, property_id, ret_code)  
pointer :: omp_get_interop_rc_desc  
integer(omp_interop_kind), intent(in) :: interop  
integer (omp_interop_rc_kind) ret_code`

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.12.7,  
OpenMP specification v6.0 (<https://www.openmp.org>), Section 26.7

## 3.12 Memory Management Routines

Routines to manage and allocate memory on the current device. They have C linkage and do not throw exceptions.

### 3.12.1 `omp_init_allocator` – Create an allocator

*Description:*

Create an allocator that uses the specified memory space and has the specified traits; if an allocator that fulfills the requirements cannot be created, `omp_null_allocator` is returned.

The predefined memory spaces and available traits can be found at Section 11.3 [Memory allocation], page 107, where the trait names have to be prefixed by `omp_atk_` (e.g. `omp_atk_pinned`) and the named trait values by `omp_atv_` (e.g. `omp_atv_true`); additionally, `omp_atv_default` may be used as trait value to specify that the default value should be used.

*C/C++:*

*Prototype:*        `omp_allocator_handle_t omp_init_allocator(  
omp_memspace_handle_t memspace,  
int ntraits,  
const omp_alloctrait_t traits[]);`

*Fortran:*

*Interface:*        `function omp_init_allocator(memspace, ntraits,  
traits)  
integer (omp_allocator_handle_kind) :: omp_init_  
allocator`

```
integer (omp_memspace_handle_kind), intent(in) ::
memspace
integer, intent(in) :: ntraits
type (omp_allocotrait), intent(in) :: traits(*)
```

*See also:* Section 11.3 [Memory allocation], page 107, Section 4.1 [OMP\_ALLOCATOR], page 59, Section 3.12.2 [omp\_destroy\_allocator], page 51,

*Reference:* OpenMP specification v5.0 (<https://www.openmp.org>), Section 3.7.2

### 3.12.2 omp\_destroy\_allocator – Destroy an allocator

*Description:*

Releases all resources used by a memory allocator, which must not represent a predefined memory allocator. Accessing memory after its allocator has been destroyed has unspecified behavior. Passing `omp_null_allocator` to the routine is permitted but has no effect.

*C/C++:*

*Prototype:*        `void omp_destroy_allocator (omp_allocator_handle_t allocator);`

*Fortran:*

*Interface:*        `subroutine omp_destroy_allocator(allocator)`  
                     `integer (omp_allocator_handle_kind), intent(in) ::`  
                     `allocator`

*See also:* Section 3.12.1 [omp\_init\_allocator], page 50,

*Reference:* OpenMP specification v5.0 (<https://www.openmp.org>), Section 3.7.3

### 3.12.3 omp\_set\_default\_allocator – Set the default allocator

*Description:*

Sets the default allocator that is used when no allocator has been specified in the `allocate` or `allocator` clause or if an OpenMP memory routine is invoked with the `omp_null_allocator` allocator.

*C/C++:*

*Prototype:*        `void omp_set_default_allocator(omp_allocator_handle_t allocator);`

*Fortran:*

*Interface:*        `subroutine omp_set_default_allocator(allocator)`  
                     `integer (omp_allocator_handle_kind), intent(in) ::`  
                     `allocator`

*See also:* Section 3.12.4 [omp\_get\_default\_allocator], page 52, Section 3.12.1 [omp\_init\_allocator], page 50, Section 4.1 [OMP\_ALLOCATOR], page 59, Section 11.3 [Memory allocation], page 107,

*Reference:* OpenMP specification v5.0 (<https://www.openmp.org>), Section 3.7.4

### 3.12.4 `omp_get_default_allocator` – Get the default allocator

*Description:*

The routine returns the default allocator that is used when no allocator has been specified in the `allocate` or `allocator` clause or if an OpenMP memory routine is invoked with the `omp_null_allocator` allocator.

*C/C++:*

*Prototype:*            `omp_allocator_handle_t omp_get_default_allocator();`

*Fortran:*

*Interface:*            `function omp_get_default_allocator()  
integer (omp_allocator_handle_kind) :: omp_get_default_allocator`

*See also:* Section 3.12.3 [`omp_set_default_allocator`], page 51, Section 4.1 [`OMP_ALLOCATOR`], page 59,

*Reference:* OpenMP specification v5.0 (<https://www.openmp.org>), Section 3.7.5

### 3.12.5 `omp_alloc` – Memory allocation with an allocator

*Description:*

Allocate memory with the specified allocator, which can either be a predefined allocator, an allocator handle or `omp_null_allocator`. If the allocators is `omp_null_allocator`, the allocator specified by the *def-allocator-var* ICV is used. *size* must be a nonnegative number denoting the number of bytes to be allocated; if *size* is zero, `omp_alloc` will return a null pointer. If successful, a pointer to the allocated memory is returned, otherwise the *fallback* trait of the allocator determines the behavior. The content of the allocated memory is unspecified.

In *target* regions, either the `dynamic_allocators` clause must appear on a *requires* directive in the same compilation unit – or the *allocator* argument may only be a constant expression with the value of one of the predefined allocators and may not be `omp_null_allocator`.

Memory allocated by `omp_alloc` must be freed using `omp_free`.

*C:*

*Prototype:*            `void* omp_alloc(size_t size,  
omp_allocator_handle_t allocator)`

*C++:*

*Prototype:*            `void* omp_alloc(size_t size,  
omp_allocator_handle_t allocator=omp_null_allocator)`

*Fortran:*

*Interface:*            `type(c_ptr) function omp_alloc(size, allocator)  
bind(C)`

```

use, intrinsic :: iso_c_binding, only : c_ptr,
c_size_t
integer (c_size_t), value :: size
integer (omp_allocator_handle_kind), value ::
allocator

```

*See also:* Section 4.1 [OMP\_ALLOCATOR], page 59, Section 11.3 [Memory allocation], page 107, Section 3.12.3 [omp\_set\_default\_allocator], page 51, Section 3.12.7 [omp\_free], page 54, Section 3.12.1 [omp\_init\_allocator], page 50,

*Reference:* OpenMP specification v5.0 (<https://www.openmp.org>), Section 3.7.6

### 3.12.6 omp\_aligned\_alloc – Memory allocation with an allocator and alignment

*Description:*

Allocate memory with the specified allocator, which can either be a predefined allocator, an allocator handle or `omp_null_allocator`. If the allocators is `omp_null_allocator`, the allocator specified by the *def-allocator-var* ICV is used. *alignment* must be a positive power of two and *size* must be a nonnegative number that is a multiple of the alignment and denotes the number of bytes to be allocated; if *size* is zero, `omp_aligned_alloc` will return a null pointer. The alignment will be at least the maximal value required by *alignment* trait of the allocator and the value of the passed *alignment* argument. If successful, a pointer to the allocated memory is returned, otherwise the *fallback* trait of the allocator determines the behavior. The content of the allocated memory is unspecified.

In *target* regions, either the *dynamic\_allocators* clause must appear on a *requires* directive in the same compilation unit – or the *allocator* argument may only be a constant expression with the value of one of the predefined allocators and may not be `omp_null_allocator`.

Memory allocated by `omp_aligned_alloc` must be freed using `omp_free`.

*C:*

*Prototype:*

```

void* omp_aligned_alloc(size_t alignment,
size_t size,
omp_allocator_handle_t allocator)

```

*C++:*

*Prototype:*

```

void* omp_aligned_alloc(size_t alignment,
size_t size,
omp_allocator_handle_t allocator=omp_null_
allocator)

```

*Fortran:*

*Interface:*

```

type(c_ptr) function omp_aligned_alloc(alignment,
size, allocator) bind(C)
use, intrinsic :: iso_c_binding, only : c_ptr,
c_size_t

```

```
integer (c_size_t), value :: alignment, size
integer (omp_allocator_handle_kind), value ::
allocator
```

*See also:* Section 4.1 [OMP\_ALLOCATOR], page 59, Section 11.3 [Memory allocation], page 107, Section 3.12.3 [omp\_set\_default\_allocator], page 51, Section 3.12.7 [omp\_free], page 54, Section 3.12.1 [omp\_init\_allocator], page 50,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.13.6

### 3.12.7 omp\_free – Freeing memory allocated with OpenMP routines

*Description:*

The `omp_free` routine deallocates memory previously allocated by an OpenMP memory-management routine. The *ptr* argument must point to such memory or be a null pointer; if it is a null pointer, no operation is performed. If specified, the *allocator* argument must be either the memory allocator that was used for the allocation or `omp_null_allocator`; if it is `omp_null_allocator`, the implementation will determine the value automatically.

Calling `omp_free` invokes undefined behavior if the memory was already deallocated or when the used allocator has already been destroyed.

*C:*

```
Prototype:      void omp_free(void *ptr,
                           omp_allocator_handle_t allocator)
```

*C++:*

```
Prototype:      void omp_free(void *ptr,
                           omp_allocator_handle_t allocator=omp_null_
                           allocator)
```

*Fortran:*

```
Interface:      subroutine omp_free(ptr, allocator) bind(C)
                  use, intrinsic :: iso_c_binding, only : c_ptr
                  type (c_ptr), value :: ptr
                  integer (omp_allocator_handle_kind), value ::
                  allocator
```

*See also:* Section 3.12.5 [omp\_alloc], page 52, Section 3.12.6 [omp\_aligned\_alloc], page 53, Section 3.12.8 [omp\_calloc], page 54, Section 3.12.9 [omp\_aligned\_calloc], page 55, Section 3.12.10 [omp\_realloc], page 56,

*Reference:* OpenMP specification v5.0 (<https://www.openmp.org>), Section 3.7.7

### 3.12.8 omp\_calloc – Allocate nullified memory with an allocator

*Description:*

Allocate zero-initialized memory with the specified allocator, which can either be a predefined allocator, an allocator handle or `omp_null_allocator`. If the allocators is `omp_null_allocator`, the allocator specified by the *def-allocator-var* ICV is used. The to-be allocated memory is for an array with *nmemb*

elements, each having a size of *size* bytes. Both *nmemb* and *size* must be nonnegative numbers; if either of them is zero, `omp_calloc` will return a null pointer. If successful, a pointer to the zero-initialized allocated memory is returned, otherwise the `fallback` trait of the allocator determines the behavior. In `target` regions, either the `dynamic_allocators` clause must appear on a `requires` directive in the same compilation unit – or the *allocator* argument may only be a constant expression with the value of one of the predefined allocators and may not be `omp_null_allocator`.

Memory allocated by `omp_calloc` must be freed using `omp_free`.

