

EVALUACIÓN DE LAS PRESTACIONES

“ AL BARRO ”



ANTECEDENTES

Algoritmia (Segundo curso)

- Análisis de la eficiencia de algoritmos secuenciales.
- Enfoque *monovariante*.
- Los modelos basados en estimaciones teóricas con cotas asintóticas de complejidad son buenos estimadores.
 - Hay un espacio de comportamiento indefinido.
- Los modelos basados en extrapolación de observaciones son parciales y presentan sesgo.

Programación Concurrente y Paralela

- Basado en los modelos usados en Algoritmia para algoritmos paralelos.
- Se irá explicando por fases durante la evolución del curso (fase I y fase II –heterogeneidad–)

ANTECEDENTES

Se define el **Tiempo de Ejecución Secuencial** como el tiempo que tarda en ejecutarse el programa en una sola unidad de proceso (procesador o *core*)

- Depende del tamaño de la entrada, del compilador, del programador, etc. Se obvian constantes del sistema y se asume que solo depende del tamaño del problema ($t(n)$)
- El análisis a priori cuenta *flop* (no pasos)
- En el análisis a posteriori se mide tiempo (p. ej. segundos)

- Expresiones recurrentes:

$$\sum_{i=1}^n 1 = n$$

$$\sum_{i=1}^n i \approx \frac{n^2}{2}$$

$$\sum_{i=1}^n i^2 \approx \frac{n^3}{3}$$

$$\sum_{\substack{i=1 \\ \text{step } i/b}}^n k \approx k \log_b(n) = k \frac{\log(n)}{\log(b)}$$

ANTECEDENTES

Ejemplos

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    b = b + y[i][j];
```

$$T(n) = \sum_{i=1}^n \sum_{j=1}^n 1 = \sum_{i=1}^n n = n^2 \text{ flop}$$

```
for (i=0; i<n; i++)
  for (j=i; j<n; j++)
    b = b + y[i][j];
```

$$T(n) = \sum_{i=1}^n \sum_{j=i}^n 1 = \sum_{i=1}^n (n - i + 1) = n^2 + n - \sum_{i=1}^n i \approx n^2 + n - \frac{n^2}{2} = \frac{n^2}{2} + n \text{ flop}$$

