

Sistemas Distribuidos y Paralelos

Ingeniería en Computación



Métricas

Universidad Nacional de La Plata



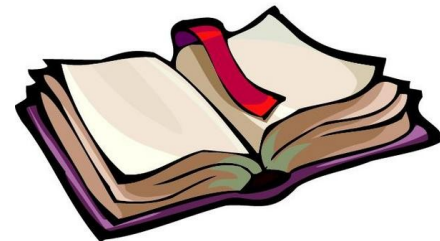
Facultad de Informática



Agenda

2

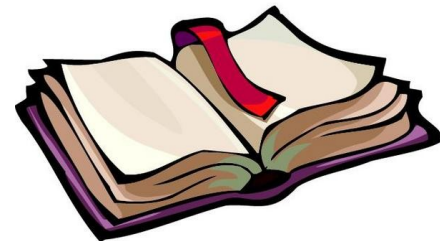
- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica



Agenda

3

- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica



Métricas

4

- Utilizamos métricas para evaluar el comportamiento de un sistema.
- En particular, en un Sistema Paralelo, interesa saber cual es la ganancia que se obtiene respecto a la ejecución de la solución secuencial.
- Generalmente, las métricas están asociadas al **rendimiento global** (o **throughput**) utilizando el **tiempo** como unidad de medida. Es decir, **throughput** es la cantidad de trabajo u operaciones que se pueden completar en un período de tiempo determinado (Flops, Mips, Mbps, etc.).



Análisis de algoritmos secuenciales

5

- Habitualmente, analizamos **de manera teórica** el comportamiento asintótico de un **algoritmo secuencial**, a partir de una estimación del tiempo de ejecución como función del tamaño del problema ($T(n)$).
- El comportamiento asintótico del $T(n)$ es idéntico en cualquier plataforma:

$$O(n) \quad O(n^x) \quad O(\log n) \quad O(n \log n) \quad O(2^n)$$

- Sin embargo, en la **práctica**, algoritmos distintos que resuelven el mismo problema y tienen el mismo comportamiento asintótico, pueden alcanzar tiempos de ejecución diferentes en la misma arquitectura debido a otros factores (Por ejemplo: Localidad).



Análisis de algoritmos paralelos

6

- Podemos analizar **teóricamente** un **algoritmo paralelo**, aunque debe evaluarse considerando el tamaño del problema y otras características:
 - Número de unidades de procesamiento
 - Sincronización/Comunicación
 - Balance de carga
- Eventualmente, un sistema paralelo también debe analizarse teniendo en cuenta el contexto de sistema operativo y hardware subyacente.
- Pueden surgir preguntas como:
 - ¿Cuál es la ganancia que se obtiene? ¿Siempre se obtiene una ganancia?
 - ¿Qué indica que un sistema paralelo es mejor que otro?
 - ¿Cuál es la mejora al aumentar el número de unidades de procesamiento?
 - ¿Cuál es la mejora al aumentar el volumen de cómputo?



Métricas

Tiempo de ejecución

7



- En nuestro caso, evaluaremos nuestros algoritmos paralelos de forma **empírica** basándonos en el **tiempo de ejecución**:



- Tiempo de ejecución del algoritmo secuencial (T_s)**: Tiempo que transcurre entre el comienzo y la finalización de la ejecución del programa secuencial
- Tiempo de ejecución del algoritmo paralelo (T_p)**: Tiempo que transcurre entre el comienzo y la finalización del último proceso/hilo del programa paralelo
- En la práctica, podemos medir el tiempo de ejecución de al menos dos formas:
 - A partir de llamadas al sistema operativo que interactúan con el **reloj del sistema**
 - Usando métricas asociadas al hardware:

$$T_s = RI \cdot CPI \cdot t_{ciclo}$$

$$T_p = \frac{RI}{P} \cdot \frac{1}{IPC} \cdot t_{ciclo}$$

RI: Cantidad de instrucciones ejecutadas

CPI: Ciclos promedio por instrucción

t_{ciclo} : Duración del ciclo de reloj

IPC: Instrucciones por ciclo de reloj

P: Número de unidades de procesamiento



Inicialmente, vamos a suponer un sistema paralelo donde todas las unidades de procesamiento son idénticas, es decir un sistema con arquitecturas homogéneas.

Métricas

Recolección de muestras

8

- Para obtener las métricas debemos ejecutar los algoritmos y obtener el tiempo de ejecución.
- Se debe tener en cuenta que dos o más ejecuciones del mismo algoritmo pueden arrojar tiempos de ejecución ligeramente diferentes.
- Para mitigar esta variabilidad, debemos correr el algoritmo varias veces y obtener el tiempo de ejecución promedio.
- Sin embargo, existen algoritmos donde una sola ejecución puede llegar a demorar horas.



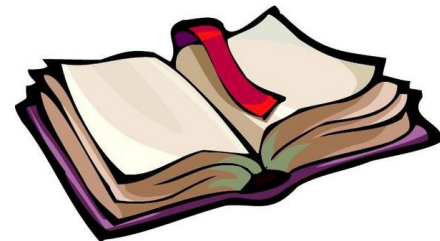
- Alternativas:

- Se acepta ejecutarlo una única vez.
- Estimar el tiempo de ejecución sin ejecutarlo (firma del código).

Agenda

9

- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica



Speedup

10

- El **Speedup** es una métrica de rendimiento que representa el beneficio relativo alcanzado al ejecutar un programa.
- Suponer un programa al cual se le hizo alguna mejora.
- El beneficio alcanzado o **Speedup** se calcula:

$$S = \frac{T_a}{T_b}$$

- T_a : tiempo antes de la mejora
- T_b : tiempo después de la mejora



Speedup

11

- Para obtener el beneficio de paralelizar una solución secuencial, el **Speedup** se calcula como:


$$S = \frac{T_s}{T_p}$$

- T_s : tiempo de ejecución del algoritmo secuencial
 - T_p : tiempo de ejecución del algoritmo paralelo utilizando **p unidades de procesamiento**
- Según el algoritmo secuencial utilizado el **speedup** es:
 - **Absoluto:** si T_s es el tiempo de ejecución del mejor programa secuencial
 - **Relativo:** si T_s es el tiempo de ejecución de un programa secuencial que no es el mejor o el tiempo de ejecución del programa paralelo ejecutado con un único proceso/hilo



Speedup lineal o perfecto

12

- En el **peor caso**, no obtenemos ningún beneficio al paralelizar entonces $S < 1$. 
- En el **mejor caso**, podemos distribuir el trabajo entre las **P unidades de procesamiento** de forma equitativa y el programa paralelo correrá P veces más rápido que el programa secuencial. El tiempo de ejecución paralelo será:

$$T_p = \frac{T_s}{P}$$

- Por lo tanto, el **speedup** será:

$$S = \frac{T_s}{T_p} = \frac{T_s}{\frac{T_s}{P}} = P$$

- Esto se conoce como **Speedup lineal o perfecto**



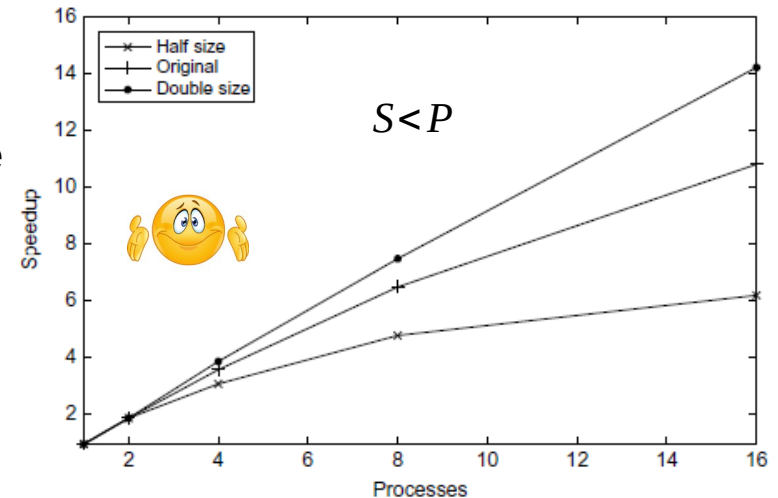
Speedup Limitaciones

13

- En la práctica, es poco probable obtener un speedup perfecto
- Las aplicaciones paralelas introducen overhead:
 - Startup (creación e inicialización de procesos/hilos)
 - Sincronización:
 - Por exclusión mutua (Regiones críticas)
 - Por condición:
 - Esperas por resultados
 - Esperas ociosas
 - Barreras
 - Comunicación
 - Desbalance de carga (distribución de trabajo no uniforme)
 - Cuellos de botella (Acceso a memoria – E/S)
 - Contención
 - Cómputo redundante

Características que no posee un algoritmo secuencial !!!

