Informe de Desempeño de Paralelización.

Leonardo A. Pérez Castilla y Luis E. Ochoa López

I. Detalles de la máquina.

La toma de las muestras se realizó en una máquina con sistema operativo **macOS Mojave** y con las siguientes características en hardware:

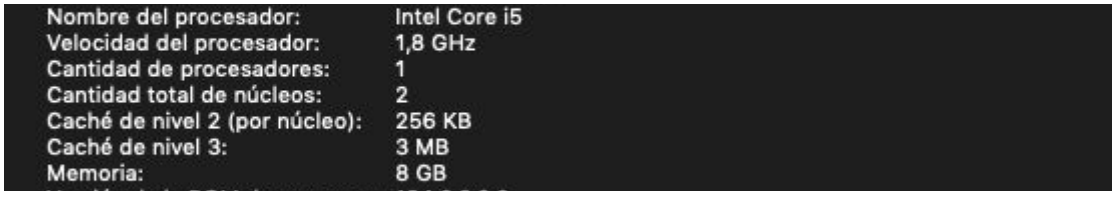

Figura 1. Resumen del hardware.

II. Resultados.

Para la realización de éste informe y con fines educativos, se recolectaron 100 muestras por cada tipo de ejecución (secuencial, 2 hilos, 4 hilos, 8 hilos y 16 hilos) utilizando en esencia dos *programas*: **dot_product.out** y **exe.sh.**

A. dot_product.out:

Es el programa principal, se encarga de realizar leer los archivos donde están los vectores y realizar el producto punto entre ellos. Éste es el pedido en *Tarea Lab. 4 - Paralelización de un código.*

B. exe.sh:

Es un script que tiene como función a la compilar y ejecutar el archivo dot_product.out. Posee dos variantes: compilar y ejecutar una vez todo, y generar un archivo de salida llamado *resultado_[v]_[te].txt* ([v] es el número del vector y [te] el tipo de ejecución) el cual almacena los tiempos de muestra.

Además de un **benchmark** proporcionado. Tiene los siguientes datos:

Nombre	Total de datos	Resultado
vec_10_1	10^1	19

vec_10_3	10^3	179847
vec_10_6	10^6	2442305
vec_10_8	10^8	29496637

Tabla 1. Benchmark.

A continuación, veremos el **promedio de tiempo** y su **speedup** para cada tipo de ejecución y vector:

Tipo de ejecución	vec_10_1	
	Promedio (Segundos)	**Speedup**
Sequential	0,00000018	1
#2	0,00012521	0,001437584857
#4	0,00017411	0,001033829188
#8	0,00023159	0,0007772356319
#16	0,00039311	0,0004578871054

Tabla 2. Promedio vec_10_1.

En la **Tabla 2.** es necesario resaltar que la ejecución con 16 hilos no tiene sentido; sin embargo, el programa **dot_product.out** detecta ello, por lo que el número de hilos utilizados es igual al *Total de datos* de la **Tabla 1.** El programa crea los 16 hilos, pero sólo utiliza 10.

Execution	vec_10_3	
	Promedio (Segundos)	**Speedup**
Sequential	0,00000556	1
#2	0,0001312	0,04237804878
#4	0,00018066	0,0307760434
#8	0,00024679	0,0225292759
#16	0,00042306	0,01314234388

Tabla 3. Promedio vec_10_3.

Execution	vec_10_6	
	Promedio (Segundos)	**Speedup**
Sequential	0,00364239	1
#2	0,00233852	1,557562048
#4	0,00223792	1,627578287
#8	0,0023126	1,575019459
#16	0,00240525	1,51434986

Tabla 4. Promedio vec_10_6.

Execution	vec_10_8	
	Promedio (Segundos)	Speedup
Sequential	0,38327128	1
#2	0,22433928	1,708444816
#4	0,20973063	1,827445424
#8	0,21501928	1,782497272
#16	0,2097618	1,827173871

Tabla 5. Promedio vec_10_8.

III. Análisis de resultados.

Finalmente explicaremos los *speedups* obtenidos con ayuda de gráficas. Para ello, tengamos en cuenta las dos premisas siguientes:

- Será **mejor** entre más lejos esté **por encima** de 1.
- Será **peor** entre más lejos esté **por debajo** de 1.

