Architecture et Programmation Parallèle

Examen Session Rattrapage

Ndeye Arame Diago - Thierry Garcia - Juan Angel Lorenzo - Mohamed Maachaoui

ING2-GSI-MI	Année 2021 - 2022

Modalités

- Durée : 2 heures.
- Vous devez rédiger votre copie à l'aide d'un stylo à encre exclusivement.
- Toutes vos affaires (sacs, vestes, trousse, etc.) doivent être placées à l'avant de la salle.
- Aucun document n'est autorisé sauf les feuilles des fonctions MPI et OpenMP fournies avec ce sujet.
- Aucune question ne peut être posée aux enseignants, posez des hypothèses en cas de doute.
- Aucune sortie n'est autorisée avant une durée incompressible d'une heure.
- Aucun déplacement n'est autorisé.
- Aucun échange, de quelque nature que ce soit, n'est possible.

Exercice 1 : Questions de cours (5 points)

1. Expliquer les différences entre : `#pragma omp single` et `#pragma omp master` **(1 point)**.

2. Le code parallèle suivant doit retourner l'addition des éléments de la boucle for. Malheureusement, le résultat affiché n'est pas correct. Expliquez les fautes existantes dans le code et corrigez-les **(2 points)**.

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
  int nthreads, i, tid;
  float total = -1.0;

 #pragma omp parallel private(tid)
 {
    tid = omp_get_thread_num();

   #pragma omp master
   {
        nthreads = omp_get_num_threads();
```

```
            printf("Number of threads = %d\n", nthreads);
            total = 0.0;
        }

        #pragma omp for
        for (i=0; i<1000; i++) {
           total = total + i*1.0;
        }

        printf ("Thread %d is done! Total= %f\n",tid,total);

