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] [5.2]
Forms a team of threads and starts parallel execution.

```c
#pragma omp parallel [clause [ , clause ] ... ]
```

**sections** [2.7.2] [2.7.2]
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.

```c
#pragma omp sections [clause [ , clause ] ... ]
{
  [#pragma omp section ]
  structured-block
  [#pragma omp section ]
  structured-block
  ...
}
```

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

```c
#pragma omp for [clause [ , clause ] ... ]
```

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

```c
#pragma omp single [clause [ , clause ] ... ]
```

** simd** [2.8.3] [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.

```c
#pragma omp simd [clause [ , clause ] ... ]
```

**task** [2.9.1] [2.11.1]
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 ] ... ]
```

**taskloop** [2.9.2]
Specifies that the iterations of one or more associated loops will be executed in parallel using OpenMP tasks.

```c
#pragma omp taskloop [clause [ , clause ] ... ]
```

### Directives and Constructs for C/C++

**chunk_size**

For the `fork` directive, the `chunk_size` clause is new in OpenMP 4.5. It specifies the size of chunks of iterations that are distributed to threads for parallel execution. The `chunk_size` clause is used as follows:

```c
#pragma omp for [chunk_size]...
```

The `chunk_size` clause allows for the automatic distribution of work in parallel regions. It can be used alongside other clauses to specify how work is distributed to threads.

### Examples

#### C Code Example

```c
#include <omp.h>

int main() {
    #pragma omp parallel
    {
        #pragma omp for chunk_size: 100
        for (int i = 0; i < 1000; i++) {
            // Work to be done
        }
    }
    return 0;
}
```

#### C++ Code Example

