3.2 Métricas de rendimiento de algoritmos paralelos

Computación de Altas Prestaciones
Grado en Ingeniería de Computadores

Raúl Durán Díaz

Departamento de Automática
Universidad de Alcalá

Curso académico 2018–2019

Universidad de Alcalá
Revisión: 1.10

Introducción

- Nos vamos a ocupar del modelado analítico de los programas paralelos.
- Definimos qué métricas vamos a usar y las acotaremos para distintos casos. Algunas son intuitivas:
  - tiempo de ejecución;
  - mejora de tiempo gracias al paralelismo;
  - ...
- La idea es **cuantificar el rendimiento**.

Contenidos

Notación asintótica

Definición

Si f y g son dos funciones de variable discreta con términos no negativos, entonces

$$f(n) = O(g(n))$$

indica que existe una constante $c > 0$ y un entero n_0 tales que

$$f(n) \leq cg(n), \quad \forall n \geq n_0.$$

Comentario

Informalmente, si $f(n) = O(g(n))$, entonces f crece como g o *más despacio*. En otras palabras, f está *acotada superiormente* por g.

Universidad de Alcalá

Notación asintótica

Definición

Sean f y g igual que en la definición anterior. Entonces

$$f(n) = \Omega(g(n))$$

indica que existe una constante $c > 0$ y un entero n_0 tales que

$$cg(n) \leq f(n), \quad \forall n \geq n_0.$$

Comentario

Es lo opuesto a lo anterior, esto es, si $f(n) = \Omega(g(n))$, entonces g crece como f o *más despacio*. En otras palabras, f está *acotada inferiormente* por g.

Notación asintótica

Definición

Sean f y g igual que en la definición anterior. Entonces

$$f(n) = \Theta(g(n))$$

indica que existen constantes $c_1, c_2 > 0$ y un entero n_0 tales que

$$c_1 g(n) \leq f(n) \leq c_2 g(n), \quad \forall n \geq n_0.$$

Comentario

Si $f = \Theta(g)$, entonces f y g crecen igual, es decir, se verifica simultáneamente $f = O(g)$ y $f = \Omega(g)$.

Notación asintótica

Ejemplo

Si $p(n)$ es un polinomio de grado d, entonces $p(n) = \Theta(n^d)$. Es decir, un polinomio crece como lo hace su término de mayor grado.

Hecho

Si $c > 1$ es un entero y $p(n)$ un polinomio, entonces $p(n) = O(c^n)$, pero $p(n) \neq \Omega(c^n)$. En otras palabras, un polinomio siempre crece estrictamente más despacio que una exponencial. Tomando logaritmos, se verifica también que $\log n = O(n)$.

Fuentes de sobrecarga

- A doble cantidad de recursos, debiera corresponder doble velocidad. . . pero esto es falso.
- Consideramos diversas fuentes de sobrecarga:
  - Interacción entre procesos.
  - Procesos inactivos por diversas causas.
  - Computación añadida.