    } /*** End of parallel region ***/
}
```

3. Quelle est la différence entre les opérations classiques MPI (par exemple, MPI_Gather) et son opération correspondante “All” (par exemple, MPI_Allgather) ? **(1 point)**.

4. Quelle est la différence entre une architecture NUMA et UMA ? **(1 point)**.

Exercice 2 : Évaluation de la performance (5 points)

1) Nous avons un ordinateur qui passe 40% de son temps à gérer des opérations d’entrée /sortie. Combien faudra-t-il accélérer le système d'entrée / sortie pour que le speedup général soit de 1.3 ? **(2,5 points)**.

2) Supposons un code parallèle qui prend 120 secondes à s’exécuter avec 2 processeurs. Si l’efficacité de la parallélisation est de E = 0.32, calculer le temps d’exécution avec 1 processeur et le Speedup de la parallélisation. Est-ce que c’est intéressant de paralléliser ce code ? Justifiez **(2,5 points)**.

Exercice 3 : OpenMP (5 points)

3) Le produit scalaire de deux vecteurs est la somme des produits des éléments correspondants de ces deux vecteurs. Supposons que nous ayons deux vecteurs - {1, 2, 3} et {4, 5, 6} -, et le produit scalaire de ces vecteurs est 1*4 + 2*5 + 3*6 = 32.
Donner une version séquentielle ainsi qu’une version parallèle de ce produit scalaire en langage C en utilisant openMP **(2,5 points)**.

4) Soit **A** une matrice carrée *n x n*. La trace de **A** est la somme des éléments diagonaux de **A**. Écrire le programme OpenMP qui permet de calculer la trace d’une matrice **(2,5 points)**.

Exercice 4 : MPI (5 points)

1) Nous souhaitons paralléliser le calcul d'un produit matrice-vecteur $\mathbf{A} \times \mathbf{X} = \mathbf{Y}$, où **A** est une matrice de taille $n \times m$, **X** un vecteur de taille m et **Y** le vecteur résultat. Écrire le code MPI en privilégiant **des opérations collectives (3 points)**.

2) La transposée A^T d'une matrice A de taille n x m est la matrice obtenue en échangeant les lignes et les colonnes de A. **(2 points)**.

- Donner une version séquentielle.
- Donner une version parallèle utilisant les primitives MPI fonctionnant sur 4 processeurs avec m=n=4 puis avec m=n=8 en expliquant le découpage réalisé.
- Est-il possible d'utiliser des opérations collectives ? Si oui, donner le code MPI correspondant.

OpenMP 4.0 API C/C++ Syntax Quick Reference Card

OpenMP C/C++

OpenMP Application Program Interface (API) is a portable, scalable model that gives parallel programmers a simple and flexible interface for developing portable parallel applications. OpenMP supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures, including Unix platforms and Windows platforms. See www.openmp.org for specifications.

4.0 Refers to functionality new in version 4.0.

[n.n.n] refers to sections in the OpenMP API specification version 4.0, and [n.n.n] refers to version 3.1.

Directives

An OpenMP executable directive applies to the succeeding structured block or an OpenMP construct. Each directive starts with **#pragma omp**. The remainder of the directive follows the conventions of the C and C++ standards for compiler directives. 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 [2.5] [2.4]

Forms a team of threads and starts parallel execution.

#pragma omp parallel *[clause[[,]clause] ...]*
  structured-block

clause:
- **if**(*scalar-expression*)
- **num_threads**(*integer-expression*)
- **default**(**shared** | **none**)
- **private**(*list*)
- **firstprivate**(*list*)
- **shared**(*list*)
- **copyin**(*list*)
- **reduction**(*reduction-identifier*: *list*)
- 4.0 **proc_bind**(**master** | **close** | **spread**)

loop [2.7.1] [2.5.1]

Specifies that the iterations of associated loops will be executed in parallel by threads in the team in the context of their implicit tasks.

#pragma omp for *[clause[[,]clause] ...]*
  for-loops

clause:
- **private**(*list*)
- **firstprivate**(*list*)
- **lastprivate**(*list*)
- **reduction**(*reduction-identifier*: *list*)
- **schedule**(*kind*[, *chunk_size*])
- **collapse**(*n*)
- **ordered**
- **nowait**

kind:
- **static:** Iterations are divided into chunks of size *chunk_size* and assigned to threads in the team in round-robin fashion in order of thread number.
- **dynamic:** Each thread executes a chunk of iterations then requests another chunk until none remain.
- **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.
- **runtime:** The schedule and chunk size are taken from the *run-sched-var* ICV.

sections [2.7.2] [2.5.2]

A noniterative worksharing construct that contains a set of structured blocks that are to be distributed among and executed by the threads in a team.

#pragma omp sections *[clause[[,] clause] ...]*
  {
  *[***#pragma omp section***]*
    structured-block
  *[***#pragma omp section***]*
structured-block]
  ...
  }

clause:
- **private**(*list*)
- **firstprivate**(*list*)
- **lastprivate**(*list*)
- **reduction**(*reduction-identifier*: *list*)
- **nowait**

single [2.7.3] [2.5.3]

Specifies that the associated structured block is executed by only one of the threads in the team.

#pragma omp single *[clause[[,]clause] ...]*
  structured-block

clause:
- **private**(*list*)
- **firstprivate**(*list*)
- **copyprivate**(*list*)
- **nowait**

4.0 simd [2.8.1]

Applied to a loop to indicate that the loop can be transformed into a SIMD loop.

#pragma omp simd *[clause[[,]clause] ...]*
  for-loops

clause:
- **safelen**(*length*)
- **linear**(*list*[:*linear-step*])
- **aligned**(*list*[:*alignment*])
- **private**(*list*)
- **lastprivate**(*list*)
