Introducción: ¿Qué es OpenMP?

OpenMp es una API (Application Program Interface) que puede ser utilizada para implementar de manera explícita paralelismo multihilo con memoria compartida en programas en Fortran y C/C++. OpenMP es portable, habiendo sido implementada en múltiples plataformas, entre las que destacan la mayoría de las plataformas de Unix y Windows NT, y estandarizada (aunque no tiene por qué estar implementada exactamente igual por todos los distribuidores)

La API posee principalmente tres tipos de componentes:

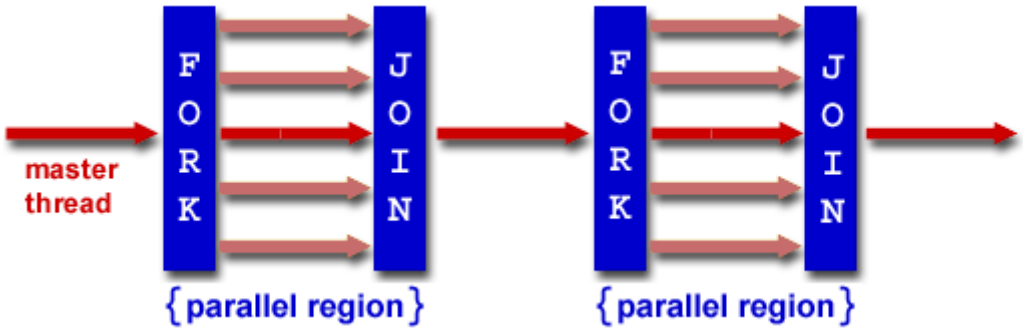
- Directivas de compilador
- Rutinas en tiempo de ejecución
- Variables de entorno

Con cuyo uso se pueden implementar programas o secciones de programas paralelos.

Es necesario resaltar que OpenMP está diseñada específicamente para gestionar sistemas paralelos con memoria compartida, y no con memoria distribuida, aunque se pueden simular las características de estos últimos sobre un sistema de memoria compartida. Además, OpenMP no garantiza el uso óptimo en cuanto a eficiencia de la memoria compartida, dado que no existen estructuras específicas para el control de la localidad de los datos en la memoria. En lugar de eso, OpenMP persigue la simplicidad y facilidad de uso, estableciendo un conjunto limitado de directivas de paralelismo, de manera que utilizando tan sólo unas cuantas de ellas se puede obtener un grado significativo de paralelismo tanto de grano grueso como fino.

El modelo de programación de OpenMP

OpenMP es un modelo de programación explícita en paralelo; esto quiere decir que el programador tiene un control completo sobre el proceso de paralelización de su programa, en lugar de dejar parte de esta carga de trabajo a la propia API. Su paradigma de paralelismo en memoria compartida está basado en la existencia de múltiples hilos, empleándose en concreto la estructura Fork – Join para el tratamiento de los mismos.



Todos los programas en OpenMP empiezan con un único hilo: el hilo maestro (master thread). El hilo maestro se ejecuta secuencialmente hasta que se encuentra una estructura de región paralela (parallel region), cuyo inicio está representado en la figura por la palabra FORK. En el fork, el hilo maestro crea un equipo de hilos paralelos, que junto con él ejecutan las sentencias incluidas en la región paralela (OpenMP permite determinar dinámicamente el número de hilos que componen un equipo, si bien algunas implementaciones no contemplan esta función). Cuando terminan la ejecución de estas sentencias, se sincronizan y se destruyen, quedando sólo el hilo maestro; en la figura, el fin de la región paralela se simboliza con la palabra JOIN.

En algunas implementaciones de OpenMP es posible situar regiones paralelas anidadas; en concreto, Averroes, la máquina sobre la que se programa en OpenMP en la Universidad de Córdoba, no incluye esta funcionalidad.