```cpp
#include <omp.h>

int main() {
    #pragma omp parallel
    {
        #pragma omp for chunk_size: 100
        for (int i = 0; i < 1000; i++) {
            // Work to be done
        }
    }
    return 0;
}
```

These examples demonstrate how the `chunk_size` clause can be used to distribute work among threads in parallel regions. By specifying `chunk_size: 100`, each thread will execute 100 iterations of the loop in parallel, allowing for efficient parallel execution on modern multi-core processors.
Directives and Constructs for C/C++ (continued)

taskloop simd [2.9.3]  
Specifies that a loop that can be executed concurrently  
using SIMD instructions, and that those iterations will also  
be executed in parallel using OpenMP tasks.

#pragma omp taskloop simd [clause [ , clause ] ...]  
for-loops

clause: Any accepted by the simd or taskloop directives  
with identical meanings and restrictions.

taskyield [2.9.4] [2.11.2]  
Specifies that the current task can be suspended in favor  
of execution of a different task.

#pragma omp taskyield

target data [2.10.1] [2.9.1]  
Creates a device data environment for the extent of the  
region.

#pragma omp target data [clause [ , clause ] ...]  
structured-block

clause: if([ target data ] : scalar-expression)  
device(integer-expression)  
map([ [map-type-modifier[,] map-type ] ] list)  
use_device_ptr(list)

target enter data [2.10.2]  
Specifies that variables are mapped to a device data  
environment.

#pragma omp target enter data [clause [ , clause ] ...]  
structured-block

clause: if([ target enter data ] : [ scalar-expression]  
device(integer-expression)  
map([ [map-type-modifier[,] map-type ] ] list)  
nowait

target exit data [2.10.3]  
Specifies that list items are unmapped from a device data  
environment.

#pragma omp target exit data [clause [ , clause ] ...]  
structured-block

clause: if([ target exit data ] : [ scalar-expression]  
device(integer-expression)  
map([ [map-type-modifier[,] map-type ] ] list)  
depend(dependence-type : list)  
nowait

target [2.10.4] [2.9.2]  
Maps variables to a device data environment and execute  
the construct on that device.

#pragma omp target [clause [ , clause ] ...]  
structured-block

clause: if([ target ] : [ scalar-expression]  
device(integer-expression)  
firstprivate(list)  
firstprivate(list)  
map([ [map-type-modifier[,] map-type ] ] list)  
is_device_ptr(list)  
defaultmap(tofrom : scalar)  
nowait  
depend(dependence-type : list)

target update [2.10.5] [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 is motion-clause or one of:

if([ target update ] : [ scalar-expression]  
device(integer-expression)  
nowait  
depend(dependence-type : list)  
nowait  
depend(dependence-type : list)  
nowait

target enter data

distribute parallel for simd [2.10.11] [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 simd [clause [ , clause ] ...]  
for-loops

clause: Any accepted by the distribute or parallel for simd  
directives.

target parallel for [2.11.1] [2.10.1]  
Shortcut for specifying a parallel construct containing one  
or more associated loops and no other statements.

#pragma omp target parallel for [clause [ , clause ] ...]  
for-loop

clause: Any accepted by the parallel or for directives,  
except the nowait clause, with identical meanings and  
restrictions.

target parallel sections [2.11.2] [2.10.2]  
Shortcut for specifying a parallel construct containing one  
target parallel construct or sections construct and no other  
statements.

#pragma omp target parallel sections [clause [ , clause ] ...]  
structured-block

clause, with identical meanings  
for-loops

clause: Any accepted by the parallel or sections  
directives, except the nowait clause, with identical  
meanings and restrictions.

target parallel [2.11.5]  
Shortcut for specifying a target construct containing a  
parallel construct and no other statements.

#pragma omp target parallel [clause [ , clause ] ...]  
structured-block

clause: Any accepted by the target or parallel directives,  
except for copyin, with identical meanings and  
restrictions.

target parallel for [2.11.6]  
Shortcut for specifying a target construct containing a  
parallel construct and no other statements.

#pragma omp target parallel for [clause [ , clause ] ...]  
for-loops

clause: Any accepted by the target or parallel for directives,  
except for copyin, with identical meanings and  
restrictions.

target parallel for simd [2.11.7]  
Shortcut for specifying a target construct containing a  
parallel for simd construct and no other statements.

#pragma omp target parallel for simd [clause [ , clause ] ...]  
for-loops

clause: Any accepted by the target or parallel for simd  
directives, except for copyin, with identical meanings  
and restrictions.
Directives and Constructs for C/C++ (continued)

target simd [2.11.8]
Shortcut for specifying a target construct containing a
distribute simd construct and no other statements.
#pragma omp target simd [clause[,clause]...] 
for-loops
clause: Any accepted by the target or simd directives with
identical meanings and restrictions.

target teams [2.11.9] [2.10.5]
Shortcut for specifying a target construct containing a
teams construct and no other statements.
#pragma omp target teams [clause[,clause]...] 
structured-block
clause: Any accepted by the target or teams directives with
identical meanings and restrictions.

teams distribute [2.11.10] [2.10.6]
Shortcuts for specifying a teams construct containing a
distribute construct and no other statements.
#pragma omp teams distribute [clause[,clause]...] 
for-loops
clause: Any clause used for distribute parallel for simd or targets

teams distribute simd [2.11.11] [2.10.7]
Shortcuts for specifying a teams construct containing a
distribute simd construct and no other statements.
#pragma omp teams distribute simd [clause[,clause]...] 
for-loops
clause: Any clause used for teams or distribute simd, with
identical meanings and restrictions.

target teams distribute [2.11.12] [2.10.8]
Shortcuts for specifying a target construct containing a
teams distribute construct and no other statements.
#pragma omp target teams distribute [clause[,clause]...] 
for-loops
clause: Any clause used for target or teams distribute

teams distribute simd [2.11.13] [2.10.9]
Shortcuts for specifying a target construct containing a
teams distribute simd construct and no other statements.
#pragma omp target teams distribute simd [clause[,clause]...] 
for-loops
clause: Any clause used for target or teams distribute simd

teams distribute parallel for [2.11.14] [2.10.10]
Shortcuts for specifying a teams construct containing a
distribute parallel for construct and no other statements.
#pragma omp teams distribute parallel for [clause[,clause]...] 
for-loops
clause: Any clause used for teams or distribute parallel for or target

target teams distribute parallel for [2.11.15] [2.10.11]
Shortcut for specifying a target construct containing a
teams distribute parallel for construct and no other statements.
#pragma omp target teams distribute parallel for [clause[,clause]...] 
for-loops
clause: Any clause used for teams distribute parallel for or target

teams distribute parallel for simd [2.11.16] [2.10.12]
Shortcut for specifying a teams construct containing a
distribute parallel for simd construct and no other statements.
#pragma omp teams distribute parallel for simd [clause[,clause]...] 
for-loops
clause: Any clause used for teams distribute parallel for simd or teams

target teams distribute parallel for simd [2.11.17] [2.10.13]
Shortcut for specifying a target construct containing a
teams distribute parallel for simd construct and no other statements.
#pragma omp target teams distribute parallel for simd [clause[,clause]...] 
for-loops
clause: Any clause used for teams distribute parallel for simd or target

master [2.13.1] [2.12.1]
Specifies a structured block that is executed by the master
thread of the team.
#pragma omp master 
structured-block

critical [2.13.2] [2.12.2]
Restricts execution of the associated structured block to a
single thread at a time.
#pragma omp critical [name] [hint (hint-expression)] 
structured-block

barrier [2.13.3] [2.12.3]
Specifies an explicit barrier at the point at which the
construct appears.
#pragma omp barrier

taskwait [2.13.4] [2.12.4]
Specifies a wait on the completion of child tasks of the
current task.
#pragma omp taskwait

taskgroup [2.13.5] [2.12.5]
Specifies a wait on the completion of child tasks of the
current task, then waits for descendant tasks.
#pragma omp taskgroup 
structured-block

atomic [2.13.6] [2.12.6]
Ensures that a specific storage location is accessed
atomically. May take one of the following three forms:
#pragma omp atomic [seq_cst] [expression-stmt] 
constants
#pragma omp atomic [seq_cst] expression-stmt

atomic clause: read, write, update, or capture

atomic (continued)
expression-stmt may be one of:

read

write

update or is not present

capture

flush [2.13.7] [2.12.7]
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.
#pragma omp flush ([list])

ordered [2.13.8] [2.12.8]
Specifies a structured block in a loop, simd, or loop SIMD
region that will be executed in the order of the loop
iterations.
#pragma omp ordered [clause[,clause]...] 
structured-block
clause: 

threads

simd

cancel [2.14.1] [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.
#pragma omp cancel construct-type-clause [clause] 
construct-type-clause:

parallel

sections

for

taskgroup

if-clause:

if (scalar-expression)

cancellation point [2.14.2] [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 activated.
#pragma omp cancellation point construct-type-clause 
construct-type-clause:

parallel

sections

for

taskgroup

Continued
### Directives and Constructs for C/C++ (continued)

**threadprivate** [2.15.2] [2.14.2]
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)
```
- **list**: A comma-separated list of file-scope, namespace-scope, or static block-scope variables that do not have incomplete types.
- **combiner**: An expression
- **specifier**: An expression
- **if**: A list of type names
- **for**: C++
- **clause**: A base language identifier (for C), or an id-expression (for C++), or one of the following operators: +, *, &, |, ^, && and ||
- **typename-list**: A list of type names
- **type-name**: type-name-list: combiner | [initializer-clause]

