2.3 Paradigmas de programación paralela: paralelismo de datos

Computación de Altas Prestaciones
Grado en Ingeniería de Computadores

Raúl Durán Díaz

Departamento de Automática
Universidad de Alcalá

Curso académico 2018–2019

Universidad de Alcalá

Revisión: 1.8

Contenidos

1 Estándar OpenMP

2 Directivas

Estándar OpenMP

- Para facilitar la programación en el modelo de hebras, se desarrolló el estándar OpenMP.
- Este estándar incluye:
  - directivas de compilación;
  - variables de entorno;
  - rutinas binarias complementarias.

Estándar OpenMP

- Existen directivas para distintos lenguajes, C, C++, y FORTRAN.
- El modelo sigue el paradigma SPMD (single program, multiple data).
- Las rutinas binarias de librería y las variables de entorno controlan la forma de ejecución.

Modelo de ejecución

- El modelo de ejecución se basa en la creación y destrucción de conjuntos de hebras.
- Las directivas de compilación definen el ámbito en que se crean y se destruyen las hebras.
- Las partes del código incluidas dentro de las directivas se denominan *regiones paralelas*.

Modo *fork-join*

- Al comienzo de la región paralela, se dispara un manojo de hebras, que ejecutan en principio el mismo código, salvo que el programador lo disponga de otro modo.
- Al final de la región, se ejecuta una barrera implícita y se produce la reunión de todas las hebras.
- El siguiente código ya se ejecuta en serie.

Modelo de memoria

- Por defecto, las variables activas en la región paralela se consideran compartidas entre todas las hebras.
- Se puede también definir un conjunto de variables privadas de cada hebra de modo que no se interfieren entre sí.

Contenidos

1 Estándar OpenMP

2 Directivas

Estructura típica de un programa

- Al principio de cada fichero fuente se debe escribir:

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

- La forma general de las directivas es:

```
# pragma omp directiva [órdenes [...] ]
```

- Para compilar, se debe añadir la opción `-fopenmp`. Por ejemplo,

```
gcc -fopenmp source.c -o executable
```

Región paralela

- Para crear una región paralela, la directiva básica es

```
# pragma omp parallel
{
...
}
```

- El código que está comprendido en el bloque se ejecutará por todas las hebras creadas.
- Al acabar el bloque, se realiza un *join* de todas las hebras y sigue la ejecución secuencial ordinaria.

Funciones auxiliares relacionadas

- Para ver cuántas hebras hay:

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

- Para ver cuál es la identificación de la hebra ejecutante:

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

- Para establecer el número de hebras:

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

- Para ver el número de procesadores:

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

Universidad de Alcalá

Órdenes de la directiva `parallel`

- Para declarar una lista de variables compartidas entre todas las hebras:

  `shared(lista_de_variables)`

- Para declarar una lista de variables privadas para cada hebra:

  `private(lista_de_variables)`

Ejemplo de región paralela

Ejemplo

```
int npoints, iam, np, mypoints;
double *x;
...
# pragma omp parallel shared(x,npoints) private(iam,np,mypoints)
{
...
}
```

Distribución de un lazo

- Dentro de una región paralela podemos distribuir el trabajo de uno o más lazos de tipo `for` entre las hebras activadas en la región paralela.

```
# pragma omp for [órdenes [...] ]
for (i = cota_inferior; i op cota_superior; incremento)
{
...
}
```

Condiciones de la directiva `for`

- El lazo ha de estar en una región paralela.
- Cada vuelta del lazo ha de ser independiente.
- La variable del índice no ha de cambiar dentro del código del lazo y se considera privada de la hebra ejecutante.

Condiciones de la directiva `for`

- El operador `op` puede ser uno de estos:

  `{ <, <=, >, >= }`
- El `incremento` puede ser de alguna de las siguientes maneras:

  `++i, --i, i++, i--, i += incr, i -= incr,`
  `i = i + incr, i = incr + i, i = i - incr, i = incr - i`
- Al finalizar el lazo, se realiza una sincronización implícita.

