Directives (continued)

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

```c
#pragma omp critical ([name_]) new-line
structured-block
```

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

```c
#pragma omp barrier new-line
```

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

```c
#pragma omp taskwait new-line
```

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

```c
#pragma omp atomic new-line
expression-stmt
expression-stmt: one of the following forms:
  b, b = expr
  ++ b
  -- b
  b = expr
```

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.

```c
#pragma omp flush [ (Unifo) ] new-line
```

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

```c
#pragma omp ordered new-line
structured-block
```

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

```c
#pragma omp threadprivate(list) new-line
```

Directives (continued)

Details

Operators legally allowed in a reduction

<table>
<thead>
<tr>
<th>Operator</th>
<th>Initialization value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>0</td>
</tr>
<tr>
<td>-=</td>
<td>0</td>
</tr>
<tr>
<td>*=</td>
<td>0</td>
</tr>
<tr>
<td>/=</td>
<td>0</td>
</tr>
<tr>
<td>&amp;=</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>=</td>
</tr>
<tr>
<td>^=</td>
<td>1</td>
</tr>
<tr>
<td>&gt;&gt;=</td>
<td>0</td>
</tr>
<tr>
<td>&lt;&lt;=</td>
<td>0</td>
</tr>
</tbody>
</table>

Schedule types for the loop construct

**static** Iterations are divided into chunks of size `chunk_size`, and the chunks are assigned to the threads in the team in a round-robin fashion in the order of the thread number.

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

**guided** Each thread executes a chunk of iterations, then requests another chunk, until no chunks remain to be assigned. The 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** The schedule and chunk size are taken from the run-sched-var ICV.

**Details**

Operators legally allowed in a reduction

<table>
<thead>
<tr>
<th>Operator</th>
<th>Initialization value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>0</td>
</tr>
<tr>
<td>-=</td>
<td>0</td>
</tr>
<tr>
<td>*=</td>
<td>0</td>
</tr>
<tr>
<td>/=</td>
<td>0</td>
</tr>
<tr>
<td>&amp;=</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>=</td>
</tr>
<tr>
<td>^=</td>
<td>1</td>
</tr>
<tr>
<td>&gt;&gt;=</td>
<td>0</td>
</tr>
<tr>
<td>&lt;&lt;=</td>
<td>0</td>
</tr>
</tbody>
</table>

Schedule types for the loop construct

**static** Iterations are divided into chunks of size `chunk_size`, and the chunks are assigned to the threads in the team in a round-robin fashion in the order of the thread number.

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

**guided** Each thread executes a chunk of iterations, then requests another chunk, until no chunks remain to be assigned. The 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** The schedule and chunk size are taken from the run-sched-var ICV.

**Summary of OpenMP 3.0**

C/C++ Syntax

Download the full OpenMP API Specification at [www.openmp.org](http://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. The parallel construct forms a team of threads and starts parallel execution.

```c
#pragma omp parallel [ [clause] [ ] ] new-line
structured-block
```

**clause:**

- `if (scalar-expression)`
- `num_threads ( [integer-expression] )`
- `default ( [shared none] )`
- `firstprivate ( ) [list]`
- `shared ( ) [list]`
- `copyin ( ) [list]`
- `copyin (operator) [list]`
- `reduction (operator: list)`

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

```c
#pragma omp for [ [clause] [ ] ] new-line
for-loop
```

**clause:**

- `if (scalar-expression)`
- `num_threads ( [integer-expression] )`
- `default (shared none)`
- `firstprivate ( ) [list]`
- `shared ( ) [list]`
- `copyin ( ) [list]`
- `copyin (operator) [list]`
- `reduction (operator: list)`

The most common form of the for loop is shown below.

```c
for ( var = lb; var <= ub; var += incr )
```

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

```c
#pragma omp sections [ [clause] [ ] ] new-line
{ [ [pragma omp section new-line structured-block ] ] [ [pragma omp section new-line structured-block ] ] } (See applicable clauses on next page.)
```

**The task construct** defines an explicit task. The data environment of the task is created according to the data-sharing attribute clauses on the task construct and any defaults that apply.

```c
#pragma omp task [ [clause] [ ] ] new-line
```

