Summary of OpenMP 3.0 Fortran Syntax

Download the full OpenMP API Specification at www.openmp.org.

Directives

An OpenMP executable directive applies to the succeeding structured block or an OpenMP Construct. 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.

In fixed form source files, the sentinel **!$omp**, **c$omp**, or ***$omp** introduce a directive and must start in column 1. An initial directive line must contain a space or zero in column 6, and continuation lines must have a character other than a space or zero in column 6. In free form source files, the sentinel **!$omp** introduces a directive and can appear in any column if preceded only by white space.

The **parallel** construct forms a team of threads and starts parallel execution.

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

clause:
- **if**(*scalar-expression*)
- **num_threads**(*scalar-integer-expression*
- **default**(**private**|**firstprivate**|**shared**|**none**)
- **private**(*list*)
- **firstprivate**(*list*)
- **shared**(*list*)
- **copyin**(*list*)
- **reduction**({*operator*|*intrinsic_procedure_name*}:*list*)

The loop construct specifies that iterations of loops will be distributed among and executed by the encountering team of threads.

```
!$omp do [clause[[ ,] clause] ... ]
        do-loops
[!$omp end do [nowait]]
```

clause:
- **private**(*list*)
- **firstprivate**(*list*)
- **lastprivate**(*list*)
- **reduction**({*operator*|*intrinsic_procedure_name*}:*list*)
- **schedule**(*kind[, chunk_size]*)
- **collapse**(*n*)
- **ordered**

(See applicable clauses on next page.)

Directives (continued)

The **sections** construct contains a set of structured blocks to be distributed among and executed by encountering team of threads.

```
!$omp sections [clause[[ ,] clause] ...]
        [!$omp section]
            structured-block
        [!$omp section
            structured-block ]
            ...
!$omp end sections [nowait]
```

clause:
- **private**(*list*)
- **firstprivate**(*list*)
- **lastprivate**(*list*)
- **reduction**({*operator*|*intrinsic_procedure_name*}: *list*)

The **single** construct specifies that the associated structured block is executed by only one of the threads in the team (not necessarily the master thread), in the context of its implicit task.

```
!$omp single [clause[[ ,] clause] ...]
        structured-block
!$omp end single [end_clause[[ ,] end_clause] ...]
```

clause:
- **private**(*list*)
- **firstprivate**(*list*)

end_clause:
- **copyprivate**(*list*)
- **nowait**

The **workshare** construct divides the execution of the enclosed structured block into separate units of work.

```
!$omp workshare structured-block
!$omp end workshare [nowait]
```

The combined parallel worksharing constructs are a shortcut for specifying a parallel construct containing one work-sharing construct and no other statements.

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

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

clause:
any clause from **parallel** or **sections**

Directives (continued)

The **parallel workshare** construct is a shortcut for a **parallel** construct containing one **workshare** construct and no other statements.

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

clause:
any clause from **parallel** or **sections**

The **task** construct defines an explicit task.

```
!$omp task [clause[ [ , ]clause] ...]
        structured-block
!$omp end task
```

clause:
- **if**(*scalar-logical-expression*)
- **untied**
- **default**(**private**|**firstprivate**|**shared**|**none**)
- **private**(*list*)
- **firstprivate**(*list*)
- **shared**(*list*)

The **master** construct specifies a structured block that is executed by the master thread of the team.

```
!$omp master
        structured-block
!$omp end master
```

The **critical** construct restricts execution of the associated structured block to a single thread at a time.

```
!$omp critical [(name)]
        structured-block
!$omp end critical [(name)]
```

The **barrier** construct specifies an explicit barrier at the point at which the construct appears.

```
!$omp barrier
```

The **taskwait** construct specifies a wait on the completion of child tasks generated since the beginning of the current task.

```
!$omp taskwait
```

The **atomic** construct ensures that a specific storage location is updated atomically, rather than exposing it to the possibility of multiple, simultaneous writing threads.

```
!$omp atomic
        statement
```

(See applicable statement on next page)

Directives (continued)

statement: one of the following forms:
- *x = x operator expr*
- *x = expr operator x*
- *x = intrinsic_procedure_name (x, expr_list)*
- *x = intrinsic_procedure_name (expr_list, x)*

The **flush** construct 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.

```
!$omp flush [(list)]
```

The **ordered** construct specifies a structured block in a loop region that will be executed in the order of the loop iterations. This sequentializes and orders the code within an ordered region while allowing code outside the region to run in parallel.

```
#pragma !$omp ordered
        structured-block
#pragma !$omp ordered
```

The **threadprivate** directive specifies that variables are replicated, with each thread having its own copy.

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

Clauses

Not all of the clauses are valid on all directives. The set of clauses that is valid on a particular directive is described with the directive. Most of the clauses accept a comma-separated list of list items. All list items appearing in a clause must be visible.

Data Sharing Attribute Clauses

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**)

Controls the default data-sharing attributes of variables that are referenced in a **parallel** or **task** construct.

shared(*list*)

Declares one or more list items to be shared by tasks generated by a **parallel** or **task** construct.

private(*list*)

Declares one or more list items to be private to a task.

firstprivate(*list*)

Declares one or more 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.

Clauses (continued)

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

reduction({*operator*|*intrinsic_procedure_name*}:*list*)
Declares accumulation into the list items using the indicated associative operator. Accumulation occurs into a private copy for each list item which is then combined with the original item.

Data Copying Clauses

These clauses support the copying of data values from private or threadprivate variables on one implicit task or thread to the corresponding variables on other implicit tasks or threads in the team.

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.

