OpenMP 5.0 API Reference Guide: Tasking

The OpenMP® API gives parallel programmers a simple and flexible interface for developing portable, scalable parallel applicatons in C/C++ and Fortran.

The OpenMP tasking features are suitable for complex applications that require to parallelize irregular algorithms. OpenMP tasks are a modern way of expressing concurrency and parallelism.

C/C++ C/C++ content | For Fortran content

Functionality new/changed in OpenMP 5.0 is in this color, and in OpenMP 4.5 is in this color.
[n.n.n] Sections in the 5.0 spec. **[n.n.n]** Sections in the 4.5 spec. • Deprecated in the 5.0 spec.

Directives and Constructs

An OpenMP executable directive applies to the succeeding structured block. A *structured-block* is an OpenMP construct or a block of executable statements with a single entry at the top and a single exit at the bottom. OpenMP directives except SIMD and **declare target** directives may not appear in **PURE** or **ELEMENTAL** procedures.

Team management constructs

parallel [2.6] [2.5]

Forms a team of threads and starts parallel execution.

C/C++
```
#pragma omp parallel [clause[ [, ]clause] ...]
    structured-block
```
For
```
!$omp parallel [clause[ [, ]clause] ...]
    structured-block
!$omp end parallel
```

clause:
private (*list*), **firstprivate** (*list*), **shared** (*list*),
reduction ([*reduction-modifier,]* *reduction-identifier*: *list*)
proc_bind (**master** | **close** | **spread**)
allocate ([*allocator :*]*list*)
C/C++ **if** ([**parallel :**] *scalar-expression*)
C/C++ **num_threads** (*integer-expression*)
C/C++ **default** (**shared** | **none**)
For **if** ([**parallel :**] *scalar-logical-expression*)
For **num_teams** (*scalar-integer-expression*)
For **default** (**shared** | **firstprivate** | **private** | **none**)

single [2.8.2] [2.7.3]

Specifies that the associated structured block is executed by only one of the threads in the team.

C/C++
```
#pragma omp single [clause[ [, ]clause] ...]
    structured-block
```
For
```
!$omp single [clause[ [, ]clause] ...]
    structured-block
!$omp end single [end_clause[ [, ]end_clause] ...]
```

clause:
private (*list*), **firstprivate** (*list*), **allocate** ([*allocator:]list*)
C/C++ **nowait**

end_clause: For
nowait

master [2.16] [2.13.1]

Specifies a structured block that is executed by the master thread of the team.

C/C++
```
#pragma omp master
    structured-block
```
For
```
!$omp master
    structured-block
!$omp end master
```

Tasking constructs

task [2.10.1] [2.9.1]

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/C++
```
#pragma omp task [clause[ [, ]clause] ...]
    structured-block
```
For
```
!$omp task [clause[ [, ]clause] ...]
    structured-block
!$omp end task
```

clause:
untied, **mergeable**
private (*list*), **firstprivate** (*list*), **shared** (*list*)
in_reduction (*reduction-identifier: list*)
depend ([*depend-modifier,]* *dependence-type* **:** *locator-list*)
priority(*priority-value*), **allocate**([*allocator:]list*)
affinity ([*aff-modifier:]* *locator-list*)
- where *aff-modifier* is **iterator**(*iterators-definition*)
detach (*event-handle*)
- where *event-handle* is of type **omp_event_handle**
C/C++ **default** (**shared** | **none**)
C/C++ **if** ([**task :**] *scalar-expression*)
C/C++ **final** (*scalar-expression*)
For **default** (**private** | **firstprivate** | **shared** | **none**)
For **if** ([**task :**] *scalar-logical-expression*)
For **final** (*scalar-logical-expression*)

taskloop [simd] [2.10.2-3] [2.9.2-3]

taskloop specifies that the iterations of one or more associated loops will be executed in parallel using OpenMP **tasks. taskloop simd** specifies that a loop can be executed concurrently using SIMD instructions, and that those iterations will also be executed in parallel using OpenMP tasks.

C/C++
```
#pragma omp taskloop [simd] [clause[ [, ]clause] ...]
    for-loops
```
For
```
!$omp taskloop [simd] [clause[ [, ]clause] ...]
    do-loops
[!$omp end taskloop [simd] ]
```

clause:
shared (*list*), **private** (*list*), **firstprivate** (*list*), **lastprivate** (*list*)
reduction ([*default ,] reduction-identifier*: *list*)
in_reduction (*reduction-identifier*: *list*)
grainsize (*grain-size*), **num_tasks** (*num-tasks*)
collapse (*n*), **priority** (*priority-value*)
untied, **mergeable**, **nogroup**, **allocate** ([*allocator:]list*)
C/C++ **if** ([**taskloop :**] *scalar-expression*)
C/C++ **default** (**shared** | **none**)
C/C++ **final** (*scalar-expr*)
For **if** ([**taskloop :**] *scalar-logical-expression*)
For **default** (**private** | **firstprivate** | **shared** | **none**)
For **final** (*scalar-logical-expr*)

taskyield [2.10.4] [2.11.2]

Specifies that the current task can be suspended in favor of execution of a different task.

