Práctica 3:

OpenMP

Parte 1:

Memoria individual

Nikita Polyanskiy

Y4441167L

Índice

1. ¿Qué es el OpenMP?

Es una abreviación de Open Multi-Processing, que es una interfaz de programación de aplicaciones (API), que puede ser utilizada para dirigir explícitamente el paralelismo de multihilo y memoria compartida.

Los objetivos de OpenMP son:

- Proporcionar un estándar entre una variedad de arquitecturas/plataformas de memoria compartida.
- Establecer un conjunto simple y limitado de directivas para programar máquinas de memoria compartida.
- Ser fácil de usar y eficaz: Se puede implementar un paralelismo significativo usando solo 3 o 4 directivas.
- Está especificada solo para C / C++ y Fortran.

2. Directivas de OpenMP

2.1 Formato de directivas en C / C++:

#pragma omp [nombre-directiva] [clausula opcional] [\n]

Ejemplo:

#pragma omp parallel default(shared) private(beta,pi)

Reglas generales:

Distingue mayusculas y minusculas.

Las directivas siguen las convenciones de los estándares C / C++ para las directivas del compilador.

Solo se puede especificar un nombre para una directiva.

Cada directiva se aplica como máximo a una declaración siguiente, que debe ser un bloque estructurado.

Las líneas directivas largas se pueden continuar en líneas sucesivas al utilizar el carácter \ al final de la línea.

2.2 Directiva PARALLEL

Se especifica un bloque de código que será ejecutado por varios subprocesos. Esta es la construcción paralela fundamental de OpenMP.

Formato:

```
#pragma omp parallel [clause ...]  newline
                    if (scalar_expression)
                    private (list)
                    shared (list)
                    default (shared | none)
                    firstprivate (list)
                    reduction (operator: list)
                    copyin (list)
                    num_threads (integer-expression)
  structured_block
```

Cuando un hilo alcanza una directiva PARALLEL, se crea un equipo de hilos y se convierte en el maestro del equipo. El maestro es miembro del equipo y tiene el hilo nº 0.

Comenzando desde el principio del bloque, el código se duplica y todos los subprocesos ejecutan ese código.

Los subprocesos ejecutaran el código hasta el final de la sección paralela, Solo el hilo maestro continua la ejecución después de ese punto.

Si algún subproceso termina en una región paralela, todos los subprocesos terminaran y el trabajo quedara sin definir.

El numero de subprocesos esta determinado por los siguientes factores:

1.Evaluación de la cláusula IF

2.Configuración de la cláusula NUM_THREADS

3.Uso de la función de biblioteca omp_set_num_threads ()

4.Configuración de la variable de entorno OMP_NUM_THREADS

5.Implementación predeterminada: por lo general, el número de CPU en un nodo, aunque podría ser dinámico (consulte la siguiente viñeta).

Ejemplo:

```
#include <omp.h>
main ()  {
int nthreads, tid;
/* Bifurca un equipo de hilos con cada hilo teniendo una variable tid privada */
#pragma omp parallel private(tid)
  {
  /* Obtiene e imprime el id del hilo*/
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);
```

```
/* Solo el hilo maestro hace esto */
if (tid == 0)
  {
  nthreads = omp_get_num_threads();
  printf("Number of threads = %d\n", nthreads);
  }
} /* Todos los hilos se unen al hilo maestro y terminan */
```

2.3 Directiva DO

La directiva DO / for especifica que las iteraciones del ciclo que le siguen inmediatamente deben ser ejecutadas en paralelo por el equipo. Esto supone que ya se ha iniciado una región paralela; de lo contrario, se ejecuta en serie en un solo procesador.

Formato:

```
#pragma omp for [clause ...]  newline
        schedule (type [,chunk])
        ordered
        private (list)
        firstprivate (list)
        lastprivate (list)
        shared (list)
        reduction (operator: list)
        collapse (n)
        nowait
  for_loop
```

Clausulas:

SCHEDULE: describe cómo se dividen las iteraciones del bucle entre los subprocesos del equipo. El programa predeterminado depende de la implementación

STATIC: Iteraciones del bucle se dividen en piezas de tamaño de trozo y luego asignados estáticamente a las roscas. Si no se especifica un fragmento, las iteraciones se dividen uniformemente (si es posible) de forma contigua entre los subprocesos.

