OpenMP Application Program Interface (API) is a portable, scalable model that gives parallel programmers a simple and flexible interface for developing portable parallel applications. OpenMP supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures, including Unix platforms and Windows platforms.

Refers to functionality new in version 4.0.

[aaa] refers to sections in the OpenMP API specification version 4.0, and [aaa] refers to version 3.1.

### Directives

OpenMP directives are specified in Fortran by using special comments that are identified by unique sentinels. Also, a special comment form is available for conditional Fortran compilation. An OpenMP executable directive applies to the succeeding structured block. A **structured-block** is a block of executable statements with a single entry at the top and a single exit at the bottom, or an OpenMP construct. OpenMP directives may not appear in a single exit at the bottom, or an OpenMP construct. OpenMP directives may not appear in

- **parallel** [2.5] [2.4]
  Forms a team of threads and starts parallel execution.
  !$omp parallel [clause[ ] , [clause ] ... ]

- **structured-block**

  - **parallel end** clause:
    - if[scalar-logical-expression]
    - num_threads[scalar-integer-expression]
    - default[ | firstprivate | shared | none]
    - private[list]
    - firstprivate[list]
    - shared[list]
    - copyin[list]
    - reduction[reduction-identifier : list]
    - nowait

- **do** [2.7.1] [2.5.1]
  Specifies that the iterations of associated loops will be executed in parallel by threads in the team.
  !$omp do [clause[ ] , [clause ] ... ]
do-loops

  - **$omp end do [nowait ]
    clause:
      - private[list]
      - firstprivate[list]
      - lastprivate[list]
      - reduction[reduction-identifier : list]
      - schedule(kind, chunk_size)
      - collapse[ | ordered]

- **single** [2.7.3] [2.5.3]
  Specifies that the associated structured block is executed by only one of the threads in the team.
  !$omp single [clause[ ] , [clause ] ... ]

- **structured-block**

  - **$omp end single [end_clause[ ] , [end_clause ] ... ]
    clause:
      - private[list]
      - firstprivate[list]
      - end_clause: copyprivate[list]
      - nowait

- **workshare** [2.7.4] [2.5.4]
  The workshare construct divides the execution of the enclosed structured block into separate units of work, each executed only once by one thread.
  !$omp workshare structured-block

  - **$omp end workshare [nowait ]
    clause:
      - The structured block must consist of only the following:
        - array or scalar assignments
        - array or scalar assignments
        - FORALL or WHERE statements
        - FORALL, WHERE, atomic, critical, or parallel constructs

- **simd** [2.8.1]
  Applied to a loop to indicate that the loop can be transformed into a SIMD loop.
  !$omp simd [clause[ ] , [clause ] ... ]
do-loops

  - **$omp end simd [end_clause[ ] , [end_clause ] ... ]
    clause:
      - safelen(length)
      - linear[list:linear-step]
      - aligned[list:alignment]
      - private[list]
      - lastprivate[list]
      - reduction[reduction-identifier : list]
      - collapse[ | ordered]

- **declare simd** [2.8.2]
  Applied to a function or a subroutine to enable the creation of one or more versions that can process multiple arguments using SIMD instructions from a single invocation from a SIMD loop.
  !$omp declare simd [proc-name[ ] , [clause ] ... ]
do-loops

  - **$omp end declare simd [end_clause[ ] , [end_clause ] ... ]
    clause:
      - dist_schedule(kind, chunk_size)

- **target** [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.
  !$omp target [data[ ] , [clause ] ... ]

  - **$omp end target [data ]
    clause:
      - device[scalar-integer-expression]
      - map[ | map-type : list]
      - if[scalar-logical-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.
  !$omp target update [clause[ ] , [clause ] ... ]
motion-clause:
  - to[list]
  - from[list]

  - clause is motion-clause or one of:
    - device[scalar-integer-expression]
    - if[scalar-logical-expression]

- **declare target** [2.9.4]
  A declarative directive that specifies that variables and functions are mapped to a device.
  For functions and subroutines:
  !$omp declare target

  - For variables, functions and subroutines:
    - !$omp declare target [list]
      - list: A comma-separated list of named variables, procedure names, and named common blocks.

