ACTIVIDAD 03:

CASO DE ESTUDIO MULTIPLICACIÓN

DE MATRICES: THREADS

PRESENTADO POR:

ANDRÉS DAVID GUEVARA JARAMILLO

JHONATAN LATORRE SIERRA

PRESENTADO A:

RAMIRO ANDRÉS BARRIOS

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

INGENIERÍA EN SISTEMAS Y COMPUTACIÓN

HIGH PERFORMANCE COMPUTING

PEREIRA, RISARALDA

2021 – 2

Universidad Tecnológica de Pereira

Resumen:

El siguiente trabajo tiene como objetivo hacer un completo análisis de la efectividad y diferencias denotadas de los modelos o infraestructuras de programación secuencial y paralela. Se tomara como ejemplo la multiplicación de matrices, elaborando códigos en C utilizando los modelos de programación anteriormente mencionados (secuenciales y paralelas), buscando de esta forma, demostrar cuál de los 2 modelos es más eficiente a la hora de ejecutar programas que exijan requisitos de cómputo elevados.

Universidad Tecnológica de Pereira

Introducción:

El presente trabajo buscará ahondar en las diferencias de los modelos de programación secuencial y paralelo, con el objetivo de determinar cuál modelo es más eficiente, se llegará a esto con los siguientes procedimientos, se diseñarán códigos en C para resolver la multiplicación de matrices nxn que serán llenadas aleatoriamente, utilizando los 2 modelos de programación. Se implementará de manera secuencial utilizando los ciclos for, tanto para llenar las matrices con la ayuda de la función Rand () Perteneciente a la librería stdlib, como para realizar la multiplicación de las matrices. Se implementará de manera paralela utilizando la librería phthread de C, la cual nos permitirá utilizar la técnica de paralelismo de forma correcta, todo esto se aplicará para los casos de 2, 4 y 8 hilos en la ejecución del programa.

Una vez hecho lo anterior se medirá el tiempo de ejecución de todos los códigos con su diferente implementación (secuencial, 2 hilos, 4 hilos, 8 hilos), estos datos se medirán en múltiples ocasiones entre 4 y 5 veces, y se irá aumentando el número n de las matrices estratégicamente, para ver como los procesos se volverán más grandes en tiempos de ejecución. Se debe tener en cuenta que todos estos intentos se harán en el mismo sistema operativo, entorno de desarrollo y lenguaje de programación para garantizar que ningún modelo tenga ventaja sobre el otro.

Una vez realizado lo anterior, para todas las implementaciones, se calculará el promedio respecto al número n de matrices, sumando el tiempo de cada intento y dividiéndolo entre el número de intentos realizados y el speedup que será hallado con una fórmula que posteriormente se explicará. Todo esto con el objetivo de poder medir exactamente los resultados y posteriormente compararlos por medio de gráficas y así, poder sacar conclusiones verdaderamente efectivas, para culminar el análisis y determinar cuál modelo de programación es más eficiente.

Marco conceptual:

Multiplicación de matrices: Según Wikipedia, la multiplicación o producto de matrices es la operación de composición efectuada entre dos matrices. Al igual que la multiplicación aritmética, viene dada por un algoritmo capaz de efectuarla. El algoritmo para la multiplicación matricial es diferente del que resuelve la multiplicación de dos números. La diferencia principal es que la multiplicación de matrices no cumple con la propiedad de conmutatividad.

Dadas dos matrices *A* y *B*, tales que el número de columnas de la matriz *A* es igual al número de filas de la matriz *B*; es decir:

$$A := (a_{ij})_{m\times n} \text{ y } B := (b_{ij})_{n\times p}$$

La multiplicación de *A* por *B*, que se denota $A \cdot B,\ A \times B,\ A \circ B$ o simplemente AB, el resultado del producto es una nueva matriz *C*:

$$C = AB := (c_{ij})_{m\times p}$$

Es decir:

$$C = AB = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{np} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

Estructura secuencial:

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. (Urbaez, Wilder, 2004)

Paralelismo

El paralelismo es un caso específico de concurrencia que se puede aplicar de tres diferentes formas dependiendo del tipo de programa en el que se va a usar, estas formas son:

Paralelismo independiente: Podría considerarse la forma más simple de paralelismo y consiste en que una o más operaciones son aplicadas independientemente a cada ítem en un grupo de datos. (Sutter, H., & Larus, J. 2005).

