

# SISTEMAS PARALELOS

---

Clase 5 – Análisis de rendimiento

Prof Dr Enzo Rucci



FACULTAD DE INFORMATICA



UNIVERSIDAD  
NACIONAL  
DE LA PLATA

# Agenda de esta clase

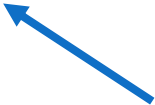
- Análisis de rendimiento

# ANÁLISIS DE RENDIMIENTO EN SISTEMAS PARALELOS

---

# Rendimiento (o prestaciones)

Cantidad de transacciones  
(efectivas) en un espacio de tiempo



- Término *amplio*

Avión	Washington a París (hs)	Velocidad (mph)	Pasajeros (p)	Throughput (pmph)
747	6.5	610	470	286700
Concorde	3h	1350	132	178200

- ¿Cuál de los presenta mejor rendimiento?
  - Depende de la **métrica** que usemos para evaluar
    - Tiempo para completar la tarea (tiempo de ejecución, latencia, tiempo de respuesta, u otras derivadas).
    - Tareas completadas por unidad de tiempo (throughput, ancho de banda)
  - Tiempo de respuesta y throughput a veces conflictúan entre sí
  - Los sistemas paralelos suelen ser sistemas dedicados, que priorizan las métricas basadas en tiempo.

# Métricas - Tiempo de ejecución

- Un algoritmo secuencial se suele evaluar por su tiempo de ejecución → En general, es posible encontrar alguna ley asintótica del tiempo de ejecución en función del tamaño de datos de entrada
- El tiempo de ejecución de un programa paralelo no sólo depende del tamaño de los datos de entrada sino también del número de procesadores y de los parámetros de comunicación de la arquitectura de soporte → es incorrecto analizar el algoritmo paralelo en forma aislada

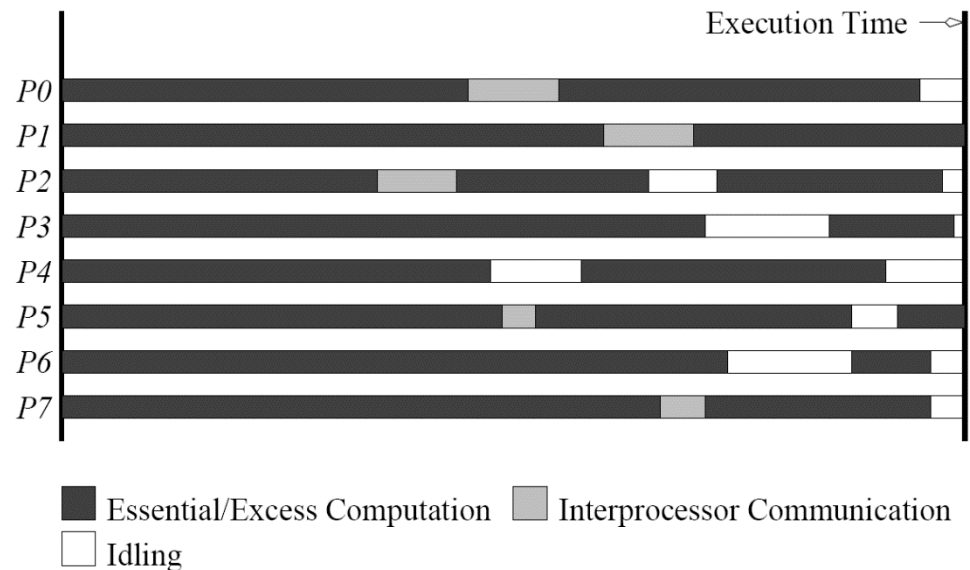
*El análisis se debe realizar a nivel de **sistema paralelo** (combinación de algoritmo paralelo y contexto de hardware y software).*

# Métricas - Tiempo de ejecución

- El ***tiempo de ejecución secuencial*** ( $T_s$ ) es el tiempo que transcurre desde el inicio hasta el fin de la ejecución sobre una máquina empleando una única unidad de procesamiento.
- El ***tiempo de ejecución paralela*** ( $T_p$ ) resume la diferencia de tiempo entre que la primera tarea que comienza hasta que la última tarea haya completado su trabajo.

# Fuentes de overhead

- Usando el doble de recursos, se espera que el programa paralelo se ejecute en la mitad del tiempo → Sin embargo, en la práctica, esto es muy raro que ocurra.
- Existen factores que generan overhead en los programas paralelos e impiden una mejora proporcional al aumento de la arquitectura:
  - Ocio
  - Interacción entre procesos
  - Cómputo adicional



# Métricas – Speedup

- El **Speedup** (**S**) refleja el beneficio de emplear procesamiento paralelo para resolver un problema dado comparado a realizarlo en forma secuencial

$$S_p(n) = \frac{T_s(n)}{T_p(n)}$$

- Es una medida de cuántas veces más rápido pudimos resolver el problema empleando el algoritmo paralelo con  $p$  unidades de procesamiento comparado al algoritmo secuencial.
- Para un problema dado, pueden existir diferentes algoritmos secuenciales, los cuales pueden tener diferentes tiempo de ejecución (complejidad) y, a su vez, pueden ser paralelizados de distintas maneras.
- Para computar el Speedup, siempre se debe considerar **el mejor algoritmo secuencial** (el que resuelva el problema en menos tiempo).