*C:*

*Prototype:*        `void* omp_calloc(size_t nmemb, size_t size,  
                         omp_allocator_handle_t allocator)`

*C++:*

*Prototype:*        `void* omp_calloc(size_t nmemb, size_t size,  
                         omp_allocator_handle_t allocator=omp_null_  
                         allocator)`

*Fortran:*

*Interface:*        `type(c_ptr) function omp_calloc(nmemb, size,  
                         allocator) bind(C)  
                         use, intrinsic :: iso_c_binding, only : c_ptr,  
                         c_size_t  
                         integer (c_size_t), value :: nmemb, size  
                         integer (omp_allocator_handle_kind), value ::  
                         allocator`

*See also:*    Section 4.1 [OMP\_ALLOCATOR], page 59, Section 11.3 [Memory allocation], page 107, Section 3.12.3 [omp\_set\_default\_allocator], page 51, Section 3.12.7 [omp\_free], page 54, Section 3.12.1 [omp\_init\_allocator], page 50,

*Reference:*   OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.13.8

### 3.12.9 `omp_aligned_calloc` – Allocate aligned nullified memory with an allocator

*Description:*

Allocate zero-initialized memory with the specified allocator, which can either be a predefined allocator, an allocator handle or `omp_null_allocator`. If the allocators is `omp_null_allocator`, the allocator specified by the *def-allocator-var* ICV is used. The to-be allocated memory is for an array with *nmemb* elements, each having a size of *size* bytes. Both *nmemb* and *size* must be non-negative numbers; if either of them is zero, `omp_aligned_calloc` will return a null pointer. *alignment* must be a positive power of two and *size* must be a multiple of the alignment; the alignment will be at least the maximal value required by `alignment` trait of the allocator and the value of the passed *alignment* argument. If successful, a pointer to the zero-initialized allocated memory is returned, otherwise the `fallback` trait of the allocator determines the behavior.

In **target** regions, either the **dynamic\_allocators** clause must appear on a **requires** directive in the same compilation unit – or the *allocator* argument may only be a constant expression with the value of one of the predefined allocators and may not be **omp\_null\_allocator**.

Memory allocated by **omp\_aligned\_calloc** must be freed using **omp\_free**.

*C:*

*Prototype:*        `void* omp_aligned_calloc(size_t nmemb, size_t size,  
                         omp_allocator_handle_t allocator)`

*C++:*

*Prototype:*        `void* omp_aligned_calloc(size_t nmemb, size_t size,  
                         omp_allocator_handle_t allocator=omp_null_  
                         allocator)`

*Fortran:*

*Interface:*        `type(c_ptr) function omp_aligned_calloc(nmemb,  
                         size, allocator) bind(C)  
                         use, intrinsic :: iso_c_binding, only : c_ptr,  
                         c_size_t  
                         integer (c_size_t), value :: nmemb, size  
                         integer (omp_allocator_handle_kind), value ::  
                         allocator`

*See also:*    Section 4.1 [OMP\_ALLOCATOR], page 59, Section 11.3 [Memory allocation], page 107, Section 3.12.3 [omp\_set\_default\_allocator], page 51, Section 3.12.7 [omp\_free], page 54, Section 3.12.1 [omp\_init\_allocator], page 50,

*Reference:*    OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.13.8

### 3.12.10 **omp\_realloc** – Reallocate memory allocated with OpenMP routines

*Description:*

The **omp\_realloc** routine deallocates memory to which *ptr* points to and allocates new memory with the specified *allocator* argument; the new memory will have the content of the old memory up to the minimum of the old size and the new *size*, otherwise the content of the returned memory is unspecified. If the new allocator is the same as the old one, the routine tries to resize the existing memory allocation, returning the same address as *ptr* if successful. *ptr* must point to memory allocated by an OpenMP memory-management routine.

The *allocator* and *free\_allocator* arguments must be a predefined allocator, an allocator handle or **omp\_null\_allocator**. If *free\_allocator* is **omp\_null\_allocator**, the implementation automatically determines the allocator used for the allocation of *ptr*. If *allocator* is **omp\_null\_allocator** and *ptr* is not a null pointer, the same allocator as *free\_allocator* is used and when *ptr* is a null pointer the allocator specified by the *def-allocator-var* ICV is used.

The *size* must be a nonnegative number denoting the number of bytes to be allocated; if *size* is zero, **omp\_realloc** will return free the memory and return

a null pointer. When *size* is nonzero: if successful, a pointer to the allocated memory is returned, otherwise the `fallback` trait of the allocator determines the behavior.

In `target` regions, either the `dynamic_allocators` clause must appear on a `requires` directive in the same compilation unit – or the `free_allocator` and `allocator` arguments may only be a constant expression with the value of one of the predefined allocators and may not be `omp_null_allocator`.

Memory allocated by `omp_realloc` must be freed using `omp_free`. Calling `omp_free` invokes undefined behavior if the memory was already deallocated or when the used allocator has already been destroyed.

*C:*

*Prototype:*        `void* omp_realloc(void *ptr, size_t size,  
                          omp_allocator_handle_t allocator,  
                          omp_allocator_handle_t free_allocator)`

*C++:*

*Prototype:*        `void* omp_realloc(void *ptr, size_t size,  
                          omp_allocator_handle_t allocator=omp_null_  
                          allocator,  
                          omp_allocator_handle_t free_allocator=omp_null_  
                          allocator)`

*Fortran:*

*Interface:*        `type(c_ptr) function omp_realloc(ptr, size,  
                          allocator, free_allocator) bind(C)  
                          use, intrinsic :: iso_c_binding, only : c_ptr,  
                          c_size_t  
                          type(C_ptr), value :: ptr  
                          integer (c_size_t), value :: size  
                          integer (omp_allocator_handle_kind), value ::  
                          allocator, free_allocator`

*See also:*    Section 4.1 [OMP\_ALLOCATOR], page 59, Section 11.3 [Memory allocation], page 107, Section 3.12.3 [omp\_set\_default\_allocator], page 51, Section 3.12.7 [omp\_free], page 54, Section 3.12.1 [omp\_init\_allocator], page 50,

*Reference:*    OpenMP specification v5.0 (<https://www.openmp.org>), Section 3.7.9

## 3.13 Environment Display Routine

Routine to display the OpenMP version number and the initial value of ICVs. It has C linkage and does not throw exceptions.

### 3.13.1 omp\_display\_env – print the initial ICV values

*Description:*

Each time this routine is invoked, the OpenMP version number and initial value of internal control variables (ICVs) is printed on `stderr`. The displayed values

are those at startup after evaluating the environment variables; later calls to API routines or clauses used in enclosing constructs do not affect the output.

If the *verbose* argument is `false`, only the OpenMP version and standard OpenMP ICVs are shown; if it is `true`, additionally, the GCC-specific ICVs are shown.

The output consists of multiple lines and starts with ‘OPENMP DISPLAY ENVIRONMENT BEGIN’ followed by the name-value lines and ends with ‘OPENMP DISPLAY ENVIRONMENT END’. The *name* is followed by an equal sign and the *value* is enclosed in single quotes.

The first line has as *name* either ‘\_OPENMP’ or ‘openmp\_version’ and shows as value the supported OpenMP version number (4-digit year, 2-digit month) of the implementation, matching the value of the `_OPENMP` macro and, in Fortran, the named constant `openmp_version`.

In each of the succeeding lines, the *name* matches the environment-variable name of an ICV and shows its value. Those line are might be prefixed by pair of brackets and a space, where the brackets enclose a comma-separated list of devices to which the ICV-value combination applies to; the value can either be a numeric device number or an abstract name denoting all devices (`all`), the initial host device (`host`) or all devices but the host (`device`). Note that the same ICV might be printed multiple times for multiple devices, even if all have the same value.

The effect when invoked from within a `target` region is unspecified.

*C/C++:*

*Prototype:*            `void omp_display_env(int verbose)`

*Fortran:*

*Interface:*           `subroutine omp_display_env(verbose)`  
                 `logical, intent(in) :: verbose`

*Example:*    Note that the GCC-specific ICVs, such as the shown `GOMP_SPINCOUNT`, are only printed when *verbose* set to `true`.

```
OPENMP DISPLAY ENVIRONMENT BEGIN
  _OPENMP = '201511'
  [host] OMP_DYNAMIC = 'FALSE'
  [host] OMP_NESTED = 'FALSE'
  [all] OMP_CANCELLATION = 'FALSE'
  ...
  [host] GOMP_SPINCOUNT = '300000'
OPENMP DISPLAY ENVIRONMENT END
```

*See also:*    Section 4.5 [OMP\_DISPLAY\_ENV], page 61, Chapter 4 [Environment Variables], page 59, Section 11.1 [Implementation-defined ICV Initialization], page 107,

*Reference:*   OpenMP specification v5.1 (<https://www.openmp.org>), Section 3.15

## 4 OpenMP Environment Variables

The environment variables which beginning with `OMP_` are defined by section 4 of the OpenMP specification in version 4.5 or in a later version of the specification, while those beginning with `GOMP_` are GNU extensions. Most `OMP_` environment variables have an associated internal control variable (ICV).

For any OpenMP environment variable that sets an ICV and is neither `OMP_DEFAULT_DEVICE` nor has global ICV scope, associated device-specific environment variables exist. For them, the environment variable without suffix affects the host. The suffix `_DEV_` followed by a non-negative device number less than the number of available devices sets the ICV for the corresponding device. The suffix `_DEV` sets the ICV of all non-host devices for which a device-specific corresponding environment variable has not been set while the `_ALL` suffix sets the ICV of all host and non-host devices for which a more specific corresponding environment variable is not set.

### 4.1 `OMP_ALLOCATOR` – Set the default allocator

*ICV:* `def-allocator-var`

*Scope:* data environment

*Description:*

Sets the default allocator that is used when no allocator has been specified in the `allocate` or `allocator` clause or if an OpenMP memory routine is invoked with the `omp_null_allocator` allocator. If unset, `omp_default_mem_alloc` is used.

The value can either be a predefined allocator or a predefined memory space or a predefined memory space followed by a colon and a comma-separated list of memory trait and value pairs, separated by `=`.

See Section 11.3 [Memory allocation], page 107, for a list of supported predefined allocators, memory spaces, and traits.

Note: The corresponding device environment variables are currently not supported. Therefore, the non-host *def-allocator-var* ICVs are always initialized to `omp_default_mem_alloc`. However, on all devices, the `omp_set_default_allocator` API routine can be used to change value.