Paralelismo Regular: Se trata de aplicar paralelismo a operaciones con datos donde sus procesos son mutuamente dependientes.

Paralelismo no estructurado: Es cuando las computaciones concurrentes difieren, significando que el acceso a los datos no es predecible y necesitan ser coordinados a través de sincronización explícita. Esta forma de paralelismo es la más común en programas escritos usando hilos.

Concurrencia:

Concurrencia es la capacidad de diferentes partes o unidades de un programa, algoritmo o problema de ejecutarse fuera de orden o en orden parcial, sin afectar el resultado final. Esto permite la ejecución paralela de las unidades concurrentes, lo que puede mejorar significativamente la velocidad general de ejecución en sistemas multiprocesador y multinúcleo. En términos más técnicos, la concurrencia se refiere a la propiedad de descomponibilidad de un programa, algoritmo o problema en componentes o unidades independientes del orden o parcialmente ordenados. Lamport, Leslie (julio de 1978). "Hora, relojes y ordenación de eventos en un sistema distribuido"

Complejidad computacional:

Estudia el orden de complejidad de un algoritmo que resuelve un problema decidible. Para ello, considera los 2 tipos de recursos requeridos durante el cómputo para resolver un problema:

- **Tiempo:** Número de pasos base de ejecución de un algoritmo para resolver un problema.
- **Espacio**: Cantidad de memoria utilizada para resolver un problema.

Marco contextual:

- **Características de la infraestructura física (HW) utilizada:**
  - AMD A8-6410 APU with AMD
  - Radeon R5 Graphics 2.00 GHz
  - 4,00 GB RAM instalada
  - Sistema operativo de 64 bits, procesador basado en x64
  - Núcleos: 4, Procesadores lógicos: 4

Desarrollo:

Para el desarrollo del ejercicio implementamos un código en c para primero realizar la resolución de matrices NxN de manera secuencial y después por hilos (threads), en ambos casos realizamos ciclos que rellenan las dos matrices que se van a multiplicar con número aleatorios entre el 0 y 9 con srand().

```
//Creación de hilos
for (rank = 0; rank < num_hilos; rank++)
    pthread_create(&tid[rank], NULL,matmul , (void*) rank);

//Unión de hilos
for (rank = 0; rank < num_hilos; rank++)
    pthread_join(tid[rank], NULL);
```

matmul: MatrixMultiplication

matmul -- SECUENCIAL

matmul -- PARALELA

```
void matmul(int col, int fil){
  int i,j,k;

  for(i=0;i<fil;i++){
   for(j=0;j<col;j++){
    C[i][j]=0;
    for(k=0;k<col;k++){
      C[i][j] += A[i][k]*B[k][j];
    }
   }
  }

}
```

```
// Función que va ejecutarse en cada hilo
void *matmul(void* id_arg){
  int i,j,k;
  long  id = (long ) id_arg;
  int filas_por_hilo = col/num_hilos;
  int start_index = id*filas_por_hilo;
  int final_index = (id+1)*filas_por_hilo;

  for(i=start_index;i<final_index;i++){
   for(j=0;j<col;j++){
    for(k=0;k<fil;k++){
      C[i][j] += A[i][k]*B[k][j];
    }
   }
  }
}
```

Pruebas:

Las pruebas que se realizaron fueron con respecto al software ya antes mencionado de multiplicación de matrices en un computador con las especificaciones ya antes mencionadas también. Realizamos pruebas del software secuencial, y con hilos (2, 4, 8 y N hilos) y los resultados se compilaron en las tablas mostradas a continuación:

Tabla de multiplicación secuencial:

