Práctica 1

Paralelismo de datos y exclusión mutua

NOTA IMPORTANTE: Es fundamental leer atentamente los enunciados de los ejercicios, para realizarlos correctamente. Los fallos derivados de no haber leído el enunciado se penalizarán fuertemente en la nota de la práctica.

1. Introducción

En esta práctica se propone paralelizar el problema de la multiplicación de matrices en un multiprocesador de memoria compartida (*multi-core*), tanto utilizando la librería `threads` de C++ como mediante el paradigma de programación OpenMP. La particularidad del problema propuesto es que las matrices que se propone multiplicar son **triangulares**, es decir tienen los elementos por debajo o por encima de la diagonal todos iguales a cero. Esto tendrá implicaciones importantes sobre el rendimiento obtenido al paralelizar diferentes implementaciones del algoritmo. Finalmente, se paralelizará la búsqueda del elemento máximo de la matriz resultante.

La práctica consta de 3 ejercicios que proponen la paralelización del programa de multiplicación de matrices triangulares, utilizando `threads` de C++ y OpenMP, el análisis de los distintos algoritmos de equilibrio de carga de OpenMP en diferentes casos y el estudio del impacto de la exclusión mutua sobre el rendimiento.

2. Objetivos

Mediante el desarrollo de esta práctica se pretenden cubrir los siguientes objetivos:

- Paralelizar un programa usando el paradigma de memoria compartida utilizando OpenMP y entender sus ventajas sobre la gestión manual de threads.
- Comprender y comprobar el impacto del desequilibrio de carga de trabajo en el rendimiento de los programas paralelos.
- Entender la importancia del correcto uso de la exclusión mutua.

3. Material Necesario

Para realizar la práctica se proporciona un único fichero comprimido `prac2.zip`, en la plataforma Moodle. Este archivo incluye:

- `MatMul.c`: Fichero con la implementación de las funciones necesarias para realizar la practica. Consta de las siguientes funciones:

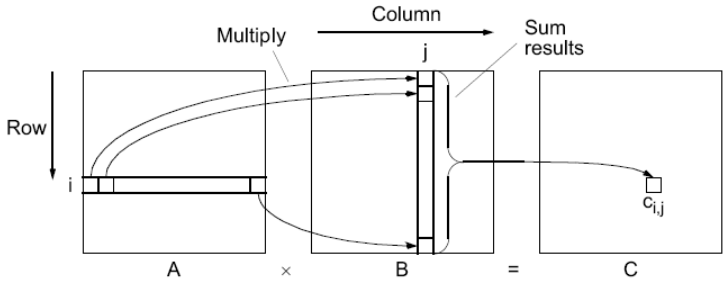

Figura 1: Multiplicación de matrices, $C = A \times B$.

- `Init_Mat_Sup`: Inicializa una matriz triangular superior con números aleatorios entre 0 y 99.
- `Init_Mat_Inf`: Inicializa una matriz triangular inferior con números aleatorios entre 0 y 99.
- `Multiplicar_Matrices`: Realiza la multiplicación de las matrices.
- `Multiplicar_Matrices_Sup`: Multiplica de forma optimizada una matriz triangular superior por una inferior.
- `Multiplicar_Matrices_Inf`: Multiplica de forma optimizada una matriz triangular inferior por una superior.
- `Escribir_Matriz`: Escribe una matriz en la pantalla. Se puede redireccionar a fichero desde la consola.
- `Calcula_Maximo`: Obtiene el máximo de una matriz.

- `ejer2.sh`: Shell script que muestra como ejecutar una serie de pruebas de forma automática a partir de comandos de Linux.

4. Problema

Una matriz es un vector bidimensional de números organizados en filas y columnas. Una matriz $N \times M$ tiene N filas y M columnas de elementos. La multiplicación de dos matrices, A y B, es una operación matemática que produce una matriz resultado C, cuyos elementos, c_{ij}, se calculan de la siguiente forma (figura 1):

$$c_{ij} = \sum_{k=0}^{P-1} a_{ik} b_{kj}$$

donde A es una matriz $M \times P$ y B es una matriz $P \times N$ ($0 \leq i < N$, $0 \leq j < M$).

En esta práctica trabajaremos con matrices **triangulares superiores** y **triangulares inferiores**. Esto significa que todos los valores de los coeficientes de la matriz que están por debajo (o por encima, respectivamente) de la diagonal son iguales a cero:

$$a_{ij} = b_{ij} = 0, \forall\ j < i$$

Asimismo por simplicidad vamos a considerar solamente matrices **cuadradas**, es decir, tienen que tener el mismo número de filas que de columnas, $N = M$.

Estas propiedades hacen que se pueda optimizar el algoritmo de multiplicación, ya que todas las operaciones de multiplicación por cero se pueden evitar.