Como indicamos anteriormente, la mayor parte del paralelismo hasta ahora descrito se efectúa mediante una serie de directivas de compilador embebidas en el código fuente del programa, escrito bien en Fortran o bien en C/C++. En este documento nos centraremos en las directivas empleadas al combinar OpenMP y C/C++; a continuación se incluye como ejemplo una estructura base de código de OpenMP embebido en C/C++:

```
#include <omp.h>

main ()  {

int var1, var2, var3;

Código Secuencial
      .
      .
      .

Inicio de la sección paralela. Generación de un equipo de hilos.
Especificación del ámbito de las variables.

#pragma omp parallel private(var1, var2) shared(var3)
  {

  Sección paralela ejecutada por todos los hilos
```

```
        .
        .
        .
  Todos los hilos se sincronizan con el hilo maestro y se destruyen.
  }
Continúa el código secuencial
      .
      .
      .
}
```

La estructura es como vemos bien sencilla: se observa al principio que se incluye el archivo `omp.h`, necesario para emplear las funciones de OpenMP. Después se declaran tres variables y se ejecuta un código secuencial. Entramos entonces en la región paralela, señalada en rojo en el código. La directiva `#pragma omp parallel`, que veremos posteriormente, marca el inicio de la región paralela. Las cláusulas que acompañan a esta directiva (que también serán vistas en puntos posteriores) indican que dos de las variables serán privadas a cada hilo, y que la restante será pública para todos ellos. A continuación de la directiva, y entre llaves, se encuentra el código paralelo, que será ejecutado por el equipo de hilos creados por el maestro. Al acabar la región paralela, el programa continúa secuencialmente con normalidad.

Directivas de OpenMP

Estructura general

Ya hemos explicado que la gran mayoría del paralelismo en OpenMP se obtiene empleando una serie de directivas de compilador que nos brinda esta librería. Estas directivas siguen una sencilla estructura, expuesta a continuación:

`#pragma omp` *`nombre [clásulas] fin de línea`*

Todas las directivas comienzan con `#pragma omp`. Posteriormente se encuentra el nombre de la directiva, y tras él, opcionalmente, una o varias cláusulas especificando el modo de funcionamiento de la directiva; estas cláusulas pueden estar, salvo excepciones, en cualquier orden o incluso repetidas. Las directivas finalizan obligatoriamente con un retorno de línea (si la directiva es demasiado larga, se puede situar el carácter '\' al final de la línea y continuar en la siguiente), y afectan por lo general al bloque de código situado inmediatamente bajo ellas. Las directivas son sensibles a mayúsculas y minúsculas, y sólo pueden contener un nombre por directiva.

NOTA SOBRE LAS DIRECTIVAS: Una restricción al uso de las directivas de OpenMP, presente en todas aquellas que afectan a un bloque de código, es el hecho de que no está permitido saltar desde esos bloques estructurados hacia cualquier otro punto de fuera del bloque (utilizando `goto` o similar); de la misma manera, tampoco se puede saltar desde fuera hacia dentro de un bloque estructurado perteneciente a una directiva.

La directiva parallel

La directiva `parallel` es la más importante dentro de todas las estructuras que ofrece OpenMP, dado que define una región paralela, esto es, un segmento de código que será ejecutado por múltiples hilos. La estructura de la directiva es

```
#pragma omp parallel [cláusulas] fin de línea
      Bloque estructurado
```

EL funcionamiento es, tal como se ha explicado anteriormente, muy sencillo: cuando un hilo encuentra una directiva `parallel`, crea un equipo de hilos (numerados en orden ascendente, siendo 0 el hilo maestro) y se convierte en el hilo maestro del equipo. La sección paralela de código se duplica de manera que cada hilo ejecuta ese código, hasta llegar al fin de la región, en la que hay una sincronización implícita de los mismos después de la cual sólo el hilo maestro continúa ejecutando código.

Pero, ¿cuántos hilos forman un equipo? Pues esto viene determinado por tres factores, expuestos en orden de prioridad:

- El uso de la función de OpenMP `omp_set_num_threads()`, que permite especificar dinámicamente el número de hilos en un equipo (ver más abajo).
- El valor establecido en la variable de entorno OMP_NUM_THREADS.
- El valor por defecto que cada implementación asigna como número de hilos por equipo.

