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

Short Contents

Table of Contents

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	
`metadirective` directive	Y	
`declare variant` directive	Y	
target-offload-var ICV and `OMP_TARGET_OFFLOAD` env variable	Y	
Nested-parallel changes to *max-active-levels-var* ICV	Y	
`requires` directive	Y	See also Chapter 12 [Offload-Target Specifics], page 113,
`teams` construct outside an enclosing target region	Y	
Non-rectangular loop nests	P	Full support for C/C++, partial for Fortran (PR110735 (`https://gcc.gnu.org/PR110735`))
`!=` as relational-op in canonical loop form for C/C++	Y	
`nonmonotonic` as default loop schedule modifier for worksharing-loop constructs	Y	
Collapse of associated loops that are imperfectly nested loops	Y	
Clauses `if`, `nontemporal` and `order(concurrent)` in `simd` construct	Y	
`atomic` constructs in `simd`	Y	
`loop` construct	Y	
`order(concurrent)` clause	Y	
`scan` directive and `in_scan` modifier for the `reduction` clause	Y	
`in_reduction` clause on `task` constructs	Y	
`in_reduction` clause on `target` constructs	P	`nowait` only stub
`task_reduction` clause with `taskgroup`	Y	
`task` modifier to `reduction` clause	Y	

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

`declare mapper` directive	N	
`omp_get_supported_active_levels` routine	Y	
Runtime routines and environment variables to display runtime thread affinity information	Y	
`omp_pause_resource` and `omp_pause_resource_all` runtime routines	Y	
`omp_get_device_num` 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	
`omp_all_memory` reserved locator	Y	
target_device trait in OpenMP Context	Y	
`target_device` selector set in context selectors	Y	
C/C++'s `declare variant` directive: elision support of preprocessed code	N	
`declare variant`: new clauses `adjust_args` and `append_args`	Y	
`dispatch` construct	Y	
device-specific ICV settings with environment variables	Y	
`assume` and `assumes` directives	Y	
`nothing` directive	Y	
`error` directive	Y	
`masked` construct	Y	
`scope` directive	Y	
Loop transformation constructs	Y	
`strict` modifier in the `grainsize` and `num_tasks` clauses of the `taskloop` construct	Y	
`align` clause in `allocate` directive	P	Only C and Fortran
`align` modifier in `allocate` clause	Y	
`thread_limit` clause to `target` construct	Y	
`has_device_addr` clause to `target` construct	Y	
Iterators in `target update` motion clauses and `map` clauses	N	
Indirect calls to the device version of a procedure or function in `target` regions	Y	

`interop` directive	Y	Cf. Chapter 12 [Offload-Target Specifics], page 113,
`omp_interop_t` object support in runtime routines	Y	
`nowait` clause in `taskwait` directive	Y	
Extensions to the `atomic` directive	Y	
`seq_cst` clause on a `flush` construct	Y	
`inoutset` argument to the `depend` clause	Y	
`private` and `firstprivate` argument to `default` clause in C and C++	Y	
`present` argument to `defaultmap` clause	Y	
`omp_set_num_teams`, `omp_set_teams_thread_limit`, `omp_get_max_teams`, `omp_get_teams_thread_limit` runtime routines	Y	
`omp_target_is_accessible` runtime routine	Y	
`omp_target_memcpy_async` and `omp_target_memcpy_rect_async` runtime routines	Y	
`omp_get_mapped_ptr` runtime routine	Y	
`omp_calloc`, `omp_realloc`, `omp_aligned_alloc` and `omp_aligned_calloc` runtime routines	Y	
`omp_alloctrait_key_t` enum: `omp_atv_serialized` added, `omp_atv_default` changed	Y	
`omp_display_env` runtime routine	Y	
`ompt_scope_endpoint_t` enum: `ompt_scope_beginend`	N	
`ompt_sync_region_t` enum additions	N	
`ompt_state_t` enum: `ompt_state_wait_barrier_implementation` and `ompt_state_wait_barrier_teams`	N	
`ompt_callback_target_data_op_emi_t`, `ompt_callback_target_emi_t`, `ompt_callback_target_map_emi_t` and `ompt_callback_target_submit_emi_t`	N	
`ompt_callback_error_t` type	N	
`OMP_PLACES` syntax extensions	Y	
`OMP_NUM_TEAMS` and `OMP_TEAMS_THREAD_LIMIT` 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	
`unconstrained` and `reproducible` modifiers on `order` clause	Y	
Support `begin/end declare target` 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 `USE`, `IMPORT`, and `IMPLICIT` as invalid	N	
Optional comma between directive and clause in the `#pragma` form	Y	
`indirect` clause in `declare target`	Y	
`device_type(nohost)`/`device_type(host)` for variables	N	
`present` modifier to the `map`, `to` and `from` clauses	Y	
Changed interaction between `declare target` and OpenMP context	Y	
Dynamic selector support in `metadirective`	Y	
Dynamic selector support in `declare variant`	P	Fortran rejects non-constant expressions in dynamic selectors; C/C++ reject expressions using argument variables. (PR113904 (`https://gcc.gnu.org/PR113904`))

2.4 OpenMP 5.2

New features listed in Appendix B of the OpenMP specification

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

[1] 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.

