IMT2112: Algoritmos Paralelos en Computación Científica 02 de 10 de 2020

Informe Tarea 2

Alumno: Lucas Suárez *Profesor: Elwin Van't Wout*

1. Preliminar

Para generar los vectores y matrices (y usar dynamic memory allocation) se utilizó el código visto en la ayudantía. En cuanto a la convergencia del método, se observa que el algoritmo no siempre converge al valor absoluto más grande de los valores propios, lo cual es consistente con la teoría pues para que exista convergencia es necesario que el vector inicial y los valores propios de la matriz cumplan ciertas condiciones; y como ambos se generan (pseudo) aleatoriamente en el programa, en muchas ocasiones el algoritmo no converge. Con respecto a esto último, entre mayor es la dimensión se observó que la convergencia es cada vez más difícil de alcanzar.

2. Resultados aumento de dimensión

Primero observemos que ocurre al aumentar el tamaño de la matriz y mantener constante la cantidad de iteraciones (10000)

Dimension	Tiempo de cómputo (10 hilos)	Tiempo de cómputo (1 hilo)
10	4.539 s	2.328 s
20	4.710 s	2.508 s
40	5.390 s	3.219 s
80	4.644 s	3.828 s
160	5.296 s	9.400 s
250	8.027 s	11.669 s
500	19.563 s	35.170 s
750	49.521 s	104.594 s
1000	83.479 s	153.060 s

3. Resultados aumento de hilos

Ahora observemos que ocurre al aumentar el número de threads manteniendo la dimensión (1000) y la cantidad de iteraciones (10000) constante.

Threads	Tiempo de cómputo
1	161.409 s
2	122.305 s
4	86.052 s
10	74.213 s
20	80.012
40	106.469
50	116.970
75	133.506
100	153.942

4. Conclusiones

En cuanto a los resultados de la sección 2 puede apreciarse que la utilización de multithreading no siempre acelera el algoritmo. En efecto, para dimensiones bajas el costo por usar multithreading excede a la ganancia que otorga, y como hemos visto en el curso esto es porque en dimensiones más bajas hay menor cantidad de operaciones y por lo tanto menor posibilidad de paralelización. En el caso contrario, para dimensiones más altas, el uso de multithreading es altamente eficiente llegando a ejecutar el algoritmo en la mitad de tiempo que tomaría usar solo un hilo secuencialmente (caso de dimensión 1000).

Con respecto a los resultados de la sección 3 se observa que en dimensión alta (1000) a medida que incrementamos la cantidad de threads el tiempo de cómputo es cada vez menor, hasta llegar a un punto de quiebre (alrededor de 10 threads) donde incrementar el número de hilos empeora el rendimiento. Esto ocurre porque el costo de usar cada vez más hilos es cada vez mayor y no compensa la ganancia de la paralelización.

Como conclusión general, cuando se quiere utilizar multithreading en un algoritmo, debe analizarse correctamente si en primer lugar vale la pena usar multithreading, es decir, si es que hay una cantidad apropiada de operaciones que pueden paralelizarse (como vimos con el aumento de dimensión). En segundo lugar, debe escogerse de manera óptima la cantidad de hilos a utilizar, pues utilizar demasiados hilos empeorará el rendimiento en lugar de mejorarlo.