Análisis de datos reto 2 HPC

Presentado por:

Cristian Camilo Perilla Castaño

Daniel Villada Moncada

Luis Miguel Ortiz Alarcón

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 de datos de tiempo obtenidos gracias a la cronometración de códigos desarrollados en c++ de multiplicación de matrices, un mismo programa que ha sido programado de tres formas diferentes, la primera de forma tradicional secuencial, la segunda utilizando concurrencia por hilos con la librería pthreads y la tercera utilizando concurrencia por procesos con fork.

**CARACTERÍSTICAS DEL COMPUTADOR**

|  |  |
| --- | --- |
| **Procesador** | AMD Ryzen R5 2600 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** | 6 núcleos 12 hilos |

**DESCRIPCIÓN DEL PROCESO**

Basados en el primer reto, en este vamos a reutilizar los datos obtenidos y a adicionar unos nuevos datos generados por el código de multiplicación de matrices pero esta vez utilizando concurrencia por procesos, vamos a analizar la información resultante mediante tablas y gráficos para luego generar conclusiones.

**DESCRIPCIÓN DE LAS GRÁFICAS**

**Tiempos en secuencial**

|  |  |  |
| --- | --- | --- |
| Tamaño Matriz | Secuencial | total celdas |
| 10 | 0 | 100 |
| 100 | 0,15625 | 10000 |
| 500 | 1,65625 | 250000 |
| 1000 | 13,4219 | 1000000 |
| 1500 | 46,0781 | 2250000 |
| 2000 | 110,281 | 4000000 |
| 2500 | 267,906 | 6250000 |
| 3000 | 482,281 | 9000000 |
| 3500 | 759,948 | 12250000 |

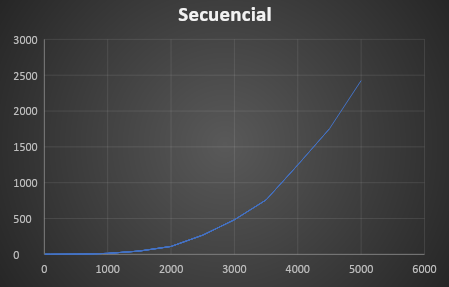
**Tiempo en hilos**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tamaño Matriz | 12 nucleos | 8 núcleos | 6 núcleos | 4 núcleos | 2 núcleos | total celdas |
| 10 | 0 | 0,0009085 | 0,0008763 | 0,0007983 | 0,0010546 | 100 |
| 100 | 0,0083896 | 0,00918 | 0,0083746 | 0,0120884 | 0,0137907 | 10000 |
| 500 | 0,253582 | 0,391152 | 0,326139 | 0,447387 | 0,862703 | 250000 |
| 1000 | 1,77788 | 2,57016 | 2,32297 | 3,33017 | 6,67776 | 1000000 |
| 1500 | 5,86188 | 9,34809 | 7,36836 | 10,9693 | 22,4652 | 2250000 |
| 2000 | 13,6723 | 20,6229 | 17,4695 | 25,9618 | 52,6603 | 4000000 |
| 2500 | 34,1269 | 49,7031 | 37,2496 | 55,7791 | 113,295 | 6250000 |
| 3000 | 63,9952 | 93,6313 | 69,2337 | 103,207 | 210,075 | 9000000 |
| 3500 | 105,309 | 153,285 | 110,955 | 167,336 | 380,119 | 12250000 |

**Tiempo en procesos**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PROCESOS |  |  |  |  |  |  |
| Tamaño Matriz | 12 nucleos | 8 núcleos | 6 núcleos | 4 núcleos | 2 núcleos | total celdas |
| 10 | 0,033 | 0,032 | 0,03 | 0,032 | 0,032 | 100 |
| 100 | 0,375 | 0,368 | 0,36 | 0,349 | 0,411 | 10000 |
| 500 | 3,122 | 3,253 | 2,963 | 2,979 | 3,933 | 250000 |
| 1000 | 10,148 | 10,304 | 9,16 | 9,372 | 16,31 | 1000000 |
| 1500 | 32,817 | 36,281 | 31,889 | 41,354 | 79,859 | 2250000 |
| 2000 | 61,119 | 74,411 | 63,114 | 91,297 | 184,38 | 4000000 |
| 2500 | 102,14 | 133,902 | 112,831 | 162,569 | 328,736 | 6250000 |
| 3000 | 161,511 | 225,045 | 196,825 | 286,565 | 575,236 | 9000000 |
| 3500 | 256,89 | 374,357 | 307,493 | 449,245 | 891,284 | 12250000 |