- **teams** [2.9.5]
  Creates a league of thread teams where the master thread of each team executes the region.
  !$omp teams [clause[ ] , [clause ] ... ]

  - **$omp end teams [end_clause[ ] , [end_clause ] ... ]
    clause:
      - num_teams[scalar-integer-expression]
      - thread_limit[scalar-integer-expression]
      - default[ | firstprivate | private | none]
      - private[list]
      - firstprivate[list]
      - shared[list]
      - reduction[reduction-identifier : list]

- **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.
  !$omp distribute [simd[ ] , [clause ] ... ]
do-loops

  - **$omp end distribute [simd ]
    clause (for distribute):
      - private[list]
      - firstprivate[list]
      - collapse[ ]
      - dist_schedule(kind, chunk_size)

  - clause (for distribute simd): Any of the clauses accepted by distribute or simd.
6.2 target teams distribute parallel do [simd] [2.10, 11, 13]

Shortcuts for specifying target constructs containing a teams distribute parallel do [simd] construct.

```fortran
!$omp target teams distribute parallel do [simd] &
{clause[ [, clause] ...] do-loops
!$omp end target teams distribute parallel do [simd]
```

clause: Any clause used for target or teams distribute parallel do [simd] construct.

6.3 target teams distribute parallel do [simd] [2.10, 11, 12]

These shortcuts 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.

```fortran
!$omp distribute parallel do [clause[ [, clause] ...] do-loops
!$omp end distribute parallel do [clause: Any accepted by the distribute or parallel loop [SIMD] directives.
```

parallel do [2.10.1] [2.6.1]

Shortcut for specifying a parallel construct containing one or more associated loops and no other statements.

```fortran
!$omp parallel do [clause[ [, clause] ...] do-loop
!$omp end parallel do [clause: Any accepted by the parallel or do directives.
```

parallel sections [2.10.2] [2.6.2]

Shortcut for specifying a parallel construct containing one or more sections construct and no other statements.

```fortran
!$omp parallel sections [clause[ [, clause] ...] structured-block
!$omp end parallel sections
```

clause: Any of the clauses accepted by the parallel or sections directives.

parallel workshare [2.10.3] [2.6.3]

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

```fortran
!$omp parallel workshare [clause[ [, clause] ...] structured-block
!$omp end parallel workshare
```

clause: Any of the clauses accepted by the parallel directive, with identical meanings and restrictions.

parallel do simd [2.10.4]

Shortcut for specifying a parallel construct containing one loop SIMD construct and no other statements.

```fortran
!$omp parallel do simd [clause[ [, clause] ...] do-loop
!$omp end parallel do simd
```

clause: Any accepted by the parallel, do or simd directives with identical meanings and restrictions.

target teams [2.10.5]

Shortcut for specifying a target construct containing a teams construct.

```fortran
!$omp target teams [clause[ [, clause] ...] structured-block
!$omp end target teams
```

clause: See clause for target or teams construct.

teams distribute [simd] [2.10.6, 2.10.7]

Shortcuts for specifying teams constructs containing a distribute or distribute [simd] construct.

```fortran
!$omp teams distribute [simd] [clause[ [, clause] ...] do-loops
!$omp end teams distribute [simd]
```

clause: Any clause used for teams or distribute [simd] construct.

target teams distribute [simd] [2.10.8, 2.10.9]

Shortcuts for specifying target constructs containing a teams distribute [simd] construct.

```fortran
!$omp target teams distribute [simd] [clause[ [, clause] ...] do-loops
!$omp end target teams distribute [simd]
```

clause: Any clause used for target or teams distribute [simd] construct.

teams distribute parallel do [simd] [2.10.10, 12]

Shortcuts for specifying teams constructs containing a distribute parallel loop [simd] construct.

```fortran
!$omp teams distribute parallel do [simd] [clause[ [, clause] ...] do-loops
!$omp end teams distribute parallel do [simd]
```

clause: Any clause used for teams or distribute parallel do [simd] construct.

atomic [2.12.6] [2.8.5]

Ensures a specific storage location is accessed atomically. [seq_cst] is 42. May take one of the following forms:

```fortran
!$omp atomic read (seq_cst)
!$omp atomic write (seq_cst)
```

flush [2.12.7] [2.8.6]

Makes a thread's temporary view of memory consistent with memory, and enforces an order on the memory operations of the variables.

```fortran
!$omp flush (list)
```

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.

```fortran
!$omp ordered structured-block
```

cancel [2.13.1]

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

```fortran
!$omp cancel construct-type-clause [if-clause]
```

if-clause: A comma-separated list of named variables and intrinsic procedure names: max, min, iand, ior, ieor.

```fortran
reduction-identifier: A base language identifier, user-defined operator, or one of the following operators: +, - , .AND., .OR., .EQV., .NEQV.
```

intrinsic procedure name is one of MAX, MIN, IAND, IOR, IEDR.