DYNAMIC: Iteraciones del bucle se dividen en piezas de tamaño de trozo , y dinámicamente programadas entre los hilos; cuando un hilo termina un fragmento, se le asigna otro dinámicamente. El tamaño de fragmento predeterminado es 1.

GUIDED: Las iteraciones se asignan dinámicamente a subprocesos en bloques a medida que los subprocesos las solicitan hasta que no quedan bloques por asignar. Similar a DYNAMIC excepto que el tamaño del bloque disminuye cada vez que se entrega una parcela de trabajo a un hilo. El tamaño del bloque inicial es proporcional a:

number_of_iterations / number_of_threads

Los bloques subsiguientes son proporcionales a

number_of_iterations_remaining / number_of_threads

El parámetro chunk define el tamaño mínimo del bloque. El tamaño de fragmento predeterminado es 1.

RUNTIME: La decisión de programación se aplaza hasta el tiempo de ejecución mediante la variable de entorno OMP_SCHEDULE. Es ilegal especificar un tamaño de fragmento para esta cláusula.

AUTO: La decisión de programación se delega al compilador y / o al sistema de ejecución.

NO WAIT / nowait: si se especifica, los subprocesos no se sincronizan al final del bucle paralelo.

ORDERED: Especifica que las iteraciones del bucle deben ejecutarse como lo harían en un programa en serie.

COLLAPSE: especifica cuántos bucles en un bucle anidado deben colapsarse en un gran espacio de iteración y dividirse de acuerdo con la cláusula de SCHEDULE. La ejecución secuencial de las iteraciones en todos los bucles asociados determina el orden de las iteraciones en el espacio de iteraciones colapsado.

Ejemplo:

```
#include <omp.h>
#define CHUNKSIZE 100
#define N     1000
main ()
{
int i, chunk;
float a[N], b[N], c[N];
/* Algunas inicializaciones*/
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;
chunk = CHUNKSIZE;
#pragma omp parallel shared(a,b,c,chunk) private(i)
  {
  #pragma omp for schedule(dynamic,chunk) nowait
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];

  } /* final de la sección paralela*/
}
```

2.4 Directiva SECTIONS

La directiva SECTIONS es una construcción de trabajo compartido no iterativo. Especifica que las secciones de código adjuntas se dividirán entre los subprocesos del equipo.

Las directivas SECTION independientes están anidadas dentro de una directiva SECTIONS. Cada SECCIÓN se ejecuta una vez por un hilo en el equipo. Diferentes secciones pueden ser ejecutadas por diferentes subprocesos. Es posible que un hilo ejecute más de una sección si es lo suficientemente rápido y la implementación lo permite.

Formato:

```
#pragma omp sections [clause ...]  newline
                private (list)
                firstprivate (list)
                lastprivate (list)
                reduction (operator: list)
                nowait
 {
 #pragma omp section   newline
   structured_block
 #pragma omp section   newline
   structured_block
 }
```

Ejemplo:

```
#include <omp.h>
#define N     1000
main ()
{
int i;
float a[N], b[N], c[N], d[N];
/* Algunas inicializaciones*/
for (i=0; i < N; i++) {
  a[i] = i * 1.5;
  b[i] = i + 22.35;
  }
#pragma omp parallel shared(a,b,c,d) private(i)
  {
  #pragma omp sections nowait
    {
    #pragma omp section
    for (i=0; i < N; i++)
      c[i] = a[i] + b[i];
    #pragma omp section
    for (i=0; i < N; i++)
      d[i] = a[i] * b[i];
    }  /* final de las secciones*/
  }  /* final de la sección paralela*/
}
```

2.5 Directiva SINGLE

La directiva SINGLE especifica que el código adjunto debe ser ejecutado por un solo subproceso en el equipo.

Puede ser útil cuando se trata de secciones de código que no son seguras para subprocesos (como E / S)

Formato:

```
#pragma omp single [clause ...]  newline
                private (list)
                firstprivate (list)
                nowait
    structured_block
```

Ejemplo:

```
#include <stdio.h>
#include <omp.h>
int main() {
  #pragma omp parallel num_threads(2)
  {
    #pragma omp single
    // Solo 1 hilo lee.
    printf_s("read input\n");

    // Multiples hilos computan el resultado
    printf_s("compute results\n");

    #pragma omp single
    // Solo 1 hilo escribe.
    printf_s("write output\n");
   }
}
```