**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 HILOS**

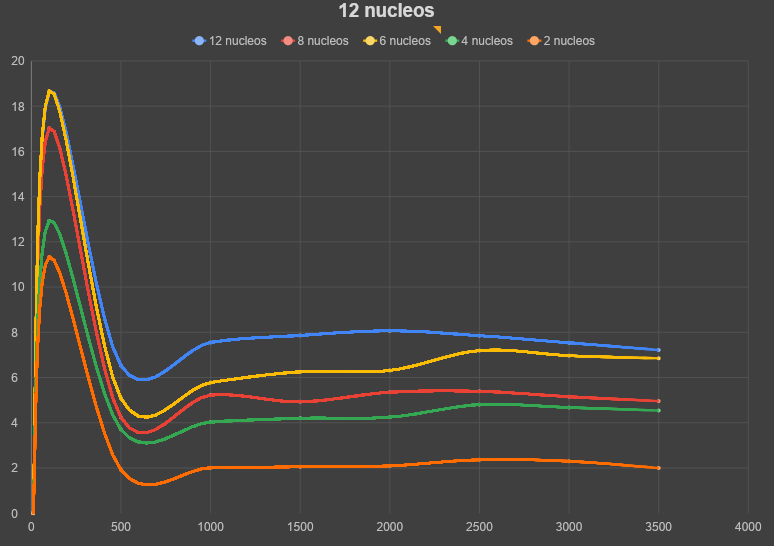
****

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 3500x3500 con dos núcleos le tarda 380 segundos y con 12 núcleos 105 segundos aproximadamente mientras que a la secuencial le tomaría alrededor de 780 segundos.

**GRÁFICAS SPEED-UP**

**GRÁFICA SPEED-UP HILOS**

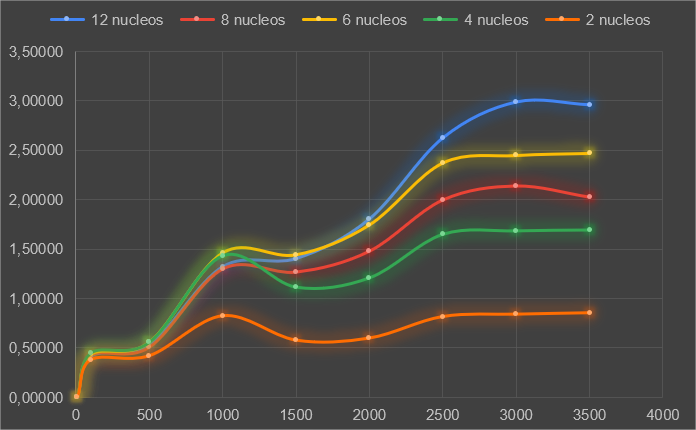
****

**El eje X de la gráfica hace referencia al número de tareas, mientras que el eje Y hace referencia al speed up rate.**

**TABLA SPEED UP PROCESOS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SPEED UP RATE |  |  |  |  |  |
|  | 12 nucleos | 8 núcleos | 6 núcleos | 4 núcleos | 2 núcleos |
| 10 | 0 | 0 | 0 | 0 | 0 |
| 100 | 18,62424907 | 17,02069717 | 18,65760753 | 12,92561464 | 11,33009927 |
| 500 | 6,531417845 | 4,234287438 | 5,078356161 | 3,702052138 | 1,919837998 |
| 1000 | 7,54938466 | 5,222204065 | 5,77790501 | 4,030394845 | 2,009940459 |
| 1500 | 7,860635155 | 4,929145954 | 6,253508243 | 4,200641791 | 2,051087905 |
| 2000 | 8,066016691 | 5,347502049 | 6,312773691 | 4,247817948 | 2,094196197 |
| 2500 | 7,850288189 | 5,390126572 | 7,192184614 | 4,802981762 | 2,364676288 |
| 3000 | 7,53620584 | 5,150852332 | 6,965986218 | 4,67294854 | 2,295756278 |
| 3500 | 7,216363274 | 4,957745376 | 6,849155063 | 4,541449539 | 1,999237081 |