- **reduction**(*reduction-identifier*: *list*)
- **collapse**(*n*)

4.0 declare simd [2.8.2]

Enables the creation of one or more versions that can process multiple arguments using SIMD instructions from a single invocation from a SIMD loop.

#pragma omp declare simd *[clause[[,]clause] ...]*
  *[***#pragma omp declare simd** *[clause[[,]clause] ...]*
  ]
  [...]
    function definition or declaration

clause:
- **simdlen**(*length*)
- **linear**(*argument-list*[:*constant-linear-step*])
- **aligned**(*argument-list*[:*alignment*])
- **uniform**(*argument-list*)
- **inbranch**
- **notinbranch**

4.0 loop simd [2.8.3]

Specifies that a loop that can be executed concurrently using SIMD instructions, and that those iterations will also be executed in parallel by threads in the team.

#pragma omp for simd *[clause[[,]clause] ...]*
  for-loops

clause:
  Any accepted by the **simd** or **for** directives with identical meanings and restrictions.

4.0 target [data] [2.9.1, 2.9.2]

These constructs create a device data environment for the extent of the region. **target** also starts execution on the device.

#pragma omp target data *[clause[[,]clause] ...]*
  structured-block

#pragma omp target *[clause[[,]clause] ...]*
  structured-block

clause:
- **device**(*integer-expression*)
- **map**([*map-type*:] *list*)
- **if**(*scalar-expression*)

4.0 target update [2.9.3]

Makes the corresponding list items in the device data environment consistent with their original list items, according to the specified motion clauses.

#pragma omp target update *clause[[,]clause] ,...]*

clause is motion-clause or one of:
- **device**(*integer-expression*)
- **if**(*scalar-expression*)

motion-clause:
- **to**(*list*)
- **from**(*list*)

4.0 declare target [2.9.4]

A declarative directive that specifies that variables and functions are mapped to a device.

#pragma omp declare target
declarations-definition-seq
#pragma omp end declare target

4.0 teams [2.9.5]

Creates a league of thread teams where the master thread of each team executes the region.

#pragma omp teams *[clause[[,]clause] ,...]*
  structured-block

clause:
- **num_teams**(*integer-expression*)
- **thread_limit**(*integer-expression*)
- **default**(**shared** | **none**)
- **private**(*list*)
- **firstprivate**(*list*)
- **shared**(*list*)
- **reduction**(*reduction-identifier*: *list*)

4.0 distribute [simd] [2.9.6, 2.9.7]

distribute specifies loops which are executed by the thread teams. **distribute simd** specifies loops which are executed concurrently using SIMD instructions.

#pragma omp distribute *[clause[[,]clause] ...]*
  for-loops

#pragma omp distribute simd *[clause[[,]clause] ...]*
  for-loops

clause:
- **private**(*list*)
- **firstprivate**(*list*)
- **collapse**(*n*)
- **dist_schedule**(*kind*[, *chunk_size*])

4.0 distribute parallel for [simd] [2.9.8, 2.9.9]

These constructs specify a loop that can be executed in parallel [using SIMD semantics in the simd case] by multiple threads that are members of multiple teams.

#pragma omp distribute parallel for *[clause[[,]clause] ...]*
  for-loops

#pragma omp distribute parallel for simd *[clause[[,]clause] ...]*
  for-loops

clause: See *clause* for **distribute**

Continued

Directives (Continued)

parallel loop [2.10.1] [2.6.1]

Shortcut for specifying a **parallel** construct containing one or more associated loops and no other statements.

#pragma omp parallel for *[clause[[,]clause] ...]*
    for-loop

clause: Any accepted by the **parallel** or **for** directives, except the **nowait** clause, with identical meanings and restrictions.

parallel sections [2.10.2] [2.6.2]

Shortcut for specifying a **parallel** construct containing one **sections** construct and no other statements.

#pragma omp parallel sections *[clause[[,]clause] ...]*
    {
    *[***#pragma omp section***]*
        structured-block
    *[***#pragma omp section**
        structured-block]
    ...
    }

clause: Any of the clauses accepted by the **parallel** or **sections** directives, except the **nowait** clause, with identical meanings and restrictions.

4.0 parallel loop simd [2.10.4]

Shortcut for specifying a **parallel** construct containing one loop SIMD construct and no other statements.

#pragma omp parallel for simd *[clause[[,]clause] ...]*
    for-loops

clause: Any accepted by the **parallel**, **for** or **simd** directives, except the **nowait** clause, with identical meanings and restrictions.

4.0 target teams [2.10.5]

Shortcut for specifying a **target** construct containing a **teams** construct.

#pragma omp target teams *[clause[[,]clause] ...]*
    structured-block

clause: See *clause* for **target** or **teams**

4.0 teams distribute [simd] [2.10.6, 2.10.7]

Shortcuts for specifying a **teams** construct containing a **distribute [simd]** construct.

#pragma omp teams distribute *[clause[[,]clause] ...]*
    for-loops
#pragma omp teams distribute simd *[clause[[,]clause] ...]*
    for-loops

clause: Any *clause* used for **teams** or **distribute [simd]**

4.0 target teams distribute [simd] [2.10.8, 2.10.9]

Shortcuts for specifying a **target** construct containing a **teams distribute [simd]** construct.

#pragma omp target teams distribute *[clause[[,]clause] ...]*
    for-loops
#pragma omp target teams distribute simd *[clause[[,]clause] ...]*
    for-loops

clause: Any *clause* used for **target** or **teams distribute [simd]**

4.0 teams distribute parallel for [simd] [2.10.10, 12]