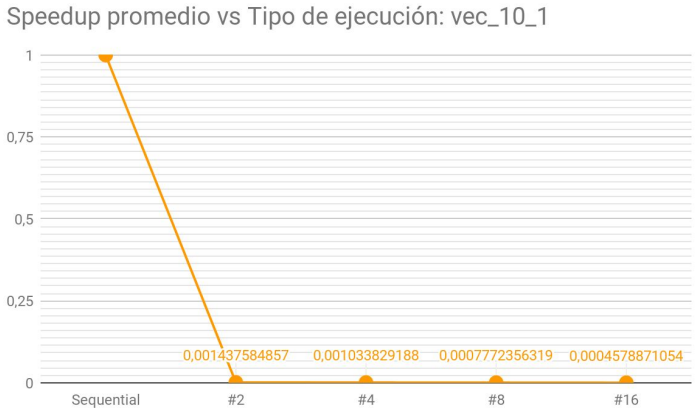

Figura 2. Speedup vec_10_1.

Aquí es evidente que los hilos generan un *overhead* o gasto de recursos (creación y cambios de contexto) que no son aprovechados debidamente para la poca cantidad de datos que el archivo posee, haciendo que el *speedup* decaiga en más de un 98% para todos los casos donde no es secuencial la ejecución.

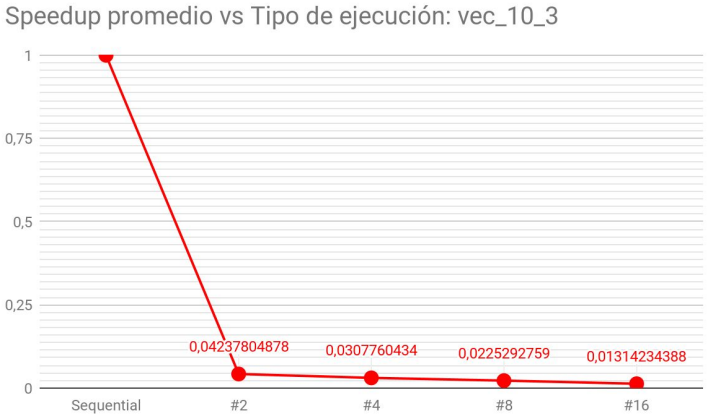

Figura 3. Speedup vec_10_3.

En este caso se ve lo mismo que con la **Figura 2.** con la diferencia que el *speedup* es mejor mínimamente.

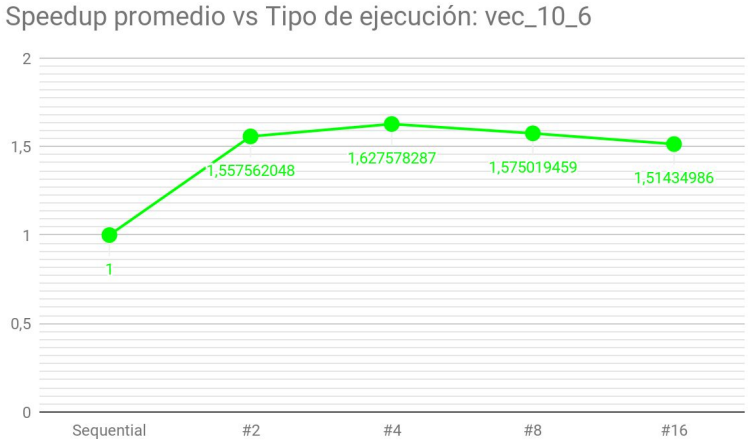

Figura 4. Speedup vec_10_6.

A partir del vector de tamaño 10^6, es notable que el uso de hilos es aprovechado debidamente; sin embargo, teniendo en cuenta los detalles de la máquina, hay que destacar que el mejor *speedup* estará por lo general en 4 hilos, más que eso, se empezará a generar un *overhead.*

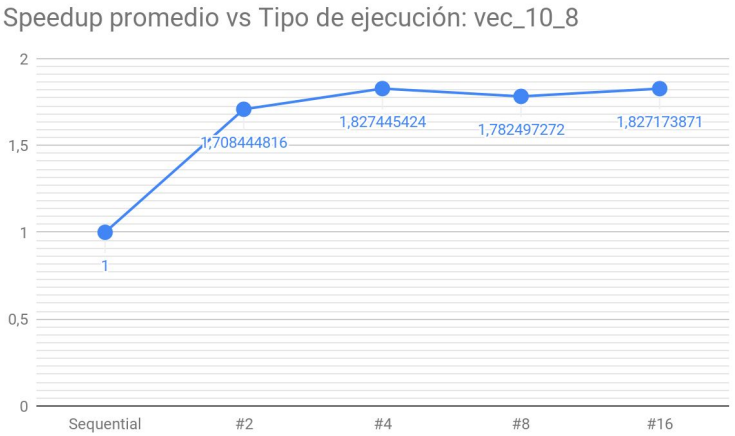

Figura 5. Speedup vec_10_8.

En éste caso vemos que con 16 hilos se obtiene el segundo mejor *speedup*, ésto puede darse porque en ese momento la máquina esté trabajando menos o tenga mínima carga.