Comentario

Si se añade la orden `nowait`, se evita esa sincronización.

Estrategias de distribución de tareas

- Para distribuir entre las hebras las distintas vueltas del lazo, se pueden usar diversas estrategias:
  - `schedule(static, block_size)`
  - `schedule(dynamic, block_size)`
  - `schedule(guided, block_size)`
  - `schedule(auto)`
  - `schedule(runtime)`

Comentario

Si no se indica nada, el sistema toma un valor por defecto que depende de la implementación.

Ejemplo de estrategias de distribución

Ejemplo

```
# pragma omp parallel shared(A, B, C) private(M, N, L)
{
...
# pragma omp for schedule(auto)
   for (...) {
   }
...
# pragma omp for schedule(static)
   for (...) {
   }
...
}
```

Distribución de tareas arbitrarias

- Para distribuir la carga de trabajo se puede emplear la directiva

```
# pragma omp sections
{
 # pragma omp section
 { ... }
 # pragma omp section
 { ... }
...
}
```

- Cada `section` denota un bloque de código independiente de los demás bloques. Todos ellos se pueden ejecutar en paralelo.
- Al final del grupo de secciones se realiza una sincronización implícita, salvo que se indique lo contrario mediante la orden `nowait`.

Universidad de Alcalá

Funciones de tiempo de ejecución

- Controlan el entorno de ejecución.
- Para establecer o averiguar el control dinámico del número de hebras:

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

- Para establecer el número (exacto o máximo) del número de hebras:

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

Esta última actúa en dos formas:

  - si el control dinámico está activado, el número de hebras es el máximo que el sistema puede usar.
  - si no lo está, es el número exacto de hebras que el sistema puede usar.

Regiones críticas

- Para sincronizar el acceso a datos compartidos, se pueden establecer regiones críticas, ejecutables por solo una hebra en cada momento.
- Usamos la siguiente directiva:

```
# pragma omp critical [nombre]
{ ... }
```

- Todas las regiones con el mismo nombre forman una sola sección crítica.

Barreras

- Una barrera sincroniza la ejecución de un grupo de hebras. Cuando todas las hebras alcanzan la barrera, se continúa la ejecución.
- La directiva es:

```
# pragma omp barrier
```

Operaciones atómicas

- Se puede realizar una asignación de modo atómico (es decir, con ausencia de interrupciones) en una variable.
- La sintaxis es la siguiente:

```
# pragma omp atomic
  sentencia
```

- La `sentencia` puede ser

```
x op= E;
x++, x--, ++x, --x
```

Esta directiva no asegura el acceso exclusivo sino que la operación se realiza de modo atómico, es decir, sin interrupciones.

Reducción

- Una construcción típica es la reducción, en donde cada hebra realiza un cómputo sobre una variable privada y, al final de la región paralela, se realiza una combinación de todas las variables privadas sobre una variable compartida mediante una operación asociativa.
- La sintaxis es la siguiente:

```
# pragma omp reduction (op: lista)
```

- La op puede ser

```
{ +, -, *, &, ^, |, &&, || }
```

- La `lista` debe ser una lista de variables compartidas, sobre las que se realizará la reducción al finalizar el bloque. No se requiere ninguna operación adicional de sincronización.

Universidad de Alcalá

Reducción

Ejemplo

```
# pragma omp parallel for reduction (+: a,y)
for (i = 0; i < n; i++)
{
   a += b[i];
   y = sum(y, c[i]);
}
```

Al finalizar el lazo, los valores acumulados en las copias privadas de las variables a e y serán reducidos siguiendo la operación establecida, en este caso, una suma.

Actualización de variables en memoria compartida

- Para que todas las hebras tengan la misma visión de todas las variables, se puede forzar la actualización.
- La sintaxis es la siguiente:

```
# pragma omp flush (lista_de_variables)
```