P	1	2	4	8	16
Half	1	1.9	3.1	4.8	6.2
Original	1	1.9	3.6	6.5	10.8
Double	1	1.9	3.9	7.5	14.2



Speedup superlineal

14

- Sin embargo, existen situaciones en que el **speedup** es mayor que un speedup perfecto:



$$S > P$$

- Esto se conoce como **speedup superlineal** y puede darse en los siguientes casos:

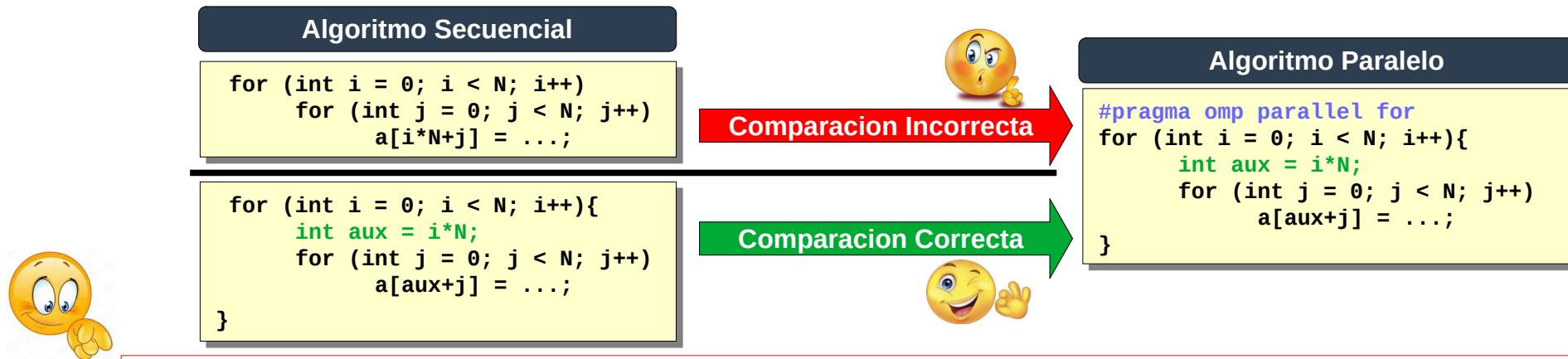


- El algoritmo secuencial no es el mejor algoritmo secuencial
- La versión paralela del algoritmo realiza menos trabajo que la versión secuencial
- Como consecuencia de la arquitectura

Speedup superlineal

Ejemplo: el algoritmo secuencial no es el mejor

- El algoritmo secuencial no es el mejor algoritmo secuencial.
- Puede ocurrir ante las siguientes situaciones:
 - Diferentes estrategias: No contamos con el código del mejor algoritmo secuencial y paralizamos sin saber la estrategia del mejor algoritmo secuencial.
 - Contamos con el código del algoritmo secuencial, paralelizamos en base a este, optimizamos el código “secuencial” del algoritmo paralelo pero no lo hacemos en la versión secuencial. Por lo tanto, omitimos que el algoritmo secuencial puede mejorarse y termina no siendo el mejor.



En problemas “regulares” es extraño o imposible obtener speedups superlineales.
Si obtenemos Speedup Superlineal, el problema suele estar en el algoritmo secuencial y no en el algoritmo paralelo!!!

Speedup superlineal

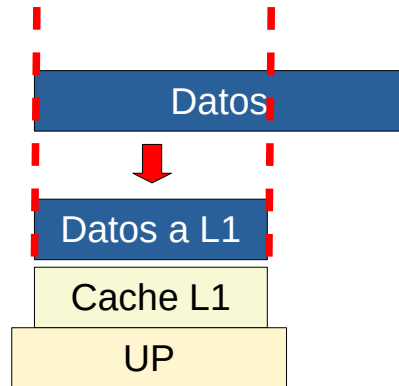
Ejemplo: consecuencia de la arquitectura

17

- El algoritmo paralelo puede utilizar de manera diferente la memoria cache respecto al algoritmo secuencial:

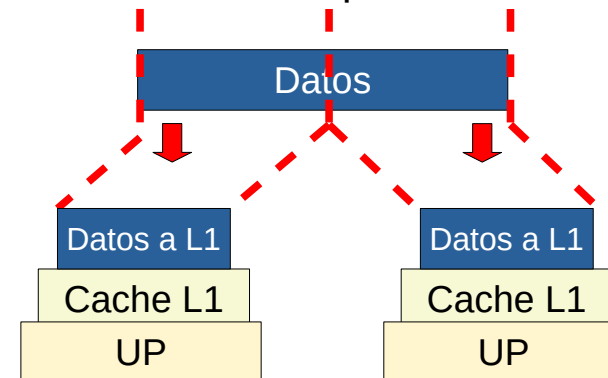
Algoritmo Secuencial

El volumen de datos a procesar podría no almacenarse por completo en la memoria cache.



Algoritmo Paralelo

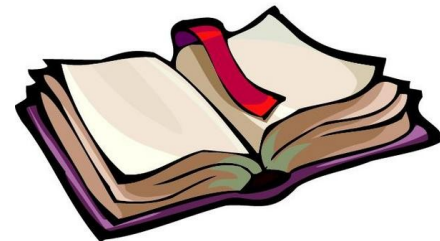
El volumen de datos a procesar se divide en porciones entre las unidades de procesamiento y cada porción podría almacenarse por completo en las caches de cada unidad de procesamiento.



Agenda

18

- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica



- La **eficiencia** es una métrica de rendimiento que indica el porcentaje de tiempo en el que las unidades de procesamiento están realizando trabajo útil:

$$E = \frac{S}{P}$$



- S : Speedup
- P : número de **unidades de procesamiento**

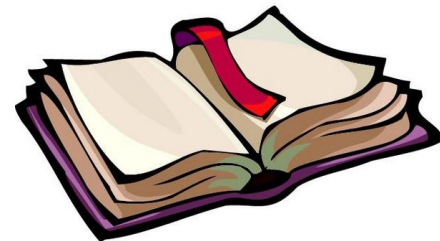
- De acuerdo a los valores de Speedup (lineal):

$$0 < E < 1$$

Agenda

20

- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica

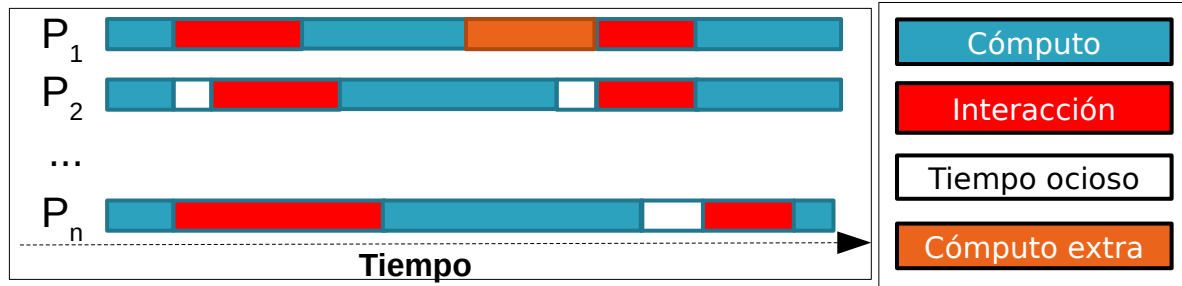


Overhead

21

- En un algoritmo paralelo existen distintas fuentes de overhead:

- Ociosidad de los procesos
- Interacción entre procesos
- Cómputo extra



- El overhead se calcula como:

$$T_o = P \cdot T_p - T_s$$

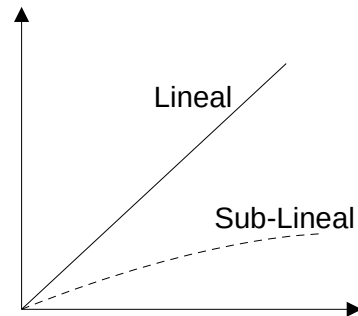
- El óptimo u overhead mínimo ocurre cuando tenemos un speedup perfecto. En ese caso $T_o = 0$:

$$T_p = \frac{T_s}{P} \quad \longrightarrow \quad T_o = P \cdot T_p - T_s = P \cdot \frac{T_s}{P} - T_s = 0$$