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

Other new OpenMP 5.2 features

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

Invoke virtual member functions of C++ objects created on the host device on other devices	N	
`mapper` as map-type modifier in `declare mapper`	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	
`_ALL` suffix to the device-scope environment variables	P	Host device number wrongly accepted
`num_threads` clause now accepts a list	N	
Abstract names added for `OMP_NUM_THREADS`, `OMP_THREAD_LIMIT` and `OMP_TEAMS_THREAD_LIMIT`	N	
Supporting increments with abstract names in `OMP_PLACES`	N	
Extension of `OMP_DEFAULT_DEVICE` and new `OMP_AVAILABLE_DEVICES` environment vars	N	
New `uid` trait for target devices and for `OMP_AVAILABLE_DEVICES` and `OMP_DEFAULT_DEVICE`	N	
New `OMP_THREADS_RESERVE` environment variable	N	
The `decl` 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 `do concurrent`	N	
All inarguable clauses take now an optional Boolean argument	N	
The `adjust_args` clause was extended to specify the argument by position and supports variadic arguments	N	
For Fortran, *locator list* can be also function reference with data pointer result	N	
Concept of *assumed-size arrays* in C and C++	N	
directive-name-modifier accepted in all clauses	N	
Extension of `interop` operation of `append_args`, allowing all modifiers of the `init` clause	Y	
New argument-free version of `depobj` with repeatable clauses and the `init` clause	N	
Undeprecate omitting the argument to the `depend` clause of the argument version of the `depend` construct	Y	
For Fortran, atomic with BLOCK 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 `looprange` clause	N	
For Fortran, handling polymorphic types in data-sharing-attribute clauses	P	`private` not supported
For Fortran, rejecting polymorphic types in data-mapping clauses	N	not diagnosed (and mostly unsupported)
New `taskgraph` construct including `saved` modifier and `replayable` clause	N	
`default` clause on the `target` directive and accepting variable categories	N	
Semantic change regarding the reference count update with `use_device_ptr` and `use_device_addr`	N	
Support for inductions	N	
Reduction over private variables with `reduction` clause	N	
Implicit reduction identifiers of C++ classes	N	
New `init_complete` clause to the `scan` directive	N	
`ref` modifier to the `map` clause	N	
New `storage` map-type modifier; context-dependent `alloc` and `release` are aliases	N	
Change of the *map-type* property from *ultimate* to *default*	N	
`self` modifier to `map` and `self` as `defaultmap` argument	N	
Mapping of *assumed-size arrays* in C, C++ and Fortran	N	
`delete` as delete-modifier not as map type	N	
For Fortran, the `automap` modifier to the `enter` clause of `declare_target`	N	
`groupprivate` directive	N	
`local` clause to `declare_target` directive	N	
`part_size` allocator trait for `interleaved` allocator partitions	N	
`pin_device`, `preferred_device` and `target_access` allocator traits	N	
`access` allocator trait changes	N	
New `partitioner` value to `partition` allocator trait	N	
Semicolon-separated list to `uses_allocators`	N	
New `need_device_addr` modifier to `adjust_args` clause	N	
`interop` clause to `dispatch`	Y	
Scope requirement changes for `declare_target`	N	
`message` and `severity` clauses to `parallel` directive	N	

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

2.6 OpenMP Technical Report 14

Technical Report (TR) 14 is the first preview for OpenMP 6.1.

New features listed in Appendix B of the OpenMP specification

The `depth` clause to `fuse` directive	N
The `attach` modifier to the `map` clause	N
The `dyn_groupprivate` clause and the `omp_get_dyn_groupprivate_ptr`, `omp_get_dyn_groupprivate_size`, and `omp_get_dyn_groupprivate_size` routines	N
`begin declare_variant` directive in Fortran	N
`grid` and `tile` modifier to the `size` clause	N
New `flatten` loop-transforming directive	N
`scaled` modifier to `simdlen` clause	N
New `omp_default_device` identifier as conforming device number	Y
Clarify when `omp_target_is_accessible` routine returns zero	N

Deprecated features, unless listed above

Deprecation of conditional-update-capture structured block without a capture statement	N

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 113,

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 113,

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 memory. If *ptr* is a null pointer, *true* is returned and if *device_num* is an invalid device number, *false* is returned.

If those conditions do not apply, *true* is returned if the association has been established by an explicit or implicit `map` clause, the `declare target` directive or a call to the `omp_target_associate_ptr` routine.

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

C/C++

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

Fortran:

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
```

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

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

3.7.4 `omp_target_is_accessible` – Check whether memory is device accessible

Description:

This routine tests whether memory, starting at the address given by *ptr* and extending *size* bytes, is accessibly on the device specified by *device_num*. If so, it returns a nonzero value and otherwise zero.

The address given by *ptr* is interpreted to be in the address space of the device and *size* must be positive.

Note that GCC's current implementation assumes that *ptr* is a valid host pointer. Therefore, all addresses given by *ptr* are assumed to be accessible on the initial device. And, to err on the safe side, this memory is only available on a non-host device that can access all host memory ([uniform] shared memory access).

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

C/C++

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

Fortran:

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 113,

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 113,

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* 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 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 is refers to the initial device or to a device with memory accessible from the host (shared memory), the `omp_get_mapped_ptr` routines 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 113,

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 113,

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 113,

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 113,

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_alloctrait), 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 that 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 prefedined 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 113,

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 `omp_get_team_num`
T	num_teams	value returned by `omp_get_num_teams`
L	nesting_level	value returned by `omp_get_level`
n	thread_num	value returned by `omp_get_thread_num`
N	num_threads	value returned by `omp_get_num_threads`
a	ancestor_tnum	value returned by `omp_get_ancestor_thread_num(omp_get_level()-1)`
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 IO, 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 IO. 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)`