Cláusulas

Existen varias cláusulas que pueden ser utilizadas para especificar el modo de funcionamiento de la directiva `parallel`. Éstas son:

- **`if(`*`expresión escalar`*`)`**: Si la expresión escalar se cumple (no es igual a cero), se creará un equipo de trabajo que ejecutará la región paralela. Si no se cumpe, el equipo no se creará y sólo el hilo maestro ejecutará dicha región, como si fuese secuencial.

- **`private(`*`lista de variables`*`)`**: Especifica que las variables de la lista serán privadas a cada hilo. Esto es, cada hilo manejará su propia copia de la variable, pudiendo modificar su valor con independencia de los demás hilos. La copia de la variable que cada hilo maneja no está inicializada.
- **`shared(`*`lista de variables`*`)`**: Indica que las variables de la lista serán compartidas por todos los hilos; cada uno de ellos puede acceder a su valor y modificarlo. Es responsabilidad del programador hacer que los accesos a esta variable sean realizados adecuadamente.
- **`firstprivate(`*`lista de variables`*`)`**: Actúa de la misma manera que la cláusula `private`, pero las copias de las variables son inicializadas al principio de la región paralela con el valor que en ese momento tiene la variable original.
- **`reduction(`*`operador`*`:`*`lista de variables`*`)`**: Considera cada variable de la lista como privada (ver la cláusula `private`), y al final de la región paralela se aplica el operador de reducción a cada copia de la variable, almacenándose el resultado del proceso en la variable global original. Por ejemplo, el código

  ```
  #pragma omp parallel reduction(+:i)
     {
     i=2;
     }
  ```

  daría para la variable `i` (suponiéndola declarada previamente) un valor final de 2N (siendo N el número de hilos del equipo), el resultado de aplicar el operador de suma a los valores de cada copia de la variable `i`.
- **`default(shared|none)`**: Permite especificar el ámbito por defecto de las variables que aparecen en la región paralela. El ámbito establecido para las variables que aparezcan en las cláusulas `private`, `shared`, `reduction`, `firstprivate` y `lastprivate` (esta última cláusula se verá más adelante) tiene prioridad sobre el establecido por `default`. Esta cláusula sólo puede aparecer una vez en cada directiva.

Ejemplo de directiva parallel

```
#include <omp.h>

main ()  {

int nthreads, tid;

/*Crea un equipo de hilos, cada uno con una copia privada de las
variables*/
#pragma omp parallel private(tid)
  {

  /*Obtiene e imprime el identificador de hilo*/
```

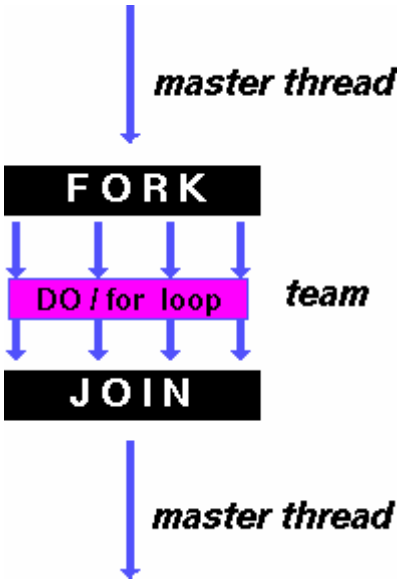
```
  tid = omp_get_thread_num();
  printf("Hola mundo desde el hilo = %d\n", tid);

  /*Sólo el hilo maestro ejecuta esto*/
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Número de hilos = %d\n", nthreads);
    }

  }  /*Todos los hilos se unen al maestro y finalizan*/

}
```

Estructuras de división de trabajo

Antes de proceder a describir las siguientes directivas de OpenMP es necesario hablar sobre un tipo especial de las mismas: las estructuras de división de trabajo. Éstas se encuentran dentro del bloque estructurado de una región paralela, y distribuyen la ejecución del código incluido en la estructura entre los miembros del equipo que la encuentran (no está permitido que sólo algunos miembros del equipo encuentren la región; deben ser todos, o ninguno). Existe una sincronización implícita de todos los miembros del equipo al final de la estructura de reparto de trabajo, aunque no al principio de ella.

