OpenMP Application Program Interface (API) is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer. A separate OpenMP reference card for Fortran is also available.

OpenMP 3.1 API C/C++ Syntax Quick Reference Card

OpenMP supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures, including Unix platforms and Windows NT platforms.

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.

Parallel [2.4]
The parallel construct forms a team of threads and starts parallel execution.

```c
#pragma omp parallel [clause[, clause]...] structured-block
clause:
  if(color-expression)
  num_threads(integer-expression)
  default(shared | none)
  private(list)
  firstprivate(list)
  shared(list)
  copyprivate(list)
  reduction(operator: list)

Loop [2.5.1]
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[, clause]...] for-loops
clause:
  private(list)
  firstprivate(list)
  lastprivate(list)
  reduction(operator: list)
  schedule(kind[, chunk_size])
  collapse(n)
  ordered
  nowait
```

Simple Parallel Loop Example

The following example demonstrates how to parallelize a simple loop using the parallel loop construct.

```c
void simple(int n, float *a, float *b)
{
  for(var = lb; var < ub; var += incr)
    b[var] = (a[var] + a[var-1]) / 2.0;
}
```

Parallel Sections [2.6.2]
The parallel sections construct is a shortcut for specifying a parallel construct containing one or more associated loops and no other statements.

```c
#pragma omp parallel sections [clause[, clause]...] structured-block

```

Taskyield [2.7.2]
The taskyield construct specifies that the current task can be suspended in favor of execution of a different task.

```c
#pragma omp taskyield
```

Master [2.8.1]
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 structured-block
```

Critical [2.8.2]
The critical construct restricts execution of the associated structured block to a single thread at a time.

```c
#pragma omp critical ([name])
```

Barrier [2.8.3]
The barrier construct specifies an explicit barrier at the point at which the construct appears.

```c
#pragma omp barrier
```

Taskwait [2.8.4]
The taskwait construct specifies a wait on the completion of child tasks of the current task.

```c
#pragma omp taskwait
```

Atomic [2.8.5]
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 [read | write | update | capture | expression-stmt]
#pragma omp atomic capture
```

Flush [2.8.6]
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 [list]
```

Ordered [2.8.7]
The ordered construct specifies a structured block in a loop region that will be executed in the order of the loop iterations. This serializes and orders the code within an ordered region while allowing code outside the region to run in parallel.

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

Threadprivate [2.9.2]
The threadprivate directive specifies that variables are replicated, with each thread having its own copy.

```c
#pragma omp threadprivate(list)
```

where expression-stmt may be one of the following forms:

- if clause...
- expression-stmt:
  - read v=x;
  - write v=x;
  - update or is not present v=x; x=x binop expr; ++v;
  - capture v=x; x=x binop expr; v=x;

```

and structured-block may be one of the following forms:

- [v=x; x=x binop expr; v=x] v=x;
- [v=x; x=x binop expr; v=x] v=x;
- [v=x; x=x binop expr; v=x] v=x;
- [v=x; x=x binop expr; v=x] v=x;
- [v=x; x=x binop expr; v=x] v=x;

```

Flush [2.8.6]
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 [list]
```

Ordered [2.8.7]
The ordered construct specifies a structured block in a loop region that will be executed in the order of the loop iterations. This serializes and orders the code within an ordered region while allowing code outside the region to run in parallel.

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

Threadprivate [2.9.2]
The threadprivate directive specifies that variables are replicated, with each thread having its own copy.

```c
#pragma omp threadprivate(list)
```

A comma-separated list of file scope, namespace scope, or static block scope variables that do not have incomplete types.
### Runtime Library Routines

#### Execution Environment Routines [3.2]

Execution environment routines affect and monitor thread IDs, processors, and the parallel environment.

- `omp_get_num_threads()`: Returns the number of processors available in the current team.
- `omp_get_max_threads()`: Returns maximum number of threads that can be used to form a new team using a `parallel` construct without a `num_threads` clause.
- `omp_get_thread_num()`: Returns the ID of the encountering thread where ID ranges from zero to the size of the team minus one.

#### Data Types For Runtime Library Routines

- `omp_lock_t`: Represents a simple lock.
- `omp_nest_lock_t`: Represents a lock that can be nested.
- `omp_sched_t`: Represents a schedule.

#### Clauses

The set of clauses that is valid on a particular directive is described with the directive. Most clauses accept a comma-separated list of items. All list items appearing in a clause must be visible.

#### Data Sharing Attribute Clauses [2.9.3]

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.

#### Data Copying Clauses [2.9.4]

These clauses support the copying of data from private 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.

#### Environment Variables

Environment variables are described in section [4] of the API specification. Environment variables 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 that specifies chunk size.

- `OMP_NUM_THREADS`: Set the nthreads-var 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_PROC_BIND`: Sets the value of the global `bind-var ICV`. The value of this environment variable must be true or false.

### Lock Routines [3.3]

Lock routines support synchronization with OpenMP locks.

- `omp_init_lock()`: Initializes an OpenMP lock.
- `omp_destroy_lock()`: Clears an OpenMP lock.
- `omp_unset_lock()`: Unsets an OpenMP lock.
- `omp_set_lock()`: Locks an OpenMP lock.
- `omp_unset_nest_lock()`: Unsets a nested OpenMP lock.
- `omp_nest_lock()`: Sets an OpenMP lock.

These routines provide a means of setting an OpenMP lock.

#### Timing Routines [3.4]

Timing routines support a portable wall clock timer.

- `double omp_get_wtime()`: Returns elapsed wall clock time in seconds.

- `double omp_get_wtick()`: Returns the precision of the timer used by `omp_get_wtime().`