```
type(*), dimension(..) :: a
```

Interface:
```
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 unmaps 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
```

Reference: OpenACC specification v2.6 (`https://www.openacc.org`), section 3.2.33. OpenACC specification v3.3 (`https://www.openacc.org`), section 3.2.28.

6.34 `acc_attach` – Let device pointer point to device-pointer target.

Description

This function updates a pointer on the device from pointing to a host-pointer address to pointing to the corresponding device data.

C/C++:

Prototype: `void acc_attach(h_void **ptr_addr);`
Prototype: `void acc_attach_async(h_void **ptr_addr, int async);`

Fortran:

Interface: `subroutine acc_attach(ptr_addr)`
Interface: `subroutine acc_attach_async(ptr_addr, async_arg)`
`type(*), dimension(..) :: ptr_addr`
`integer(acc_handle_kind), value :: async_arg`

Reference: OpenACC specification v2.6 (`https://www.openacc.org`), section 3.2.34. OpenACC specification v3.3 (`https://www.openacc.org`), section 3.2.29.

6.35 `acc_detach` – Let device pointer point to host-pointer target.

Description

This function updates a pointer on the device from pointing to a device-pointer address to pointing to the corresponding host data.

C/C++:

Prototype: `void acc_detach(h_void **ptr_addr);`
Prototype: `void acc_detach_async(h_void **ptr_addr, int async);`
Prototype: `void acc_detach_finalize(h_void **ptr_addr);`
Prototype: `void acc_detach_finalize_async(h_void **ptr_addr, int async);`

Fortran:

Interface: `subroutine acc_detach(ptr_addr)`
Interface: `subroutine acc_detach_async(ptr_addr, async_arg)`
Interface: `subroutine acc_detach_finalize(ptr_addr)`
Interface: `subroutine acc_detach_finalize_async(ptr_addr, async_arg)`
`type(*), dimension(..) :: ptr_addr`
`integer(acc_handle_kind), value :: async_arg`

Reference: OpenACC specification v2.6 (`https://www.openacc.org`), section 3.2.35. OpenACC specification v3.3 (`https://www.openacc.org`), section 3.2.29.

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[1].

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.

[1] 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[1]. 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

[1] 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()`[2]

[2] 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_prof_info.thread_id`
Always `-1`; not yet implemented.

`acc_prof_info.async`
- Not yet implemented correctly for `acc_ev_compute_construct_start`.
- In a compute construct, for host-fallback execution/`acc_device_host` it always is `acc_async_sync`. It is unclear if that is the expected behavior.
- For `acc_ev_device_init_start` and `acc_ev_device_init_end`, it will always be `acc_async_sync`. It is unclear if that is the expected behavior.

`acc_prof_info.async_queue`
There is no *limited number of asynchronous queues* in libgomp. This always has the same value as `acc_prof_info.async`.

`acc_prof_info.src_file`
Always `NULL`; not yet implemented.

`acc_prof_info.func_name`
Always `NULL`; not yet implemented.

`acc_prof_info.line_no`
Always `-1`; not yet implemented.

`acc_prof_info.end_line_no`
Always `-1`; not yet implemented.

`acc_prof_info.func_line_no`
Always `-1`; not yet implemented.

`acc_prof_info.func_end_line_no`
Always `-1`; not yet implemented.

`acc_event_info.event_type`, `acc_event_info.*.event_type`
Relating to `acc_prof_info.event_type` discussed above, in this implementation, this will always be the same value as `acc_prof_info.event_type`.

`acc_event_info.*.parent_construct`
- Will be `acc_construct_parallel` for all OpenACC compute constructs as well as many OpenACC Runtime API calls; should be the one matching the actual construct, or `acc_construct_runtime_api`, respectively.
- Will be `acc_construct_enter_data` or `acc_construct_exit_data` when processing variable mappings specified in OpenACC *declare* directives; should be `acc_construct_declare`.
- For implicit `acc_ev_device_init_start`, `acc_ev_device_init_end`, and explicit as well as implicit `acc_ev_alloc`, `acc_ev_free`, `acc_ev_enqueue_upload_start`, `acc_ev_enqueue_upload_end`, `acc_ev_enqueue_download_start`, and `acc_ev_enqueue_download_end`, will be `acc_construct_parallel`; should reflect the real parent construct.

`acc_event_info.*.implicit`
For `acc_ev_alloc`, `acc_ev_free`, `acc_ev_enqueue_upload_start`, `acc_ev_enqueue_upload_end`, `acc_ev_enqueue_download_start`, and

`acc_ev_enqueue_download_end`, this currently will be `1` also for explicit usage.

`acc_event_info.data_event.var_name`
: Always `NULL`; not yet implemented.

`acc_event_info.data_event.host_ptr`
: For `acc_ev_alloc`, and `acc_ev_free`, this is always `NULL`.

`typedef union acc_api_info`
: ... as printed in *5.2.3. Third Argument: API-Specific Information*. This should obviously be `typedef` *`struct`* `acc_api_info`.

`acc_api_info.device_api`
: Possibly not yet implemented correctly for `acc_ev_compute_construct_start`, `acc_ev_device_init_start`, `acc_ev_device_init_end`: will always be `acc_device_api_none` for these event types. For `acc_ev_enter_data_start`, it will be `acc_device_api_none` in some cases.

`acc_api_info.device_type`
: Always the same as `acc_prof_info.device_type`.

`acc_api_info.vendor`
: Always `-1`; not yet implemented.

`acc_api_info.device_handle`
: Always `NULL`; not yet implemented.

`acc_api_info.context_handle`
: Always `NULL`; not yet implemented.

`acc_api_info.async_handle`
: Always `NULL`; not yet implemented.

Remarks about certain event types:

`acc_ev_device_init_start`, `acc_ev_device_init_end`

- When a compute construct triggers implicit `acc_ev_device_init_start` and `acc_ev_device_init_end` events, they currently aren't *nested within* the corresponding `acc_ev_compute_construct_start` and `acc_ev_compute_construct_end`, but they're currently observed *before* `acc_ev_compute_construct_start`. It's not clear what to do: the standard asks us provide a lot of details to the `acc_ev_compute_construct_start` callback, without (implicitly) initializing a device before?
- Callbacks for these event types will not be invoked for calls to the `acc_set_device_type` and `acc_set_device_num` functions. It's not clear if they should be.

`acc_ev_enter_data_start`, `acc_ev_enter_data_end`, `acc_ev_exit_data_start`, `acc_ev_exit_data_end`

- Callbacks for these event types will also be invoked for OpenACC *host_data* constructs. It's not clear if they should be.

