Práctica 1

Guillermo Marquínez

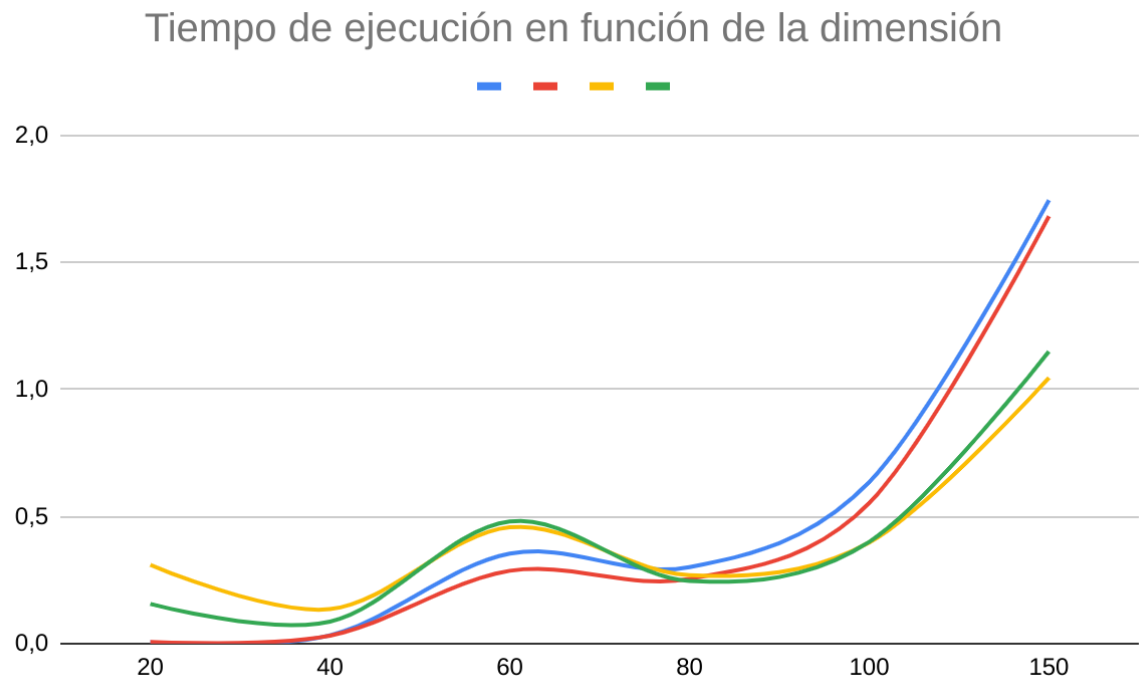
En esta práctica he implementado la suma y el producto por matrices en forma flattened tanto por filas como por columnas. Además he utilizado la biblioteca multiprocessing de python para comparar los tiempos en computación secuencial y paralela. El módulo multiprocessing hace que varias funciones completamente independientes se puedan ejecutar en paralelo. Para que tenga sentido utilizar multiprocessing en la suma o el producto de matrices he dividido la matriz en cuatro bloques. De esta manera creo cuatro funciones independientes, donde cada una se centra en dar los resultados de su bloque correspondiente. Después, estos resultados se tienen que unir en una matriz final, que será una variable en memoria compartida, para que todos los procesos 'child' tengan acceso a ella y puedan añadir su resultado.

Esta última razón implica que las funciones con multiprocessing tienen un bucle for más, que se necesita para el pegado de la solución a la matriz en la variable compartida. En realidad, se podría omitir este for y pegar el resultado del producto de cada bucle directamente en la matriz de memoria compartida, pero de esta manera no podríamos aprovechar que ya tenemos la función del producto por filas y columnas definida. Además del ciclo for, la memoria compartida hace más lento el programa.