Examples:

```
OMP_ALLOCATOR=omp_high_bw_mem_alloc
OMP_ALLOCATOR=omp_large_cap_mem_space
OMP_ALLOCATOR=omp_low_lat_mem_space:pinned=true,partition=nearest
```

*See also:* Section 11.3 [Memory allocation], page 107, Section 3.12.4 [`omp_get_default_allocator`], page 52, Section 3.12.3 [`omp_set_default_allocator`], page 51, Chapter 12 [Offload-Target Specifics], page 111,

*Reference:* OpenMP specification v5.0 (<https://www.openmp.org>), Section 6.21

## 4.2 OMP\_AFFINITY\_FORMAT – Set the format string used for affinity display

*ICV: affinity-format-var*

*Scope: device*

*Description:*

Sets the format string used when displaying OpenMP thread affinity information. Special values are output using % followed by an optional size specification and then either the single-character field type or its long name enclosed in curly braces; using %% displays a literal percent. The size specification consists of an optional 0. or . followed by a positive integer, specifying the minimal width of the output. With 0. and numerical values, the output is padded with zeros on the left; with ., the output is padded by spaces on the left; otherwise, the output is padded by spaces on the right. If unset, the value is “level %L thread %i affinity %A”.

Supported field types are:

t	team_num	value returned by <code>omp_get_team_num</code>
T	num_teams	value returned by <code>omp_get_num_teams</code>
L	nesting_level	value returned by <code>omp_get_level</code>
n	thread_num	value returned by <code>omp_get_thread_num</code>
N	num_threads	value returned by <code>omp_get_num_threads</code>
a	ancestor_tnum	value returned by <code>omp_get_ancestor_thread_num(omp_get_level()-1)</code>
H	host	name of the host that executes the thread
P	process_id	process identifier
i	native_thread_id	native thread identifier
A	thread_affinity	comma separated list of integer values or ranges, representing the processors on which a process might execute, subject to affinity mechanisms

For instance, after setting

```
OMP_AFFINITY_FORMAT="%0.2a!%n!%.4L!%N;%.2t;%0.2T;{%team_num};{%num_teams};%A"■
```

with either `OMP_DISPLAY_AFFINITY` being set or when calling `omp_display_affinity` with NULL or an empty string, the program might display the following:

```
00!0!  1!4; 0;01;0;1;0-11
00!3!  1!4; 0;01;0;1;0-11
00!2!  1!4; 0;01;0;1;0-11
00!1!  1!4; 0;01;0;1;0-11
```

*See also:* Section 4.4 [`OMP_DISPLAY_AFFINITY`], page 61,

*Reference:* OpenMP specification v5.0 (<https://www.openmp.org>), Section 6.14

### 4.3 OMP\_CANCELLATION – Set whether cancellation is activated

*ICV:* `cancel-var`

*Scope:* global

*Description:*

If set to **TRUE**, the cancellation is activated. If set to **FALSE** or if unset, cancellation is disabled and the `cancel` construct is ignored.

*See also:* Section 3.1.8 [`omp_get_cancellation`], page 17,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 4.11

### 4.4 OMP\_DISPLAY\_AFFINITY – Display thread affinity information

*ICV:* `display-affinity-var`

*Scope:* global

*Description:*

If set to **FALSE** or if unset, affinity displaying is disabled. If set to **TRUE**, the runtime displays affinity information about OpenMP threads in a parallel region upon entering the region and every time any change occurs.

*See also:* Section 4.2 [`OMP_AFFINITY_FORMAT`], page 60,

*Reference:* OpenMP specification v5.0 (<https://www.openmp.org>), Section 6.13

### 4.5 OMP\_DISPLAY\_ENV – Show OpenMP version and environment variables

*ICV:* none

*Scope:* not applicable

*Description:*

If set to **TRUE**, the runtime displays the same information to `stderr` as shown by the `omp_display_env` routine invoked with `verbose` argument set to **false**. If set to **VERBOSE**, the same information is shown as invoking the routine with `verbose` set to **true**. If unset or set to **FALSE**, this information is not shown. The result for any other value is unspecified.

*See also:* Section 3.13.1 [`omp_display_env`], page 57,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 4.12

### 4.6 OMP\_DEFAULT\_DEVICE – Set the device used in target regions

*ICV:* `default-device-var`

*Scope:* data environment

*Description:*

Set to choose the device which is used in a **target** region, unless the value is overridden by `omp_set_default_device` or by a `device` clause. The value shall

be the nonnegative device number. If no device with the given device number exists, the code is executed on the host. If unset, `OMP_TARGET_OFFLOAD` is **mandatory** and no non-host devices are available, it is set to `omp_invalid_device`. Otherwise, if unset, device number 0 is used.

*See also:* Section 3.6.3 [omp-get-default-device], page 27, Section 3.6.2 [omp-set-default-device], page 27, Section 4.17 [OMP\_TARGET\_OFFLOAD], page 66,

*Reference:* OpenMP specification v5.2 (<https://www.openmp.org>), Section 21.2.7

## 4.7 OMP\_DYNAMIC – Dynamic adjustment of threads

*ICV:* dyn-var

*Scope:* global

*Description:*

Enable or disable the dynamic adjustment of the number of threads within a team. The value of this environment variable shall be **TRUE** or **FALSE**. If undefined, dynamic adjustment is disabled by default.

*See also:* Section 3.1.6 [omp-set-dynamic], page 17,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 4.3

## 4.8 OMP\_MAX\_ACTIVE\_LEVELS – Set the maximum number of nested parallel regions

*ICV:* max-active-levels-var

*Scope:* data environment

*Description:*

Specifies the initial value for the maximum number of nested parallel regions. The value of this variable shall be a positive integer. If undefined, then if `OMP_NESTED` is defined and set to true, or if `OMP_NUM_THREADS` or `OMP_PROC_BIND` are defined and set to a list with more than one item, the maximum number of nested parallel regions is initialized to the largest number supported, otherwise it is set to one.

*See also:* Section 3.1.15 [omp-set-max-active-levels], page 20, Section 4.10 [OMP\_NESTED], page 63, Section 4.13 [OMP\_PROC\_BIND], page 64, Section 4.12 [OMP\_NUM\_THREADS], page 63,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 4.9

## 4.9 OMP\_MAX\_TASK\_PRIORITY – Set the maximum priority

number that can be set for a task.

*ICV:* max-task-priority-var

*Scope:* global

*Description:*

Specifies the initial value for the maximum priority value that can be set for a task. The value of this variable shall be a non-negative integer, and zero is allowed. If undefined, the default priority is 0.

*See also:* Section 3.4.1 [omp-get-max-task-priority], page 25,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 4.14

## 4.10 OMP\_NESTED – Nested parallel regions

*ICV:* max-active-levels-var

*Scope:* data environment

*Description:*

Enable or disable nested parallel regions, i.e., whether team members are allowed to create new teams. The value of this environment variable shall be **TRUE** or **FALSE**. If set to **TRUE**, the number of maximum active nested regions supported is by default set to the maximum supported, otherwise it is set to one. If **OMP\_MAX\_ACTIVE\_LEVELS** is defined, its setting overrides this setting. If both are undefined, nested parallel regions are enabled if **OMP\_NUM\_THREADS** or **OMP\_PROC\_BINDS** are defined to a list with more than one item, otherwise they are disabled by default.

Note that the **OMP\_NESTED** environment variable was deprecated in the OpenMP specification 5.0 in favor of **OMP\_MAX\_ACTIVE\_LEVELS**.

*See also:* Section 3.1.15 [omp-set-max-active-levels], page 20, Section 3.1.9 [omp-set-nested], page 18, Section 4.8 [OMP\_MAX\_ACTIVE\_LEVELS], page 62,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 4.6

## 4.11 OMP\_NUM\_TEAMS – Specifies the number of teams to use by teams region

*ICV:* nteams-var

*Scope:* device

*Description:*

Specifies the upper bound for number of teams to use in teams regions without explicit **num\_teams** clause. The value of this variable shall be a positive integer. If undefined it defaults to 0 which means implementation defined upper bound.

*See also:* Section 3.3.3 [omp-set-num-teams], page 23,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 6.23

## 4.12 OMP\_NUM\_THREADS – Specifies the number of threads to use

*ICV:* nthreads-var

*Scope:* data environment

*Description:*

Specifies the default number of threads to use in parallel regions. The value of this variable shall be a comma-separated list of positive integers; the value specifies the number of threads to use for the corresponding nested level. Specifying more than one item in the list automatically enables nesting by default. If undefined one thread per CPU is used.

When a list with more than value is specified, it also affects the *max-active-levels-var* ICV as described in Section 4.8 [OMP\_MAX\_ACTIVE\_LEVELS], page 62.

*See also:* Section 3.1.1 [omp\_set\_num\_threads], page 15, Section 4.8 [OMP\_MAX\_ACTIVE\_LEVELS], page 62,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 4.2

### 4.13 OMP\_PROC\_BIND – Whether threads may be moved between CPUs

*ICV:* bind-var

*Scope:* data environment

*Description:*

Specifies whether threads may be moved between processors. If set to **TRUE**, OpenMP threads should not be moved; if set to **FALSE** they may be moved. Alternatively, a comma separated list with the values **PRIMARY**, **MASTER**, **CLOSE** and **SPREAD** can be used to specify the thread affinity policy for the corresponding nesting level. With **PRIMARY** and **MASTER** the worker threads are in the same place partition as the primary thread. With **CLOSE** those are kept close to the primary thread in contiguous place partitions. And with **SPREAD** a sparse distribution across the place partitions is used. Specifying more than one item in the list automatically enables nesting by default.

When a list is specified, it also affects the *max-active-levels-var* ICV as described in Section 4.8 [OMP\_MAX\_ACTIVE\_LEVELS], page 62.

When undefined, **OMP\_PROC\_BIND** defaults to **TRUE** when **OMP\_PLACES** or **GOMP\_CPU\_AFFINITY** is set and **FALSE** otherwise.

*See also:* Section 3.2.1 [omp\_get\_proc\_bind], page 22, Section 4.21 [GOMP\_CPU\_AFFINITY], page 67, Section 4.14 [OMP\_PLACES], page 64, Section 4.8 [OMP\_MAX\_ACTIVE\_LEVELS], page 62,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 4.4

### 4.14 OMP\_PLACES – Specifies on which CPUs the threads should be placed

*ICV:* place-partition-var

*Scope:* implicit tasks

*Description:*

The thread placement can be either specified using an abstract name or by an explicit list of the places. The abstract names **threads**, **cores**, **sockets**, **ll\_caches** and **numa\_domains** can be optionally followed by a positive number in parentheses, which denotes the how many places shall be created. With **threads** each place corresponds to a single hardware thread; **cores** to a single core with the corresponding number of hardware threads; with **sockets** the place corresponds to a single socket; with **ll\_caches** to a set of cores that shares the last level cache on the device; and **numa\_domains** to a set of cores for

which their closest memory on the device is the same memory and at a similar distance from the cores. The resulting placement can be shown by setting the `OMP_DISPLAY_ENV` environment variable.

