CMCP – Conceptos y Métodos de la Computación Paralela

Master en Computación en la Nube y de Altas Prestaciones

T5. Análisis de Algoritmos Paralelos

J. E. Roman

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Curso 2024-2025

Contenido

Apartado 1

Evaluación de Prestaciones

- Parámetros Absolutos
- Parámetros Relativos

Evaluación de Prestaciones

El principal objetivo en la computación paralela es reducir al máximo el tiempo de finalización del algoritmo

- Debemos analizar el coste de las diferentes partes de un programa paralelo

El análisis de prestaciones permite contestar a preguntas como:

- Dados dos algoritmos paralelos, ¿cuál es mejor a priori?
- ¿Qué grado de aprovechamiento del computador paralelo está haciendo el algoritmo?
- ¿Cómo varían las prestaciones al variar el número de procesadores? ¿Y el tamaño del problema?

Tipos de Análisis

Análisis a priori (teórico)

- Realizado antes de la implementación del programa, sobre el pseudocódigo o el diseño del programa
- Independiente de la máquina en la que se ejecuta
- Permite ver la forma en que crece el coste en función de la talla del problema, n
- El coste se expresa en número de operaciones (*flops*)

Análisis a posteriori (experimental)

- Realizado sobre una implementación y máquina concreta y utilizando un determinado conjunto de datos de entrada
- Permite analizar cuellos de botella y detectar condiciones no observadas en el diseño
- El coste se mide en segundos
- Se mide el tiempo real (de pared) del fragmento relevante, con primitivas portables (`omp_get_wtime`, `MPI_Wtime`)

Concepto de Flop

Flop: *floating point operation* - unidad de medida para:

- Coste de los algoritmos
- Rendimiento de los computadores (flop/s)

1 flop = coste de una operación elemental en coma flotante (producto, suma, división, resta)

- Consideramos despreciable el coste de las operaciones en aritmética entera
- El coste de otras operaciones en coma flotante se evaluará en base al Flop

  → por ejemplo, una raíz cuadrada igual a 8 flops

Supone una unidad de medida del coste independiente de la máquina (el tiempo que tarda un flop varía de un procesador a otro)

Notación Asintótica

Notación $\mathcal{O}$

- Permite acotar superiormente, salvo constantes y asintóticamente, la forma en que crece una función
- En la práctica se corresponde con el término de orden superior de la expresión del coste sin considerar su coeficiente
  - Ejemplo: la multiplicación matriz por vector es $\mathcal{O}(n^2)$

Notación o (o-pequeña)

- Tiene en cuenta además el coeficiente de mayor orden
- Adecuado cuando comparamos dos algoritmos que tienen el mismo orden $\mathcal{O}$
  - Ejemplo: el producto de matriz triangular por vector puede realizarse mediante el algoritmo convencional con coste $o(2n^2)$ o mediante un algoritmo optimizado $o(n^2)$

Parámetros Absolutos

- Tiempo de ejecución de un algoritmo secuencial: $t(n)$
- Tiempo de ejecución de un algoritmo paralelo: $t(n,p)$
  - Tiempo aritmético: $t_a(n,p)$
  - Tiempo de comunicaciones: $t_c(n,p)$
- Coste total: $C(n,p)$
- *Overhead*: $t_o(n,p)$

Notación:

- Cuando la talla del problema es siempre n, sin ambigüedad, se omitirá, por ejemplo: $t(p)$
- A veces usaremos subíndices en vez de funciones: t_p, C_p

Tiempo de Ejecución Secuencial

Expresiones útiles para el cálculo del coste computacional:

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

```
x = 0
PARA i=1,2,...,n
  PARA j=1,2,...,n
    x(i) = x(i)+a(i,j)*b(j)
  FPARA
FPARA
```

$$t(n) = \sum_{i=1}^{n}\sum_{j=1}^{n} 2$$
$$= 2\sum_{i=1}^{n} n = 2n^2 \text{ flops}$$

```
PARA j=n,n-1,...,1
  x(j) = b(j)/u(j,j)
  PARA i=1,2,...,j-1
    b(i) = b(i)+u(i,j)*x(j)
  FPARA
FPARA
```

$$t(n) = \sum_{j=1}^{n}\left(1 + \sum_{i=1}^{j-1} 2\right)$$
$$= \sum_{j=1}^{n}(1 + 2(j-1))$$
$$= 2\sum_{j=1}^{n} j - n \approx n^2 \text{ flops}$$