- Generalmente, el **overhead tiende a crecer**:
 - **Linealmente** en función del número de **unidades de procesamiento (P)**:
 - Cada proceso/hilo introduce una parte intrínsecamente lineal no paralelizable
 - Desbalance de carga con unidades de procesamiento ociosas
 - Aumento de las comunicaciones entre unidades de procesamiento
 - **Sub-linealmente** en función del tamaño del problema o **carga de trabajo (W)**
 - Esto implica que T_o será menor cuanto mayor sea W
- Por lo tanto, el overhead puede escribirse como función del número de unidades de procesamiento P y de la carga de trabajo W :



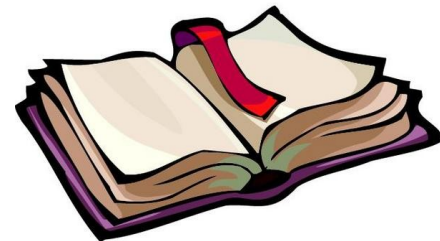
$$T_o(W, P) = P \cdot T_p - W$$



Agenda

23

- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica



Balance de carga

24

- Existen situaciones en que se producen desbalances de carga: procesos o hilos que realizan mayor (o menor) cantidad de trabajo que otros.
- El balance de carga es un porcentaje que se mide como:

$$B = \frac{\textit{Promedio}(T)}{\textit{MAX}(T)}$$

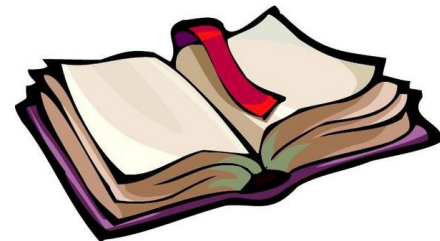


- ***Promedio (T)***: promedio de los tiempos de ejecución de los procesos/hilos.
- ***MAX(T)***: tiempo de ejecución del proceso/hilo que más tardó.
- La carga estará más balanceada cuando la métrica tienda a 1
- La carga estará más desbalanceada cuando la métrica tienda a 0

Agenda

25

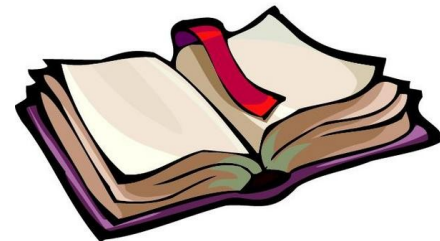
- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica



Agenda

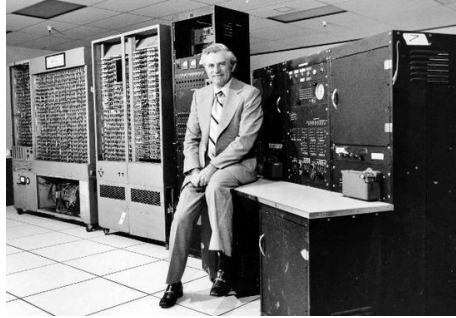
26

- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica



Ley de Amdahl

27



- En la década de 1960 Gene Amdahl hizo una observación que se conoce como la ley de Amdahl:

G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In AFIPS Conference Proceedings, vol. 30, 483–485, AFIPS Press, Reston, VA, 1967.

- **Ley de Amdahl:** “A menos que prácticamente todo un programa secuencial esté paralelizado, el speedup posible va a estar muy limitado independientemente del número de **procesadores*** disponibles”

*Léase procesadores como unidades de procesamiento, en términos modernos.

Ley de Amdahl

Ejemplo

28

Supongamos que paralelizamos el 90% de un programa secuencial.
Además, suponemos una paralelización perfecta (el **Speedup** de esa parte es P).

- Supongamos que el tiempo de ejecución secuencial es: $T_s = 20 \text{ seg}$

- El tiempo de ejecución de la parte paralelizada:

$$T_{\text{parte paralelizada}} = \frac{0,9 \cdot T_s}{p} = \frac{0,9 \cdot 20 \text{ seg}}{p} = \frac{18}{p} \text{ seg}$$

- El tiempo de ejecución de la parte NO paralelizada:

$$T_{\text{parte NO paralelizada}} = 0,1 \cdot T_s = 0,1 \cdot 20 \text{ seg} = 2 \text{ seg}$$

Ley de Amdahl

Ejemplo

29

- El tiempo paralelo será entonces:

$$T_p = T_{\text{parte paralelizada}} + T_{\text{parte NO paralelizada}} = \frac{18}{P} + 2$$

- El Speedup será:

$$S = \frac{T_s}{T_p} = \frac{20}{\frac{18}{P} + 2}$$

- Por lo tanto:

$$\lim_{P \rightarrow \infty} S = \lim_{P \rightarrow \infty} \frac{20}{\frac{18}{P} + 2} = 10$$



Conclusión Amdahl: aunque paralelizamos perfectamente el 90% del programa, e incluso si tenemos, por ejemplo 1000 unidades de procesamiento, nunca conseguiremos un speedup mejor que 10!!!

Ley de Amdahl

Formalización

30

- En general:
 - Si una fracción f de un programa secuencial permanece sin paralelizar o no puede ser paralelizada (inherentemente secuencial), entonces la ley de Amdahl dice que no podemos obtener una aceleración mejor que $\frac{1}{f}$
 - En nuestro ejemplo: $f = 1 - 0,9 = 0,1 = \frac{1}{10} = 0,1$
 - Por lo tanto no podríamos obtener un speedup mejor que $\frac{1}{f} = \frac{1}{0,1} = 10$
 - Si la fracción f fuera bastante pequeña, digamos $\frac{1}{100}$, y tenemos un sistema con miles de unidades de procesamiento, no podremos obtener una aceleración mejor que 100.



Ley de Amdahl

Formalización

31

- Sea $T^*(n)$ el tiempo de ejecución del mejor algoritmo secuencial para un problema de tamaño n .
- Sea f una fracción de un programa paralelo que debe ejecutarse secuencialmente.

$$0 \leq f \leq 1$$

- $f.T^*(n)$ es la fracción secuencial del tiempo de ejecución secuencial.
- $(1-f).T^*(n)$ es la fracción paralelizable del tiempo de ejecución secuencial.
 - Si se ejecuta completamente con P unidades de procesamiento es $\frac{(1-f)}{P} \cdot T^*(n)$



Ley de Amdahl

Formalización

32

- El **Speedup** es:

$$S_p(n) = \frac{T_s}{T_p} = \frac{T^*(n)}{\underbrace{f \cdot T^*(n)}_{\text{Fracción secuencial}} + \underbrace{\frac{(1-f)}{p} \cdot T^*(n)}_{\text{Fracción paralela con } p \text{ unidades de procesamiento}}} = \frac{1}{f + \frac{(1-f)}{p}} \leq \frac{1}{f}$$



Si el 20% de un programa debe ejecutarse secuencialmente entonces el máximo **Speedup** alcanzable está limitado a $\frac{1}{f} = \frac{1}{0,2} = 5$, independientemente del número de unidades de procesamiento que se utilicen.



¿Es muy malo un Speedup tan pequeño?

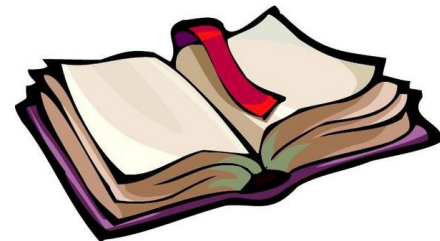
En muchos casos, obtener una aceleración de 5 o 10 es más que aceptable. 🤔



Agenda

33

- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica



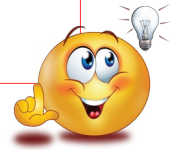
Límites en la ley de Amdahl – Ley de Gustafson

34

- Pero ... muchos programas alcanzan enormes Speedups en grandes sistemas con miles de unidades de procesamiento!!!



La ley de Amdahl no tiene en cuenta el tamaño del problema!!!