Shortcuts for specifying a **teams** construct containing a **distribute parallel for [simd]** construct.

#pragma omp teams distribute parallel for *[clause[[,]clause] ...]*
    for-loops
#pragma omp teams distribute parallel for simd *[clause[[,]clause] ...]*
    for-loops

clause: Any *clause* used for **teams** or **distribute parallel for [simd]**

4.0 target teams distribute parallel for [simd] [2.10.11, 13]

Shortcut for specifying a **target** construct containing a **teams distribute parallel for [simd]** construct.

**#pragma omp target teams distribute parallel for **
    [clause[[,]clause] ...]
    for-loops

**#pragma omp target teams distribute parallel for simd **
    [clause[[,]clause] ...]
    for-loops

clause: Any *clause* used for **target** or **teams distribute parallel for [simd]**

task [2.11.1] [2.7.1]

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

#pragma omp task *[clause[[,]clause] ...]*
    structured-block

clause:
- **if**(*scalar-expression*)
- **final**(*scalar-expression*)
- **untied**
- **default**(**shared** | **none**)
- **mergeable**
- **private**(*list*)
- **firstprivate**(*list*)
- **shared**(*list*)
- 4.0 **depend**(*dependence-type*: *list*)

The list items that appear in the **depend** clause may include array sections.

dependence-type: The generated task will be a dependent task of all previously generated sibling tasks that reference at least one of the list items...

- **in:** ...in an **out** or **inout** clause.
- **out** and **inout**: ...in an **in**, **out**, or **inout** clause.

taskyield [2.11.2] [2.7.2]

Specifies that the current task can be suspended in favor of execution of a different task.

#pragma omp taskyield

master [2.12.1] [2.8.1]

Specifies a structured block that is executed by the master thread of the team.

#pragma omp master
    structured-block

critical [2.12.2] [2.8.2]

Restricts execution of the associated structured block to a single thread at a time.

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

barrier [2.12.3] [2.8.3]

Specifies an explicit barrier at the point at which the construct appears.

#pragma omp barrier

taskwait [2.12.4] [2.8.4], 4.0 taskgroup [2.12.5]

These constructs each specify a wait on the completion of child tasks of the current task. **taskgroup** also waits for descendant tasks.

#pragma omp taskwait

#pragma omp taskgroup
    structured-block

atomic [2.12.6] [2.8.5]

Ensures that a specific storage location is accessed atomically. *[seq_cst]* is 4.0.

#pragma omp atomic *[***read** | **write** | **update** | **capture***]*
    *[***seq_cst***]*
    expression-stmt

#pragma omp atomic capture *[***seq_cst***]*
    structured-block

where *expression-stmt* may be one of:

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

(Continued >)

atomic (continued)

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

{v = x; x binop= expr;}
{v = x; x = x binop expr;}
{x = x binop expr; v = x;}
{v = x; x = expr;}
{v = x; ++x;}
{x++; v = x;}
{v = x; --x;}
{x--; v = x;}
{x binop= expr; v = x;}
{v = x; x = expr binop x;}
{x = expr binop x; v = x;}
{v = x; x++;}
{++x; v = x;}
{v = x; x--;}
{--x; v = x;}

flush [2.12.7] [2.8.6]

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)]*

ordered [2.12.8] [2.8.7]

Specifies a structured block in a loop region that will be executed in the order of the loop iterations.

#pragma omp ordered
    structured-block

4.0 cancel [2.13.1]

Requests cancellation of the innermost enclosing region of the type specified. The **cancel** directive may not be used in place of the statement following an **if**, **while**, **do**, **switch**, or **label**.

#pragma omp cancel *construct-type-clause[[,] if-clause]*

construct-type-clause:
- **parallel**
- **sections**
- **for**
- **taskgroup**

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

4.0 cancellation point [2.13.2]

Introduces a user-defined cancellation point at which tasks check if cancellation of the innermost enclosing region of the type specified has been requested.

#pragma omp cancellation point *construct-type-clause*

construct-type-clause:
- **parallel**
- **sections**
- **for**
- **taskgroup**

threadprivate [2.14.2] [2.9.2]

Specifies that variables are replicated, with each thread having its own copy. Each copy of a threadprivate variable is initialized once prior to the first reference to that copy.

#pragma omp threadprivate(*list*)

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

4.0 declare reduction [2.15]

Declares a *reduction-identifier* that can be used in a **reduction** clause.

#pragma omp declare reduction(*reduction-identifier* **:** *typename-list* **:** *combiner*) *[initializer-clause]*

reduction-identifier: A base language identifier or one of the following operators: +, -, *, &, |, ^, && and ||
In C++, this may also be an *operator-function-id*

typename-list: A list of type names

combiner: An expression

initializer-clause: **initializer** (**omp_priv** = *initializer* | *function-name* (*argument-list*))

Runtime Library Routines

Return types are shown in green.

Execution environment routines affect and monitor threads, processors, and the parallel environment. The library routines are external functions with “C” linkage.

Execution Environment Routines

omp_set_num_threads [3.2.1] [3.2.1]
Affects the number of threads used for subsequent parallel regions not specifying a **num_threads** clause, by setting the value of the first element of the *nthreads-var* ICV of the current task to *num_threads*.

void **omp_set_num_threads**(int *num_threads*);

