Summary of OpenMP 3.0 C/C++ Syntax

Directives
An OpenMP executable directive applies to the succeeding structured block or on 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 [ clauses[ ] ] [ new-line ]

structured-block

class: if( scalar-expression )
num_threads( integer-expression )
default( shared | none )
private( list )
firstprivate( list )
lastprivate( list )
reduction( operator: list )
none

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.

#pragma omp single [ clauses[ ] ] [ new-line ]

structured-block

class: private( list )
firstprivate( list )
lastprivate( list )
reduction( operator: list )
none

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 union of the clauses allowed for the parallel and worksharing constructs.

#pragma omp parallel for [ clauses[ ] ] [ new-line ]

for-loop

class: private( list )
firstprivate( list )
lastprivate( list )
reduction( operator: list )
none

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

#pragma omp for [ clauses[ ] ] [ new-line ]

for-loop

class: 
private( list )
firstprivate( list )
lastprivate( list )
reduction( operator: list )
none

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

#pragma omp sections [ clauses[ ] ] [ new-line ]

structured-block

class: if( scalar-expression )
default( shared | none )
collapsible( list )
order( list )
nowait

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

#pragma omp sections [ clauses[ ] ] [ new-line ]

structured-block

class: if( scalar-expression )
default( shared | none )
collapsible( list )
order( list )
nowait

The construct appears. The clause class must be either private, firstprivate, lastprivate, or reduction. The clause list can be empty.

The shared construct specifies that a structured block in a team is executed by all of the threads in the team. The shared construct is a shortcut for the parallel construct.

#pragma omp shared [ clauses[ ] ] [ new-line ]

structured-block

class: private( list )
firstprivate( list )
lastprivate( list )
reduction( operator: list )
none

The construct appears. The clause class must be either private, firstprivate, lastprivate, or reduction. The clause list can be empty.

The master construct specifies that 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.

#pragma omp master [ clauses[ ] ] [ new-line ]

structured-block

The task construct specifies that the associated structured block is executed at a specific time. The task construct is a shortcut for the parallel construct.

#pragma omp task [ clauses[ ] ] [ new-line ]

structured-block

class: if( scalar-expression )
default( shared | none )
collapsible( list )
order( list )
nowait

The task construct specifies an explicit task, that is, a piece of work that will be performed by one of the threads in the team. The task construct is a shortcut for the parallel construct.

#pragma omp task [ clauses[ ] ] [ new-line ]

structured-block

class: if( scalar-expression )
default( shared | none )
collapsible( list )
order( list )
nowait

The taskwait construct specifies a point at which the associated task waits for other tasks to complete.

#pragma 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.

#pragma omp atomic new-line
expression-stmt; one of the following forms:

x += expr
x -= expr
x *= expr
x /= expr

The construct appears. The clause class must be atomic. The clause list can be empty.

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

#pragma omp critical [ clauses[ ] ] new-line

structured-block

The barrier construct specifies that the associated structured block will be executed only after all other threads in the team have finished executing the associated structured block.

#pragma omp barrier new-line

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 [ clauses[ ] ] new-line

The ordered construct specifies a structured block in a region that will be executed in the order of the loop iterations. This sequencing occurs within the region, and the order is enforced by a linear barrier at the end of the region.

#pragma omp ordered new-line

structured-block

class: if( scalar-expression )
default( shared | none )
collapsible( list )
order( list )
nowait

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

#pragma omp threadprivate( list ) new-line

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

#pragma omp threadprivate( list ) new-line

The construct appears. The clause class must be threadprivate. The clause list can be empty.

Data Sharing Attribute 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 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.

taskprivate( list );

Declares one or more list items to be taskprivate to the task construct.

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.

copyprivate( list );

Copies the value from the data environment of one implicit task to the data environments of the other implicit tasks belonging to the parallel region.

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.
Runtime Library Routines
Execution environment routines affect and monitor threads, processes, 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

```c
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
thread using a "parallel" construct without a "num_threads" clause.
int omp_get_thread_num(void);
Returns the ID of the encountering 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.
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
Enables or disables dynamic adjustment of the number of threads
available.
void omp_set_num_threads(int
Affects the number of threads used for subsequent parallel
regions that do not specify a num_threads clause.
```

Timing Routines

```c
int omp_get_thread_limit(void);
Returns the maximum number of OpenMP threads available to the
program.
void omp_set_max_active_levels(int
Limits the number of nested active parallel regions, by setting the
max-active-levels-var ICV.
int omp_get_max_active_levels(void);
Returns the value of the max-active-levels-var 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
Returns, for a given given nested level of the current thread, the thread
number of the ancestor or the current thread.
int omp_get_team_size(void);
Returns the number of nested, active parallel regions
enclosing the task that contains the call.
```

Lock Routines

```c
void omp_init_lock(omp_lock_t *
These routines initialize an OpenMP lock.
void omp_destroy_lock(omp_lock_t *
These routines initialize an OpenMP lock.
void omp_destroy_lock(omp_lock_t *
These routines initialize an OpenMP lock.
```

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 run-time 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 number-of-thread-var ICV for the number of threads to use for
parallel regions.
OMP_MAX_ACTIVE_LEVELS size
Sets the max-active-levels-var ICV for the runtime schedule type and chunk
size. Valid OpenMP schedule types are static, dynamic, guided, or
auto. Size is a positive integer.
OMP_DYNAMIC 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_WAIT_POLICY policy
Sets the wait-policy-var ICV that controls the desirable behavior of
waiting threads. Valid values for policy are active (waiting threads
consume processor cycles while waiting) or passive.
OMP_MAX_ACTIVE_LEVELS size
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.
```

Timing Routines

```c
double omp_get_wtime(void);
Returns elapsed wall clock time in seconds.
double omp_get_wtick(void);
Returns time since the start of the program.
```

Details

<table>
<thead>
<tr>
<th>Operator</th>
<th>Initialization value</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>1</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>&amp;&amp;</td>
<td>1</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.
auto
The decision regarding scheduling is delegated to the compiler and/or runtime system.
```

Schedule variables

```
tau
The schedule and chunk size are taken from the
nan-sched-var ICV.
```

Copyright © 1997-2008 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.”
Rev 1 1/98-982