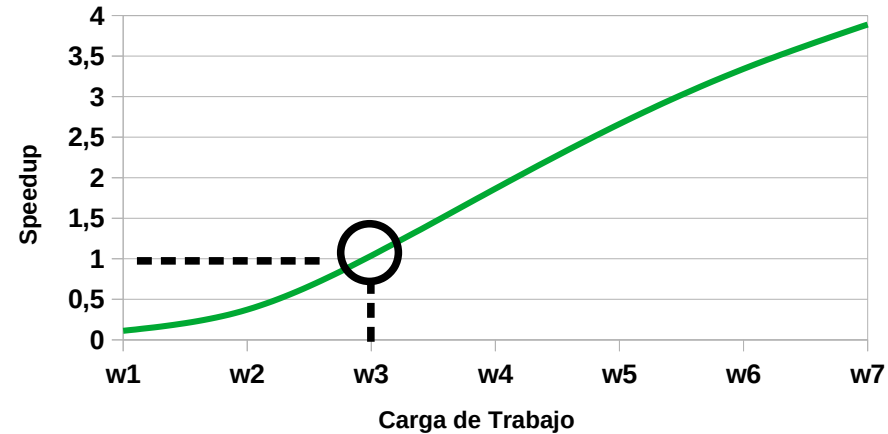
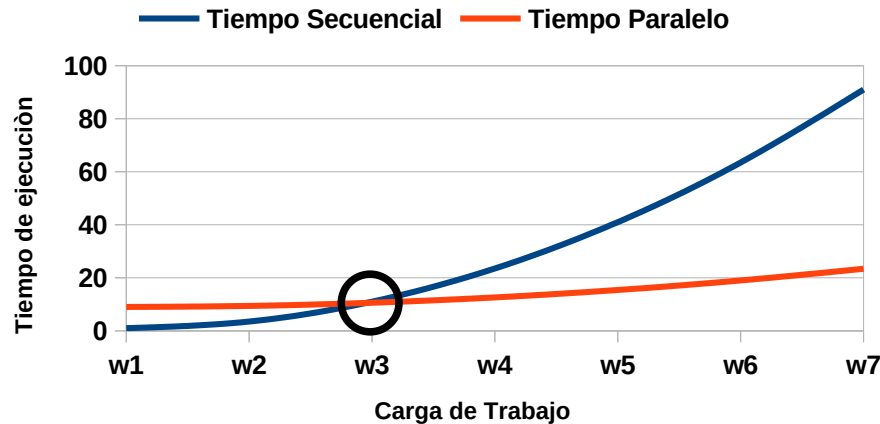
- **Ley de Gustafson (o Ley de Gustafson-Barsis):** A medida que aumentamos el tamaño del problema, la fracción "inherentemente secuencial" del programa disminuye en tamaño y es posible alcanzar mayor speedup.

J.L. Gustafson, Reevaluating Amdahl's law, Commun. ACM 31 (5) (1988) 532–533.

Límites en la ley de Amdahl – Ley de Gustafson

35

- A veces, nos encontramos situaciones en que el tiempo de ejecución del algoritmo paralelo, ejecutado con P unidades de procesamiento, no obtiene mejoras respecto al tiempo de ejecución del algoritmo secuencial ($\text{Speedup} < 1$)
- La razón es que la carga de trabajo del algoritmo (W) no es lo suficientemente grande, y la fracción secuencial es significativa respecto a la fracción paralelizable. A medida que la carga de trabajo se incrementa, el Speedup aumenta.



Tiene sentido paralelizar el algoritmo a partir de la carga de trabajo $W3$ (speedup > 1)

Ley de Gustafson

Formalización

36

- Si ahora tenemos en cuenta el tamaño del problema (n) en las ecuaciones.
- Sea t_f el tiempo de ejecución de la parte no paralelizable.
- Sea $t_v(n,p)$ el tiempo de ejecución de la parte paralelizable con p unidades de procesamiento para un problema de tamaño n .
- Podemos expresar el Speedup como:

$$S_p(n) = \frac{t_f + t_v(n, 1)}{t_f + t_v(n, p)}$$



Ley de Gustafson

Formalización

37

- Si asumimos una paralelización perfecta de la parte paralelizable:

$$t_v(n,1) = T^*(n) - t_f$$

- Y:

$$t_v(n,p) = \frac{T^*(n) - t_f}{p}$$

- Podemos expresar el Speedup como:

$$S_p(n) = \frac{T_s}{T_p} = \frac{t_f + t_v(n,1)}{t_f + t_v(n,p)} = \frac{t_f + T^*(n) - t_f}{t_f + \frac{T^*(n) - t_f}{p}} = \frac{\frac{t_f}{T^*(n) - t_f} + 1}{\frac{t_f}{T^*(n) - t_f} + \frac{1}{p}}$$



Ley de Gustafson

Formalización

38

- Entonces, el límite del Speedup a medida que incrementamos el tamaño de problema:

$$\lim_{n \rightarrow \infty} S_p(n) = \lim_{n \rightarrow \infty} \frac{\frac{t_f}{T^*(n) - t_f} + 1}{\frac{t_f}{T^*(n) - t_f} + \frac{1}{p}} = p$$



Si el tiempo de ejecución $T^*(n)$ se incrementa de forma monótona, con el incremento de n , entonces es posible aproximar el **Speedup** al número de unidades de procesamiento utilizadas.




Limitaciones de la Ley de Gustafson

39

- Algunos problemas pueden verse limitados bajo la ley de Gustafson y rige la ley de Amdahl:



- Problemas con bajo volumen de datos no variable en el tiempo. Ej: trabajar a nivel continentes, países, longitud de una cadena de ADN etc.
- Algunos algoritmos no lineales: suelen alcanzar speedup muy bajo.

- Las arquitecturas actuales poseen múltiples cores. Estos problemas requieren el uso de un subconjunto de estos (no pueden acelerar con más cores). 

- Conociendo el grado de paralelismo se podría alcanzar mayor eficiencia energética activando el número óptimo de cores.



Amdahl vs. Gustafson - Metáfora de la carretera

40

- Ley de Amdahl:
 - Un auto viaja entre dos ciudades que se encuentran a 60km de distancia, y ya recorrió 1 hora viajando la mitad de la distancia a 30km/h. Sin importar cuán rápido viaje la otra mitad es imposible alcanzar una velocidad promedio de 90km/h antes de terminar el recorrido. Debido a que ya le tomó 1 hora y tiene que recorrer 60km, aunque viaje infinitamente rápido sólo alcanzaría una velocidad promedio de 60km/h.
- Ley de Gustafson:
 - Un auto ha estado viajando algún tiempo a 90km/h. Teniendo distancia y tiempo suficiente para viajar, la velocidad promedio del auto podría alcanzar eventualmente los 90km/h, sin importar cuánto o a qué velocidad haya viajado. Por ejemplo, si el auto demoró 1 hora viajando a 30km/h, podría alcanzar los 90km/h de velocidad promedio conduciendo a 120km/h durante 2 horas, o a 150km/h durante 1 hora, etc.

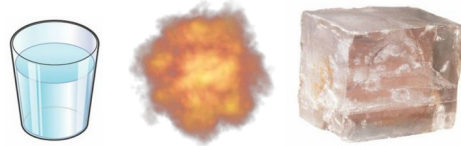


Amdahl vs. Gustafson – Relación en la granularidad

41

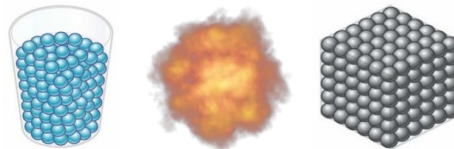
- Ley de Amdahl:

- Supone que los requerimientos se mantendrán invariantes en el tiempo. Por ejemplo, una simulación de una explosión donde la granularidad mas fina es modelar objetos macroscópicos que sufren el impacto.



- Ley de Gustafson:

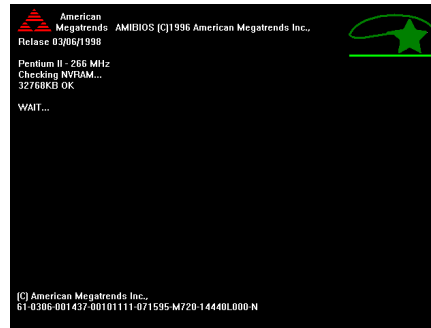
- Supone que mayor potencia de cómputo permitiría modelar problemas más profundamente. La simulación de la explosión utilizando mayor potencia de cómputo permitiría modelar a un nivel de mayor granularidad (materiales microscópicos como moléculas o átomos).