omp_get_num_threads [3.2.2] [3.2.2]
Returns the number of threads in the current team. The binding region for an **omp_get_num_threads** region is the innermost enclosing **parallel** region.

int **omp_get_num_threads**(void);

omp_get_max_threads [3.2.3] [3.2.3]
Returns an upper bound on the number of threads that could be used to form a new team if a **parallel** construct without a **num_threads** clause were encountered after execution returns from this routine.

int **omp_get_max_threads**(void);

omp_get_thread_num [3.2.4] [3.2.4]
Returns the thread number of the calling thread within the current team.

int **omp_get_thread_num**(void);

omp_get_num_procs [3.2.5] [3.2.5]
Returns the number of processors that are available to the device at the time the routine is called.

int **omp_get_num_procs**(void);

omp_in_parallel [3.2.6] [3.2.6]
Returns *true* if the *active-levels-var* ICV is greater than zero; otherwise it returns *false*.

int **omp_in_parallel**(void);

omp_set_dynamic [3.2.7] [3.2.7]
Returns the value of the *dyn-var* ICV, which indicates if dynamic adjustment of the number of threads is enabled or disabled.

void **omp_set_dynamic**(int *dynamic_threads*);

omp_get_dynamic [3.2.8] [3.2.8]
This routine returns the value of the *dyn-var* ICV, which is *true* if dynamic adjustment of the number of threads is enabled for the current task.

int **omp_get_dynamic**(void);

4.0 omp_get_cancellation [3.2.9]
Returns the value of the *cancel-var* ICV, which controls the behavior of cancel construct and cancellation points.

int **omp_get_cancellation**(void);

omp_set_nested [3.2.10] [3.2.9]
Enables or disables nested parallelism, by setting the *nest-var* ICV.

void **omp_set_nested**(int *nested*);

omp_get_nested [3.2.11] [3.2.10]
Returns the value of the *nest-var* ICV, which indicates if nested parallelism is enabled or disabled.

int **omp_get_nested**(void);

omp_set_schedule [3.2.12] [3.2.11]
Affects the schedule that is applied when **runtime** is used as schedule kind.

void **omp_set_schedule**(omp_sched_t *kind*, int *modifier*);

kind: one of the folowing, or an implementation-defined schedule:
omp_sched_static = 1
omp_sched_dynamic = 2
omp_sched_guided = 3
omp_sched_auto = 4

omp_get_schedule [3.2.13] [3.2.12]
Returns the value of *run-sched-var* ICV, which is the schedule applied when **runtime** schedule is used.

void **omp_get_schedule**(
omp_sched_t **kind*, int **modifier*);

See *kind* above.

omp_get_thread_limit [3.2.14] [3.2.13]
Returns the value of the *thread-limit-var* ICV, which is the maximum number of OpenMP threads available.

int **omp_get_thread_limit**(void);

omp_set_max_active_levels [3.2.15] [3.2.14]
Limits the number of nested active parallel regions, by setting *max-active-levels-var* ICV.

void **omp_set_max_active_levels**(int *max_levels*);

omp_get_max_active_levels [3.2.16] [3.2.15]
Returns the value of *max-active-levels-var* ICV, which determines the maximum number of nested active **parallel** regions.

int **omp_get_max_active_levels**(void);

omp_get_level [3.2.17] [3.2.16]
For the enclosing device region, returns the *levels-vars* ICV, which is the number of nested **parallel** regions that enclose the task containing the call.

int **omp_get_level**(void);

omp_get_ancestor_thread_num [3.2.18] [3.2.17]
Returns, for a given nested level of the current thread, the thread number of the ancestor of the current thread.

int **omp_get_ancestor_thread_num**(int *level*);

omp_get_team_size [3.2.19] [3.2.18]
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_team_size**(int *level*);

omp_get_active_level [3.2.20] [3.2.19]
Returns the value of the *active-level-vars* ICV, which determines the number of active, nested **parallel** regions enclosing the task that contains the call.

int **omp_get_active_level**(void);

omp_in_final [3.2.21] [3.2.20]
Returns *true* if the routine is executed in a final task region; otherwise, it returns *false*.

int **omp_in_final**(void);

4.0 omp_get_proc_bind [3.2.22]
Returns the thread affinity policy to be used for the subsequent nested **parallel** regions that do not specify a *proc_bind* clause.

omp_proc_bind_t **omp_get_proc_bind**(void);

Returns one of:
omp_proc_bind_false = 0
omp_proc_bind_true = 1
omp_proc_bind_master = 2
omp_proc_bind_close = 3
omp_proc_bind_spread = 4

4.0 omp_set_default_device [3.2.23]
Controls the default target device by assigning the value of the *default-device-var* ICV.

void **omp_set_default_device**(int *device_num*);

4.0 omp_get_default_device [3.2.24]
Returns the default target device.

int **omp_get_default_device**(void);

4.0 omp_get_num_devices [3.2.25]
Returns the number of target devices.

int **omp_get_num_devices**(void);

4.0 omp_get_num_teams [3.2.26]
Returns the number of teams in the current **teams** region, or 1 if called from outside of a **teams** region.

int **omp_get_num_teams**(void);

4.0 omp_get_team_num [3.2.27]
Returns the team number of calling thread. The team number is an integer between 0 and one less than the value returned by **omp_get_num_teams**, inclusive.

int **omp_get_team_num**(void);

4.0 omp_is_initial_device [3.2.28]
Returns *true* if the current task is executing on the host device; otherwise, it returns *false*.

int **omp_is_initial_device**(void);

Lock Routines
General-purpose lock routines. Two types of locks are supported: *simple locks* and *nestable locks*. A nestable lock can be set multiple times by the same task before being unset; a simple lock cannot be set if it is already owned by the task trying to set it.