Tiempo de Ejecución Paralelo

Tiempo que tarda un algoritmo paralelo en p procesadores

- Desde que empieza el primero hasta que acaba el último

Se descompone en tiempo aritmético y de comunicaciones

$$t(n,p) = t_a(n,p) + t_c(n,p)$$

t_a corresponde a todos los tiempos de cálculo

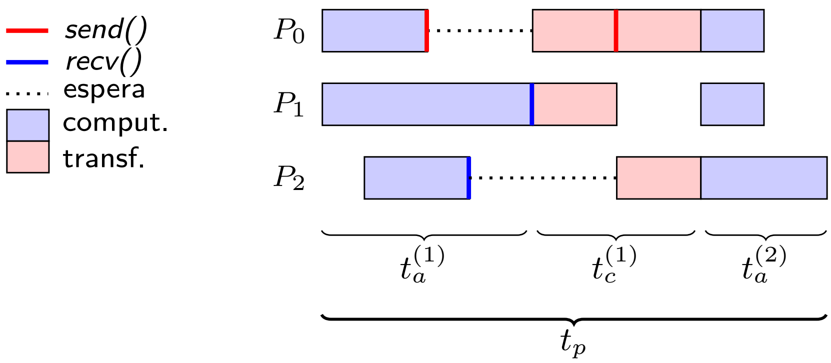
- Todos los procesadores calculan concurrentemente
- Es como mínimo igual al máximo tiempo aritmético

t_c corresponde a tiempos asociados a transferencia de datos

- En memoria distribuida t_c=tiempo de envío de mensajes
- En memoria compartida t_c=tiempo de sincronización

Tiempo de Ejecución Paralelo: Componentes

Ej.: paso de mensajes con tres procesos, P_0 envía a P_1 y P_2



En la práctica:

- No hay separación clara entre fases de cálculo y comunicación (P_1 no tiene que esperar)
- A veces se puede solapar comunicación y cálculo (con operaciones no bloqueantes, por ejemplo P_2)

$$t_p = t_a + t_c - t_{\mathrm{overlap}}$$ t_{overlap}: tiempo de solapamiento

Modelado del Tiempo de Comunicación

Suponiendo paso de mensajes, P_0 y P_1 en nodos distintos con conexión directa

Tiempo necesario para enviar un mensaje de n bytes: $t_s + t_w n$

- Tiempo de establecimiento de la comunicación, t_s
- Ancho de banda, w (máximo número de bytes por seg.)
- Tiempo de envío de 1 byte, $t_w = 1/w$

En la práctica es más complicado:

- Red conmutada, de latencia no uniforme, colisiones, ...

Recomendaciones:

- Agrupar varios mensajes en uno solo (n grande, t_s único)
- Evitar muchas comunicaciones simultáneas

En memoria compartida, las consideraciones son distintas

Ejemplo: Sistema Triangular

Resuelve un sistema de ecuaciones triangular inferior $Lx = b$, con L y b distribuidos por bloques de filas consecutivas

Secuencial

```
PARA j=1,2,...,n
  x(j) = b(j)/L(j,j)
  PARA i=j+1,j+2,...,n
    b(i) = b(i)-L(i,j)*x(j)
  FPARA
FPARA
```

Paralelo

```
EN CADA P(pr), pr=0 HASTA p-1
  nb = n/p
  PARA j=1,2,...,n
    SI fila j∈P(pr)
      x(j) = b(j)/L(j,j)
      send(x(j),k) PARA k>pr
    SI NO
      SI pr>j/nb, recv(x(j))
    FSI
    PARA i=j+1,j+2,...,n
      SI fila i∈P(pr)
        b(i) = b(i)-L(i,j)*x(j)
      FSI
    FPARA
  FPARA
```

$T_1 = n^2$ flops
$T_p = \frac{2n^2}{p}$ flops $+ \frac{np}{2}(t_s + t_w)$

Coste Total y *Overhead*

La ejecución de un algoritmo paralelo suele implicar un tiempo extra con respecto del algoritmo secuencial

