Computación Paralela

Alumna:
Esther Milagros Bautista Peralata

Semestre: *VIII - semestre*

Docente:	*Fred Torres Cruz*
Facultad:	Ingeniería Estadística e Informática
Año académico:	November 23, 2022- II

Contents

1 Trabajo Encargado

Analice la diferencia de la Sección Crítica y bloqueo en Open MP y haga una comparación.

2 Sección crítica y mutua exclusión

El método más sencillo de comunicación entre los procesos de un programa concurrente es el uso común de unas variables de datos. Esta forma tan sencilla de comunicación puede llevar, no obstante, a errores en el programa ya que el acceso concurrente puede hacer que la acción de un proceso interfiera en las acciones de otro de una forma no adecuada.

Para evitar este tipo de errores se pueden identificar aquellas regiones de los procesos que acceden a variables compartidas y dotarlas de la posibilidad de ejecución como si fueran una única instrucción. Se denomina Sección Crítica a aquellas partes de los procesos concurrentes que no pueden ejecutarse de forma concurrente o, también, que desde otro proceso se ven como si fueran una única instrucción. Esto quiere decir que si un proceso entra a ejecutar una sección crítica en la que se accede a unas variables compartidas, entonces otro proceso no puede entrar a ejecutar una región crítica en la que acceda a variables compartidas con el anterior.

Las secciones críticas se pueden mutuo excluir. Para conseguir dicha exclusión se deben implementar protocolos software que impidan o el acceso a una sección crítica mientras está siendo utilizada por un proceso.

La directiva crítica omp identifica una sección de código que debe ser ejecutada por un solo subproceso a la vez.

```
#pragma omp critical [(nombre)]
{ structured block }
```

El bloque lo ejecuta en exclusión mutua respecto a todas las secciones críticas con el mismo nombre global.

(Opcional) Nombre para identificar el código crítico. El nombre debe ir entre paréntesis.

Los nombres actúan como identificadores globales. Todas las secciones críticas que no tienen nombre son tratadas como la misma.

Usos: Un subproceso espera al comienzo de una región crítica identificada por un nombre dado hasta que ningún otro subproceso en el programa esté ejecutando una región crítica con ese mismo nombre. Las secciones críticas no nombradas específicamente por la invocación de la directiva crítica omp se asignan al mismo nombre no especificado.

Ventajas:

- Secuencializan código (depuración)

- Acceso seguro a memoria compartida.

Desventajas:

- Disminuyen eficiencia al reducir paralelismo.

3 Funciones de bloqueo en Open MP

Para las siguientes funciones, la variable de bloqueo debe tener el tipo omp_lock_t. Solo se debe tener acceso a esta variable a través de estas funciones. Todas las funciones de bloqueo requieren un argumento que tiene un puntero al tipo omp_lock_t.

```
* La función omp_init_lock inicializa un bloqueo simple.
* La función omp_destroy_lock quita un bloqueo simple.
* La función omp_set_lock espera hasta que haya disponible
  un bloqueo simple.
* La función omp_unset_lock libera un bloqueo simple.
* La función omp_test_lock prueba un bloqueo simple.
```

Para las siguientes funciones, la variable de bloqueo debe tener el tipo $omp\_nest\_lock\_t$. Solo se debe tener acceso a esta variable a través de estas funciones. Todas las funciones de bloqueo anidado requieren un argumento que tiene un puntero al tipo $omp\_nest\_lock\_t$.

```
* La función omp_init_nest_lock inicializa un bloqueo anidado.
* La función omp_destroy_nest_lock quita un bloqueo anidado.
* La función omp_set_nest_lock espera hasta que haya disponible
  un bloqueo anidado.
* La función omp_unset_nest_lock libera un bloqueo anidado.
* La función omp_test_nest_lock prueba un bloqueo anidado.
```

Las funciones de bloqueo openMP acceden a la variable de bloqueo de tal forma que siempre leen y actualizan el valor más actual de la variable de bloqueo. Por lo tanto, no es necesario que un programa OpenMP incluya directivas flush explícitas para asegurarse de que el valor de la variable de bloqueo sea coherente entre diferentes subprocesos. (Puede que sea necesario que las directivas flush hagan coherentes los valores de otras variables).

3.1 omp_init_lock y $omp\_init\_nest\_lock$ (funciones)

Estas funciones proporcionan los únicos medios para inicializar un bloqueo. Cada función inicializa el bloqueo asociado al parámetro bloqueo para su uso en próximas llamadas. El formato es como sigue:

```
#include <omp.h>
void omp_init_lock(omp_lock_t *lock);
void omp_init_nest_lock(omp_nest_lock_t *lock);
```

El estado inicial se desbloquea (es decir, ningún subproceso posee el bloqueo). Para un bloqueo anidado, el recuento de anidamiento inicial es cero. No es compatible llamar a cualquiera de estas rutinas con una variable de bloqueo que ya se ha inicializado.

