Trabajo Nro 2 Sistemas Paralelos

Ivan Mindlin William Gaggiotti

1- Algoritmo secuencial
No tenemos mucho para detallar de este algoritmo, definimos 3 funciones para poder calcular el mínimo, máximo y promedio respectivamente con cada función. Definimos 5 matrices, A,B,C,D,resultado_intermedio, la matriz A se inició con valores random y se almacenó orientado a filas, B y C se iniciaron como matriz identidad para así poder verificar el resultado final, también se almacenaron orientado a filas ya que de igual manera sigue siendo matriz identidad. Por ultimo la matriz D y resul_intermedio se iniciaron en 0 y se almacenaron orientadas a filas.

Primero calculamos la constante.

$$\left(\frac{maxA \cdot maxB \cdot maxC - minA \cdot minB \cdot minC}{avgA \cdot avgB \cdot avgC}\right)$$

Luego realizamos la multiplicación de A con B y almacenamos en resul_intermedio.
Luego realizamos la multiplicación de resul_intermedio con C y almacenamos en D.
Finalmente realizamos la multiplicación entre D y la constante.

La comparación la hacemos entre D y A*cons, ya que al multiplicar A con B y C que son al matriz identidad el resultado será A, para comparar con D solo hace falta multiplicar A*cons.

2-
Para este ejercicio sí pudimos aprovechar los algoritmos de paralelización. El problema fue dividido en dos partes.
Primero se realiza el cálculo de la constante, y después el cálculo de la matriz. Para el cálculo de la constante usamos un algoritmo similar al de las prácticas. Al ejecutar la función que realiza el cómputo se hace una división del vector, según el id del hilo. Al ser una matriz en lugar de un vector, se divide en subsecciones. En este caso, como todas las matrices tienen las mismas dimensiones y sólo se utilizan para lectura, usamos la misma iteración para calcular mínimo, máximo y suma de los elementos de la subsección para matriz A, B y C. Usamos variables locales para que puedan trabajar de forma independiente. Luego con mutex y semáforos se accede a las variables globales que mantienen los valores generales para cada Matriz. Luego de calcular estos valores se finalizan los trabajos de los hilos, y se calcula el valor de la constante con la que multiplicaremos a la matriz.
Para finalizar se hace la multiplicación de las matrices. Para eso creamos nuevamente hilos para repartir la carga de trabajo. Cada hilo utiliza su ID para determinar dónde comienza y termina su subsección de la matriz para operar. Primero se realiza la multiplicación, en la sección correspondiente a cada hilo, de a la matriz A por B. Y luego éste resultado se multiplica por C. Por cada elemento de la matriz D que se termina de computar, se lo multiplica por la constante calculada anteriormente

3- ACLARACION: Este punto en concreto lo realizamos de 2 maneras distintas, la primera manera esta explicada en la parte A y la segunda en la parte B. El por que de esto fue el siguiente, primero realizamos el algoritmo de la forma A, luego de una consulta con nuestro ayudante vimos que no era lo pedido en el enunciado y realizamos la B.

A- Para este punto se tomó el algoritmo desarrollado en 1 y se modificó para poder paralelizarlo. Primero se usó un parallel y dentro de este un for para el seteo de las matrices, seguido de esto se uso un section para realizar el cálculo de los máximos, mínimos y promedios. Dentro de las funciones que realizan dichas operaciones recién mencionadas, se uso un parallel for para paralelizar las búsquedas, donde se decidió crear la mitad de hilos que en el programa principal, esto para evitar un overhead, si por cada hilo del main que llama a una de estas funciones creamos la misma cantidad de hilos que el programa principal tendríamos demasiados.
Inmediatamente después de la barrera implícita del section tenemos un single, para realizar el cálculo de:

$$\left(\frac{maxA \cdot maxB \cdot maxC - minA \cdot minB \cdot minC}{avgA \cdot avgB \cdot avgC}\right)$$

finalizado esto termina el parallel.
Para terminar nos falta el cálculo de la multiplicación de matrices. Usamos un anidamiento de parallel for para dicha operación, para el primer parallel for se crean la cantidad de hilos establecida por parámetros, y por cada uno de estos se crean 2 para los otros parallel for de abajo.