```
for (i=0; i<n; i++)
  for (j=i; j<n; j++)
    for (k=i; k<n; k++)
      b = b + y[i][k];
```

$$T(n) = \sum_{i=1}^n \sum_{j=i}^n \sum_{k=i}^n 1 \approx \sum_{i=1}^n \sum_{j=i}^n (n - i) = \sum_{i=1}^n (n - i)^2 \approx \frac{n^3}{3} \text{ flop}$$

ANTECEDENTES

- Los algoritmos paralelos se usan para **acelerar** la resolución de problemas de alto coste computacional.
- El objetivo fundamental es **reducir** el tiempo de ejecución: *“si el tiempo de ejecución secuencial es $t(n)$ se busca que sea $\frac{t(n)}{p}$ (teórico máximo) cuando se usan p procesadores”*
- Es necesario disponer de herramientas para evaluar las prestaciones de los algoritmos paralelos de forma **fiable y precisa**.
- Es recomendable **considerar** la eficiencia desde la fase de diseño para construir algoritmos óptimos.
- El **Rendimiento** de un programa paralelo es *“un conjunto de aspectos cualitativamente dispares y, por tanto, complejos: tiempo de ejecución, productividad (throughput), latencia, portabilidad, escalabilidad, etc.”*



ANTECEDENTES

Extrapolando observaciones...

“... Se ha implementado un algoritmo paralelo y se ha obtenido un incremento de velocidad de 10.8 con 12 procesadores para un tamaño del problema 100 ...”

Qué sucede

¿ con 1000 procesadores ?, ¿ cuando n es 10 ?, ¿ si el coste de las comunicaciones es 10 veces mayor ?, etc.

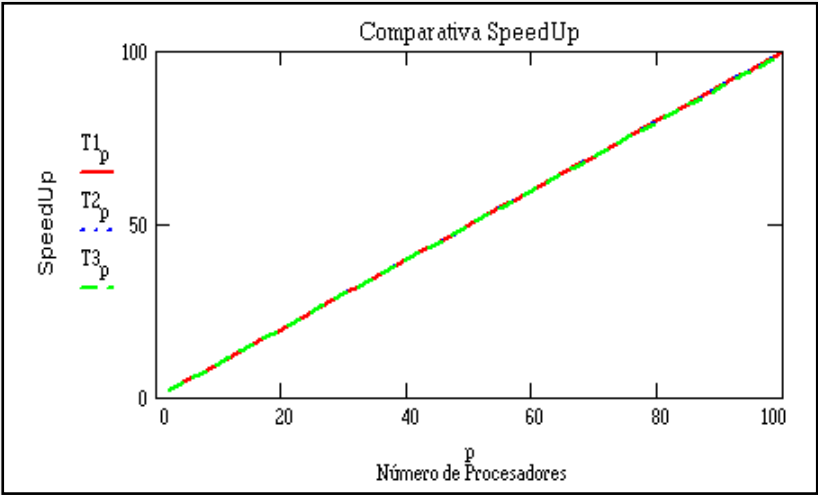
Conjeturando con ecuaciones

“... Sean 3 algoritmos que resuelven el mismo problema cuyas expresiones analíticas del tiempo de ejecución son las siguientes. ¿cuál es mejor?”

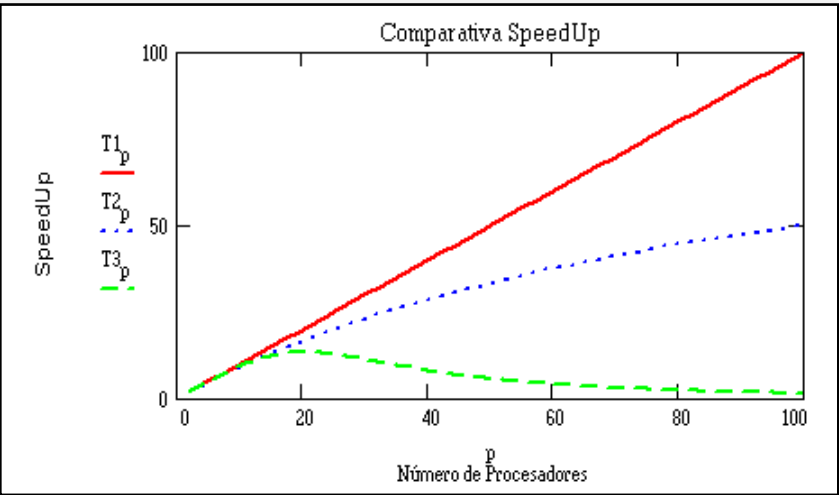
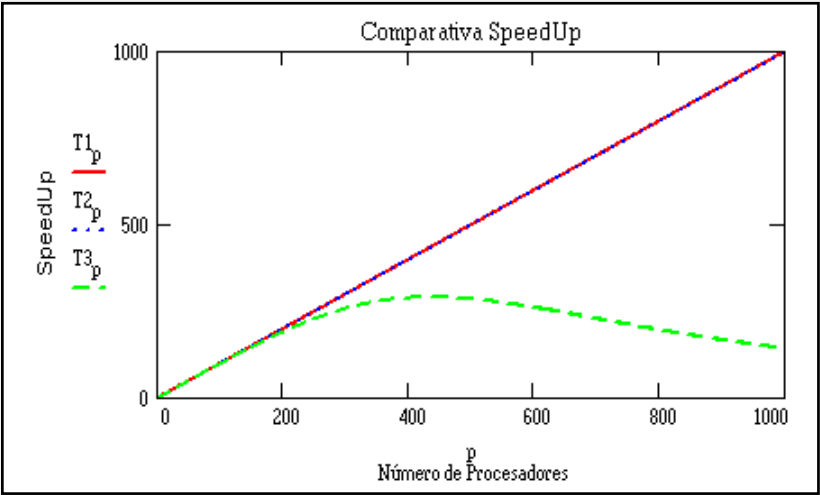
$$T_1(n, p) = \frac{n+n^2}{p}; \quad T_2(n, p) = \frac{n+n^2}{p} + 100; \quad T_3(n, p) = \frac{n+n^2}{p} + p^2$$

ANTECEDENTES

El ejemplo cuando n es 10000



Con igual n y p hasta 1000



Con n y p hasta 100

ANTECEDENTES

A Priori: Teórico

- En la etapa de diseño, independiente de la máquina.
- Permite identificar la mejor opción a la hora de implementar y descubrir los tamaño de los problemas adecuados.
- Incremental respecto a los modelos usados en Algoritmia (θ , \mathbf{O} , Ω , etc.)

A Posteriori: Empírico

- Sobre una implementación y máquina específica. Permite analizar cuellos de botella, dependencias, conflictos, etc. no observados en el diseño.

¿Qué es recomendable?

- Usar una combinación adecuada de ambos tipos de análisis. Es importante saber modelizar y medir adecuadamente.

ANTECEDENTES

Parámetros Absolutos

- Permiten conocer el coste real de los algoritmos.
- Muy importantes en problemas de tiempo real.
- Se deben normalizar y, aún así, no son útiles para comparar algoritmos.
- De todos ellos usaremos el **Tiempo de Ejecución**.
- Son la base para el cálculo de los parámetros relativos.

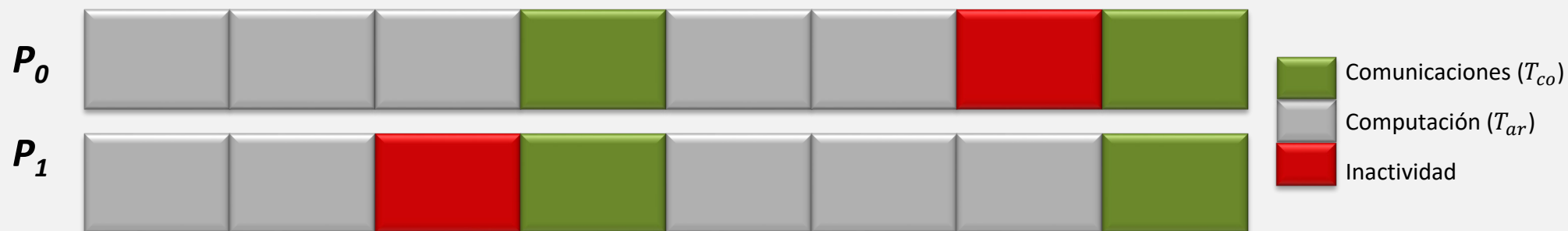
Relativos

- Estiman la efectividad con que los algoritmos usan los recursos (grado de aprovechamiento de los recursos)
- Permiten comparar algoritmos, tanto paralelos como secuenciales.
- Suele estudiarse **Coste, Overhead, Eficiencia, Speedup y Escalabilidad**.

TIEMPO DE EJECUCIÓN PARALELO

Def. Tiempo transcurrido desde que empieza el primer procesador hasta que termina el último.

- Saber cuándo un procesador empieza/termina no es fácil y no siempre es igual (relación de orden parcial)
- Al igual que en secuencial, admite análisis a priori y a posteriori.
- Un modelo simplificado, generalización del secuencial, es aquel donde el tiempo de ejecución paralelo depende del tamaño del problema (n) y del número de procesadores (p): $t(n, p)$
- Primera aproximación: $T(n, p) = T_{ar}(n, p) + T_{co}(n, p)$



$$T_{p1}(n, p) = 5 + 2 = 7; T_{p0}(n, p) = 5 + 2 = 7 \Rightarrow T(n, p) = 7 < 8$$

TIEMPO DE EJECUCIÓN PARALELO

- Segunda aproximación:

$$T(n, p) = T_{ar}(n, p) + T_{co}(n, p) - T_{sol}(n, p) + T_{ov}(n, p)$$

$T_{sol}(n, p)$ tiempo de solapamiento entre T_{ar} y T_{co}

$T_{ov}(n, p)$ tiempo de sobrecarga (esperas, creación procesos, etc.)

- En algoritmos síncronos $T_{sol} = 0 \Rightarrow T_{ar}(n, p) + T_{co}(n, p) + T_{ov}(n, p)$
- En asíncronos $T(n, p) = \max(T_{ar}, T_{co}) + T_{ov}$ donde $\frac{T_{ar}+T_{co}}{2} \leq \max(T_{ar}, T_{co}) \leq T_{ar} + T_{co}$
- T_{ov} y T_{sol} son difíciles de estimar por lo que en este curso se admite, *para el análisis de situaciones complejas*, la aproximación:

$$T(n, p) \cong T_{ar}(n, p) + T_{co}(n, p)$$

- Para simplificar su cálculo
 - Estudiar $T_{ar}(n, p)$ y $T_{co}(n, p)$ por separado y luego combinar.
 - Basar el estudio en una arquitectura paralela ideal.

TIEMPO DE EJECUCIÓN PARALELO

$T_{ar}(n, p)$

- Tiempo que el algoritmo emplea realizando cálculos. Se expresa en flop (en segundos en el modelo empírico). Se estima teóricamente como

$$T_{ar}(n, p) = nt_c \quad Ec. 1$$

siendo n el número de flop y t_c el tiempo por flop.

$T_{co}(n, p)$

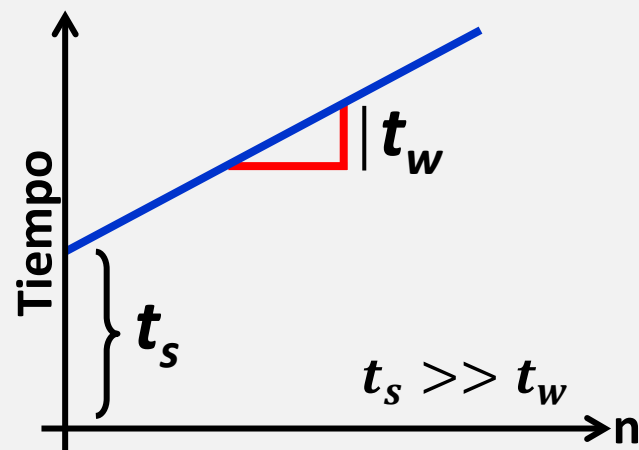