Perfiles de ejecución

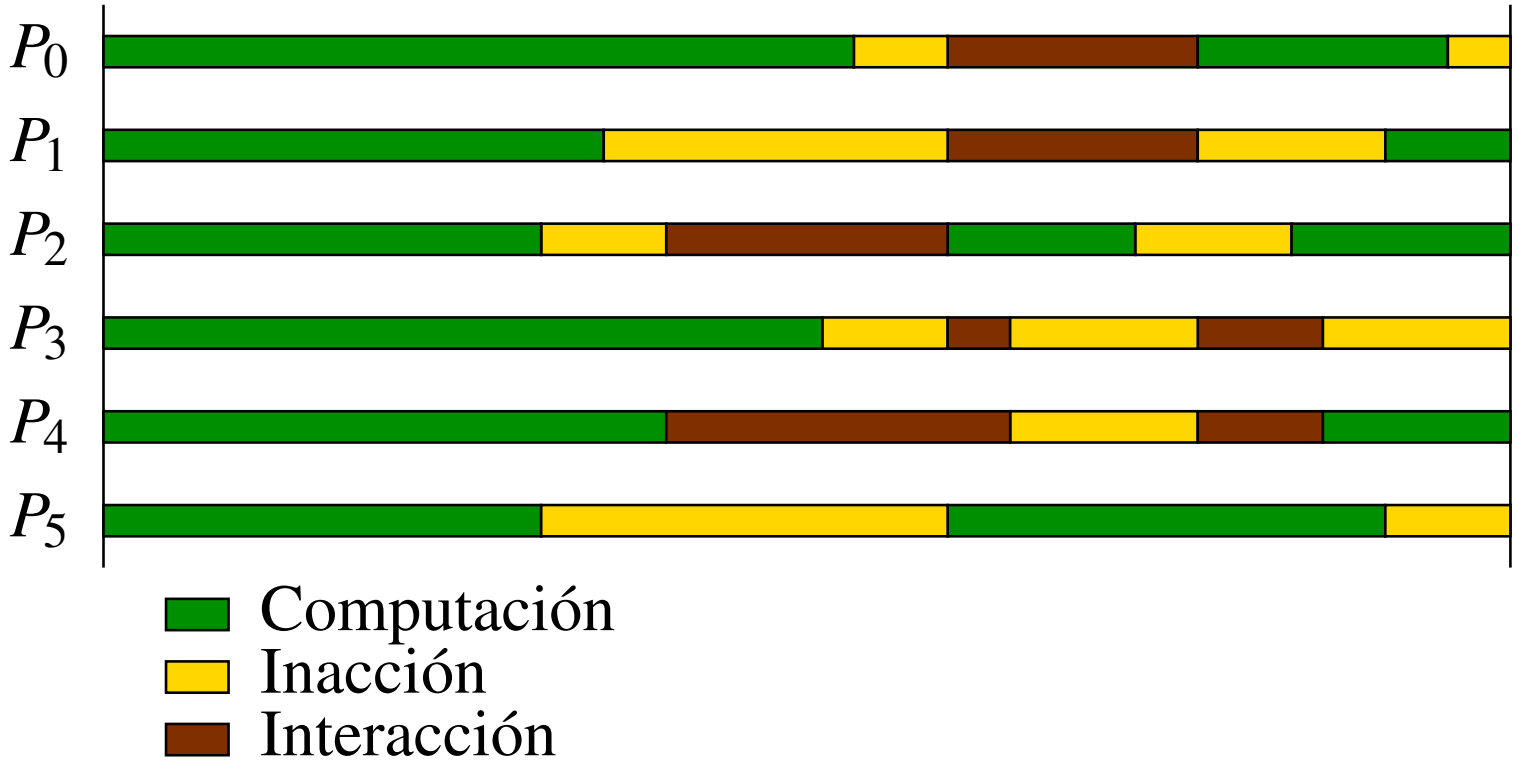

Figura: Perfil de ejecución de varios procesos

Interacción entre procesos

Sobrecarga por interacción

Tiempo necesario para comunicar datos entre los procesos: suele ser la fuente más importante de sobrecarga en la programación paralela.

Inactividad

Sobrecarga por inactividad

- Desequilibrio de carga computacional (unos procesos han de computar más que otros).
- Desequilibrio por diferencias en el hardware (un procesador es más rápido que otro).
- Sincronización entre procesos (unos procesos han de esperar a otros).
- Porciones indivisibles (tareas secuenciales que solo puede hacer un proceso).

Computación añadida

Sobrecarga por computación añadida

Cuando se paraleliza un algoritmo serie, es frecuente que haya que introducir cambios que provocan un aumento de la computación en el algoritmo paralelo con respecto a su equivalente secuencial.

Contenidos

Tiempo de ejecución

Definición

El **tiempo de ejecución serie** de un programa es el tiempo que pasa entre el comienzo y el fin del programa en un computador secuencial. Lo denotamos T_S.

Definición

El **tiempo de ejecución paralelo** de un programa es el tiempo que pasa entre el comienzo del primer proceso y el fin del último proceso involucrados en la ejecución de un programa en un computador paralelo. Lo denotamos T_P.

Sobrecarga total

Definición

Si tenemos un sistema con p elementos de proceso, denominamos **sobrecarga computacional** al siguiente valor:

$$T_O = pT_P - T_S. \quad (1)$$

Ganancia o *speedup*

Definición

Dado un sistema dotado con p elementos de proceso iguales, definimos la **ganancia** G como la relación de tiempos de ejecución serie frente a paralelo:

$$G = \frac{T_S}{T_P}.$$

- En principio, $G \leq p$.

Suma de n números con n elementos de proceso

Ejemplo

Obviamente, $T_S = \Theta(n)$. Para implementar el paralelismo, podemos suponer que n es potencia de 2 y que usamos un algoritmo en árbol, con lo que $T_P = \Theta(\log n)$. Por tanto:

$$G = \Theta\left(\frac{n}{\log n}\right).$$

Suma en paralelo

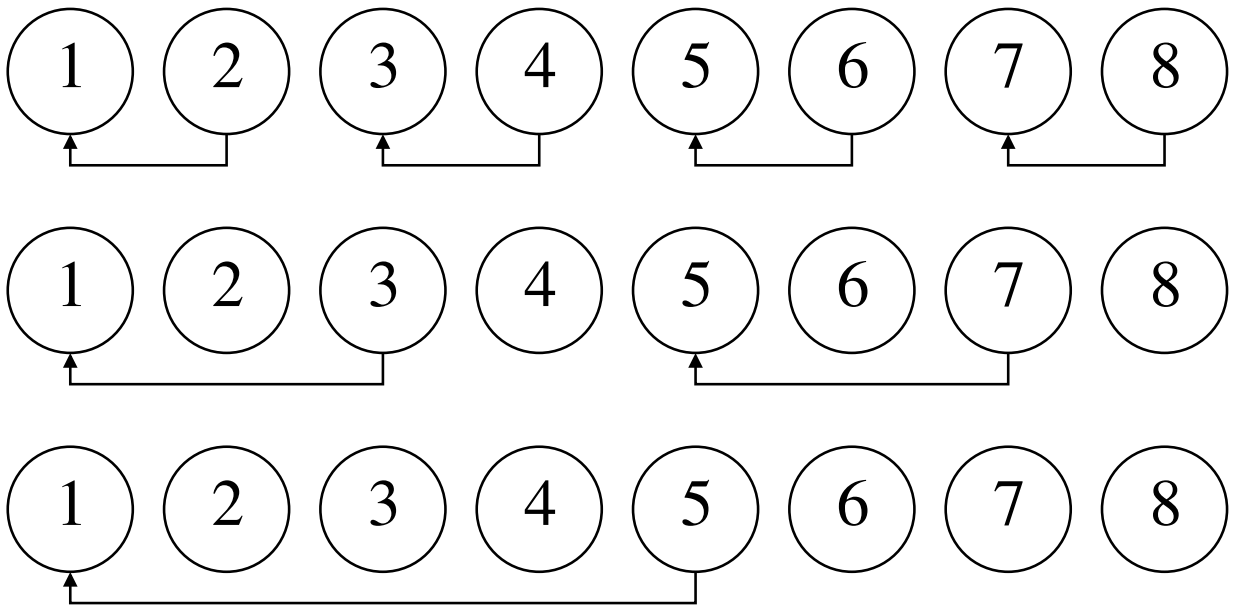

Figura: Sumando 8 números con 8 elementos de proceso

Eficiencia

Definición

Denominamos **eficiencia** a la proporción de tiempo durante el cual un elemento de proceso está siendo útil. Se calcula como:

$$E = \frac{G}{p}.$$

Comentario

Observemos que la ganancia es, como mucho, p momento en que tenemos la eficiencia unitaria, esto es, todos los elementos de proceso están activos durante todo el tiempo de proceso paralelo.

Eficiencia de la suma paralela

Ejemplo

La suma paralela presenta una eficiencia de:

$$E_{\text{suma}} = \frac{\Theta\left(\frac{n}{\log n}\right)}{n} = \Theta\left(\frac{1}{\log n}\right).$$

Coste

Definición

Denominamos **coste** C al producto

$$C = pT_P.$$

Comentario

El **coste** cuando se usa un solo elemento de proceso es el tiempo de ejecución del algoritmo secuencial más rápido que se conozca.

Coste óptimo

Definición

Decimos que un sistema paralelo es de **coste óptimo** si el coste de resolver un problema en tal sistema crece asintóticamente con relación al tamaño de la entrada al mismo ritmo que lo hace el algoritmo secuencial óptimo equivalente.

Ejemplo

El coste de la suma de n números sobre n elementos de proceso es $\Theta(n \log n)$. Puesto que el coste secuencial es $\Theta(n)$, el algoritmo no es de coste óptimo.

Efecto de la granularidad

- En la práctica, los ejemplos anteriores no son realistas, pues asumen que $n \approx p$, cuando lo cierto es que $n \gg p$.
- Si adoptamos el método de hacer que cada proceso acepte realizar el trabajo que antes realizaban n/p procesos, el coste no cambia: ni mejora ni empeora.
- Sin embargo, con algunas modificaciones en el algoritmo, en ocasiones podemos mejorar el coste e, incluso, optimizarlo si no lo era.

Sumar n elementos en p elementos de proceso

Ejemplo

- El tiempo de proceso sería $\Theta\left((n/p)\log p\right) + \Theta\left(n/p\right) = \Theta\left((n/p)\log p\right)$.
- El coste paralelo sería $\Theta\left(n \log p\right)$. Puesto que el coste serie sigue siendo $\Theta(n)$, no es de coste óptimo.

Suma en paralelo con p elementos de proceso

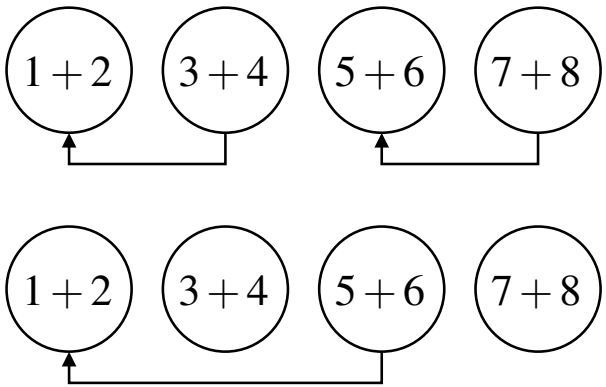

Figura: Sumando 8 números con 4 elementos de proceso

Sumar n elementos en p elementos de proceso

Ejemplo

- Si lo cambiamos, según la figura anterior, entonces

$$T_P = \Theta(n/p + \log p) \quad \Rightarrow \quad C = pT_P = \Theta(n + p \log p).$$

Con tal que $n = \Omega(p \log p)$ (cosa razonable), se tiene que $C = \Theta(n)$, lo que sí resulta óptimo.

Contenidos

Escalabilidad de sistemas

- El desarrollo de programas paralelos suele hacerse con versiones «de salón», usando entradas mucho más pequeñas que las que realmente se usarán más tarde.
- La cuestión es: ¿cómo cambia el rendimiento cuando el tamaño del problema se «escala hacia arriba»?
- Con frecuencia, una extrapolación simple no funciona bien.

Jugando con la eficiencia

- Recordando la ecuación (1), se tiene que

$$E = \frac{G}{p} = \frac{T_S}{pT_P} = \frac{1}{1 + \frac{T_O}{T_S}}, \qquad (2)$$

Ganancia y número de elementos de proceso

- Si en la ecuación (2), mantenemos constante T_S, es decir, el tamaño de la entrada, sería necesario que T_O se mantuviera constante para que E no baje.
- Sin embargo, como todo programa tiene una cierta componente secuencial (no paralelizable), T_O se incrementa como función de p. De hecho, si llamamos t_s al tiempo no paralelizable, tendremos un tiempo mínimo de sobrecarga por inactividad de $(p-1) \times t_s$, que es lineal en p.
- En muchos casos, el crecimiento es súper-lineal.

Ganancia y número de elementos de proceso

Ejemplo

Sea el caso de la suma de n números con p elementos de proceso. Supongamos que todas las operaciones toman tiempo unitario, incluida la comunicación.