Estructuras de división de trabajo: La directiva for

La primera de las estructuras que veremos será la directiva `for`, que distribuye las iteraciones de un bucle entre los miembros de un equipo. Representa un tipo de "paralelismo de datos".



Su formato es

```
#pragma omp for [cláusulas] fin de línea
   bucle for
```

La estructura `for` especifica que las iteraciones del bucle for que aparece inmediatamente a continuación deben ser ejecutadas en paralelo por el equipo. Suponemos que está incluída dentro de una región paralela; si esto no fuese así, el bucle for sería ejecutado secuencialmente por un único hilo.

Cláusulas

- **schedule(*tipo,[segmento]*)**: Describe la forma de división de las iteraciones del bucle entre los hilos que conforman el equipo. El tipo de reparto es establecida por el valor de la variable *tipo*, que puede tomar los siguientes:

  - STATIC: Las iteraciones son divididas en grupos de tamaño *segmento* que son posteriormente asignados estáticamente a los hilos del equipo. Si el valor de *segmento* no está especificado, las iteraciones se reparten alternativamente y por igual (siempre que sea posible) a cada miembro del equipo.
  - DYNAMIC: Las iteraciones son divididas en grupos de tamaño *segmento* y repartidas de manera dinámica entre los hilos. Cuando un hilo termina con un segmento, se le asigna inmediatamente otro. El tamaño de segmento por defecto es 1.
  - GUIDED: El tamaño de segmento es reducido exponencialmente conforme se vayan ejecutando las iteraciones del bucle. El valor de la variable *segmento* indica el tamaño mínimo que tendrán los segmentos, y su valor por defecto es 1.
  - RUNTIME: El tamaño de los segmentos se decide en tiempo de ejecución, en función del valor de la variable de entorno OMP_SCHEDULE, cuyo valor por defecto varía de una implementación a otra. No se puede dar un valor a la variable *segmento* al utilizar el tipo RUNTIME.

  En todos los casos, el valor de *segmento* debe ser una expresión estática, dado que no hay forma de sincronizar la evaluación de su valor entre los distintos hilos.

  Sólo se puede especificar una cláusula `schedule` por uso de la directiva.
- **nowait**: Si aparece en la directiva, los hilos no se sincronizan al llegar al final de la estructura `for`.
- **private(*lista de variables*)**: Explicada en la directiva `parallel`.

- **`shared(`*`lista de variables`*`)`**: Explicada en la directiva `parallel`.
- **`firstprivate(`*`lista de variables`*`)`**: Explicada en la directiva `parallel`.
- **`lastprivate(`*`lista de variables`*`)`**: Las variables de la lista se comportan como si de una cláusula `private` se tratese, con la excepción de que el hilo que ejecuta la última iteración del bucle for copia el valor de sus variables privadas en las variables globales originales.
- **`reduction(`*`operador`*`:`*`lista de variables`*`)`**: Explicada en la directiva `parallel`.

Restricciones

La corrección del programa no puede depender de que un determinado hilo ejecute una determinada operación, puesto que no se puede saber con seguridad qué hilo ejecutará qué iteración.

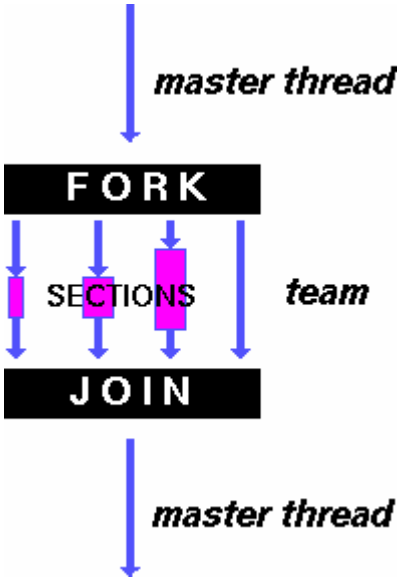
Asímismo, el bucle for que aparece tras la directiva debe estar escrito en forma canónica (consultar la especificación de OpenMP).