Alternatively, the placement can be specified explicitly as comma-separated list of places. A place is specified by set of nonnegative numbers in curly braces, denoting the hardware threads. The curly braces can be omitted when only a single number has been specified. The hardware threads belonging to a place can either be specified as comma-separated list of nonnegative thread numbers or using an interval. Multiple places can also be either specified by a comma-separated list of places or by an interval. To specify an interval, a colon followed by the count is placed after the hardware thread number or the place. Optionally, the length can be followed by a colon and the stride number – otherwise a unit stride is assumed. Placing an exclamation mark (!) directly before a curly brace or numbers inside the curly braces (excluding intervals) excludes those hardware threads.

For instance, the following specifies the same places list: "{0,1,2}, {3,4,6}, {7,8,9}, {10,11,12}"; "{0:3}, {3:3}, {7:3}, {10:3}"; and "{0:2}:4:3".

If `OMP_PLACES` and `GOMP_CPU_AFFINITY` are unset and `OMP_PROC_BIND` is either unset or `false`, threads may be moved between CPUs following no placement policy.

*See also:* Section 4.13 [`OMP_PROC_BIND`], page 64, Section 4.21 [`GOMP_CPU_AFFINITY`], page 67, Section 3.2.1 [`omp_get_proc_bind`], page 22, Section 4.5 [`OMP_DISPLAY_ENV`], page 61,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 4.5

## 4.15 `OMP_STACKSIZE` – Set default thread stack size

*ICV:* `stacksize-var`

*Scope:* device

*Description:*

Set the default thread stack size in kilobytes, unless the number is suffixed by B, K, M or G, in which case the size is, respectively, in bytes, kilobytes, megabytes or gigabytes. This is different from `pthread_attr_setstacksize` which gets the number of bytes as an argument. If the stack size cannot be set due to system constraints, an error is reported and the initial stack size is left unchanged. If undefined, the stack size is system dependent.

*See also:* Section 4.23 [`GOMP_STACKSIZE`], page 68,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 4.7

## 4.16 `OMP_SCHEDULE` – How threads are scheduled

*ICV:* `run-sched-var`

*Scope:* data environment

*Description:*

Allows to specify `schedule type` and `chunk size`. The value of the variable shall have the form: `type[,chunk]` where `type` is one of `static`, `dynamic`,

**guided** or **auto** The optional **chunk** size shall be a positive integer. If undefined, dynamic scheduling and a chunk size of 1 is used.

*See also:* Section 3.1.11 [omp\_set\_schedule], page 19,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Sections 2.7.1.1 and 4.1

## 4.17 OMP\_TARGET\_OFFLOAD – Controls offloading behavior

*ICV:* target-offload-var

*Scope:* global

*Description:*

Specifies the behavior with regard to offloading code to a device. This variable can be set to one of three values - **MANDATORY**, **DISABLED** or **DEFAULT**.

If set to **MANDATORY**, the program terminates with an error if any device construct or device memory routine uses a device that is unavailable or not supported by the implementation, or uses a non-conforming device number. If set to **DISABLED**, then offloading is disabled and all code runs on the host. If set to **DEFAULT**, the program tries offloading to the device first, then falls back to running code on the host if it cannot.

If undefined, then the program behaves as if **DEFAULT** was set.

Note: Even with **MANDATORY**, no run-time termination is performed when the device number in a **device** clause or argument to a device memory routine is for host, which includes using the device number in the *default-device-var* ICV. However, the initial value of the *default-device-var* ICV is affected by **MANDATORY**.

*See also:* Section 4.6 [OMP\_DEFAULT\_DEVICE], page 61,

*Reference:* OpenMP specification v5.2 (<https://www.openmp.org>), Section 21.2.8

## 4.18 OMP\_TEAMS\_THREAD\_LIMIT – Set the maximum number of threads imposed by teams

*ICV:* teams-thread-limit-var

*Scope:* device

*Description:*

Specifies an upper bound for the number of threads to use by each contention group created by a teams construct without explicit **thread\_limit** clause. The value of this variable shall be a positive integer. If undefined, the value of 0 is used which stands for an implementation defined upper limit.

*See also:* Section 4.19 [OMP\_THREAD\_LIMIT], page 67, Section 3.3.5 [omp\_set\_teams\_thread\_limit], page 24,

*Reference:* OpenMP specification v5.1 (<https://www.openmp.org>), Section 6.24

## 4.19 OMP\_THREAD\_LIMIT – Set the maximum number of threads

*ICV:* thread-limit-var

*Scope:* data environment

*Description:*

Specifies the number of threads to use for the whole program. The value of this variable shall be a positive integer. If undefined, the number of threads is not limited.

*See also:* Section 4.12 [OMP\_NUM\_THREADS], page 63, Section 3.3.6 [omp\_get\_thread\_limit], page 24,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 4.10

## 4.20 OMP\_WAIT\_POLICY – How waiting threads are handled

*Description:*

Specifies whether waiting threads should be active or passive. If the value is **PASSIVE**, waiting threads should not consume CPU power while waiting; while the value is **ACTIVE** specifies that they should. If undefined, threads wait actively for a short time before waiting passively.

*See also:* Section 4.24 [GOMP\_SPINCOUNT], page 68,

*Reference:* OpenMP specification v4.5 (<https://www.openmp.org>), Section 4.8

## 4.21 GOMP\_CPU\_AFFINITY – Bind threads to specific CPUs

*Description:*

Binds threads to specific CPUs. The variable should contain a space-separated or comma-separated list of CPUs. This list may contain different kinds of entries: either single CPU numbers in any order, a range of CPUs (M-N) or a range with some stride (M-N:S). CPU numbers are zero based. For example, **GOMP\_CPU\_AFFINITY="0 3 1-2 4-15:2"** binds the initial thread to CPU 0, the second to CPU 3, the third to CPU 1, the fourth to CPU 2, the fifth to CPU 4, the sixth through tenth to CPUs 6, 8, 10, 12, and 14 respectively and then starts assigning back from the beginning of the list. **GOMP\_CPU\_AFFINITY=0** binds all threads to CPU 0.

There is no libgomp library routine to determine whether a CPU affinity specification is in effect. As a workaround, language-specific library functions, e.g., **getenv** in C or **GET\_ENVIRONMENT\_VARIABLE** in Fortran, may be used to query the setting of the **GOMP\_CPU\_AFFINITY** environment variable. A defined CPU affinity on startup cannot be changed or disabled during the runtime of the application.

If both **GOMP\_CPU\_AFFINITY** and **OMP\_PROC\_BIND** are set, **OMP\_PROC\_BIND** has a higher precedence. If neither has been set and **OMP\_PROC\_BIND** is unset, or when **OMP\_PROC\_BIND** is set to **FALSE**, the host system handles the assignment of threads to CPUs.

*See also:* Section 4.14 [OMP\_PLACES], page 64, Section 4.13 [OMP\_PROC\_BIND], page 64,

## 4.22 GOMP\_DEBUG – Enable debugging output

### *Description:*

Enable debugging output. The variable should be set to 0 (disabled, also the default if not set), or 1 (enabled).

If enabled, some debugging output is printed during execution. This is currently not specified in more detail, and subject to change.

## 4.23 GOMP\_STACKSIZE – Set default thread stack size

### *Description:*

Set the default thread stack size in kilobytes. This is different from `pthread_attr_setstacksize` which gets the number of bytes as an argument. If the stack size cannot be set due to system constraints, an error is reported and the initial stack size is left unchanged. If undefined, the stack size is system dependent.

*See also:* Section 4.15 [OMP\_STACKSIZE], page 65,

*Reference:* GCC Patches Mailinglist (<https://gcc.gnu.org/ml/gcc-patches/2006-06/msg00493.html>), GCC Patches Mailinglist (<https://gcc.gnu.org/ml/gcc-patches/2006-06/msg00496.html>)

## 4.24 GOMP\_SPINCOUNT – Set the busy-wait spin count

### *Description:*

Determines how long a threads waits actively with consuming CPU power before waiting passively without consuming CPU power. The value may be either INFINITE, INFINITY to always wait actively or an integer which gives the number of spins of the busy-wait loop. The integer may optionally be followed by the following suffixes acting as multiplication factors: k (kilo, thousand), M (mega, million), G (giga, billion), or T (tera, trillion). If undefined, 0 is used when OMP\_WAIT\_POLICY is PASSIVE, 300,000 is used when OMP\_WAIT\_POLICY is undefined and 30 billion is used when OMP\_WAIT\_POLICY is ACTIVE. If there are more OpenMP threads than available CPUs, 1000 and 100 spins are used for OMP\_WAIT\_POLICY being ACTIVE or undefined, respectively; unless the GOMP\_SPINCOUNT is lower or OMP\_WAIT\_POLICY is PASSIVE.

*See also:* Section 4.20 [OMP\_WAIT\_POLICY], page 67,

## 4.25 GOMP\_RTEMS\_THREAD\_POOLS – Set the RTEMS specific thread pools

### *Description:*

This environment variable is only used on the RTEMS real-time operating system. It determines the scheduler instance specific thread pools. The format for GOMP\_RTEMS\_THREAD\_POOLS is a list of optional `<thread-pool-count>[<priority>]<scheduler-name>` configurations separated by `:` where:

- `<thread-pool-count>` is the thread pool count for this scheduler instance.

- `$<priority>` is an optional priority for the worker threads of a thread pool according to `pthread_setschedparam`. In case a priority value is omitted, then a worker thread inherits the priority of the OpenMP primary thread that created it. The priority of the worker thread is not changed after creation, even if a new OpenMP primary thread using the worker has a different priority.
- `@<scheduler-name>` is the scheduler instance name according to the RTEMS application configuration.

In case no thread pool configuration is specified for a scheduler instance, then each OpenMP primary thread of this scheduler instance uses its own dynamically allocated thread pool. To limit the worker thread count of the thread pools, each OpenMP primary thread must call `omp_set_num_threads`.

*Example:* Lets suppose we have three scheduler instances `I0`, `WRK0`, and `WRK1` with `GOMP_RTEMS_THREAD_POOLS` set to `"1@WRK0:3$4@WRK1"`. Then there are no thread pool restrictions for scheduler instance `I0`. In the scheduler instance `WRK0` there is one thread pool available. Since no priority is specified for this scheduler instance, the worker thread inherits the priority of the OpenMP primary thread that created it. In the scheduler instance `WRK1` there are three thread pools available and their worker threads run at priority four.