- Tiempo que el algoritmo emplea enviando mensajes (en memoria distribuida, MD) o en operaciones de sincronización (en memoria compartida, MC). Por simplicidad se usará el mismo modelo. De igual forma se usa el mismo modelo para comunicaciones externas e internas (y mismos valores para las constantes)
- El intercambio de información entre 2 procesos/procesadores vecinos (P2P) se estima con

$$T_{co}(n, p) = t_s + nt_w \quad Ec. 2$$

donde t_s es la latencia (*start-up time*) y t_w el ancho de banda.

TIEMPO DE EJECUCIÓN PARALELO

$$T_{co}(n, p)$$



$t_s \gg t_w$ En consecuencia, enviar mensajes grandes en vez de varios pequeños puede reducir el impacto de las comunicaciones

$$t_s + nt_w \ll n(t_s + t_w) = nt_s + nt_w$$

- Las constantes t_s y t_w pueden “variar” en función del tipo de red, tráfico, etc. No consideraremos este efecto.
- El modelo visto no vale para operaciones más complejas (difusiones, recolecciones, etc.) Estas se deben expresar en función de las P2P.
- En MC el sincronismo se puede modelizar como mensajes de tamaño constante (coste constante de la sincronización) o en función del número de procesadores.

FLOP / FLOPs

Se define **Flop** como una operación en coma flotante.

- 1 flop = coste de una operación básica en coma flotante: suma, resta, multiplicación, división.
- El coste de otras operaciones en coma flotante se expresan en término de su número de flop.
- En este tipo de enfoques el resto de operaciones (aritmética entera, etc.) no se suelen *contabilizar*.

Se define **FLOPs** como el número de flop por segundo.

- Es una medida independiente del entorno *software/middleware*.
- *Theoretical Peak double precision floating point Performance* (TPP_{dp}) se expresa en GFLOPs. Es una estimación *muy optimista*.
- Se acepta, por norma general, que TPP_{sp} (simple precision) = $2TPP_{dp}$
- El conocimiento de TPP es de utilidad en el diseño.

FLOP / FLOPs

Estimación de TPP_{dp}

$$TPP_{dp} = chasis \times nodos_{chasis} \times sockets_{nodo} \times cores_{socket} \times clock_{GHz} \times \left(\frac{n^o \text{ flop}}{ciclo} \right)^{**}$$

$$TPP_{dp} = sockets \times cores_{socket} \times clock_{GHz} \times (AVX \text{ ó } FMA)^*$$

Modelo	Nº Sockets	Cores x Socket	Frecuencia (GHz)	Flop por cycle (DP – SP)	TPP _{dp} (Gflop)	TPPsp (Gflop)
Xeon PHi 31S1P	57	4	1.1	4 – 8	1003.0	2006.0
Xeon E5-2603 v3	2	6	1.6	16 – 32	307.0	614.0
Xeon E5620	2	4	2.4	8 – 16	153.6	307.2
Ryzen 7 3700x	1	8	2.2	16 – 32	281.6	563.2
I3-2100	1	2	3.1	8 – 16	49.6	99.2
Cortex-A15	1	4	2.2	2 – 8	17.6	70.4

* SSE2/AVX (128)=8; AVX2 (256)=16; AVX 512=32; FMA=16; etc. (*doubles*). ** 4x, 8x, 16x, etc. según el procesador.

Visitar “<https://ark.intel.com/content/www/es/es/ark.html#@Processors>” para ver las especificaciones de procesadores Intel.

Visitar “<https://www.spec.org>” para ver resultados de benchmarks de procesadores/servidores.

COSTE Y SOBRECARGA

Coste

- Se define como

$$C(n, p) = pT(n, p) \quad Ec. 3$$

- Un algoritmo es de coste óptimo si $C(n, p)$ es proporcional a $T(n)$.

Ejemplo:

$$T(n) = 100; p = 10; T(n, p) = 10 \Rightarrow C(n, p) = 10 * 10 = 100 = T(n)$$

Función de Sobrecarga u *Overhead*

- Expresa el coste añadido con respecto al algoritmo secuencial o, en otras palabras, el tiempo extra que los procesadores colectivamente consumen respecto al tiempo del algoritmo secuencial óptimo.

$$T_0(n, p) = C(n, p) - T(n) \quad Ec. 4$$

SPEEDUP Y EFICIENCIA

Speedup

- Expresa la ganancia de velocidad del paralelo respecto al secuencial.

$$S(n, p) = \frac{T(n)}{T(n, p)} \quad Ec. 5$$

- Todo algoritmo paralelo se puede simular en una máquina secuencial ejecutando en serie los pasos paralelos (obviando problemas de comunicaciones). Por tanto $S(n, p)$ está acotada por:

$$T(n, 1) \leq pT(n, p) \Rightarrow S(n, p) \leq p$$

- Si en el cálculo de $S(n, p)$ se usa $T(n, 1)$ como $T(n)$ (paralelo con un solo procesador) entonces $S(n, p)$ expresa la bondad del diseño paralelo.
- Si en el cálculo de $S(n, p)$ se usa el tiempo del algoritmo secuencial óptimo conocido entonces $S(n, p)$ expresa la eficacia (prestaciones) del algoritmo paralelo.

SPEEDUP Y EFICIENCIA

Speedup

- $S(n, p) = p$ *Speedup Lineal*. Las comunicaciones no penalizan, no hay dependencia de datos, los procesadores trabajan (totalmente) concurrentemente, la carga está balanceada, etc.
- $S(n, p) < p$ *Speedup Sub-lineal*. Hay dependencia de datos, no siempre es posible dividir el problema en p subproblemas concurrentes, etc.
- $S(n, p) > p$ *Speedup Super-lineal* El algoritmo secuencial no es óptimo, se ha incurrido en un error de cálculo o hay efectos colaterales (p. ej. cachés)

Eficiencia

- Grado de aprovechamiento que el algoritmo paralelo hace del sistema (trabajo útil al resolver el problema)

$$0 \leq E(n, p) = \frac{S(n, 1)}{p} = \frac{T(n, 1)}{pT(n, p)} \leq 1 \quad Ec. 6$$

EJEMPLOS

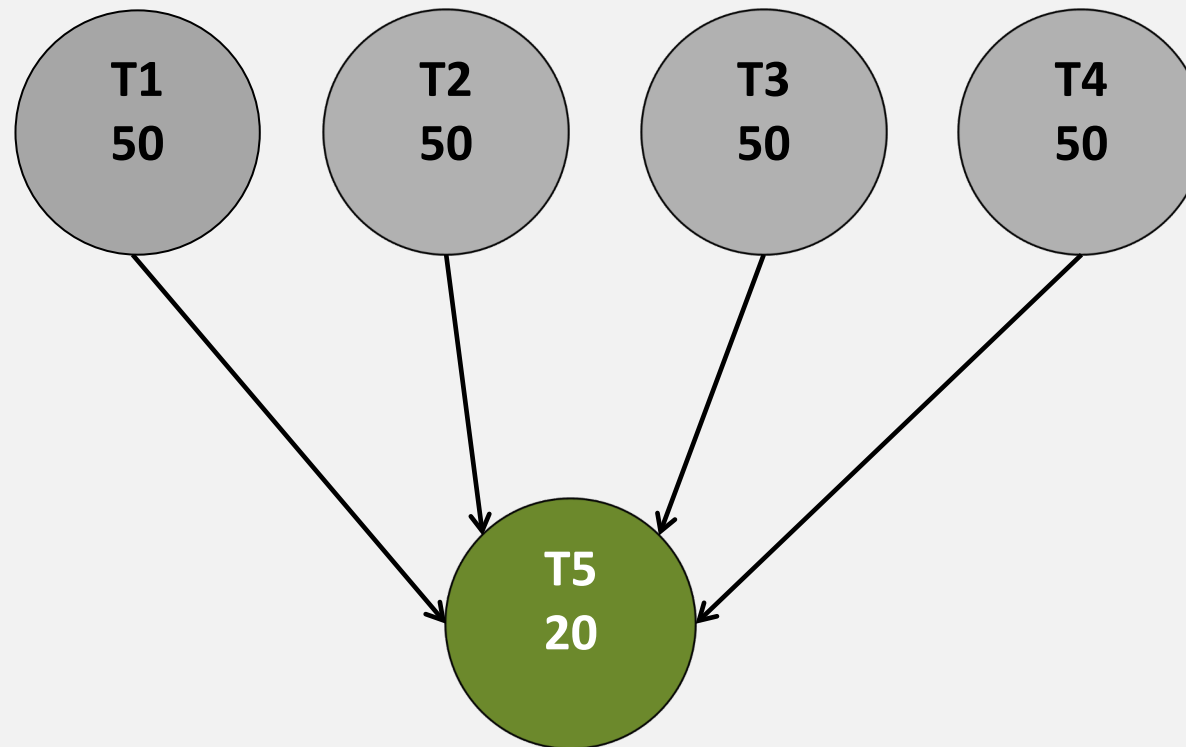
1. Estimar el tiempo de ejecución del siguiente algoritmo secuencial