# Métricas – Speedup

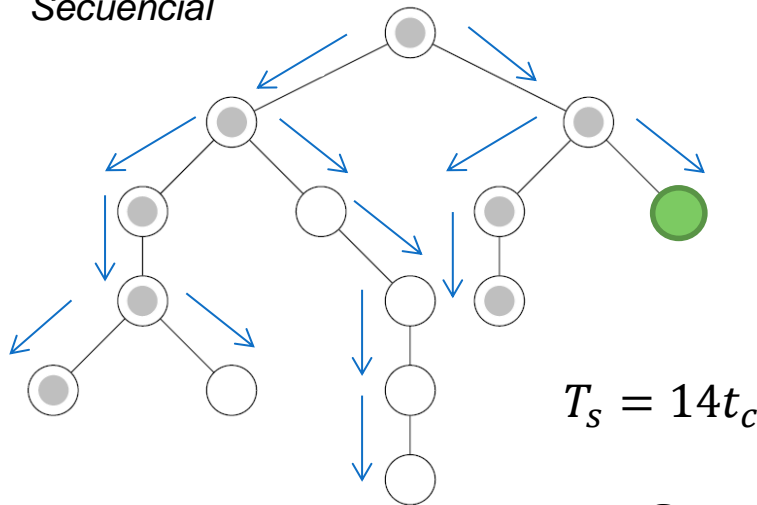
- Límites del Speedup:

- Si  $S_p(n) < 1$  entonces el algoritmo paralelo tarda más que el mejor algoritmo secuencial  $\rightarrow S_p(n)$  debe ser mayor a 1
- El mejor resultado se logra si somos capaces de distribuir el trabajo entre las unidades de procesamiento sin introducir ocio, interacción ni cómputo adicional  $\rightarrow$  Situación poco usual
  - Con  $p$  unidades de procesamiento  $\rightarrow S_p(n) = p$  (conocido como *Speedup lineal*, *Speedup óptimo*, *Speedup perfecto*)
- Teóricamente, siempre se cumple que  $S_p(n) \leq p$ 
  - Un Speedup mayor a  $p$  sólo es posible si cada unidad de procesamiento requiere menos de  $\frac{T_s(n)}{p}$  unidades de tiempo
  - Entonces podríamos construir un nuevo algoritmo secuencial que emule las  $p$  unidades de procesamiento usando una única unidad física, resolviendo el problema en menos de  $T_s$  unidades de tiempo  $\rightarrow$  **Contradicción**

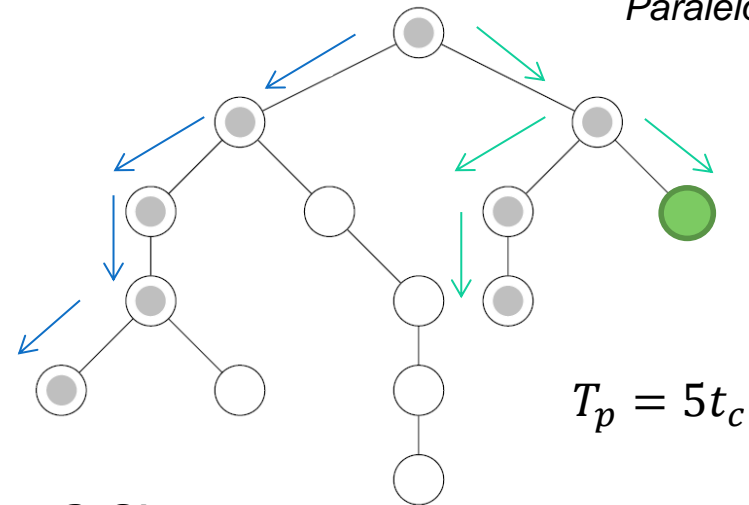
# Métricas – Speedup

- En la práctica, a veces se puede dar  $S_p(n) > p$  (fenómeno conocido como *Speedup superlineal*)
  - Un motivo puede ser que la versión paralela del algoritmo realice *menos* trabajo que la versión secuencial
  - Ejemplo: búsqueda *Depth-First-Search* en árbol binario, donde expandir cada nodo cuesta  $t_c$

Secuencial



Paralelo ( $p=2$ )



$$S_p = \frac{14t_c}{5t_c} = 2.8!$$

# Métricas – Speedup

- Un segundo motivo de Speedup superlineal es la combinación de características de hardware y distribución de los datos del algoritmo paralelo que ponen en desventaja al algoritmo secuencial
- Por ejemplo, consideremos un procesador con 64Kb de caché y un algoritmo secuencial con 80% de hits en la caché:
  - Si el algoritmo paralelo emplea 2 procesadores, disminuye el volumen de datos con los que se trabaja, llevando la tasa de hits al 90%
  - Siguiendo este razonamiento, al emplear 4 procesadores, se podría alcanzar una tasa de 100% de hits.

# Métricas – Speedup

- Calculemos las relaciones para 20.000 accesos a memoria considerado que:
  - la latencia de la caché es 2 ns;
  - el tiempo de acceso a memoria es 100 ns;
  - el tiempo de acceso a disco es 1000 ns;
  - los fallos de caché se reparten 80% en RAM y 20% en disco.

