Cómputo paralelo con openMP y C

Sergio Ivvan Valdez Peña

Centro de Investigación en Matemática A.C.

Guanajuato, México.

OpenMP

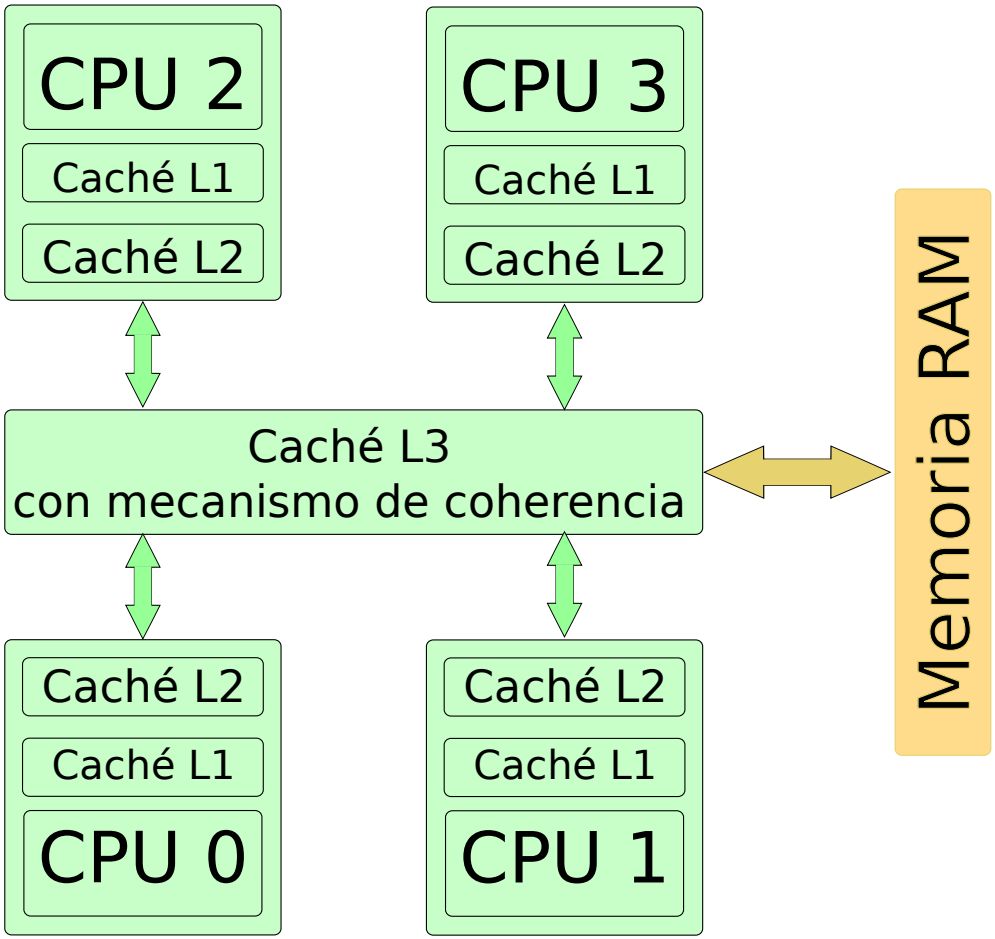
OpenMP: Open Multiprocessing. Es un estándar para C/C++ y FORTRAN para realizar cómputo paralelo en memoria compartida utilizando multi-hilos.
Consiste en:

- Directivas de compilador.
- Rutinas para tiempo de ejecución.
- Variables de entorno.

Breve instroducción a la arquitectura de la computadora

- Modelo de memoria compartida.
- Caches.
- Cache hit y cache miss.
- Predictores y copia de la memoria al cache.

Modelo de memoria compartida



OpenMP

En **no** es OpenMP:

- No es para paralelizar en memoria distribuida.
- No está implementado igual por todos los fabricantes de compiladores.
- No garantiza el uso más eficiente de la memoria compartida.
- No se le requiere checar por dependencia de los datos, conflictos de datos, o *deadlock*.
- No se requiere que se verifique si el programa es *conforme* con el estándar.
- No es síncrono (para entrada o salida).

Directivas de paralelización

Las directivas de paralelización siguen la siguiente estructura:
`#pragma omp` *directiva [clausula] nueva linea*
Directivas de paralización:
#pragma omp atomic
#pragma omp parallel
#pragma omp for
#pragma omp parallel for
#pragma omp ordered
#pragma omp section
#pragma omp sections
#pragma omp parallel sections
#pragma omp single
#pragma omp master
#pragma omp critical
#pragma omp barrier
#pragma omp flush
#pragma omp threadprivate

Clausulas

private(list)
shared(list)
default(none—shared)
copyin(list) firstprivate(list)
lastprivate(list)
reduction(operator:list)
schedule
collapse
ordered
nowait

Modelo de memoria

OpenMP provee de consistencia relajada de memoria, es decir cada hilo puede tener en cache sus datos, y no se le requiere mantener la consistencia de la memoria con los demas hilos todo el tiempo.
El programador es responsable de de asegurar que cada variable se *flushea* por todos los hilos cuando se necesite.