```
for (i=0; i<n; i++)  
  for (j=0; j<m; j++)  
    c[i][j] = 0.0;  
    for (r=0; r<k; r++)  
      c[i][j] += a[i][r] * b[r][j];
```

$$T(n, m, k) = \sum_{i=1}^n \sum_{j=1}^m \left(1 + \sum_{t=1}^k 2 \right) t_c = nmt_c + 2nmkt_c \in \theta(nmk)$$

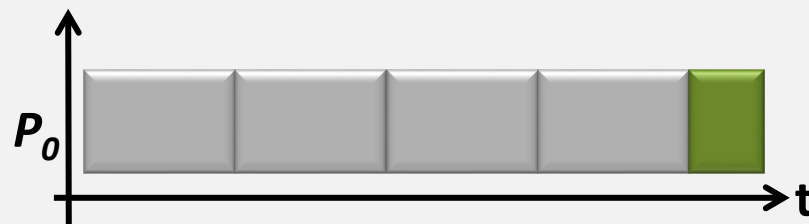
EJEMPLOS

2. Calcular el tiempo de ejecución secuencial del siguiente grafo donde el coste de cada tarea es el expresado en el dibujo. Calcular el tiempo de ejecución paralelo, coste, sobrecarga, eficiencia y speedup cuando se dispone de 4 procesadores.

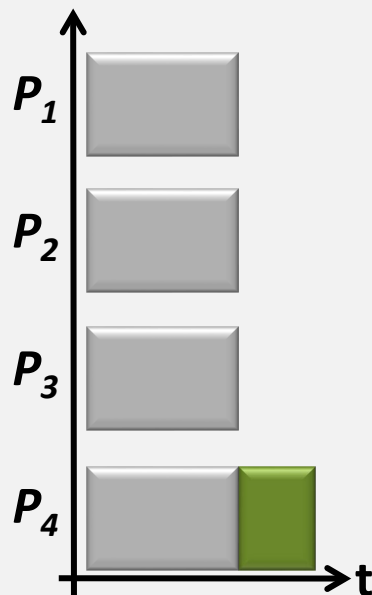


EJEMPLOS

2. Calcular el tiempo de ejecución secuencial del siguiente grafo ...



$$T(n) = 50 \times 4 + 20 = 220$$



$$T(n, 4) = 50 + 20 = 70$$

$$C(n, 4) = 4T(n, 4) = 280$$

$$T_0(n, 4) = 280 - 220 = 60$$

$$S(n, 4) = \frac{220}{70} = 3,14$$

$$E(n, 4) = \frac{3,14}{4} = 0,786$$

¿ Y con 2 y 3 procesadores ? ¿ Y con más de 4 procesadores ? ¿ Y con 4 procesadores y pesos para las tareas 1-4: 50, 20, 30 y 40, respectivamente ?

EJEMPLOS

3. Sea $p = n/2$. Estimar el tiempo de ejecución de los siguientes algoritmos, así como el coste, sobrecarga, eficiencia y speedup.

