OpenMP

Alvarado Morán Óscar
Bermúdez Marbán Dante
García Avendaño Martín

12 de noviembre del 2019

El que lea esto puede reclamar un premio al equipo expositor.

1. Introducción

Open MP es una librería para los lenguajes C, C++ Y Fortran Orientada al paralelismo sumamente popular por su compatibilidad con casi todas las arquitecturas de procesadores y sistemas operativas que nos ayuda a ejecutar paralelismo a partir de solo escribir ciertas directivas.

2. Open MP

Open MP es una API dirigida específicamente hacia el paralelismo que destaca en múltiples puntos como lo son:

Estandarización OpennMP es bastante estándar en cuanto a arquitecturas en las cuales se puede implementar es decir la mayoría de las arquitecturas actuales pueden utilizar OpenMP sin ningún problema esto debido al gran impulso y respaldo que tiene por marcas poderosas como intel.

Clara y Discreta La implementación de OpenMP es sumamente fácil debido a que esta es sumamente Clara debido a que solo existen un limitado nombre de instrucciones las cuales con solo invocarlas realizan paralelismo sin tener que definir parámetros de mas.

Portabilidad OpenMP se encuentra disponible para 3 de los lenguajes mas importantes en la actualidad como lo son C, C++ Y Fortran además de contar con un foro publico lo que aumenta la ayuda para principiantes y expertos en general, además se encuentra dispoble para windows, linux y Unix lo que lo hace estar disponible para casi todas las computadores.

2.1. Historia

OpenMP vio la luz en un principio como una biblioteca de apoyo a Fortran en 1887 pero con el tiempo se incorporo también para C actualizándose en un principio una versión para Fortran un año y el siguiente una para C asi hasta el año 2005 que se saca una actualización tanto de Fortran como de C al mismo tiempo.

2.2. Modelo Fork Join

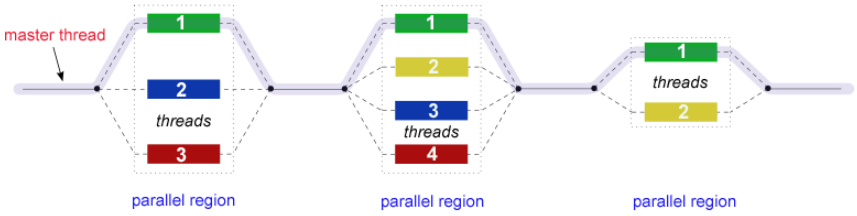

Figura 1: Diagrama del modelo fork-join

2.3. Descripción de los componentes de la API

- **Directivas del compilador(19)**
- **Rutinas de biblioteca en tiempo de ejecución(32)**

2.3.1. Directivas del Compilador

- **Son comentarios en el código fuente ignorados por el compilador**
- **Las directivas sirven para:** Endrear una región paralela es decir determinar una region paralela con una simple directiva, de igual forma nos sirven para dividir bloques de código entre hilos, distribuir bucles entre hijos, serializar secciones de código y sincronizar trabajos entre hilos

2.4. Rutinas de biblioteca en tiempo de ejecucion

- **Establecer y consultar el numero de hilos**
- **Consultar el identificador único de un hilo hijo, el padre del hilo**
- **Establecer y consultar la función de subprocesos dinámico**
- **Consultar si esta en una región paralela y de estarlo determinar en cual**

- **Establecer y consultar paralelismo anidado**
- **Establecer, inicializar y terminar bloques de hilos anidados**

2.5. Variables de Entorno

OpenMP nos proporciona variables de entorno para controlar la ejecución del código paralelo en tiempo de ejecución como ejemplo tenemos las siguientes:

- **Establecer el número de hilos**
- **Especificar cómo se dividen las interacciones de bucle**
- **Hilos de unión a procesadores**
- **Habilitar/Deshabilitar el paralelismo anidado**
- **Habilitar/Deshabilitar hilos dinámicos**
- **Establecer el tamaño de la pila de hilos**

2.6. Instalación

Como vimos anteriormente, OpenMP no es un programa, por lo que no se tiene que instalar. Sin embargo, sí se tiene que tener un compilador que pueda, valga la redundancia, compilar un archivo con instrucciones de OpenMP. Se recomienda el compilador gcc, que normalmente viene incluido con la instalación de Linux, pero en caso contrario es posible instalar ejecutando los siguientes comandos en la terminal:

```
1 $ sudo apt update
2 $ sudo apt install build-essential
```

3. Directivas

3.1. for

Esta directiva especifica que las iteraciones del loop que se encuentra a continuación de ésta serán ejecutadas en paralelo por el equipo (de hilos). El trabajo se divide entre los hilos que especifiquemos (y los que alcancen con el hardware con el que se esté trabajando) sin algún orden específico.

```
1 ...
2 #pragma omp parallel for
3     for(){
4     ...
```

```
5       }
6   ...
```

3.2. sections

Es una directiva que, a diferencia del for, éste no divide el trabajo, mas bien las secciones (que son independientes) son divididas entre los hilos.

```
1   ...
2   #pragma omp sections
3   {
4       #pragma omp section
5       {
6           ...
7       }
8       ...
9       #pragma omp section
10      {
11          ...
12      }
13  }
14  ...
```

- Cada sección se ejecuta una vez por un hilo.
- Es posible que un hilo ejecute más de una sección si es lo suficientemente rápido y la implementación lo permite
- Si hay mas hilos que secciones, habrá hilos que ejecuten una sección y habrá otros que no harán algo
- Si hay mas secciones que hilos, la implementación de OpenMP se encarga de como distribuir las secciones extras.
- Hay una barrera implícita al final de `SECTIONS` (ver 3.6).