Secuencial	Dimensión NxN							
Tiempo/Dimensión	16	200	400	600	800	1000	1200	1400
Tiempo(ms)	30,24	8153,89	34945,91	106104,03	251033,96	418796,01	628125,81	761395,52
	31,12	8148,72	34487,81	95323,68	276659,16	402263,60	624565,41	785370,83
	28,9	9147,18	34234,43	92455,2	255268,88	406727,69	637884,91	750374,86
	29	8603,22	33844,57	98944,59	265222,25	397194,11	639951,14	764093,08
	28,08	8719,87	33488,83	120497,16	267339,71	391159,94	644830,69	767982,76
Promedio(ms)	29,42	8554,58	34200,31	102664,93	263104,79	403228,27	635071,59	765843,41
Promedio(s)	0,029	8,555	34,200	102,665	263,105	403,228	635,072	765,843
SpeedUp	1	1	1	1	1	1	1	1

Tabla de multiplicación paralela con 2 hilos:

2 Hilos	Dimensión NxN							
Tiempo/Dimensión	16	200	400	600	800	1000	1200	1400
Tiempo(ms)	48,88	5811,84	18207,19	38075,94	68082,58	103871,24	141204,27	187630,34
	51,00	5762,23	17978,14	34482,49	74488,88	100197,37	140492,19	192731,47
	46,90	5629,51	17851,45	33526,33	69141,31	101189,39	143156,09	185285,52
	46,93	6052,87	17656,52	35689,46	71629,65	99070,81667	143569,34	188204,29
	50,00	5892,24	17478,65	42873,65	72159,02	97729,89167	144545,25	189031,88
Promedio(ms)	48,742	5829,737	17834,390	36929,574	71100,288	100411,742	142593,427	188576,700
Promedio(s)	0,049	5,830	17,834	36,930	71,100	100,412	142,593	188,577
SpeedUp	0,60349296	1,4674032	1,91766077	2,7800194	3,700474351	4,015748167	4,453722785	4,06117728

Tabla de multiplicación paralela con 4 hilos:

Universidad Tecnológica de

4 Hilos	Dimensión NxN							
Tiempo/Dimensión	16	200	400	600	800	1000	1200	1400
Tiempo(ms)	46,9	5793,98	23620,14	43499,47	69331,29	102574,61	145046,65	185866,41
	31,25	5956,06	23089,12	42130,17	71133,81	100411,52	143495,52	192790,53
	37,83	5946,89	22882,98	42364,08	70351,52	101180,66	143830.97	188693,49
	46,88	5848,31	23349,81	41529,18	71292,44	100994,98	142944,39	191943,89
	31,24	5885,21	22981,23	42961,49	72802,55	99298,01	144393,26	190357,43
Promedio(ms)	38,820	5886,090	23184,656	42496,878	70982,322	100891,955	143969,955	189930,350
Promedio(s)	0,039	5,886	23,185	42,497	70,982	100,892	143,970	189,930
SpeedUp	0,75773656	1,45335443	1,47512691	2,4158229	3,706624229	3,996634494	4,411139764	4,03223292

Tabla de multiplicación paralela con 8 hilos:

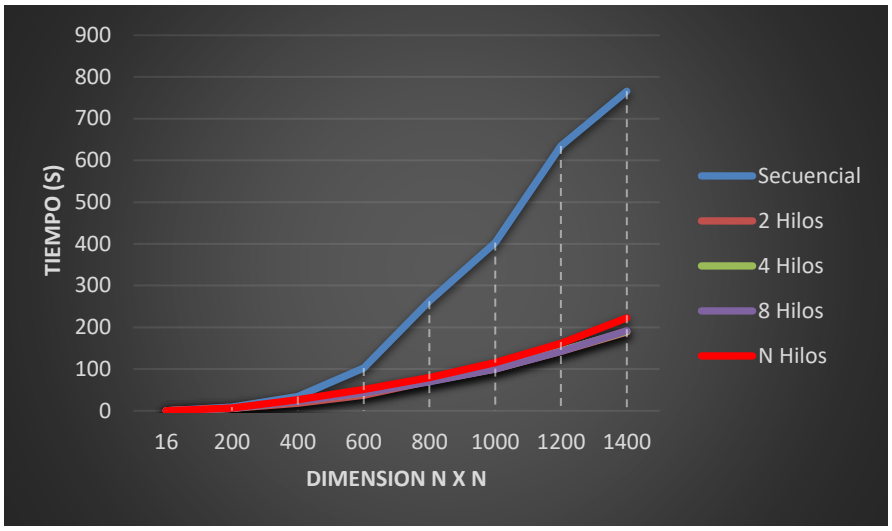
8 Hilos	Dimensión NxN							
Tiempo/Dimensión	16	200	400	600	800	1000	1200	1400
Tiempo(ms)	31,24	5770,04	22899,33	42585,46	67691,07	100506,06	141097,51	186237,9
	46,86	6017,65	22498,62	43232.44	69043,09	100277,73	143080.75	195738,26
	31,29	5939,22	22688,5	42586,46	69332,24	99487,52	141098,51	190479,22
	31,31	5885,96	22484,65	43232.45	70329,97	99860,02	144098,51	195059,78
	31,24	5970,55	22379,24	42587,46	70150,555	100873,56	143230,67	191180,44
Promedio(ms)	34,388	5916,684	22590,0683	42586,46	69309,385	100200,978	142381,3	191739,12
Promedio(s)	0,0344	5,9167	22,5901	42,5865	69,3094	100,2010	142,3813	191,7391
SpeedUp	0,85539529	1,44583943	1,51395337	2,41074116	3,796091876	4,024194929	4,46035816	3,99419487