**clause:**

- `if (scalar-expression)`
- `default (shared none)`
- `private ( ) [list]`
- `firstprivate ( ) [list]`
- `shared ( ) [list]`

The combined parallel worksharing constructs are a shortcut for specifying a parallel construct containing one worksharing construct and no other statements. Permitted clauses are the unions of the clauses allowed for the parallel and worksharing constructs.

```c
#pragma omp parallel for [ [clause] [ ] ] new-line
for-loop
```

**clause:**

- `if (scalar-expression)`
- `num_threads ( [integer-expression] )`
- `default (shared none)`
- `firstprivate ( ) [list]`
- `shared ( ) [list]`
- `copyin ( ) [list]`
- `copyin (operator) [list]`
- `reduction (operator: list)`

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

```c
#pragma omp sections [ [clause] [ ] ] new-line
{ [ [pragma omp section new-line structured-block ] ] [ [pragma omp section new-line structured-block ] ] } (See applicable clauses on next page.)
```

**The task construct** defines an explicit task. The data environment of the task is created according to the data-sharing attribute clauses on the task construct and any defaults that apply.

```c
#pragma omp task [ [clause] [ ] ] new-line
```

**clause:**

- `if (scalar-expression)`
- `default (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. There is no implied barrier either on entry to, or exit from, the master construct.

```c
#pragma omp master new-line
structured-block
```
The critical construct restricts execution of the associated structured block to a single thread at a time.

```c
#pragma omp critical ([name]) new-line structured-block
```

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

```c
#pragma omp barrier new-line
```

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

```c
#pragma omp taskwait new-line
```

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

```c
#pragma omp atomic new-line expression-stmt
```

expression-stmt: one of the following forms:
- `x` (a variable)
- `x++` or `x--`
- `x=` or `x+=expr`
- `x >>= expr` or `x &= expr` or `x ^= expr`

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.

```c
#pragma omp flush [ (Durf)] new-line
```

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

```c
#pragma omp ordered new-line structured-block
```

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

```c
#pragma omp threadprivate(list) new-line
```

### Operators legally allowed in a reduction

<table>
<thead>
<tr>
<th>Operator</th>
<th>Initialization value</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>0</td>
</tr>
<tr>
<td>+=</td>
<td>0</td>
</tr>
<tr>
<td>-=</td>
<td>0</td>
</tr>
<tr>
<td>&amp;=</td>
<td>0</td>
</tr>
<tr>
<td>^=</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>0</td>
</tr>
<tr>
<td>^</td>
<td>0</td>
</tr>
<tr>
<td>^</td>
<td>0</td>
</tr>
<tr>
<td>~</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>1</td>
</tr>
<tr>
<td>^</td>
<td>1</td>
</tr>
<tr>
<td>^</td>
<td>1</td>
</tr>
</tbody>
</table>

### Schedule types for the loop construct

#### static
Intervals are divided into chunks of size `chunk_size`, and the chunks are assigned to the threads in a round-robin fashion in the order of the thread number.

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

#### guided
Each thread executes a chunk of iterations, then requests another chunk, until no chunks remain to be assigned. The 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
The schedule and chunk size are taken from the run-sched-var ICV.

### The most common form of the for loop is shown below:

```
for(var = lb; var <= ub; var += incr)
```

### Summary of OpenMP 3.0 C/C++ Syntax

Download the full OpenMP API Specification at [www.openmp.org](http://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. The parallel construct forms a team of threads and starts parallel execution.

```c
#pragma omp parallel [clause [ ] ... ] new-line structured-block
```

#### clause
- `if (scalar-expression)`
- `num_threads (integer-expression)`
- `default (shared | none)`
- `private (list)`
- `firstprivate (list)`
- `shared (list)`
- `copyin (list)`
- `reduction (operator: list)`

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

```c
#pragma omp for [ clause [ ] ... ] new-line for-loops
```

#### clause
- `private (list)`
- `firstprivate (list)`
- `lastprivate (list)`
- `copyprivate (list)`

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

```c
#pragma omp section [ clause [ ] ... ] new-line structured-block
```