**Secuencial**  $\rightarrow 16.000 \times 2 \text{ ns} + 3.200 \times 100 \text{ ns} + 800 \times 1000 \text{ ns} = 1152 \text{ ms}$

**Paralelo ( $p=2$ )**  $\rightarrow 18.000 \times 2 \text{ ns} + 1.600 \times 100 \text{ ns} + 400 \times 1000 \text{ ns} = 0,596 \text{ ms}$

**Paralelo ( $p=4$ )**  $\rightarrow 20.000 \times 2 \text{ ns} = 0,04 \text{ ms} \rightarrow$

$$s_p = \frac{1152}{0,4} = 2880$$

*¿Es una causa real de speedup superlineal?*



# Métricas – Speedup

- Lo visto hasta ahora asume que todas las unidades de procesamiento empleadas son idénticas
- En arquitecturas heterogéneas, el Speedup se debe calcular considerando la Potencia Cómputo Total ( $pct$ ) en lugar del número de unidades de procesamiento ( $p$ )

$$pct = \sum_{i=0}^{p-1} pcr_i$$

$$pcr_i = \frac{p_i}{p_m}$$

$pct$	Potencia de Cómputo Total
$pcr$	Potencia de Cómputo Relativa
$p_i$	Potencia del procesador $i$
$p_m$	Potencia del mejor procesador

# Métricas – Speedup

- Consideremos una arquitectura compuesta por 8 procesadores ( $p_0..p_7$ ), donde  $p_0$  es el de mayor potencia,  $p_1..p_4$  tienen un 75% de la potencia de  $p_0$  y  $p_5..p_7$  tienen el 50% de potencia de  $p_0$ ; entonces:

$$p_{cr_0} = 1$$

$$p_{cr_{1..4}} = 0,75$$

$$p_{cr_{5..7}} = 0,5$$

$$p_{ct} = \sum_{i=0}^7 p_{cr_i} = 1 + 4 \times 0,75 + 3 \times 0,5 = 5,5$$

- El límite superior para el Speedup en esta arquitectura es 5,5 más allá de contar con 8 procesadores

*¿Cómo calcular potencia de cada unidad de procesamiento?*

# Métricas – Eficiencia

- Sólo un sistema paralelo ideal con  $p$  unidades de procesamiento puede reportar speedups iguales a  $p \rightarrow$  En la práctica es difícil que ocurra debido a las fuentes de overhead
- La **Eficiencia** es una medida de la fracción de tiempo en la cual las unidades de procesamiento son empleadas en forma útil

$$E_p(n) = \frac{S_p(n)}{S_{opt}}$$

- En arquitecturas homogéneas  $S_{opt} = p$  mientras que en heterogéneas  $S_{opt} = pct$
- Si  $S_p(n) = p$  (sistema paralelo ideal), entonces  $E_p(n) = 1$
- En la práctica,  $S_p(n) \leq p$  lo que implica que  $E_p(n) \leq 1$
- Por definición,  $E_p(n) > 0$ . Por lo tanto,  $0 < E_p(n) \leq 1$

# Métricas – Overhead total

- El **overhead total** de un sistema paralelo se define como la diferencia entre la suma del tiempo requerido por todas las unidades de procesamiento y el del mejor algoritmo secuencial para resolver el mismo problema empleando una única de unidad de procesamiento.

$$OT_p(n) = pT_p(n) - T_s(n)$$



# Métricas – Overhead de las comunicaciones

- El ***overhead de las comunicaciones*** de un sistema paralelo se define como la relación entre el tiempo requerido por las comunicaciones de nuestra solución y el tiempo total que esta requiera.

$$OC_p(n) = \frac{T_{comm_p}(n)}{T_p(n)} \times 100$$

# Métricas – Ejemplos de Speedup, Eficiencia y Overhead

Tiempos de ejecución y valores de Speedup y Eficiencia para algoritmos que resuelven multiplicación de matriz-vector

	$p$	Order of Matrix				
		1024	2048	4096	8192	16,384
$T_s(n)$	1	4.1	16.0	64.0	270	1100
	2	2.3	8.5	33.0	140	560
$T_p(n)$	4	2.0	5.1	18.0	70	280
	8	1.7	3.3	9.8	36	140
	16	1.7	2.6	5.9	19	71

$S_p(n)$

$p$	Order of Matrix				
	1024	2048	4096	8192	16,384
1	-	-	-	-	-
2	1.8	1.9	1.9	1.9	2.0
4	2.1	3.1	3.6	3.9	3.9
8	2.4	4.8	6.5	7.5	7.9
16	2.4	6.2	10.8	14.2	15.5