Ejemplo de directiva for

```
#include <omp.h>
#define TAMSEGMENTO 100
#define N     1000

main ()
{

int i, segmento;
float a[N], b[N], c[N];

/*Inicializaciones*/
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;
segmento = TAMSEGMENTO;

#pragma omp parallel shared(a,b,c,segmento) private(i)
  {

  #pragma omp for schedule(dynamic,segmento) nowait
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];

  }  /*fin de la región paralela*/
}
```

Estructuras de división de trabajo: La directiva sections

La estructura `sections` divide el trabajo en varias secciones bien diferenciadas, cada una ejecutada por un hilo. Se puede usar para implementar un tipo de "paralelismo funcional".




```
#pragma omp sections [cláusulas] fin de línea
  {
   #pragma omp section fin de línea
        bloque estructurado
   #pragma omp section fin de línea
        bloque estructurado
  ...
   }
```

La directiva `sections` conforma una estructura no iterativa de división de trabajo. Especifica que las secciones (sections) contenidas en ella son ejecutadas cada una por un hilo del equipo. Qué sección ejecuta cada hilo es algo que no se puede determinar con exactitud; un hilo puede ejecutar más de una sección en una estructura `sections`, si es suficientemente rápido y la implementación lo permite.

Hay una sincronización implícita al final de la estructura, de la misma manera que ocurre con la directiva `for`.

Cláusulas

- **`nowait`**: Explicada en la directiva `for`.
- **`private(`*`lista de variables`*`)`**: Explicada en la directiva `parallel`.

- **`firstprivate(`*`lista de variables`*`)`**: Explicada en la directiva `parallel`.
- **`lastprivate(`*`lista de variables`*`)`**: Explicada en la directiva `parallel`, con la salvedad de que el valor copiado en la variable original al final de la estructura es el que tenía la copia de la variable tras la ejecución de la última sección.
- **`reduction(`*`operador`*`:`*`lista de variables`*`)`**: Explicada en la directiva `parallel`.

Restricciones

Las directivas `section` no pueden aparecer en otro lugar que no sea en el interior de una directiva `sections`.

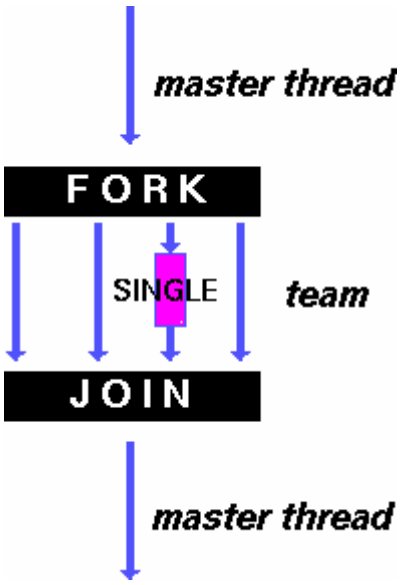
Ejemplo de directiva sections

```
#include <omp.h>
#define N     1000

main ()
{

int i;
float a[N], b[N], c[N];

/*Inicializaciones*/
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;

#pragma omp parallel shared(a,b,c) private(i)
  {

  #pragma omp sections nowait
    {

    #pragma omp section
    for (i=0; i < N/2; i++)
      c[i] = a[i] + b[i];

    #pragma omp section
    for (i=N/2; i < N; i++)
      c[i] = a[i] + b[i];

    }  /*fin de las secciones*/
  }  /*fin de la región paralela*/
}
```

En el código de arriba, cada uno de los bucles for será ejecutado por un hilo del equipo. En caso de haber más de dos hilos en el equipo, la implementación será la que determine cómo actúan los hilos que no tienen una sección asignada.

Estructuras de división de trabajo: La directiva single

La directiva `single` secuencializa una sección del código dentro de una región paralela.




```
#pragma omp single [cláusulas] fin de línea
   bloque estructurado
```

Esta directiva obliga a que el código que contiene sea ejecutada por un único hilo. Aquellos hilos que no estén ejecutando el código de la directiva esperarán al final de la misma.

La directica `single` se suele utilizar para tratar con secciones de código que no son "thread- safe", como por ejemplo las operaciones de entrada/salida.

Cláusulas

- **`nowait`**: Explicada en la directiva `for`.
- **`private(`*`lista de variables`*`)`**: Explicada en la directiva `parallel`.
- **`firstprivate(`*`lista de variables`*`)`**: Explicada en la directiva `parallel`.