operator is one of +, -. , .AND., .OR., .EQV., .NEQV.

```fortran
!$omp atomic capture (seq_cst)
```

taskgroup [2.14.2] [2.9.2]

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

```fortran
!$omp cancellation point construct-type-clause
```

```fortran
taskgroup if-clause:
```

```fortran
if(clarical-logical-expression)
```

threadprivate [2.14.2] [2.9.2]

Specifies that variables are replicated, with each thread having its own copy.

```fortran
!$omp threadprivate(list)
```

list: A comma-separated list of named variables and named common blocks.

declare reduction [2.15]

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

```fortran
!$omp declare reduction(reduction-identifier : type-list : &
combiner) [initilizer-clause]
```

reduceion-identifier: A base language identifier, 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.

type-list: A list of type specifiers

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

initializer-clause: initializer (omp_priv = expression OR subroutine-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.

subroutine omp_set_num_threads(num_threads)
integer 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. If called from the sequential part of a program, this routine returns 1.

integer function omp_get_num_threads()

**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.

integer function omp_get_max_threads()

**omp_set_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.

integer function omp_set_num_procs()

**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.

logical function omp_in_parallel()

**omp_set_dynamic** [3.2.7] [3.2.7]
Enables or disables dynamic adjustment of the number of threads available for the execution of subsequent parallel regions.

subroutine omp_set_dynamic(dynamic_threads)
llogical 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.

logical function omp_get_dynamic()

**omp_get_thread_num** [3.2.4] [3.2.4]
Returns the thread number of the calling thread, within the team executing the parallel region.

integer function omp_get_thread_num()

**omp_set_num_threads** [3.2.2] [3.2.2] & **omp_set_num_procs** [3.2.5] [3.2.5] and **omp_get_num_threads** [3.2.2] [3.2.2] and **omp_get_max_threads** [3.2.3] [3.2.3] are semantically equivalent.

**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.

integer function omp_get_level()

**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.

integer function omp_get_ancestor_thread_num()

**omp_get_team_num** [3.2.27]
Returns the number of teams in the current team region, or 1 if called from outside of a teams region.

integer function omp_get_team_num()

**omp_set_default_device** [3.2.24]
Returns the value of the default-device-var ICV, which determines default target device.

integer function omp_set_default_device()

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

integer function omp_get_num_devices()

**omp_get_num_teams** [3.2.26]
Returns the number of teams in the current teams region.

integer function omp_get_num_teams()

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

integer function omp_get_team_num()

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

integer function omp_is_initial_device()

Lock Routines

General-purpose lock routines.

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

subroutine omp_init_lock(svar)
integer(kind=omp_lock_kind) svar

subroutine omp_init_nested_lock(svar)
integer(kind=omp_nest_lock_kind) svar

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

subroutine omp_destroy_lock(svar)
integer(kind=omp_lock_kind) svar

subroutine omp_destroy_nested_lock(svar)
integer(kind=omp_nest_lock_kind) svar

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

subroutine omp_set_lock(svar)
integer(kind=omp_lock_kind) svar

subroutine omp_set_nested_lock(svar)
integer(kind=omp_nest_lock_kind) svar

subroutine omp_set_default_lock(svar)
integer(kind=omp_lock_kind) svar

subroutine omp_set_default_nested_lock(svar)
integer(kind=omp_nest_lock_kind) svar

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

subroutine omp_unset_lock(svar)
integer(kind=omp_lock_kind) svar

subroutine omp_unset_nested_lock(svar)
integer(kind=omp_nest_lock_kind) svar

**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.

logical function omp_test_lock(svar)
integer(kind=omp_lock_kind) svar

function omp_test_nested_lock(svar)
integer(kind=omp_nest_lock_kind) svar

**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 precision function omp_get_wtime()

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

double precision function omp_get_wtick()
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.

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/private | firstprivate | shared | none
Explicitly determines the default data-sharing attributes of variables that are referenced in 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.

shared(list)
Declares one or more list items to be shared by tasks generated by a parallel, task, or teams construct. The corresponding nested level. policy can be the values true, false, or a comma-separated list of master, close, or spread in quotes.

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 one or more list items to be firstprivate 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 lastprivate to a SIMD 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 linear 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.

Operators for reduction (initialization values)

[Continued ->]

SIMD Clauses [2.8.1, 2.8.2]
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.

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

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

aligned(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, the default alignment that SIMD instructions in the target platforms use is 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.