Análisis de datos reto 1 HPC

Presentado por:

Cristian Camilo Perilla Castaño

Daniel Villada Moncada

Luis Miguel Ortiz Alarcon

Carlos A. Vivas Reyes

Presentado a:

Ramiro Andres Barrios Valencia

Universidad Tecnológica de Pereira

Facultad de Ingenierías

High Performance Computing

Pereira

2020

**Introducción.**

Este documento presenta un análisis a los datos de tiempo obtenidos en los programas desarrollados en c++ de multiplicación de matrices, un primer programa desarrollado de forma secuencial (estándar) y un segundo usando hilos apoyados en la librería pthread.

**CARACTERÍSTICAS DEL COMPUTADOR**

|  |  |
| --- | --- |
| **Procesador** | AMD Ryzen R5 2600 6 núcleos 12 hilos overclockeado a 4Ghz |
| **Ram** | 2 modulos de 8gb c/u DDR4 3200MHz |
| **Disco duro** | SSD kingston UV400 hasta 550MB/s lectura y 350MB/s escritura |
| **S.O** | Ubuntu 18.04lts |
| **Número de cpu** | 12 |

**DESCRIPCIÓN DEL PROCESO**

El fin de esta práctica era construir un programa secuencial que multiplicara matrices de tamaño N x N y correrlo en un computador de mediana potencia y características para ver como los núcleos interactuaban y cuanto tarda , despues, la práctica giró en torno a construir el mismo programa para multiplicación de matrices pero usando programación concurrente con el fin de explotar al máximo la capacidad computacional, obteniendo unos datos de tiempo por cada ejecución de matriz, que se lanzaba al programa iniciando una multiplicación básica de una matriz 10 x 10 y aumentando gradualmente hasta llegar a multiplicar una matriz de tamaño 5000 x 5000.

Cuando se ejecutó por primera vez la multiplicación de matrices de forma secuencial era evidente que a la máquina le costaba mantener el ritmo de multiplicaciones debido a que la forma en como estaba diseñado el programa no era óptimo y tenia un limite de matriz que llegaba a 100 x 100, de ahí en adelante el programa se interrumpia por el S.O debido a que se llenaba la memoria al almacenar los datos obtenidos por la multiplicación .

Al momento de hacer el programa con hilos se hizo concurrencia en tres diferentes instancias:

-La multiplicación entre fila y columna

-La multiplicación entre fila y matriz

-La multiplicación de matrices

Analizando la tercer opción, fue descartada de inmediato ya que ni siquiera generaba más de un hilo de trabajo.

La primer opcion tambien se descarto debido a que generaba tantos hilos como celdas había, a este efecto se le conoce como alta granularidad además de existir otro aspecto que ralentizaba el proceso como lo era el constante cambio de contexto.

Por lo que la mejor la segunda opción fue la mejor debido a que si lograba reducir el tiempo de ejecución del algoritmo secuencial y no sufria por los problemas antes mencionados.

**EXPLICACION DEL CODIGO**

Vamos a describir ambos códigos al mismo tiempo ya que el concurrente tiene una pequeña modificación en código en comparación con el secuencial, se inicia con la carga de librerías típica, incluyendo además una de tiempo (chrono) y la de concurrencia (thread).

Luego empezamos a encontrar funciones, la primera se encarga de llenar la matriz con números random de 1 a 1000, la siguiente función muestra la matriz, esta es utilizada para comprobar la veracidad de la multiplicación de matrices pero normalmente no es llamada ya que no es la finalidad de este código, luego está la función “multiplication” que es la encargada de multiplicar matrices, en el caso de concurrencia, esta sería la fracción paralelizable, esta función cuenta con 2 de los 3 ciclos en cascada para recorrer de forma correcta las matrices.

