Pontificia Universidad Católica Madre y Maestra

Campus Santo Tomás de Aquino

Departamento de Ingeniería

ISC-307-T001

Resultados comparativos entre algoritmos secuenciales y paralelos para la multiplicación de matrices

Javier Alexander Falcón Pérez 2016-5265

Manuel Molina 2016-5468

Santo Domingo, 03/10/2018

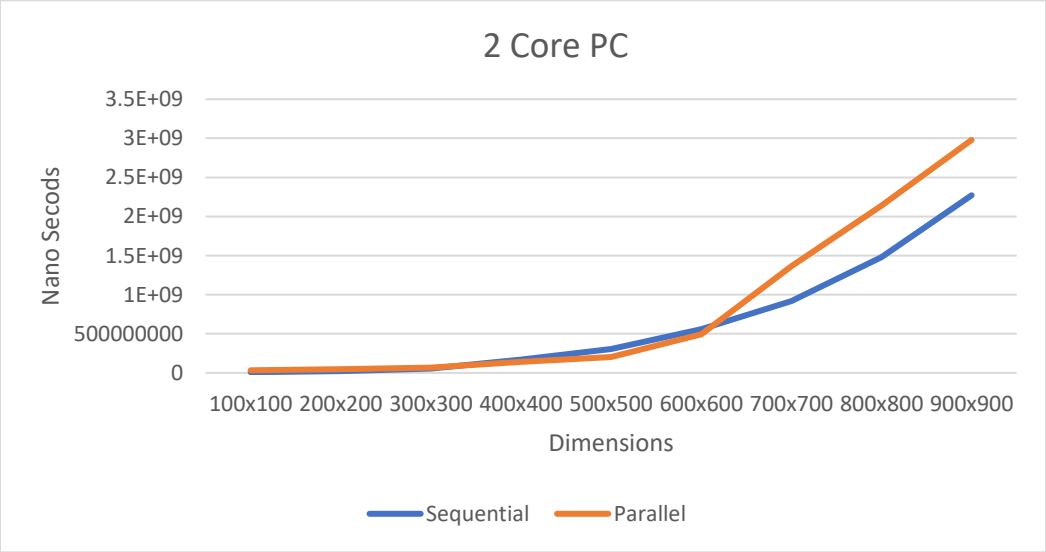
Resolución del problema

Para la concepción del algoritmo paralelo de multiplicación de matrices tomamos como guía los ejemplos de creación de hilos realizados en la clase. A diferencia con la implementación secuencial, el código de multiplicación paralelo de dos matrices debió separarse en partes. La clase **MultiplierThread** sería la encargada de contener el código que realiza la multiplicación de una fila de la matriz A con todas las columnas de la matriz B. Esto nos aseguraría que cada *thread* computara el resultado de únicamente una fila de la matriz resultante. Dicho esto, entendimos rápidamente que la función ***multiply()*** que se sobrescribe en la clase **ParallelMatrix** ya no sería la encargada de multiplicar, sino de inicializar cada hilo y ponerlo a correr para que cada uno computara lo establecido. Finalmente, fue necesario utilizar el método ***join()*** discutido en clase para indicarle a cada hilo que debe esperar por la terminación de los demás antes de devolver la matriz resultante. Visualizar esta versión del algoritmo no fue tan complicada como esperábamos, por lo que su codificación fue relativamente sencilla.

Resultados

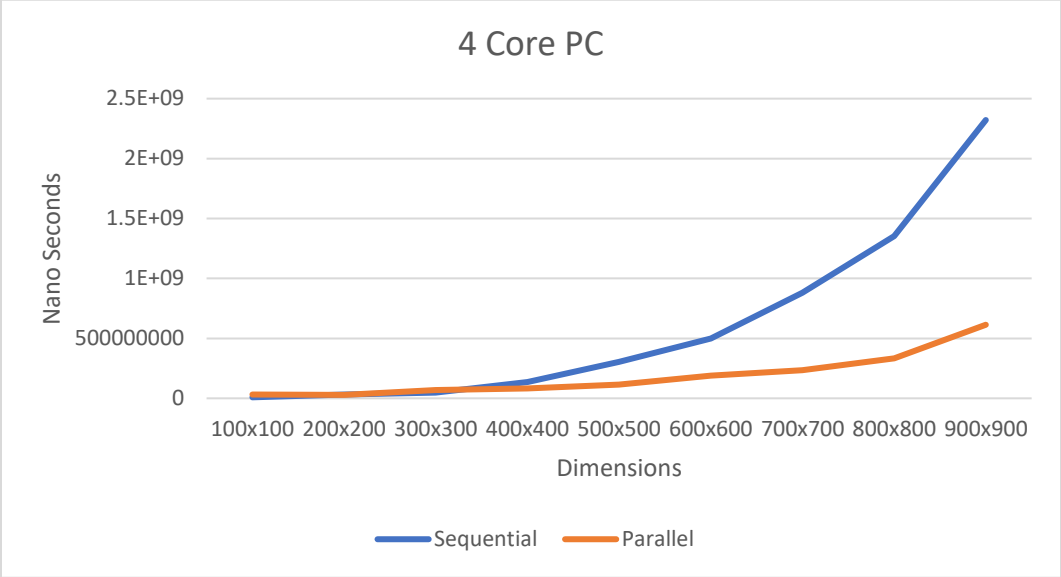
A continuación, se presentarán los resultados arrojados, en términos de tiempo y tamaño, por la ejecución del algoritmo secuencial y paralelo para multiplicar matrices cuadradas. El objetivo del análisis de dichos resultados es verificar si lo fundamentado teóricamente se cumple en la práctica. Es importante destacar que el mismo algoritmo fue computado en máquinas de 2, 4 y 6 *cores* en su CPU para fines de demostración del efecto que tiene la cantidad de núcleos en la computación concurrente.