**GRÁFICA SPEED UP PROCESOS**

****

**El eje X de la gráfica hace referencia al número de tareas, mientras que el eje Y hace referencia al speed up rate.**

En esta gráfica podemos ver un comportamiento intermedio con respecto a las gráficas anteriores, esto sucede ya que la implementación de procesos no fue tan eficiente como con hilos pero si fue mejor que la secuencial.

**CONCLUSIONES**

* La programación secuencial para códigos que pueden ser concurrentes definitivamente es una pérdida de uso de máquina ya que podemos evidenciar mejoras de hasta x18 veces más rápido en el caso de la multiplicación de matrices con nuestra máquina y con la implementación lograda.
* Con concurrencia, a nivel externo de la máquina, se evidencia la generación de calor y la activación a tope de la refrigeración, sobre todo en el cálculo de matrices muy grandes como el caso de 3500x3500 donde existen 12.250.000 celdas a tratar.
* 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 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.
* La implementación más eficaz que fuimos capaz de lograr hasta ahora es la de hilos, la utilización de fork para procesos fue más compleja y para nosotros menos efectiva.
* Nosotros en los programas que hicimos no restringimos en uso de núcleos con código si no que nos aprovechamos de una bondad de AMD Ryzen llamada Chiplets, en concreto, en la microarquitectura de este procesador llamada Zen+. En esta encontramos tres chips de silicio que conforman el procesador como tal que encontramos bajo el IHS, se dividen en el controlador de entradas y salidas y dos chips de silicio más, cada uno de estos son un conjunto de 4 núcleos y 8 hilos dando un total por procesador con esta arquitectura de 8 núcleos y 16 hilos, todos estos interconectados por medio de infinity fabric, sin embargo, en la versión de procesador con la que se hicieron las pruebas cuenta con 6 núcleos y 12 hilos, esto porque de fábrica es posible que los conjuntos de núcleos (chiplets) vinieran defectuosos y simplemente en la interconexión no se conectan dichos núcleos defectuosos logrando así varios modelos que vienen desde los 4 núcleos 4 hilos hasta los 8 nucleos 16 hilos, sabiendo esto, la arquitectura también permite deshabilitar estos conjuntos de núcleos a nivel de software desde la bios, en nuestro caso podemos tener activados 2, 4 y 6 núcleos, aparte de esto también tenemos la opción de habilitar otra característica llamada SMT o Simultaneous Multithreading.

En las pruebas efectuadas con anterioridad, utilizamos 2, 4 y 6 núcleos sin SMT, 4 núcleos con SMT para un total de 8 núcleos y 6 núcleos con SMT para un total de 12 núcleos.

Se pudieron evidenciar resultados interesantes como por ejemplo, sabemos que el Hyperthreading (intel) o Simultaneous Multithreading (amd) puede aumentar hasta en un 30% el rendimiento de un procesador con respecto a su misma versión pero sin esta característica, podemos decir entonces que dos núcleos físicos o “reales” son más rápidos que un núcleo con dos “núcleos lógicos” y tiene sentido ya que estos nucleos logicos solo es agregan ciertas características como la memoria caché de primer nivel y registros independientes, pero que siguen compartiendo más componentes con el otro núcleo lógico engañando de esta forma al sistema operativo haciéndole creer que hay dos núcleos completos.

En las pruebas hallamos que 6 nucleos fisicos rinden más que 8 núcleos (que en realidad serian 4 nucleos fisicos u 8 lógicos), como mencionamos anteriormente, estos 8 nucleos logicos en promedio son 30% más eficientes que 4 nucleos fisicos pero 6 nucleos fisicos son un 50% más eficientes que estos mismos 4 nucleos fisicos.