- Callbacks for these event types will also be invoked when processing variable mappings specified in OpenACC *declare* directives. It's not clear if they should be.

Callbacks for the following event types will be invoked, but dispatch and information provided therein has not yet been thoroughly reviewed:

- `acc_ev_alloc`
- `acc_ev_free`
- `acc_ev_update_start`, `acc_ev_update_end`
- `acc_ev_enqueue_upload_start`, `acc_ev_enqueue_upload_end`
- `acc_ev_enqueue_download_start`, `acc_ev_enqueue_download_end`

During device initialization, and finalization, respectively, callbacks for the following event types will not yet be invoked:

- `acc_ev_alloc`
- `acc_ev_free`

Callbacks for the following event types have not yet been implemented, so currently won't be invoked:

- `acc_ev_device_shutdown_start`, `acc_ev_device_shutdown_end`
- `acc_ev_runtime_shutdown`
- `acc_ev_create`, `acc_ev_delete`
- `acc_ev_wait_start`, `acc_ev_wait_end`

For the following runtime library functions, not all expected callbacks will be invoked (mostly concerning implicit device initialization):

- `acc_get_num_devices`
- `acc_set_device_type`
- `acc_get_device_type`
- `acc_set_device_num`
- `acc_get_device_num`
- `acc_init`
- `acc_shutdown`

Aside from implicit device initialization, for the following runtime library functions, no callbacks will be invoked for shared-memory offloading devices (it's not clear if they should be):

- `acc_malloc`
- `acc_free`
- `acc_copyin`, `acc_present_or_copyin`, `acc_copyin_async`
- `acc_create`, `acc_present_or_create`, `acc_create_async`
- `acc_copyout`, `acc_copyout_async`, `acc_copyout_finalize`, `acc_copyout_finalize_async`
- `acc_delete`, `acc_delete_async`, `acc_delete_finalize`, `acc_delete_finalize_async`

- `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`

11 OpenMP-Implementation Specifics

11.1 Implementation-defined ICV Initialization

affinity-format-var	See Section 4.2 [OMP_AFFINITY_FORMAT], page 60.
def-allocator-var	See Section 4.1 [OMP_ALLOCATOR], page 59.
max-active-levels-var	See Section 4.8 [OMP_MAX_ACTIVE_LEVELS], page 62.
dyn-var	See Section 4.7 [OMP_DYNAMIC], page 62.
nthreads-var	See Section 4.12 [OMP_NUM_THREADS], page 63.
num-devices-var	Number of non-host devices found by GCC's run-time library
num-procs-var	The number of CPU cores on the initial device, except that affinity settings might lead to a smaller number. On non-host devices, the value of the *nthreads-var* ICV.
place-partition-var	See Section 4.14 [OMP_PLACES], page 64.
run-sched-var	See Section 4.16 [OMP_SCHEDULE], page 65.
stacksize-var	See Section 4.15 [OMP_STACKSIZE], page 65.
thread-limit-var	See Section 4.18 [OMP_TEAMS_THREAD_LIMIT], page 66,
wait-policy-var	See Section 4.20 [OMP_WAIT_POLICY], page 67, and Section 4.24 [GOMP_SPINCOUNT], page 68,

11.2 OpenMP Context Selectors

`vendor` is always `gnu`. References are to the GCC manual.

For the host compiler, `kind` always matches `host`, `cpu` and `any`; for the offloading architectures AMD GCN and Nvidia PTX, `kind` always matches `nohost`, `gpu` and `any`. For the x86 family of computers, AMD GCN and Nvidia PTX the following traits are supported in addition; while OpenMP is supported on more architectures, GCC currently does not match any `arch` or `isa` traits for those.

Note that for AMD GCN and Nvidia PTX, the `isa` is currently an exact match between the compiled-for ISA architecture and the matching `isa` trait value. For instance, when compiling for `gfx942`, the `isa` trait value `gfx9-4-generic` is not matched and, likewise, `gfx942` is not matched when compiling for its generic architecture.

`arch`	`isa`
`x86`, `x86_64`, `i386`, `i486`, `i586`, `i686`, `ia32`	See `-m...` flags in "x86 Options" (without `-m`)
`amdgcn`, `gcn`	See `-march=` in "AMD GCN Options"
`nvptx`, `nvptx64`	See `-march=` in "Nvidia PTX Options"

11.3 Memory allocation

The description below applies to:

- Explicit use of the OpenMP API routines, see Section 3.12 [Memory Management Routines], page 50.

- The `allocate` clause, except when the `allocator` modifier is a constant expression with value `omp_default_mem_alloc` and no `align` modifier has been specified. (In that case, the normal `malloc` allocation is used.)
- The `allocate` directive for variables in static memory; while the alignment is honored, the normal static memory is used.
- Using the `allocate` directive for automatic/stack variables, except when the `allocator` clause is a constant expression with value `omp_default_mem_alloc` and no `align` clause has been specified. (In that case, the normal allocation is used: stack allocation and, sometimes for Fortran, also `malloc` [depending on flags such as `-fstack-arrays`].)
- In Fortran, the `allocators` directive and the executable `allocate` directive for Fortran pointers and allocatables is supported, but requires that files containing those directives has to be compiled with `-fopenmp-allocators`. Additionally, all files that might explicitly or implicitly deallocate memory allocated that way must also be compiled with that option.
- The used alignment is the maximum of the value the `align` clause and the alignment of the type after honoring, if present, the `aligned` (`GNU::aligned`) attribute and C's `_Alignas` and C++'s `alignas`. However, the `align` clause of the `allocate` directive has no effect on the value of C's `_Alignof` and C++'s `alignof`.

GCC supports the following predefined allocators and predefined memory spaces:

Predefined allocators	**Associated predefined memory spaces**
omp_default_mem_alloc	omp_default_mem_space
omp_large_cap_mem_alloc	omp_large_cap_mem_space
omp_const_mem_alloc	omp_const_mem_space
omp_high_bw_mem_alloc	omp_high_bw_mem_space
omp_low_lat_mem_alloc	omp_low_lat_mem_space
omp_cgroup_mem_alloc	omp_low_lat_mem_space (implementation defined)
omp_pteam_mem_alloc	omp_low_lat_mem_space (implementation defined)
omp_thread_mem_alloc	omp_low_lat_mem_space (implementation defined)
ompx_gnu_pinned_mem_alloc	omp_default_mem_space (GNU extension)
ompx_gnu_managed_mem_alloc	ompx_gnu_managed_mem_space (GNU extension)

Each predefined allocator, including `omp_null_allocator`, has a corresponding allocator class template that meet the C++ allocator completeness requirements. These are located in the `omp::allocator` namespace, and the `ompx::allocator` namespace for gnu extensions. This allows the allocator-aware C++ standard library containers to use OpenMP allocation routines; for instance:

```
std::vector<int, omp::allocator::cgroup_mem<int>> vec;
```

The following allocator templates are supported:

Predefined allocators	**Associated allocator template**
omp_null_allocator	omp::allocator::null_allocator

omp_default_mem_alloc	omp::allocator::default_mem
omp_large_cap_mem_alloc	omp::allocator::large_cap_mem
omp_const_mem_alloc	omp::allocator::const_mem
omp_high_bw_mem_alloc	omp::allocator::high_bw_mem
omp_low_lat_mem_alloc	omp::allocator::low_lat_mem
omp_cgroup_mem_alloc	omp::allocator::cgroup_mem
omp_pteam_mem_alloc	omp::allocator::pteam_mem
omp_thread_mem_alloc	omp::allocator::thread_mem
ompx_gnu_pinned_mem_alloc	ompx::allocator::gnu_pinned_mem
ompx_gnu_managed_mem_alloc	ompx::allocator::gnu_managed_mem

The following traits are available when constructing a new allocator; if a trait is not specified or with the value `default`, the specified default value is used for that trait. The predefined allocators use the default values of each trait, except that the `omp_cgroup_mem_alloc`, `omp_pteam_mem_alloc`, and `omp_thread_mem_alloc` allocators have the `access` trait set to `cgroup`, `pteam`, and `thread`, respectively. For each trait, a named constant prefixed by `omp_atk_` exists; for each non-numeric value, a named constant prefixed by `omp_atv_` exists.

Trait	**Allowed values**	**Default value**
`sync_hint`	`contended`, `uncontended`, `serialized`, `private`	`contended`
`alignment`	Positive integer being a power of two	1 byte
`access`	`all`, `cgroup`, `pteam`, `thread`	`all`
`pool_size`	Positive integer (bytes)	See below.
`fallback`	`default_mem_fb`, `null_fb`, `abort_fb`, `allocator_fb`	See below
`fb_data`	*allocator handle*	(none)
`pinned`	`true`, `false`	See below
`partition`	`environment`, `nearest`, `blocked`, `interleaved`	`environment`

For the `fallback` trait, the default value is `null_fb` for the `omp_default_mem_alloc` allocator and any allocator that is associated with device memory; for all other allocators, it is `default_mem_fb` by default.

For the `pinned` trait, the default value is `true` for predefined allocator `ompx_gnu_pinned_mem_alloc` (a GNU extension), and `false` for all others.

The following description applies to the initial device (the host) and largely also to non-host devices; for the latter, also see Chapter 12 [Offload-Target Specifics], page 113.

For the memory spaces, the following applies:

- `omp_default_mem_space` is supported
- `omp_const_mem_space` maps to `omp_default_mem_space`
- `omp_low_lat_mem_space` is only available on supported devices, and maps to `omp_default_mem_space` otherwise.
- `omp_large_cap_mem_space` maps to `omp_default_mem_space`, unless the memkind library is available
- `omp_high_bw_mem_space` maps to `omp_default_mem_space`, unless the memkind library is available

- ompx_gnu_managed_mem_space is a GNU extension that provides managed memory accessible by both host and devices. The memory space is available if the offload target associated with the *default-device-var* ICV supports managed memory (see Chapter 12 [Offload-Target Specifics], page 113). This memory is accessible by both the host and the device at the same address, so it need not be mapped with map clauses. Instead, use the is_device_ptr clause or has_device_addr clause to indicate that the pointer is already accessible on the device. If managed memory is not supported by the default device, as configured at the moment the allocator is called, then the allocator will use the fall-back setting. If the default device is configured differently when the memory is freed, via omp_free or omp_realloc, the result may be undefined.

On Linux systems, where the memkind library (https://github.com/memkind/memkind) (libmemkind.so.0) is available at runtime and the respective memkind kind is supported, it is used when creating memory allocators requesting

- the partition trait interleaved except when the memory space is omp_large_cap_mem_space (uses MEMKIND_HBW_INTERLEAVE)
- the memory space is omp_high_bw_mem_space (uses MEMKIND_HBW_PREFERRED)
- the memory space is omp_large_cap_mem_space (uses MEMKIND_DAX_KMEM_ALL or, if not available, MEMKIND_DAX_KMEM)

On Linux systems, where the numa library (https://github.com/numactl/numactl) (libnuma.so.1) is available at runtime, it used when creating memory allocators requesting

- the partition trait nearest, except when both the libmemkind library is available and the memory space is either omp_large_cap_mem_space or omp_high_bw_mem_space

Note that the numa library will round up the allocation size to a multiple of the system page size; therefore, consider using it only with large data or by sharing allocations via the pool_size trait. Furthermore, the Linux kernel does not guarantee that an allocation will always be on the nearest NUMA node nor that after reallocation the same node will be used. Note additionally that, on Linux, the default setting of the memory placement policy is to use the current node; therefore, unless the memory placement policy has been overridden, the partition trait environment (the default) will be effectively a nearest allocation.

Additional notes regarding the traits:

- The pinned trait is supported on Linux hosts, but is usually subject to the OS ulimit/rlimit locked memory settings (see Chapter 12 [Offload-Target Specifics], page 113, for exceptions). The implementation uses a custom allocator to try to use as few memory pages as possible. At present, freed pinned memory is not returned to the OS (although it may be reused by subsequent pinned allocations).
- The default for the pool_size trait is no pool and for every (re)allocation the associated library routine is called, which might internally use a memory pool. Currently, the same applies when a pool_size has been specified, except that once allocations exceed the the pool size, the action of the fallback trait applies.
- For the partition trait, the partition part size will be the same as the requested size (i.e. interleaved or blocked has no effect), except for interleaved when the memkind library is available. Furthermore, for nearest and unless the numa library is available, the memory might not be on the same NUMA node as thread that allocated

the memory; on Linux, this is in particular the case when the memory placement policy is set to preferred.

- The `access` trait has no effect such that memory is always accessible by all threads. (Except on supported no-host devices.)
- The `sync_hint` trait has no effect.

See also: Chapter 12 [Offload-Target Specifics], page 113,

12 Offload-Target Specifics

The following sections present notes on the offload-target specifics

12.1 AMD Radeon (GCN)

On the hardware side, there is the hierarchy (fine to coarse):

- work item (thread)
- wavefront
- work group
- compute unit (CU)

All OpenMP and OpenACC levels are used, i.e.

- OpenMP's simd and OpenACC's vector map to work items (thread)
- OpenMP's threads ("parallel") and OpenACC's workers map to wavefronts
- OpenMP's teams and OpenACC's gang use a threadpool with the size of the number of teams or gangs, respectively.

The used sizes are

- Number of teams is the specified `num_teams` (OpenMP) or `num_gangs` (OpenACC) or otherwise the number of CU. It is limited by two times the number of CU.
- Number of wavefronts is 4 for gfx900 and 16 otherwise; `num_threads` (OpenMP) and `num_workers` (OpenACC) overrides this if smaller.
- The wavefront has 102 scalars and 64 vectors
- Number of workitems is always 64
- The hardware permits maximally 40 workgroups/CU and 16 wavefronts/workgroup up to a limit of 40 wavefronts in total per CU.
- 80 scalars registers and 24 vector registers in non-kernel functions (the chosen procedure-calling API).
- For the kernel itself: as many as register pressure demands (number of teams and number of threads, scaled down if registers are exhausted)

The implementation remark:

- I/O within OpenMP target regions and OpenACC compute regions is supported using the C library `printf` functions and the Fortran `print`/`write` statements.
- Reverse offload regions (i.e. `target` regions with `device(ancestor:1)`) are processed serially per `target` region such that the next reverse offload region is only executed after the previous one returned.
- OpenMP code that has a `requires` directive with `self_maps` or `unified_shared_memory` is only supported if all AMD GPUs have the `HSA_AMD_SYSTEM_INFO_SVM_ACCESSIBLE_BY_DEFAULT` property; for discrete GPUs, this may require setting the `HSA_XNACK` environment variable to '`1`'; for systems with both an APU and a discrete GPU that does not support XNACK, consider using `ROCR_VISIBLE_DEVICES` to enable only the APU. If not supported, all AMD GPU devices are removed from the list of available devices ("host fallback").

- The available stack size can be changed using the GCN_STACK_SIZE environment variable; the default is 32 kiB per thread.
- Low-latency memory (omp_low_lat_mem_space) is supported when the the access trait is set to cgroup. The default pool size is automatically scaled to share the 64 kiB LDS memory between the number of teams configured to run on each compute-unit, but may be adjusted at runtime by setting environment variable GOMP_GCN_LOWLAT_POOL=*bytes*.
- omp_low_lat_mem_alloc cannot be used with true low-latency memory because the definition implies the omp_atv_all trait; main graphics memory is used instead.
- omp_cgroup_mem_alloc, omp_pteam_mem_alloc, and omp_thread_mem_alloc, all use low-latency memory as first preference, and fall back to main graphics memory when the low-latency pool is exhausted.
- Pinned memory allocated using omp_alloc with the ompx_gnu_pinned_mem_alloc allocator or the pinned trait is obtained via the CUDA API when an NVPTX device is present. This provides a performance boost for NVPTX offload code and also allows unlimited use of pinned memory regardless of the OS ulimit/rlimit settings.
- Managed memory allocated with the OpenMP ompx_gnu_managed_mem_alloc allocator or in the ompx_gnu_managed_mem_space is not currently supported for AMD GPU devices; attempting to use it in an allocator will trigger the fall-back trait.
- The OpenMP routines omp_target_memcpy_rect and omp_target_memcpy_rect_async and the target update directive for non-contiguous list items use the 3D memory-copy function of the HSA library. Higher dimensions call this functions in a loop and are therefore supported.
- The unique identifier (UID), used with OpenMP's API UID routines, is the value returned by the HSA runtime library for HSA_AMD_AGENT_INFO_UUID. For GPUs, it is currently 'GPU-' followed by 16 lower-case hex digits, yielding a string like GPU-f914a2142fc3413a. The output matches the one used by rocminfo.

12.1.1 OpenMP interop – Foreign-Runtime Support for AMD GPUs

On AMD GPUs, the foreign runtimes are HIP (C++ Heterogeneous-Compute Interface for Portability) and HSA (Heterogeneous System Architecture), where HIP is the default. The interop object is created using OpenMP's interop directive or, implicitly, when invoking a declare variant procedure that has the append_args clause. In either case, the prefer_type modifier determines whether HIP or HSA is used.

When specifying the targetsync modifier: For HIP, a stream is created using hipStreamCreate. For HSA, a queue is created of type HSA_QUEUE_TYPE_MULTI with a queue size of 64.

Invoke the Section 3.11 [Interoperability Routines], page 46, on an interop object to obtain the following properties. For properties with integral (int), pointer (ptr), or string (str) data type, call omp_get_interop_int, omp_get_interop_ptr, or omp_get_interop_str, respectively. Note that device_num is the OpenMP device number while device is the HIP device number or HSA device handle.

When using HIP with C and C++, the __HIP_PLATFORM_AMD__ preprocessor macro must be defined before including the HIP header files.

For the API routine call, add the prefix `omp_ipr_` to the property name; for instance:

```
omp_interop_rc_t ret;
int device_num = omp_get_interop_int (my_interop_obj, omp_ipr_device_num, &ret);
```

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	`1`
`vendor_name`	`const char *`	str	`"amd"`
`device_num`	`int`	int	
`platform`	N/A		
`device`	`hipDevice_t`	int	
`device_context`	`hipCtx_t`	ptr	
`targetsync`	`hipStream_t`	ptr	

Available properties for an HSA interop object:

Property	**C data type**	**API routine**	**value (if constant)**
`fr_id`	`omp_interop_fr_t`	int	`omp_fr_hsa`
`fr_name`	`const char *`	str	`"hsa"`
`vendor`	`int`	int	`1`
`vendor_name`	`const char *`	str	`"amd"`
`device_num`	`int`	int	
`platform`	N/A		
`device`	`hsa_agent *`	ptr	
`device_context`	N/A		
`targetsync`	`hsa_queue *`	ptr	

12.2 nvptx

On the hardware side, there is the hierarchy (fine to coarse):

- thread
- warp
- thread block
- streaming multiprocessor

All OpenMP and OpenACC levels are used, i.e.

- OpenMP's simd and OpenACC's vector map to threads
- OpenMP's threads ("parallel") and OpenACC's workers map to warps
- OpenMP's teams and OpenACC's gang use a threadpool with the size of the number of teams or gangs, respectively.

The used sizes are

- The `warp_size` is always 32
- CUDA kernel launched: `dim={#teams,1,1}, blocks={#threads,warp_size,1}`.

- The number of teams is limited by the number of blocks the device can host simultaneously.

Additional information can be obtained by setting the environment variable to `GOMP_DEBUG=1` (very verbose; grep for `kernel.*launch` for launch parameters).

GCC generates generic PTX ISA code, which is just-in-time compiled by CUDA, which caches the JIT in the user's directory (see CUDA documentation; can be tuned by the environment variables `CUDA_CACHE_{DISABLE,MAXSIZE,PATH}`.

Note: While PTX ISA is generic, the `-mptx=` and `-march=` commandline options still affect the used PTX ISA code and, thus, the requirements on CUDA version and hardware.

The implementation remark:

- I/O within OpenMP target regions and OpenACC compute regions is supported using the C library `printf` functions. Additionally, the Fortran `print`/`write` statements are supported within OpenMP target regions, but not yet within OpenACC compute regions.
- Compilation OpenMP code that contains `requires reverse_offload` requires at least `-march=sm_35`, compiling for `-march=sm_30` is not supported.
- For code containing reverse offload (i.e. `target` regions with `device(ancestor:1)`), there is a slight performance penalty for *all* target regions, consisting mostly of shutdown delay Per device, reverse offload regions are processed serially such that the next reverse offload region is only executed after the previous one returned.
- OpenMP code that has a `requires` directive with `self_maps` or `unified_shared_memory` runs on nvptx devices if and only if all of those support the `pageableMemoryAccess` property;[1] otherwise, all nvptx device are removed from the list of available devices ("host fallback").
- The default per-warp stack size is 128 kiB; see also `-msoft-stack` in the GCC manual.
- Low-latency memory (`omp_low_lat_mem_space`) is supported when the the `access` trait is set to `cgroup`, and libgomp has been built for PTX ISA version 4.1 or higher (such as in GCC's default configuration). The default pool size is 8 kiB per team, but may be adjusted at runtime by setting environment variable `GOMP_NVPTX_LOWLAT_POOL=`*`bytes`*. The maximum value is limited by the available hardware, and care should be taken that the selected pool size does not unduly limit the number of teams that can run simultaneously.
- `omp_low_lat_mem_alloc` cannot be used with true low-latency memory because the definition implies the `omp_atv_all` trait; main graphics memory is used instead.
- `omp_cgroup_mem_alloc`, `omp_pteam_mem_alloc`, and `omp_thread_mem_alloc`, all use low-latency memory as first preference, and fall back to main graphics memory when the low-latency pool is exhausted.
- Managed memory allocated on the host with the `ompx_gnu_managed_mem_alloc` allocator or in the `ompx_gnu_managed_mem_space` (both GNU extensions) allocate memory in the CUDA Managed Memory space using `cuMemAllocManaged`. This memory is accessible by both the host and the device at the same address, so it need not be mapped with `map` clauses. Instead, use the `is_device_ptr` clause or `has_device_addr` clause

[1] `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#um-requirements`

to indicate that the pointer is already accessible on the device. The CUDA runtime will automatically handle data migration between host and device as needed. If managed memory is not supported by the default device, as configured at the moment the allocator is called, then the allocator will use the fall-back setting. If the default device is configured differently when the memory is freed, via `omp_free` or `omp_realloc`, the result may be undefined.

- The OpenMP routines `omp_target_memcpy_rect` and `omp_target_memcpy_rect_async` and the `target update` directive for non-contiguous list items use the 2D and 3D memory-copy functions of the CUDA library. Higher dimensions call those functions in a loop and are therefore supported.
- The unique identifier (UID), used with OpenMP's API UID routines, consists of the '`GPU-`' prefix followed by the 16-bytes UUID as returned by the CUDA runtime library. This UUID is output in grouped lower-case hex digits; the grouping of those 32 digits is: 8 digits, hyphen, 4 digits, hyphen, 4 digits, hyphen, 16 digits. This leads to a string like `GPU-a8081c9e-f03e-18eb-1827-bf5ba95afa5d`. The output matches the format used by `nvidia-smi`.

12.2.1 OpenMP `interop` – Foreign-Runtime Support for Nvidia GPUs

On Nvidia GPUs, the foreign runtimes APIs are the CUDA runtime API, the CUDA driver API, and HIP, the C++ Heterogeneous-Compute Interface for Portability that is—on CUDA-based systems—a very thin layer on top of the CUDA API. By default, CUDA is used. The interop object is created using OpenMP's `interop` directive or, implicitly, when invoking a `declare variant` procedure that has the `append_args` clause. In either case, the `prefer_type` modifier determines whether CUDA, CUDA driver, or HSA is used.

When specifying the `targetsync` modifier, a CUDA stream is created using the `CU_STREAM_DEFAULT` flag.

Invoke the Section 3.11 [Interoperability Routines], page 46, on an interop object to obtain the following properties. For properties with integral (int), pointer (ptr), or string (str) data type, call `omp_get_interop_int`, `omp_get_interop_ptr`, or `omp_get_interop_str`, respectively. Note that `device_num` is the OpenMP device number while `device` is the CUDA, CUDA Driver, or HIP device number.

When using HIP with C and C++, the `__HIP_PLATFORM_NVIDIA__` preprocessor macro must be defined before including the HIP header files.

For the API routine call, add the prefix `omp_ipr_` to the property name; for instance:

```
omp_interop_rc_t ret;
int device_num = omp_get_interop_int (my_interop_obj, omp_ipr_device_num, &ret);
```

Available properties for a CUDA runtime API interop object:

Property	**C data type**	**API routine**	**value (if constant)**
`fr_id`	`omp_interop_fr_t`	int	`omp_fr_cuda`
`fr_name`	`const char *`	str	`"cuda"`
`vendor`	`int`	int	`11`
`vendor_name`	`const char *`	str	`"nvidia"`
`device_num`	`int`	int	

platform	N/A		
device	int	int	
device_context	N/A		
targetsync	cudaStream_t	ptr	

Available properties for a CUDA driver API interop object:

Property	**C data type**	**API routine**	**value (if constant)**
fr_id	omp_interop_fr_t	int	omp_fr_cuda_driver
fr_name	const char *	str	"cuda_driver"
vendor	int	int	11
vendor_name	const char *	str	"nvidia"
device_num	int	int	
platform	N/A		
device	CUdevice	int	
device_context	CUcontext	ptr	
targetsync	CUstream	ptr	

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	

13 The libgomp ABI

The following sections present notes on the external ABI as presented by libgomp. Only maintainers should need them.

13.1 Implementing MASKED and MASTER construct

```
if (omp_get_thread_num () == thread_num)
  block
```

Hereby, *thread_num* has the value of the argument to the `filter` clause or zero if not specified.

Alternately, we generate two copies of the parallel subfunction and only include this in the version run by the *thread_num* thread. Surely this is not worthwhile though...

13.2 Implementing CRITICAL construct

Without a specified name,

```
void GOMP_critical_start (void);
void GOMP_critical_end (void);
```

so that we don't get COPY relocations from libgomp to the main application.

With a specified name, use omp_set_lock and omp_unset_lock with name being transformed into a variable declared like

```
omp_lock_t gomp_critical_user_<name> __attribute__((common))
```

Ideally the ABI would specify that all zero is a valid unlocked state, and so we wouldn't need to initialize this at startup.

13.3 Implementing ATOMIC construct

The target should implement the `__sync` builtins.

Failing that we could add

```
void GOMP_atomic_enter (void)
void GOMP_atomic_exit (void)
```

which reuses the regular lock code, but with yet another lock object private to the library.

13.4 Implementing FLUSH construct

Expands to the `__sync_synchronize` builtin.

13.5 Implementing BARRIER construct

```
void GOMP_barrier (void)
```

13.6 Implementing THREADPRIVATE construct

In _most_ cases we can map this directly to `__thread`. Except that OMP allows constructors for C++ objects. We can either refuse to support this (how often is it used?) or we can implement something akin to .ctors.

Even more ideally, this ctor feature is handled by extensions to the main pthreads library. Failing that, we can have a set of entry points to register ctor functions to be called.

13.7 Implementing PRIVATE clause

In association with a PARALLEL, or within the lexical extent of a PARALLEL block, the variable becomes a local variable in the parallel subfunction.

In association with FOR or SECTIONS blocks, create a new automatic variable within the current function. This preserves the semantic of new variable creation.

13.8 Implementing FIRSTPRIVATE LASTPRIVATE COPYIN and COPYPRIVATE clauses

This seems simple enough for PARALLEL blocks. Create a private struct for communicating between the parent and subfunction. In the parent, copy in values for scalar and "small" structs; copy in addresses for others TREE_ADDRESSABLE types. In the subfunction, copy the value into the local variable.

It is not clear what to do with bare FOR or SECTION blocks. The only thing I can figure is that we do something like:

```
#pragma omp for firstprivate(x) lastprivate(y)
for (int i = 0; i < n; ++i)
  body;
```

which becomes

```
{
  int x = x, y;

