Directives

An OpenMP executable directive applies to the succeeding structured block or an OpenMP construct. Each directive starts with #pragma omp. The remainder of the directive follows the conventions of the C and C++ standards for compiler directives. A structured-block is a single statement or a compound statement with a single entry at the top and a single exit at the bottom.

parallel [2.5] [2.4]
Forms a team of threads and starts parallel execution.

#pragma omp parallel [clause[ [, ] clause] ...]
structured-block
clause:
- if (scalar-expression)
- num_threads (integer-expression)
- default (shared | none)
- private (list)
- firstprivate (list)
- shared (list)
- copyin (list)
- reduction (reduction-identifier: list)
- proc_bind (master | close | spread)

loop [2.7.1] [2.5.1]
Specifies that the iterations of associated loops will be executed in parallel by threads in the team in the context of their implicit tasks.

#pragma omp for [clause[ [, ] clause] ...]
for-loops
clause:
- private (list)
- firstprivate (list)
- lastprivate (list)
- reduction (reduction-identifier: list)
- schedule (kind[, chunk_size])
- collapse(n)
- ordered
- nowait

kind:
- static: Iterations are divided into chunks of size chunk_size and assigned to threads in the team in round-robin fashion in order of thread number.
- dynamic: Each thread executes a chunk of iterations then requests another chunk until none remain.
- guided: Each thread executes a chunk of iterations then requests another chunk until no chunks remain to be assigned.
- auto: The decision regarding scheduling is delegated to the compiler and/or runtime system.
- runtime: The schedule and chunk size are taken from the run-sched-var ICV.

sections [2.7.2] [2.5.2]
A noniterative worksharing construct that contains a set of structured blocks that are to be distributed among and executed by the threads in a team.

#pragma omp sections [clause[ [, ] clause] ...]
{ [#pragma omp section] [structured-block] [#pragma omp section] [structured-block] }
clause:
- private (list)
- firstprivate (list)
- lastprivate (list)
- reduction (reduction-identifier: list)
- nowait

single [2.7.3] [2.5.3]
Specifies that the associated structured block is executed by only one of the threads in the team.

#pragma omp single [clause[ [, ] clause] ...]
structured-block
clause:
- private (list)
- firstprivate (list)
- copyprivate (list)
- nowait

simd [2.8.1]
Applied to a loop to indicate that the loop can be transformed into a SIMD loop.

#pragma omp simd [clause[ [, ] clause] ...]
for-loops
clause:
- safelen (length)
- linear (list: linear-step)
- aligned (list: alignment)
- private (list)
- lastprivate (list)
- reduction (reduction-identifier: list)
- collapse(n)
- ordered
- nowait

declare simd [2.8.2]
Enables the creation of one or more versions that can process multiple arguments using SIMD instructions from a single invocation from a SIMD loop.

#pragma omp declare simd [clause[ [, ] clause] ...]
structured-block
clause:
- simdlen (length)
- linear (argument-list: constant-linear-step)
- aligned (argument-list: alignment)
- uniform (argument-list)
- inbranch
- notinbranch

loop simd [2.8.3]
Specifies that a loop that can be executed concurrently using SIMD instructions, and that those iterations will also be executed in parallel by threads in the team.

#pragma omp for simd [clause[ [, ] clause] ...]
for-loops
clause:
- Any accepted by the simd or for directives with identical meanings and restrictions.

target [data] [2.9.1, 2.9.2]
These constructs create a device data environment for the extent of the region. target also starts execution on the device.

#pragma omp target data [clause[ [, ] clause] ...]
structured-block
clause:
- device (integer-expression)
- map (map-type: list)
- if (scalar-expression)

target update [2.9.3]
Makes the corresponding list items in the device data environment consistent with their original list items, according to the specified motion clauses.

#pragma omp target update [clause[ [, ] clause] ...]
clause:
- motion-clause
- to (list)
- from (list)

declare target [2.9.4]
A declarative directive that specifies that variables and functions are mapped to a device.

#pragma omp declare target declarations-definition-seq
#pragma omp end declare target

target teams [2.9.5]
Creates a league of thread teams where the master thread of each team executes the region.

#pragma omp teams [clause[ [, ] clause] ...]
structured-block
clause:
- num_teams (integer-expression)
- thread_limit (integer-expression)
- default (shared | none)
- private (list)
- firstprivate (list)
- lastprivate (list)
- shared (list)
- reduction (reduction-identifier: list)

define [2.9.6, 2.9.7]
distribute [2.9.6, 2.9.7]
distribute specifies loops which are executed by the thread teams. distribute simd specifies loops which are executed concurrently using SIMD instructions.