- Primera fase: un paso en que se hace una suma local de n/p números, lo que lleva aproximadamente un tiempo de n/p.
- Segunda fase: $\log p$ pasos cada uno con una suma y una comunicación, lo que lleva un tiempo de $2 \log p$.

Ganancia y número de elementos de proceso

Ejemplo

Suma de n números con p elementos de proceso

$$\begin{aligned} T_P &= \frac{n}{p} + 2\log p. \\ G &= \frac{n}{\frac{n}{p} + 2\log p}. \\ E &= \frac{1}{1 + \frac{2p\log p}{n}}. \end{aligned}$$

Suma de n números con p elementos de proceso

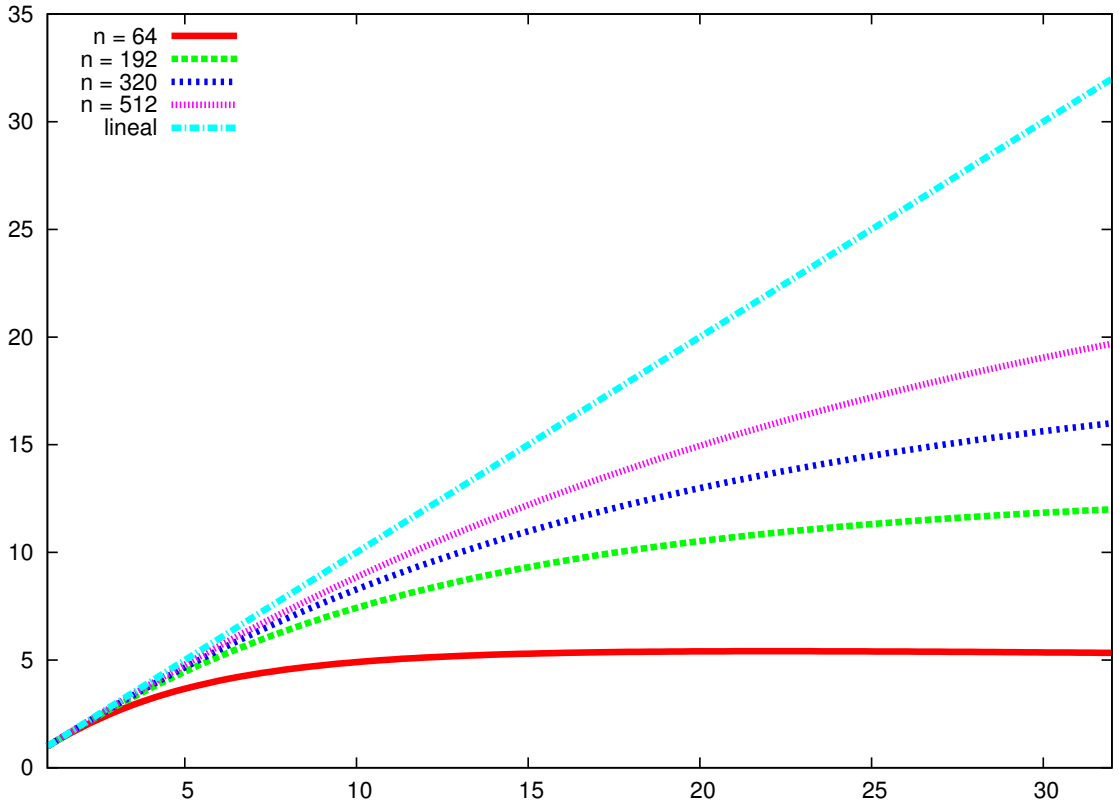

Figura: Ganancia en función del número de elementos de proceso p

Suma de n números con p elementos de proceso

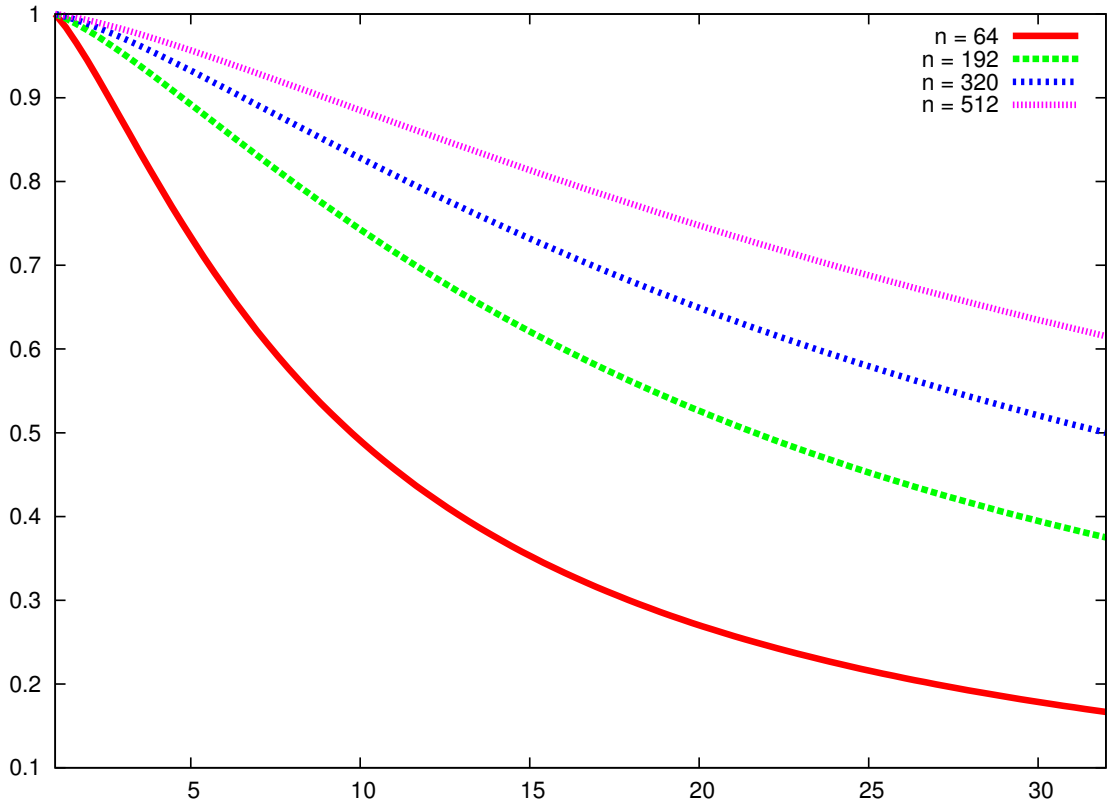

Figura: Eficiencia en función del número de elementos de proceso p

Ley de Amdahl

- Tamaños de problema crecientes proporcionan valores crecientes de ganancia y eficiencia, pero...
  - La ganancia se satura al aumentar el número de elementos de proceso p.
  - La eficiencia cae al aumentar el número de elementos de proceso p.

> ▶ Esta es la *ley de Amdahl*[1] de los rendimientos decrecientes.

[1] G.M. Amdahl, Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities, *AFIPS Conference Proceedings*, (30), pp. 483–485, 1967.

Concepto de escalabilidad

- Si observamos la ecuación (2), podemos mantener la eficiencia constante si somos capaces de aumentar T_S en igual proporción que T_O cuando aumenta el número de elementos de proceso p.
- Cuando tal cosa ocurre, decimos que el sistema paralelo es **escalable**.

Definición

La **escalabilidad** captura la capacidad de un sistema paralelo para aumentar (suficientemente) su ganancia al aumentar el número de elementos de proceso y el tamaño del problema.

Escalabilidad en el problema de la suma

- En el problema de la suma de n números usando p elementos de proceso, tenemos que

$$E = \frac{1}{1 + \frac{2p \log p}{n}}.$$

Es claro que para que la eficiencia no decaiga al aumentar p, el tamaño del problema debe aumentar en proporción $p \log p$.

Escalabilidad en el problema de la suma

Ejemplo

Si $n = 64$ y $p = 4$,

$$E = \frac{1}{1 + \frac{2 \cdot 4 \log 4}{64}} = 0{,}8.$$

Si doblamos el número de elementos de proceso a $p = 8$, entonces el valor de n ha de multiplicarse por