Luego encontramos el main, definimos variables locales, pedimos datos y se llenan las matrices y creamos un vector para almacenar los hilos en el orden que son ejecutados, luego iniciamos el contador para inmediatamente encontrarnos con el primer ciclo de la multiplicación de matrices en el que empezamos a crear los hilos con las multiplicaciones de filas por matriz independiente llamando la función multiplication y a almacenarlos en el vector de hilos mencionado anteriormente, luego de que termine este proceso empezamos a recorrer el vector de hilos almacenados con el fin de poderlos leer en el orden correcto y también dar espera en caso de que alguno de estos no haya finalizado, por último detenemos el contador y efectuamos la resta de tiempo final menos tiempo inicial y así obtenemos el tiempo de ejecución para visualizarlo en pantalla.

**DESCRIPCIÓN DE LAS GRÁFICAS**

**Tabla de datos para el reto**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tamaño Matriz | Secuencial (promedio) | Paralelizado(promedio) | total celdas | Observed speedup  (wall-clock time of serial execution/wall-clock time of parallel execution) |
| 10 | 0 | 0 | 100 |  |
| 100 | 0,15625 | 0,0090186 | 10000 | 17,32530548 |
| 500 | 1,65625 | 0,282046 | 250000 | 5,872269062 |
| 1000 | 13,4219 | 1,88423 | 1000000 | 7,123281128 |
| 1500 | 46,0781 | 6,18571 | 2250000 | 7,449120634 |
| 2000 | 110,281 | 14,7171 | 4000000 | 7,493392041 |
| 2500 | 267,906 | 36,765 | 6250000 | 7,286984904 |
| 3000 | 482,281 | 69,7156 | 9000000 | 6,917834746 |
| 3500 | 759,948 | 112,914 | 12250000 | 6,730325735 |
| 4000 | 1247,84 | 174 | 16000000 | 7,184707508 |
| 4500 | 1752,5 | 255 | 20250000 | 6,859636762 |
| 5000 | 2425,03 | 354,793 | 25000000 | 0,006835055934 |
|  |  |  |  |  |

**Tabla de datos para matriz 10x10 (con 12 hilos)**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz 10x10 | Secuencial (promedio) | Paralelizado(promedio) |
| 1 | 0 | 0 |
| 2 | 0,000125 | 0,000004 |
| 3 | 0,000235 | 0,000026 |
| 4 | 0,00002 | 0,000001 |
| 5 | 0,000004 | 0,000001 |
| promedio | 0 | 0 |

**Tabla de datos para tamaño 100x100 (con 12 hilos)**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz 100x100 | Secuencial (promedio) | Paralelizado(promedio) |
| 1 | 0,18625 | 0,0091183 |
| 2 | 0,14925 | 0,0089189 |
| 3 | 0,16325 | 0,0090186 |
| 4 | 0,15621 | 0,0100206 |
| 5 | 0,12629 | 0,0080166 |
| promedio | 0,15625 | 0,0090186 |

**Tabla de datos para tamaño 500x500 (con 12 hilos)**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz 500x500 | Secuencial (promedio) | Paralelizado(promedio) |
| 1 | 1,75629 | 0,289046 |
| 2 | 1,61925 | 0,376046 |
| 3 | 1,55625 | 0,282046 |
| 4 | 1,65621 | 0,182046 |
| 5 | 1,69325 | 0,282050 |
| promedio | 1,65625 | 0,282042 |

**Tabla de datos para tamaño 1000x1000 (con 12 hilos)**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz 1000x1000 | Secuencial (promedio) | Paralelizado(promedio) |
| 1 | 13,4229 | 1,86523 |
| 2 | 13,4202 | 1,86123 |
| 3 | 13,3219 | 1,78423 |
| 4 | 13,1219 | 1,68423 |
| 5 | 13,4219 | 1,98423 |
| promedio | 13,4219 | 1,88423 |

**Tabla de datos para tamaño 1500x1500 (con 12 hilos)**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz 1500x1500 | Secuencial (promedio) | Paralelizado(promedio) |
| 1 | 45,0781 | 6,19571 |
| 2 | 44,0780 | 6,18251 |
| 3 | 46,0791 | 6,18571 |
| 4 | 46,0781 | 6,29571 |
| 5 | 47,0681 | 6,09571 |
| promedio | 46,0781 | 6,18571 |