5. Datos de Entrada y Resultados de Salida

El programa MatMul recibe dos parámetros en la línea de comandos, que deben darse en el orden indicado. Si no se introducen alguno de los parámetros, el programa le asigna los valores por defecto indicados:

- *Dimensión de las matrices*: Por simplicidad se utilizan matrices cuadradas, por lo que es suficiente indicar una dimensión. Por defecto este valor es 5.
- *Tamaño del Bloque*: Indica el tamaño de bloque a utilizar con los distintos algoritmos de equilibrio de carga de trabajo (cláusula `schedule`). Por defecto tiene el valor 1.

6. Desarrollo

NOTA: Verificación de Resultados En todos los ejercicios es muy importante comprobar que los resultados obtenidos en la versión paralela son correctos. Para ello debes almacenar los resultados de la versión secuencial y de la paralela en diferentes ficheros y compararlos con el comando `diff` de linux. Si no lo conoces puedes consultar el manual en línea (`man diff`). Las pruebas deben utilizar una matriz de dimensión tal que requiera unos 10 segundos de cómputo en secuencial. Para las implementaciones OpenMP, **utiliza una única región** `parallel` que englobe todas las porciones de código a paralelizar.

Ejercicio 1

Paraleliza la multiplicación de matrices usando la función `Multiplicar_Matrices` mediante `threads` de C++, de modo que cada thread se encargue de una porción contigua de la matriz de salida. Mide el tiempo de ejecución de la multiplicación e imprímelo por pantalla. A continuación, paraleliza la misma función utilizando las directivas y/o funciones de *runtime* de la API de OpenMP. El programa debe determinar en tiempo de ejecución el número máximo de hilos y crear ese número de threads en la región paralela. Añade al código las funciones de OpenMP necesarias para medir el tiempo de ejecución de la función `Multiplicar_Matrices` e imprímelo por pantalla. Contesta a las siguientes cuestiones:

1. Explica cómo se reparte el trabajo entre los threads en cada una de las implementaciones.
2. Justifica cómo has etiquetado las variables.
3. Compara la ganancia (speedup) obtenido mediante ambas paralelizaciones y el esfuerzo invertido en cada una de ellas.

Ejercicio 2

Trabajaremos ahora con la función `Multiplicar_Matrices_Inf`. Paraleliza utilizando C++ y OpenMP de manera equivalente al ejercicio anterior.

1. Compara el speedup obtenido por ambas implementaciones con los resultados del ejercicio anterior. ¿Qué está sucediendo?

Añade la clausula `schedule` para modificar el reparto de trabajo entre los threads. Mide el tiempo de respuesta con los tres algoritmos disponibles (`static, dynamic, guided`) sin poner ningún tamaño de bloque. Contesta a las siguientes cuestiones:

2. ¿Qué diferencias observas en cuanto a la ganancia obtenida sin la clausula `schedule`? Explica a qué se debe esta diferencia.
3. ¿Cuál de los tres algoritmos de equilibrio de carga obtiene mejor resultado? Explica porqué ocurre esto en función de cómo se reparte la carga de trabajo y las operaciones que tiene que realizar cada thread.
4. Explica cómo implementarías una solución equivalente utilizando `threads` de C++.

5. Para el algoritmo `static` piensa cuál será el tamaño del bloque óptimo, en función de cómo se reparte la carga de trabajo.
6. Para el algoritmo `dynamic` determina experimentalmente el tamaño del bloque óptimo. Para ello mide el tiempo de respuesta para al menos 10 tamaños de bloque diferentes. Presenta una gráfica de los resultados y explícalos.

Ejercicio 3

Paraleliza el cálculo del máximo del resultado utilizando la función `Calcula_Maximo` mediante la librería `threads`. Realiza dos implementaciones: una en la que que la variable que contiene el máximo sea actualizada dentro del cuerpo del bucle y otra en la que esta variable sea actualizada **únicamente** fuera del bucle, acumulando el resultado parcial de cada thread en otra variable. Compara el rendimiento de ambas implementaciones para ejecuciones con 2 y 8 threads con respecto de secuencial, considerando sólo el tiempo del cálculo del máximo y no el de todo el programa.

A continuación, realiza cuatro implementaciones distintas de esta función utilizando OpenMP y comprueba el rendimiento de cada una de ellas, calculando el speedup obtenido. Explica los resultados obtenidos. Las cuatro implementaciones son las siguientes:

a. Implementación mediante directivas OpenMP.

b. Implementación mediante funciones de runtime.

c. Ejecución secuencial de esa parte del código.

c. Implementación con variables privadas a cada thread y selección del máximo de todas ellas.

7. Evaluación

La evaluación de estas prácticas se realizará mediante una única entrega, que tendrá como fecha límite el día **5 de Diciembre de 2023.** Esta entrega constará de una memoria en formato `pdf` siguiendo la plantilla proporcionada. En esta memoria se recogerán los resultados de todos los ejercicios propuestos en las diferentes prácticas. Es fundamental **explicar los resultados y conclusiones obtenidas**, ya que en caso contrario se penalizará fuertemente la nota. Adicionalmente, se deben entregar los ficheros con el código fuente de los programas desarrollados. La entrega del informe y los programas se realizará en un único fichero comprimido en `zip`, a través de la plataforma Moodle.