#pragma omp distribute [clause[ [, ] clause] ...]
for-loops
#pragma omp target simd [clause[ [, ] clause] ...]
for-loops
clause:
- private (list)
- firstprivate (list)
- lastprivate (list)
- collapse (n)
- dist_schedule (kind, chunk_size)

define [2.9.8, 2.9.9]
distribute parallel for [simd] [2.9.8, 2.9.9]
These constructs specify a loop that can be executed in parallel [using SIMD semantics in the simd case] by multiple threads that are members of multiple teams.

#pragma omp distribute parallel for [clause[ [, ] clause] ...]
for-loops
#pragma omp target simd parallel for [clause[ [, ] clause] ...]
for-loops
clause:
- See clause for distribute
Directives (Continued)

**parallel loop** [2.10.1] [2.6.1]
Shortcut for specifying a parallel construct containing one or more associated loops and no other statements.

```c
#pragma omp parallel for [clause [, clause] ...] [for-loop]
clause: Any accepted by the parallel or for directives, except the nowait clause, with identical meanings and restrictions.
```

**parallel sections** [2.10.2] [2.6.2]
Shortcut for specifying a parallel construct containing one or more sections construct and no other statements.

```c
#pragma omp parallel sections [clause [, clause] ...]
 clause: Any clause used for parallel or sections directives, except the nowait clause, with identical meanings and restrictions.
```

**#pragma omp target teams distribute parallel for simd**
Shortcut for specifying a `target` construct containing a parallel loop SIMD construct and no other statements.

```c
#pragma omp target teams distribute parallel for simd [clause [, clause] ...]
clause: Any accepted by the parallel, for or simd directives, except the nowait clause, with identical meanings and restrictions.
```

**target teams** [2.10.5]
Shortcut for specifying a target construct containing a teams construct.

```c
#pragma omp target teams [clause [, clause] ...]
structured-block
clause: See clause for target or teams
```

**teams distribute [simd]** [2.10.6, 2.10.7]
Shortcuts for specifying a teams construct containing a structured-block [simd] construct.

```c
#pragma omp teams distribute [simd] [clause [, clause] ...]
for-loops
clause: Any used for teams or distribute [simd]
```

**target teams distribute [simd]** [2.10.8, 2.10.9]
Shortcuts for specifying a target construct containing a teams distribute [simd] construct.

```c
#pragma omp target teams distribute [simd] [clause [, clause] ...]
for-loops
clause: Any clause used for target or teams distribute [simd]
```

**teams distribute parallel for [simd]** [2.10.10, 12]
Shortcuts for specifying a teams construct containing a structured-block parallel for [simd] construct.

```c
#pragma omp teams distribute parallel for [ simd] [clause [, clause] ...]
for-loops
clause: Any clause used for teams or distribute parallel for [ simd]
```

**target teams distribute parallel for [simd]** [2.10.11, 13]
Shortcuts for specifying a target construct containing a teams distribute parallel for [simd] construct.

```c
#pragma omp target teams distribute parallel for [ simd] [clause [, clause] ...]
for-loops
clause: Any clause used for target or teams distribute parallel for [simd]
```

**task** [2.11.1] [2.7.1]
 Defines an explicit task. The data environment of the task is created according to data-sharing attribute clauses on task construct and any defaults that apply.

```c
#pragma omp task [clause [, clause] ...]
structured-block
clause: ...
```

**atomic** [2.12.6] [2.8.5]
Ensures that a specific storage location is accessed atomically. `[seq_cst]` is.

```c
#pragma omp atomic [read | write | update | capture] [seq_cst] expression-stmt
```

**flush** [2.12.7] [2.8.6]
Executes the OpenMP flush operation, which makes a thread’s temporary view of memory consistent with memory, and enforces an order on the memory operations of the variables.

```c
#pragma omp flush ([lists])
```

**ordered** [2.12.8] [2.8.7]
Specifies a structured block in a loop region that will be executed in the order of the loop iterations.

```c
#pragma omp ordered structured-block
```

**cancel** [2.13.1]
Requests cancellation of the innermost enclosing region of the type specified. The `cancel` directive may not be used in place of the statement following an `if`, `while`, `do`, `switch`, or `label`.

```c
#pragma omp cancel construct-type-clause[ [, ] if-clause]
```