### Runtime Library Routines for C/C++

Execution environment routines affect and monitor threads, processors, and the parallel environment. The library routines are external functions with “C” linkage.

**omp_get_nested** [3.2.11] [3.2.10]
Returns the value of the nest-var ICV, which indicates if nested parallelism is enabled or disabled.

```c
int omp_get_nested(void);
```

**omp_set_nested** [3.2.12] [3.2.11]
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_nested(int num_threads);
```

**omp_get_num_threads** [3.2.2] [3.2.1]
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] [3.2.2]
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] [3.2.3]
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] [3.2.4]
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] [3.2.5]
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** [3.2.7] [3.2.6]
Enables or disables dynamic adjustment of the number of threads available for the execution of subsequent parallel regions by setting the value of the dyn-var ICV.

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

**omp_get_dynamic** [3.2.8] [3.2.7]
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_cancellation** [3.2.9] [3.2.8]
Returns the value of the cancel-var ICV, which is true if cancellation is activated; otherwise it returns false.

```c
int omp_get_cancellation(void);
```

**omp_set_cancellation** [3.2.10] [3.2.9]
Enables or disables nested parallelism, by setting the nest-var ICV.

```c
void omp_set_cancellation(int nested);
```

**omp_get_level** [3.2.17] [3.2.16]
Returns the number of the enclosing parallel 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_ancestor_thread_num** [3.2.18] [3.2.17]
Returns, for a given nested level of the current thread, the thread number of the ancestor of the current thread.

```c
int omp_get_ancestor_thread_num(void);
```

**omp_get_team_size** [3.2.19] [3.2.18]
Returns, for a given nested level of the current thread, the size of the thread team to which the ancestor or the current thread belongs.

```c
int omp_get_team_size(void);
```

**omp_get_active_level** [3.2.20] [3.2.19]
Returns the value of the active-level-vars ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.

```c
int omp_get_active_level(void);
```

**omp_in_final** [3.2.21] [3.2.20]
Returns true if the routine is executed in a final task region; otherwise, it returns false.

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

**omp_get_proc_bind** [3.2.22] [3.2.21]
Returns the thread affinity policy to be used for the subsequent nested parallel regions that do not specify a proc_bind clause.