  // for stuff

  if (i == n)
    y = y;
}
```

where the "x=x" and "y=y" assignments actually have different uids for the two variables, i.e. not something you could write directly in C. Presumably this only makes sense if the "outer" x and y are global variables.

COPYPRIVATE would work the same way, except the structure broadcast would have to happen via SINGLE machinery instead.

13.9 Implementing REDUCTION clause

The private struct mentioned in the previous section should have a pointer to an array of the type of the variable, indexed by the thread's *team_id*. The thread stores its final value into the array, and after the barrier, the primary thread iterates over the array to collect the values.

13.10 Implementing PARALLEL construct

```
#pragma omp parallel
{
  body;
}
```

becomes

```
void subfunction (void *data)
{
```

```
  use data;
  body;
}

setup data;
GOMP_parallel_start (subfunction, &data, num_threads);
subfunction (&data);
GOMP_parallel_end ();
```

```
void GOMP_parallel_start (void (*fn)(void *), void *data, unsigned num_threads)
```

The *FN* argument is the subfunction to be run in parallel.

The *DATA* argument is a pointer to a structure used to communicate data in and out of the subfunction, as discussed above with respect to FIRSTPRIVATE et al.

The *NUM_THREADS* argument is 1 if an IF clause is present and false, or the value of the NUM_THREADS clause, if present, or 0.

The function needs to create the appropriate number of threads and/or launch them from the dock. It needs to create the team structure and assign team ids.

```
void GOMP_parallel_end (void)
```

Tears down the team and returns us to the previous `omp_in_parallel()` state.

13.11 Implementing FOR construct

```
#pragma omp parallel for
for (i = lb; i <= ub; i++)
  body;
```

becomes

```
void subfunction (void *data)
{
  long _s0, _e0;
  while (GOMP_loop_static_next (&_s0, &_e0))
  {
    long _e1 = _e0, i;
    for (i = _s0; i < _e1; i++)
      body;
  }
  GOMP_loop_end_nowait ();
}

GOMP_parallel_loop_static (subfunction, NULL, 0, lb, ub+1, 1, 0);
subfunction (NULL);
GOMP_parallel_end ();
```

```
#pragma omp for schedule(runtime)
for (i = 0; i < n; i++)
  body;
```

becomes

```
{
  long i, _s0, _e0;
  if (GOMP_loop_runtime_start (0, n, 1, &_s0, &_e0))
    do {
      long _e1 = _e0;
      for (i = _s0, i < _e0; i++)
        body;
    } while (GOMP_loop_runtime_next (&_s0, _&e0));
```

```
  GOMP_loop_end ();
}
```

Note that while it looks like there is trickiness to propagating a non-constant STEP, there isn't really. We're explicitly allowed to evaluate it as many times as we want, and any variables involved should automatically be handled as PRIVATE or SHARED like any other variables. So the expression should remain evaluable in the subfunction. We can also pull it into a local variable if we like, but since its supposed to remain unchanged, we can also not if we like.

If we have SCHEDULE(STATIC), and no ORDERED, then we ought to be able to get away with no work-sharing context at all, since we can simply perform the arithmetic directly in each thread to divide up the iterations. Which would mean that we wouldn't need to call any of these routines.

There are separate routines for handling loops with an ORDERED clause. Bookkeeping for that is non-trivial...

13.12 Implementing ORDERED construct

```
void GOMP_ordered_start (void)
void GOMP_ordered_end (void)
```

13.13 Implementing SECTIONS construct

A block as

```
#pragma omp sections
{
  #pragma omp section
  stmt1;
  #pragma omp section
  stmt2;
  #pragma omp section
  stmt3;
}
```

becomes

```
for (i = GOMP_sections_start (3); i != 0; i = GOMP_sections_next ())
  switch (i)
    {
    case 1:
      stmt1;
      break;
    case 2:
      stmt2;
      break;
    case 3:
      stmt3;
      break;
    }
GOMP_barrier ();
```

13.14 Implementing SINGLE construct

A block like

```
#pragma omp single
```

```
{
  body;
}
```

becomes

```
if (GOMP_single_start ())
  body;
GOMP_barrier ();
```

while

```
#pragma omp single copyprivate(x)
  body;
```

becomes

```
datap = GOMP_single_copy_start ();
if (datap == NULL)
  {
    body;
    data.x = x;
    GOMP_single_copy_end (&data);
  }
else
  x = datap->x;
GOMP_barrier ();
```

13.15 Implementing OpenACC's PARALLEL construct

```
void GOACC_parallel ()
```

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.

GNU General Public License

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

   Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   a. The work must carry prominent notices stating that you modified it, and giving a relevant date.
   b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
   c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
   d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source.

The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

   "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

   When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

   Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
   b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
   c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or
e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance.

However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) year name of author

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or (at
your option) any later version.

This program is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see https://www.gnu.org/licenses/.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
program Copyright (C) year name of author
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `https://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `https://www.gnu.org/licenses/why-not-lgpl.html`.

GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `https://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

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

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
  with the Invariant Sections being list their titles, with
  the Front-Cover Texts being list, and with the Back-Cover Texts
  being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Funding Free Software

If you want to have more free software a few years from now, it makes sense for you to help encourage people to contribute funds for its development. The most effective approach known is to encourage commercial redistributors to donate.

Users of free software systems can boost the pace of development by encouraging for-a-fee distributors to donate part of their selling price to free software developers—the Free Software Foundation, and others.

The way to convince distributors to do this is to demand it and expect it from them. So when you compare distributors, judge them partly by how much they give to free software development. Show distributors they must compete to be the one who gives the most.

To make this approach work, you must insist on numbers that you can compare, such as, "We will donate ten dollars to the Frobnitz project for each disk sold." Don't be satisfied with a vague promise, such as "A portion of the profits are donated," since it doesn't give a basis for comparison.

Even a precise fraction "of the profits from this disk" is not very meaningful, since creative accounting and unrelated business decisions can greatly alter what fraction of the sales price counts as profit. If the price you pay is $50, ten percent of the profit is probably less than a dollar; it might be a few cents, or nothing at all.

Some redistributors do development work themselves. This is useful too; but to keep everyone honest, you need to inquire how much they do, and what kind. Some kinds of development make much more long-term difference than others. For example, maintaining a separate version of a program contributes very little; maintaining the standard version of a program for the whole community contributes much. Easy new ports contribute little, since someone else would surely do them; difficult ports such as adding a new CPU to the GNU Compiler Collection contribute more; major new features or packages contribute the most.

By establishing the idea that supporting further development is "the proper thing to do" when distributing free software for a fee, we can assure a steady flow of resources into making more free software.

Library Index

A

E

F

I