El coste total paralelo contabiliza el total de tiempo empleado en un algoritmo paralelo

$$C(n,p) = p \cdot t(n,p)$$

El *overhead* indica cuál es el coste añadido con respecto del algoritmo secuencial

$$t_o(n,p) = C(n,p) - t(n)$$

Parámetros Relativos

Los parámetros relativos sirven para comparar un algoritmo paralelo con otro

- Velocidad y tiempo medio de una operación
- Speedup: $S(n,p)$
- Eficiencia: $E(n,p)$

Normalmente se aplican en el análisis experimental, aunque el speedup y la eficiencia se pueden obtener en el análisis teórico

Velocidad y Tiempo Medio de una Operación

Velocidad $V(n,p)$: número de operaciones en coma flotante por unidad de tiempo

- Se obtiene dividiendo el número de flops de un algoritmo por el tiempo en segundos consumido
- Indica cuan cerca/lejos estamos de la potencia de pico
- La unidad elemental de medida es el flop/seg (se suelen usar múltiplos como Mflop/seg, Gflop/seg, etc.)
- A menudo se dice Gflops refiriéndose a Gflop/seg

Tiempo medio de una operación:

$$t_m(n,p) = \frac{1}{V(n,p)}$$

Es el tiempo medio que se requiere para ejecutar un flop

Speedup y Eficiencia

El *speedup* indica la ganancia de velocidad que consigue el algoritmo paralelo con respecto a un algoritmo secuencial

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

Hay que indicar a qué se refiere $t(n)$

- Puede ser el mejor algoritmo secuencial conocido
- Puede ser el algoritmo paralelo ejecutado en 1 procesador

La eficiencia mide el grado de aprovechamiento que un algoritmo paralelo hace de un computador paralelo

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

Suele expresarse en tanto por cien (o tanto por 1)

Speedup: Casos Posibles

$S(n,p) < 1$

"Speed-down"

El algoritmo paralelo es más lento que el algoritmo secuencial

$1 < S(n,p) < p$

Caso sublineal

El algoritmo paralelo es más rápido que el secuencial, pero no aprovecha toda la capacidad de los proc.

$S(n,p) = p$

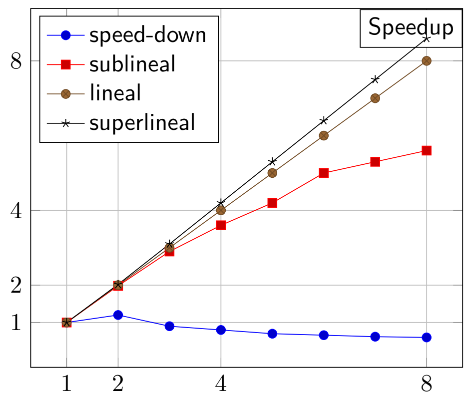
Caso lineal

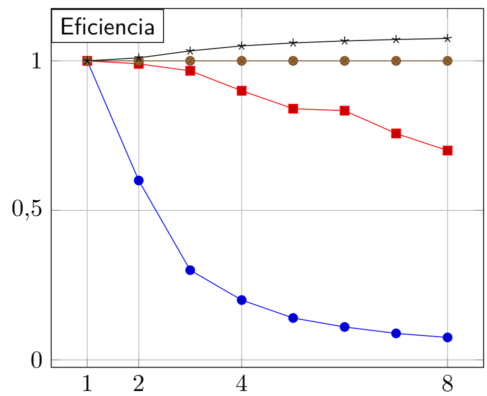
El algoritmo paralelo es lo más rápido posible, aprovechamiento de los procesadores al 100 %

$S(n,p) > p$

Caso superlineal

Situación anómala, el algoritmo paralelo tiene menor coste que el secuencial





Apartado 2

Análisis de Algoritmos Paralelos

- Ley de Amdahl
- Escalabilidad
- Modelo Roofline

Comportamiento de un Algoritmo Paralelo

Es interesante analizar el comportamiento que un algoritmo paralelo tendrá ante cambios en las condiciones de trabajo