```
/* Algoritmo Secuencial */
s=0.0;
for (i=0; i<n; i++)
    s = s + v[i];
```

$$T(n) = (1 + n)t_c \cong nt_c$$

```
/* Paralelo. En cada Pi: i=0,1,..., (n/2) -1 */
in=2*i; des=1;
for (k=1; k<=log2(n) && (i % des==0); k++)
    a[in]=a[in]+a[in+des];
    des=des*2;
fin para
```

$$T(n, p) \cong (2\log_2 n)t_c$$

$$S(n, p) = \frac{n}{2\log_2 n}$$

$$C(n, p) = pT(n, p) = nt_c \log_2 n; T_0(n, p) = C(n, p) - T(n) = nt_c(\log_2 n - 1)$$

$$E(n, p) = \frac{1}{\log_2 n}$$

EJEMPLOS

4. Con $2 \leq p \leq n/2$ calcula $T(n, p)$. Comparar con el anterior ¿Qué observa?

```
/* en cada Pi: i=0,1,...,p-1 */

suma=0.0;
for (j=i*n/p; j<((i+1)*n/p); j++)
    suma=suma+a[j];

a[i]=suma; in=i*2; desp=1;
if (i % 2==0)
    activo=true;
else
    activo=false;

for (k=1; k<=log2(p); k++)
    if activo
        a[in]=a[in]+a[in + desp];
        desp=desp*2;
    if (i % desp !=0)
        activo=false
```

$$T(n, p) \cong \left(\frac{n}{p} + 2 \log_2 p \right) t_c$$

$$T(n, p = 2) \cong \left(\frac{n}{2} + 2 \right) t_c \cong \frac{n}{2} t_c$$

$$T\left(n, p = \frac{n}{2}\right) \cong 2 t_c \log_2 n$$

$$2 t_c \log_2 n \leq T(n, p) \leq \frac{n}{2} t_c$$

Si $n = 64$ y $p = n/2$ ¿ $S(n, p)$ es? ¿Y la eficiencia comparada con la del algoritmo paralelo anterior?

$$T(64, 32) \cong 12 t_c; \quad E(64, 32) \cong \frac{1}{6}$$

La eficiencia del anterior: la misma

EJEMPLOS

5. Estimar el tiempo de ejecución del siguiente algoritmo secuencial

Repetir el número de veces establecido

$$B = W^T W H$$

$$C = W^T A$$

$$H = H \circledast C \oslash B$$

$$D = W H H^T$$

$$E = A H^T$$

$$W = W \circledast E \oslash D$$

Fin repetir

Siendo $W, H, A \in \mathbb{R}^{n \times n}$, $n \geq 1$ y siendo \circledast y \oslash el producto y la división a nivel de elemento (element-wise)

$$T(n) = 2n^3(2 + 2 + 1 + 1)t_c + n^2(2 + 2)t_c$$

$$T(n) = 12n^3t_c + 4n^2t_c$$

MODELOS DE RENDIMIENTO

Ley de Amdahl (Tamaño fijo del problema)

- Mide el incremento de velocidad máximo de los algoritmos paralelos.
- El tiempo de ejecución de un algoritmo se expresa como

$$t(n) = \alpha(n) + \beta(n)$$

donde

$\alpha(n)$ es el tiempo de la parte secuencial, no paralelizable.

$\beta(n)$ es el tiempo de la parte paralelizable.

- El tiempo paralelo se puede definir como

$$t(n, p) = \alpha(n) + \frac{\beta(n)}{p}$$

- Consecuentemente $S(n, p)$ máximo será

$$\lim_{p \rightarrow \infty} \frac{t(n)}{t(n, p)} = \lim_{p \rightarrow \infty} \frac{\alpha(n) + \beta(n)}{\alpha(n) + \frac{\beta(n)}{p}} = 1 + \frac{\beta(n)}{\alpha(n)}$$

RETOMANDO LA LEY DE AMDAHL

Sea $\alpha(n) + \beta(n) = 1$, entonces

$$S(n, p) = \frac{T(n)}{T(n, p)} = \frac{p}{(p - 1)\alpha(n) + 1} \leq \frac{1}{\alpha(n)}$$

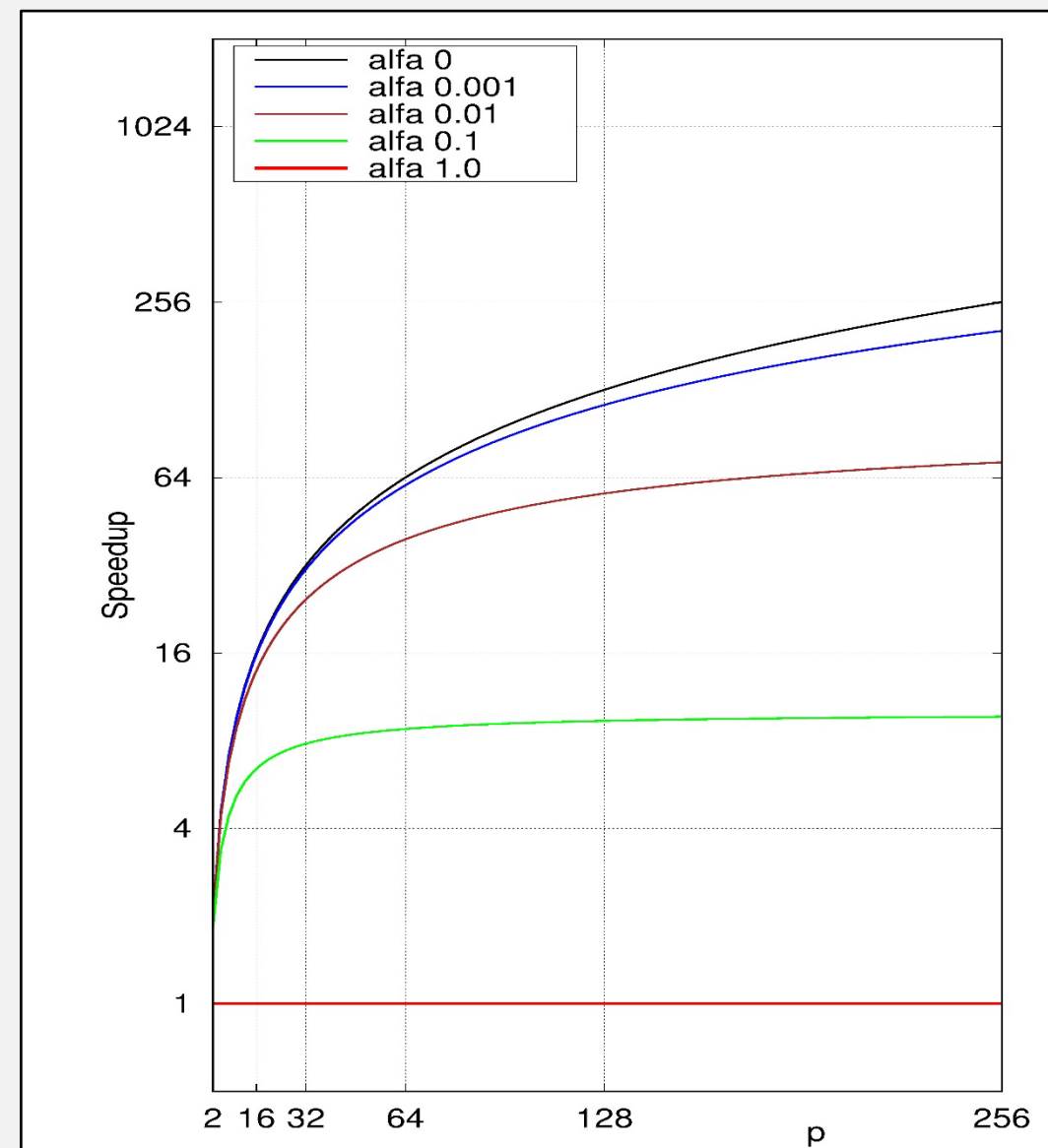
Por ejemplo:

- Algoritmo paralelizable en un 90%

$$S(n, p) = \frac{p}{(p - 1)0.1 + 1} < \frac{1}{0.1} = 10$$

- Algoritmo paralelizable en un 50%

$$S(n, p) = \frac{p}{(p - 1)0.5 + 1} < \frac{1}{0.5} = 2$$



MODELOS DE RENDIMIENTO

Ley de Amdahl

- Haciendo manipulaciones algebraicas

$$S(n, p) = \frac{1}{(1 - PF) + \frac{PF}{p}}$$

donde PF es el % de fracción paralela (expresado en tantos por uno)

- Se define **Rendimiento Efectivo** (RE) de un algoritmo paralelo como

$$RE_{dp} = \frac{1}{(1 - PF) + \frac{PF}{p}} \times clock_{GHz} \times (AVX \text{ ó } FMA)$$

$$RE_{dp} = \frac{1}{(1 - PF) + \frac{PF}{p}} \times clock_{GHz} \times \left(\frac{n^o \text{ flop}}{ciclo} \right)$$

MODELOS DE RENDIMIENTO

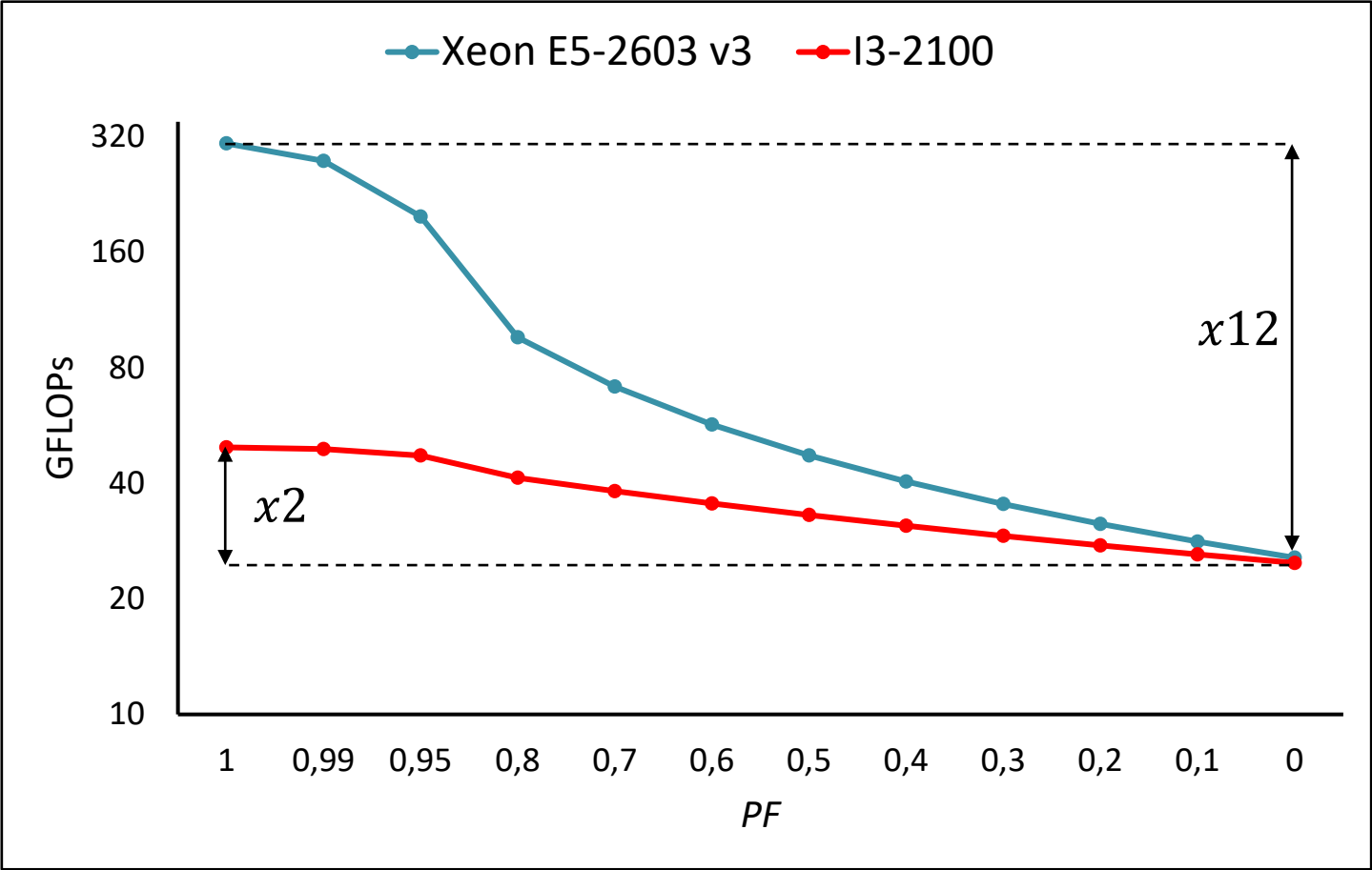
Ley de Amdahl

