



# Computación Paralela y Distribuida

2020-II

---

José Fiestas

November 10, 2020

Universidad Nacional de Ingeniería  
[jose.fiestas@uni.edu.pe](mailto:jose.fiestas@uni.edu.pe)

## Unidad 2: Metodos de paralelismo

Objetivos:

1. Velocidad, eficiencia, escalabilidad. Ley de Ahmdal
2. DAG (Directed Acyclic Graphs)
3. Caso práctico: Algoritmo de N-cuerpos en paralelo
4. Operaciones basicas de paralelismo
5. Modelos computacionales en paralelo (PRAM)
6. Broadcast/Reduccion

**Velocidad, eficiencia,  
escalabilidad. Ley de Ahmdal**

---

El tiempo de ejecución en paralelo (normalizado a la unidad) es:

$$T^s = s + p$$

Resolverlo en N procesadores, implica que:

$$T^p = s + \frac{p}{N}$$

## strong scaling

Si definimos **performance** como **trabajo por tiempo**:

$$P^s = \frac{p+s}{T^s} = 1$$

mientras que en paralelo

$$P^p = \frac{p+s}{T^p} = \frac{p+s}{s+(1-s)/N} = \frac{1}{s+(1-s)/N}$$

Y midiendo la escalabilidad, como:

$$S = \frac{P^p}{P^s} = \frac{1}{s+p/N}$$

## weak scaling

Si definimos el tiempo de ejecución para un problema variable, tenemos:

$$T^s = s + pN^\alpha$$

$$T^p = s + pN^{\alpha-1}$$

Siendo el performance:

$$P^s = \frac{s+p}{T^s} = 1$$

y en paralelo:

$$P^p = \frac{s+pN^\alpha}{T^p} = \frac{s+pN^\alpha}{s+pN^{\alpha-1}}$$

Siendo la escalabilidad :

$$S = \frac{P^p}{P^s} = \frac{s+pN^\alpha}{s+pN^{\alpha-1}} = P^p$$

Para  $\alpha = 0$  obtenemos **strong scaling**

## Ley de Amdahl

Intenta responder la pregunta sobre que tan rápido ejecuta un código en paralelo cuando utilizo  $N$  procesos. El speedup se incrementa con  $N$ .

E.g., para  $N \rightarrow \infty$ ,  $S = 1/s$

$$S = \frac{1}{s + \frac{1-s}{N}}$$

Se trata del poder computacional en paralelo

$$\epsilon = \frac{\text{performance } N \text{ procesos}}{N \times \text{performance en 1 proceso}} = \frac{\text{speedup}}{N}$$

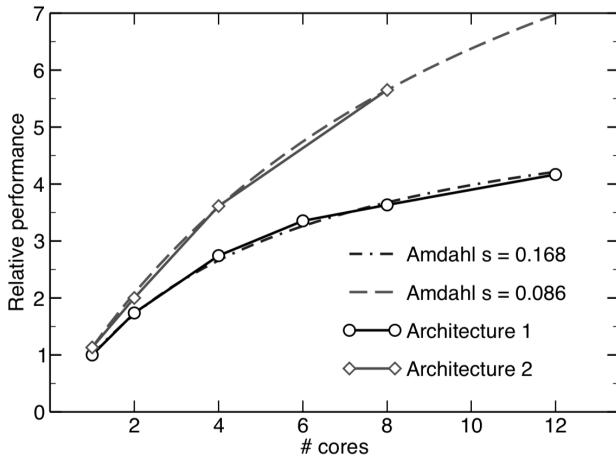
Considerando weak scaling, ya que en el límite  $\alpha \rightarrow 0$  deriva en Amdahl.

Si el trabajo es  $s + pN^\alpha$ , obtenemos:

$$\epsilon = \frac{S}{N} = \frac{s + (1-s)N^\alpha}{(s + (1-s)N^{\alpha-1}) \times N} = \frac{sN^{-\alpha} + (1-s)}{sN^{1-\alpha} + (1-s)}$$

$$\text{Para } \alpha = 0, \epsilon = \frac{1}{sN + (1-s)}$$

$$\text{Para } \alpha = 1, \epsilon = sN^{-1} + (1-s)$$



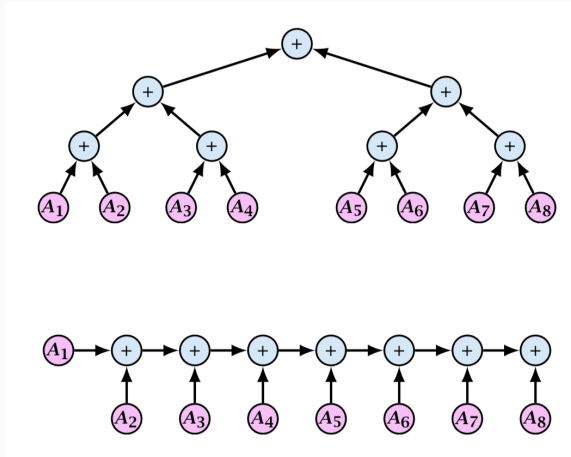


# DAG (Directed Acyclic Graphs)

---

# Modelo DAG (Directed Acyclic Graph)

- vértices representan operaciones (instrucciones simples o en bloques)
- aristas representan dependencias (precedencia)



# Modelo DAG (Directed Acyclic Graph)

El **scheduling** o 'plan de ejecución' de un DAG  $(V,E)$ , asigna a cada nodo  $v$  un tiempo de ejecución  $t_v$  y un procesador  $p_v$ , implementando, de esta manera, el algoritmo paralelo correspondiente. I.e.

$$- p_v \in \{1, \dots, p\}, t_v \in \{1, \dots, p\}$$

$$- t_u = t_v, \rightarrow p_u \neq p_v$$

$$- (u, v) \in E \rightarrow t_u = t_v + 1$$

donde,  $t_x = 0$  para los nodos de input.