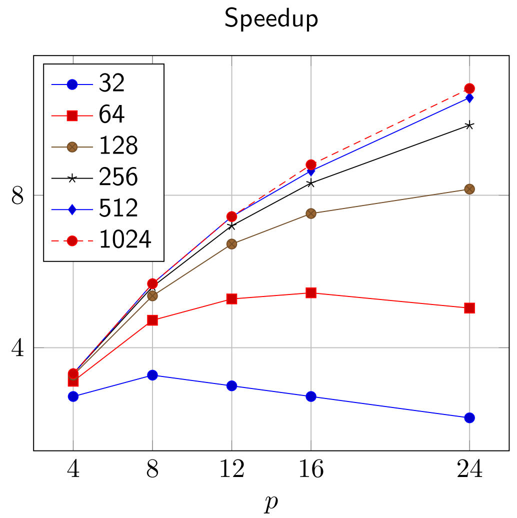
- Analizar la bondad de un algoritmo paralelo para abordar casos de mayor complejidad
- Precisar los aspectos que pueden ser la causa de la pérdida de prestaciones de un algoritmo paralelo

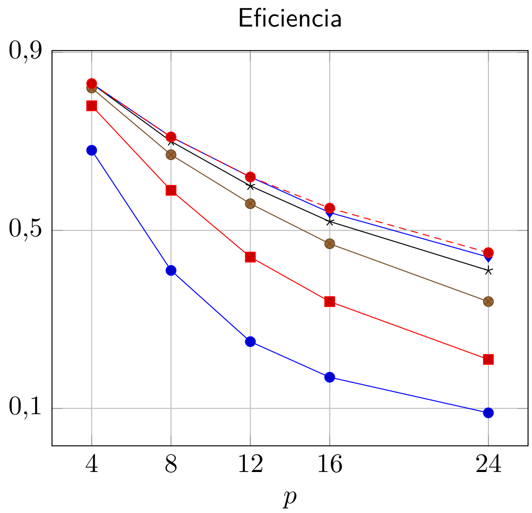
Los conceptos de speedup y eficiencia permiten conocer la mejora y el grado de aprovechamiento que un algoritmo paralelo hace de una determinada configuración

No obstante, ambos parámetros son muy dependientes del tamaño del problema y del número de procesadores, por lo que las conclusiones no tienen por qué ser las mismas cuando alguno de estos parámetros cambia

Variación de Prestaciones

- Normalmente la eficiencia disminuye a medida que se incrementa el número de procesadores
- El efecto es menos acusado para tamaños de problema más grandes





Ley de Amdahl

Muchas veces una parte del problema no se puede paralelizar
→ La Ley de Amdahl mide el speedup máximo alcanzable

Dado un algoritmo secuencial, descomponemos $t(n) = t_s + t_p$

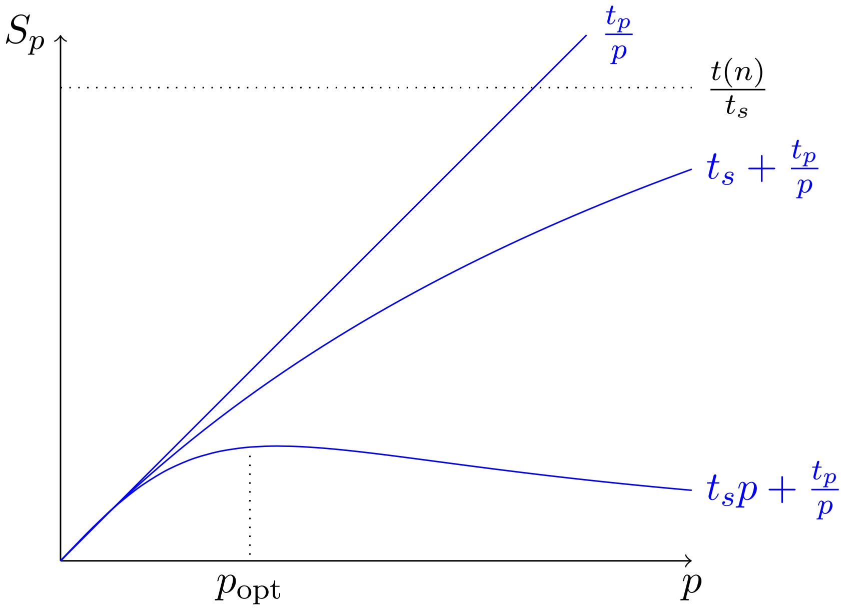
- t_s es el tiempo de la parte intrínsecamente secuencial
- t_p es el tiempo de la parte perfectamente paralelizable (se puede resolver con p procesadores)

