Libro de uso OpenMP

Daniel Asensi Roch

October 27, 2021

Contents

1 ¿Qué es OpenMp

OpenMp es una API para la programación multiproceso de memoria compartida en multiples plataformas, permite el uso de concurrencia en los lenguajes C++, C y Fortran, utilizando de base el modelo de Fork-Join.

OpenMp se compone de un conjunto de directivas para el compilador, rutinas de bibliotecas y variables de entorno que influyen en el comportamiento del programa, es un modelo de programación facilmente portable y escalable, que proporciona una interfaz simple y flexible para el desarrollo de aplicaciones paralelas.

2 Tarea 0

2.1 La variable Chunk

Openmp divide las diferentes directivas del programa en diferentes fragmentos llamados Chunk, estos fragmentos serán indicados con un integer, este tendrá diferentes clausulas de planificación, que repartirá la carga a procesar en diferentes hilos del tamaño del chunk.

2.2 Explicación del Pragma

2.2.1 Uso de shared

En el programa que disponemos actualmente disponemos una variable compartida por todos los procesos llamada Shared, esta será completamente pública y todos los procesos podrán acceder a ella, sus argumentos son las variables accesibles por los procesos.

2.2.2 Uso del private

Los procesos poseen copias de las variables que serán modificadas dentro de cada uno de ellos pero que no afectarán a la solución final, estas variables copia que deben de tener todos los procesos se especifican en el private.

2.3 Uso de Schedule

Schedule será el planificador de tareas, este se encargará de asignar las diferentes iteraciones dentro de los hilos, además este dispone de diferentes maneras de asignación:

2.3.1 Schedule: static

En esta implementación de Schedule, se asigna el entero de que define la variable chunk de manera estática y no modificable, contando cada uno de los hilos con el mismo número de chunks

2.3.2 Schedule:dynamic

En esta implementación de Schedule, se asiganará una cantidad determinada de chunks cada vez que un hilo este sin trabajo.

2.3.3 Schedule:guided

En esta implementación de Schedule, cada hilo toma iteraciones dinámicamente y progresivamente va tomando menos iteraciones.

2.3.4 Schedule:auto

En esta implementación de Schedule, el tipo de schedule es asiganado por el compilador o en tiempo de ejecución.

2.3.5 Schedule:runtime

En esta implementación de Schedule, el tamaño del chunk y el tipo de schedule viene dado por el run-sched-var ICV.

2.4 Tiempo y otras medidas de rendimiento

La aceleración y la eficiencia son las dos únicas medidas que podemos utilizar para comprobar que la paralización en OpenMp esta siendo llevada acabo de la manera más optima.

3 Tarea 1: Estudio del API OpenMP

En OpenMp todas las directivas del programa comienzan con **pragma omp**, las directivas siguen el convenio de C y C++ para los compiladores, mediante bloques estructurados. Un bloque estructurado es una declaración única o una declaración compuesta con una sola entrada en la parte superior y una única salida en la parte inferior.

3.1 Directivas

3.1.1 Directiva Parallel

Forma un equipo de hilos y inicia la ejecución en los mismos de manera paralela.

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

Se pueden utilizar diferentes clausulas:

1. clause:
2. if([parallel :] scalar-expression)
3. num_threads(integer-expression)
4. default(shared — none)
5. private(list)
6. firstprivate(list)
7. shared(list)
8. copyin(list)
9. reduction(reduction-identifier: list)
10. proc_bind(master — close — spread)

3.1.2 Directiva For

Especifica que las iteraciones asociadas a los bucles serán ejecutadas en paralelo por los hilos en el equipo de las tareas implicitas

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

Se pueden utilizar diferentes clausulas:

1. clause:
2. private(list)
3. firstprivate(list)
4. lastprivate(list)
5. linear(list [: linear-step])
6. reduction(reduction-identifier : list)
7. schedule([modifier [, modifier] :] kind[, chunk_size])
8. collapse(n)
9. ordered[(n)]

10. nowait

Puede implementar diferentes modificadores:

- Monotónico: cada hilo ejecuta los fragmentos que es asignados en orden de iteración lógica creciente.
- No monotónico: los fragmentos se asignan a los subprocesos en cualquier orden y el comportamiento de una aplicación que depende sobre el orden de ejecución de los fragmentos no se especifica.
- Simd: se ignora cuando el bucle no está asociado con una construcción SIMD, de lo contrario new_chunk_size para todos excepto el primer y último trozo es : [chunk_size/ simd_width] * simd_width

3.1.3 Directiva Sections

Una construcción de trabajo compartido no iterativo que contiene un conjunto de bloques estructurados que se distribuirán entre y ejecutado por los hilos en un equipo.