**cancellation point** [2.13.2]
Introduces a user-defined cancellation point at which tasks check if cancellation of the innermost enclosing region of the type specified has been requested.

```c
#pragma omp cancellation point construct-type-clause [label][or]
```

**threadprivate** [2.14.3] [2.9.2]
Specifies that variables are replicated, with each thread having its own copy. Each copy of a threadprivate variable is initialized once prior to the first reference to that copy.

```c
#pragma omp threadprivate([list])
```

**#pragma omp declare reduction**

Declares a reduction identifier that can be used in a reduction clause.

```c
#pragma omp declare reduction(reduction-identifier : typename-list : combiner) [initializer-clause]
```

**reduction-identifier**: A base language identifier or one of the following operators: `+`, `-`, `*`, `/`, `|`, `^`, `&` and `||` in C++; this may also be an operator-function id.

```c
reduction-identifier : A list of type names
```

**initializer-clause**: An expression

```c
initializer-clause : initializer (omp_priv = initializer | function-name (argument-list))
```
Runtime Library Routines

Execution Environment Routines

**omp_set_num_threads** [3.2.1] [3.2.1]
Affects the number of threads used for subsequent parallel regions not specifying a num_threads clause, by setting the value of the first element of the nthreads-var ICV of the current task to num_threads.

```c
void omp_set_num_threads(int num_threads);
```

**omp_get_num_threads** [3.2.2] [3.2.2]
Returns the number of threads in the current team. The binding region for an omp_get_num_threads region is the innermost enclosing parallel region.

```c
int omp_get_num_threads(void);
```

**omp_get_max_threads** [3.2.3] [3.2.3]
Returns an upper bound on the number of threads that could be used to form a new team if a parallel construct without a num_threads clause were encountered after execution returns from this routine.

```c
int omp_get_max_threads(void);
```

**omp_get_thread_num** [3.2.4] [3.2.4]
Returns the thread number of the calling thread within the current team.

```c
int omp_get_thread_num(void);
```

**omp_get_num_procs** [3.2.5] [3.2.5]
Returns the number of processors that are available to the device at the time the routine is called.

```c
int omp_get_num_procs(void);
```

**omp_in_parallel** [3.2.6] [3.2.6]
Returns true if the active-levels-var ICV is greater than zero; otherwise it returns false.

```c
int omp_in_parallel(void);
```

**omp_set_dynamic** [3.2.8] [3.2.8]
This routine returns the value of the dyn-var ICV, which indicates if dynamic adjustment of the number of threads is enabled or disabled.

```c
void omp_set_dynamic(int dynamic_threads);
```

**omp_get_dynamic** [3.2.8] [3.2.8]
This routine returns the value of the dyn-var ICV, which is true if dynamic adjustment of the number of threads is enabled for the current task.

```c
int omp_get_dynamic(void);
```

**omp_get_default_device** [3.2.10] [3.2.9]
Returns the default target device.

```c
int omp_get_default_device(void);
```

**omp_set_default_device** [3.2.10] [3.2.9]
Assigning the value of the default-device-var ICV.

```c
void omp_set_default_device(int default_device);
```

**omp_get_level** [3.2.17] [3.2.16]
For the enclosing device region, returns the levels-vars ICV, which is the number of nested parallel regions that enclose the task containing the call.

```c
int omp_get_level(void);
```

**omp_get_thread_limit** [3.2.14] [3.2.13]
Returns the maximum number of active threads.

```c
int omp_get_thread_limit(void);
```

**omp_get_max_active_levels** [3.2.16] [3.2.15]
Limits the number of nested active parallel regions, by setting max-active-levels-var ICV.

```c
void omp_set_max_active_levels(int max_levels);
```

**omp_get_max_thread_level** [3.2.16] [3.2.15]
Returns the value of max-active-levels-var ICV, which determines the maximum number of nested active parallel regions.

```c
int omp_get_max_thread_level(void);
```

**omp_get_schedule** [3.2.13] [3.2.12]
Returns the value of run-sched-var ICV, which is the schedule applied when runtime schedule is used.

```c
void omp_get_schedule(  
  omp_sched_t *kind, int *modifier);
```