$E_p(n)$

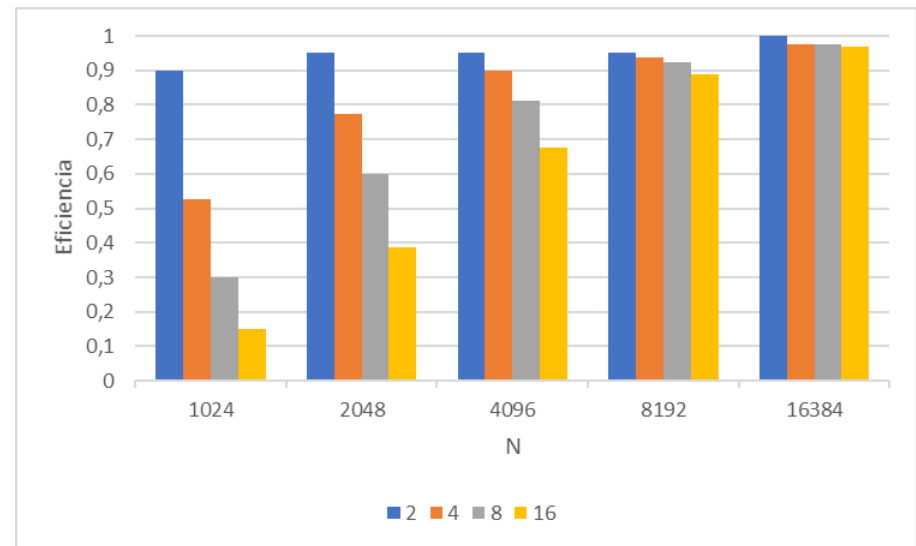
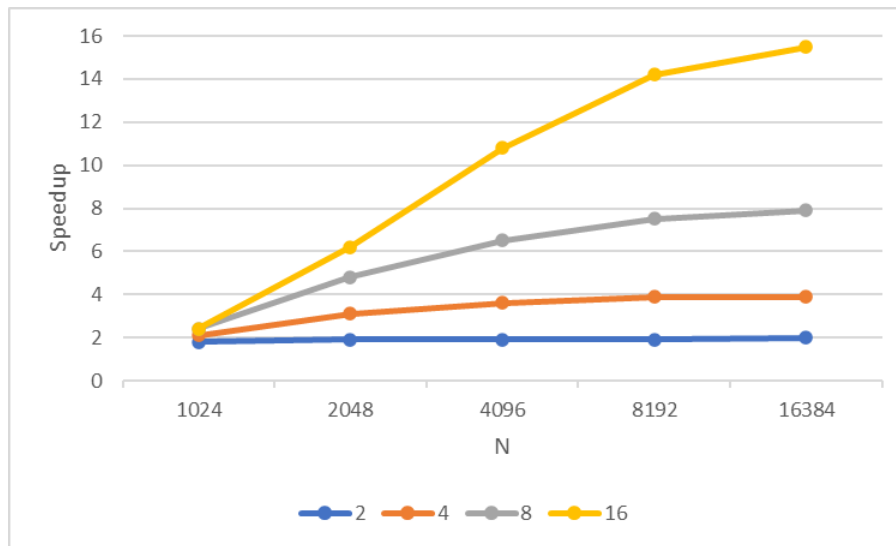
$p$	Order of Matrix				
	1024	2048	4096	8192	16,384
1	-	-	-	-	-
2	0.89	0.94	0.97	0.96	0.98
4	0.51	0.78	0.89	0.96	0.98
8	0.30	0.61	0.82	0.94	0.98
16	0.15	0.39	0.68	0.89	0.97

$T_s$ ,  $T_p$ ,  $S$  y  $E$  dependen del tamaño de problema

$T_p$ ,  $S$  y  $E$  además dependen de  $p$

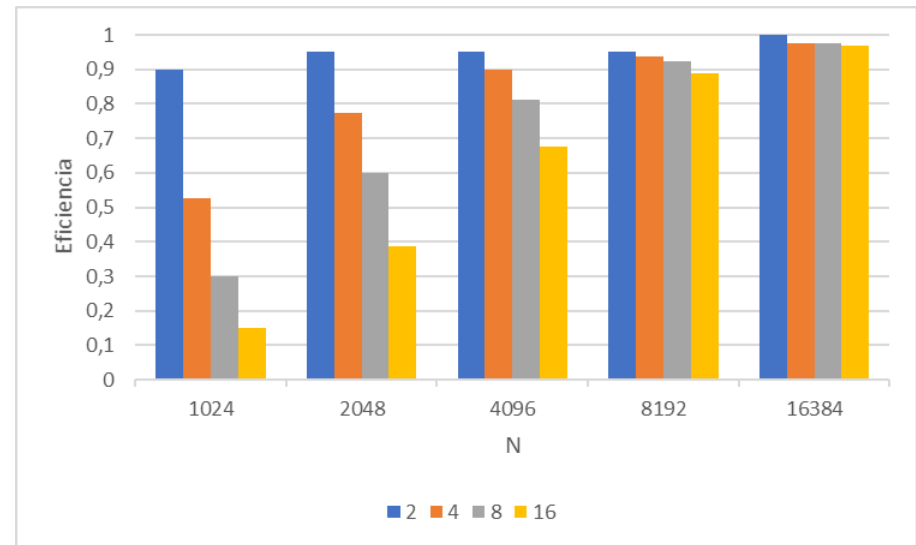
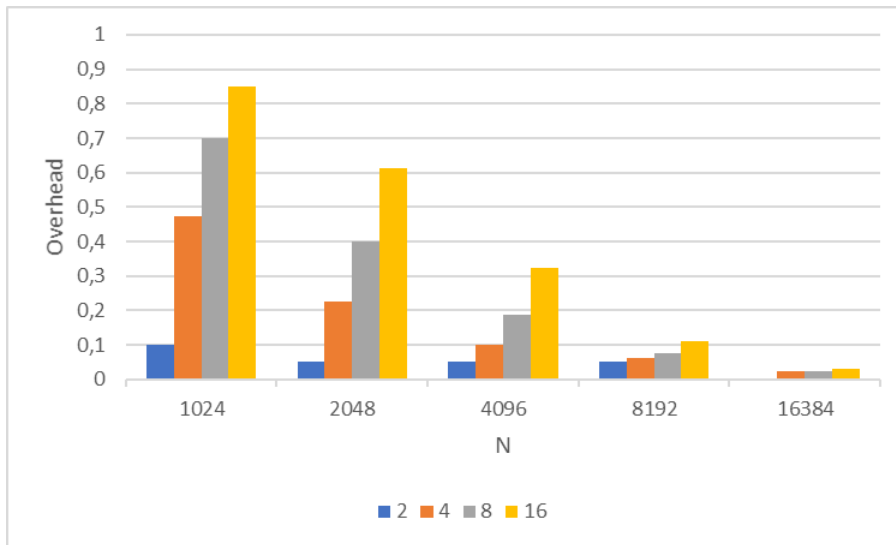
# Métricas – Ejemplos de Speedup, Eficiencia y Overhead