Initialize lock [3.3.1] [3.3.1]
Initialize an OpenMP lock.

void **omp_init_lock**(omp_lock_t **lock*);
void **omp_init_nest_lock**(omp_nest_lock_t **lock*);

Destroy lock [3.3.2] [3.3.2]
Ensure that the OpenMP lock is uninitialized.

void **omp_destroy_lock**(omp_lock_t **lock*);
void **omp_destroy_nest_lock**(omp_nest_lock_t **lock*);

Set lock [3.3.3] [3.3.3]
Sets an OpenMP lock. The calling task region is suspended until the lock is set.

void **omp_set_lock**(omp_lock_t **lock*);
void **omp_set_nest_lock**(omp_nest_lock_t **lock*);

Unset lock [3.3.4] [3.3.4]
Unsets an OpenMP lock.

void **omp_unset_lock**(omp_lock_t **lock*);
void **omp_unset_nest_lock**(omp_nest_lock_t **lock*);

Test lock [3.3.5] [3.3.5]
Attempt to set an OpenMP lock but do not suspend execution of the task executing the routine.

int **omp_test_lock**(omp_lock_t **lock*);
int **omp_test_nest_lock**(omp_nest_lock_t **lock*);

Timing Routines
Timing routines support a portable wall clock timer.These record elapsed time per-thread and are not guaranteed to be globally consistent across all the threads participating in an application.

omp_get_wtime [3.4.1] [3.4.1]
Returns elapsed wall clock time in seconds.

double **omp_get_wtime**(void);

omp_get_wtick [3.4.2] [3.4.2]
Returns the precision of the timer (seconds between ticks) used by **omp_get_wtime**.

double **omp_get_wtick**(void);

Environment Variables [4]

Environment variable names are upper case, and the values assigned to them are case insensitive and may have leading and trailing white space.

4.0 [4.11] **OMP_CANCELLATION** *policy*
Sets the *cancel-var* ICV. *policy* may be **true** or **false**. If **true**, the effects of the cancel construct and of cancellation points are enabled and cancellation is activated

4.0 [4.13] **OMP_DEFAULT_DEVICE** *device*
Sets the *default-device-var* ICV that controls the default device number to use in device constructs.

4.0 [4.12] **OMP_DISPLAY_ENV** *var*
If *var* is **TRUE**, instructs the runtime to display the OpenMP version number and the value of the ICVs associated with the environment variables as *name=value* pairs. If *var* is **VERBOSE**, the runtime may also display vendor-specific variables. If *var* is **FALSE**, no information is displayed.

[4.3] [4.3] **OMP_DYNAMIC** *dynamic*
Sets the *dyn-var* ICV. If **true**, the implementation may dynamically adjust the number of threads to use for executing **parallel** regions.

[4.9] [4.8] **OMP_MAX_ACTIVE_LEVELS** *levels*
Sets the *max-active-levels-var* ICV that controls the maximum number of nested active **parallel** regions.

[4.6] [4.5] **OMP_NESTED** *nested*
Sets the *nest-var* ICV to enable or to disable nested parallelism. Valid values for *nested* are **true** or **false**.

[4.2] [4.2] **OMP_NUM_THREADS** *list*
Sets the *nthreads-var* ICV for the number of threads to use for **parallel** regions.

4.0 [4.5] **OMP_PLACES** *places*
Sets the *place-partition-var* ICV that defines the OpenMP places available to the execution environment. *places* is an abstract name (**threads**, **cores**, **sockets**, or implementation-defined), or a list of non-negative numbers.

[4.4] [4.4] **OMP_PROC_BIND** *policy*
Sets the value of the global *bind-var* ICV, which sets the thread affinity policy to be used for parallel regions at the corresponding nested level. *policy* can be the values **true**, **false**, or a comma-separated list of **master**, **close**, or **spread** in quotes.

[4.1] [4.1] **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**.

[4.7] [4.6] **OMP_STACKSIZE** *size*[**B** | **K** | **M** | **G**]
Sets the *stacksize-var* ICV that specifies the size of the stack for threads created by the OpenMP implementation. *size* is a positive integer that specifies stack size. If unit is not specified, *size* is measured in kilobytes (K).

[4.10] [4.9] **OMP_THREAD_LIMIT** *limit*
Sets the *thread-limit-var* ICV that controls the number of threads participating in the OpenMP program.

[4.8] [4.7] **OMP_WAIT_POLICY** *policy*
Sets the *wait-policy-var* ICV that provides a hint to an OpenMP implementation about the desired behavior of waiting threads. Valid values for *policy* are **ACTIVE** (waiting threads consume processor cycles while waiting) and **PASSIVE**.

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 list items. All list items appearing in a clause must be visible, according to the scoping rules of the base language. Not all of the clauses listed in this section are valid on all directives. The set of clauses that is valid on a particular directive is described with the directive.

Data Sharing Attribute Clauses [2.14.3] [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**)
Explicitly determines the default data-sharing attributes of variables that are referenced in a **parallel**, **task**, or **teams** construct, causing all variables referenced in the construct that have implicitly determined data-sharing attributes to be shared.

shared(*list*)
Declares one or more list items to be shared by tasks generated by a **parallel**, **task**, or **teams** construct. The programmer must ensure that storage shared by an explicit **task** region does not reach the end of its lifetime before the explicit task region completes its execution.

private(*list*)
Declares one or more list items to be private to a task or a SIMD lane. Each task that references a list item that appears in a **private** clause in any statement in the construct receives a new list item.