## 5 Enabling OpenACC

To activate the OpenACC extensions for C/C++ and Fortran, the compile-time flag `-fopenacc` must be specified. This enables the OpenACC directive `#pragma acc` in C/C++ and, in Fortran, the `!$acc` sentinel in free source form and the `c$acc`, `*$acc` and `!$acc` sentinels in fixed source form. The flag also arranges for automatic linking of the OpenACC runtime library (Chapter 6 [OpenACC Runtime Library Routines], page 73).

See <https://gcc.gnu.org/wiki/OpenACC> for more information.

A complete description of all OpenACC directives accepted may be found in the OpenACC (<https://www.openacc.org>) Application Programming Interface manual, version 2.6.



## 6 OpenACC Runtime Library Routines

The runtime routines described here are defined by section 3 of the OpenACC specifications in version 2.6. They have C linkage, and do not throw exceptions. Generally, they are available only for the host, with the exception of `acc_on_device`, which is available for both the host and the acceleration device.

### 6.1 `acc_get_num_devices` – Get number of devices for given device type

*Description*

This function returns a value indicating the number of devices available for the device type specified in *devicetype*.

*C/C++:*

*Prototype:*            `int acc_get_num_devices(acc_device_t devicetype);`

*Fortran:*

*Interface:*            `integer function acc_get_num_devices(devicetype)`  
                          `integer(kind=acc_device_kind) devicetype`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.1.

### 6.2 `acc_set_device_type` – Set type of device accelerator to use.

*Description*

This function indicates to the runtime library which device type, specified in *devicetype*, to use when executing a parallel or kernels region.

*C/C++:*

*Prototype:*            `acc_set_device_type(acc_device_t devicetype);`

*Fortran:*

*Interface:*            `subroutine acc_set_device_type(devicetype)`  
                          `integer(kind=acc_device_kind) devicetype`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.2.

### 6.3 `acc_get_device_type` – Get type of device accelerator to be used.

*Description*

This function returns what device type will be used when executing a parallel or kernels region.

This function returns `acc_device_none` if `acc_get_device_type` is called from `acc_ev_device_init_start`, `acc_ev_device_init_end` callbacks of the OpenACC Profiling Interface (Chapter 10 [OpenACC Profiling Interface], page 101), that is, if the device is currently being initialized.

*C/C++:*

*Prototype:*        `acc_device_t acc_get_device_type(void);`

*Fortran:*

*Interface:*        `function acc_get_device_type(void)  
                      integer(kind=acc_device_kind) acc_get_device_type`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.3.

## 6.4 `acc_set_device_num` – Set device number to use.

*Description*

This function will indicate to the runtime which device number, specified by *devicenum*, associated with the specified device type *devicetype*.

*C/C++:*

*Prototype:*        `acc_set_device_num(int devicenum, acc_device_t  
  devicetype);`

*Fortran:*

*Interface:*        `subroutine acc_set_device_num(devicenum,  
  devicetype)  
                      integer devicenum  
                      integer(kind=acc_device_kind) devicetype`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.4.

## 6.5 `acc_get_device_num` – Get device number to be used.

*Description*

This function returns which device number associated with the specified device type *devicetype*, will be used when executing a parallel or kernels region.

*C/C++:*

*Prototype:*        `int acc_get_device_num(acc_device_t devicetype);`

*Fortran:*

*Interface:*        `function acc_get_device_num(devicetype)  
                      integer(kind=acc_device_kind) devicetype  
                      integer acc_get_device_num`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.5.

## 6.6 `acc_get_property` – Get device property.

*Description*

These routines return the value of the specified *property* for the device being queried according to *devicenum* and *devicetype*. Integer-valued and string-valued properties are returned by `acc_get_property` and `acc_get_property_string` respectively. The Fortran `acc_get_property_string` subroutine returns the string retrieved in its fourth argument while the remaining entry points are functions, which pass the return value as their result.

Note for Fortran, only: the OpenACC technical committee corrected and, hence, modified the interface introduced in OpenACC 2.6. The kind-value parameter `acc_device_property` has been renamed to `acc_device_property_kind` for consistency and the return type of the `acc_get_property` function is now a `c_size_t` integer instead of a `acc_device_property` integer. The parameter `acc_device_property` is still provided, but might be removed in a future version of GCC.

*C/C++:*

```
Prototype:      size_t acc_get_property(int devicenum, acc_device_t
                    devicetype, acc_device_property_t property);
Prototype:      const char *acc_get_property_string(int devicenum,
                    acc_device_t devicetype, acc_device_property_t
                    property);
```

*Fortran:*

```
Interface:      function acc_get_property(devicenum, devicetype,
                    property)
Interface:      subroutine acc_get_property_string(devicenum,
                    devicetype, property, string)
                    use ISO_C_Binding, only: c_size_t
                    integer devicenum
                    integer(kind=acc_device_kind) devicetype
                    integer(kind=acc_device_property_kind) property
                    integer(kind=c_size_t) acc_get_property
                    character(*) string
```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.6.

## 6.7 `acc_async_test` – Test for completion of a specific asynchronous operation.

*Description*

This function tests for completion of the asynchronous operation specified in *arg*. In C/C++, a non-zero value is returned to indicate the specified asynchronous operation has completed while Fortran returns `true`. If the asynchronous operation has not completed, C/C++ returns zero and Fortran returns `false`.

*C/C++:*

```
Prototype:      int acc_async_test(int arg);
```

*Fortran:*

```
Interface:      function acc_async_test(arg)
                    integer(kind=acc_handle_kind) arg
                    logical acc_async_test
```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.9.

## 6.8 `acc_async_test_all` – Tests for completion of all asynchronous operations.

### *Description*

This function tests for completion of all asynchronous operations. In C/C++, a non-zero value is returned to indicate all asynchronous operations have completed while Fortran returns `true`. If any asynchronous operation has not completed, C/C++ returns zero and Fortran returns `false`.

### *C/C++:*

*Prototype:*            `int acc_async_test_all(void);`

### *Fortran:*

*Interface:*            `function acc_async_test()  
                          logical acc_get_device_num`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.10.

## 6.9 `acc_wait` – Wait for completion of a specific asynchronous operation.

### *Description*

This function waits for completion of the asynchronous operation specified in `arg`.

### *C/C++:*

*Prototype:*            `acc_wait(arg);`  
*Prototype (OpenACC 1.0 compatibility):*  
                 `acc_async_wait(arg);`

### *Fortran:*

*Interface:*            `subroutine acc_wait(arg)  
                          integer(acc_handle_kind) arg`  
*Interface (OpenACC 1.0 compatibility):*  
                          `subroutine acc_async_wait(arg)  
                          integer(acc_handle_kind) arg`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.11.

## 6.10 `acc_wait_all` – Waits for completion of all asynchronous operations.

### *Description*

This function waits for the completion of all asynchronous operations.

### *C/C++:*

*Prototype:*            `acc_wait_all(void);`  
*Prototype (OpenACC 1.0 compatibility):*  
                 `acc_async_wait_all(void);`

*Fortran:*

*Interface:*            subroutine acc\_wait\_all()  
*Interface (OpenACC 1.0 compatibility):*    subroutine acc\_async\_wait\_all()

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.13.

## 6.11 acc\_wait\_all\_async – Wait for completion of all asynchronous operations.

*Description*

This function enqueues a wait operation on the queue *async* for any and all asynchronous operations that have been previously enqueued on any queue.

*C/C++:*

*Prototype:*            acc\_wait\_all\_async(int async);

*Fortran:*

*Interface:*            subroutine acc\_wait\_all\_async(async)  
                          integer(acc\_handle\_kind) async

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.14.

## 6.12 acc\_wait\_async – Wait for completion of asynchronous operations.

*Description*

This function enqueues a wait operation on queue *async* for any and all asynchronous operations enqueued on queue *arg*.

*C/C++:*

*Prototype:*            acc\_wait\_async(int arg, int async);

*Fortran:*

*Interface:*            subroutine acc\_wait\_async(arg, async)  
                          integer(acc\_handle\_kind) arg, async

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.12.

## 6.13 acc\_init – Initialize runtime for a specific device type.

*Description*

This function initializes the runtime for the device type specified in *devicetype*.

*C/C++:*

*Prototype:*            acc\_init(acc\_device\_t devicetype);

*Fortran:*

*Interface:*            subroutine acc\_init(devicetype)  
                          integer(acc\_device\_kind) devicetype

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.7.

## 6.14 `acc_shutdown` – Shuts down the runtime for a specific device type.

### Description

This function shuts down the runtime for the device type specified in *devicetype*.

### C/C++:

*Prototype:*            `acc_shutdown(acc_device_t devicetype);`

### Fortran:

*Interface:*            `subroutine acc_shutdown(devicetype)  
                          integer(acc_device_kind) devicetype`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.8.

## 6.15 `acc_on_device` – Whether executing on a particular device

### Description:

This function returns whether the program is executing on a particular device specified in *devicetype*. In C/C++ a non-zero value is returned to indicate the device is executing on the specified device type. In Fortran, `true` is returned. If the program is not executing on the specified device type C/C++ returns zero, while Fortran returns `false`.

Note that in GCC, depending on *devicetype*, the function call might be folded to a constant in the compiler; compile with `-fno-builtin-acc_on_device` if a run-time function is desired.

### C/C++:

*Prototype:*            `acc_on_device(acc_device_t devicetype);`

### Fortran:

*Interface:*            `function acc_on_device(devicetype)  
                          integer(acc_device_kind) devicetype  
                          logical acc_on_device`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.17.

## 6.16 `acc_malloc` – Allocate device memory.

### Description

This function allocates *bytes* bytes of device memory. It returns the device address of the allocated memory.

### C/C++:

*Prototype:*            `d_void* acc_malloc(size_t bytes);`

### Fortran:

*Interface:*            `type(c_ptr) function acc_malloc(bytes)  
                          integer(c_size_t), value :: bytes`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.18.  
openacc specification v3.3 (<https://www.openacc.org>), section 3.2.16.

## 6.17 acc\_free – Free device memory.

### Description

Free previously allocated device memory at the device address `data_dev`.

### C/C++:

*Prototype:*            `void acc_free(d_void *data_dev);`

### Fortran:

*Interface:*            `subroutine acc_free(data_dev)`  
                          `type(c_ptr), value :: data_dev`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.19.  
 openacc specification v3.3 (<https://www.openacc.org>), section 3.2.17.

## 6.18 acc\_copyin – Allocate device memory and copy host memory to it.

### Description

In C/C++, this function allocates *len* bytes of device memory and maps it to the specified host address in *a*. The device address of the newly allocated device memory is returned.