3.2 *omp_destroy_lock* y *omp_destroy_nest_lock* (funciones)

Estas funciones se aseguran de que el bloqueo de variable de bloqueo apuntado no está inicializado. El formato es como sigue:

```
#include <omp.h>
void omp_destroy_lock(omp_lock_t *lock);
void omp_destroy_nest_lock(omp_nest_lock_t *lock);
```

No es compatible llamar a cualquiera de estas rutinas con una variable de bloqueo que no esté inicializada o que no esté bloqueada.

3.3 *omp_set_lock* y *omp_set_nest_lock* (funciones)

Cada una de estas funciones bloquea el subproceso que ejecuta la función hasta que el bloqueo especificado está disponible y, a continuación, establece el bloqueo. Un bloqueo simple está disponible si está desbloqueado. Un bloqueo anidado está disponible si está desbloqueado o si ya es propiedad del subproceso que ejecuta la función. El formato es como sigue:

```
#include <omp.h>
void omp_set_lock(omp_lock_t *lock);
void omp_set_nest_lock(omp_nest_lock_t *lock);
```

Para un bloqueo simple, el argumento de la función *omp_set_lock* debe apuntar a una variable de bloqueo inicializada. La propiedad del bloqueo se concede al subproceso que ejecuta la función.

Para un bloqueo anidado, el argumento de la función *omp_set_nest_lock* debe apuntar a una variable de bloqueo inicializada. El recuento de anidamiento se incrementa y se concede o mantiene la propiedad del bloqueo.

3.4 *omp_unset_lock* y *omp_unset_nest_lock* (funciones)

Estas funciones proporcionan los medios para liberar la propiedad de un bloqueo. El formato es como sigue:

```
#include <omp.h>
void omp_unset_lock(omp_lock_t *lock);
void omp_unset_nest_lock(omp_nest_lock_t *lock);
```

El argumento de cada una de estas funciones debe apuntar a una variable de bloqueo inicializada propiedad del subproceso que ejecuta la función. Si el subproceso no posee dicho bloqueo, el comportamiento no está definido.

Para un bloqueo simple, la función *omp_unset_lock* libera el subproceso que ejecuta la función de la propiedad del bloqueo.

Para un bloqueo anidado, la función *omp_unset_nest_lock* disminuye el recuento de anidamiento y libera el subproceso que ejecuta la función de la propiedad del bloqueo si el recuento resultante es cero.

3.5 *omp_test_lock* y *omp_test_nest_lock* (funciones)

Estas funciones intentan establecer un bloqueo, pero no bloquean la ejecución del subproceso. El formato es como sigue:

```
#include <omp.h>
int omp_test_lock(omp_lock_t *lock);
int omp_test_nest_lock(omp_nest_lock_t *lock);
```

El argumento debe apuntar a una variable de bloqueo inicializada. Estas funciones intentan establecer un bloqueo de la misma manera que *omp_set_lock* y *omp_set_nest_lock*, salvo que no bloquean la ejecución del subproceso.

Para un bloqueo simple, la función *omp_test_lock* devuelve un valor distinto de cero si el bloqueo se establece correctamente; de lo contrario, devuelve cero.

Para un bloqueo anidado, la función *omp_test_nest_lock* devuelve el nuevo recuento de anidamiento si el bloqueo se ha establecido correctamente; de lo contrario,S devuelve cero.

4 Analizamos casos

4.1 Usar una variable de bloqueo sin inicializar la variable

De acuerdo con la especificación OpenMP 2.0, todas las variables de bloqueo deben inicializarse a través de la llamada de función *omp_init_lock* u *omp_init_nest_lock* (según el tipo de variable). Una variable de bloqueo solo se puede usar después de la inicialización. Un intento de usar (establecer, desarmar, probar) en una variable de bloqueo no inicializada en un programa C++ provocará un error en tiempo de ejecución.

CORRECTO:

```
omp_lock_t myLock;
omp_init_lock(&myLock);
#pragma omp parallel num_threads(2)
{
    ...
    omp_set_lock(&myLock);
    ...
}
```

4.2 Desactivar un bloqueo de otro hilo

Si se establece un bloqueo en un subproceso, un intento de desactivar este bloqueo en otro subproceso provocará un comportamiento impredecible. Consideremos el siguiente ejemplo:

INCORRECTO:

```
omp_lock_t myLock;
omp_init_lock(&myLock);
#pragma omp parallel sections
{
    #pragma omp section
    {
          ...
          omp_set_lock(&myLock);
          ...
    }
    #pragma omp section
    {
          ...
          omp_unset_lock(&myLock);
          ...
    }
}
```

Este código provocará un error en tiempo de ejecución en un programa C++. Dado que las operaciones de activación y desactivación de bloqueos son similares a la entrada y salida de una sección crítica , todos los subprocesos que utilizan bloqueos deben realizar ambas operaciones. Aquí hay una versión correcta del código:

CORRECTO:

```
omp_lock_t myLock;
omp_init_lock(&myLock);
#pragma omp parallel sections
```

```
{
    #pragma omp section
    {
          ...
          omp_set_lock(&myLock);
          ...
          omp_unset_lock(&myLock);
          ...
    }
    #pragma omp section
    {
          ...
          omp_set_lock(&myLock);
          ...
          omp_unset_lock(&myLock);
          ...
    }
}
```

4.3 Usar un candado como barrera

La función omp_set_lock bloquea la ejecución de un subproceso hasta que la variable de bloqueo esté disponible, es decir, hasta que el mismo subproceso llame a la función omp_unset_lock. Por tanto, como ya se ha comentado en la descripción del error anterior, cada uno de los hilos debería llamar a ambas funciones. Un desarrollador con una comprensión insuficiente de OpenMP puede intentar usar la función omp_set_lock como barrera, es decir, en lugar de la directiva de barrera #pragma omp (ya que la directiva no se puede usar dentro de una sección paralela, a la que se aplica la directiva de secciones #pragma omp) . Como resultado se creará el siguiente código:

INCORRECTO:

```
omp_lock_t myLock;
omp_init_lock(&myLock);
#pragma omp parallel sections
{
    #pragma omp section
    {
          ...
          omp_set_lock(&myLock);
          ...
    }
    #pragma omp section
    {
          ...
          omp_set_lock(&myLock);
```

```
            omp_unset_lock(&myLock);
            ...
    }
}
```

A veces, el programa se ejecutará con éxito. A veces no lo hará. Esto depende del hilo que termine su ejecución primero. Si el subproceso que bloquea la variable de bloqueo sin liberarla finaliza primero, el programa funcionará como se esperaba. En todos los demás casos, el programa esperará infinitamente a que el subproceso, que funciona incorrectamente con la variable de bloqueo, desactive la variable. Ocurrirá un problema similar si el desarrollador coloca la llamada a la función omp_test_lock dentro de un bucle (y esa es la forma en que generalmente se usa la función). En este caso, el bucle hará que el programa se cuelgue, porque el bloqueo nunca se desactivará.

Dado que este error es similar al anterior, la versión corregida del código seguirá siendo la misma:

CORRECTO:

```
omp_lock_t myLock;
omp_init_lock(&myLock);
#pragma omp parallel sections
{
    #pragma omp section
    {
            ...
            omp_set_lock(&myLock);
            ...
            omp_unset_lock(&myLock);
            ...
    }
    #pragma omp section
    {
            ...
            omp_set_lock(&myLock);
            ...
            omp_unset_lock(&myLock);
            ...
    }
}
```

5 Evidencia del trabajo en LATEX

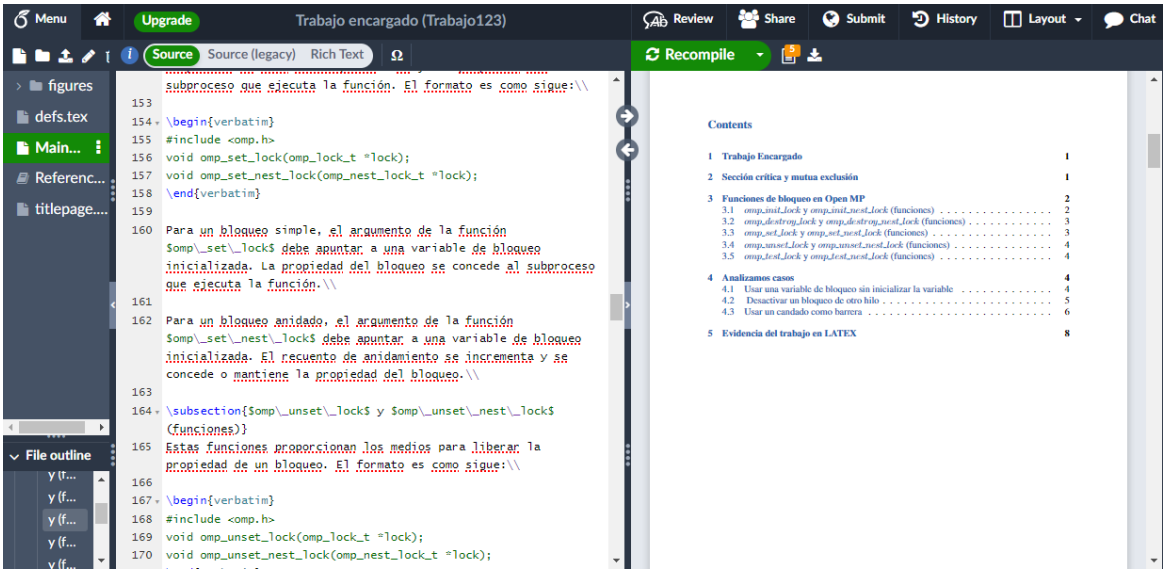

Figure 1: https://www.overleaf.com/read/grwrnnctnrsx