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 MPI [1] (*Message Passing Interface*) para la implementación de estos.

2. Multiplicación de una matriz por un vector

La idea de este algoritmo usando MPI, es dividir a la matriz en partes más pequeñas, específicamente enviar N filas ($N = total\_filas/num\_procesadores$) de la matriz a cada proceso, para que cada uno puede realizar la multiplicación de estas y el vector, finalizado esto, se envían los resultados al proceso principal para procesar el resultado general.

Cuadro 1. Comparación de tiempos (en milisegundos) del algoritmo de multiplicación de una matriz por un vector

Procesos	Dimensión de la matriz						
	1024	2048	4096	8192	16384	32768	65536
1	0.160	0.316	1.084	2.144	4.473	7.266	13.143
2	0.160	0.305	0.590	1.287	2.337	4.899	10.447
4	0.098	0.171	0.367	0.863	1.246	2.532	5.510
8	69.998	139.992	129.999	59.982	79.994	106.834	140.022
16	385.620	369.991	399.990	339.993	379.998	340.905	369.094

En el cuadro 1 y el gráfico 1 se aprecia que el tiempo de ejecución de un único proceso no es siempre el más lento. En este caso, cuando se ejecutan 16 procesos y 8 proces, el tiempo de ejecución resulta ser mayor al tiempo de ejecución de un solo proceso, esto puede ser debido a que la máquina en la que fueron tomados estos tiempos solo cuenta con 4 procesadores, y al tener más procesos ejecutándose al mismo tiempo, hay un *overhead* al cambiar de un proceso a otro. También se puede ver la clara diferencia que hay al ejecutar el programa con 2 y 4 procesos, comparado con la ejecución de este en un solo proceso.

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.

Otra vez se puede apreciar el *overhead* de tener más procesos que la cantidad de procesadores del computador, en este caso la diferencia es muy notable, por esta razón en la gráfica 3 solo se consideran los tiempos de ejecución del algoritmo en 1, 2 y 4 procesos.

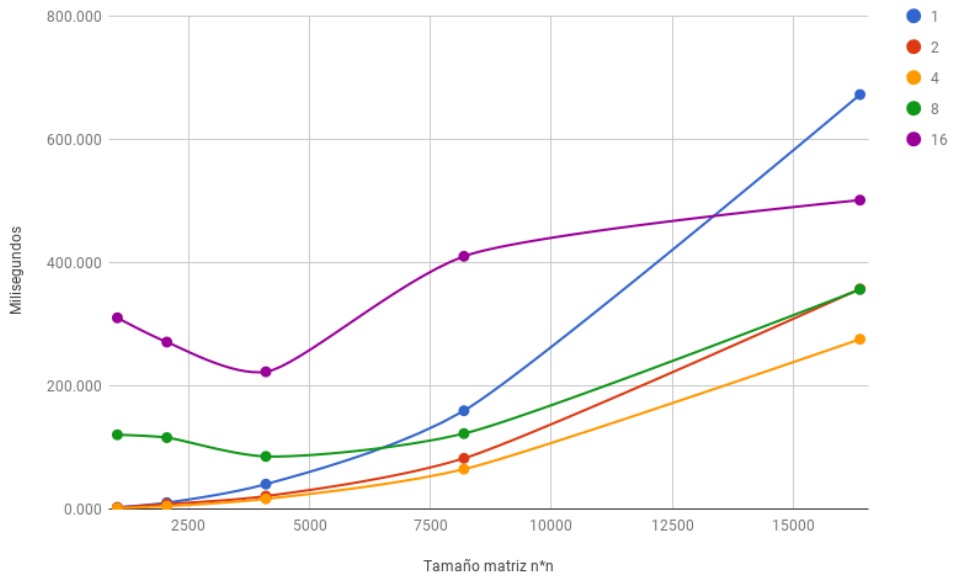

Figura 1. Gráfica del cuadro 1

Cuadro 2. Comparación de tiempos (en milisegundos) del algoritmo odd even sort

Procesadores	Cantidad de elementos del array						
	1024	2048	4096	8192	16384	32768	65536
1	0.160	0.316	1.084	2.144	4.473	7.266	13.143
2	0.160	0.305	0.590	1.287	2.337	4.899	10.447
4	0.098	0.171	0.367	0.863	1.246	2.532	5.510
8	69.998	139.992	129.999	59.982	79.994	106.834	140.022
16	385.620	369.991	399.990	339.993	379.998	340.905	369.094

Referencias

[1] (). MPICH — High-Performance Portable MPI, dirección: `https://www.mpich.org/` (visitado 19-04-2018).