Tabla de multiplicación paralela con N hilos:

N Hilos	Dimensión NxN							
Tiempo/Dimensión	16	200	400	600	800	1000	1200	1400
Tiempo(ms)	100,28	6797,46	25870,47	50184,1	79385,2	113494,13	157409,99	222286,55
	100,28	6658,13	25740,01	55131,31	79144,95	117269,85	164362,9	232550,75
	122,17	6842,76	27178,55	47936,02	78904,7	114054,42	162213,85	218473,4
	116,23	6511,42	27571,09	48835,73	81664,45	115499,76	156132,77	220623,75
	115,92	6634,07	26225,13	51711,69	78424,2	116779,90	165534,70	218717,18
Promedio(ms)	110,976	6688,76667	26517,05	50759,77	79504,7	115419,6117	161130,8433	222530,325
Promedio(s)	0,111	6,689	26,517	50,760	79,505	115,420	161,131	222,530
SpeedUp	0,26506031	1,27894654	1,28974792	2,02256496	3,309298612	3,493585377	3,941340964	3,4415238

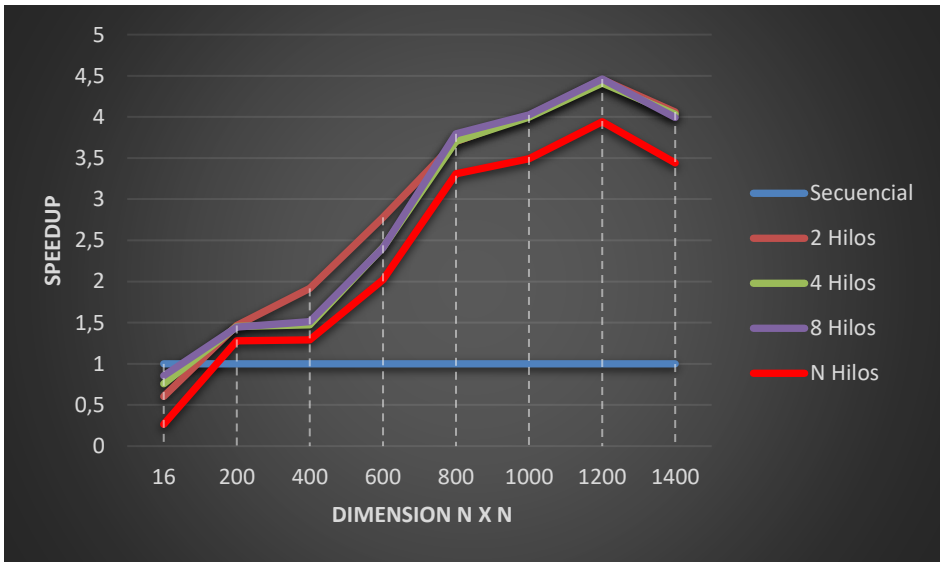
Para ilustrar los resultados de una manera más visual, hemos realizado un par de gráficas comparando el desarrollo de la secuencial contra el desarrollo de la paralela con cada una de las situaciones en cantidad de hilos implementados, y por otra parte realizamos también una gráfica en la que se puede apreciar el SpeedUp de todos los casos presentes.