Gráficos de Speedup y Eficiencia para las tablas anteriores



# Métricas – Ejemplos de Speedup, Eficiencia y Overhead

Gráfico de Overhead (com) y Eficiencia para las tablas anteriores



# Métricas – Ley de Amdahl

- Los factores de overhead limitan los beneficios del procesamiento paralelo
- Una restricción importante proviene de aquellas secciones de código que no pueden ser paralelizadas → bloque de ejecución secuencial
- La **Ley de Amdahl** (Amdahl, 1967) permite estimar el Speedup alcanzable en aquellos programas paralelos que contienen partes secuenciales:
- Dada una fracción  $f$ ,  $0 \leq f \leq 1$ , de un programa paralelo que debe ser ejecutada secuencialmente, el tiempo de ejecución paralela se calcula como:

$$T_p(n) = f \times T_s(n) + \frac{(1 - f) \times T_s(n)}{p}$$

- Entonces el Speedup ahora puede re-escribirse de la siguiente forma:

$$S^A_P(n) = \frac{T_s(n)}{f \times T_s(n) + \frac{(1 - f) \times T_s(n)}{p}} = \frac{1}{f + \frac{(1 - f)}{p}}$$

# Métricas – Ley de Amdahl

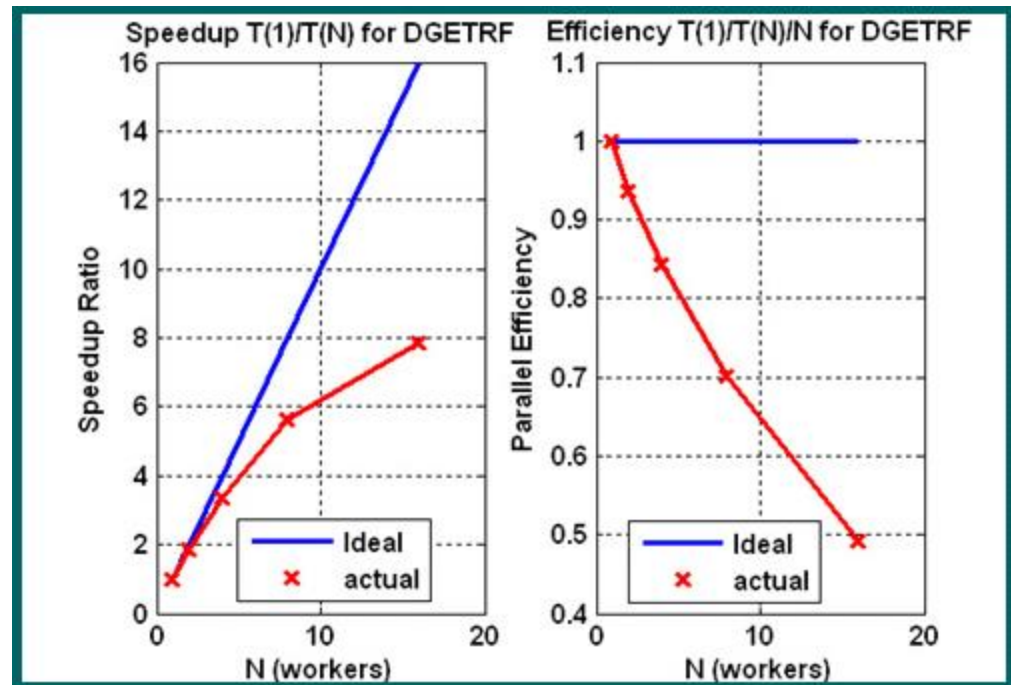
- Es importante notar que, aun con un número infinito de unidades de procesamiento, el Speedup estará limitado a  $\frac{1}{f}$ , ya que:

$$\lim_{p \rightarrow \infty} \left( \frac{1}{f + \frac{(1-f)}{p}} \right) = \frac{1}{f}$$

- Por ejemplo, con 5% de ejecución secuencial ( $f = 0.05$ ) en un programa paralelo, el máximo Speedup alcanzable será **20**, sin importar el número de unidades de procesamiento que podamos emplear.
- Es importante tener en cuenta esta características si vamos a emplear una gran cantidad de unidades de procesamiento, aunque veremos más adelante que esta estimación puede ser considerada *poco realista*.