#### clause
- `if (scalar-expression)`
- `num_threads (integer-expression)`
- `default (shared | none)`
- `private (list)`
- `firstprivate (list)`
- `shared (list)`
- `copyin (list)`
- `reduction (operator: list)`

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

```c
#pragma omp sections [ clause [ ] ... ] new-line [ ]
```

#### clause
- `if (scalar-expression)`
- `default (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. There is no implied barrier either on entry to, or exit from, the master construct.

```c
#pragma omp master new-line structured-block
```
Directives (continued)

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

#pragma omp critical [ (name) ] new-line structured-block

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

#pragma omp barrier new-line

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

#pragma omp taskwait new-line

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

#pragma omp atomic new-line

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

#pragma omp ordered new-line structured-block

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

#pragma omp threadprivate(list) new-line

## Operators legally allowed in a reduction

<table>
<thead>
<tr>
<th>Operator</th>
<th>Initialization value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;</td>
<td>0</td>
</tr>
<tr>
<td>&amp;&amp;&amp;</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
</tr>
<tr>
<td></td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td></td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>^=</td>
<td>0</td>
</tr>
<tr>
<td>+=</td>
<td>0</td>
</tr>
<tr>
<td>-=</td>
<td>0</td>
</tr>
<tr>
<td>*=</td>
<td>0</td>
</tr>
<tr>
<td>/=</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>=</td>
</tr>
<tr>
<td>&amp;=</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>=&amp;</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Schedule types for the loop construct

**static** Iterations are divided into chunks of size chunk_size, and the chunks are assigned to the threads in the team in a round-robin fashion in the order of the thread number.

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

**guided** Each thread executes a chunk of iterations, then requests another chunk, until no chunks remain to be assigned. The 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** The schedule and chunk size are taken from the run-time Var.CIC.

## Summary of OpenMP 3.0 C/C++ Syntax

Download the full OpenMP API Specification at [www.openmp.org](http://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.

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

```openmp
#pragma omp parallel [clause[ ], ... ] new-line structured-block
```

**clause**: if (scalar-expression) num_threads (integer-expression)

```openmp
default (shared | none) private (list)
```

**firstprivate** (list)

```openmp
shared (list)
```

**copyin** (list)

```openmp
reduction (operator: list)
```

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

```openmp
#pragma omp for [ clause[ ], ... ] new-line
```

The most common form of the for loop is shown below.

```
for (var = lb; var <= ub; var = var + inc)
```

### OpenMP 3.0 APL Syntax

```

description
```

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

```openmp
#pragma omp sections [ clause[ ], ... ] new-line
```

```
[ #pragma omp section new-line structured-block ]
```

The master construct specifies a structured block that is executed by the master thread of the team. There is no implied barrier either on entry to, or exit from, the master construct.

```openmp
#pragma omp master new-line structured-block
```

Copyright © 1997-2008 OpenMP Architecture Review Board. Permission to copy without 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.”

Rev 1329 02/06
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(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 an implicit 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 private to an implicit task, and causes the corresponding original item to be updated after the end of the region.

reduction(operator | 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 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.

copyinprivate(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.

Execution 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 are defined in the file "omp.h".

Execution Environment Routines
void omp_set_num_threads(int num_threads);
  Affects the value of the max-thread-count ICV, which determines the maximum number of threads available.

int omp_get_num_threads(void);
  Returns the number of threads in the current team.

void omp_set_max_threads(int max_threads);
  Returns the value of the max-thread-count ICV.

int omp_get_max_threads(void);
  Returns the number of threads available for subsequent parallel regions.

void omp_get_thread_num(void);
  Returns the ID of the enclosing thread where ID ranges from zero to the size of the team minus 1.

int omp_get_num_procs(void);
  Returns the number of processors available to the program.

void omp_set_max_num_procs(int num_procs);
  Sets the runtime maximum number of processors available.