In Fortran, two (2) forms are supported. In the first form, *a* specifies a contiguous array section. The second form *a* specifies a variable or array element and *len* specifies the length in bytes.

### C/C++:

*Prototype:*            `void *acc_copyin(h_void *a, size_t len);`  
*Prototype:*            `void *acc_copyin_async(h_void *a, size_t len, int`  
                          `async);`

### Fortran:

*Interface:*            `subroutine acc_copyin(a)`  
                          `type(*), dimension(..) :: a`  
*Interface:*            `subroutine acc_copyin(a, len)`  
                          `type(*), dimension(..) :: a`  
                          `integer len`  
*Interface:*            `subroutine acc_copyin_async(a, async)`  
                          `type(*), dimension(..) :: a`  
                          `integer(acc_handle_kind) :: async`  
*Interface:*            `subroutine acc_copyin_async(a, len, async)`  
                          `type(*), dimension(..) :: a`  
                          `integer len`  
                          `integer(acc_handle_kind) :: async`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.20.

## 6.19 `acc_present_or_copyin` – If the data is not present on the device, allocate device memory and copy from host memory.

### *Description*

This function tests if the host data specified by `a` and of length `len` is present or not. If it is not present, device memory is allocated and the host memory copied. The device address of the newly allocated device memory is returned.

In Fortran, two (2) forms are supported. In the first form, `a` specifies a contiguous array section. The second form `a` specifies a variable or array element and `len` specifies the length in bytes.

Note that `acc_present_or_copyin` and `acc_pcopyin` exist for backward compatibility with OpenACC 2.0; use Section 6.18 [`acc_copyin`], page 79, instead.

### *C/C++:*

*Prototype:*        `void *acc_present_or_copyin(h_void *a, size_t len);`  
*Prototype:*        `void *acc_pcopyin(h_void *a, size_t len);`

### *Fortran:*

*Interface:*        `subroutine acc_present_or_copyin(a)`  
                       `type(*), dimension(..) :: a`  
*Interface:*        `subroutine acc_present_or_copyin(a, len)`  
                       `type(*), dimension(..) :: a`  
                       `integer len`  
*Interface:*        `subroutine acc_pcopyin(a)`  
                       `type(*), dimension(..) :: a`  
*Interface:*        `subroutine acc_pcopyin(a, len)`  
                       `type(*), dimension(..) :: a`  
                       `integer len`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.20.

## 6.20 `acc_create` – Allocate device memory and map it to host memory.

### *Description*

This function allocates device memory and maps it to host memory specified by the host address `a` with a length of `len` bytes. In C/C++, the function returns the device address of the allocated device memory.

In Fortran, two (2) forms are supported. In the first form, `a` specifies a contiguous array section. The second form `a` specifies a variable or array element and `len` specifies the length in bytes.

### *C/C++:*

*Prototype:*        `void *acc_create(h_void *a, size_t len);`  
*Prototype:*        `void *acc_create_async(h_void *a, size_t len, int`  
                       `async);`

### *Fortran:*

*Interface:*        `subroutine acc_create(a)`

```

Interface:      type(*), dimension(..) :: a
                 subroutine acc_create(a, len)
                 type(*), dimension(..) :: a
                 integer len
Interface:      subroutine acc_create_async(a, async)
                 type(*), dimension(..) :: a
                 integer(acc_handle_kind) :: async
Interface:      subroutine acc_create_async(a, len, async)
                 type(*), dimension(..) :: a
                 integer len
                 integer(acc_handle_kind) :: async

```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.21.

## 6.21 acc\_present\_or\_create – If the data is not present on the device, allocate device memory and map it to host memory.

### *Description*

This function tests if the host data specified by *a* and of length *len* is present or not. If it is not present, device memory is allocated and mapped to host memory. In C/C++, the device address of the newly allocated device memory is returned.

In Fortran, two (2) forms are supported. In the first form, *a* specifies a contiguous array section. The second form *a* specifies a variable or array element and *len* specifies the length in bytes.

Note that `acc_present_or_create` and `acc_pcreate` exist for backward compatibility with OpenACC 2.0; use Section 6.20 [`acc_create`], page 80, instead.

### *C/C++:*

```

Prototype:      void *acc_present_or_create(h_void *a, size_t len)
Prototype:      void *acc_pcreate(h_void *a, size_t len)

```

### *Fortran:*

```

Interface:      subroutine acc_present_or_create(a)
                 type(*), dimension(..) :: a
Interface:      subroutine acc_present_or_create(a, len)
                 type(*), dimension(..) :: a
                 integer len
Interface:      subroutine acc_pcreate(a)
                 type(*), dimension(..) :: a
Interface:      subroutine acc_pcreate(a, len)
                 type(*), dimension(..) :: a
                 integer len

```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.21.

## 6.22 `acc_copyout` – Copy device memory to host memory.

### *Description*

This function copies mapped device memory to host memory which is specified by host address *a* for a length *len* bytes in C/C++.

In Fortran, two (2) forms are supported. In the first form, *a* specifies a contiguous array section. The second form *a* specifies a variable or array element and *len* specifies the length in bytes.

### *C/C++:*

```
Prototype:      acc_copyout(h_void *a, size_t len);
Prototype:      acc_copyout_async(h_void *a, size_t len, int async);
Prototype:      acc_copyout_finalize(h_void *a, size_t len);
Prototype:      acc_copyout_finalize_async(h_void *a, size_t len,
                                           int async);
```

### *Fortran:*

```
Interface:      subroutine acc_copyout(a)
                  type(*), dimension(..) :: a
Interface:      subroutine acc_copyout(a, len)
                  type(*), dimension(..) :: a
                  integer len
Interface:      subroutine acc_copyout_async(a, async)
                  type(*), dimension(..) :: a
                  integer(acc_handle_kind) :: async
Interface:      subroutine acc_copyout_async(a, len, async)
                  type(*), dimension(..) :: a
                  integer len
                  integer(acc_handle_kind) :: async
Interface:      subroutine acc_copyout_finalize(a)
                  type(*), dimension(..) :: a
Interface:      subroutine acc_copyout_finalize(a, len)
                  type(*), dimension(..) :: a
                  integer len
Interface:      subroutine acc_copyout_finalize_async(a, async)
                  type(*), dimension(..) :: a
                  integer(acc_handle_kind) :: async
Interface:      subroutine acc_copyout_finalize_async(a, len,
                                                         async)
                  type(*), dimension(..) :: a
                  integer len
                  integer(acc_handle_kind) :: async
```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.22.

## 6.23 acc\_delete – Free device memory.

### *Description*

This function frees previously allocated device memory specified by the device address *a* and the length of *len* bytes.

In Fortran, two (2) forms are supported. In the first form, *a* specifies a contiguous array section. The second form *a* specifies a variable or array element and *len* specifies the length in bytes.

### *C/C++:*

```
Prototype:      acc_delete(h_void *a, size_t len);
Prototype:      acc_delete_async(h_void *a, size_t len, int async);
Prototype:      acc_delete_finalize(h_void *a, size_t len);
Prototype:      acc_delete_finalize_async(h_void *a, size_t len,
                                         int async);
```

### *Fortran:*

```
Interface:      subroutine acc_delete(a)
                  type(*), dimension(..) :: a
Interface:      subroutine acc_delete(a, len)
                  type(*), dimension(..) :: a
                  integer len
Interface:      subroutine acc_delete_async(a, async)
                  type(*), dimension(..) :: a
                  integer(acc_handle_kind) :: async
Interface:      subroutine acc_delete_async(a, len, async)
                  type(*), dimension(..) :: a
                  integer len
                  integer(acc_handle_kind) :: async
Interface:      subroutine acc_delete_finalize(a)
                  type(*), dimension(..) :: a
Interface:      subroutine acc_delete_finalize(a, len)
                  type(*), dimension(..) :: a
                  integer len
Interface:      subroutine acc_delete_finalize_async(a, async)
                  type(*), dimension(..) :: a
                  integer(acc_handle_kind) :: async
Interface:      subroutine acc_delete_finalize_async(a, len, async)
                  type(*), dimension(..) :: a
                  integer len
                  integer(acc_handle_kind) :: async
```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.23.

## 6.24 `acc_update_device` – Update device memory from mapped host memory.

### *Description*

This function updates the device copy from the previously mapped host memory. The host memory is specified with the host address *a* and a length of *len* bytes.

In Fortran, two (2) forms are supported. In the first form, *a* specifies a contiguous array section. The second form *a* specifies a variable or array element and *len* specifies the length in bytes.

### *C/C++:*

```
Prototype:      acc_update_device(h_void *a, size_t len);
Prototype:      acc_update_device(h_void *a, size_t len, async);
```

### *Fortran:*

```
Interface:      subroutine acc_update_device(a)
                  type(*), dimension(..) :: a
Interface:      subroutine acc_update_device(a, len)
                  type(*), dimension(..) :: a
                  integer len
Interface:      subroutine acc_update_device_async(a, async)
                  type(*), dimension(..) :: a
                  integer(acc_handle_kind) :: async
Interface:      subroutine acc_update_device_async(a, len, async)
                  type(*), dimension(..) :: a
                  integer len
                  integer(acc_handle_kind) :: async
```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.24.

## 6.25 `acc_update_self` – Update host memory from mapped device memory.

### *Description*

This function updates the host copy from the previously mapped device memory. The host memory is specified with the host address *a* and a length of *len* bytes.

In Fortran, two (2) forms are supported. In the first form, *a* specifies a contiguous array section. The second form *a* specifies a variable or array element and *len* specifies the length in bytes.

### *C/C++:*

```
Prototype:      acc_update_self(h_void *a, size_t len);
Prototype:      acc_update_self_async(h_void *a, size_t len, int
                  async);
```

### *Fortran:*

```
Interface:      subroutine acc_update_self(a)
                  type(*), dimension(..) :: a
```

```

Interface:      subroutine acc_update_self(a, len)
                  type(*), dimension(..) :: a
                  integer len
Interface:      subroutine acc_update_self_async(a, async)
                  type(*), dimension(..) :: a
                  integer(acc_handle_kind) :: async
Interface:      subroutine acc_update_self_async(a, len, async)
                  type(*), dimension(..) :: a
                  integer len
                  integer(acc_handle_kind) :: async

```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.25.

## 6.26 acc\_map\_data – Map previously allocated device memory to host memory.

### *Description*

This function maps previously allocated device and host memory. The device memory is specified with the device address *data\_dev*. The host memory is specified with the host address *data\_arg* and a length of *bytes*.

