Análisis de algoritmos usando MPI

Fredy Álvarez Béjar
Algoritmos Paralelos,
Ciencia de la computación - Universidad Católica San Pablo

1

1. Introducción

En este trabajo se analizará el rendimiento de dos algoritmos, multiplicación de una matriz por un vector y *odd even sort*, usando OpenMP para la implementación de estos. Todos los tiempos se tomaron usando un procesador AMD A10-4600M que cuenta con 4 núcleos .

2. Multiplicación de una matriz por un vector

La idea de este algoritmo usando OpenMP es paralelizar la parte en la que se recorre todos los elementos de la matriz y del vector para luego multiplicarlos y aplicarlos al resultado.

Threads	Tamaño de la matriz				
	8000000 x 8	80000 x 800	8000 x 8000	800 x 80000	8 x 8000000
1	161.566	154.354	151.699	153.304	177.113
2	107.000	82.250	99.803	87.895	233.686
4	73.247	71.289	56.994	84.327	268.194
8	77.690	57.241	54.368	81.837	287.880

Cuadro 1. Tiempos de ejecución del algoritmo, usando distintos tamaños de matriz y distintas cantidades de threads. Tiempo en milisegundos

En esta comparación se aprecia el impacto del uso de la memoria caché en este algoritmo, todas las operaciones se realizan con la misma cantidad de elementos (64000000), lo que cambia son las dimensiones de la matriz. El menor tiempo de ejecución se encuentra en el vector cuya cantidad de elementos, que es igual a la cantidad de columnas de la matriz, pueda entrar y permanecer más tiempo en memoria caché. En la imagen 1 se observa que los tiempos de ejecución del algoritmo con una matriz de tamaño 8 x 80000000 es el peor de todos, debido a que el vector, con 8000000 es demasiado grande como para poder entrar completamente en memoria caché, cada vez que un thread requiere accesarlo, necesita buscarlo en memoria principal, lo que ocasiona un deterioro en el desempeño.

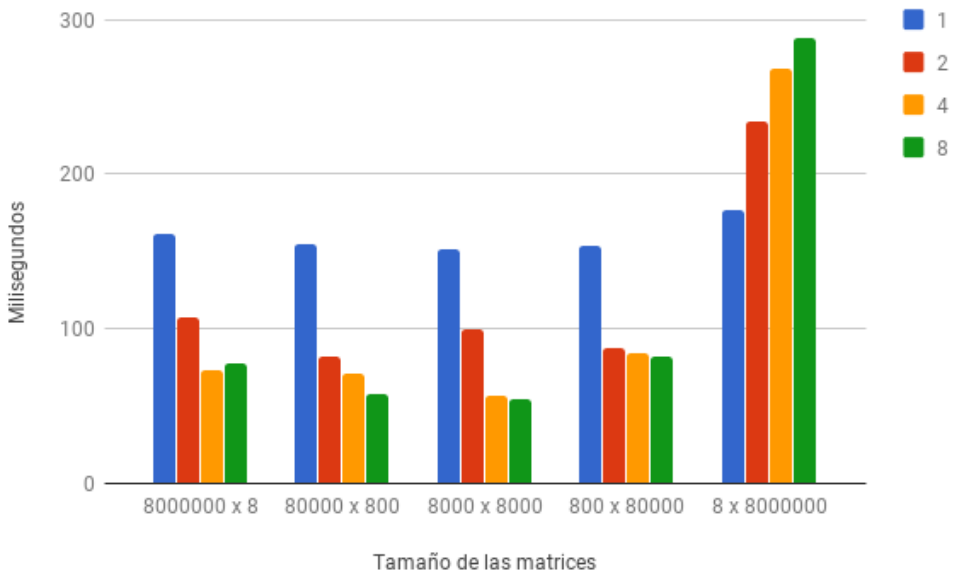

Figura 1. Gráfico del cuadro 2

3. Odd even sort

Este algoritmo es una variación de el *bubble sort* tradicional, la diferencia es que en lugar de comparar los elementos contiguos en la lista, se comparan los elementos con índices pares/impares en momentos distintos.

Método	Threads			
	1	2	4	8
Dos *for* paralelos	176.583	117.787	92.058	798.602
Dos *for*	158.762	98.434	78.316	376.807

Cuadro 2. Comparación de tiempos del algoritmo Odd Even Sort, usando un array de 10000 elementos. Tiempos en milisegundos

Aquí se están comparando dos formas de implementar este algoritmo, una en la cual se implementan dos *for* paralelos dentro de otro *for*, otra en la que la directiva de crear threads está en el *for* exterior, así los dos *for* interiores usan los threads ya creados. La primera versión es un poco más lenta, ya que en cada ciclo del *for* exterior, se deben crear threads para los *for* internos, y luego unirlos al proceso principal, en cambio en la segunda versión los threads solo se crean una vez.

También se observa en la tabla 3 que al usar más threads que cores disponibles, el tiempo empeora drásticamente, esto debido al overhead de crear y unir estos threads.

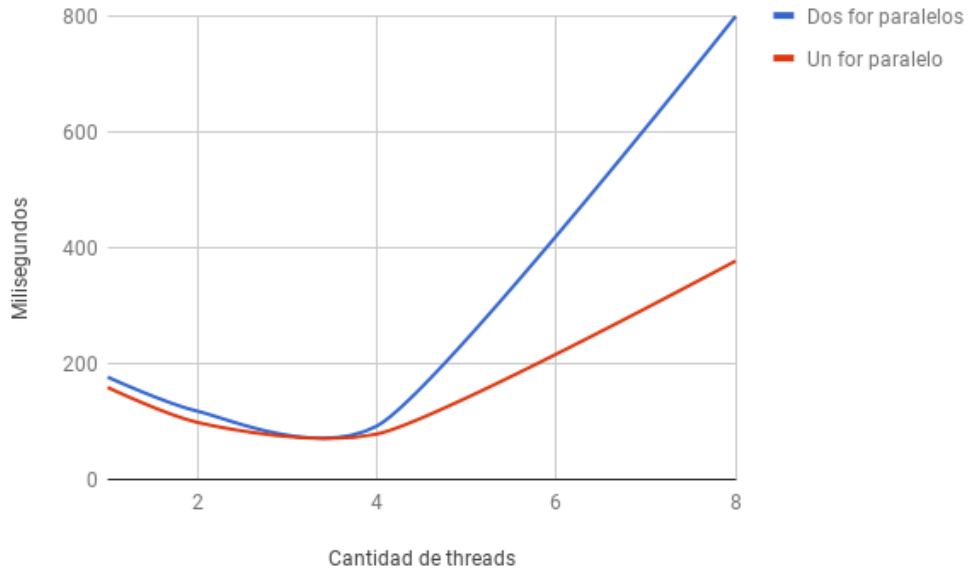

Figura 2. Gráfica del cuadro 3