**omp_set_schedule**

```c
void omp_set_schedule(omp_sched_t kind, int modifier);
```

**omp_in_final** [3.2.21] [3.2.20]
Returns true if the current task is executing on the host device; otherwise, it returns false.

```c
int omp_in_final(void);
```

**omp_get_num_devices** [3.2.25]
Returns the number of target devices.

```c
int omp_get_num_devices(void);
```

**omp_get_num_teams** [3.2.26]
Returns the number of teams in the current teams region, or 1 if called from outside of a teams region.

```c
int omp_get_num_teams(void);
```

**omp_get_team_num** [3.2.27]
Returns the team number of calling thread. The team number is an integer between 0 and one less than the value returned by omp_get_num_teams, inclusive.

```c
int omp_get_team_num(void);
```

**omp_is_initial_device** [3.2.28]
Returns true if the current team is running on the host device; otherwise, it returns false.

```c
int omp_is_initial_device(void);
```

Lock Routines

General-purpose lock routines. Two types of locks are supported: simple locks and nestable locks. A nestable lock can be set multiple times by the same task before being unset; a simple lock cannot be set if it is already owned by the task trying to set it.

**Initialize lock** [3.3.1] [3.3.1]
Initialize an OpenMP lock.

```c
void omp_init_lock(omp_lock_t *lock);
```

```c
void omp_init_nest_lock(omp_nest_lock_t *lock);
```

**Destroy lock** [3.3.2] [3.3.2]
Ensure that the OpenMP lock is uninitialized.

```c
void omp_destroy_lock(omp_lock_t *lock);
```

```c
void omp_destroy_nest_lock(omp_nest_lock_t *lock);
```

**Set lock** [3.3.3] [3.3.3]
Sets an OpenMP lock. The calling task region is suspended until the lock is set.

```c
void omp_set_lock(omp_lock_t *lock);
```

```c
void omp_set_nest_lock(omp_nest_lock_t *lock);
```

**Unset lock** [3.3.4] [3.3.4]
Unsets an OpenMP lock.

```c
void omp_unset_lock(omp_lock_t *lock);
```

```c
void omp_unset_nest_lock(omp_nest_lock_t *lock);
```

**Test lock** [3.3.5] [3.3.5]
Attempt to set an OpenMP lock but do not suspend execution of the task executing the routine.

```c
int omp_test_lock(omp_lock_t *lock);
```

```c
int omp_test_nest_lock(omp_nest_lock_t *lock);
```

Timing Routines

Timing routines support a portable wall clock timer. These records elapsed time per-thread and are not guaranteed to be globally consistent across all the threads participating in an application.

**omp_get_wtime** [3.4.1] [3.4.1]
Returns elapsed wall clock time in seconds.

```c
double omp_get_wtime(void);
```

**omp_get_wtick** [3.4.2] [3.4.2]
Returns the precision of the timer (seconds between ticks) used by omp_get_wtime.

```c
double omp_get_wtick(void);
```
Environment Variables [4]

**[4.11]** OMP\_CANCELLATION policy
Sets the cancel-var ICV. policy may be true or false. If true, the effects of the cancel construct and of cancellation points are enabled and cancellation is activated

**[4.13]** OMP\_DEFAULT\_DEVICE device
Sets the default-device-var ICV that controls the default device number to use in device constructs.

**[4.12]** OMP\_DISPLAY\_ENV var
If var is TRUE, instructs the runtime to display the OpenMP version number and the value of the ICVs associated with the environment variables as name=value pairs. If var is VERBOS, the runtime may also display vendor-specific variables. If var is FALSE, no information is displayed.

**[4.3]** OMP\_DYNAMIC dynamic
Sets the dyn-var ICV. If true, the implementation may dynamically adjust the number of threads to use for executing parallel regions.

**[4.8]** OMP\_MAX\_ACTIVE\_LEVELS levels
Sets the max-active-levels-var ICV that controls the maximum number of nested active parallel regions.

**[4.5]** OMP\_NESTED nested
Sets the nest-var ICV to enable or to disable nested parallelism. Valid values for nested are true or false.

**[4.2]** OMP\_NUM\_THREADS list
Sets the ntreads-var ICV for the number of threads to use for parallel regions.

**[4.5]** OMP\_PLACES places
Sets the place-partition-var ICV that defines the OpenMP places available to the execution environment. places is an abstract name (threads, cores, sockets, or implementation-defined), or a list of non-negative numbers.