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

void omp_set_dynamic(int dynamic);
  Enables or disables dynamic adjustment of the number of threads available.

int omp_get_dynamic(void);
  Returns the value of the dyn-var internal control variable (ICV), determining whether dynamic adjustment of the number of threads is enabled or disabled.

void omp_set_nested(int nested);
  Enables or disables nested parallelism, by setting the next-nest ICV.

int omp_get_nested(void);
  Returns the value of the next-nest ICV, which determines if nested parallelism is enabled or disabled.

void omp_set_thread_affinity(int *affinity);
  Assigns thread IDs to a list of thread IDs. Affinity values are non-negative integers.

int omp_get_thread_limit(void);
  Returns the number of nested, active parallel regions enclosing the task that contains the call.

int omp_get_max_active_levels(void);
  Returns the number of nested active parallel regions.

void omp_set_max_active_levels(int max_active_levels);
  Sets the max-active-levels ICV, which controls the maximum number of nested active parallel regions.

int omp_get_active_levels(void);
  Returns the number of nested active parallel regions enclosing the task that contains the call.

int omp_get_dynamic(void);
  Returns true if the call to the routine is enclosed by an active parallel region; otherwise, it returns false.

int omp_get_thread_num(void);
  Returns the ID of the current thread.

int omp_get_thread_num(int level);
  Returns, for a given nested level of the current thread, the thread number of the ancestor or the current thread.

int omp_get_thread_num(int thread, int 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.

int omp_get_num_threads(void);
  Returns the total number of active parallel regions.

int omp_get_team_size(int level);
  Returns the number of threads in the current team.

Data Sharing Attribute Clauses (continued)

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
  Sets the runtime schedule type and chunk size. Valid OpenMP schedule types are dynamic, guided, or auto. Chunk is a positive integer.

OMP_NUM_THREADS
  Sets the thread-count ICV for the number of threads to use for parallel regions.

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

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

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

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

OMP_THREAD_LIMIT
  Sets the thread-limit-var ICV that controls the maximum number of threads participating in the OpenMP program.
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(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.
- **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: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 copy 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.
- **copyprivate(list)**: Broadcasts a value from the data environment of one implicit task to the data environments of other implicit tasks belonging to the parallel region.

Execution Environment 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 are defined in the file “omp.h”.

Runtime Library Routines

- **int omp_get_thread_limit(void):** Returns the maximum number of OpenMP threads available to the program.
- **void omp_set_max_active_levels(int max_levels):** Limits the number of nested active parallel regions, by setting the max-active-levels ICV.
- **int omp_get_max_active_levels(void):** Returns the value of the max-active-levels ICV, which determines the maximum number of nested active parallel regions.
- **int omp_get_level(void):** Returns the number of nested parallel regions enclosing the task that contains the call.
- **int omp_get_ancestor_thread_num(int level):** Returns, for a given nested level of the current thread, the thread number of its ancestor or the current thread.
- **int omp_get_team_size(int 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.
- **int omp_get_active_level(void):** Returns the number of nested, active parallel regions enclosing the task that contains the call.

Lock Routines

- **void omp_init_lock(omp_lock_t *lock):** Initializes an OpenMP lock.
- **void omp_destroy_lock(omp_lock_t *lock):** These routines ensure that the OpenMP lock is uninitialized.
- **int omp_set_nest_lock(omp_nest_lock_t *lock):** These routines provide a means of setting an OpenMP lock.
- **void omp_unset_nest_lock(omp_nest_lock_t *lock):** These routines provide a means of setting an OpenMP lock.
- **int omp_get_max_active_levels(void):** Returns the value of the max-active-levels ICV.

Runtime Library Routines (continued)

- **int omp_get_thread_limit(void):** Returns the maximum number of OpenMP threads available to the program.
- **void omp_set_max_active_levels(int max_levels):** Limits the number of nested active parallel regions, by setting the max-active-levels ICV.
- **int omp_get_max_active_levels(void):** Returns the value of the max-active-levels ICV, which determines the maximum number of nested active parallel regions.
- **int omp_get_level(void):** Returns the number of nested parallel regions enclosing the task that contains the call.
- **int omp_get_ancestor_thread_num(int level):** Returns, for a given nested level of the current thread, the thread number of its ancestor or the current thread.
- **int omp_get_team_size(int 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.
- **int omp_get_active_level(void):** Returns the number of nested, active parallel regions enclosing the task that contains the call.