Data-set 1: Tiempo de ejecución (ns) vs Tamaño de la matriz (Medido en 2 núcleos)



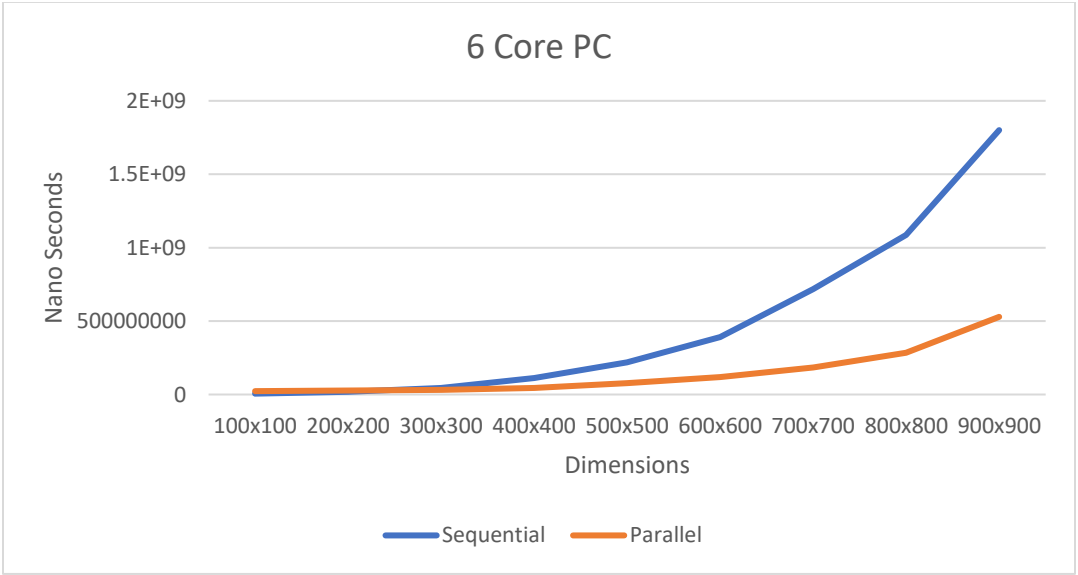
En la gráfica podemos observar cómo se comportan los algoritmos en una máquina de dos núcleos. Se muestra un comportamiento relativamente similar cuando las matrices son menores a 600x600 elementos. Sin embargo, cuando llegamos a ese número, el algoritmo paralelo se torna más lento que el algoritmo secuencial. En principio, esto fue una señal de alarma para nosotros debido a que no había concordancia con los fundamentos teóricos, y más aún cuando el algoritmo paralelo no cuenta con una región crítica que pueda ralentizarlo. Sin embargo, en los siguientes data-sets nos llevamos un alivio.

Data-set 2: Tiempo de ejecución (ns) vs Tamaño de la matriz (Medido en 4 núcleos)



La gráfica anterior es la representación de la ejecución de los algoritmos en una máquina con 4 núcleos. A partir de aquí sucedió un giro interesante en el comportamiento de ambos. Si bien con dos núcleos el algoritmo secuencial fue mejor siempre, en este hardware sí se aprecia cómo el algoritmo paralelo se comporta mucho mejor que el algoritmo secuencial cuando la cantidad de elementos tiende a infinito.

Data-set 3: Tiempo de ejecución (ns) vs Tamaño de la matriz (Medido en 6 núcleos)



Finalmente, el último data set muestra el comportamiento de los algoritmos en máquinas con 6 núcleos. El resultado es bastante parecido al de 4 núcleos, pero en este se puede apreciar una ligera mejora en el rendimiento del algoritmo paralelo. Esto nos hace plantear que la cantidad de núcleos en el CPU condicionan la rapidez de los algoritmos concurrentes.

Conclusión

A partir de la data resultante de los experimentos, afirmamos que obtuvimos los resultados esperados comparando con los fundamentos teóricos: la paralelización llega a ser más eficiente que una computación secuencial. Esto es debido a la división y ejecución de un algoritmo de manera simultánea, que permite computar sus partes en mucho menos tiempo. Fue interesante observar cómo el comportamiento estuvo afectado por la cantidad de *cores* en cada máquina, por lo que asumimos que la computación paralela es más eficiente a medida que exista un mayor número de núcleos. Por último, queremos mencionar que estos resultados no se cumplen para una data pequeña; en nuestra experiencia, cuando la data de entrada es pequeña, los algoritmos secuenciales se comportan de mejor o igual manera que los paralelos.