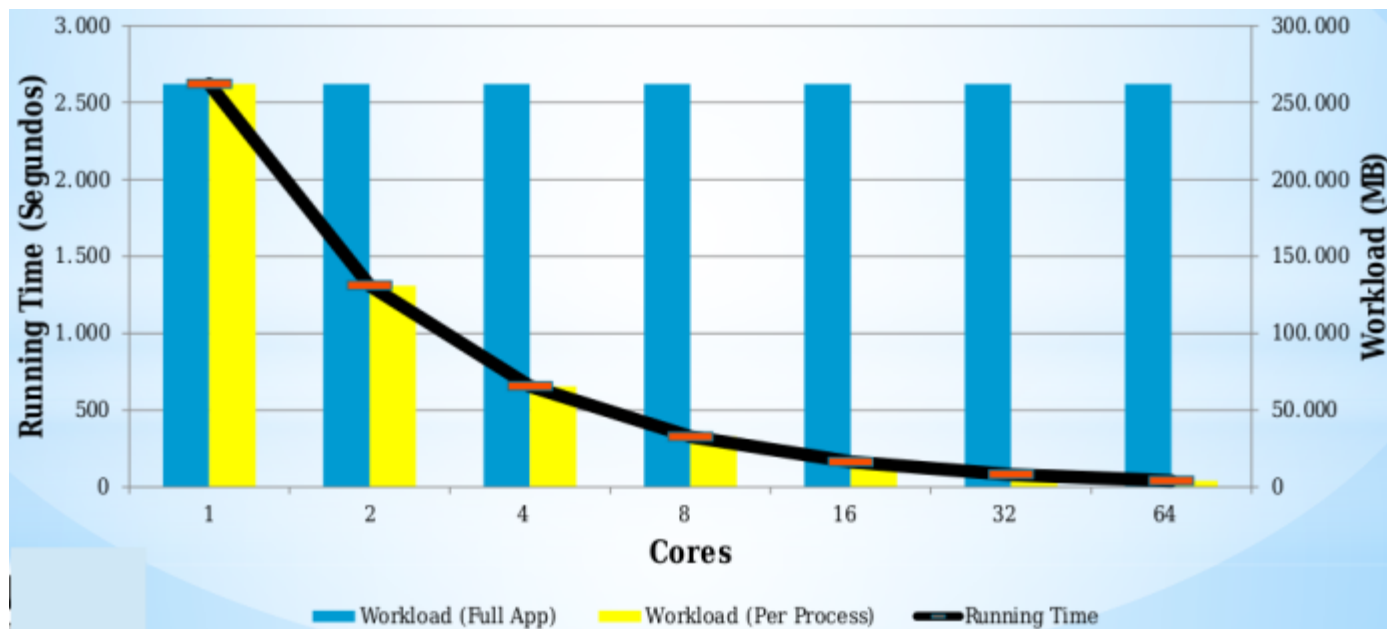
# Métricas – Escalabilidad

- En el contexto de análisis de rendimiento, la escalabilidad hace referencia a la capacidad que tiene un sistema de mantener un nivel de Eficiencia fijo al incrementar tanto el número de unidades de procesamiento como el tamaño del problema de resolver → En ese caso, se dice que el sistema es **escalable**
- Dicho de otra manera, la escalabilidad de un sistema paralelo es una medida de su capacidad de incrementar el Speedup en forma proporcional al número de unidades de procesamiento empleadas