```c
void omp_get_proc_bind(void);
```

This routine returns the value of the active-level-vars ICV, which determines the number of active, nested parallel regions enclosing the task that contains the call.

```c
int omp_get_active_level(void);
```

Returns true if the routine is executed in a final task region; otherwise, it returns false.

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

Returns one of:
- `omp_proc_bind_false` = 0
- `omp_proc_bind_true` = 1
- `omp_proc_bind_master` = 2
- `omp_proc_bind_close` = 3
- `omp_proc_bind_spread` = 4

**omp_get_num_places** [3.2.23]
Returns the number of places available to the execution environment in the place list.

```c
int omp_get_num_places(void);
```

**omp_get_place_num_procs** [3.2.24]
Returns the number of processors available to the execution environment in the specified place.

```c
int omp_get_place_num_procs(int place_num);
```

**omp_get_place_proc_ids** [3.2.25]
Returns the numerical identifiers of the processors available to the execution environment in the specified place.

```c
void omp_get_place_proc_ids(int place_num, int *ids);
```

**omp_get_place_num** [3.2.26]
Returns the place number of the place to which the encountering thread is bound.

```c
int omp_get_place_num(void);
```

**omp_get_partition_num_places** [3.2.27]
Returns the number of places in the place partition of the innermost implicit task.

```c
int omp_get_partition_num_places(void);
```

**omp_get_partition_place_nums** [3.2.28]
Returns the list of place numbers corresponding to the places in the place-partition-vars ICV of the innermost implicit task.

