Mariana Osorio Vásquez

**Multiplicación de matrices**

Se elaboró un algoritmo de multiplicación de matrices con hilos, estas matrices son de datos double generados aleatoriamente. Para esto, se realizan los siguientes archivos:

**mmClasicaOpenMP.c**

Este archivo contiene el código de la creación y multiplicación de matrices. Entre sus funciones están:

* InicioMuestra

*Entradas*: No tiene entradas.

*Posibles salidas en pantalla:* No produce salidas en pantalla.

*Comportamiento*: Esta función captura el tiempo de inicio de una operación. Utiliza la función gettimeofday() para obtener el tiempo actual del sistema y lo almacena en la variable global inicio.

*Diseño*: La función gettimeofday(&inicio, (void \*)0) se encarga de obtener el tiempo. El segundo argumento es nulo porque no se necesita la zona horaria.

* FinMuestra

*Entradas*: No tiene entradas.

*Posibles salidas en pantalla:* Un número de tipo flotante con 9 dígitos que representa el tiempo transcurrido en microsegundos, seguido de un salto de línea.

*Comportamiento*: Esta función calcula el tiempo transcurrido desde que se llamó a InicioMuestra(). Resta el tiempo de inicio (inicio) al tiempo actual (fin) y luego convierte el resultado a microsegundos para mostrarlo en pantalla.

*Diseño:*

1. Se obtiene el tiempo actual usando gettimeofday().
2. Se restan los campos tv\_usec (microsegundos) y tv\_sec (segundos) de la estructura fin con los de inicio.
3. Se calcula el tiempo total en microsegundos usando la fórmula (fin.tv\_sec \* 1000000 + fin.tv\_usec).
4. Se imprime el resultado formateado con printf("%9.0f \n", tiempo).

* impMatrix

*Entradas*: Un puntero a un arreglo de tipo double (matrix) que representa la matriz y un entero D que indica la dimensión de la matriz.

*Posibles salidas en pantalla:* Una representación de la matriz en formato de tabla si su dimensión es menor que 9. De lo contrario, no imprime nada.

*Comportamiento*: Imprime los elementos de la matriz en la consola. Se incluye una condición para evitar imprimir matrices muy grandes, ya que la salida podría ser excesivamente larga.

*Diseño*:

1. Se verifica si la dimensión D es menor que 9.
2. Si es así, se itera sobre todos los elementos del arreglo.
3. Cada vez que el índice es un múltiplo de D, se inserta un salto de línea para representar una nueva fila.
4. Cada elemento se imprime seguido de un espacio.
5. Al final, se imprime una línea de separación.
6. Si la dimensión es mayor o igual a 9, la función no hace nada y termina.

* iniMatrix

*Entradas*: Dos punteros a arreglos de tipo double (m1 y m2) que representan las matrices y un entero D que indica la dimensión de las matrices.

*Posibles salidas en pantalla:* No produce salidas en pantalla.

*Comportamiento*: Inicializa las matrices de entrada con valores aleatorios.

*Diseño*:

1. Se recorre un bucle desde i=0 hasta D\*D-1 para cubrir todos los elementos de las matrices.
2. Dentro del bucle, se asigna un valor aleatorio entre 0 y 99 a cada elemento de ambas matrices (\*m1 y \*m2) usando la función rand()%100.
3. Los punteros m1 y m2 se incrementan en cada iteración (m1++, m2++) para avanzar al siguiente elemento en los arreglos. multiMatrix

* multMatrix

*Entradas*: Tres punteros a arreglos de tipo double: mA y mB (matrices de entrada) y mC (matriz de salida), además de un entero D que es la dimensión de las matrices.

*Posibles salidas en pantalla*: No produce salidas en pantalla.

*Comportamiento*: Multiplica las matrices mA y mB y almacena el resultado en mC. Este proceso se paraleliza utilizando OpenMP.

*Diseño*:

1. Se declaran las variables Suma, pA y pB para almacenar la suma del producto y los punteros para recorrer las matrices.
2. #pragma omp parallel: Crea un equipo de hilos para ejecutar el bloque de código en paralelo.
3. #pragma omp for: Divide las iteraciones del bucle externo (el de la variable i) entre los hilos del equipo.
4. El bucle externo recorre las filas de la matriz mA (i).
5. El bucle intermedio recorre las columnas de la matriz mB (j).
6. Dentro de estos bucles, se inicializan los punteros pA y pB para apuntar al inicio de la fila i de mA y al inicio de la columna j de mB, respectivamente.
7. Un bucle interno (k) recorre los elementos para calcular el producto punto de la fila i de mA y la columna j de mB.
8. pA++ y pB+=D se utilizan para avanzar a través de los elementos de la fila de mA y la columna de mB.
9. El resultado del producto punto (Suma) se asigna al elemento correspondiente en la matriz resultante mC (mC[i\*D+j]).