### *C/C++:*

```

Prototype:      void acc_map_data(h_void *data_arg, d_void
                  *data_dev, size_t bytes);

```

### *Fortran:*

```

Interface:      subroutine acc_map_data(data_arg, data_dev, bytes)
                  type(*), dimension(*) :: data_arg
                  type(c_ptr), value :: data_dev
                  integer(c_size_t), value :: bytes

```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.26.  
OpenACC specification v3.3 (<https://www.openacc.org>), section 3.2.21.

## 6.27 acc\_unmap\_data – Unmap device memory from host memory.

### *Description*

This function unmmaps previously mapped device and host memory. The latter specified by *data\_arg*.

### *C/C++:*

```

Prototype:      void acc_unmap_data(h_void *data_arg);

```

### *Fortran:*

```

Interface:      subroutine acc_unmap_data(data_arg)
                  type(*), dimension(*) :: data_arg

```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.27.  
OpenACC specification v3.3 (<https://www.openacc.org>), section 3.2.22.

## 6.28 `acc_deviceptr` – Get device pointer associated with specific host address.

### *Description*

This function returns the device address that has been mapped to the host address specified by *data\_arg*.

### *C/C++:*

*Prototype:*            `void *acc_deviceptr(h_void *data_arg);`

### *Fortran:*

*Interface:*            `type(c_ptr) function acc_deviceptr(data_arg)`  
                          `type(*), dimension(*) :: data_arg`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.28.  
 OpenACC specification v3.3 (<https://www.openacc.org>), section 3.2.23.

## 6.29 `acc_hostptr` – Get host pointer associated with specific device address.

### *Description*

This function returns the host address that has been mapped to the device address specified by *data\_dev*.

### *C/C++:*

*Prototype:*            `void *acc_hostptr(d_void *data_dev);`

### *Fortran:*

*Interface:*            `type(c_ptr) function acc_hostptr(data_dev)`  
                          `type(c_ptr), value :: data_dev`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.29.  
 OpenACC specification v3.3 (<https://www.openacc.org>), section 3.2.24.

## 6.30 `acc_is_present` – Indicate whether host variable / array is present on device.

### *Description*

This function indicates whether the specified host address in *a* and a length of *len* bytes is present on the device. In C/C++, a non-zero value is returned to indicate the presence of the mapped memory on the device. A zero is returned to indicate the memory is not mapped on the device.

In Fortran, two (2) forms are supported. In the first form, *a* specifies a contiguous array section. The second form *a* specifies a variable or array element and *len* specifies the length in bytes. If the host memory is mapped to device memory, then a `true` is returned. Otherwise, a `false` is return to indicate the mapped memory is not present.

### *C/C++:*

*Prototype:*            `int acc_is_present(h_void *a, size_t len);`

*Fortran:*

```

Interface:      function acc_is_present(a)
                  type(*), dimension(..) :: a
                  logical acc_is_present
Interface:      function acc_is_present(a, len)
                  type(*), dimension(..) :: a
                  integer len
                  logical acc_is_present

```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.30.

### 6.31 acc\_memcpy\_to\_device – Copy host memory to device memory.

*Description*

This function copies host memory specified by host address of *data\_host\_src* to device memory specified by the device address *data\_dev\_dest* for a length of *bytes* bytes.

*C/C++:*

```

Prototype:      void acc_memcpy_to_device(d_void* data_dev_dest,
                  h_void* data_host_src, size_t bytes);
Prototype:      void acc_memcpy_to_device_async(d_void* data_dev_
                  dest,
                  h_void* data_host_src, size_t bytes, int async_arg);

```

*Fortran:*

```

Interface:      subroutine acc_memcpy_to_device(data_dev_dest, &
                  data_host_src, bytes)
Interface:      subroutine acc_memcpy_to_device_async(data_dev_
                  dest, &
                  data_host_src, bytes, async_arg)
                  type(c_ptr), value :: data_dev_dest
                  type(*), dimension(*) :: data_host_src
                  integer(c_size_t), value :: bytes
                  integer(acc_handle_kind), value :: async_arg

```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.31  
 OpenACC specification v3.3 (<https://www.openacc.org>), section 3.2.26.

### 6.32 acc\_memcpy\_from\_device – Copy device memory to host memory.

*Description*

This function copies device memory specified by device address of *data\_dev\_src* to host memory specified by the host address *data\_host\_dest* for a length of *bytes* bytes.

*C/C++:*

```

Prototype:      void acc_memcpy_from_device(h_void* data_host_
                  dest,
                  d_void* data_dev_src, size_t bytes);
Prototype:      void acc_memcpy_from_device_async(h_void*
                  data_host_dest,
                  d_void* data_dev_src, size_t bytes, int async_arg);

```

*Fortran:*

```

Interface:      subroutine acc_memcpy_from_device(data_host_dest,
              &
              data_dev_src, bytes)
Interface:      subroutine acc_memcpy_from_device_async(data_host_
              dest, &
              data_dev_src, bytes, async_arg)
              type(*), dimension(*) :: data_host_dest
              type(c_ptr), value :: data_dev_src
              integer(c_size_t), value :: bytes
              integer(acc_handle_kind), value :: async_arg

```

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 3.2.32.  
OpenACC specification v3.3 (<https://www.openacc.org>), section 3.2.27.

### 6.33 `acc_memcpy_device` – Copy memory within a device.

Description

This function copies device memory from one memory location to another on the current device. It copies *bytes* bytes of data from the device address, specified by *data\_dev\_src*, to the device address *data\_dev\_dest*. The `_async` version performs the transfer asynchronously using the queue associated with *async\_arg*.

 $C/C_{++}$ 

```

Prototype:      void acc_memcpy_device(d_void* data_dev_dest,
                        d_void* data_dev_src, size_t bytes);
Prototype:      void acc_memcpy_device_async(d_void* data_dev_
                        dest,
                        d_void* data_dev_src, size_t bytes, int async_arg);

```

*Fortran:*

```

Interface:      subroutine acc_memcpy_device(data_dev_dest, &
                data_dev_src, bytes)
Interface:      subroutine acc_memcpy_device_async(data_dev_dest,
                &
                data_dev_src, bytes, async_arg)
                type(c_ptr), value :: data_dev_dest
                type(c_ptr), value :: data_dev_src
                integer(c_size_t), value :: bytes
                integer(acc_handle_kind), value :: async_arg

```



### 6.36 `acc_get_current_cuda_device` – Get CUDA device handle.

#### *Description*

This function returns the CUDA device handle. This handle is the same as used by the CUDA Runtime or Driver API's.

#### *C/C++:*

*Prototype:*        `void *acc_get_current_cuda_device(void);`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section A.2.1.1.

### 6.37 `acc_get_current_cuda_context` – Get CUDA context handle.

#### *Description*

This function returns the CUDA context handle. This handle is the same as used by the CUDA Runtime or Driver API's.

#### *C/C++:*

*Prototype:*        `void *acc_get_current_cuda_context(void);`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section A.2.1.2.

### 6.38 `acc_get_cuda_stream` – Get CUDA stream handle.

#### *Description*

This function returns the CUDA stream handle for the queue *async*. This handle is the same as used by the CUDA Runtime or Driver API's.

#### *C/C++:*

*Prototype:*        `void *acc_get_cuda_stream(int async);`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section A.2.1.3.

### 6.39 `acc_set_cuda_stream` – Set CUDA stream handle.

#### *Description*

This function associates the stream handle specified by *stream* with the queue *async*.

This cannot be used to change the stream handle associated with `acc_async_sync`.

The return value is not specified.

#### *C/C++:*

*Prototype:*        `int acc_set_cuda_stream(int async, void *stream);`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section A.2.1.4.

## 6.40 `acc_prof_register` – Register callbacks.

*Description:*

This function registers callbacks.

*C/C++:*

*Prototype:*        `void acc_prof_register (acc_event_t, acc_prof_callback, acc_register_t);`

*See also:*    Chapter 10 [OpenACC Profiling Interface], page 101,

*Reference:*   OpenACC specification v2.6 (<https://www.openacc.org>), section 5.3.

## 6.41 `acc_prof_unregister` – Unregister callbacks.

*Description:*

This function unregisters callbacks.

*C/C++:*

*Prototype:*        `void acc_prof_unregister (acc_event_t, acc_prof_callback, acc_register_t);`

*See also:*    Chapter 10 [OpenACC Profiling Interface], page 101,

*Reference:*   OpenACC specification v2.6 (<https://www.openacc.org>), section 5.3.

## 6.42 `acc_prof_lookup` – Obtain inquiry functions.

*Description:*

Function to obtain inquiry functions.

*C/C++:*

*Prototype:*        `acc_query_fn acc_prof_lookup (const char *);`

*See also:*    Chapter 10 [OpenACC Profiling Interface], page 101,

*Reference:*   OpenACC specification v2.6 (<https://www.openacc.org>), section 5.3.

## 6.43 `acc_register_library` – Library registration.

*Description:*

Function for library registration.

*C/C++:*

*Prototype:*        `void acc_register_library (acc_prof_reg, acc_prof_reg, acc_prof_lookup_func);`

*See also:*    Chapter 10 [OpenACC Profiling Interface], page 101, Section 7.3 [ACC\_PROFLIB], page 93,

*Reference:*   OpenACC specification v2.6 (<https://www.openacc.org>), section 5.3.



## 7 OpenACC Environment Variables

The variables `ACC_DEVICE_TYPE` and `ACC_DEVICE_NUM` are defined by section 4 of the OpenACC specification in version 2.0. The variable `ACC_PROFLIB` is defined by section 4 of the OpenACC specification in version 2.6.

### 7.1 `ACC_DEVICE_TYPE`

*Description:*

Control the default device type to use when executing compute regions. If unset, the code can be run on any device type, favoring a non-host device type.

Supported values in GCC (if compiled in) are

- `host`
- `nvidia`
- `radeon`

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 4.1.

### 7.2 `ACC_DEVICE_NUM`

*Description:*

Control which device, identified by device number, is the default device. The value must be a nonnegative integer less than the number of devices. If unset, device number zero is used.

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 4.2.

### 7.3 `ACC_PROFLIB`

*Description:*

Semicolon-separated list of dynamic libraries that are loaded as profiling libraries. Each library must provide at least the `acc_register_library` routine. Each library file is found as described by the documentation of `dlopen` of your operating system.

*See also:* Section 6.43 [`acc_register_library`], page 91, Chapter 10 [OpenACC Profiling Interface], page 101,

*Reference:* OpenACC specification v2.6 (<https://www.openacc.org>), section 4.3.



## 8 CUDA Streams Usage

This applies to the `nvptx` plugin only.

The library provides elements that perform asynchronous movement of data and asynchronous operation of computing constructs. This asynchronous functionality is implemented by making use of CUDA streams<sup>1</sup>.