Lock Routines

- **void omp_init_lock(omp_lock_t *lock):** Initializes an OpenMP lock.
- **void omp_destroy_lock(omp_lock_t *lock):** These routines ensure that the OpenMP lock is uninitialized.
- **int omp_set_nest_lock(omp_nest_lock_t *lock):** These routines provide a means of setting an OpenMP lock.
- **void omp_unset_nest_lock(omp_nest_lock_t *lock):** These routines provide a means of setting an OpenMP lock.
- **int omp_get_max_active_levels(void):** Returns the value of the max-active-levels ICV.

Runtime Library Routines (continued)

- **int omp_get_thread_limit(void):** Returns the maximum number of OpenMP threads available to the program.
- **void omp_set_max_active_levels(int max_levels):** Limits the number of nested active parallel regions, by setting the max-active-levels ICV.
- **int omp_get_max_active_levels(void):** Returns the value of the max-active-levels ICV, which determines the maximum number of nested active parallel regions.
- **int omp_get_level(void):** Returns the number of nested parallel regions enclosing the task that contains the call.
- **int omp_get_ancestor_thread_num(int level):** Returns, for a given nested level of the current thread, the thread number of its ancestor or the current thread.
- **int omp_get_team_size(int 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.
- **int omp_get_active_level(void):** Returns the number of nested, active parallel regions enclosing the task that contains the call.

Lock Routines

- **void omp_init_lock(omp_lock_t *lock):** Initializes an OpenMP lock.
- **void omp_destroy_lock(omp_lock_t *lock):** These routines ensure that the OpenMP lock is uninitialized.
- **int omp_set_nest_lock(omp_nest_lock_t *lock):** These routines provide a means of setting an OpenMP lock.
- **void omp_unset_nest_lock(omp_nest_lock_t *lock):** These routines provide a means of setting an OpenMP lock.
- **int omp_get_max_active_levels(void):** Returns the value of the max-active-levels ICV.

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(int value):** Sets the run-schedule 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 the nthreads-var ICV for the number of threads to use for parallel regions. Valid values for dynamic are true or false.
- **OMP_DYNAMIC** Sets the dyn-var ICV for the dynamic adjustment of threads to use for parallel regions. Valid values for dynamic are true or false.
- **OMP_NESTED** Sets the nest-var ICV to enable or disable nested parallelism. Valid values for nested are true or false.
- **OMP_STACKSIZE size** Sets the stacksize-var ICV that specifies the size of the stack for threads created by the OpenMP implementation. Valid values for size (a positive integer) are size, sizex, sizexK, sizexM, sizexG. If units B, M, G or 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_NUM_ACTIVE_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.

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(int value):** Sets the run-schedule 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 the nthreads-var ICV for the number of threads to use for parallel regions. Valid values for dynamic are true or false.
- **OMP_DYNAMIC** Sets the dyn-var ICV for the dynamic adjustment of threads to use for parallel regions. Valid values for dynamic are true or false.
- **OMP_NESTED** Sets the nest-var ICV to enable or disable nested parallelism. Valid values for nested are true or false.
- **OMP_STACKSIZE size** Sets the stacksize-var ICV that specifies the size of the stack for threads created by the OpenMP implementation. Valid values for size (a positive integer) are size, sizex, sizexK, sizexM, sizexG. If units B, M, G or 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_NUM_ACTIVE_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.
Data Sharing Attribute Clauses

These clauses apply only to variables whose names are referenced in a parallel or task construct.

- `shared`: Declares one or more list items to be shared by tasks generated by a parallel or task construct.
- `private`: Declares one or more list items to be private to a task.
- `firstprivate`: Declares one or more list items to be private to an implicit task, and initializes each of them with the value that the corresponding original item had when the construct is encountered.
- `lastprivate`: 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: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

- `copy(list)`: Copies the value of the master thread's 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.