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

Se pueden utilizar diferentes clausulas:

1. clause:
2. private(list)
3. firstprivate(list)
4. lastprivate(list)
5. reduction(reduction-identifier: list)
6. nowait

3.1.4 Directiva Single

Especifica que se ejecuta el bloque estructurado asociado por solo uno de los hilos del equipo.

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

Se pueden utilizar diferentes clausulas:

1. clause:
2. private(list)
3. firstprivate(list)
4. copyprivate(list)
5. nowait

3.1.5 Directiva Simd

Aplicado a un bucle para indicar que el bucle se puede transformado en un bucle SIMD.

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

Se pueden utilizar diferentes clausulas:

1. safelen(length)
2. simdlen(length)
3. linear(list[: linear-step])
4. aligned(list[: alignment])
5. private(list)
6. lastprivate(list)
7. reduction(reduction-identifier : list)
8. collapse(n)

3.1.6 Directiva Declare Simd

Permite la creación de una o más versiones que pueden procesar múltiples argumentos usando instrucciones SIMD de una única invocación de un bucle SIMD.

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

Se pueden utilizar diferentes clausulas:

1. simdlen(length)
2. linear(linear-list[: linear-step])
3. aligned(argument-list[: alignment])
4. uniform(argument-list)
5. inbranch
6. notinbranch

3.1.7 Directiva For simd

Especifica que un bucle que se puede ejecutar al mismo tiempo utilizando instrucciones SIMD, y que esas iteraciones también ser ejecutado en paralelo por subprocesos en el equipo.

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

Clausulas: Cualquiera aceptado por las directivas simd o for con significados y restricciones idénticos.

3.1.8 Directiva Task

Define una tarea explícita. El entorno de datos de la tarea se crea de acuerdo con las cláusulas de atributos de intercambio de datos en construcción de la tarea y los valores predeterminados que se apliquen.

```
#pragma omp task [clause[ [, ]clause] ...]
structured-block
 function definition or declaration
```

Se pueden utilizar diferentes clausulas:

1. clause:
2. if([task :] scalar-expression)
3. final(scalar-expression)
4. untied
5. default(shared — none)
6. mergeable
7. private(list)
8. firstprivate(list)
9. shared(list)
10. depend(dependence-type : list)
11. priority(priority-value)

TaskLoop Specifies that the iterations of one or more associated loops will be executed in parallel using OpenMP tasks.

```
#pragma omp taskloop [clause[ [, ]clause] ...]
for-loops
```

1. clause: ([taskloop :] scalar-expression)
2. shared(list)
3. private(list)
4. firstprivate(list)
5. lastprivate(list)
6. default(shared — none)
7. grainsize(grain-size)
8. num_tasks(num-tasks)
9. collapse (n)
10. final(scalar-expression)
11. priority(priority-value)
12. untied
13. mergeable
14. nogroup

3.1.9 Directiva Master

La construcción master de un bloque estructurado especifica que es ejecutada por la hebra principal del equipo (master). No implica sincronizacion ni a la entrada ni a la salida.

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

3.1.10 Directiva Critical

La construcción master de un bloque estructurado especifica que La construcción crititал restringe la ejecución asociada al bloque por una sola hebra al mismo tiempo.

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

3.1.11 Directiva Barrier

La contrucción barrier especifica de forma explicita un punto de sincronización para todas las hebras.

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

3.1.12 Directiva Taskwait

Especifica una espera en la finalización de las tareas secundarias del tarea actual.

```
#pragma omp taskwait
```

3.1.13 Directiva Atomic

Garantiza que se acceda a una ubicación de almacenamiento específica atómicamente. Puede tomar una de las siguientes tres formas:

```
#pragma omp atomic [seq_cst[,]] atomic-clause [[,]seq_cst]
expression-stmt

#pragma omp atomic [seq_cst]
expression-stmt

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

atomic clause: read, write, update, or capture
```

if *atomic 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;*

3.1.14 Directiva Flush

Ejecuta la operación de vaciado de OpenMP, que hace la vista temporal de la memoria de un hilo consistente con memoria, y hace cumplir una orden en las operaciones de memoria de las variables.

```
#pragma omp flush [(list)]
```

3.1.15 Directiva Ordered

Especifica un bloque estructurado en un bucle, simd o bucle SIMD región que se ejecutará en el orden del bucle iteraciones.

```
    #pragma omp ordered [clause[[, ] clause]...]
        structured-block

    clause:
        threads
        simd
#pragma omp ordered clause[[[, ] clause]...]
    clause:
        depend (source)
        depend (sink : vec)
```

3.1.16 Directiva Threadprivate

Especifica que las variables se replican con cada hilo, teniendo su propia copia. Cada copia de una variable privada de hilo se inicializa una vez antes de la primera referencia a esa copia.