C/C++
```
#pragma omp taskyield
```
For
```
!$omp taskyield
```

Synchronization constructs

taskwait [2.17.5] [2.13.4]

Specifies a wait on the completion of child tasks of the current task.

C/C++
```
#pragma omp taskwait [clause[ [, ] clause] ...]
```
For
```
!$omp taskwait [clause[ [, ] clause] ...]
```

clause:
depend ([*depend-modifier,]* *dependence-type* **:** *locator-list*)

taskgroup [2.17.6] [2.13.5]

Specifies a wait on the completion of child tasks of the current task, and waits for descendant tasks.

C/C++
```
#pragma omp taskgroup [clause[ [, ]clause] ...]
    structured-block
```
For
```
!$omp taskgroup [clause[ [, ]clause] ...]
    structured-block
!$omp end taskgroup
```

clause:
task_reduction (*reduction-identifier : list*)
allocate ([*allocator:]list*)

depobj construct

depobj [2.17.10.1]

Stand-alone directive that initalizes, updates, or destroys an OpenMP depend objects.

C/C++
```
#pragma omp depobj (depobj) clause
```
For
```
!$omp depobj (depobj) clause
```

clause:
depend (*dependence-type:locator*)
destroy, update (*dependence-type*)

Cancellation constructs

cancel [2.18.1] [2.14.1]

Requests cancellation of the innermost enclosing region of the type specified.

C/C++
```
#pragma omp cancel construct-type-clause[ [, ] if-clause]
```
For
```
!$omp cancel construct-type-clause[ [, ]if-clause]
```

construct-type-clause: **parallel**, **taskgroup**

if-clause:
C/C++ **if** ([**cancel** :] *scalar-expression*)
For **if** ([**cancel** :] *scalar-logical-expression*)

cancellation point [2.18.2] [2.14.2]

Introduces a user-defined cancellation point at which tasks check if cancellation of the innermost enclosing region of the type specified has been activated.

C/C++
```
#pragma omp cancellation point construct-type-clause
```
For
```
!$omp cancellation point construct-type-clause
```

construct-type-clause: Any accepted for *construct-type-clause* by the **cancel** construct.

declare directive

declare reduction [2.19.5.7] [2.16]

Declares a *reduction-identifier* used in a **reduction** clause.

C/C++
```
#pragma omp declare reduction (reduction-identifier :
    typename-list : combiner) [initializer-clause]
```
For
```
!$omp declare reduction (
    reduction-identifier : type-list : combiner)
    [initializer-clause]
```

type-list or *typename-list:* A list of type specifiers

initializer-clause: **initializer** (*initializer-expr*)
where *initializer-expr* is **omp_priv** = *initializer* or *function-name* (*argument-list*

reduction-identifier: C/C++
A base language identifer (for C), or an *id-expression* (for C++), or one of the following operators: +, -, *, &, |, ^, &&, and ||

combiner: C/C++
An expression

reduction-identifier: For
A base language identifier, user defined operator, or one of the following operators:
+, -, *, .and., .or., .eqv., .negv., or one of the following intrinsic procedure names: max, min, iand, ior, ieor.

combiner: For
An assignment statement or a subroutine name followed by an argument list.

parallel master [2.13.6]

Shortcut for specifying a **parallel** construct containing one **master** construct and no other statements.

C/C++
```
#pragma omp parallel master [clause[ [, ]clause] ...]
    structured-block
```

For
```
!$omp parallel master [clause[ [, ]clause] ...]
    structured-block
$omp end parallel master
```

clause:
Any clause used for **parallel** or **master**.

master taskloop [simd] [2.13.7-8]

Shortcut for specifying a **master** construct containing a **taskloop [simd]** construct and no other statements.

C/C++
```
#pragma omp master taskloop [simd] [clause[ [, ]clause] ...]
    for-loops
```

For
```
!$omp master taskloop [simd] [clause[ [, ]clause] ...]
    do-loops
[$omp end master taskloop [simd]]
```

clause:
Any clause used for **master** or **taskloop [simd]**.

parallel master taskloop [simd] [2.13.9-10]

Shortcut for specifying a **parallel** construct containing a **master taskloop [simd]** construct and no other statements.

C/C++
```
#pragma omp parallel master taskloop [simd]\
    [clause[ [, ]clause] ...]
    for-loops
```

For
```
!$omp parallel master taskloop [simd] [clause[ [, ]clause] ...]
    do-loops
[$omp end parallel master taskloop [simd]]
```

clause:
Any clause used for **parallel** or **master taskloop [simd]** directives, except the **in_reduction** clause.

Clauses

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.