Amdahl vs. Gustafson – Ejemplo booteo

42

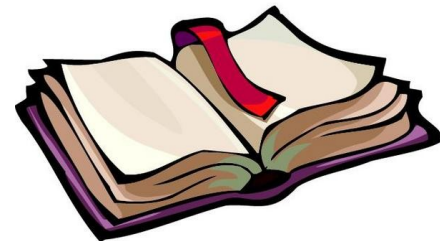
- Ley de Amdahl:
 - Supone que si paralelizamos el inicio de una máquina (**booteo**) el speedup se verá limitado por la parte secuencial, *“no se podrá hacer más nada”*.
- Ley de Gustafson:
 - Supone que *“sí se puede hacer algo más”*. Aunque el sistema no pueda iniciar más rápido, pueden incrementarse las funcionalidades y características del sistema. El tiempo de inicio del sistema operativo será el mismo pero incluirá más características.



Agenda

43

- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica



Escalabilidad:

Propiedad deseable de un sistema, red o proceso, que indica su habilidad para reaccionar y adaptarse sin perder calidad, manejar el crecimiento continuo de trabajo y estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos.



Escalabilidad en los sistemas paralelos

45

Definición (programa paralelo escalable): Un programa paralelo es escalable si mantiene constante la eficiencia al aumentar el número de unidades de procesamiento aumentando también el tamaño del problema.

- El análisis de escalabilidad nos da una idea teórica del comportamiento del sistema al incrementar su capacidad de cómputo.
- Nos permite:
 - Comparar el comportamiento de diferentes algoritmos
 - Predecir efectos en el rendimiento del sistema al cambiar alguno de sus parámetros
 - Optimizar la implementación paralela

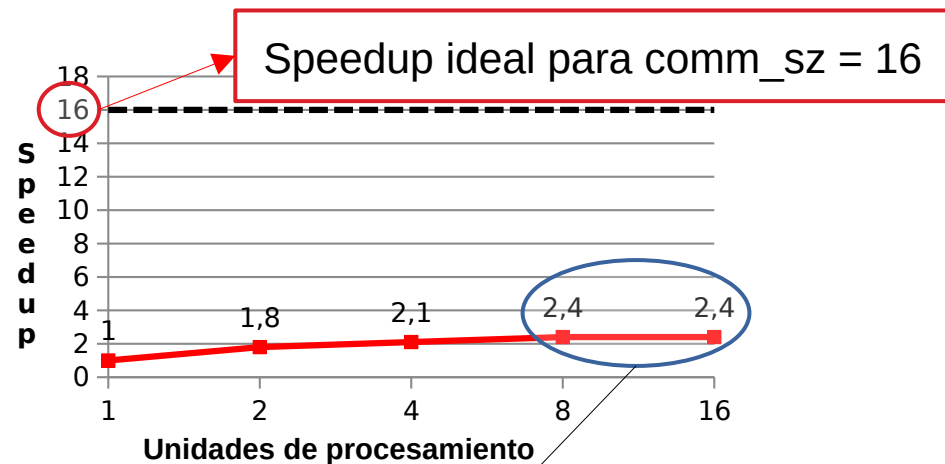
- Un programa paralelo puede ser:
 - **Fuertemente escalable:** mantiene la eficiencia constante sin incrementar el tamaño del problema.
 - **Débilmente escalable:** mantiene la eficiencia constante al incrementar el tamaño del problema al mismo tiempo que se incrementa el número de unidades de procesamiento.
 - **No escalable:** no se cumplen los casos anteriores.
- Analicemos el siguiente ejemplo¹: tiempos (en milisegundos) de las ejecuciones de un algoritmo que resuelve la multiplicación de una matriz cuadrada por un vector.

comm_sz	1024	2048	4096	8192	16,384
1	4.1	16.0	64.0	270	1100
2	2.3	8.5	33.0	140	560
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

¹Libro: *An introduction to parallel programming* - Peter Pacheco - 2011

Table 3.6 Speedups of Parallel Matrix-Vector Multiplication

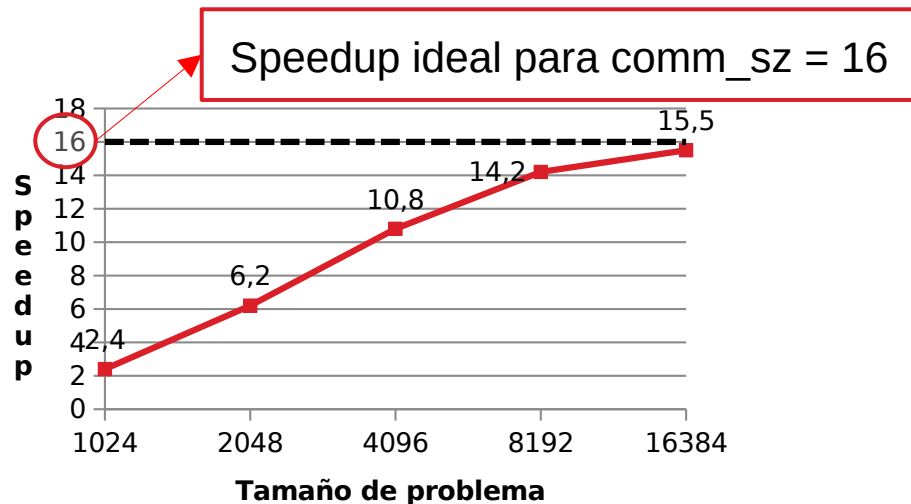
comm_sz	Order of Matrix				
	1024	2048	4096	8192	16,384
1	1.0	1.0	1.0	1.0	1.0
2	1.8	1.9	1.9	1.9	2.0
4	2.1	3.1	3.6	3.9	3.9
8	2.4	4.8	6.5	7.5	7.9
16	2.4	6.2	10.8	14.2	15.5



Sin aumento en el speedup incrementando el número de unidades de procesamiento (**Amdahl**)

Table 3.6 Speedups of Parallel Matrix-Vector Multiplication

comm_sz	Order of Matrix				
	1024	2048	4096	8192	16,384
1	1.0	1.0	1.0	1.0	1.0
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



Aumento en el speedup (cercano al ideal)
incrementando el tamaño del problema (**Gustafson**)

Escalabilidad

Tabla de Eficiencia

49

Table 3.7 Efficiencies of Parallel Matrix-Vector Multiplication

comm_sz	Order of Matrix				
	1024	2048	4096	8192	16,384
1	1.00	1.00	1.00	1.00	1.00
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

No es fuertemente escalable

La eficiencia para un tamaño de problema determinado (*1024 idem hasta 8192*) decae al incrementar el número de unidades de procesamiento.

Table 3.7 Efficiencies of Parallel Matrix-Vector Multiplication

comm_sz	Order of Matrix				
	1024	2048	4096	8192	16,384
1	1.00	1.00	1.00	1.00	1.00
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

Es posible que sea **fuertemente escalable** a partir de este tamaño de problema. La eficiencia a partir de aquí se mantiene constante al incrementar el número de unidades de procesamiento.

Table 3.7 Efficiencies of Parallel Matrix-Vector Multiplication

comm_sz	Order of Matrix				
	1024	2048	4096	8192	16,384
1	1.00	1.00	1.00	1.00	1.00
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

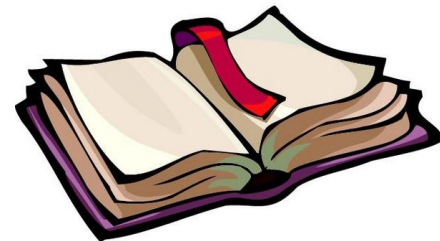
Débilmente escalable

La eficiencia se mantiene relativamente constante incrementando al mismo tiempo el tamaño de problema y el número de unidades de procesamiento.