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

list: una lista separada por comas de variables de alcance de archivo, espacio de nombres o alcance de bloque estático que no tienen tipos incompletos.

3.2 Clausulas de compartición de datos

3.2.1 Default(shared—none)

Determina explícitamente los atributos predeterminados de intercambio de datos. de variables a las que se hace referencia en paralelo, equipos o construcción de generación de tareas, lo que hace que todas las variables a las que se hace referencia en la construcción que han determinado implícitamente los atributos de intercambio de datos que se compartirán.

3.2.2 shared(list)

Declara uno o más elementos de la lista para ser compartidos por tareas. generado por un paralelo, equipos o tareas que generan construir. El programador debe asegurarse de que el almacenamiento compartido por una región de tarea explícita no llega al final de su vida antes de que la región de tareas explícita complete su ejecución.

3.2.3 private(list)

Declara que uno o más elementos de la lista son privados para una tarea o un carril SIMD. Cada tarea que hace referencia a un elemento de la lista que aparece en una cláusula privada en cualquier declaración en el la construcción recibe un nuevo elemento de lista.

3.2.4 firstprivate(list)

Declara que los elementos de la lista son privados para una tarea e inicializa cada uno de ellos con el valor que el correspondiente el elemento original tiene cuando se encuentra la construcción.

3.2.5 lastprivate(list)

Declara que uno o más elementos de la lista son privados para un implícito tarea o en un carril SIMD, y provoca la correspondiente elemento de la lista original que se actualizará después del final de la región.

3.2.6 reduction(operator:list)

Especifica un identificador de reducción y uno o más elementos de la lista. El identificador de reducción debe coincidir con un identificador de reducción del mismo nombre y tipo para cada de los elementos de la lista.

3.2.7 linear(linear-list[:linear-step])

Declara que uno o más elementos de la lista son privados para un SIMD carril y tener una relación lineal con respecto a el espacio de iteración de un bucle.

3.2.8 nowait

Evita la sincronización implícita de las hebras al terminar el bloque de una directiva.

3.3 Clausulas de copia de datos

3.3.1 copyin(list)

Copia el valor del threadprivate del hilo maestro variable al hilo variable privada entre sí miembro del equipo que ejecuta la región paralela.

3.3.2 copyprivate(list)

Copia el valor del threadprivate del hilo maestro variable al hilo variable privada entre sí miembro del equipo que ejecuta la región paralela.

3.4 Tipos de planificadores para bucles

3.4.1 Static

Las iteraciones se dividen en trozos de tamaño chunk_size y asignado a subprocesos en el equipo en moda por turnos en orden de número de hilo.

3.4.2 Dynamic

Cada hebra ejecuta una parte de las iteraciones, a continuación pide otro parte hasta que no quedan trozos para su distribución.

3.4.3 Guided

Cada hilo ejecuta una gran cantidad de iteraciones luego solicita otro fragmento hasta que no queden fragmentos para ser asignado.

3.4.4 Auto

La decisión de la plantificación del reparto es delegada en el compilador o el sistema.

3.4.5 Runtime

El planificador y el tamaño de las partes las toma de run-sched-var (VIC)

3.5 Rutinas de ejecución

3.5.1 omp_set_num_threads

```
void omp_set_num_threads(int num_threads);
```

Afecta al numero de hebras usadas posteriormente por la regiones paralelas no especificadas por la clausula num_threads.

3.5.2 omp_get_num_threads

```
int omp_get_num_threads(void);
```

Devuelve el numero de hebras en el equipo actual.

3.5.3 omp_get_thread_num

```
int omp_get_thread_num(void);
```

Devuelve el identificador de la hebra actual, donde el identificador esta entre 0 y el tamaño máximo del equipo menos 1.

3.5.4 omp_get_num_procs

```
int omp_get_num_procs(void);
```

Devuelve el numero de procesadores disponibles para el programa.

3.5.5 omp_in_parallel

```
int omp_in_parallel(void);
```

Devuelve true si la llamada a la rutina se realiza dentro de una región paralela activa; en otro caso devuelve false.

3.5.6 omp_set_dynamic

```
void omp_set_dynamic(int dynamic_threads);
```

Activa o desactiva el ajuste dinámico del número de subprocesos disponibles.

3.5.7 omp_set_dynamic

```
int omp_get_dynamic(void);
```

Devuelve el valor de la variable de control interno (VIC) dyn-var, para determinar si el ajuste dinámico del número de procesos está activado o desactivado.

3.5.8 omp_set_nested

```
void omp_set_nested(int nested);
```

Activa o desactiva el paralelismo anidado, mediante la modificación de nest-var (VIC).