# Métricas – Escalabilidad

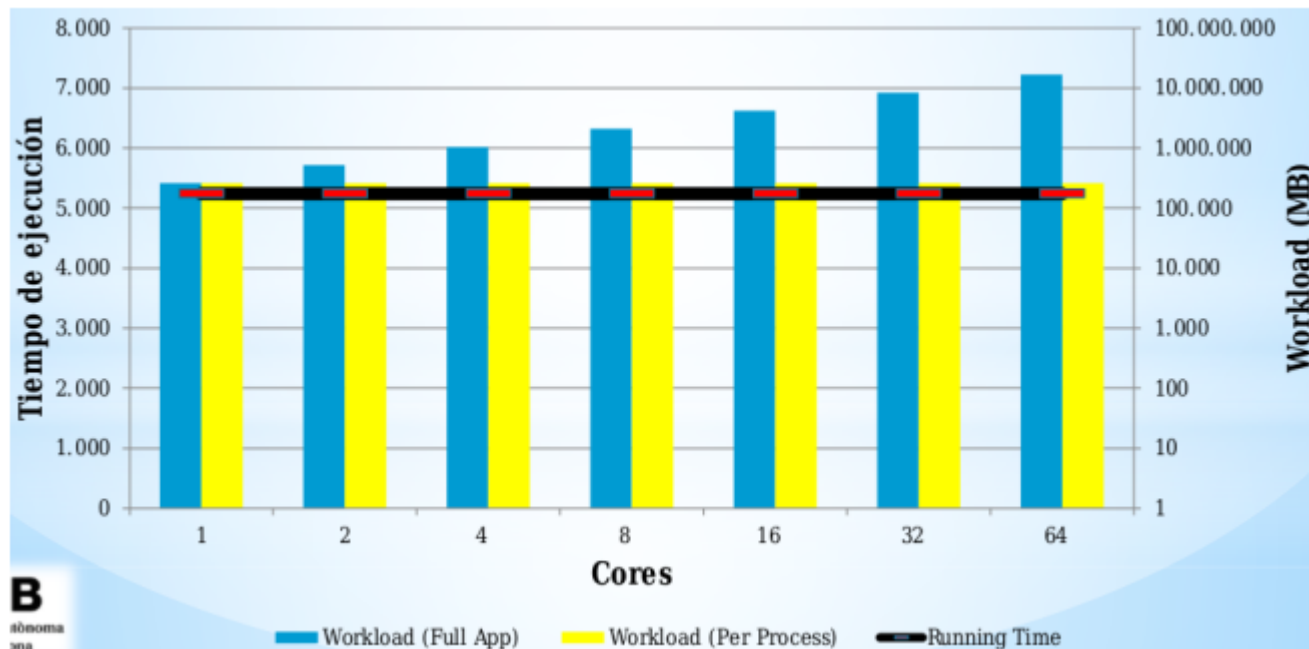
- Casos especiales:
  - *Escalabilidad fuerte*: Cuando al incrementar el número de unidades de procesamiento, no resulta necesario aumentar el tamaño de problema para mantener la eficiencia en un valor fijo.





# Métricas – Escalabilidad

- Casos especiales:
  - *Escalabilidad débil*: Cuando al incrementar el número de unidades de procesamiento, resulta necesario también aumentar el tamaño de problema para mantener la eficiencia en un valor fijo.



# Métricas – Ley de Gustafson

- El incremento en el Speedup por un tamaño mayor de problema no es percibido por la Ley de Amdahl

$$\lim_{p \rightarrow \infty} \left( \frac{1}{f \times + \frac{(1-f)}{p}} \right) = \frac{1}{f}$$

# Métricas – Ley de Gustafson

- El incremento en el Speedup por un tamaño mayor de problema no es percibido por la Ley de Amdahl
- En la década del 80, Gustafson observó en la práctica que:
  - Un multiprocesador más grande usualmente permite resolver un problema de mayor tamaño en un tiempo de ejecución determinado (*escalabilidad*)  
→ el tamaño de problema seleccionado depende frecuentemente del número de unidades de procesamiento disponibles
  - Al incrementar el tamaño del problema y el número de unidades de procesamiento para mantener el tiempo de ejecución constante, la fracción secuencial de los programas se mantiene fija o no crece en forma proporcional al tamaño de la entrada.
- Por lo tanto, asumir que el tamaño de problema es fijo resulta tan válido como que el tiempo de ejecución paralela lo es

# Métricas – Ley de Gustafson

- Basándose en sus observaciones, Gustafson re-escribió la ecuación para estimar el máximo speedup alcanzable (conocido como *Speedup escalado*).
- Dada una fracción  $f'$ ,  $0 \leq f' \leq 1$ , de un programa paralelo que debe ser ejecutada secuencialmente pero que no crece en forma proporcional al tamaño de problema, el Speedup escalado se calcula como:

$$S_p^S(n) = \frac{T_s(n)}{T_p(n)} = \frac{f' \times T_p(n) + (1 - f') \times T_p(n) \times p}{T_p(n)} = p + (1 - p) \times f'$$

- Esta versión requiere 2 suposiciones: (1)  $T_p(n)$  se mantiene constante y (2)  $f' \times T_p(n)$  no escala en forma proporcional al aumento de  $n$  y  $p$
- Según Gustafson: con 5% de ejecución secuencial ( $f' = 0.05$ ) y 20 procesadores, el speedup alcanzable sería  $20 + (1 - 20) \times 0,05 = \mathbf{19.05}$  en lugar de  $\frac{1}{0.05 \times + \frac{(1-0.05)}{20}} = \mathbf{10.26}$  que se obtendría con la ecuación de Amdahl.

# Métricas – Escalabilidad: resumen y comentarios finales

- Escalabilidad fuerte (Amdhal)

- El tamaño del problema se mantiene fijo a medida que se incrementa la cantidad de procesadores.
- El objetivo es resolver el mismo problema de forma más rápida
- El *escalado perfecto* se logra cuando el problema se resuelve en  $1/P$  unidades de tiempo (comparado al secuencial)

- Escalabilidad débil (Gustafson)

- El tamaño del problema *por procesador* se mantiene fijo a medida que se incrementa la cantidad de procesadores  
→ El tamaño total del problema es proporcional al número de procesadores usados.
- El objetivo es resolver un problema más grande en la misma cantidad de tiempo
- El *escalado perfecto* se logra cuando se resuelve un problema  $P$  veces más grande en la misma cantidad de tiempo que el secuencial.