Agenda

52

- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica



- Las leyes de Amdahl y Gustafson se conocen también como modelos de speedup.
 - **Modelo de carga fija (Amdahl):**
 - Asociado con escalabilidad fuerte.
 - La **carga de trabajo** no cambia (**constante**), el speedup está acotado por la parte no paralelizable y no se obtienen beneficios al agregar unidades de procesamiento.
 - El speedup ideal se logra cuando el problema se resuelve P veces más rápido que el algoritmo secuencial.
 - **Modelo de tiempo fijo (Gustafson):**
 - Asociado con escalabilidad débil.
 - A veces, se requiere **precisión** más que un tiempo mínimo. En este modelo, usado en aplicaciones de precisión crítica, se quiere resolver un problema lo mas grande posible (**incrementar el tamaño del problema**), sobre una máquina más potente (incrementar el número de unidades de procesamiento), obteniendo resultados más precisos sin aumentar el tiempo de ejecución.
 - El speedup ideal se logra cuando se resuelve un problema P veces más grande en la misma cantidad de tiempo que el secuencial.

Modelos de speedup

54



Modelo de tiempo fijo?
Gustafson?
Escalabilidad débil?

Supongamos que tenemos un problema que secuencialmente se resuelve con un costo de UNA unidad de tiempo por cada UNIDAD de la carga de trabajo.

El tiempo de resolver secuencialmente N elementos es N : $T_s(N) = N$

Débilmente escalable con una eficiencia del 0,90.




Eficiencia	N		
P	512	1024	2048
8	0,90		
16		0,90	
32			0,90

$$E = \frac{S}{P}$$
$$S = E \cdot P$$

Speedup	N		
P	512	1024	2048
8	7,2		
16		14,4	
32			28,8

$$S = \frac{T_s}{T_p}$$
$$T_p = \frac{T_s}{S}$$

Tiempo	N		
o			
P	512	1024	2048
8	71,11		
16		71,11	
32			71,11



Modelo de isoeficiencia

55

Definición (Sistemas Paralelos escalables): combinaciones de algoritmos y arquitecturas que cumplen la condición de mantener la **eficiencia constante** al incrementar simultáneamente el número de unidades de procesamiento y el tamaño del problema.

- Entonces, para que nuestro Sistema Paralelo sea escalable debemos incrementar el tamaño del problema al aumentar el número de unidades de procesamiento pero...



¿Cuánto?

- El **Modelo de Isoeficiencia** parte de un sistema paralelo con P unidades de procesamiento idénticas y considera válidas las leyes de **Amdahl** y **Gustafson**.
- A partir de este modelo se obtiene la **función de isoeficiencia**:

La **función de isoeficiencia** da información acerca de cómo debe crecer el tamaño del problema en función del número de unidades de procesamiento para poder mantener la eficiencia constante en **Sistemas Paralelos escalables**.

Modelo de isoeficiencia

56

- Para obtener la función de isoeficiencia es necesario redefinir algunas métricas en función del tamaño del problema o carga de trabajo.
- Definimos el tamaño de problema W como el número total de operaciones necesarias para resolver el algoritmo secuencial.
- Decimos que el tiempo de ejecución secuencial es:

$$T_s = T_c \cdot W$$

- T_c : constante, dependiente de la máquina, que representa el costo de ejecutar cada operación.
- Por simplicidad, asumimos que $T_c=1$. Por lo tanto, el tamaño de problema es igual al tiempo de ejecución secuencial:

$$T_s = W$$

Modelo de isoeficiencia

57

- Luego, redefinimos:

Overhead

$$T_o(W, p) = p \cdot T_p - W$$



Tiempo paralelo (p unidades de procesamiento)

$$T_p = \frac{W + T_o(W, p)}{p}$$



Tiempo secuencial

$$T_s = T_1 = \frac{W + T_o(W, p)}{p} = \frac{W + T_o(W, 1)}{1} = \frac{W + 0}{1} = W$$



Speedup

$$S = \frac{W}{T_p} = \frac{W}{\frac{W + T_o(W, p)}{p}} = \frac{W \cdot p}{W + T_o(W, p)}$$

Modelo de isoeficiencia

58

- Partimos de la ecuación de eficiencia y despejamos W :

$$E = \frac{S}{p} = \frac{\frac{W \cdot p}{W + T_o(W, p)}}{p} = \frac{W}{W + T_o(W, p)} = \frac{1}{1 + \frac{T_o(W, p)}{W}} \quad \Rightarrow \quad \frac{T_o(W, p)}{W} = \frac{1 - E}{E}$$

Función de isoeficiencia $\rightarrow W = \frac{E}{1 - E} \cdot T_o(W, p) = K \cdot T_o(W, p)$



$$W = K \cdot T_o(W, p)$$

- K es una constante que depende de la eficiencia y debemos mantener.
- Cuanto menor sea la función de isoeficiencia mejor: pequeños incrementos en el tamaño del problema son suficientes para usar eficientemente mas procesadores (el sistema será más escalable).



Modelo de isoeficiencia

Ejemplo

59

- Supongamos dos problemas:

Problema 1

- El overhead crece linealmente con un orden $2p \cdot \log_2(p) \rightarrow T_o(W,p) = 2p \cdot \log_2(p)$
- Función de isoeficiencia (**chica**):

$$W = K \cdot T_o(W,p) = K \cdot 2p \cdot \log_2(p)$$

- Para $p=16 \rightarrow W=128.K$
- Usar 32 unidades de procesamiento ($p=32$) requiere que $W=320.K$ para mantener la eficiencia constante.
- W tiene que crecer en un factor de 2,5:

$$320.K = 2,5 \cdot 128.K$$

W tiene que crecer un poco mas del doble



Problema 2

- El overhead crece **NO linealmente** con un orden $2^p \rightarrow T_o(W,p) = 2^p$
- Función de isoeficiencia (**grande**):

$$W = K \cdot T_o(W,p) = K \cdot 2^p$$

- Para $p=16 \rightarrow W=2^{16}.K$
- Usar 32 unidades de procesamiento ($p=32$) requiere que $W=2^{32}.K$ para mantener la eficiencia constante.
- W tiene que crecer en un factor de 2^{16} :

$$2^{32}.K = 2^{16} \cdot 2^{16}.K$$

W tiene que aumentar 65536 veces



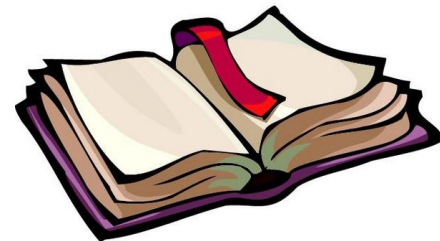
En el **problema 2** asumimos que el overhead tiene orden 2^p , sólo para ejemplificar. Recordar, el overhead crece linealmente en función de p y sub-linealmente en función de w .



Agenda

60

- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica



Métricas en arquitecturas heterogéneas

61



- Cuando las arquitecturas paralelas tienen unidades de procesamiento con características diferentes, las métricas presentan algunas variaciones.

- Suponer p unidades de procesamiento con distinta potencia de cómputo:

$$c_0, c_1, c_2, \dots, c_p.$$

- Llamamos **unidad de procesamiento “base”** a la unidad de procesamiento más potente y llamamos c_{base} a su potencia de cómputo.

Métricas en arquitecturas heterogéneas

Eficiencia Heterogénea

62

- Para analizar que ocurre con las métricas sobre arquitecturas heterogéneas, partimos definiendo la eficiencia como:

$$E = \frac{C_{base}}{C_{promedio}} \cdot \frac{T_{secBase}}{p \cdot T_p}$$

- C_{base} : potencia de cómputo de la unidad de procesamiento “base”.
 - $C_{promedio}$: potencia de cómputo promedio de todas las unidades de procesamiento.
 - $T_{secBase}$: tiempo de ejecución del algoritmo secuencial ejecutado en la unidad de procesamiento “base”.
- Si las unidades de procesamiento tienen la misma potencia de cómputo, la eficiencia heterogénea coincide con la eficiencia homogénea.