$$\frac{8 \log 8}{4 \log 4} = 3,$$

es decir, ha de ser $n = 3 \cdot 64 = 192$.

Isoeficiencia

En resumen:

- para un problema de tamaño dado, aumentar el número de elementos de proceso hace bajar la eficiencia del sistema;
- en muchos casos, aumentar el tamaño del problema manteniendo el número de elementos de proceso, hace subir la eficiencia del sistema.

Isoeficiencia

- ¿A qué ritmo se debe incrementar el *tamaño del problema* para que se mantenga la *eficiencia* al aumentar el número de elementos de proceso?

Tamaño del problema

Definición

Identificamos el **tamaño de un problema** o **carga computacional** (denotada W) con el número de pasos de computación necesarios para resolver el problema utilizando el algoritmo secuencial óptimo que exista.

Comentario

Si asumimos que cada paso de computación necesita una unidad de tiempo, entonces la carga W puede identificarse con el tiempo serie T_S.

Función de isoeficiencia

- Consideramos que la sobrecarga depende de W y de p.
- Si queremos que E permanezca constante, es obvio que W y T_O deben incrementarse al mismo ritmo, es decir $W/T_O = \Theta(1)$.

Función de isoeficiencia

En concreto, reescribimos la ecuación (1) como

$$T_P = \frac{W + T_O(W,p)}{p}, \tag{3}$$

de donde la eficiencia es ahora

$$E = \frac{1}{1 + \frac{T_O(W,p)}{W}}.$$

Despejando,

$$W = \frac{E}{1-E} T_O(W,p). \tag{4}$$

Universidad de Alcalá

Función de isoeficiencia

Definición

Denominamos **función de isoeficiencia** a la ecuación

$$W = KT_O(W, p).$$

Comentario

Observemos que si en la ecuación (4) mantenemos la eficiencia constante, entonces podemos identificar $K = E/(1 - E)$.
Idealmente, hay que escribirla en función solo de p.

Ejemplo de función de isoeficiencia

Ejemplo

Supongamos un sistema para el que $T_O = p^{3/2} + p^{3/4}W^{3/4}$. La isoeficiencia será $W = Kp^{3/2} + Kp^{3/4}W^{3/4}$. Usando el primer término:

$$W = Kp^{3/2}.$$

Usando el segundo:

$$W = Kp^{3/4}W^{3/4} \quad \Rightarrow \quad W = K^4p^3.$$

Así pues, los términos de la función de sobrecarga tienen que crecer como $\Theta(p^{3/2})$ y como $\Theta(p^3)$ para que la eficiencia no baje al aumentar p, de los que el más significativo es el segundo. Por tanto $W = \Theta(p^3)$.

