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

```fortran
!$omp parallel workshare [clause[, ... clause] ...]
```

- `structured-block`
- `any clause from parallel or sections`

The task construct defines an explicit task.

```fortran
!$omp task [clause[, ... clause] ...]
```

- `structured-block`
- `any clause from task or sections`

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

```fortran
!$omp critical [name]
```

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

```fortran
!$omp barrier
```

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

```fortran
!$omp sections [clause[, ... clause] ...]
```

- `structured-block`
- `any clause from sections`

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

```fortran
!$omp parallel [clause[, ... clause] ...]
```

- `structured-block`
- `any clause from parallel or sections`

The do-loops construct forms a team of threads and starts parallel execution.

```fortran
!$omp do [clause[, ... clause] ...]
```

The section 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.

```fortran
!$omp section [clause[, ... clause] ...]
```

- `structured-block`
- `any clause from sections`

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

```fortran
!$omp parallel do [clause[, ... clause] ...]
```

- `structured-block`

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

```fortran
!$omp workshare [structured-block]
```

- `structured-block`

The non-blocking directives apply 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.

```fortran
!$omp atomic
```

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

```fortran
!$omp auto
```

The allocation of iterations to chunks is done according to groups of threads.

```fortran
!$omp static
```

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

```fortran
!$omp dynamic
```

The value of `chunk_size` is divided into chunks of chunk size, chunks are assigned to threads in team in order of thread number.

```fortran
!$omp schedule(chunk_size)
```

The thread decides which iteration it executes at runtime.

```fortran
!$omp runtime
```

The variable is shared among all threads in the team.

```fortran
!$omp shared
```

The variable is private to each thread in the team.

```fortran
!$omp private
```

Each thread updates the chunk of iterations specified by the `chunk_size` clause.

```fortran
!$omp reduction(chunk
```

Copyright © 1997-2009 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."
Directives (continued)

statement: one of the following forms:
 x = x operator exp1
 x = * operator x
 x = intrinsic procedure name(s, exp2 list)
 x = intrinsic procedure name (exp 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.

#pragma 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 sequential-
izes 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.

#pragma omp threadprivate ([/list])

Clauses (continued)

data private ([/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. A copy 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 thread-
private 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.

copyin ([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 (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_get_dynamic (dynamic_threads)
Enables/disables dynamic adjustment of number of threads available.

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

subroutine omp_get_num_threads(nest)
Limits the number of active parallel regions, by setting the parallel region.

subroutine omp_get_max_threads()
Affects the number of threads used for subsequent
parallel regions.

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

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
time; Prototypes for the runtime routines appear in the include file
"omp_lib.h" and the Fortran module "omp_lib".

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.