

Function

Description

Execution Environment Routines

```
subroutine omp_set_num_threads(num_threads)  
integer num_threads
```

Sets the number of threads to use for subsequent parallel regions.

```
integer function omp_get_num_threads()
```

Returns the number of threads that are being used in the current parallel region.

```
integer function omp_get_max_threads()
```

Returns the maximum number of threads that are available for parallel execution.

```
integer function omp_get_thread_num()
```

Determines the unique thread number of the thread currently executing this section of code.

```
integer function omp_get_num_procs()
```

Determines the number of processors available to the program.

```
logical function omp_in_parallel()
```

Returns `.true.` if called within the dynamic extent of a parallel region executing in parallel; otherwise returns `.false..`

```
subroutine omp_set_dynamic(dynamic_threads)  
logical dynamic_threads
```

Enables or disables dynamic adjustment of the number of threads used to execute a parallel region. If *dynamic_threads* is `.true.`, dynamic threads are enabled. If *dynamic_threads* is `.false.`, dynamic threads are disabled. Dynamics threads are disabled by default.

```
logical function omp_get_dynamic()
```

Returns `.true.` if dynamic thread adjustment is enabled, otherwise returns `.false..`

```
subroutine omp_set_nested(nested)
integer nested
```

Enables or disables nested parallelism. If *nested* is `.true.`, nested parallelism is enabled. If *nested* is `.false.`, nested parallelism is disabled. Nested parallelism is disabled by default.

```
logical function omp_get_nested()
```

Returns `.true.` if nested parallelism is enabled, otherwise returns `.false.`.

Lock Routines

```
subroutine omp_init_lock(lock)
integer (kind=omp_lock_kind)::lock
```

Initializes the lock associated with *lock* for use in subsequent calls.

```
subroutine omp_destroy_lock(lock)
integer (kind=omp_lock_kind)::lock
```

Causes the lock associated with *lock* to become undefined.

```
subroutine omp_set_lock(lock)
integer (kind=omp_lock_kind)::lock
```

Forces the executing thread to wait until the lock associated with *lock* is available. The thread is granted ownership of the lock when it becomes available.

```
subroutine omp_unset_lock(lock)
integer (kind=omp_lock_kind)::lock
```

Releases the executing thread from ownership of the lock associated with *lock*. The behavior is undefined if the executing thread does not own the lock associated with *lock*.

```
logical omp_test_lock(lock)
integer (kind=omp_lock_kind)::lock
```

Attempts to set the lock associated with *lock*. If successful, returns `.true.`, otherwise returns `.false.`.

```
subroutine omp_init_nest_lock(lock)
integer (kind=omp_nest_lock_kind)::lock
```

Initializes the nested lock associated with *lock* for use in the subsequent calls.

```
subroutine omp_destroy_nest_lock(lock)
integer(kind=omp_nest_lock_kind)::lock
```

Causes the nested lock associated with *lock* to become undefined.

```
subroutine omp_set_nest_lock(lock)
integer(kind=omp_nest_lock_kind)::lock
```

Forces the executing thread to wait until the nested lock associated with *lock* is available. The thread is granted ownership of the nested lock when it becomes available.

```
subroutine omp_unset_nest_lock(lock)
integer(kind=omp_nest_lock_kind)::lock
```

Releases the executing thread from ownership of the nested lock associated with *lock* if the nesting count is zero. Behavior is undefined if the executing thread does not own the nested lock associated with *lock*.

```
integer omp_test_nest_lock(lock)
integer(kind=omp_nest_lock_kind)::lock
```

Attempts to set the nested lock associated with *lock*. If successful, returns the nesting count, otherwise returns zero.

Timing Routines

```
double-precision function omp_get_wtime()
```

Returns a double-precision value equal to the elapsed wallclock time (in seconds) relative to an arbitrary reference time. The reference time does not change during program execution.

```
double-precision function omp_get_wtick()
```

Returns a double-precision value equal to the number of seconds between successive clock ticks.