Modelo	Nº Sockets	Cores x Socket	Frecuencia (GHz)	Flop por cycle (DP – SP)	TPP _{dp} (Gflop)	TPP _{sp} (Gflop)
Xeon E5-2603 v3	2	6	1.6	16 – 32	307.0	614.0
Xeon PHi 31S1P	57	4	1.1	4 – 8	1003.0	2006.0
I3-2100	1	2	3.1	8 –16	49.6	99.2
Ryzen 7 3700x	1	8	2.2	16 – 32	281.6	563.2

Modelo	$Re_{dp} (PF = 0.99)$	$Re_{sp} (PF = 0.99)$	$Re_{dp} (PF = 0.95)$	$Re_{sp} (PF = 0.95)$
Xeon E5-2603 v3	276.7	553.5	198.2	396.4
Xeon PHi 31S1P	306.8	613.6	81.2	162.5
I3-2100	49.1	98.2	47.2	94.5
Ryzen 7 3700x	263.2	526.4	208.6	417.2

MODELOS DE RENDIMIENTO

Ley de Amdahl y RE



Que dice exactamente lo mismo que la gráfica de la transparencia 26

MODELOS DE RENDIMIENTO

Ley de Amdahl

- La eficiencia decrece monótonamente al aumentar el número de procesadores y mantener constante el tamaño del problema.
- El tiempo de ejecución puede aumentar al crecer el valor de p .
- No es productivo usar más procesadores que una cantidad determinada para un tamaño de problema fijo.
- Los conceptos de speedup y eficiencia permiten conocer la mejora y el grado de aprovechamiento que un algoritmo paralelo hace de una determinada configuración.
- No obstante, ambos parámetros son dependientes de n y p . Las conclusiones pueden no ser las mismas cuando alguno de estos parámetros cambie.

MODELOS DE ISORENDIMIENTO

Def. Caracterizan la escalabilidad del sistema manteniendo constante una métrica de rendimiento (eficiencia, utilización, etc.)

- Función de Isotiempo
 - Métrica/prestaciones: tiempo.
 - Recursos: número de procesadores.
- ...
- Función de Isoeficiencia
 - Métrica/prestaciones: eficiencia.
 - Recursos: número de procesadores.

Función de Isoeficiencia

- ¿Cómo debe crecer el tamaño del problema (W) en función de p para mantener la eficiencia constante?
- **Def. Algoritmo Escalable** aquel cuya función de isoeficiencia es lineal respecto al número de procesadores.

ESCALABILIDAD

- Procedimiento de cálculo **Nº 1**
 - Fijar la eficiencia a un valor deseado y despejar la carga computacional W
 - A menor aumento de W al aumentar p mayor escalabilidad.
- Ejemplo. ¿Cuál es más escalable si se desea una eficiencia de 1/3?

Algoritmo	W	$T(n,1)$	$T(n,p)$
A	n^3	n^3c	$n^3c/p + n^2b/\sqrt{p}$
B	n^3	n^3c	$n^32c/p + n^2b/2\sqrt{p}$

$$E(W,p)_A = \frac{n^3c}{\left(\frac{n^3c}{p} + \frac{n^2b}{\sqrt{p}}\right)p} = \frac{1}{1 + \frac{b\sqrt{p}}{nc}}$$

$$E(W,p)_A = \frac{1}{3} \rightarrow n = \frac{b\sqrt{p}}{2c}; \quad W = n^3 \rightarrow W = \frac{b^3}{(2c)^3} p\sqrt{p}$$

Iguales

$$E(W,p)_B = \frac{n^3c}{\left(\frac{2n^3c}{p} + \frac{n^2b}{2\sqrt{p}}\right)p} = \frac{1}{2 + \frac{b\sqrt{p}}{2nc}}$$

$$E(W,p)_b = \frac{1}{3} \rightarrow n = \frac{b\sqrt{p}}{2c}; \quad W = n^3 \rightarrow W = \frac{b^3}{(2c)^3} p\sqrt{p}$$

ESCALABILIDAD

- Ejemplo. Sean los datos de la tabla anterior. ¿Cuál es más escalable si se desea una eficiencia de 1/4?

$$E(W, p)_A = \frac{n^3 c}{\left(\frac{n^3 c}{p} + \frac{n^2 b}{\sqrt{p}}\right) p} = \frac{1}{1 + \frac{b\sqrt{p}}{nc}}$$

$$E(W, p)_A = \frac{1}{4} \rightarrow n = \frac{b\sqrt{p}}{3c}$$

$$W = n^3 \rightarrow W = \frac{b^3}{(3c)^3} p\sqrt{p}$$

Analíticamente diferentes pero *iguales* (cambia la constante)

$$E(W, p)_B = \frac{n^3 c}{\left(\frac{2n^3 c}{p} + \frac{n^2 b}{2\sqrt{p}}\right) p} = \frac{1}{2 + \frac{b\sqrt{p}}{2nc}}$$

$$E(W, p)_B = \frac{1}{4} \rightarrow n = \frac{b\sqrt{p}}{4c}$$

$$W = n^3 \rightarrow W = \frac{b^3}{(4c)^3} p\sqrt{p}$$

ESCALABILIDAD

- Procedimiento de cálculo **Nº 2** (Del libro *“Introduction to Parallel Computing...”*)