Terminado todo el procesamiento se muestran los tiempos de procesamiento y se verifica si los resultados son correctos.

B-En esta parte dejamos el esqueleto del A, pero cambiamos el manejo de threads que hacíamos, esto para que se crearán y respetaran exactamente los threads pasados por parámetro.
Creamos todos los hilos en el único parallel del algoritmo, y luego se van realizando los sections y for con esos hilos creados. Los parallel for de las funciones fueron retirados para no crear más hilos de los exigidos por parámetro.
Luego la lógica del algoritmo es la misma que en A.

Tiempos :

Secuencial

Tamaño	Tiempo
512	3.195 s
1024	25.962 s

2048	207.537 s
4096	1670.648 s

OpenMP

Tamaño↓/threads ->	2	4	8
512	1.596 s	0.797 s	0.403 s
1024	12.938 s	6.483 s	3.251 s
2048	103.046 s	51.645 s	25.973 s
4096	831.831 s	416.429 s	208.834 s

Pthreads

Tamaño↓/threads ->	2	4	8
512	1.589 s	0.794 s	0.425 s
1024	12.930 s	6.477 s	3.245 s
2048	103.071 s	51.656 s	25.852 s
4096	831.578 s	416.325 s	208.746 s

Comparativas :

SpeedUp Pthreads respecto a Secuencial

Tamaño/Threads	2	4	8
512	3.195/1.589=**2.010**	3.195/0.794=**4.023**	3.195/0.425=**7.517**
1024	25.962/12.930**=2.007**	25.962/6.477=**4.008**	25.962/3.245=**8.000**
2048	207.537/103.071=**2.013**	207.537/51.656=**4.017**	207.537/25.852=**8.027**

4096	1670.648/831.578=**2.009**	1670.648/416.325=**4.012**	1670.648/208.746=**8.003**

SpeedUp openMP respecto a Secuencial

Tamaño↓/threads ->	2	4	8
512	3.195 / 1.596 = **2.001**	3.195 / 0.797=**4.008**	3.195 /0.403=**7.928**
1024	25.962 / 12.938=**1.999**	25.962 / 6.483=**4.004**	25.962 / 3.251=**7.985**
2048	207.537 / 103.046=**2.014**	207.537 / 51.645=**4.018**	207.537 / 25.973=**7.994**
4096	1670.648/831.831=**2.008**	1670.648/416.429=**4.011**	1670.648 / 208.834=**7.999**

En la tablas podemos ver que se refleja el caso ideal que se busca cuando se paraleliza algún programa. Generalmente cuando uno agregá más unidades de procesamiento para realizar cómputo paralelo, se espera que el tiempo se vaya reduciendo a la mitad con cada unidad extra de procesamiento agregada, pero esto no siempre es posible, ya sea por que se requiera sincronización, o acceso exclusivo a ciertos recursos, etc, por lo tanto las cpu no siempre están trabajando sino que tendrán tiempo ocioso. El speedup nos permite saber cuánta mejora de tiempo tuvimos y compararlo con el costo empleado, y poder llegar a una conclusión de si fue óptimo o no.
En nuestro caso podemos ver que los tiempos se van reduciendo a la mitad en todos los casos, es decir con 2 hilos realizamos el problema el doble de rápido, con 4 hilos 4 veces más rápido y con 8 hilos 8 veces más rápido. Esto se debe a que el algoritmo posee un alto nivel de paralelismo y no hay prácticamente tiempo ocioso para los hilos, cosa que pasaría si tuviéramos muchas secciones críticas o sincronizaciones.