Todas estas funciones se usan en el main, donde:

1. Se verifica que el usuario haya proporcionado los argumentos de línea de comandos correctos: la dimensión de la matriz (SIZE) y el número de hilos (Hilos).
2. Se asigna dinámicamente memoria para las tres matrices (matrixA, matrixB, matrixC) usando calloc(), que inicializa todos los valores en 0.
3. Se establece el número de hilos de OpenMP con omp\_set\_num\_threads(TH).
4. Se inicializan las matrices de entrada (matrixA y matrixB) con valores aleatorios llamando a iniMatrix().
5. Se muestran las matrices de entrada si sus dimensiones son pequeñas llamando a impMatrix().
6. Se inicia la medición del tiempo con InicioMuestra().
7. Se realiza la multiplicación de matrices llamando a multiMatrix().
8. Se finaliza la medición del tiempo y se imprime el resultado con FinMuestra().
9. Se muestra la matriz resultante (matrixC) si es pequeña.
10. Finalmente, se libera la memoria asignada dinámicamente con free() para evitar fugas de memoria.

**MakeFile**

Para simplificar la compilación del código, se utiliza un Makefile. Este archivo automatiza el proceso de construcción, permitiendo compilar el programa con un simple comando. Las variables FOPENMP = -fopenmp -O3 son cruciales, ya que habilitan el soporte de OpenMP para el paralelismo y optimizan el código, garantizando un mejor rendimiento. El objetivo ALL se encarga de compilar el programa principal mmClasicaOpenMP, mientras que el objetivo clean elimina el archivo ejecutable, facilitando la limpieza del proyecto.

Este Makefile centraliza el proceso de compilación y enlazado de librerías, evitando que el usuario tenga que recordar y escribir las banderas de compilación manualmente cada vez. Con solo ejecutar make, el código fuente se transforma en un ejecutable listo para ser utilizado, agilizando el flujo de trabajo del desarrollador.

**Lanzador.pl**

El script en Perl, Lanzador.pl, es una herramienta de automatización diseñada para la evaluación rigurosa del rendimiento del programa mmClasicaOpenMP. Su propósito principal es ejecutar el programa de forma sistemática y por lotes, controlando dos variables clave: el tamaño de la matriz y el número de hilos.

El script define variables cruciales para su funcionamiento. El arreglo @Size\_Matriz contiene una serie de dimensiones que van desde tamaños pequeños hasta matrices muy grandes. De manera similar, el arreglo @Num\_Hilos establece los números de hilos que se usarán en las pruebas, lo que permite evaluar cómo el programa se escala con la adición de recursos. Un aspecto fundamental es que cada combinación de tamaño y número de hilos se ejecuta 30 veces. Este número no es arbitrario, es un valor estándar en la estadística que ayuda a garantizar la validez de los resultados y a reducir el impacto de las fluctuaciones en el sistema, como la actividad de otros procesos o las interrupciones del sistema operativo. Esto es esencial para evitar la contaminación de datos, asegurando que el rendimiento medido sea representativo del comportamiento real del programa.

El núcleo del script consiste en bucles anidados que recorren cada combinación de parámetros. Para cada iteración, el script crea un nombre de archivo único, como mmClasicaOpenMP-240-Hilos-1.dat, y ejecuta el programa. La salida de cada ejecución, que es el tiempo transcurrido en microsegundos, se redirige y se añade (>>) a su respectivo archivo de datos. Este método garantiza que los resultados de cada prueba queden organizados y no se mezclen, facilitando el posterior análisis estadístico, la creación de gráficos y la interpretación de los datos para entender el comportamiento del algoritmo. En resumen, Lanzador.pl convierte una tarea manual, repetitiva y propensa a errores en un proceso automatizado, eficiente y confiable para la recolección de datos de rendimiento.

**Plan de pruebas**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Descripción del caso | Valores de entrada | Resultado esperado | Resultado obtenido | Evidencia de ejecución |
| Multiplicación de matriz 5x5 |  |  |  |  |
| Multiplicación de matriz 6x6 |  |  |  |  |
| Multiplicación de matriz 7x7 |  |  |  |  |