Directivas combinadas: Las directivas parallel for y parallel sections

Muchas de las regiones paralelas que se emplean en un programa contienen un único bucle for paralelo, o una directiva `section`. Para evitar tener que incluir tanto la directiva `parallel` como la de reparto de trabajo correspondiente (`for` o `sections`), ambas se pueden combinar en una única directiva: `parallel for` o `parallel sections`. La estructura de la primera es:

```
#pragma omp parallel for [cláusulas] fin de línea
   bloque estructurado
```

Y funciona como una sección paralela con una directiva `for` en su interior. La carga de trabajo se reparte siempre a partes iguales entre los hilos que componen el equipo (schedule STATIC). El resto de las cláusulas normalmente permitidas para las directivas `parallel` y `for` se pueden emplear también en las directivas `parallel for`, con la excepción de la cláusula `nowait` de la directiva `for`.

La estructura de la segunda directiva combinada, parallel sections, es:

```
#pragma omp parallel sections [cláusulas] fin de línea
   {
   #pragma omp section fin de línea
      bloque estructurado
   #pragma omp section fin de línea
      bloque estructurado
     ...
   }
```

Y funciona de manera análoga a la anterior, y todas las cláusulas permitidas para las directivas `parallel` y `sections` lo son también para ella.

Ejemplo de la directiva parallel for

```
#include <omp.h>
#define N       1000
#define TAMSEGMENTO   100

main ()  {

int i, segmento;
float a[N], b[N], c[N];

/*Inicializaciones*/
for (i=0; i < N; i++)
  a[i] = b[i] = i * 1.0;
segmento = TAMSEGMENTO;
```

```
#pragma omp parallel for \
   shared(a,b,c,segmento) private(i) \
   schedule(static,segmento)
  for (i=0; i < n; i++)
    c[i] = a[i] + b[i];
}
```

El código de la sección paralela sería exactamente igual que este otro:

```
#pragma omp parallel \
    {
    #pragma omp for shared(a,b,c,segmento) private(i) \
       schedule(static,segmento)
          for (i=0; i < n; i++)
                c[i] = a[i] + b[i];
    }
```

Para finalizar la descripción de las directivas de división de trabajo presentamos una tabla resumen de las cláusulas que pueden ser empleadas con cada una de las directivas hasta ahora vistas:

Cláusula	Directiva					
	PARALLEL	FOR	SECTIONS	SINGLE	PARALLEL FOR	PARALLEL SECTIONS
IF	●				●	●
PRIVATE	●	●	●	●	●	●
SHARED	●	●			●	●
DEFAULT	●				●	●
FIRSTPRIVATE	●	●	●	●	●	●
LASTPRIVATE		●	●		●	●
REDUCTION	●	●	●		●	●
COPYIN	●				●	●
SCHEDULE		●			●	
ORDERED		●			●	
NOWAIT		●	●	●		

Estructuras de sincronización

Las siguientes directivas que veremos forman parte del grupo de directivas de sincronización de OpenMP. Éstas definen estructuras que proporcionan formas de control sobre la manera en que actúa cada hilo respecto a los demás en el equipo, coordinándolo con ellos.

Estructuras de sincronización: La directiva barrier

`barrier` fuerza a los hilos de un equipo a sincronizarse en un determinado punto de la ejecución.

```
#pragma omp barrier fin de línea
```

Cuando un hilo encuentra una directiva `barrier`, espera ahí hasta que todos los demás hilos del equipo han llegado también a ese punto. Después, todos continúan la ejecución del código que se encuentre a continuación.

Las directivas `barrier` deben hallarse siempre dentro de un bloque estructurado.

Estructuras de sincronización: La directiva master

La directiva de sincronización `master` especifica que una determinada región de código sólo va a ser ejecutada por el hilo maestro del equipo.

```
#pragma omp master fin de línea
   structured_block
```

Únicamente el hilo maestro ejecutará el bloque estructurado situado bajo la directiva; los demás hilos continuarán ejecutando el código inmediatamente a continuación del bloque, sin que exista ninguna sincronización implícita.