$T$  es la longitud del plan de ejecución

Se busca que el algoritmo en paralelo tenga un **trabajo** eficiente (mínimo) y una profundidad (**span**) mínima

# Modelo DAG (Directed Acyclic Graph)

Considere la función :

$$f(x)=x, \text{ si } x \leq 1$$

$$f(x)= f(x-1)+f(x-2), \text{ i.e. } (a,b)=(f(x-1) \parallel f(x-2))$$

Entonces, el trabajo (W)

$$W(n)=1, \text{ si } n \leq 1$$

$$W(n)= W(n-1)+W(n-2)+1$$

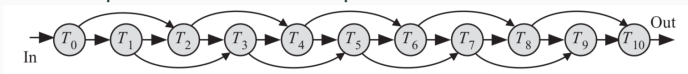
La profundidad (span)

$$S(n)=1, \text{ si } n \leq 1$$

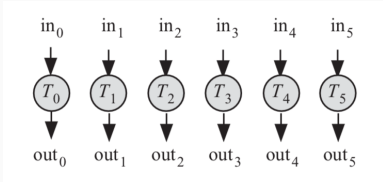
$$S(n)= \text{MAX}(S(n-1),S(n-2))+1$$

Podemos clasificar a los algoritmos de la siguiente forma:

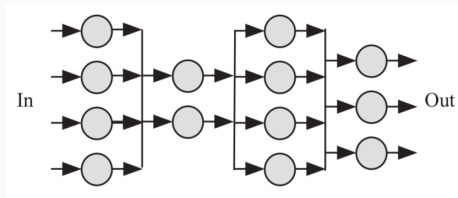
- **Secuenciales**, no pueden ser paralelizados porque todas las tareas tienen dependencias en tareas previas



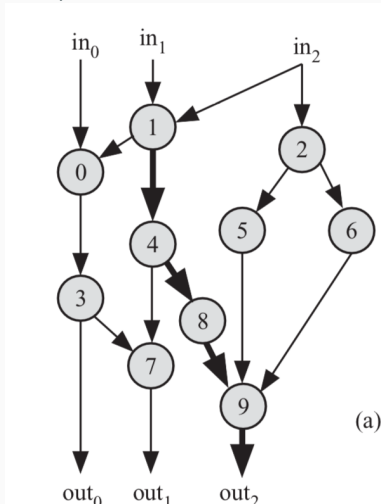
- **Paralelos**, donde todas las tareas pueden ser ejecutadas simultáneamente



- **SPA** (Secuencial-Paralelo) El algoritmo está separado en niveles, que se ejecutan en paralelo, pero los cuales tienen una forma secuencial de ejecución.



- **NSPA** (No-Secuencial-Paralelo) El algoritmo no contiene ninguno



de los patrones anteriores

## Importancia del paralelismo y estrategias.

- Es impracticable mejorar performance en un solo procesador (secuencial)
- Se necesita software que detecte paralelismo en un algoritmo dado
- paralelismo debe enfocarse a todo nivel: algoritmo, código, sistema operativo, compilador, hardware
- Se debe considerar el tiempo de comunicación entre procesos así como de proceso a memoria. Se diferencian entonces algoritmos limitados por **tiempo de ejecución** por proceso, de los limitados por **comunicación** entre procesos
- se intenta en muchas ocasiones, que el hardware se adapte al software



# Ejercicios

---

# Ejercicio 1

Un código de N-cuerpos realiza la siguiente operación iterando un millón de veces

$$a_0 + G * (m_1 * m_2) / (x^2 + y^2 + z^2) \quad (1)$$

Si el cálculo en 10 nodos demora 1  $\mu s$ , calcule los FLOPS (reales) del algoritmo. Si las especificaciones en cada procesador indican 2.5 TFLOPS (teóricos), ¿Cual es la eficiencia real de cada procesador ?





## Ejercicio 2

Considere el caso en el que es trabajo este definido solo por la parte en paralelo.

Calcule performance y speedup para el caso de weak y strong scaling.

Comente los resultados

¿Cambia la escalabilidad en estos casos (speedup)

-  David B. Kirk and Wen-mei W. Hwu *Programming Massively Parallel Processors: A Hands-on Approach*. 2nd. Morgan Kaufmann, 2013. isbn: 978-0-12-415992-1.
-  Norm Matloff. *Programming on Parallel Machines*. University of California, Davis, 2014.
-  Peter S. Pacheco. *An Introduction to Parallel Programming*. 1st. Morgan Kaufmann, 2011. isbn: 978-0-12-374260- 5.
-  Michael J. Quinn. *Parallel Programming in C with MPI and OpenMP*. 1st. McGraw-Hill Education Group, 2003. isbn: 0071232656.
-  Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Program- ming*. 1st. Addison-Wesley Professional, 2010. isbn: 0131387685, 9780131387683.