Allocate Clause [2.11.4]

allocate (*[allocator:] list*)
Specifies the memory allocator to be used to obtain storage for private variables of a directive.

allocator:
C/C++ An expression of type **const omp_allocator_t***
For An integer expression of the **omp_allocator_t** *kind*

Data Sharing Attribute Clauses [2.19.4] [2.15.3]

Applies only to variables whose names are visible in the construct on which the clause appears.

shared (*list*)
Declares list items to be shared by generated tasks.

private (*list*)
Declares list items to be private to a task or a SIMD lane.

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

lastprivate-modifier: **conditional**

Depend Clause [2.17.11] [2.13.9]

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

depend (*[depend-modifier,]dependence-type* **:** *locator-list*)
dependence-modifier: **iterator** (*iterators-definition*)
dependence-type: **in**, **out**, **inout**, **mutexinoutset**, **depobj**

- **in**: The generated task will be dependent of all previously generated sibling tasks that reference at least one of the list items in an **out** or **inout** *dependence-type* list.
- **out** and **inout**: The generated task will be dependent of all previously generated sibling tasks that reference at least one of the list items in an **in**, **out**, or **inout** *dependence-type* list.
- **mutexinoutset**: If the storage location of at least one list item matches that of one appearing in a **depend** clause with an **in**, **out**, or **inout** *dependence-type* on a construct from which a sibling task was previously generated, then the generated task will be a dependent task of that sibling. If the storage location of at least one of the list items is the same as that of a list item appearing in a **depend** clause with a **mutexinoutset** *dependence-type* on a construct from which a sibling task was previously generated, then the sibling tasks will be mutually exclusive.
- **depobj**: The task dependences are derived from the **depend** clause specified in the depobj constructs that initalized dependences represented by the depend objects specified on in the **depend** clause as if the **depend** clauses of the depobj constructs were specified in the current construct.

If Clause [2.15] [2.12]

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.

if (*[directive-name-modifier :] scalar-expression*) C/C++
if (*[directive-name-modifier :] scalar-logical-expression*) For

Reduction Clauses [2.19.5]

reduction (*[reduction-modifier ,] reduction-identifier* : *list*)
Specifies a *reduction-identifier* and one or more list items.

reduction-modifier: **inscan**, **task**, **default**

reduction-identifier: C++ Either an *id-expression* or one of the following operators: **+**, **-**, *****, **&**, **|**, **^**, **&&**, **||**

reduction-identifier: C Either an *identifier* or one of the following operators: **+**, **-**, *****, **&**, **|**, **^**, **&&**, **||**

reduction-identifier: For Either a base language identifier, or a user-defined operator, or one of the following operators: **+**, **-**, *****, **.and.**, **.or.**, **.eqv.**, **.neqv.**, or one of the following intrinsic procedure names: **max**, **min**, **iand**, **ior**, **ieor**.

task_reduction (*reduction-identifier: list*)
Specifies a reduction among tasks.

reduction-identifier: Same as for **reduction**

in_reduction (*reduction-identifier*: *list*)

reduction-identifier: Same as for **reduction**

Tasking Clauses [2.10] [2.9]

affinity (*[aff-modifier:] locator-list*)
A hint to execute closely to the location of the list items. *aff-modifier* is **iterator** (*iterators-definition*).

allocate (*[allocator:]list*)
See Allocate Clause, page 9 of this guide.

collapse (*n*)
Constant positive integer expression specifying how many loops are associated with the taskloop construct.

final (*scalar-expression*) C/C++
final (*scalar-logical-expression*) For
The generated task will be a final task if the final expression evaluates to true.

firstprivate (*list*)
See Data Sharing Attribute Clauses, page 9 of this guide.

grainsize (*grain-size*)
Causes the number of logical loop iterations assigned to each created task to be >= 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.

if (*[* **task :** *] scalar-expression*) C/C++
if (*[* **task :** *] scalar-logical-expression*) For
See If Clause, page 10 of this guide.

in_reduction (*reduction-identifier*: *list*)
See Reduction Clauses in this guide.

mergeable
Specifies that the generated task is a mergeable task.

nogroup
Prevents an implicit taskgroup region to be created.

num_tasks (*num-tasks*)
Create as many tasks as the minimum of the *num-tasks* expression and the number of logical loop iterations.

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

reduction (*[default ,] reduction-identifier: list*)
See Reduction Clauses in this guide.

untied
If present, any thread in the team can resume the task region after a suspension.

Iterators

iterator [2.1.6]

Identifiers that expand to multiple values in the clause on which they appear.

iterator (*iterators-definition*)

iterators-definition:
iterator-specifier [, iterators-definition]

iterators-specifier:
[iterator-type] identifier = range-specification

iterator-type: A type name or specifier.

identifier: A base language identifier.

range-specification: *begin* : *end*[: *step*]
begin, *end*: Expressions for which their types can be converted to *iterator-type*

step: An integral expression.