3.5.9 omp_get_nested

```
int omp_get_nested(void);
```

Devuelve el valor del nest-var ()VIC, que determina si el paralelismo anidado está activado o desactivado.

3.5.10 omp_set_schedule

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

Afecta al planificador por defecto usado en las rutinas, modifica la variable run-sched-var (VIC).

3.5.11 omp_get_schedule

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

Devuelve el planificador aplicado en las rutinas de planificación.

3.5.12 omp_get_thread_limit

```
int omp_get_thread_limit(void)
```

Devuelve el numero máximo de hebras disponibles para el programa.

3.5.13 omp_set_max_active_levels

```
void omp_set_max_active_levels(int max_levels);
```

Limita el numero regiones paralelas anidadas, modifica la variable max-active-levels-var (VIC).

3.5.14 omp_get_max_active_levels

```
int omp_get_max_active_levels(void);
```

Devuelve el valor de la max-active-levels-var (VIC), que determina el número máximo de regiones paralelas anidadas.

3.5.15 omp_get_level

```
int omp_get_level(void);
```

Devuelve el número de regiones paralelas anidadas que contiene la llamada.

3.5.16 omp_get_ancestor_thread_num

```
int omp_get_ancestor_thread_num(int level);
```

Devuelve, para un determinado nivel anidado de la hebra actual, el numero de hebra padre o al que actualmente pertenece.

3.5.17 omp_get_team_size

```
int omp_get_team_size(int level);
```

Devuelve, para un determinado nivel anidado de la hebra actual, el tamaño del equipo del hebra padre o al que actualmente pertenece.

3.5.18 omp_get_active_level

```
int omp_get_active_level(void);
```

Devuelve, para un determinado nivel anidado de la hebra actual, el numero de hebra del padre.

3.6 Cerrojos

3.6.1 omp_init_[nest]_lock

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

Esta rutina inicializa los cerrojos de OpenMP

3.6.2 omp_destroy_[nest]_lock

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

Estas rutinas de asegurar que el bloqueo de OpenMP no está inicializado.

3.6.3 omp_set_[nest]_lock

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

Estas rutinas proporcionan un medio de establecer un bloqueo de OpenMP.

3.6.4 omp_unset_[nest]_lock

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

Estas rutinas proporcionan un medio de establecer un bloqueo de OpenMP.

3.6.5 omp_test_[nest]_lock

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

Estas rutinas comprueba si el cerrojo esta bloqueado, no suspendiendo la ejecución de la tarea.

3.7 Rutinas de tiempo

3.7.1 omp_get_wtime

```
double omp_get_wtime(void);
```

Devuelve el valor del reloj en segundos.

3.7.2 omp_get_wtick

```
double omp_get_wtick(void);
```

Devuelve la precisión del reloj.

3.8 Rutinas de tiempo

3.8.1 OMP_SCHEDULE type[,chunk]

Establece run-sched-var (VIC) al tipo de planificador y tamaño de las partes. Los tipos validos para el planificador OpenMP son static, dynamic, guied, o auto. Las partes es un entero positivo.

3.8.2 OMP_NUM$_THREADSnum$

Establece nthreads-var (VIC) el numero de hebras que usa una región paralela

3.8.3 OMP_DYNAMIC dynamic

Establece dyn-var (VIC) el ajuste dinámico de las hebras que usa la región paralela. Los valores validos para dynamic son true o false.

3.8.4 OMP_NESTED nested

Establece nest-var (VIC) a habilitado o deshabilitado el anidamiento paralelo. Los valores validos para nested es true or false.

3.8.5 OMP_STACKSIZE size

Establece stacksize-var (VIC) que especifica el tamaño de la pila de hebras que crea la implementación de OpenMP. Los valores para size (entero positivo) son size, sizeB, sizeK, sizeM, sizeG. Si la unidad B, K, M o G no se especifica, el tamaño se mide en kilobytes (K).

3.8.6 OMP_WAIT_POLICY policy

Establece wait-policy-var (VIC) que controla el comportamiento esperado de la espera de las hebras. Los valores validos para policy es activo (las hebras consumen ciclos de procesamiento mientras esperan) y pasivo.

3.8.7 OMP_MAX_ACTIVE_LEVELS levels

Establece max-active-levels-var (VIC) que controla el máximo numero de anidamientos activos en una región paralela.

3.8.8 OMP_THREAD_LIMIT limit

Establece thread-limit-var (VIC) que controla el máximo numero de hebras que participan en un programa OpenMP.

References

https://lsi2.ugr.es/jmantas/ppr/ayuda/omp$_ayuda.php?ayuda = omp_variableshttps : //www.openmp.org/wp-content/uploads/OpenMP - 4.5 - 1115 - CPP - web.pdf$