Tema 4.4 Programación mediante directivas OpenMP: Sincronización

Computación de Altas Prestaciones

Carlos García Sánchez

10 de octubre de 2022

- 'Using OpenMP : portable shared memory parallel programming", Barbara Chapman, et all. 2008
- "OpenMP 5.2", https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf

Outline

¿Qué es?

Mecanismos

- Los hilos deben sincronizarse para imponer un cierto orden
- OpenMP proporciona diferentes mecanismos de sincronización:
  - *barrier*
  - *critical*
  - *atomic*
  - *taskwait*
  - *ordered*
  - *locks*

Barrera *barrier*

`#pragma omp barrier`

- Hilos no pueden pasar de la barrera hasta que todos los hilos alcancen hayan llegado
- Todo el trabajo generado previamente se completa (sincronización)
- Algunas construcciones conllevan una barrera implícita al final de la misma
  - Ej: La construcción `#pragma omp parallel`

Barrera

example_barrier.c

```
...
#pragma omp parallel
{
    foo();
    #pragma omp barrier
    bar();
}
```

Fuerza a que todos los hilos acaben **foo** antes de comenzar con **bar**

Barrera

example_barrier.c

```
...
#pragma omp parallel
{
    foo();
    #pragma omp barrier
    bar();
}
```

Barrera implícita por la región **parallel**

Construcción `critical`

```
#pragma omp critical [(name)]
```

- Proporciona una región de exclusión mutua donde solo un hilo puede estar al mismo tiempo
- Por defecto todas las regiones críticas son las mismas salvo que tengan un nombre específico
  - Solo se sincronizan los hilos de una región con el mismo nombre

Construcción critical

example_critical.c

```
...
int x=1;
#pragma omp parallel num_threads(2)
{
   #pragma omp critical
   x++;
}
printf("%d\n", x);
```

Un solo hilo en la sección crítica

Imprimirá 3!!

Acceso exclusivo *critical*

example_critical.c

```
int x=1, y=0;
#pragma omp parallel num_threads(4)
{
   // One thread can update x while another updates y
   #pragma omp critical (x)
   x++;
   #pragma omp critical (y)
   y++;
}
printf("%d\n", x);
```

Construcción `atomic`

`#pragma omp atomic`

- Mecanismo de exclusión mutua para hacer operaciones **read&update**
  - Ej: `x+=1, x=x-foo()`
    - *foo* no está protegido
- Normalmente más eficiente que la consstrucción **critical**
- Ojo: no es compatible con la construcción **critical**

Construcción atomic

example_atomic.c

```
...
int x=1;
#pragma omp parallel num_threads(2)
{
   #pragma omp atomic
   x++;
}
printf("%d\n", x);
```

Un solo hilo en la sección crítica

Imprimirá 3!!

Construcción atomic

example_atomic.c

```
int x=1;
#pragma omp parallel num_threads(4)
{
   #pragma omp critical
   x++;
   #pragma omp atomic
   x++;
}
printf("%d\n", x);
```

Un solo hilo en la sección crítica

... **pero otro hilo puede estar en sección atómica!!!**

Construcción `master`

`#pragma omp master`

- El bloque es ejecutado por el hilo **maestro**
  - Útil cuando se quiere ejecutar una región secuencialmente
  - No tiene barrera implicita al final

Cerrojos

- OpenMP proporciona cerrojos a bajo nivel

Locks

- *omp_init_lock*: inicialización
- *omp_set_lock*: adquisición del cerrojo
- *omp_unset_lock*: devolución del cerrojo
- *omp_test_lock*: chequeo de si el cerrojo está libre (no bloqueante)
- *omp_destroy_lock*: libera recursos de cerrojo

Cerrojos

example_locks.c

```
#include<omp.h>
void foo()
{
   omp_lock_t lock;
   omp_init_lock(&lock);
   #pragma omp parallel
   {
      omp_set_lock(&lock);
      //mutual exclusion region
      omp_unset_lock(&lock);
   }
   omp_destroy_lock(&lock);
}
```