# Métricas – Escalabilidad: resumen y comentarios finales

- La capacidad de un programa paralelo de escalar su rendimiento depende de un conjunto de factores interrelacionados → Agregar más procesadores no suele ser una solución directa.
- Los algoritmos pueden tener límites inherentes para su escalabilidad.
  - Incluso agregar más recursos puede perjudicar el rendimiento en un determinado punto.
  - Es una situación usual en varias aplicaciones paralelas.
- Los recursos de hardware desempeñan un factor fundamental en la escalabilidad. Ejemplos:
  - Ancho de banda del bus de memoria en un multiprocesador
  - Ancho de banda de red
  - Memoria principal disponible en una máquina (o un conjunto de ellas)
  - Frecuencia del reloj del procesador

# Métricas – Desbalance de carga

- En arquitecturas heterogéneas o soluciones paralelas que contemplan problemas irregulares, resulta útil poder medir el ***Desbalance de carga***:

$$D = \frac{\max_{i=0..p-1} (T_{pi}(n)) - \min_{i=0..p-1} (T_{pi}(n))}{\text{prom}_{i=0..p-1} (T_{pi}(n))}$$

- Si todas las unidades de procesamiento toman el mismo tiempo, entonces  $D = 0 \rightarrow$  Poco usual
- En general, se debe intentar que D esté lo más cerca posible de 0

# Recomendaciones a la hora de optimizar

- Tener en cuenta el principio de Pareto aplicado a la ingeniería de software:

*90% del tiempo de ejecución se dedica a ejecutar 10% del código*

- También conocida como “regla 90/10”.

Two independent parts **A** **B**

Original process



*¿Qué será más beneficioso? ¿Acelerar 5x la parte B? ¿o 2x la parte A?*



# Recomendaciones a la hora de optimizar

- Tener en cuenta el principio de Pareto aplicado a la ingeniería de software:

*90% del tiempo de ejecución se dedica a ejecutar 10% del código*

- En este contexto, se conoce como “regla 90/10”.

Two independent parts **A** **B**

Original process



Make **B** 5x faster



# Recomendaciones a la hora de optimizar

- Tener en cuenta el principio de Pareto aplicado a la ingeniería de software:

*90% del tiempo de ejecución se dedica a ejecutar 10% del código*

- En este contexto, se conoce como “regla 90/10”.

Two independent parts **A** **B**



# Recomendaciones a la hora de optimizar

- En programas pequeños, uno puede identificar fácilmente donde está sección de código que más demanda computacional tiene (a esto se lo conoce como *hotspot*).
  - Observando y analizando el código (habitualmente los bucles)
  - Instrumentando el código “a mano” (tomar tiempos de diferentes secciones)
- En programas de complejidad mediana o alta, se realiza un proceso llamado *profiling* (o perfilado).
- Mediante herramientas de software, se analiza y mide el rendimiento de una aplicación o programa → el objetivo principal es identificar los puntos críticos de la aplicación que consumen la mayor cantidad de tiempo de ejecución o recursos del sistema.
- Hay muchos profilers diferentes, según el lenguaje, el aspecto a mejorar, la modalidad de uso, entre otros
- C (gprof, Valgrind, VTune), Python (cProfile, profile), Julia (profile)

# Recomendaciones para medir tiempos de ejecución

- Usualmente interesa analizar la mejora lograda en determinada parte del programa → En la práctica, el tiempo de ejecución no siempre se considera desde el que programa empieza hasta que el mismo termina
- Por ejemplo, si consideramos un problema de ordenación por burbuja, mediremos la parte que ordena el vector, pudiendo descartar:
  - Reserva de memoria
  - Lectura de datos de entrada
  - Impresión de datos de salida
  - Liberación de memoria

# Recomendaciones para medir tiempos de ejecución

- Aunque la solución paralela involucre  $p$  tareas,  $T_p(n)$  debe ser un único valor que contemple el tiempo que transcurre desde que la primera tarea comenzó a ejecutar hasta que la última haya completado su trabajo → Tener en cuenta que este tiempo puede hacer referencia a una determinada parte del programa
- En ocasiones, no todas las tareas comienzan y terminan al mismo tiempo → Para asegurarnos una medición correcta, puede ser útil emplear *barreras*
- La precisión en la medición puede ser un tema importante → No es lo mismo medir un programa que dure unos pocos segundos a otro que dure horas...

# Recomendaciones para medir tiempos de ejecución

- Otro factor a tener en cuenta es la variabilidad en las mediciones
  - Se recomienda repetir las pruebas un determinado número de veces y calcular el promedio o la mediana → ¿Cuántas veces repetirlo?
  - En ocasiones, también puede ser de interés reportar el tiempo mínimo y el máximo

# Bibliografía usada para esta clase

- Capítulo 5, An Introduction to Parallel Computing. Design and Analysis of Algorithms (2da Edition). Grama A., Gupta A., Karypis G. & Kumar V. (2003) Inglaterra: Pearson Addison Wesley.
- Capítulo 4, Parallel Programming for Multicore and Cluster Systems. Rauber, T. & Rünger, G. (2010). EEUU: Springer-Verlag Berlin Heidelberg.
- Capítulo 2, An Introduction to Parallel Programming. Pacheco, P. (2011) EEUU: Elsevier.
- Capítulo 1, Parallel Programming. Wilkinson, B. & Allen, M. (2005) EEUU, Pearson