Runtime Library Routines

Execution environment routines affect and monitor threads, processors, and the parallel environment. Lock routines support synchronization with OpenMP locks. Timing routines support a portable wall clock timer. Prototypes for the runtime library routines appear in the include file "omp_lib.h" and the Fortran module "omp_lib".

Execution Environment Routines

subroutine omp_set_num_threads (*num_threads*)
integer *num_threads*
Affects the number of threads used for subsequent **parallel** regions that do not specify a **num_threads** clause.

integer function omp_get_num_threads()
Returns the number of threads in the current team.

integer function omp_get_max_threads()
Returns maximum number of threads that could be used to form a new team using a "parallel" construct without a "num_threads" clause.

integer function omp_get_thread_num()
Returns the ID of the encountering thread where ID ranges from zero to the size of the team minus 1.

integer function omp_get_num_procs()
Returns the number of processors available to the program.

Runtime Library Routines (continued)

logical function omp_in_parallel()
Returns *true* if the call to the routine is enclosed by an active **parallel** region; otherwise, it returns *false*.

subroutine omp_set_dynamic (*dynamic_threads*)
logical *dynamic_threads*
Enables/disables dynamic adjustment of number of threads available.

logical function omp_get_dynamic()
Returns value of *dyn-var* internal control variable (ICV), determining if dynamic adjustment of number of threads is enabled or disabled.

subroutine omp_set_nested(*nested*)
logical *nested*
Enables or disables nested parallelism, by setting the *nest-var* ICV.

logical function omp_get_nested()
Returns the value of the *nest-var* ICV, which determines if nested parallelism is enabled or disabled.

subroutine omp_set_schedule(*kind*, *modifier*)
integer(*kind*=**omp_sched_kind**) *kind*
integer modifier
Affects the schedule that is applied when **runtime** is used as schedule kind, by setting the value of the *run-sched-var* ICV.

subroutine omp_get_schedule(*kind*, *modifier*)
integer(*kind*=**omp_sched_kind**) *kind*
integer modifier
Returns the schedule applied when **runtime** schedule is used.

integer function omp_get_thread_limit()
Returns max number of OpenMP threads available to the program.

subroutine omp_set_max_active_levels(*max_levels*)
integer max_levels
Limits the number of nested active **parallel** regions, by setting the *max-active-levels-var* ICV.

integer function omp_get_max_active_levels()
Returns the value of the *max-activelevels-var ICV*, which determines the maximum number of nested active **parallel** regions.

integer function omp_get_level()
Returns the number of nested **parallel** regions enclosing the task that contains the call.

integer function omp_get_ancestor_thread_num(*level*)
integer *level*
Returns, for a given nested level of the current thread, the thread number of the ancestor or the current thread.

integer function omp_get_team_size(*level*)
integer *level*
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.

integer function omp_get_active_level()
Returns the number of nested, active **parallel** regions enclosing the task that contains the call.

Runtime Library Routines (continued)

Lock Routines

subroutine omp_{init|destroy|set|unset}_[nest_]lock(*var*)
integer(**kind=omp_[nest_]lock_kind**) *var*
These routines initialize, unitialize, set, or unset a (nested) OpenMP lock.

logical function omp_test_[nest_]lock(*var*)
integer(**kind=omp_[nest_]lock_kind**) *var*
These routines attempt to set an OpenMP lock but do not suspend execution of the task executing the routine.

Timing Routines

omp_get_wtime()
Returns elapsed wall clock time in seconds.

omp_get_wtick()
Returns the precision of the timer used by **omp_get_wtime**.

Environment Variables

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

OMP_SCHEDULE *type*[,*chunk*]
Sets the *run-sched-var* ICV for the runtime schedule type and chunk size. Valid OpenMP schedule types are **static**, **dynamic**, **guided**, or **auto**. *Chunk* is a positive integer.

OMP_NUM_THREADS *num*
Sets *nthreads-var* ICV for number of threads for **parallel** regions.

OMP_DYNAMIC *dynamic*
Sets *dyn-var* ICV *for* the dynamic adjustment of threads to use for **parallel** regions. Valid values for *dynamic* are **true** or **false**.

OMP_NESTED *nested*
Sets the *nest-var* ICV to enable or to disable nested parallelism. Valid values for *nested* are true or false.

OMP_STACKSIZE *size*
Sets *stacksize-var* ICV that specifies size of stack for threads created by the OpenMP implementation. Valid values for *size* (a positive integer) are *size*, *size***B**, *size***K**, *size***M**, *size***G**. If units **B**, **K**, **M** or **G** are not specified, size is measured in kilobytes (**K**).

OMP_WAIT_POLICY *policy*
Sets the *wait-policy-var* ICV that controls the desired behavior of waiting threads. Valid values for *policy* are **active** (waiting threads consume processor cycles while waiting) and **passive**.

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

OMP_THREAD_LIMIT *limit*
Sets the *thread-limit-var* ICV that controls the maximum number of threads participating in the OpenMP program.

Details

Operators legally allowed in a reduction

Operator	Initialization value	Operator	Initialization value
+	0	|	0
*	1	^	0
-	0	&&	1
&	~0	||	0

Schedule types for the loop construct

static Iterations are divided into chunks of *chunk_size*, chunks are assigned to threads in team in order of thread number.

dynamic Each thread executes a chunk of iterations, then requests another chunk until no chunks remain to be distributed.

guided Same as "dynamic," however chunk sizes start large and shrink to the indicated *chunk_size* as chunks are scheduled.

auto The decision regarding scheduling is delegated to the compiler and/or runtime system.

runtime Schedule and chunk size are taken from run-sched-var ICV.

Rev 0309-001