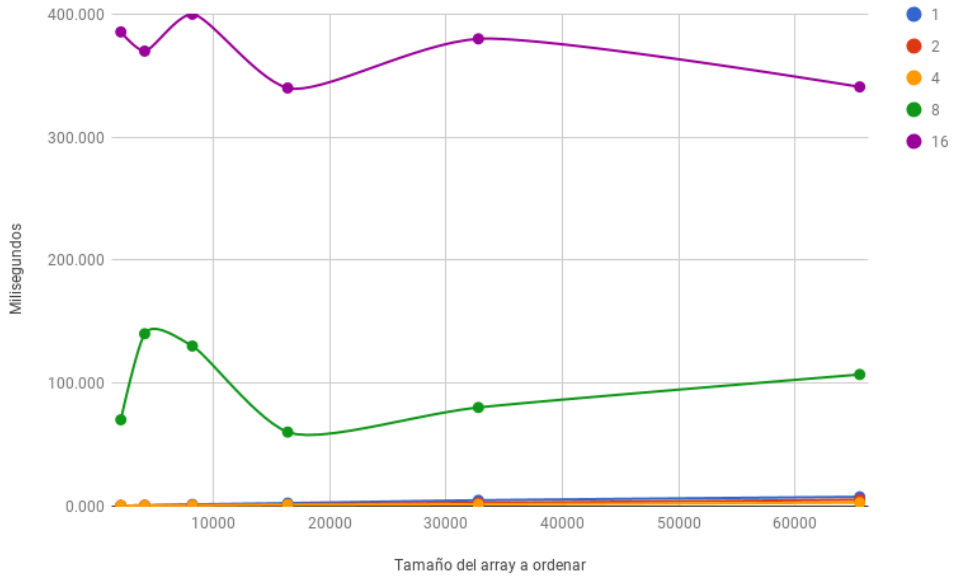

Figura 2. Gráfica del cuadro 2

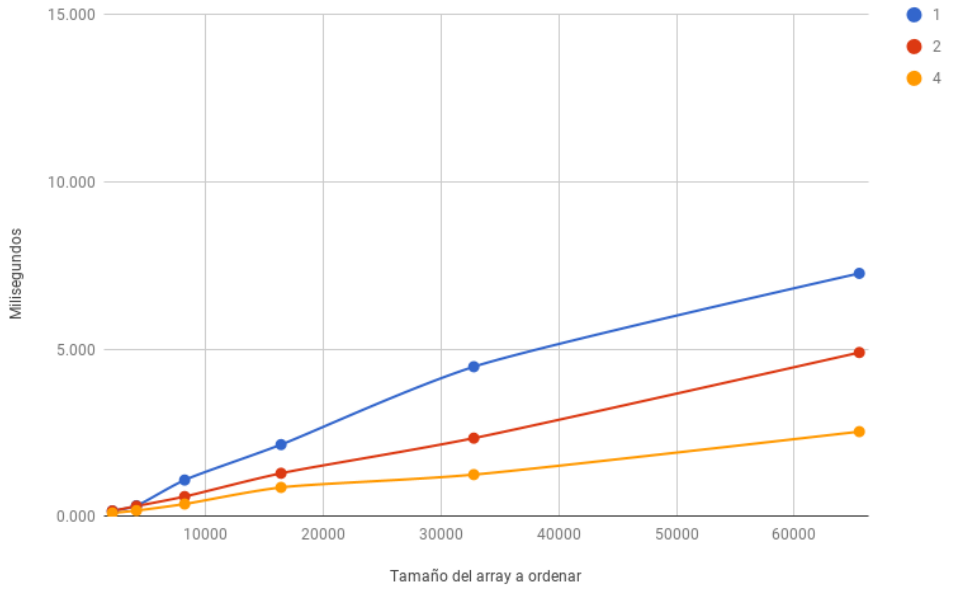

Figura 3. Gráfica del cuadro 2, considerando solo los tiempos de 1, 2 y 4 procesadores