Métricas en arquitecturas heterogéneas

Límite del Speedup Heterogéneo

63

- A partir de la eficiencia óptima ($E = 1$), obtenemos el límite del **Speedup heterogéneo**:

$$E = 1 = \frac{C_{base}}{C_{promedio}} \cdot \frac{T_{secBase}}{p \cdot T_p} \quad \xrightarrow{\text{Despejamos el Speedup}} \quad \frac{T_{secBase}}{T_p} = p \cdot \frac{C_{promedio}}{C_{base}} \quad \xrightarrow{\quad} \quad \boxed{S_{het} \leq \underbrace{p \cdot \frac{C_{promedio}}{C_{base}}}_{\text{Speedup heterogéneo}}}$$

- La potencia promedio de nuestro sistema compuesto por p unidades de procesamiento:

$$C_{promedio} = \frac{\sum_{i=0}^{p-1} C_i}{p} = \frac{C_{Total}}{p}$$

- Luego, normalizamos la potencia de cómputo base ($C_{base} = 1$), entonces el límite del **Speedup heterogéneo** es:

$$\boxed{S_{het} \leq p \cdot \frac{C_{promedio}}{C_{base}} \leq p \cdot C_{promedio} \leq p \cdot \frac{C_{Total}}{p} \leq C_{Total}}$$

Métricas en arquitecturas heterogéneas

Ejemplo: Cálculo del límite del Speedup Heterogéneo

64

- Calcular el límite del speedup heterogéneo para la siguiente arquitectura paralela:
 - 8 unidades de procesamiento.
 - Suponemos la potencia de la única unidad de procesamiento base normalizada ($c_{base} = 1$).
 - 4 unidades de procesamiento tienen un 75% de la potencia de la unidad de procesamiento base.
 - 3 unidades de procesamiento tienen un 50% de la potencia de la unidad de procesamiento base.
- Para una arquitectura homogénea el límite en el **speedup** es 8, pero para la arquitectura heterogénea del ejemplo es:

$$S_{het} \leq 1 + 4 \cdot 0,75 + 3 \cdot 0,50 \leq 5,5$$

Métricas en arquitecturas heterogéneas

Potencia de cómputo relativa

65

- Simbolizamos la **potencia de cómputo** de la **unidad de procesamiento i** como c_i .

Definimos la **potencia de cómputo relativa** (Pcr_i) como la potencia de cómputo de una unidad de procesamiento i respecto a la potencia de cómputo de la mejor unidad de procesamiento de la arquitectura:

$$Pcr_i = \frac{c_i}{c_{base}}$$

¿Cómo determinamos la potencia de cómputo de una unidad de procesamiento?



Métricas en arquitecturas heterogéneas

Potencia de cómputo relativa

66

- Podemos obtener la potencia de cómputo en función del tiempo de ejecución de una aplicación.
- Consideramos que la potencia de cómputo de una unidad de procesamiento i es inversamente proporcional al tiempo (T_i) que tarda una aplicación en ejecutarse en esa unidad de procesamiento:

$$c_i = \frac{1}{T_i}$$

- La potencia de cómputo relativa expresada en función del tiempo de ejecución:

$$Pcr_i = \frac{c_i}{c_{base}} = \frac{\frac{1}{T_i}}{\frac{1}{T_{base}}} = \frac{T_{base}}{T_i}$$



Métricas en arquitecturas heterogéneas

Ejemplo: Potencia de cómputo relativa

67

- Calcular la potencia de cómputo relativa para cada unidad de procesamiento de una arquitectura heterogénea con 5 unidades de procesamiento.
- Se ejecuta una aplicación, con una cierta carga de trabajo, en cada unidad de procesamiento de la arquitectura y se obtienen los siguientes tiempos de ejecución:

UP	Tiempo en segundos
P_0	40
P_1	24
P_2	40
P_3	30
P_4	24

P_1 y P_4 son las más rápidas.

P_3 es intermedia.

P_0 y P_2 son las más lentas.

Métricas en arquitecturas heterogéneas

Ejemplo: Potencia de cómputo relativa

68

- Se toma como base el tiempo de la unidad de procesamiento más rápida, la mejor de la arquitectura, la más potente.
- Podemos tomar P_1 o P_4 . Elegimos P_1 como base y la denominamos P_{base} .
- Calculamos la potencia de cómputo relativa de cada unidad de procesamiento:

UP	Tiempo en segundos	Pcr
P_0	40	$\frac{24}{40}=0,6$
$P_1 = P_{base}$	24	$\frac{24}{24}=1$
P_2	40	$\frac{24}{40}=0,6$
P_3	30	$\frac{24}{30}=0,8$
P_4	24	$\frac{24}{24}=1$

La potencia relativa de la unidad de procesamiento base queda normalizada a 1.

Podemos calcular la potencia total del sistema (c_{Total}) y en consecuencia el límite del speedup heterogéneo:

$$S_{Het} \leq c_{Total} = \sum_{i=0}^4 pcr_i = 0,6 + 1 + 0,6 + 0,8 + 1 = 4$$

Métricas en arquitecturas heterogéneas

Ejemplo: Distribución de carga

69

- El ejemplo anterior nos puede ser útil si queremos distribuir estáticamente una carga de trabajo de manera balanceada y proporcional a la potencia relativa.
- Obtenemos el porcentaje que representa la potencia de cómputo relativa de cada unidad de procesamiento con respecto a la potencia total:

$(c_{Total} = 4 \rightarrow c_{Total}$ es el 100%):

UP	Tiempo en segundos	Pcr	Porcentaje (Pcr de C_{Total})
P_0	40	0,6	0,6 de 4 = 15%
$P_1 = P_{base}$	24	1	1 de 4 = 25%
P_2	40	0,6	0,6 de 4 = 15%
P_3	30	0,8	0,8 de 4 = 20%
P_4	24	1	1 de 4 = 25%

Según el porcentaje, si se tiene una carga de trabajo de 100 datos la distribución es la siguiente:

P_1 y P_4 recibirán 25 datos cada una.


P_3 recibirá 20 datos.

P_0 y P_2 recibirán 15 datos cada una.

Métricas en arquitecturas heterogéneas

Consideraciones

70

- La potencia de cómputo de una unidad de procesamiento puede depender del tipo de aplicación.
 - Una unidad de procesamiento puede ser la más potente para un tipo de aplicaciones pero no para otras.
- La potencia de cómputo de una unidad de procesamiento puede depender del volumen de la carga de trabajo.
 - Una unidad de procesamiento puede ser la más potente para una carga de trabajo chica pero no para una carga de trabajo mayor.
 - Esto impacta en la distribución estática. El cálculo de la potencia de cómputo se hace en base al algoritmo secuencial y una carga de trabajo fija. Cuando se distribuye el trabajo del algoritmo paralelo, cada unidad de procesamiento siempre recibe una carga de trabajo menor.
- La estrategia de distribución estática propuesta es sólo una aproximación (el problema de distribución balanceada es **NP-Completo**). Tener en cuenta que debe ejecutarse la aplicación en cada unidad de procesamiento (**profiling**) para calcular la potencia, lo que insume tiempo.



Métricas en arquitecturas heterogéneas

Consideraciones

71



- Si una arquitectura heterogénea posee un equipo multicore (simétrico/homogéneo)



¿Cómo se considera la potencia de cómputo de ese equipo?

- Dos opciones:
 - De cada core individual: calcular el porcentaje de carga de acuerdo al tiempo secuencial (un core) y multiplicarlo por el número de cores
 - Del multicore en su conjunto: calcular el porcentaje de carga de acuerdo al tiempo paralelo (ejecución sobre todos los cores)

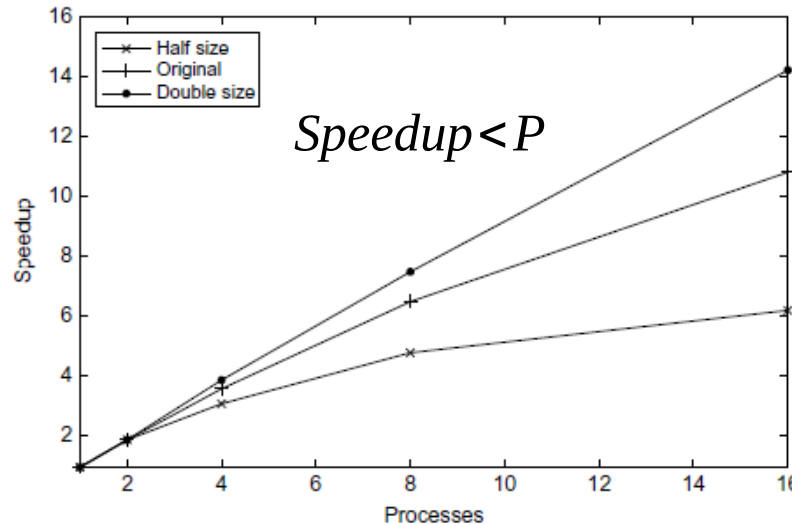
Métricas en arquitecturas heterogéneas

Consideraciones

72



- Queremos medir la potencia de cómputo de un equipo multicore homogéneo.
- Pero es poco probable tener un paralelismo perfecto:



Por lo tanto, NO podemos considerar el tiempo de un core y escalarlo al número de cores de la arquitectura, mucho menos si el speedup del algoritmo paralelo está lejos del ideal.