firstprivate(*list*)
Declares 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 or to a SIMD lane, and causes the corresponding original list item to be updated after the end of the region.

4.0 **linear**(*list[:linear-step]*)
Declares one or more list items to be private to a SIMD lane and to have a linear relationship with respect to the iteration space of a loop.

reduction(*reduction-identifier*:*list*)
Specifies a *reduction-identifier* and one or more list items. The *reduction-identifier* must match a previously declared *reduction-identifier* of the same name and type for each of the list items.

Operators for reduction (initialization values)			
+	(0)	|	(0)
*	(1)	^	(0)
-	(0)	&&	(1)
&	(~0)	||	(0)
max (Least representable number in **reduction** list item type)			
min (Largest representable number in **reduction** list item type)			

Data Copying Clauses [2.14.4] [2.9.4]

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.

4.0 Map Clause [2.14.5]

map(*[map-type*:*]list*)
Map a variable from the task's data environment to the device data environment associated with the construct.
map-type:

- **alloc**: On entry to the region each new corresponding list item has an undefined initial value.
- **to**: On entry to the region each new corresponding list item is initialized with the original list item's value.
- **from**: On exit from the region the corresponding list item's value is assigned to each original list item.

(Continued >)

- **tofrom:** (Default) On entry to the region each new corresponding list item is initialized with the original list item's value, and on exit from the region the corresponding list item's value is assigned to each original list item.

4.0 SIMD Clauses [2.8.1]

safelen(*length*)
If used then no two iterations executed concurrently with SIMD instructions can have a greater distance in the logical iteration space than its value.

collapse(*n*)
A constant positive integer expression that specifies how many loops are associated with the loop construct.

simdlen(*length*)
A constant positive integer expression that specifies the number of concurrent arguments of the function.

aligned(*argument-list[:alignment]*)
Declares one or more list items to be aligned to the specified number of bytes. *alignment*, if present, must be a constant positive integer expression. If no optional parameter is specified, implementation-defined default alignments for SIMD instructions on the target platforms are assumed.

uniform(*argument-list*)
Declares one or more arguments to have an invariant value for all concurrent invocations of the function in the execution of a single SIMD loop.

inbranch
Specifies that the function will always be called from inside a conditional statement of a SIMD loop.

notinbranch
Specifies that the function will never be called from inside a conditional statement of a SIMD loop.

MPI Quick Reference in C

#include <mpi.h>

Environmental Management:

int **MPI_Init**(int *argc, char **argv[])

int **MPI_Finalize**(void)

int **MPI_Initialized**(int *flag)

int **MPI_Finalized**(int *flag)

int **MPI_Comm_size**(MPI_Comm comm, int *size)

int **MPI_Comm_rank**(MPI_Comm comm, int *rank)

int **MPI_Abort**(MPI_Comm comm, int errorcode)

double **MPI_Wtime**(void)

double **MPI_Wtick**(void)

Blocking Point-to-Point-Communication:

int **MPI_Send** (void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Related: MPI_Bsend, MPI_Ssend, MPI_Rsend

int **MPI_Recv** (void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

int **MPI_Probe** (int source, int tag, MPI_Comm comm, MPI_Status *status)

int **MPI_Get_count** (MPI_Status *status, MPI_Datatype datatype, int *count)

Related: MPI_Get_elements

int **MPI_Sendrecv** (void *sendbuf, int sendcount, MPI_Datatype sendtype, int dest, int sendtag, void *recvbuf, int recvcount, MPI_Datatype recvtype, int source, int recvtag, MPI_Comm comm, MPI_Status *status)

int **MPI_Sendrecv_replace** (void *buf, int count, MPI_Datatype datatype, int dest, int sendtag, int source, int recvtag, MPI_Comm comm, MPI_Status *status)

int **MPI_Buffer_attach** (void *buffer, int size)

int **MPI_Buffer_detach** (void *bufferptr, int *size)

Non-Blocking Point-to-Point-Communication:

int **MPI_Isend** (void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)

Related: MPI_Ibsend, MPI_Issend, MPI_Irsend

int **MPI_Irecv** (void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)

int **MPI_Iprobe** (int source, int tag, MPI_Comm comm, int *flag, MPI_Status *status)

int **MPI_Wait** (MPI_Request *request, MPI_Status *status)

int **MPI_Test** (MPI_Request *request, int *flag, MPI_Status *status)

int **MPI_Waitall** (int count, MPI_Request request_array[], MPI_Status status_array[])

Related: MPI_Testall

int **MPI_Waitany** (int count, MPI_Request request_array[], int *index, MPI_Status *status)

Related: MPI_Testany

int **MPI_Waitsome** (int incount, MPI_Request request_array[], int *outcount, int index_array[], MPI_Status status_array[])

Related: MPI_Testsome,

int **MPI_Request_free** (MPI_Request *request)

Related: MPI_Cancel

int **MPI_Test_cancelled** (MPI_Status *status, int *flag)

Collective Communication:

int **MPI_Barrier** (MPI_Comm comm)

int **MPI_Bcast** (void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

int **MPI_Gather** (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

int **MPI_Gatherv** (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount_array[], int displ_array[], MPI_Datatype recvtype, int root, MPI_Comm comm)

