Reporte: Optimización OpenMP

Daniel Rojas M.

2016089821

1. Ley de Amdahl

Para entender primero la ley de Amdahl, es importante entender que usualmente un sistema de información, como lo puede ser un algoritmo, tiene segmentos secuenciales no paralelizables y segmentos paralelizables. Un sistema puede ser acelerado paralelizando esos segmentos, pero solamente puede ser acelerado un cierto porcentaje en función del porcentaje que representen del total del sistema. La ley de Amdahl estima la mejora de tiempo teórica del sistema,

Se puede resumir mediante la siguiente ecuación:

$$S = \frac{1}{(1-p) + \frac{p}{s}} \quad (1)$$

Donde S representa la mejora de velocidad general del programa, p el porcentaje del programa que es paralelizable y s la mejora en tiempo de la parte paralelizable.

Por ejemplo, para un programa donde el 60% de la ejecución es paralelizable y luego de acelerar se obtiene una mejora de 6 veces el tiempo, se tiene:

$$S = \frac{1}{(1-0.6) + \frac{0.6}{6}} = \frac{1}{0.4 + 0.1} = 2$$

Es decir que el programa se acelerará el doble (un factor de 2). Si se analiza la ecuación (1) se puede notar que si se hace tender S al infinito (una mejora increíblemente mayor) se puede estimar el máximo posible de mejora para todo el programa de la forma:

$$S_{max} = \frac{1}{1-p} \quad (2)$$

Para esta aplicación (calcular π mediante la aproximación de una integral como una serie numérica), este cuello de botella de código no paralelizable se encuentra en la terminación del paso (inverso al número de pasos) y en la suma de cada valor para cada paso. Aunque fuese infinitamente paralelizable, estas operaciones siempre se deben hacer secuencialmente.

2. Resultados de pi sin optimizar y optimizado

Para el cálculo de pi sin optimizar (pi.c) se obtuvo un promedio de 0.1452 segundos (tabla 1.). Este será el valor de comparación para las demás optimizaciones.

Tabla 1. pi.c sin optimizar

Iteración	Tiempo Total (s)
1	0.135771
2	0.136221
3	0.158811
4	0.136295
5	0.158690
Promedio	**0.145200**

Agregando una reducción dentro de un bloque *for* paralelizable (pi_omp_private.c), el cálculo de pi se reduce a 0.084121 segundos (tabla 2), una mejora de 1.726 aproximadamente. Esto se debe a que si existe una parte considerable del programa que puede ser paralelizable.

Tabla 2. pi.c con optimización de reducción paralela

Iteración	Tiempo Total (s)
1	0.084586
2	0.084747
3	0.088374
4	0.082365
5	0.080531
Promedio	**0.084121**

Al contrario de la reducción paralela, la distribución de carga entre equipos (pi_omp_teams.c) parece incrementar el tiempo promedio de una aplicación a 1.1553 segundo; un factor de 0.9349 (7% más lento) como se muestra en la tabla 3. Esto se debe al gasto adicional de crear los equipos y distribuir la carga.

Tabla 3. pi.c con optimización de distribución de trabajo en equipos

Iteración	Tiempo Total (s)
1	0.159256
2	0.150522
3	0.150317
4	0.159173
5	0.157278
Promedio	0.155300

Finalmente, se calcula pi con reducción paralela variando los hilos (pi_omp_threads.c) y se puede notar en la tabla 4 como alrededor de los 4 hilos se logra la máxima optimización en promedio y a partir de 4 hilos no se mejoran los tiempos. La representación visual de los datos se puede ver en la figura 1.

Tabla 4. pi.c con optimización

Tiempo Total (s)	Iteración					
Hilos	1	2	3	4	5	**Promedio**
1	0.144461	0.163685	0.157343	0.151090	0.162440	**0.155804**
2	0.070938	0.130973	0.131599	0.131398	0.113591	**0.115700**
3	0.080514	0.087888	0.087597	0.087577	0.089653	**0.086646**
4	0.068349	0.067604	0.067988	0.068311	0.068210	**0.068092**
5	0.070520	0.080310	0.069782	0.070477	0.071011	**0.072420**
6	0.069894	0.072513	0.068155	0.069558	0.067561	**0.069536**
7	0.067494	0.068402	0.069188	0.067525	0.068348	**0.068191**
8	0.068150	0.068090	0.067018	0.070003	0.069040	**0.068460**
9	0.071028	0.070773	0.067701	0.068720	0.067908	**0.069226**
10	0.067519	0.066832	0.067706	0.070950	0.067531	**0.068108**
11	0.068432	0.069189	0.067835	0.068318	0.070212	**0.068797**
12	0.070096	0.067851	0.068994	0.068088	0.067447	**0.068495**

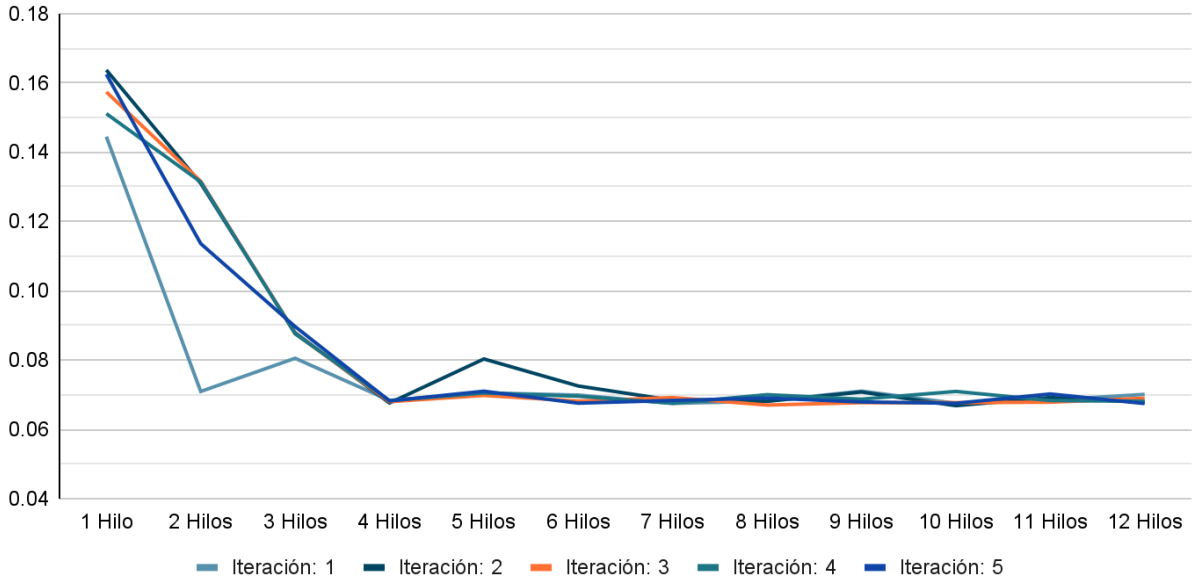

Figura 1. Tiempo de ejecución según cantidad de hilos para 5 iteraciones