Quiz 3 - Programación Paralela

Juan Pablo Ovalles Ceron

March 2025

1 Algorithms Performance I (Amdahl's law)

$$S = \frac{1}{(1-P) + \frac{P}{N}}$$

Where:

- **S** = Speedup of the system
- **P** = Proportion of the program that can be parallelized
- **N** = Number of processors
- **1 - P** = Portion of the program that is inherently sequential and cannot be parallelized

1. Assume 1% of the runtime of a program is not parallelizable. This program runs on 61 cores of an Intel Xeon Phi. Under the assumption that the program runs at the same speed on all of those cores, and there are no additional overheads, what is the parallel speedup?

- **P** = 0.99
- **N** = 61
- **1 - P** = 0.01

$$S = \frac{1}{(0.01) + \frac{0.99}{61}} = 38.125$$

Analisis: Esto muestra que incluyo una pequeña porción de código secuencial puede limitar el desempeño general, pues el valor del speedup es mucho más bajo que el numero de cores. Teniendo en cuenta que si no tuvieramos porción secuencial el speedup sería de 61 se pierde bastante potencia.

2. Assume 0.1% of the runtime of a program is not parallelizable. This program is supposed to run on the Tianhe-2 supercomputer, which consists of 3,120,000 cores. Under the assumption that the program runs at the same speed on all of those cores, and there are no additional overheads, what is the parallel speedup on 30, 30,000 and 3,000,000 cores?

- **P** = 0.999
- **1 - P** = 0.001

- **N** = 30 $$S = \frac{1}{(0.001) + \frac{0.999}{30}} = 29.154$$
- **N** = 30000 $$S = \frac{1}{(0.001) + \frac{0.999}{30000}} = 967.77$$
- **N** = 3000000 $$S = \frac{1}{(0.001) + \frac{0.999}{3000000}} = 999.66$$

Analisis: Para 30 cores casi se consigue una eficiencia total con un speedup cercano a 29, mientrás que para 30.000 y 3'000.000 la cosa es distinta pues no estuvieron ni cerca de alcanzar tal eficiencia con un speedup de 967 y 999 respectivamente. Esto también nos hace ver el comportamiento logaritmico de la formula pues aunque se aumentarán los cores 100 veces el speedup terminó siendo similar, es decir, su aumento fue muy poco.

3. Assume that the program invokes a broadcast operation. This broadcast adds overhead, depending on the number of cores involved. There are two broadcast implementations available. One adds a parallel overhead of 0.0001n, the other one $0.0005 \cdot \log(n)$. For which number of cores do you get the highest speedup for both implementations?

- Incluyendo el overhead tenemos:

$$S = \frac{1}{(1-P) + \frac{P}{N} + overhead(n)}$$

- Asumiendo que todo el programa es paralelizable tenemos que P = 1 y (1-P) = 0. En ese sentido la formúla que nos queda es:

$$S = \frac{1}{\frac{1}{N} + overhead(n)}$$

- Ahora revisemos el caso en el que **overhead(n) = 0.0001n**.

$$S = \frac{1}{\frac{1}{n} + 0.0001n}$$

- Podemos derivar el speedup en función de n e igualar el resultado a 0 para encontrar un punto máximo.

$$S' = -\frac{-\frac{1}{n^2} + 0.0001}{\frac{1}{n^2} + 0.0002 + 0.00000001n^2} = 0$$

$$n = 100, n = -100$$

$$S(n = 100) = \frac{1}{\frac{1}{100} + 0.0001 * 100} = 50$$

- Teniendo en cuenta que n representa el núemro de cores, descartamos el -100 dado que no tiene sentido contar con una cantidad negativa de cores. Es así que encontramos que la cantidad de cores que aseguran un speedup óptimo es de 100 cores asegurando un speedup de 50. También podemos verlo gráficamente así:

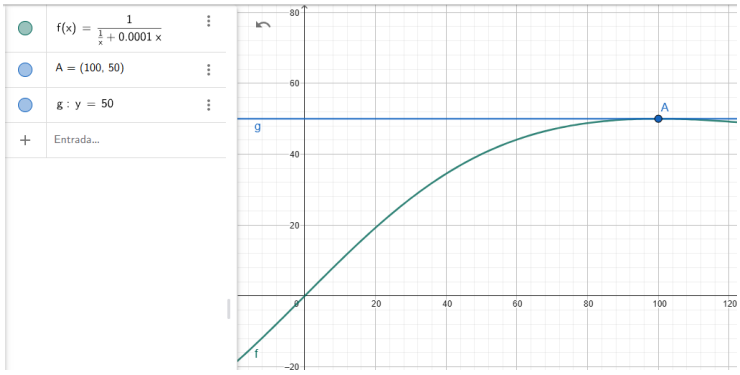

Figure 1: Speedup máximo con overhead(n) = 0.0001n

- Ahora revisemos el caso en el que **overhead(n) = 0.0005 log(n)**.

$$S = \frac{1}{\frac{1}{n} + 0.0005 * log(n)}$$

- De nuevo podemos derivar el speedup en función de n para igualar el resultado a 0 y encontrar un punto máximo.

$$S' = -\frac{-1 + 0.00021n}{(1 + 0.0005n \log_{10}(n))^2} = 0$$

$$n = 4761.90476 \approx 4761$$

$$S(n = 4761) = \frac{1}{\frac{1}{4761} + 0.0005 * log(4761)} = 488.06$$

- En este caso pudimos encontrar que el número óptimo de cores corresponde a 4761 cores lo que nos asegura un speedup de 488.06. Graficamente tenemos:

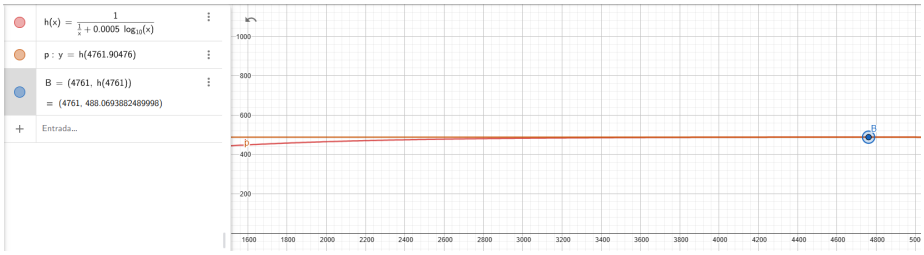

Figure 2: Speedup máximo con overhead(n) = 0.0005*log(n)

4. By Amdahl's law, it does not make much sense to run a program on millions of cores, if there is only a small fraction of sequential code (which is often inevitable, i.e., reading input data). Why do people build such systems anyway?

Respuesta: Los supercomputadores con millones de núcleos se construyen porque hay tareas que pueden dividirse en muchas partes y ejecutarse en paralelo, como la inteligencia artificial, simulaciones o el clima. Además, permiten procesar muchos trabajos al mismo tiempo y ayudan en la investigación, aunque siempre haya algo de código que no se pueda paralelizar. Hay que considerar también que una solución de programación para un problema masivamente paralelo puede no tener código secuencial, también conocidos como algoritmos vergonzosamente paralelos.

5. We can find the minimum from an unordered collection of n natural numbers by performing a reduction along a binary tree: In each round, each processor compares two elements, and the smaller element gets to the next round, the bigger one is discarded. What is the work and depth of this algorithm?

- **Work:**

Para el trabajo se tiene que en cada ronda la cantidad de números se reduce a la mitad, puesto que por cada comparación solo uno pasa de ronda, y se repite el roceso hasta quedar con un solo número. Es decir, en la primera ronda se hacen n/2 comparaciones, en la segunda ronda n/4 comparaciones y asi sucesivamente hasta la última comparación que es una sola. En ese sentido podemos ver el trabajo de la siguiente manera:

$$W(n) = \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots + 1$$

Esta es una serie geométrica con primer término $\frac{n}{2}$ y razón $\frac{1}{2}$ que converge a n. Por lo tango podemos concluir que el trabajo está acotado por $O(n)$.

- **Depth:**

En este caso hablamos del numero de rondas para conseguir el número más pequeño. Dado que la cantidad de elementos a evaluar es reducida a la mitad en cada ronda el número de rondas esta acotado por una función logaritmica, es decir $Depth = O(log(n))$.

2 Algorithms Performance I (Speedup)

Explore and compare three different implementations of Conway's Game of Life in a notebook environment. The activity aims to develop analytical skills in evaluating algorithm efficiency, code structure, and environment behaviour.

1. Implementation Review: En la versión secuencial se recorre la matriz de celdas de manera iterativa y sin dividir el trabajo. Mientrás que en la versión de multithreading se usan hilos para ejecutar partes de la matriz en paralelo, dentro del mismo proceso. Se divide la malla horizontalmente en particiones. Cada partición es asignada a un hilo. Por último usando procesos se divide la matriz entre múltiples procesos similar al caso de los hilos, dividiendo la carga de trabajo. La versión secuencial y multithreading en su procesamiento comparten memoria de manera natural, mientrás que para la versión usando procesos requiere definir de manera explicita la memoria compartida.

2. Performance Analysis:

- Tiempo promedio de ejecución (en segundos)

Implementación	**Google Colab**	**Computador Local**
Secuencial	2.64	3.50
Multithreading	6.86	5.84
Multiprocessing	2.34	2.98

Table 1: Comparación de tiempos de ejecución

Multithreading fue más lento que la versión secuencial, probablemente por causas ajenas a la implementación como el hardware. Multiprocessing mejoró ligeramente el rendimiento, pero no logró una aceleración significativa. Las diferencias entre Google Colab y los computadores locales pueden deberse a la configuración del hardware y la gestión de recursos.

- Speedup por implementación

Implementación	**Speedup Colab**	**Speedup Local**
Multithreading	0.3806	0.5972
Multiprocessing	1.11	1.1705

Table 2: Comparación de Speedup en Google Colab y Computador Local

Multithreading tiene un speedup menor a 1 en ambos entornos, lo que indica que, en lugar de acelerar la ejecución, en realidad la hace más lenta como ya veíamos en el análisis de los tiempos de ejecución. Esto puede deberse a la Global Interpreter Lock (GIL) de Python, que limita la ejecución de hilos en CPUs con Python puro. Multiprocessing logra un speedup mayor a 1 en ambos entornos, lo que significa que sí logra una aceleración respecto a la ejecución secuencial. Speedup en el computador local es mayor que en Google Colab, aunque la diferencia no es muy significativa, esto puede ser porque el hardware de Colab puede tener restricciones en la ejecución de procesos paralelos.