$$T_0(n, p) = C(n, p) - T(n) = pT(n, p) - T(n) \qquad T(n, p) = \frac{T(n) + T_0(n, p)}{p}$$

$$E(n, p) = \frac{T(n)}{pT(n, p)} = \frac{T(n)}{T(n) + T_0(n, p)} = \frac{1}{1 + \frac{T_0(n, p)}{T(n)}}$$

$$T(n) = \frac{E(n, p)}{1 - E(n, p)} T_0(n, p); \text{ sea } K = \frac{E(n, p)}{1 - E(n, p)} \rightarrow W \propto K T_0(n, p)$$

Función de Isoeficiencia

$$I(W, p) = \frac{pT(n, p) - T(n)}{T(n)} = \frac{O(p^r)}{O(T(n)^s)}$$

Comparar r con s . Si $r \leq s$ **algoritmo escalable**

ESCALABILIDAD

- Sean los siguientes tiempos de ejecución. Estudiar la escalabilidad.

$$T(n) = \frac{2n^3}{3} t_c$$

$$T_1(n, p) = \frac{n^3}{p} t_c + nt_s + \frac{n^2}{2} t_w$$

$$T_2(n, p) = \frac{2n^3}{3p} t_c + nt_s + \left(\frac{n^3}{3p} + \frac{n^2}{4}\right) t_w$$

$$T_3(n, p) = \frac{n^3}{p} t_c + t_s n \log p + \frac{n^2 \log p}{2} t_w$$

$$T_{01}(n, p) = \frac{n^3}{3} t_c + t_s np + \frac{n^2 p}{2} t_w$$

$$t_c: n^3 = W_1 \propto \left(K \frac{n^3}{3}\right)$$

No depende de ***p*** luego no afecta a la escalabilidad

$$t_s: n^3 = W_1 \propto Knp \cong KW_1^{\frac{1}{3}}p; \quad W_1^{\frac{2}{3}} \propto Kp; \quad W_1 \propto (Kp)^{\frac{3}{2}} \in \theta(p^{\frac{3}{2}})$$

$$t_w: n^3 = W_1 \propto K \frac{n^2}{2} p \cong K \frac{W_1^{\frac{2}{3}}}{2} p; \quad W_1^{\frac{1}{3}} \propto Kp; \quad W_1 \propto (Kp)^3 \in \theta(p^3)$$

ESCALABILIDAD

- Sean los siguientes tiempos de ejecución. Estudiar la escalabilidad.

$$T_{0^2}(n,p) = t_s np + (\frac{n^3}{3} + \frac{n^2 p}{4})t_w$$

... ..

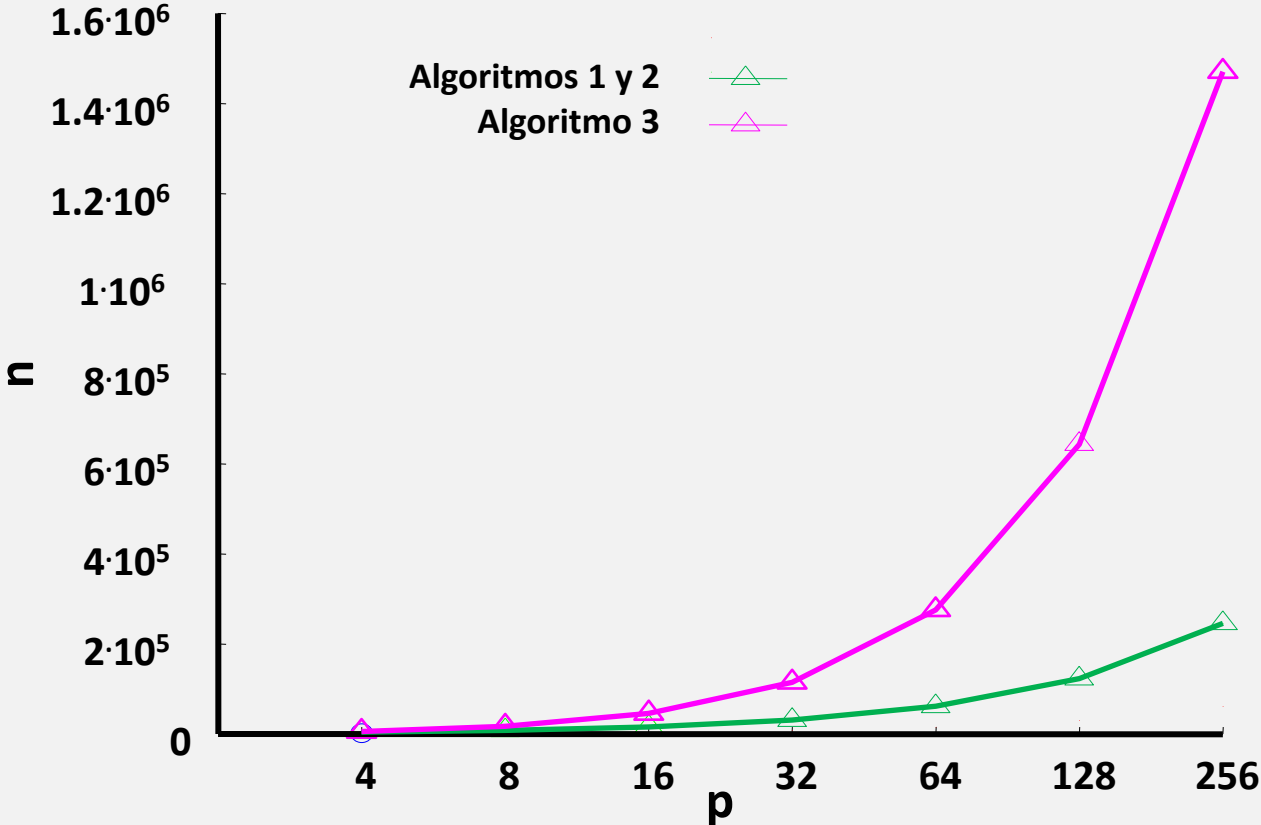
$$I_2 \in \theta(p^3)$$

$$T_{0^3}(n,p) = \frac{1}{3}n^3 t_c + t_s np \log p + \frac{n^2 p \log p}{2} t_w$$

... ..

$$I_3 \in \theta((p \log p)^3)$$

Comparativa empírica de las curvas de Isoeficiencia para **E=0.65**



ESCALABILIDAD

- Sean los siguientes tiempos de ejecución. Estudiar la escalabilidad.

$$T(n) = ZN^2t_c \quad T(n,p) = \frac{ZN^2}{p}t_c + 4(t_s + ZNt_w)$$

$$T_0(n,p) = 4p(t_s + ZNt_w) = 4pt_s + 4pZNt_w$$

$$t_s: ZN^2 = W \propto Kp \in \theta(p)$$

$$t_w: N_{Z=1}^2 = N^2 = W \propto KNp;$$

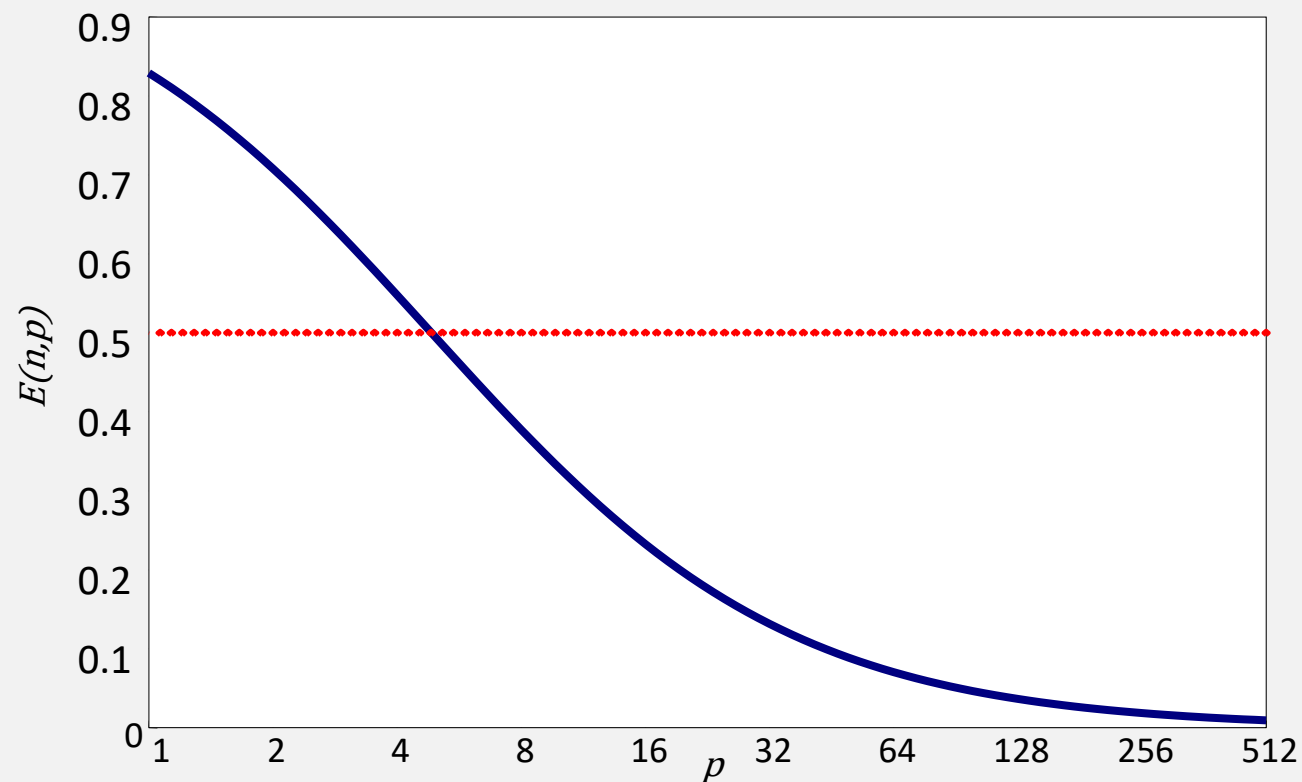
$$W^{1/2} \propto Kp; W \propto (Kp)^2 \in \theta(p^2)$$

$$t_w: N_{Z=N}^2 = N^3 = W \propto Kn^2p;$$

$$W^{1/3} \propto Kp; W \propto (Kp)^3 \in \theta(p^3)$$

$$Z=128, N=1024, p \in [1, \dots, 512]$$

$$t_c=1E^{-08}, t_s=6.3E^{-05}, t_w=1.1E^{-06}$$



EFICIENCIA ESCALABILIDAD

Otra forma de evaluar la facilidad de un sistema paralelo para mantener la eficiencia constante. El tamaño del problema y el número de procesadores se multiplican por el mismo factor (r) y se observa si el algoritmo escala, esto es:

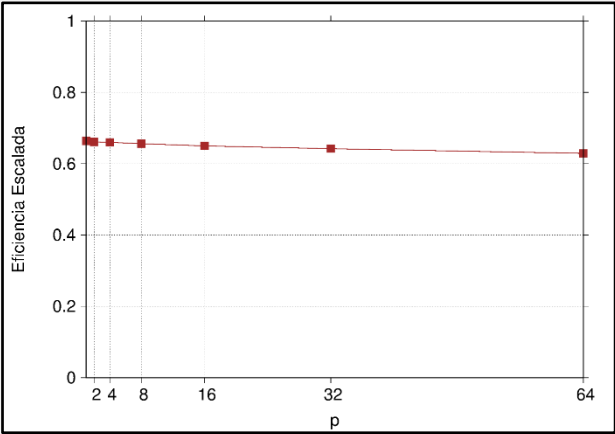
$$E_{scl}(W, r) = \frac{T_{paralelo}(W, 1)}{T_{paralelo}(rW, r)} \text{ o } \frac{T(W)}{T_{paralelo}(rW, r)}$$

Consideraciones

- Si al aumentar el tamaño del problema (W) y los procesadores en la misma cantidad la eficiencia permanece constante \rightarrow escalable.
- Algunos autores afirman que mientras $E_{scl}(W, r) > 0$ los algoritmos pueden considerarse cuasi-escalables (siempre y cuando sea constante)
- Multiplicar por 2 es una buena estrategia.
- Puede conducir a conclusiones erróneas si el intervalo de observación no es adecuado.

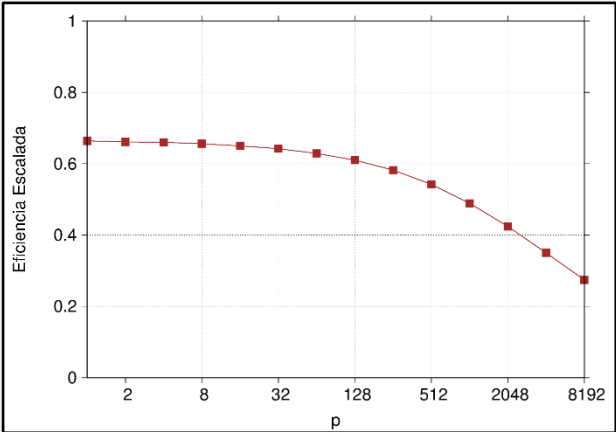
EFICIENCIA ESCALABILIDAD

- Del ejercicio $T(n) = \left(\frac{2n^3}{3}\right) t_c$ y $T_1(n, p) = \left(\frac{n^3}{p}\right) t_c + nt_s + \left(\frac{n^2}{2}\right) t_w$



La eficiencia es aprox. constante al multiplicar W y p por 2 \rightarrow escalable