The primary means by that the asynchronous functionality is accessed is through the use of those OpenACC directives which make use of the `async` and `wait` clauses. When the `async` clause is first used with a directive, it creates a CUDA stream. If an `async-argument` is used with the `async` clause, then the stream is associated with the specified `async-argument`.

Following the creation of an association between a CUDA stream and the `async-argument` of an `async` clause, both the `wait` clause and the `wait` directive can be used. When either the clause or directive is used after stream creation, it creates a rendezvous point whereby execution waits until all operations associated with the `async-argument`, that is, stream, have completed.

Normally, the management of the streams that are created as a result of using the `async` clause, is done without any intervention by the caller. This implies the association between the `async-argument` and the CUDA stream is maintained for the lifetime of the program. However, this association can be changed through the use of the library function `acc_set_cuda_stream`. When the function `acc_set_cuda_stream` is called, the CUDA stream that was originally associated with the `async` clause is destroyed. Caution should be taken when changing the association as subsequent references to the `async-argument` refer to a different CUDA stream.

---

<sup>1</sup> See "Stream Management" in "CUDA Driver API", TRM-06703-001, Version 5.5, for additional information



## 9 OpenACC Library Interoperability

### 9.1 Introduction

The OpenACC library uses the CUDA Driver API, and may interact with programs that use the Runtime library directly, or another library based on the Runtime library, e.g., CUBLAS<sup>1</sup>. This chapter describes the use cases and what changes are required in order to use both the OpenACC library and the CUBLAS and Runtime libraries within a program.

### 9.2 First invocation: NVIDIA CUBLAS library API

In this first use case (see below), a function in the CUBLAS library is called prior to any of the functions in the OpenACC library. More specifically, the function `cublasCreate()`.

When invoked, the function initializes the library and allocates the hardware resources on the host and the device on behalf of the caller. Once the initialization and allocation has completed, a handle is returned to the caller. The OpenACC library also requires initialization and allocation of hardware resources. Since the CUBLAS library has already allocated the hardware resources for the device, all that is left to do is to initialize the OpenACC library and acquire the hardware resources on the host.

Prior to calling the OpenACC function that initializes the library and allocate the host hardware resources, you need to acquire the device number that was allocated during the call to `cublasCreate()`. The invoking of the runtime library function `cudaGetDevice()` accomplishes this. Once acquired, the device number is passed along with the device type as parameters to the OpenACC library function `acc_set_device_num()`.

Once the call to `acc_set_device_num()` has completed, the OpenACC library uses the context that was created during the call to `cublasCreate()`. In other words, both libraries share the same context.

```
/* Create the handle */
s = cublasCreate(&h);
if (s != CUBLAS_STATUS_SUCCESS)
{
    fprintf(stderr, "cublasCreate failed %d\n", s);
    exit(EXIT_FAILURE);
}

/* Get the device number */
e = cudaGetDevice(&dev);
if (e != cudaSuccess)
{
    fprintf(stderr, "cudaGetDevice failed %d\n", e);
    exit(EXIT_FAILURE);
}

/* Initialize OpenACC library and use device 'dev' */
acc_set_device_num(dev, acc_device_nvidia);
```

Use Case 1

---

<sup>1</sup> See section 2.26, "Interactions with the CUDA Driver API" in "CUDA Runtime API", Version 5.5, and section 2.27, "VDPAU Interoperability", in "CUDA Driver API", TRM-06703-001, Version 5.5, for additional information on library interoperability.

### 9.3 First invocation: OpenACC library API

In this second use case (see below), a function in the OpenACC library is called prior to any of the functions in the CUBLAS library. More specifically, the function `acc_set_device_num()`.

In the use case presented here, the function `acc_set_device_num()` is used to both initialize the OpenACC library and allocate the hardware resources on the host and the device. In the call to the function, the call parameters specify which device to use and what device type to use, i.e., `acc_device_nvidia`. It should be noted that this is but one method to initialize the OpenACC library and allocate the appropriate hardware resources. Other methods are available through the use of environment variables and these is discussed in the next section.

Once the call to `acc_set_device_num()` has completed, other OpenACC functions can be called as seen with multiple calls being made to `acc_copyin()`. In addition, calls can be made to functions in the CUBLAS library. In the use case a call to `cublasCreate()` is made subsequent to the calls to `acc_copyin()`. As seen in the previous use case, a call to `cublasCreate()` initializes the CUBLAS library and allocates the hardware resources on the host and the device. However, since the device has already been allocated, `cublasCreate()` only initializes the CUBLAS library and allocates the appropriate hardware resources on the host. The context that was created as part of the OpenACC initialization is shared with the CUBLAS library, similarly to the first use case.

```
dev = 0;

acc_set_device_num(dev, acc_device_nvidia);

/* Copy the first set to the device */
d_X = acc_copyin(&h_X[0], N * sizeof (float));
if (d_X == NULL)
{
    fprintf(stderr, "copyin error h_X\n");
    exit(EXIT_FAILURE);
}

/* Copy the second set to the device */
d_Y = acc_copyin(&h_Y1[0], N * sizeof (float));
if (d_Y == NULL)
{
    fprintf(stderr, "copyin error h_Y1\n");
    exit(EXIT_FAILURE);
}

/* Create the handle */
s = cublasCreate(&h);
if (s != CUBLAS_STATUS_SUCCESS)
{
    fprintf(stderr, "cublasCreate failed %d\n", s);
    exit(EXIT_FAILURE);
}

/* Perform saxpy using CUBLAS library function */
s = cublasSaxpy(h, N, &alpha, d_X, 1, d_Y, 1);
if (s != CUBLAS_STATUS_SUCCESS)
{
```

```
        fprintf(stderr, "cublasSaxpy failed %d\n", s);
        exit(EXIT_FAILURE);
    }

    /* Copy the results from the device */
    acc_memcpy_from_device(&h_Y1[0], d_Y, N * sizeof (float));
```

#### Use Case 2

## 9.4 OpenACC library and environment variables

There are two environment variables associated with the OpenACC library that may be used to control the device type and device number: `ACC_DEVICE_TYPE` and `ACC_DEVICE_NUM`, respectively. These two environment variables can be used as an alternative to calling `acc_set_device_num()`. As seen in the second use case, the device type and device number were specified using `acc_set_device_num()`. If however, the aforementioned environment variables were set, then the call to `acc_set_device_num()` would not be required.

The use of the environment variables is only relevant when an OpenACC function is called prior to a call to `cudaCreate()`. If `cudaCreate()` is called prior to a call to an OpenACC function, then you must call `acc_set_device_num()`<sup>2</sup>

---

<sup>2</sup> More complete information about `ACC_DEVICE_TYPE` and `ACC_DEVICE_NUM` can be found in sections 4.1 and 4.2 of the OpenACC (<https://www.openacc.org>) Application Programming Interface”, Version 2.6.



## 10 OpenACC Profiling Interface

### 10.1 Implementation Status and Implementation-Defined Behavior

We’re implementing the OpenACC Profiling Interface as defined by the OpenACC 2.6 specification. We’re clarifying some aspects here as *implementation-defined behavior*, while they’re still under discussion within the OpenACC Technical Committee.

This implementation is tuned to keep the performance impact as low as possible for the (very common) case that the Profiling Interface is not enabled. This is relevant, as the Profiling Interface affects all the *hot* code paths (in the target code, not in the offloaded code). Users of the OpenACC Profiling Interface can be expected to understand that performance is impacted to some degree once the Profiling Interface is enabled: for example, because of the *runtime* (libgomp) calling into a third-party *library* for every event that has been registered.

We’re not yet accounting for the fact that *OpenACC events may occur during event processing*. We just handle one case specially, as required by CUDA 9.0 `nvprof`, that `acc_get_device_type` (Section 6.3 [`acc_get_device_type`], page 73)) may be called from `acc_ev_device_init_start`, `acc_ev_device_init_end` callbacks.

We’re not yet implementing initialization via a `acc_register_library` function that is either statically linked in, or dynamically via `LD_PRELOAD`. Initialization via `acc_register_library` functions dynamically loaded via the `ACC_PROFLIB` environment variable does work, as does directly calling `acc_prof_register`, `acc_prof_unregister`, `acc_prof_lookup`.

As currently there are no inquiry functions defined, calls to `acc_prof_lookup` always returns `NULL`.

There aren’t separate *start*, *stop* events defined for the event types `acc_ev_create`, `acc_ev_delete`, `acc_ev_alloc`, `acc_ev_free`. It’s not clear if these should be triggered before or after the actual device-specific call is made. We trigger them after.

Remarks about data provided to callbacks:

`acc_prof_info.event_type`

It’s not clear if for *nested* event callbacks (for example, `acc_ev_enqueue_launch_start` as part of a parent compute construct), this should be set for the nested event (`acc_ev_enqueue_launch_start`), or if the value of the parent construct should remain (`acc_ev_compute_construct_start`). In this implementation, the value generally corresponds to the innermost nested event type.

`acc_prof_info.device_type`

- For `acc_ev_compute_construct_start`, and in presence of an `if` clause with *false* argument, this still refers to the offloading device type. It’s not clear if that’s the expected behavior.
- Complementary to the item before, for `acc_ev_compute_construct_end`, this is set to `acc_device_host` in presence of an `if` clause with *false* argument. It’s not clear if that’s the expected behavior.







- `acc_update_device`, `acc_update_device_async`
- `acc_update_self`, `acc_update_self_async`
- `acc_map_data`, `acc_unmap_data`
- `acc_memcpy_to_device`, `acc_memcpy_to_device_async`
- `acc_memcpy_from_device`, `acc_memcpy_from_device_async`





















Available properties for an HIP interop object:

Property	C data type	API routine	value (if constant)
fr_id	omp_interop_fr_t	int	omp_fr_hip
fr_name	const char *	str	"hip"
vendor	int	int	11
vendor_name	const char *	str	"nvidia"
device_num	int	int	
platform	N/A		
device	hipDevice_t	int	
device_context	hipCtx_t	ptr	
targetsync	hipStream_t	ptr	













## 14 Reporting Bugs

Bugs in the GNU Offloading and Multi Processing Runtime Library should be reported via Bugzilla (<https://gcc.gnu.org/bugzilla/>). Please add "openacc", or "openmp", or both to the keywords field in the bug report, as appropriate.















































## Library Index

### A

acc\_get\_property ..... 74  
acc\_get\_property\_string ..... 74

### E

Environment Variable ... 59, 60, 61, 62, 63, 64, 65,  
66, 67, 68

### F

FDL, GNU Free Documentation License ..... 137

### I

Implementation specific setting.. 63, 65, 66, 68, 107