Gráfica de rendimiento secuencial vs paralelo (En tiempo y dimensión de la matriz):



Como podemos ver de una manera mucho más clara en la gráfica que en las tablas, al relacionar el tiempo de ejecución con las diferentes dimensiones que le hemos dado a N para las matrices, podemos ver como la línea de tendencia de la secuencial está muy por encima de las otras líneas y esta tendencia va aumentando conforme se va aumentando la dimensión de la matriz, llegan incluso a duplicar o triplicar el tiempo con respecto a las otras líneas y por otro lado vemos que entre las líneas que están más bajas se encuentran las de 4 y 8 hilos, indicándonos que estos son los códigos más eficientes, pero que no hay mucha mejora tampoco en la de 8 hilos con respecto a la de 4, están casi a la par, por lo que podemos deducir que no siempre más hilos significa más eficiencia y esto lo podemos confirmar al observar la línea de N hilos, en donde se está asignando un hilo por cada fila de la matriz por lo que podemos llegar a tener muchísimos hilos y esto por supuesto que en la vida real no sería práctico debido a sus altos costos y no tan buen rendimiento.

Gráfica de SpeedUp secuencial vs paralelo (En factor de SpeedUp y dimensión de la matriz):



En esta gráfica podemos observar que el SpeedUp crece en un principio de manera muy rápida, sobre todo en el tramo de N entre 400 y 800, para después alcanzar un pico de alrededor de 4,5 con 8 hilos, pero vale la pena aclarar que con 2 hilos y 4 hilos también están muy cerca de este valor y tal y como vimos en la gráfica pasada también N hilos no tiene tanta eficiencia como se podría pensar erradamente que entre más hilos tenga, más rápido será, como podemos ver esto no se cumple.

Conclusiones:

Hay varias conclusiones que podemos sacar de esta práctica:

- Paralelizando podemos aumentar la eficiencia de un código de manera considerable.
- No siempre más hilos significan más efectividad.
- El dividir las funciones que el código tiene que realizar, para que el procesador pueda realizar varias simultáneamente es la clave para aumentar la eficiencia de este.
- No es solo importante tener un código limpio y organizado, también tenemos que entrar a evaluar que tan eficiente es nuestro código en varios aspectos, relación calidad/precio, calidad/rendimiento, etc.

¿Cómo se sintieron?

Ninguno de los dos teníamos experiencia trabajando con hilos por lo que en un principio fue un poco más complicado entender el funcionamiento de estos y como implementarlos en nuestro código, pero después de comprender más el tema pudimos desarrollarnos bien como equipo de trabajo.

¿Qué dificultades tuvieron?

Una de las dificultades fue cuadrar los tiempos para realizarlo ya que nuestros tiempos no coincidían mucho y en cuanto a la realización como tal, una de las principales dificultades fue entender el papel de los punteros en este tema y el realizar las pruebas ya que tomaban mucho tiempo, pero el script para esto explicado en clase nos ahorró mucho trabajo.

¿El tiempo fue suficiente para hacer la actividad?

Consideramos que el tiempo para hacer la actividad fue prudente, no es mucho tiempo pero se puede desarrollar la actividad sin muchos percances.

Compartir su percepción sobre el resultado del trabajo grupal

El trabajo en grupo estuvo bien coordinado a pesar de no coincidir en tiempos, ya que cuando uno estaba indagando sobre el código, el otro aprovechaba para adelantar el documento, después uno trabajaba en las pruebas y el otro en el documento y así pudimos concretar una buena pareja de trabajo.

Referencias:

gettimeofday() samples: https://cpp.hotexamples.com/es/examples/-/-/gettimeofday/cpp-gettimeofday-function-examples.html

Threads in c and c++: https://www.geeksforgeeks.org/multiplication-of-matrix-using-threads/

Multithread and reference code: http://stg-pepper.blogspot.com/2017/07/programacion-multithread-en-c-pthreadh.html