Execution Environment Routines

- void omp_set_num_threads(int num_threads); Affects the value of threads used for subsequent parallel regions that do not specify a num_threads clause.
- int omp_get_num_threads(void); Returns the number of threads in the current team.
- int omp_get_max_threads(void); Returns maximum number of threads that could be used to form a new team using a "parallel" construct without a "num_threads" clause.
- void omp_set_thread_num(int id); Returns the ID of the enclosing thread where ID ranges from zero to the size of the team minus 1.
- int omp_get_thread_num(void); Returns the number of processors available to the program.
- int omp_get_num_procs(void); Returns the number of processes.
- void omp_set_dynamic(int dynamic); Enables or disables dynamic adjustment of the number of threads available.
- int omp_get_dynamic(void); Returns the value of the dyn-var internal control variable (ICV), determining whether dynamic adjustment of the number of threads is enabled or disabled.
- void omp_set_nested(int nested); Enables or disables nested parallelism, by setting the next-nest ICV.
- int omp_get_nested(void); Returns the value of the next-nest ICV, which determines if nested parallelism is enabled or disabled.
- void omp_set_schedule(omp_sched_t kind, int modifier); Affects the schedule that is applied when runtime is used as a directive and, by setting the value of the run-sched ICV.
- void omp_get_schedule(omp_sched_t *kind, int *modifier); Returns the value of the run-sched ICV.

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: Sets the run-sched ICV for the runtime schedule type and chunk size. Valid OpenMP schedule types are dynamic, guided, or auto. Chunk is a positive integer.
- OMP_NUM_THREADS: Sets the number of OpenMP threads to use for parallel regions. Valid values for dynamic are true or false.
- OMP_DYNAMIC: Sets the dynamic schedule for the dynamic adjustment of threads to use for parallel regions. Valid values are true or false.
- OMP_NESTED: Sets the next-nest ICV to enable or disable nested parallelism. Valid values are true or false.
- OMP_STACKSIZE: Sets the stacksize ICV that specifies the size of the stack for threads created by the OpenMP creation method. Valid values for size (a positive integer) are sizex, sizexK, sizexM, sizexG. If units B, M, or G are not specified, size is measured in kilobytes (K).
- OMP_WAIT_POLICY: Sets the wait-policy 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_NUM_ACTIVE_LEVELS: Sets the max-active-levels ICV that controls the maximum number of nested active parallel regions.
- OMP_THREAD_LIMIT: Sets the number of OpenMP threads participating in the OpenMP program.

Garbage Collector introduce a reference graph and a garbage collection algorithm, which is based on the reference graph and triggers the garbage collection, to ensure the correctness of the program.
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(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 an implicit task, and 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 private to a task, and initializes the corresponding original item to be updated after the end of the region.
reduction(operator;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 copy 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.
copyinprivate(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 are defined in the file "omp.h".

Execution Environment Routines
void omp_set_num_threads(int num_threads);
Affects the value of the max-thread-count for subsequent parallel regions that do not specify a num_threads clause.
int omp_get_num_threads(void);
Returns the number of threads in the current team.
int omp_get_max_threads(void);
Returns the value of the max-thread-count used for the current parallel region; otherwise, it returns the maximum number of active parallel regions.

Runtime Library Routines (continued)
int omp_get_thread_limit(void);
Returns the precision of the timer used by omp_set_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
Sets the schedule type to use for parallel regions.
OMP_DYNAMIC
Sets the dynamic schedule type to use for parallel regions.
OMP_MAX_ACTIVE_LEVELS
Sets the maximum number of nested active parallel regions.
OMP_THREAD_LIMIT
Sets the maximum number of nested active parallel regions.
OMP_NUM_THREADS
Sets the number of OpenMP threads available to the program.
OMP_NUM_MAX_ACTIVE_LEVELS
Sets the maximum number of parallel regions.
OMP_WAIT_POLICY
Sets the default behavior of waiting threads. Valid values for policy are active (waiting threads consume processor cycles while waiting) and passive.