El tiempo mínimo paralelo alcanzable será $t(n,p) = t_s + \frac{t_p}{p}$

Speedup máximo:

$$\lim_{p\to\infty} S(n,p) = \lim_{p\to\infty} \frac{t(n)}{t(n,p)} = \lim_{p\to\infty} \frac{t_s + t_p}{t_s + \frac{t_p}{p}} = 1 + \frac{t_p}{t_s}$$

Ley de Amdahl: Representación



El número óptimo de procesadores para resolver un problema de tamaño fijo se puede obtener minimizando la expresión del tiempo paralelo: $\frac{dt(n,p)}{dp} = 0$

Ley de Amdahl: Consecuencias

La ley de Amdahl también se puede expresar así

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

donde $\alpha \in [0,1]$ es la fracción no paralelizable. Ejemplos:

- Si un algoritmo es paralelizable en un 90 %, el speedup máximo será $S(n,p) = p/((p-1) \cdot 0{,}1 + 1) < 1/0{,}1 = 10$
- Si un algoritmo es paralelizable en un 50 %, el speedup máximo será $S(n,p) = p/((p-1) \cdot 0{,}5 + 1) < 1/0{,}5 = 2$

Visión más optimista: Ley de Gustafson $S(n,p) = p - \alpha(p-1)$

- Más procesadores implica resolver problemas más grandes
- Hay que considerar $t_s(n)$ y $t_p(n)$ - habitualmente $t_s(n)$ es constante o crece más lentamente que $t_p(n)$
- Gustafson supone $t(n) = t_s + pt_p$ y α decrece con n

Escalabilidad. Isoeficiencia

Es interesante estudiar cómo escala un algoritmo paralelo, al variar p y n

- Se dice que presenta una buena escalabilidad si al aumentar p se siguen manteniendo las prestaciones sin tener que incrementar mucho el tamaño n

La función de Isoeficiencia, $I(p)$, indica la forma en que debe crecer n para mantener la eficiencia constante

$$E(n,p) = \frac{t(n)}{pt(n,p)} = \frac{t(n)}{C(n,p)} = \frac{t(n)}{t_o(n,p) + t(n)}$$

Si $E(n,p) = \text{cte}$, se tiene $t(n) \propto t_o(n,p)$. Para obtener $I(p)$ se comparan los términos de mayor orden de t y t_o

- Si $I(p) = p$ la talla n debe crecer proporcionalmente a p
- Si $I(p) = p^2$ la talla n debe crecer con p^2

Escalabilidad Fuerte y Débil

Para estudiar las prestaciones de un algoritmo, a veces se hacen dos análisis

- Strong scaling: ver cómo se comporta el speedup para un tamaño de problema constante
- Weak scaling: analizar el caso en que la talla del problema crece con el número de procesadores, de forma que el trabajo por procesador se mantiene más o menos constante

---

Métricas para *weak scaling*

- Isotiempo: se hace crecer C y p en la misma proporción, $C = kC_0$ $p = kp_0$, el tiempo paralelo t_p idealmente se mantiene constante
- Speedup escalado: el speedup relativo a kC_0 en lugar de C_0

Degradación de Prestaciones

Posibles causas de la degradación de prestaciones:

- Muchas dependencias de datos
  - Alto porcentaje de tiempo secuencial
  - Muchas comunicaciones
- Mala distribución de la carga
- Alta sobrecarga (*overhead*)