Función de isoeficiencia y coste óptimo

- Un sistema es de coste óptimo si este es proporcional al tamaño del problema, es decir,

$$C = pT_P = \Theta(W).$$

- Recordando la ecuación (3), obtenemos la siguiente cadena de igualdades:

$$\begin{aligned} W + T_O(W,p) &= \Theta(W) \\ T_O(W,p) &= O(W) \\ W &= \Omega\left(T_O(W,p)\right) \end{aligned}$$

> ▶ Un sistema es de coste óptimo si y solo si su función de sobrecarga crece asintóticamente más despacio que el tamaño del problema.

Tiempo mínimo de ejecución

- Si suponemos una carga computacional fijada, podemos tratar de calcular cuál es el valor de p que nos proporciona el mínimo tiempo de ejecución paralelo.
- Sea $T_P = T_P(W, p)$ y supongamos que T_P es derivable con respecto a p. Entonces,

$$\frac{\mathrm{d}}{\mathrm{d}p} T_P = 0,$$

nos daría el valor de p que minimiza el tiempo de ejecución para un tamaño del problema fijado.

Tiempo mínimo de ejecución

Ejemplo

Supongamos que para el problema de sumar n números con p procesos se tiene que

$$T_P = \frac{n}{p} + 2 \log p.$$

Derivando e igualando a 0, tenemos que $p = n/2$ nos da el T_P mínimo, que vale

$$T_P^{\min} = 2 \log n.$$

Ganancia escalada

Definición

Definimos **ganancia escalada** la ganancia que se obtiene cuando escalamos el tamaño del problema linealmente con respecto al número de elementos de proceso, es decir, $W = \Theta(p)$.

Comentario

Esta métrica está relacionada con la isoeficiencia si esta es lineal o casi lineal.

Ganancia escalada

- Consideramos dos métodos para escalar el tamaño del problema:
  1. el tamaño del problema se escala para llenar la memoria del sistema con más elementos de proceso (restricción de memoria);
  2. el tamaño del problema se escala manteniendo una cota superior del tiempo de ejecución (restricción de tiempo de ejecución).

Ejemplo de multiplicación de matrices

Ejemplo

Para multiplicar dos matrices $n \times n$ en serie se necesitan en modo secuencial $t_c n^3$ operaciones, donde t_c representa el coste en tiempo de una operación multiplicación-suma. El tiempo paralelo (para el algoritmo «simple») usando $p = q^2$ procesos, es:

$$T_P = t_c \frac{n^3}{p} + t_0 \log p + 2\frac{n^2}{r_\infty q}.$$

Por tanto, la ganancia:

$$G = \frac{t_c n^3}{t_c \frac{n^3}{p} + t_0 \log p + 2\frac{n^2}{r_\infty q}}.$$

Ganancia escalada con restricción de memoria

Ejemplo

La memoria necesaria es $m = \Theta(n^2)$, pero también $m = \Theta(p)$. Por tanto, $n^2 = k \times p$ para cierta k. Sustituyendo en la ecuación de la ganancia, obtenemos la ganancia escalada con restricción de memoria:

$$G' = \frac{t_c(k \times p)^{3/2}}{t_c \frac{(k \times p)^{3/2}}{p} + t_0 \log p + 2\frac{k \times p}{r_\infty q}} = O(p).$$

Por tanto es aproximadamente lineal con respecto al número de elementos de proceso.

Ganancia escalada con restricción de tiempo de ejecución

Ejemplo

Se tiene que $T_P = O(n^3/p)$. Si se quiere que sea constante (que tenga un "tope"), entonces $n^3 = k \times p$ para cierta k. Sustituyendo igual que antes:

$$G'' = \frac{t_c(k \times p)}{t_c \frac{k \times p}{p} + t_0 \log p + 2\frac{(k \times p)^{2/3}}{r_\infty q}} = O(p^{5/6}).$$

En este caso no alcanzamos la linealidad, la ganancia escalada es solo sub-lineal.

Fracción serie-paralelo

- Otro método consiste en fijar una carga computacional y dividirla en un trozo totalmente serie y un trozo totalmente paralelizable, esto es, $W = W_{\text{ser}} + W_{\text{par}}$.
- Definimos la *fracción serie* como $f = W_{\text{ser}}/W$.
- Suponiendo tiempos de ejecución unitarios, podremos escribir

$$T_P = T_{\text{ser}} + \frac{T_{\text{par}}}{p} = f \times T_S + \frac{T_S - f \times T_S}{p}.$$

- Puesto que la ganancia G es $G = T_S/T_P$, despejando f y sustituyendo obtenemos:

$$f = \frac{1/G - 1/p}{1 - 1/p}.$$

Universidad de Alcalá

Revisión: 1.10

Fracción serie-paralelo

- Es fácil ver que cuanto más pequeño es f obtenemos mejores eficiencias.
- Si es f resulta creciente con p creciente, indica mala escalabilidad.

Fracción serie en el producto de matrices

Ejemplo

$$
\begin{aligned}
f &= \frac{\frac{t_c \frac{n^3}{p} + t_0 \log p + 2\frac{n^2}{r_\infty q}}{t_c n^3} - 1/p}{1 - 1/p} \\
&= \frac{p t_0 \log p + 2\frac{n^2 q}{r_\infty}}{t_c n^3} \times \frac{1}{p-1} \\
&\approx \frac{t_0 \log p + \frac{2n^2}{r_\infty q}}{t_c n^3}.
\end{aligned}
$$

Fracción serie en el producto de matrices

Es interesante notar que en la expresión

$$f \approx \frac{t_0 \log p + \frac{2n^2}{r_\infty q}}{t_c n^3}$$

- el numerador corresponde a la sobrecarga del paralelismo;
- el denominador es el tiempo secuencial.