```c
void omp_get_partition_place_nums(int *place_nums);
```
**Runtime Library Routines for C/C++ (continued)**

**omp_set_default_device** [3.2.28] [3.2.23]
Controls the default target device by assigning the value of the `default_device_var` ICV.
void `omp_set_default_device`(int device_num);

**omp_get_default_device** [3.2.30] [3.2.24]
Returns the value of the `default_device_var` ICV, which determines default target device.
int `omp_get_default_device`();

**omp_get_num_devices** [3.2.31] [3.2.25]
Returns the number of target devices.
int `omp_get_num_devices`();

**omp_get_num_teams** [3.2.32] [3.2.26]
Returns the number of teams in the current team region, or 1 if called outside of a team region.
int `omp_get_num_teams`();

**omp_get_max_team_priority** [3.2.23] [3.2.24]
Returns the priority of the thread. The priority may be different from the value returned by `omp_get_max_task_priority`.
int `omp_get_max_team_priority`();

**omp_get_default_device** [3.2.28] [3.2.23]
Returns the number of teams in the current team region.
int `omp_get_num_teams`();

**omp_get_max_task_priority** [3.2.32] [3.2.26]
Returns the maximum value that can be specified in the priority clause.
int `omp_get_max_task_priority`();

**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]
Initializes an OpenMP lock.
void `omp_init_lock`(omp_lock_t *lock);
void `omp_init_nested_lock`(omp_nest_lock_t *lock);

**Initialize lock with hint** [3.3.2]
Initializes an OpenMP lock with a hint.
void `omp_init_lock_with_hint`(omp_lock_t *lock, omp_lock_hint_t hint);
void `omp_init_nested_lock_with_hint`(omp_nest_lock_t *lock, omp_nest_lock_hint_t hint);

**Destroy lock** [3.3.3] [3.3.2]
Ensures that the OpenMP lock is uninitialized.
void `omp_destroy_lock`(omp_lock_t *lock);
void `omp_destroy_nested_lock`(omp_nest_lock_t *lock);

**Set lock** [3.3.4] [3.3.3]
Sets an OpenMP lock. The calling task region is suspended until the lock is set.
void `omp_set_lock`(omp_lock_t *lock);
void `omp_set_nested_lock`(omp_nest_lock_t *lock);

**Unset lock** [3.3.5] [3.3.4]
Unsets an OpenMP lock.
void `omp_unset_lock`(omp_lock_t *lock);
void `omp_unset_nested_lock`(omp_nest_lock_t *lock);

**Test lock** [3.3.6] [3.3.5]
Attempts to set an OpenMP lock but does not suspend execution of the task executing the routine.
int `omp_test_lock`(omp_lock_t *lock);
int `omp_test_nested_lock`(omp_nest_lock_t *lock);

**Timing Routines**
Timing routines support a portable wall clock timer. These record 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.
double `omp_get_wtime`();

double `omp_get_wtick`();

**Device Memory Routines**
Timing routines support allocation and management of pointers in the data environments of target devices.

**omp_target_alloc** [3.5.1]
Allocates memory in a device data environment.
void* `omp_target_alloc`(size_t size, int device_num);

**omp_target_free** [3.5.2]
Frees the memory device allocated by the `omp_target_alloc` routine.
void `omp_target_free`(void *device_ptr, int device_num);

**omp_target_is_present** [3.5.3]
Validates whether a host pointer has an associated device buffer on a given device.
int `omp_target_is_present`(void *ptr, int device_num);

**omp_target_memcpy** [3.5.4]
Copies memory between any combination of host and device pointers.
int `omp_target_memcpy`(void *dst, void *src, size_t length, size_t dst_offset, size_t src_offset, int dst_device_num, int src_device_num);

**omp_target_memcpy_rect** [3.5.5]
Copies a rectangular subvolume from a multi-dimensional array to another multi-dimensional array.
int `omp_target_memcpy_rect`(void *device_ptr, const size_t *src_dimensions, const size_t *srcOffsets, const size_t *src_device_num, const size_t *dst_dimensions, int dst_device_num, int src_device_num);

**omp_target_memcpy** [3.5.6]
Maps a device pointer, which may be returned from `omp_target_alloc` or implementation-defined runtime routines, to a host pointer.
int `omp_target_memcpy`(void *host_ptr, void *device_ptr, int src_device_num);

**omp_target_associate_ptr** [3.5.7]
Removes the associated pointer for a given device from a host pointer.
int `omp_target_associate_ptr`(void *host_ptr, int device_num);
Clauses

The set of clauses that is valid on a particular directive is described with the directive. Most clauses accept a comma-separated list of list items. All list items appearing in a clause must be visible, according to the scoping rules of the base language. Not all of the clauses listed in this section are valid on all directives.

If Clause [2.12]

The effect of the if clause depends on the construct to which it is applied.

if([directive-name-modifier [: scalar-expression]])

For combined or composite constructs, it only applies to the semantics of the construct named in the directive-name-modifier if one is specified. If none is specified for a combined or composite construct then the if clause applies to all constructs to which an if clause can apply.

Depend Clause [2.13.9]

Enforces additional constraints on the scheduling of tasks or loop iterations. These constraints establish dependences only between sibling tasks or between loop iterations.

depend(dependence-type: list)

Where dependence-type may be in, out, or inout:

in: The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an in, out, or inout dependence-type list.

out: The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an in, out, or inout dependence-type list.

inout: The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items in an in, out, or inout dependence-type list.

SIMD Clauses [2.8]

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.

Data Copying Clauses [2.15.4] [2.14.4]