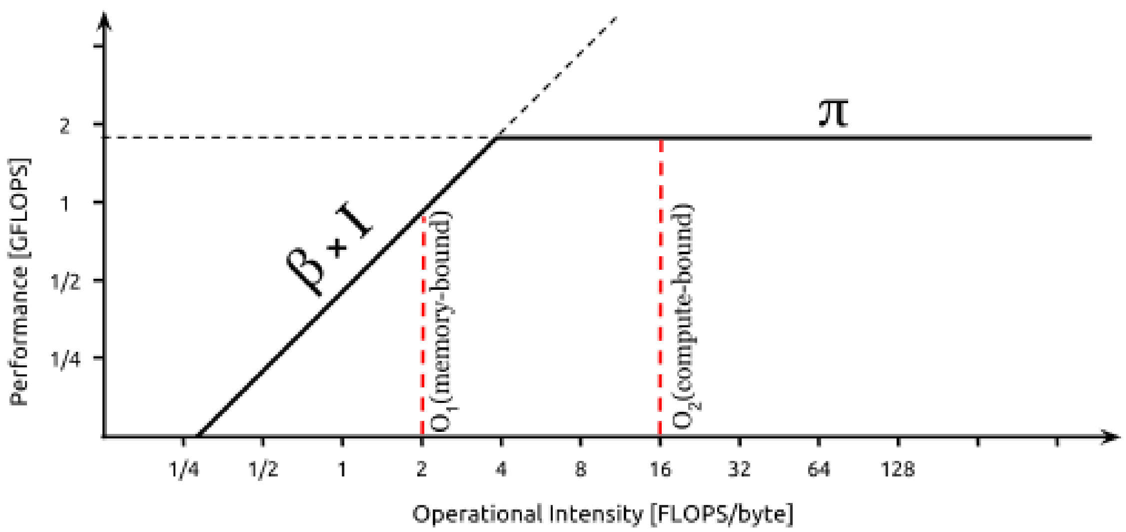
Algunas recomendaciones:

- Eliminar barreras/sincronizaciones innecesarias
- Minimizar el tiempo secuencial (p.e., regiones críticas)
- Repartir el trabajo equitativamente
- Aumentar la granularidad

Modelo Roofline

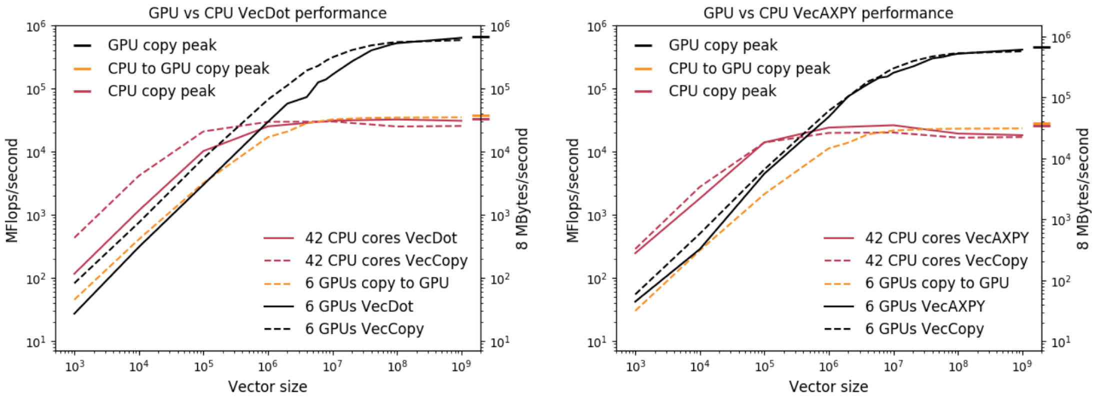
Para estimar el rendimiento visualmente, a partir de la localidad y las características de la máquina

- Intensidad aritmética $I = W/Q$, donde W es el trabajo realizado (flops) y Q el número de bytes transferidos
- Límites del hardware: π potencia de pico, β ancho de banda máximo



Ejemplo Real

Intesidad aritmética difícil de estimar (depende de la cache) $\longrightarrow$
Alternativa: ejecutar para diferentes tamaños



Fuente: H. Morgan et al. "Evaluation of PETSc on a Heterogeneous Architecture the OLCF Summit System", Tech. Rep. ALN-19/41 (2020)

- El rendimiento de GPU es mayor para tamaños grandes
- La CPU es competitiva si el problema es pequeño