Estructuras de sincronización: La directiva critical

La directiva `critical` define una sección crítica; esto es, sólo un hilo podrá ejecutar el código de esa sección en un determinado momento.

```
#pragma omp critical [nombre] fin de línea
   structured_block
```

Si un hilo alcanza la sección crítica en el momento en el que otro está ejecutándola, se bloquea a la espera de que el hilo que se encuentra en la sección crítica concluya su ejecución.

Cláusulas

- **`nombre`**: Los nombres de las secciones críticas actúan como identificadores, de manera que dos secciones críticas diferentes con el mismo nombre serán tratadas como si fuera una sola a efectos de sincronización. De la misma manera, todas las secciones críticas sin ningún nombre especificado serán tratadas como si fueran una misma sección crítica.

Ejemplo de directiva critical

Al ejecutar el código:

```
#include <omp.h>

main()
{

int x;
x = 0;

#pragma omp parallel shared(x)
  {
  x = x + 1;
  }  /*fin de la región paralela*/
}
```

Es posible que dos hilos ejecuten la sentencia `x = x + 1` antes de que cualquiera de ellos actualice el valor de `x`. Es decir, ambos hilos considerarían que `x` vale 0, y almacenarían como nuevo resultado el valor 1, en lugar de 2, que sería el correcto. Para solucionar este tipo de problemas se recurre a la directiva critical:

```
#include <omp.h>

main()
{

int x;
x = 0;

#pragma omp parallel shared(x)
  {
  #pragma omp critical
    x = x + 1;
  }  /*Fin de la región paralela*/
}
```

De manera que sólo se permite que un hilo acceda y actualice el valor de `x` en cada momento, obteniéndose siempre al final el valor correcto.

Estructuras de sincronización: La directiva atomic

Esta directiva establece que una determinada variable debe actualizarse de manera atómica.

```
#pragma omp atomic fin de línea
   sentencia de asignación
```

La directiva atomic no es más que una versión reducida de la directiva critical, que permite únicamente garantizar que ningún hilo leerá el contenido de una variable que aparece en la sentencia inmediatamente posterior mientras otro está actualizando su valor.

Restricciones

La expresión de la sentencia de asignación, considerando a `x` como la variable a actualizar, debe poseer uno de los siguientes formatos:

- x *operador*= *expresión*
- *x++*
- *++x*
- *x--*
- --x

NOTA: *expresión* no debe hacer referencia a la variable `x`.

Estructuras de sincronización: La directiva ordered

La directiva `ordered` especifica que las iteraciones del bucle contenido en ella deben ser ejecutadas en el mismo orden en que se ejecutarían de manera secuencial.

```
#pragma omp ordered fin de línea
   bloque estructurado
```

Restricciones

Cualquier bucle paralelo que contenga una directiva `ordered` debe contener a su vez la cláusula `ordered`. Asímismo, las directivas `ordered` deben aparecer siempre dentro del ámbito de un bucle paralelo, bien sea creado con la directiva `for` o con `parallel for`. Sólo puede haber una directiva ordered en un determinado bucle, y sólo un hilo puede ejecutar su bloque estructurado en un determinado momento.

Estructuras de sincronización: La directiva flush

La directiva `flush` establece un punto de sincronización en el cual el programa debe ofrecer una visión consistente de la memoria, actualizando en ese momento el valor de las variables que sean visibles por los hilos.

#pragma omp flush *[(lista de variables)] fin de línea*

Salvo que en ellas se especifique la cláusula `nowait`, la directiva `flush` está implicita en las siguientes directivas de OpenMP: `barrier`, `critical` (tanto al inicio como al final de la misma), `ordered` (al inicio y al final), `parallel` (al final), `for` (al final), `sections` (al final) y `single` (al final).

Cláusulas

- **(*lista de variables*):** Especifica de manera opcional qué variables deben ser actualizadas en la directiva `flush`. Si no aparece esta cláusula, todas las variables son actualizadas.