3.3. single

Esta directiva especifica que el código será ejecutado por un solo hijo

```
1   ...
2   #pragma omp single
3   {
4       ...
5   }
```

Los hilos que no ejecutan la directiva, se esperan a que termine, es decir, hay una barrera implícita (ver 3.6) al menos que se le indique con `nowait`.

3.4. master

Esta directiva es similar a `single`, con la diferencia de que el código será ejecutado por el hilo 0 (el maestro).

```
1 ...
2 #pragma omp master
3 {
4     ...
5 }
6 ...
```

A diferencia de `single`, no existe una barrera implícita.

3.5. critical

Directiva para garantizar la exclusión mutua, es decir, que el código se ejecute por únicamente un hilo.

```
1 ...
2 #pragma omp critical [name]
3 {
4     // Region crítica
5 }
6 ...
```

Se pueden asignar nombres para crear diferentes secciones críticas, se puede decir que son identificadores. Los que no tengan nombre son tratados como la misma sección

3.6. barrier

Esta directiva sirve para poner una barrera virtual a los hilos que ya terminaron de hacer lo que se les asignó y así tengan que esperar a que lso demár hilos terminen.

```
1  ...
2  #pragma omp parallel
3  {
4      // Hilos haciendo cosas;
5      ...
6      #pragma omp barrier
7      ...
8      // Más cosas con hilos bien organizados;
9      ...
10 }
11 ...
```

3.7. atomic

Esta directiva asegura que una se acceda a una dirección específica de almacenamiento atómicamente, protegiendo a dicha dirección de los problemas típicos al tratar con paralelismo y concurrencia. En la literatura lo describen como una mini - sección crítica.

```
1 ...
2 #pragma omp parallel
3 {
4     ...
5     #pragma omp atomic
6     // Operación que se desee hacer atómicamente.
7     ...
8 }
9 ...
```

4. Cláusulas

4.1. shared

Cláusula que se utiliza para indicar que una variable es compartida. Esto quiere decir que existe una dirección en memoria donde todos los hilos pueden leer y escribir. No garantiza la inexistencia de condición de carrera.

```
shared(var1,var2,...)
```

En general, casi todas las variables son compartidas debido a que OpenMP está basado en el modelo de memoria compartida

4.2. private

Cláusula para indicar que la variable es privada, es decir, existirá una copia de cada variable para cada hilo.

Nota: se asume que la variable no está inicializada (para inicialización ver siguiente subsección)

```
private(var1,var2,...)
```

En general, toda variable que sea declarada dentro de una región paralela, será privada, es decir, cada hilo tendrá su propia variable y los demás hilos no tienen acceso a tal. Ejemplos comunes de esto son las variables de los ciclos *for* y todas las variables que se utilicen en alguna función. Cuando se termina la región paralela, estas variables desaparecen.

4.3. firstprivate

Cláusula que es similar a la de `private`, con la diferencia de que se hace una inicialización

```
firstprivate(var1,var2,...)
```

ejemplo firstprivate

```
1 ...
2 int x = 3;
3 #pragma omp parallel firstprivate(x)
4 {
5     ...
6 }
7 ...
```

En este ejemplo, a diferencia de `private`, cada variable privada x iniciará con el valor 3.

4.4. lastprivate

Cláusula que es similar a la de `private`, con la diferencia de que se obtiene una copia del último valor (por ejemplo, de la última iteración o de la última sección).

```
lastprivate(var1,var2,...)
```

Ejemplo lastprivate

```
1 ...
2 int x;
3 #pragma omp parallel for lastprivate(x)
4 for(...){
5     ...
6 }
7 ...
```

En el ejemplo, al terminar el *for*, la variable original x tendrá almacenado el último valor que se tuvo en la región paralela (antes de desaparecer).

4.5. reduction

Esta cláusula, como su nombre lo dice, sirve para reducir diferentes tipos de operaciones, ya sea suma, resta, multiplicación, etc. Básicamente se hace un divide y vencerás según tengo entendido.

```
1 ...
2 #pragma omp parallel for shared(<variables compartidas>)
      reduction (<operación>: <variables en la que se guarda
       la operación>)
3 ...
4 // Operación que se desee reducir, un loop con una suma
      tipo += por ejemplo.
5 ...
```

Referencias

[1] https://computing.llnl.gov/tutorials/openMP/

[2] https://docs.microsoft.com/en-us/cpp/parallel/openmp/reference/openmp-directives?view=vs-2019#atomic