PRÁCTICA 3 – SISTEMAS EMPOTRADOS

Entrenamiento con ARM: OpenMP y OpenCV

Lucas Serrano Jiménez

César San Blas Leal

tarea 1.1

**Se deberá estudiar el API de OpenMP y su uso con GNU GCC (gcc, g++), comprobando el correcto funcionamiento de algunos de los ejemplos que hay disponibles en Internet.**

La API de OpenMP es una biblioteca de programación que permite a los programadores paralelizar sus aplicaciones (basados en C, C++ o Fortran) de forma explícita; es decir, el programador debe especificar qué partes del código deben ejecutarse en paralelo. El compilador y el sistema de ejecución se encargarán de distribuir el trabajo entre los distintos procesadores.

Esta API no realiza comprobaciones de ningún tipo en el programa, por lo tanto es responsabilidad del programador asegurarse de que la aplicación sea correcta y segura.

GNU GCC es un compilador de código abierto que soporta OpenMP. Para compilar un programa que utiliza OpenMP en GCC, se debe utilizar la opción -fopenmp.

Algunas de las directivas y cláusulas más comunes de OpenMP son las siguientes:

* **Directiva parallel:** Declara una región de código que se ejecutará en paralelo.
* **Directiva for:** Declara un bucle que se ejecutará en paralelo.
* **Cláusula private:** Declara que una variable solo será visible para el hilo que la creó.
* **Cláusula shared:** Declara que una variable será visible para todos los hilos.
* **Cláusula reduction:** Declara que una variable será compartida por todos los hilos y que su valor será la suma de los valores individuales.

A continuación, un sencillo programa de ejemplo que muestra el uso de algunas de estas directivas y cláusulas:

#include <stdio.h>

#include <omp.h>

int main() {

  int i, n = 1000;

  float sum = 0.0f;

  // Sección en paralelo

  #pragma omp parallel

  {

//`sum` será visible para todos los hilos

#pragma omp for

for (i = 0; i < n; i++) {

   sum += i;

}

  }

  printf("sum = %f\n", sum);

  return 0;

}

Cuyo resultado muestra el valor final de *sum* como suma de todos los bucles realizados en paralelo, aunque cada ejecución termina con un valor distinto. Esto se debe a que cada hilo tiene su propia variable *sum* y al final se juntan todas, produciendo variaciones porque no hay un orden establecido. Si se declarase *sum* como *shared*, escribiendo *#pragma omp for reduction(+:sum)* antes del *for* compartido, el resultado se estabiliza en: *sum = 499500.00.*

tarea 1.2

**1. Explique qué hace el código que se añade a continuación.**

El código expuesto en el guión de la práctica suma los valores de los arrays a y b, previamente inicializados secuencialmente de 0 a N-1. Lo característico del código es que utiliza OpenMP para ejecutar las sumas en paralelo, distribuyendo la carga de trabajo entre varios hilos.

**2. Delimite las distintas regiones OpenMP en las que se está expresando paralelismo.**

Las regiones donde se expresa paralelismo en el código implementado son:

* #pragma omp parallel shared(a, b, c, nthreads, chunk) private(i, tid)

Esta línea comienza la región paralela con OpenMP. En ella se establecen las variables que se mantendrán inalteradas entre hilos (*i* y *tid*) y las que se compartirán (*a, b, c, nthreads* y *chunk*).

* #pragma omp for schedule(dynamic,chunk)

Esta directiva divide el bucle for en tareas que se ejecutarán en paralelo entre los hilos.

**3. Explique con detalle qué hace el modificador de OpenMP *schedule*. ¿Cuáles pueden ser**

**sus argumentos? ¿Qué función tiene la variable *chunk* en el código? ¿A qué afecta?**

El operador *shedule* sirve para configurar la distribución de las iteraciones del bucle for entre los hilos paralelos.

El valor de chunk sirve para especificar cuántas iteraciones se asignan a cada hilo. En este caso, este valor viene definido por la constante CHUNKSIZE que vale 10, por lo que por cada bloque atenderá a 10 iteraciones.

**4. ¿Qué función tiene el modificador de OpenMP dynamic en el código?**

Establece que las iteraciones se asignan a medida que los hilos terminan su trabajo anterior, lo que ayuda a equilibrar la carga de trabajo de manera más eficiente si algunas iteraciones toman más tiempo que otras y a evitar que hayan hilos inactivos.

**5. Investigue qué pasa si no declara como privadas las variables *i* y *tid*.**

Si la variable *i* no fuese privada, se modificaría simultáneamente por todos los hilos que estén trabajando con ella por lo que el bucle for no funcionaría correctamente. En el caso de *tid,* si dentro del hilo el valor pudiera ser alterado mientras se ejecuta, al mostrar por pantalla el resultado de la suma el identificador del hilo sería erróneo.

tarea 1.3

**Diseñe un programa en C/C++ para multiplicar dos matrices cuadradas con elementos de tipo double(punto flotante de 64 bits) entre 0 y 1. Después multiplique la matriz resultante por otra matriz de enteros entre 0 y 255 elemento a elemento.**