**Tabla de datos para tamaño 2000x2000 (con 12 hilos)**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz 2000x2000 | Secuencial (promedio) | Paralelizado(promedio) |
| 1 | 109,281 | 14,7171 |
| 2 | 111,265 | 14,7854 |
| 3 | 110,281 | 14,7171 |
| 4 | 110,381 | 14,8771 |
| 5 | 110,291 | 14,7187 |
| promedio | 110,281 | 14,7171 |

**Tabla de datos para tamaño 2500x2500 (con 12 hilos)**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz 2500x2500 | Secuencial (promedio) | Paralelizado(promedio) |
| 1 | 267,905 | 32,865 |
| 2 | 268,176 | 35,745 |
| 3 | 269,906 | 36,785 |
| 4 | 267,916 | 37,765 |
| 5 | 267,908 | 40,665 |
| promedio | 267,906 | 36,765 |

**Tabla de datos para tamaño 3000x3000 (con 12 hilos)**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz 3000x3000 | Secuencial (promedio) | Paralelizado(promedio) |
| 1 | 481,281 | 68,7154 |
| 2 | 483,281 | 69,7056 |
| 3 | 482,281 | 69,7156 |
| 4 | 472,281 | 69,7158 |
| 5 | 492,281 | 70,7256 |
| promedio | 482,281 | 69,7156 |

**Tabla de datos para tamaño 3500x3500 (con 12 hilos)**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz 3500x3500 | Secuencial (promedio) | Paralelizado(promedio) |
| 1 | 758,948 | 112,914 |
| 2 | 759,948 | 110,915 |
| 3 | 760,948 | 114,914 |
| 4 | 769,948 | 112,913 |
| 5 | 749,948 | 102,914 |
| promedio | 759,948 | 122,914 |

**Tabla de datos para tamaño 4000x4000 (con 12 hilos)**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz 4000x4000 | Secuencial (promedio) | Paralelizado(promedio) |
| 1 | 1347,83 | 174 |
| 2 | 1147,85 | 173 |
| 3 | 1247,84 | 175 |
| 4 | 1248,94 | 174 |
| 5 | 1246,74 | 174 |
| promedio | 1247,84 | 174 |

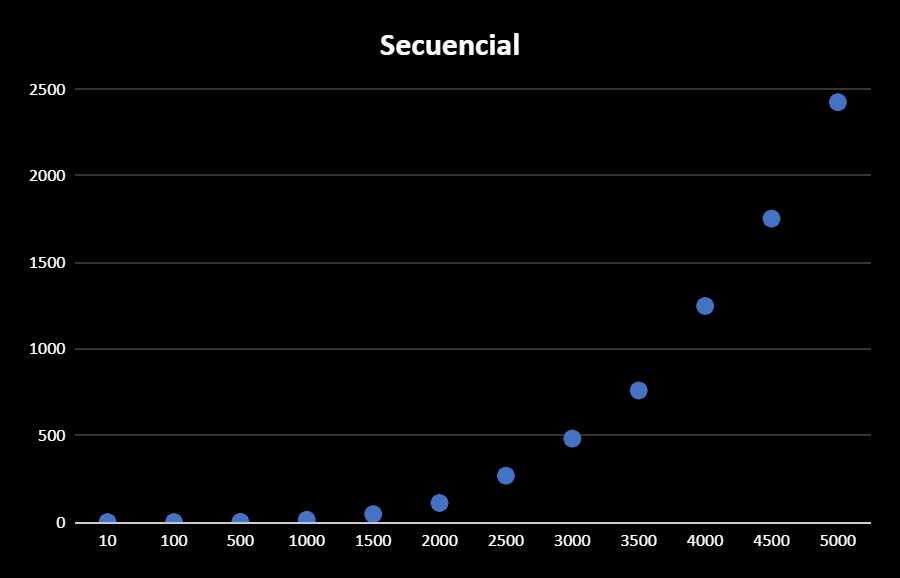
**Tabla de datos para tamaño 4500x4500 (con 12 hilos)**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz 4500x4500 | Secuencial (promedio) | Paralelizado(promedio) |
| 1 | 1762,3 | 236 |
| 2 | 1752,5 | 255 |
| 3 | 1742,5 | 255 |
| 4 | 1852,5 | 254 |
| 5 | 1652,7 | 265 |
| promedio | 1752,5 | 255 |