int **MPI_Scatter** (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

int **MPI_Scatterv** (void *sendbuf, int sendcount_array[], int displ_array[] MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

int **MPI_Allgather** (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

Related: MPI_Alltoall

int **MPI_Allgatherv** (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount_array[], int displ_array[], MPI_Datatype recvtype, MPI_Comm comm)

Related: MPI_Alltoallv

int **MPI_Reduce** (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

int **MPI_Allreduce** (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

Related: MPI_Scan, MPI_Exscan

int **MPI_Reduce_scatter** (void *sendbuf, void *recvbuf, int recvcount_array[], MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

int **MPI_Op_create** (MPI_User_function *func, int commute, MPI_Op *op)

int **MPI_Op_free** (MPI_Op *op)

Derived Datatypes:

int **MPI_Type_commit** (MPI_Datatype *datatype)

int **MPI_Type_free** (MPI_Datatype *datatype)

int **MPI_Type_contiguous** (int count, MPI_Datatype oldtype, MPI_Datatype *newtype)

int **MPI_Type_vector** (int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)

int **MPI_Type_indexed** (int count, int blocklength_array[], int displ_array[], MPI_Datatype oldtype, MPI_Datatype *newtype)

int **MPI_Type_create_struct** (int count, int blocklength_array[], MPI_Aint displ_array[], MPI_Datatype oldtype_array[], MPI_Datatype *newtype)

int **MPI_Type_create_subarray** (int ndims, int size_array[], int subsize_array[], int start_array[], int order, MPI_Datatype oldtype, MPI_Datatype *newtype)

int **MPI_Get_address** (void *location, MPI_Aint *address)

int **MPI_Type_size** (MPI_Datatype *datatype, int *size)

int **MPI_Type_get_extent** (MPI_Datatype datatype, MPI_Aint *lb, MPI_Aint *extent)

int **MPI_Pack** (void *inbuf, int incount, MPI_Datatype datatype, void *outbuf, int outcount, int *position, MPI_Comm comm)

int **MPI_Unpack** (void *inbuf, int insize, int *position, void *outbuf, int outcount, MPI_Datatype datatype, MPI_Comm comm)

int **MPI_Pack_size** (int incount, MPI_Datatype datatype, MPI_Comm comm, int *size)

Related: MPI_Type_create_hvector, MPI_Type_create_hindexed, MPI_Type_create_indexed_block, MPI_Type_create_darray, MPI_Type_create_resized, MPI_Type_get_true_extent, MPI_Type_dup, MPI_Pack_external, MPI_Unpack_external, MPI_Pack_external_size

Groups and Communicators:

int **MPI_Group_size** (MPI_Group group, int *size)

int **MPI_Group_rank** (MPI_Group group, int *rank)

int **MPI_Comm_group** (MPI_Comm comm, MPI_Group *group)

int **MPI_Group_translate_ranks** (MPI_Group group1, int n, int rank1_array[], MPI_Group group2, int rank2_array[])

int **MPI_Group_compare** (MPI_Group group1, MPI_Group group2, int *result)

MPI_IDENT, MPI_COMGRUENT, MPI_SIMILAR, MPI_UNEQUAL

int **MPI_Group_union** (MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

Related: MPI_Group_intersection, MPI_Group_difference

int **MPI_Group_incl** (MPI_Group group, int n, int rank_array[], MPI_Group *newgroup)

Related: MPI_Group_excl

int **MPI_Comm_create** (MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm)

int **MPI_Comm_compare** (MPI_Comm comm1, MPI_Comm comm2, int *result)

MPI_IDENT, MPI_COMGRUENT, MPI_SIMILAR, MPI_UNEQUAL

int **MPI_Comm_dup** (MPI_Comm comm, MPI_Comm *newcomm)

int **MPI_Comm_split** (MPI_Comm comm, int color, int key, MPI_Comm *newcomm)

int **MPI_Comm_free** (MPI_Comm *comm)

Topologies:

int **MPI_Dims_create** (int nnodes, int ndims, int *dims)

int **MPI_Cart_create** (MPI_Comm comm_old, int ndims, int dims_array[], int periods_array[], int reorder, MPI_Comm *comm_cart)

int **MPI_Cart_shift** (MPI_Comm comm, int direction, int disp, int *rank_source, int *rank_dest)

int **MPI_Cartdim_get** (MPI_Comm comm, int *ndim)

int **MPI_Cart_get** (MPI_Comm comm, int naxdim, int *dims, int *periods, int *coords)

int **MPI_Cart_rank** (MPI_Comm comm, int coords_array[], int *rank)

int **MPI_Cart_coords** (MPI_Comm comm, int rank, int maxdims, int *coords)

int **MPI_Cart_sub** (MPI_Comm comm_old, int remain_dims_array[], MPI_Comm *comm_new)

int **MPI_Cart_map** (MPI_Comm comm_old, int ndims, int dims_array[], int periods_array[], int *new_rank)

int **MPI_Graph_create** (MPI_Comm comm_old, int nnodes, int index_array[], int edges_array[], int reorder, MPI_Comm *comm_graph)

int **MPI_Graph_neighbors_count** (MPI_Comm comm, int rank, int *nneighbors)

int **MPI_Graph_neighbors** (MPI_Comm comm, int rank, int maxneighbors, int *neighbors)

int **MPI_Graphdims_get** (MPI_Comm comm, int *nnodes, int *nedges)

int **MPI_Graph_get** (MPI_Comm comm, int maxindex, int maxedges, int *index, int *edges)

int **MPI_Graph_map** (MPI_Comm comm_old, int nnodes, int index_array[], int edges_array[], int *new_rank)

int **MPI_Topo_test** (MPI_Comm comm, int *topo_type)

Wildcards:

MPI_ANY_TAG, MPI_ANY_SOURCE

Basic Datatypes:

MPI_CHAR, MPI_SHORT, MPI_INT, MPI_LONG, MPI_UNSIGNED_CHAR, MPI_UNSIGNED_SHORT, MPI_UNSIGNED, MPI_UNSIGNED_LONG MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE, MPI_BYTE, MPI_PACKED

Predefined Groups and Communicators:

MPI_GROUP_EMPTY, MPI_GROUP_NULL, MPI_COMM_WORLD, MPI_COMM_SELF, MPI_COMM_NULL

Reduction Operations:

MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_BAND, MPI_BOR, MPI_BXOR, MPI_LAND, MPI_LOR, MPI_LXOR

Status Object:

status.MPI_SOURCE, status.MPI_TAG, status.MPI_ERROR