**[4.4]** OMP\_PROC\_BIND policy
Sets the value of the global bind-var ICV, which sets the thread affinity policy to be used for parallel regions at the corresponding nested level. policy can be the values true, false, or a comma-separated list of master, close, or spread in quotes.

**[4.7]** OMP\_STACKSIZE size/B | K | M | G
Sets the stack-size-var ICV that specifies the size of the stack for threads created by the OpenMP implementation. size is a positive integer that specifies stack size. If unit is not specified, size is measured in kilobytes (K).

**[4.10]** OMP\_THREAD\_LIMIT limit
Sets the thread-limit-var ICV that controls the number of threads participating in the OpenMP program.

**[4.8]** OMP\_WAIT\_POLICY policy
Sets the wait-policy-var ICV that provides a hint to an OpenMP implementation about the desired behavior of waiting threads. Valid values for policy are ACTIVE (waiting threads consume processor cycles while waiting) and PASSIVE.

---

**Data Sharing Attribute Clauses [2.14.3] [2.9.3]
Data-sharing attribute clauses apply only to variables whose names are visible in the construct on which the clause appears.

**default(shared | none)**
Explicitly determines the default data-sharing attributes of variables that are referenced in a parallel, task, or teams construct, causing all variables referenced in the construct that have implicitly determined data-sharing attributes to be shared.

**shared(list)**
Declares one or more list items to be shared by tasks generated by a parallel, task, or teams construct. The programmer must ensure that storage shared by an explicit task region does not reach the end of its lifetime before the explicit task region completes its execution.

**private(list)**
Declares one or more list items to be private to a task or a SIMD lane. Each task that references a list item that appears in a private clause in any statement in the construct receives a new list item.

**firstprivate(list)**
Declares list items to be private to a task, and initializes each of them with the value that the corresponding original item has when the construct is encountered.

**lastprivate(list)**
Declares one or more list items to be private to an implicit task or to a SIMD lane, and causes the corresponding original list item to be updated after the end of the region.

**linear(list|linear-step)**
Declares one or more list items to be private to a SIMD lane and to have a linear relationship with respect to the iteration space of a loop.

**reduction(reduction-identifier:list)**
Specifies a reduction-identifier and one or more list items. The reduction-identifier must match a previously declared reduction-identifier of the same name and type for each of the list items.

**copyin(list)**
Copies the value of the master thread’s threadprivate variable to the threadprivate variable of each other member of the team executing the parallel region.

**copyprivate(list)**
Broadcasts a value from the data environment of one implicit task to the data environments of the other implicit tasks belonging to the parallel region.

**Map Clause [2.14.5]**
**map(map-type:list)**
Map a variable from the task’s data environment to the device data environment associated with the construct.

**allocate**: On entry to the region each new corresponding list item has an undefined initial value.

**to**: On entry to the region each new corresponding list item is initialized with the original list item’s value.

**from**: On exit from the region the corresponding list item’s value is assigned to each original list item.

---

**SIMD Clauses [2.8.1]**
**safelen(length)**
If used then no two iterations executed concurrently with SIMD instructions can have a greater distance in the logical iteration space than its value.

**collapse(n)**
A constant positive integer expression that specifies how many loops are associated with the loop construct.

**simdlen(length)**
A constant positive integer expression that specifies the number of concurrent arguments of the function.

**aligned(argument-list|alignment)**
Declares one or more list items to be aligned to the specified number of bytes. alignment, if present, must be a constant positive integer expression. If no optional parameter is specified, implementation-defined default alignments for SIMD instructions on the target platforms are assumed.

**uniform(argument-list)**
Declares one or more arguments to have an invariant value for all concurrent invocations of the function in the execution of a single SIMD loop.

**inbranch**
Specifies that the function will always be called from inside a conditional statement of a SIMD loop.

**notinbranch**
Specifies that the function will never be called from inside a conditional statement of a SIMD loop.

---

Copyright © 2013 OpenMP Architecture Review Board. Permission to copy without fee all or part of this material is granted, provided the OpenMP Architecture Review Board copyright notice and the title of this document appear. Notice is given that copying is by permission of the OpenMP Architecture Review Board. Products or publications based on one or more of the OpenMP specifications must acknowledge the copyright by displaying the following statement: “OpenMP is a trademark of the OpenMP Architecture Review Board. Portions of this product/publication may have been derived from the OpenMP Language Application Program Interface Specification.”

© 2013 OpenMP ARB

OpenMP API 4.0 C/C++