Contenidos

Coste del intercambio de datos

- El tiempo de ejecución de los algoritmos paralelos viene marcado por el tiempo debido al esfuerzo computacional y el tiempo involucrado en las comunicaciones de los datos.
- Estamos interesados sobre todo en operaciones de comunicación con un esquema de tipo colectivo que van a aparecer más adelante en los algoritmos concretos que estudiaremos.

Coste de la operación básica

- La operación básica es transmitir un mensaje de tamaño m entre dos nodos de la red.
- Considerando conmutación vermiforme, el coste de la operación básica es:

$$\begin{aligned} T_{ij}(m) &= t_0 + n_{ij}\delta + \frac{m}{r_\infty} \\ &\approx t_0 + \frac{m}{r_\infty}. \end{aligned}$$

  donde $\delta \ll t_0$, de donde la aproximación.
- Naturalmente, el valor de t_0 y r_∞ es totalmente dependiente de las implementaciones reales y del tipo de red empleada.

Operaciones colectivas

Difusión uno-a-todos

Un proceso fuente tiene un dato M (el mensaje) que se ha de difundir. Al terminar, todos los procesos deben estar en posesión de una copia del mensaje M.
Implementada en `MPI_Bcast`.

Reducción todos-a-uno

Cada proceso P_i está en posesión de un dato M_i. El dato de cada proceso se combina a través de una operación asociativa (una suma, por ejemplo) y el resultado de la "reducción" se almacena en un solo proceso destino.
Implementada en `MPI_Reduce`.

Operaciones colectivas

Difusión personalizada uno-a-todos

Un proceso fuente (el proceso *raíz*) manda un dato distinto M_i a cada proceso. Al terminar, cada proceso tiene un dato distinto, "personalizado".
Implementada en `MPI_Scatter`.

Reunión personalizada todos-a-uno

Cada proceso P_i envía un dato distinto M_i a un proceso determinado (el proceso *raíz*). Al terminar el proceso raíz ha recibido un dato "personalizado" de cada uno de los otros nodos.
Implementada en `MPI_Gather`.

Operaciones colectivas

Difusión todos-a-todos

Cada proceso P_i es fuente de un dato M_i que se ha de difundir a todos los demás. Al terminar, todos los procesos deben estar en posesión de una copia de todos los M_i de los demás.
Implementada en `MPI_Allgather`.

Reducción todos-a-todos

Cada proceso P_i está en posesión de un dato M_i. El dato de cada proceso se combina a través de una operación asociativa (una suma, por ejemplo) y el resultado de la "reducción" se almacena en todos los nodos.
Implementada en `MPI_Allreduce`.

Red paradigma para cálculo de costes

Para la estimación de costes de comunicación, consideramos la red de la figura:

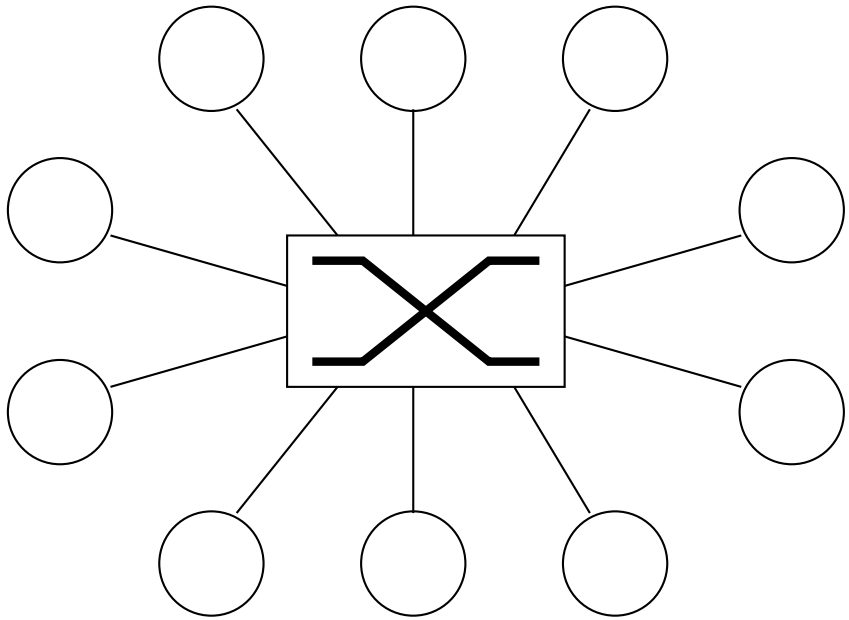

Figura: Red totalmente conectada a través de un conmutador

Difusión uno-a-todos

Supongamos, como antes, un mensaje de tamaño m que ha de difundirse a 8 nodos. En este caso, el nodo raíz es el 0 y damos estos pasos:

- Paso 1: $0 \rightarrow 4$
- Paso 2: $0 \rightarrow 2$, $4 \rightarrow 6$
- Paso 3: $0 \rightarrow 1$, $2 \rightarrow 3$, $4 \rightarrow 5$, $6 \rightarrow 7$.