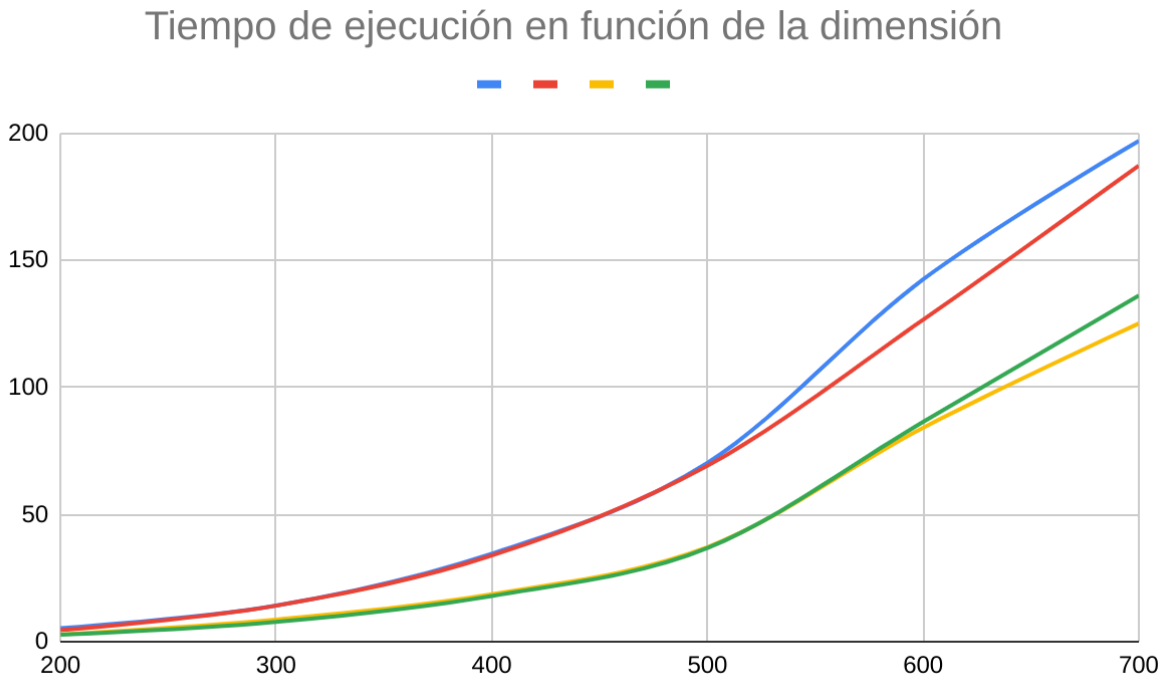
Por lo tanto, es normal que para matrices de dimensión pequeña, sea más costosa la implementación en paralelo. Sin embargo, cuanto más grande sea la matriz más beneficioso será la implementación en paralelo.

Mi ordenador solamente tiene dos CPU así que, idealmente, el tiempo de ejecución del programa en paralelo tenderá a ser la mitad que en secuencial. Cuanto más grande sea la matriz más nos acercaremos a esta cota. Sin embargo, como las matrices se dividen en cuatro bloques, este programa podría paralizarse en cuatro CPUs, y así ser, idealmente, cuatro veces más rápido que en secuencial. Para ello, basta con ejecutar el programa en un ordenador con cuatro CPUs.

Vamos ahora a comparar tiempos, tanto en el producto por filas como por columnas y en secuencial como en paralelo. En la siguiente gráfica se ha computado el programa para matrices cuadradas de dimensión 20, 40, 60, 80, 100 y 150. **La línea azul representa el tiempo empleado para el producto en secuencial y por filas, la línea roja en secuencial y por columnas, la amarilla en paralelo por filas y la verde en paralelo por columnas.** La primera conclusión, que además confirma la discusión hecha en el segundo párrafo nos dice que, más o menos, a partir de matrices cuadradas con dimensión mayor que 80, comienza a ser más eficiente emplear el paralelismo.



En el siguiente gráfico, se ha computado el programa para matrices cuadradas de dimensión 200, 300, 400, 500, 600 y 700. Aquí ya podemos notar como el tiempo en paralelo es significativamente menor que en secuencial, tanto por filas como por columnas. Como conclusión final, parece que es más eficiente la notación por columnas en secuencial, sin embargo, utilizando multiprocessing la notación flattened por filas es más eficiente.



Como discusión final, si quisiéramos paralelizar en más procesos el producto o suma de matrices, teniendo en cuenta que los bloques son también matrices, podemos aplicar la

subdivisión de matrices a cada bloque y aplicar las mismas funciones. Así, podríamos paralelizar hasta en 16, 64 o 256 procesos.