Other Environment Variables
OMP_NUM_THREADS
Sets the number of OpenMP threads available to the program.
OMP_NUM_MAX_ACTIVE_LEVELS
Sets the maximum number of parallel regions.
OMP_THREAD_LIMIT
Sets the maximum number of nested active parallel regions.
## Directives (continued)

### Operators legally allowed in a reduction

<table>
<thead>
<tr>
<th>Operator</th>
<th>Initialization value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>0</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>*</td>
<td>0</td>
</tr>
<tr>
<td>/</td>
<td>0</td>
</tr>
<tr>
<td>&amp;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>0</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>0</td>
</tr>
<tr>
<td>!&amp;</td>
<td>0</td>
</tr>
<tr>
<td>!</td>
<td></td>
</tr>
<tr>
<td>!^</td>
<td>0</td>
</tr>
</tbody>
</table>

### Schedule types for the loop construct

#### static

- Iterations are divided into chunks of size `chunk_size`, and the chunks are assigned to the threads in a round-robin fashion in the order of the thread number.

#### dynamic

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

#### guided

- Each thread executes a chunk of iterations, then requests another chunk, until no chunks remain to be assigned.
- The 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

- The schedule and chunk size are taken from the run-sched-var ICV.

### Summary of OpenMP 3.0 C/C++ Syntax

Download the full OpenMP API Specification at [www.openmp.org](http://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.

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

**#pragma omp parallel**

```c
#pragma omp parallel [clause...] new-line structured-block
```

- **clause**: any clause allowed for the parallel construct.
- **structured-block**: any OpenMP construct returns a structured block.

**The most common form of the for loop is shown below**

```c
for (var = lb; var relational-op ub; var += incr) {
  ...;
}
```

#### The sections construct

The sections construct consists of a set of structured blocks that are to be distributed among and executed by the encountering team of threads.

```c
#pragma omp sections [clause...] new-line structured-block
```

- **clause**: any clause allowed for the parallel sections construct.
- **structured-block**: any OpenMP construct

#### The task construct

The task construct defines an explicit task. The data environment of the task is created according to the data-sharing attribute clauses on the task construct and any defaults that apply.

```c
#pragma omp task [clause...] new-line structured-block
```

- **clause**: any clause allowed for the task construct.

#### The taskwait construct

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

```c
#pragma omp taskwait
```

- This clause is executed by only one of the threads in the team (not necessarily the master thread), in the context of its implicit task.

### Details

#### The critical construct

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

```c
#pragma omp critical (name) new-line structured-block
```

#### The barrier construct

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

```c
#pragma omp barrier new-line
```

#### The taskwait construct

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

```c
#pragma omp taskwait new-line
```

#### The atomic construct

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

```c
#pragma omp atomic new-line expression-stmt
```

#### The flush construct

The flush construct executes the OpenMP flush operation, which makes a block to a single thread at a time.

```c
#pragma omp flush [flushlist] new-line
```

#### The ordered construct

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

```c
#pragma omp ordered new-line structured-block
```

#### The threadprivate directive

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

```c
#pragma omp threadprivate(list) new-line
```

#### An order on the memory of variables

Each thread has its own copy of the variables.

#### Ordering the code within an ordered region while allowing code outside the region to run in parallel

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

```c
#pragma omp for [clause...] new-line for-loops
```

- **clause**: any clause allowed for the for-loops.
- **for-loops**: any OpenMP construct

#### An order on the memory of variables

The combined parallel worksharing constructs are a shortcut for specifying a parallel construct containing one worksharing construct and no other statements. Permitted clauses are the unions of the clauses allowed for the parallel and worksharing constructs.

```c
#pragma omp parallel for [clause...] new-line for-loops
```

#### The sections construct

The sections construct consists of a set of structured blocks that are to be distributed among and executed by the encountering team of threads.

```c
#pragma omp sections [clause...] new-line structured-block
```

- **clause**: any clause allowed for the parallel sections construct.
- **structured-block**: any OpenMP construct

#### The task construct

The task construct specifies a structured block that is executed by the master thread of the team. There is no implied barrier either on entry to, or exit from, the master construct.

```c
#pragma omp task new-line structured-block
```

- **structured-block**: any OpenMP construct

#### The taskwait construct

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

```c
#pragma omp taskwait
```

- This clause is executed by only one of the threads in the team (not necessarily the master thread), in the context of its implicit task.