Deberíamos considerar el tiempo de ejecución de la aplicación paralela.

Métricas en arquitecturas heterogéneas

En la práctica

73

- Un vez que distribuimos la carga y ejecutamos nuestra aplicación paralela en el sistema heterogéneo:



¿Cómo calculamos empíricamente nuestras métricas?

- El Speedup:

$$S = \frac{T_{secBase}}{T_p}$$

- La Eficiencia:

$$E = \frac{c_{base}}{c_{promedio}} \cdot \frac{T_{secBase}}{p \cdot T_p}$$



$$\left(\begin{array}{l} (a) c_{base} = 1 \text{ (Normalizada)} \\ (b) c_{promedio} = \frac{\sum_{i=0}^{p-1} c_i}{p} = \frac{c_{Total}}{p} \end{array} \right)$$



$$E = \frac{S}{c_{Total}} = \frac{S}{S_{het}}$$

- El Overhead:

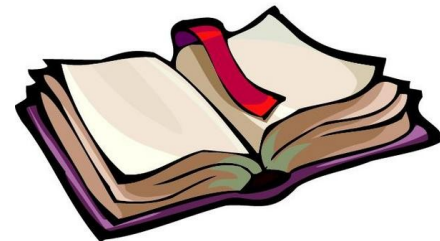
$$T_o = c_{Total} \cdot T_p - T_{secBase} = S_{het} \cdot T_p - T_{secBase}$$



Agenda

74

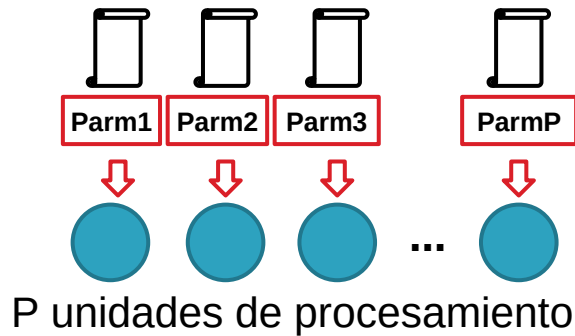
- I. Métricas: Introducción
- II. Speedup
- III. Eficiencia
- IV. Overhead
- V. Balance de carga
- VI. Escalabilidad en los sistemas paralelos
 - i. Ley de Amdahl
 - ii. Ley de Gustafson
 - iii. Escalabilidad
 - iv. Modelo de isoeficiencia
- VII. Métricas en sistemas con arquitecturas heterogéneas
- VIII. Ejecución paramétrica



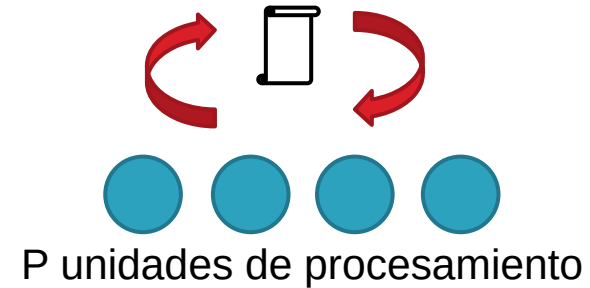
Ejecución paramétrica

75

- Una aplicación debe ser ejecutada varias veces sobre una arquitectura.
- Cada ejecución se realiza modificando el valor de ciertos parámetros.
- Dada una arquitectura paralela, existen dos alternativas:



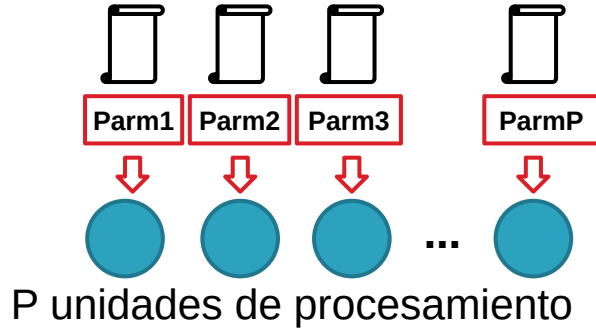
Alternativa 1- Algoritmo secuencial: Ejecutar simultáneamente P instancias de la versión secuencial. Cada instancia con parámetros diferentes.



Alternativa 2 – Algoritmo paralelo: Ejecutar consecutivamente una versión paralela (P veces). Cada ejecución con parámetros diferentes.

Ejecución paramétrica

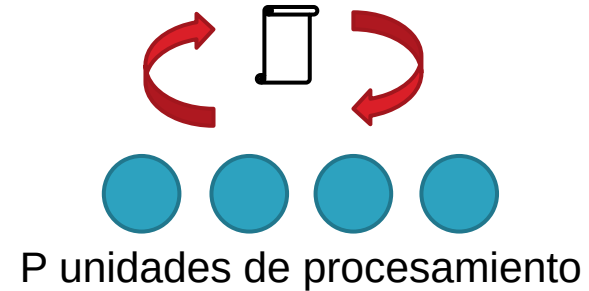
76



Alternativa 1- Algoritmo secuencial: Ejecutar simultáneamente P instancias de la versión secuencial. Cada instancia con parámetros diferentes.

El tiempo de ejecución total es igual al tiempo secuencial:

$$T_{total} = T_s$$



Alternativa 2 – Algoritmo paralelo: Ejecutar consecutivamente una versión paralela (P veces). Cada ejecución con parámetros diferentes.

El tiempo de ejecución total es igual a P veces el tiempo paralelo:

$$T_{total} = P \cdot T_p$$

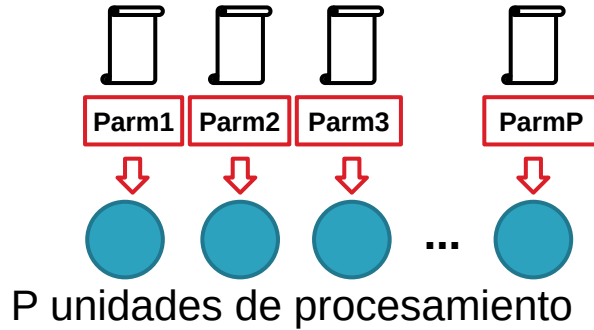
En el mejor caso hay un paralelismo perfecto: $S = \frac{T_s}{T_p} = p$

De la ecuación de Speedup despejamos T_p y reemplazamos en la ecuación de T_{total} :

$$\frac{T_s}{T_p} = p \rightarrow T_p = \frac{T_s}{p} \rightarrow T_{total} = p \cdot T_p = p \cdot \frac{T_s}{p} = T_s$$

Ejecución paramétrica

77



Alternativa 1- Algoritmo secuencial: Ejecutar simultáneamente P instancias de la versión secuencial. Cada instancia con parámetros diferentes.

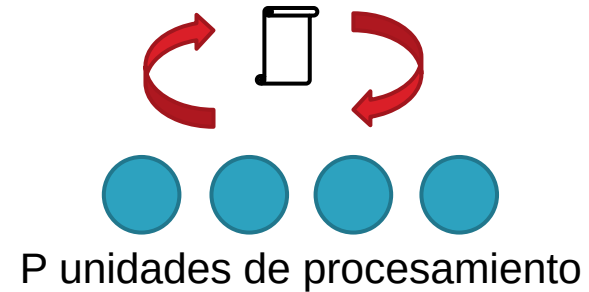
El tiempo de ejecución total es igual al tiempo

$$T_{total} = T_s$$



Algunos casos convenientes:

- Speedup superlineal
- Obtención de datos parciales
- Aplicaciones de tiempo real



Alternativa 2 – Algoritmo paralelo: Ejecutar consecutivamente una versión paralela (P veces). Cada ejecución con parámetros diferentes.

El tiempo

El tiempo total en las dos alternativas es el mismo.
Entonces...
¿Conveniente paralelizar?

$$S = \frac{T_s}{T_p} = p$$

De la ecuación de Speedup despejamos T_p y reemplazamos en la ecuación de T_{total} :

$$\frac{T_s}{T_p} = p \rightarrow T_p = \frac{T_s}{p} \rightarrow T_{total} = p \cdot T_p = p \cdot \frac{T_s}{p} = T_s$$