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.

copyprivarate(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 [2.15.5] [2.14.5]

map([map-type-modifier[,][map-type: list]])

Map a variable from the task’s data environment to the device data environment associated with the construct.

map-type:

alloc: 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.

tofrom: [Default] On entry to the region each new corresponding list item is initialized with the original list item’s value, and on exit from the region the corresponding list item’s value is assigned to each original list item.

release: On exit from the region, the corresponding list item’s reference count is decremented by one.

delete: On exit from the region, the corresponding list item’s reference count is set to zero.

map-type-modifier:

Must be always.

Defaultmap Clause [2.15.5.2]

defaultmap(tofrom: scalar)

Causes all scalar variables referenced in the construct that have implicitly determined data-mapping attributes to have the tofrom map-type.

Tasking Clauses [2.9]

final(scalar-logical-expr)

The generated task will be a final task if the final expression evaluates to true.

mergeable

Specifies that the generated task is a mergeable task.

priority(priority-value)

A non-negative numerical scalar expression that specifies a hint for the priority of the generated task.

grainsize(grain-size)

Causes the number of logical loop iterations assigned to each created task to be greater than or equal to the minimum of the value of the grain-size expression and the number of logical loop iterations, but less than two times the value of the grain-size expression.

num_tasks(num-tasks)

Create as many tasks as the minimum of the num-tasks expression and the number of logical loop iterations.
Environment Variables [4]

Environment variable names are upper case, and the values assigned to them are case insensitive and may have leading and trailing white space.

[4.11] [4.13] 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 VERBOSE, 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.

[4.9] OMP_MAX_ACTIVE_LEVELS levels
Sets the max-active-levels-var ICV that controls the maximum number of nested active parallel regions.

ICV Environment Variable Values

The host and target device ICVs are initialized before any OpenMP API construct or OpenMP API routine executes. After the initial values are assigned, the values of any OpenMP environment variables that were set by the user are read and the associated ICVs for the host device are modified accordingly. The method for initializing a target device’s ICVs is implementation defined.

Table of ICV Initial Values (Table 2.1) and Ways to Modify and to Retrieve ICV Values (Table 2.2) [2.3.2-3] [2.3.2-3]

<table>
<thead>
<tr>
<th>ICV</th>
<th>Environment variable</th>
<th>Initial value</th>
<th>Ways to modify value</th>
<th>Ways to retrieve value</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>dyn-var</td>
<td>OMP_DYNAMIC</td>
<td>Initial value is implementation defined if the implementation supports dynamic adjustment of the number of threads; otherwise, the initial value is false.</td>
<td>omp_set_dynamic()</td>
<td>omp_get_dynamic()</td>
<td>Sec 4.3</td>
</tr>
<tr>
<td>nest-var</td>
<td>OMP_NESTED</td>
<td>false</td>
<td>omp_set_nested()</td>
<td>omp_get_nested()</td>
<td>Sec 4.6</td>
</tr>
<tr>
<td>nthreads-var</td>
<td>OMP_NUM_THREADS</td>
<td>Implementation defined. The value of this ICV is a list.</td>
<td>omp_set_num_threads()</td>
<td>omp_get_max_threads()</td>
<td>Sec 4.2</td>
</tr>
<tr>
<td>run-sched-var</td>
<td>OMP_SCHEDULE</td>
<td>Implementation defined</td>
<td>omp_set_schedule()</td>
<td>omp_get_schedule()</td>
<td>Sec 4.1</td>
</tr>
<tr>
<td>def-sched-var</td>
<td>(none)</td>
<td>Implementation defined</td>
<td>(none)</td>
<td>(none)</td>
<td>---</td>
</tr>
<tr>
<td>bind-var</td>
<td>OMP_PROC_BIND</td>
<td>Implementation defined. The value of this ICV is a list.</td>
<td>(none)</td>
<td>omp_get_proc_bind()</td>
<td>Sec 4.4</td>
</tr>
<tr>
<td>stacksize-var</td>
<td>OMP_STACKSIZE</td>
<td>Implementation defined</td>
<td>(none)</td>
<td>(none)</td>
<td>Sec 4.7</td>
</tr>
<tr>
<td>wait-policy-var</td>
<td>OMP_WAIT_POLICY</td>
<td>Implementation defined</td>
<td>(none)</td>
<td>(none)</td>
<td>Sec 4.8</td>
</tr>
<tr>
<td>thread-limit-var</td>
<td>OMP_THREAD_LIMIT</td>
<td>Implementation defined</td>
<td>thread_limit clause</td>
<td>omp_get_thread_limit()</td>
<td>Sec 4.10</td>
</tr>
<tr>
<td>max-active-levels-var</td>
<td>OMP_MAX_ACTIVE_LEVELS</td>
<td>The initial value is the number of levels of parallelism that the implementation supports.</td>
<td>omp_set_max_active_level()</td>
<td>omp_get_max_active_level()</td>
<td>Sec 4.9</td>
</tr>
<tr>
<td>active-levels-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_active_level()</td>
<td>---</td>
</tr>
<tr>
<td>levels-var</td>
<td>(none)</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_level()</td>
<td>---</td>
</tr>
<tr>
<td>place-partition-var</td>
<td>OMP_PLACES</td>
<td>Implementation defined</td>
<td>(none)</td>
<td>omp_get_partition_num_places()</td>
<td>Sec 4.5</td>
</tr>
<tr>
<td>cancel-var</td>
<td>OMP_CANCELLATION</td>
<td>false</td>
<td>(none)</td>
<td>omp_get_cancellation()</td>
<td>Sec 4.11</td>
</tr>
<tr>
<td>default-device-var</td>
<td>OMP_DEFAULT_DEVICE</td>
<td>Implementation defined</td>
<td>omp_set_default_device()</td>
<td>omp_get_default_device()</td>
<td>Sec 4.13</td>
</tr>
<tr>
<td>max-task-priority-var</td>
<td>OMP_MAX_TASK_PRIORITY</td>
<td>zero</td>
<td>(none)</td>
<td>omp_get_max_task_priority()</td>
<td>Sec 4.14</td>
</tr>
</tbody>
</table>

© 2015 OpenMP ARB