**Tabla de datos para tamaño 5000x5000 (con 12 hilos)**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz 5000x5000 | Secuencial (promedio) | Paralelizado(promedio) |
| 1 | 2475,03 | 356,793 |
| 2 | 2385,03 | 354,783 |
| 3 | 2425,03 | 354,803 |
| 4 | 2426,03 | 354,788 |
| 5 | 2424,03 | 354,793 |
| promedio | 2425,03 | 353,798 |

**GRÁFICA SECUENCIAL**

****

Esta gráfica hace referencia a la programación de la matriz de forma secuencial (sin hilos) donde el eje X es el tamaño de las matrices que se usaron para elaborar el reto ( 10x10 hasta 5000x5000), el eje Y hace referencia al tiempo (s) en que tarda el programa al ejecutar el código de la matriz secuencial.

Podemos observar que cada vez que se amplía el tamaño de la matriz al computador le cuesta más a nivel de procesamiento debido a que la complejidad estándar del código de las matrices secuenciales es de la forma O(n³) en tiempo siendo así un programa ineficiente que abarca todos los recursos del PC.

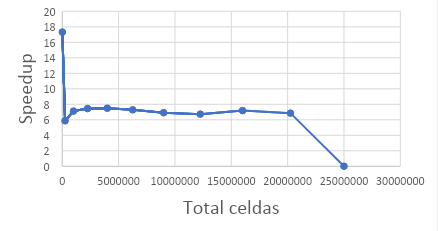
**GRÁFICA PARALELIZADA**

****

Esta gráfica hace referencia a la programación de la matriz de forma paralela (con hilos) donde el eje X es el tamaño de las matrices que se usaron para elaborar el reto ( 10x10 hasta 5000x5000),y el eje Y hace referencia al tiempo (s) en que tarda el programa al ejecutar el código de la matriz secuencial.

Podemos observar que cada vez que se va ampliando el tamaño de la matriz al computador le tarda un poco más de segundos en arrojar el resultado pero también vemos que es mínimo, a una matriz paralelizada de 5000x5000 le tarda 354 segundos aproximadamente mientras que a la ecuencial le tomaría alrededor de 2500 segundos.

**GRÁFICA SPEED-UP**

****

**CONCLUSIONES**

* En programación secuencial, la forma de ejecutar sería de forma secuencial (paso a paso) y hasta que no se tenga toda la información no se puede procesar.En el caso de usar hilos, se pueden lanzar tareas en paralelo provocando una mejora del rendimiento de la máquina.
* El uso e implementación de hilos en un programa de matrices provoca que la máquina que esté ejecutando dicho programa pueda hacer mejor uso de los recursos físicos (procesador y memoria).
* La implementación de muchos hilos para aplicaciones excesivamente complejas pues provocará excesiva sobrecarga.
* La programación con hilos no indica que siempre se va a tener una mejora de rendimiento y la decisión de usar hilos o no depende el programador y de las necesidades que se tengan.
* La implementación de concurrencia no es fácil ya que hay muchos factores que son afectados como el resultado, la cantidad de recursos consumidos, el código secuencial, el tiempo de ejecución… etc, por esto hay que ser conscientes en la forma de trabajo de la concurrencia ya que si no se controlan estos aspectos los resultados pueden verse fatalmente afectados con respecto al funcionamiento secuencial y la salida esperada, no todo se puede paralelizar.
* La granularidad juega un papel esencial en la concurrencia ya que esta debe estar bien equilibrada para un óptimo desempeño, una granularidad alta puede generar la creación de muchos hilos que pueden entorpecer el proceso con los cambios de contexto y la saturación de hilos en el sistema operativo y una granularidad baja implica la falta de aprovechamiento de los recursos de hardware para un buen tiempo de ejecución.