La directiva threadprivate

Esta directiva, encuadrada ya fuera de las estructuras de sincronización, hace que una serie de variables se conviertan en privadas, y persistentes a lo largo de la ejecución de varias regiones paralelas.

lista de variables
#pragma omp threadprivate (*lista de variables*)

`threadprivate` debe aparecer después de la declaración de las variables a las que afecta (las que se encuentran en su lista de variables). Cada hilo de los equipos recibe su propia copia privada de las variables, que no son destruidas al acabar una región paralela. Además, la primera vez que se encuentra una región paralela, las variables declaradas como `threadprivate` no deben tener un valor definido (salvo que la región paralela en cuestión posea la cláusula `copyin`).

Restricciones

OpenMP sólo garantiza el correcto funcionamiento de la directiva `threadprivate` si el número de hilos por equipo no varía de una región paralela a otra.

Ejemplo de directiva threadprivate

```
#include <omp.h>

int alfa;
#pragma omp threadprivate(alfa)

main ()  {

/*Primera región paralela*/
#pragma omp parallel private(i,beta)
  for (i=0; i < 10; i++)
     alfa[i] = beta[i] = i;

/*Segunda región paralela*/
#pragma omp parallel
  printf("alfa[3]= %d y beta[3]= %d\n",alfa[3],beta[3]);
}
```

En este ejemplo, la impresión final de cada hilo será `alfa[3]=3 y beta[3]=` *indefinido*. ¿Por qué? La razón es sencilla: Las copias del vector `alfa`, al ser `threadprivate`, no son destruidas al salir de la primera región paralela, con lo que conservan su valor ser impresas en la segunda. Sin embargo, la el vector `beta` es simplemente utilizado como private, con lo que los valores asignado a las copias en la primera región paralela se pierde al finalizar ésta, con lo que el valor obtenido en la impresión es indefinido.

Rutinas específicas de OpenMP

El estándar de OpenMP define una serie de procedimientos que realizan multitud de funciones. Describimos a continuación algunas de las más importantes (NOTA: para poder emplear estas rutinas es necesario incluir el archivo "`omp.h`" en la cabecera del fichero de código).

La función omp_set_num_threads

La función `omp_set_num_threads` determina el número de hilos que compondrán cada equipo. Su formato es

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

Donde `num_threads` es el número (positivo) de hilos que deben componer un equipo.

Esta función puede ser llamada únicamente en las partes secuenciales del código, fuera de las regiones paralelas, y como explicamos antes, tiene

preferencia sobre el valor establecido para la variable de entorno OMP_NUM_THREADS.

La función omp_get_num_threads

Devuelve el número de hilos que han sido expandidos para la región paralela actual. Formato:

```
int omp_get_num_threads(void)
```

Esta función es llamada normalmente desde una región paralela de código; no obstante, si se llama a `omp_get_num_threads` desde una región secuencial del programa, o bien desde una región paralela que está secuencializada (con la directiva `single`), el valor obtenido en la función será 1.

La función omp_get_thread_num

Devuelve el número identificador dentro del equipo del hilo que ejecuta la función.

```
int omp_get_thread_num(void)
```

El número que devuelve la función varía de 0 (para el hilo maestro) hasta el número de hilos en el equipo-1. Si es llamada desde una región secuencial (o desde una región paralela anidada, en las implementaciones que permitan esta posibilidad), el valor devuelto será 0.

La función omp_get_num_procs

Esta función devuelve el número de procesadores disponibles en ese momento.

```
int omp_get_num_procs(void)
```

Para saber más sobre OpenMP

Este documento es sólo una introducción muy básica a la programación en paralelo con OpenMP. Hay muchos temas que no hemos tratado, tales como las funciones de sincronización y algunas otras, las variables de entorno o la especificación de OpenMP para Fortran, y que son importantes para el conocimiento en profundidad de la API. Para obtener información más completa

y detallada sobre OpenMP, se puede acudir a las siguientes fuentes (todas ellas en inglés):

- http://www.openmp.org – Página oficial de OpenMP.
- http://www.llnl.gov/computing/tutorials/openMP/ - Un sencillo tutorial sobre OpenMP. Escueto pero completo.
- "Parallel programming in OpenMP". Rohit Chandra et al. – Un libro didáctico y explicativo sobre OpenMP.