Directivas de compilador

Las directivas de compilador se encargan de:

1. Indicar cuando se debe generar una región paralela.
2. Dividir bloques de codigo entre los hilos.
3. Distribuir las iteraciones de un ciclo entre los hilos.
4. Serializar secciones de código (dentro de una región paralela).
5. Sincronización del trabajo entre los hilos.

Librerias de rutinas de tiempo de ejecución

Las rutinas son funciones que pueden servir para:

- Fijar y obtener el número de hilos.
- Obtener el identificador de cada hilo, el tamaño del *equipo* de hilos.
- Fijar y obtener (el valor) de la funcionalidad de hilos dinámicos.
- Saber si se está en una región paralela y en que nivel.
- Fijar y obtener el valor de la funcionalidad de paralelismo anidado.
- Fijar, inicializar, y terminar candados (locks) y locks anidados.
- Obtener el tiempo (wall time).

Variables de entorno

- Fijar el número de hilos.
- Especificar como se deben de dividir las iteraciones en un ciclo.
- Fijar un hilo a un procesador.
- Habilitar o deshabilitar el paralelismo anidado.
- Habilitar o deshabilitar la funcionalidad de hilos dinámicos.
- Fijar el tamaño del stack de cada hilo.
- Fijar la politica de espera (wait) de los hilos.

Estructura de un programa en C

Programa normal

```
#include <stdio.h>


int main()
{
int n,identificador;

/ / Código serial aquí


}
```

Programa paralelo

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

int main()
{
int n,identificador;

/ Código paralelo

#pragma omp parallel private(identificador)
}
```

Directivas de OpenMP

Inicia región paralela

```
#pragma omp parallel
```

Alcance de las variables

```
#pragma omp parallel private(var01,var02) shared(var03,var04)
```

Las variables: var01 y var02 son privadas, lo que quiere decir que se genera memoria para cada una de estas variables para cada proceso, cuando término la región paralela se regresa la memoria.

Las variables: var03 y var04, son variables compartidas que quiere decir que todos los procesos accesan a la misma memoria, que es la memoria que tenia el hilo principal.

Directivas OMP: región paralela

```
 //Inicio de región paralela
#pragma omp parallel private(identificador)
{


/* Obtiene el numero identificador de cada hilo*/
identificador = omp_get_thread_num();


/* Obtiene el número de hilos que se levantaron*/
nhilos = omp_get_num_threads();
} //fin de región paralela
```

Compilacion de un programa con gcc y openMP

```
gcc -lm -std=c99 -fopenmp codigo.c -o ejecutable
```

-lm indica que se utiliza la librería matemática.
-std=c99 indica que se utilizará el estándar 99 de C.
-fopenmp indica que se utiliza OpenMP.

Directivas OMP: for paralelo

```
 #pragma omp for schedule(static,chunk) nowait
for (i=0; i < N; i++)
```

chunk: Se procesarán “chunk” índices en cada proceso por vez.
static: quiere decir que se repartirá el trabajo de manera estática un chunk para cada proceso por vez, hasta que se término de procesar todos los datos.

Otras opciones: “dynamic” y “guided”

nowait: indica que no es necesario esperar a que todos los procesos terminen para continuar procesando.

Cálculo de tiempo de cómputo

double tiempoinicial=omp_get_wtime();

Tarea 13

13.1 Hacer un for paralelo para la suma de dos vectores, mostrando en que hilo se procesa cada indice.

13.2 Hacer una función para la suma de dos vectores en paralelo. a) Solamente utilizando una región paralela, y dividiendo el inicio y final del for (no usar parallel for, sino variables privadas de inicio y fin). b) Usando un parallel for con dynamic y static en el schedule. Calcular los tiempo de suma de 10000 datos (si tardan muy poco, repetir la suma las veces necesarias para que se realice en el orden de los segundos). Calcular el Speed Up y la eficiencia.

13.3 Programar el método de Gauss-Seidel o Jacobi (el que resulte mas eficiente). Comentar en el reporte como se realizan las operaciones, y cual es la versión del método que se programó. Probar paralelizando el for exterior, el interior, repartiendo por bloques de renglones, etc. escribir en su reporte que versión funciona mejor y porqué (al menos el mejor contra el peor).

Secciones paralelas

```
 #pragma omp parallel shared(a,b,c,chunk)
private(i,j)
{
#pragma omp sections nowait
{
#pragma omp section
.//operaciones que realiza el primer proceso
#pragma omp section
.//operaciones que realiza el segundo proceso
} //fin de las secciones paralelas
}//fin de la región paralela
```

Tarea 13

13.4 Realice el 13.2 utilizando secciones.