La realidad es distinta, la eficiencia no es constante



SISTEMAS HÍBRIDOS/HETEROGÉNEOS

- Sistemas conformados por ítems de distinta naturaleza (arquitectura, etc.). Ejemplos: equipo con CPU+GPU, ordenadores con distintas prestaciones, etc.
- Los distintos grupos suelen hacer (no necesariamente) los mismos cálculos y/o comunicaciones.
- Generalización del modelo: varias constantes de cálculo y/o comunicaciones.
- Se puede obtener $T_i(n, p)$ de cada grupo *distinto* por separado y luego componer $T(n, p)$
- El resto del proceso es igual, con la salvedad de que analíticamente es más complejo.

Ejemplo

Sea un clúster con p ordenadores iguales conectados a una red, con t_s la constante de establecimiento y t_w la inversa del ancho de banda. Cada ordenador tiene una CPU con k núcleos, con t_c la constante de cálculo. Cada ordenador tiene la misma GPU de apoyo, siendo t_{cGPU} , t_{sGPU} y t_{wGPU} las constantes de cálculo y de comunicaciones CPU/GPU, respectivamente. La GPU es **9x** más potente que la CPU. Sea $T(n) = n^2 t_c$ y un diseño paralelo sin dependencias externas donde todos los elementos realizan la misma computación. Se pide A) tiempo de ejecución paralelo de cada ordenador, B) tiempo de ejecución del sistema si el coste de las comunicaciones CPU/GPU es 0 y C) análisis de la Escalabilidad y Eficiencia Escalada desde b)

SISTEMAS HÍBRIDOS/HETEROGÉNEOS

a) Tiempo de ejecución paralelo de cada ordenador

1. Ordenadores (p) iguales y hacen el mismo trabajo. Reparto: $\frac{n^2}{p}$ por ordenador.

2. GPU 9x CPU. Reparto dentro de cada ordenador: $\frac{n^2}{p} = x + 9x = 10x$; $x = \frac{n^2}{10p}$ para CPU y $\frac{9n^2}{10p}$ GPU

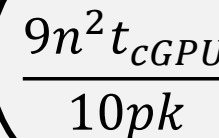
3. La ecuación del tiempo de ejecución paralelo del cada ordenador es:

$$T(n, p, k) = T_{EnviaGPU}(n) + T_{RecibeGPU}(n) + T_{swap}(n) + \max(T_{ar_GPU}(n, p, k), T_{ar_CPU}(n, p, k))$$

4. Se envía y recibe la misma cantidad de datos a/de la GPU, no hay dependencias externas y reparto balanceado ($T_{ar_GPU}(n, p, k) = T_{ar_CPU}(n, p, k)$). Entonces:

$$T(n, p, k) = 2T_{co_GPU}(n) + T_{ar_CPU}(n, p, k) = 2 \left(t_{sgpu} + \frac{9n^2}{10p} t_{wgpu} \right) + \frac{n^2 t_c}{10pk}$$

Igual de válido



$$\frac{9n^2 t_{cGPU}}{10pk}$$

SISTEMAS HÍBRIDOS/HETEROGÉNEOS

b) Tiempo de ejecución del sistema si el coste de las comunicaciones CPU/GPU es 0

$$T(n, p, k) = 2p \left(t_s + \frac{n^2}{p} t_w \right) + \frac{n^2 t_c}{10pk}$$

donde el primer término viene de distribuir/recolectar los datos entre los ordenadores, usando el modelo de comunicaciones P2P no óptimo.

c) Análisis de la Escalabilidad

$$T_0(n, p, k) = pT(n, p, k) - T(n) = \frac{n^2 t_c}{10k} + 2p^2 \left(t_s + \frac{n^2}{p} t_w \right) - n^2 t_c = \left(\frac{1}{10k} - 1 \right) n^2 t_c + 2p^2 t_s + 2n^2 p t_w$$

I_{t_c} : no depende de p , no influye en el análisis (ya visto)

I_{t_s} : $W \propto Kp^2 \in O(p^2)$

I_{t_w} : $W \propto Kpn^2$; $1 \propto Kp$

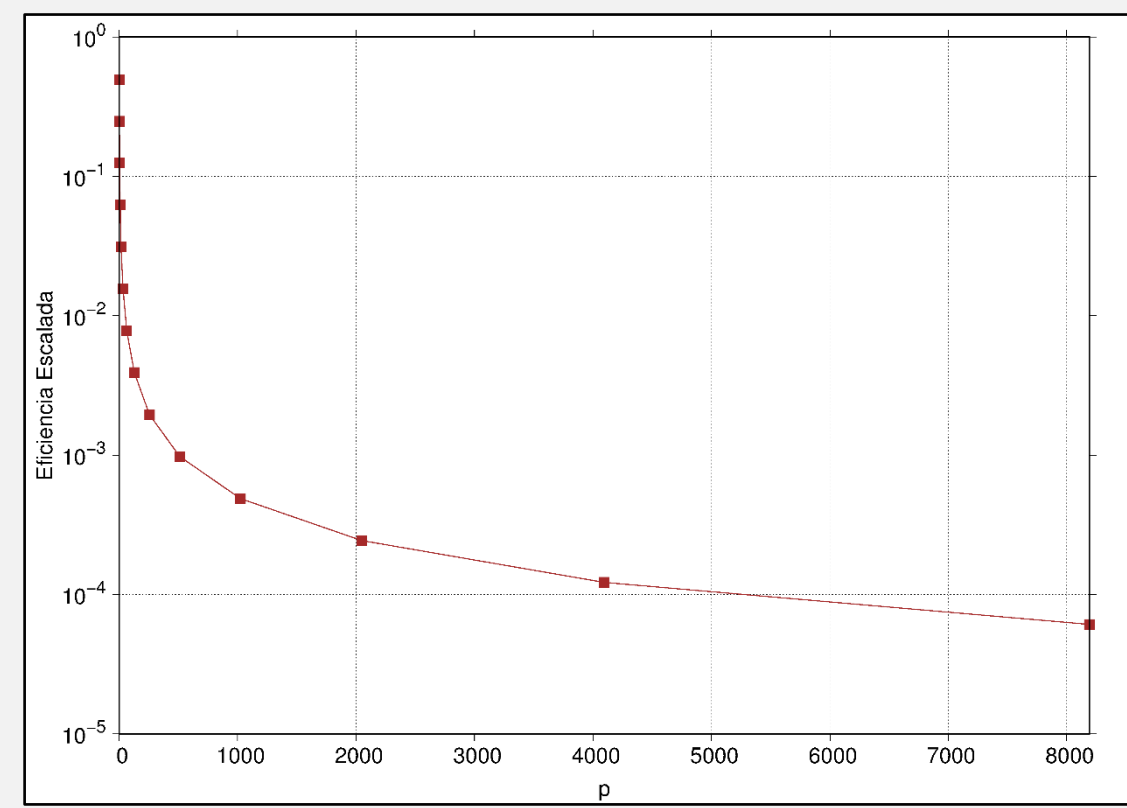
Al aumentar W y p la parte correspondiente a W no crece (función constante) y la de p crece linealmente o, si se quiere, la de W crece n^2 y pn^2 la de p $\left(I(W, p) = \frac{O(pn^2)}{O(n^2)} \right)$

Es un caso “trivial” → **no escala**

SISTEMAS HÍBRIDOS/HETEROGÉNEOS

c) Eficiencia Escalada

Suponiendo $k = 4$ y todas las constantes con valor 1. Otros valores de k y/o las constantes cambiarán la escala del eje Y, no la forma de la función.



La eficiencia no permanece constante → no escala