Ahora el coste puede modelarse como

$$\left(t_0 + \frac{m}{r_\infty} \right) \log p.$$

Reducción todos-a-uno (I)

Tenemos un dato de tamaño m en todos los nodos y queremos reducirlo en uno de ellos (el nodo raíz) mediante una operación, por ejemplo, una suma. Si usamos 8 nodos y el raíz es el 0:

Paso 1: $\left\{ \begin{array}{cccc} 1 \to 0, & 3 \to 2, & 5 \to 4, & 7 \to 6. \\ (1) \to (0+1), & (3) \to (2+3), & (5) \to (4+5), & (7) \to (6+7). \end{array} \right.$

Paso 2: $\left\{ \begin{array}{cc} 2 \to 0, & 6 \to 4. \\ (2+3) \to (0+1+2+3), & (6+7) \to (4+5+6+7). \end{array} \right.$

Paso 3: $\left\{ \begin{array}{c} 4 \to 0. \\ (4+5+6+7) \to (0+1+2+3+4+5+6+7). \end{array} \right.$

Universidad de Alcalá

Reducción todos-a-uno (II)

- Observemos que, en cada paso, el tamaño del mensaje es siempre el mismo, es decir, es el tamaño del dato una vez reducido por la operación suma.
- Por tanto, el coste puede modelarse como $\log p$ comunicaciones de un mensaje de tamaño m:

$$\left(t_0 + \frac{m}{r_\infty}\right) \log p.$$

Comentario

Hemos ignorado los costes de la operación reductiva.

Difusión personalizada uno-a-todos (I)

El proceso raíz tiene p datos de tamaño m y ha de mandarle uno distinto ("personalizado") a cada nodo. En este caso, el raíz es el 0 y tenemos 8 nodos.

Paso 1: $\begin{cases} 0 \to 4. \\ (4,5,6,7) \to (4,5,6,7). \end{cases}$

Paso 2: $\begin{cases} 0 \to 2, & 4 \to 6. \\ (2,3) \to (2,3), & (6,7) \to (6,7). \end{cases}$

Paso 3: $\begin{cases} 0 \to 1, & 2 \to 3, & 4 \to 5, & 6 \to 7. \\ (1) \to (1), & (3) \to (3), & (5) \to (5), & (7) \to (7). \end{cases}$

Difusión personalizada uno-a-todos (II)

- En cada paso, el tamaño de la información intercambiada cambia, por lo que debemos sumar los costes en cada paso.
- Podemos escribir, entonces,

$$\begin{aligned}\sum_{\ell=0}^{\log p-1}\left(t_0+2^\ell\frac{m}{r_\infty}\right) &= t_0\log p+\frac{m}{r_\infty}\sum_{\ell=0}^{\log p-1}2^\ell\\ &= t_0\log p+\frac{m}{r_\infty}(p-1).\end{aligned}$$

Reunión personalizada todos-a-uno (I)

Cada proceso envía un datos distinto a un nodo prefijado (nodo raíz) que los reúne. Aquí el proceso raíz es el 0 y tenemos 8 nodos.

Paso 1: $\begin{cases} 1 \to 0, & 3 \to 2, & 5 \to 4, & 7 \to 6. \\ (1) \to (0,1), & (3) \to (2,3), & (5) \to (4,5), & (7) \to (6,7). \end{cases}$

Paso 2: $\begin{cases} 2 \to 0, & 6 \to 4. \\ (2,3) \to (0,1,2,3), & (6,7) \to (4,5,6,7). \end{cases}$

Paso 1: $\begin{cases} 4 \to 0. \\ (4,5,6,7) \to (0,1,2,3,4,5,6,7). \end{cases}$

Reunión personalizada todos-a-uno (II)

- Vemos que la operación es completamente simétrica a la difusión: el coste va a ser el mismo, pues también en cada paso cambia el tamaño de la información intercambiada.
- Podemos escribir, entonces,

$$\begin{aligned}\sum_{\ell=0}^{\log p-1}\left(t_0+2^\ell\frac{m}{r_\infty}\right) &= t_0\log p+\frac{m}{r_\infty}\sum_{\ell=0}^{\log p-1}2^\ell\\ &= t_0\log p+\frac{m}{r_\infty}(p-1).\end{aligned}$$

Universidad de Alcalá

Difusión todos-a-todos (I)

Cada proceso envía un dato a todos los demás: no hay nodo raíz.
El resultado es que todos los procesos tienen todos los datos.
Supongamos 8 nodos, como antes.

Paso 1: $\begin{cases} 1 \leftrightarrow 0, & 3 \leftrightarrow 2, & 5 \leftrightarrow 4, & 7 \leftrightarrow 6. \\ (0,1) \leftrightarrow (0,1), & (2,3) \leftrightarrow (2,3), & (4,5) \leftrightarrow (4,5), & (6,7) \leftrightarrow (6,7). \end{cases}$

Paso 2: $\begin{cases} 2 \leftrightarrow 0, & 3 \leftrightarrow 1, & 6 \leftrightarrow 4, & 7 \leftrightarrow 5. \\ (0\text{–}3) \leftrightarrow (0\text{–}3), & (0\text{–}3) \leftrightarrow (0\text{–}3), & (4\text{–}7) \leftrightarrow (4\text{–}7), & (4\text{–}7) \leftrightarrow (4\text{–}7). \end{cases}$

Paso 3: $\begin{cases} 4 \leftrightarrow 0, & 5 \leftrightarrow 1, & 6 \leftrightarrow 2, & 7 \leftrightarrow 3. \\ (0\text{–}7) \leftrightarrow (0\text{–}7), & (0\text{–}7) \leftrightarrow (0\text{–}7), & (0\text{–}7) \leftrightarrow (0\text{–}7), & (0\text{–}7) \leftrightarrow (0\text{–}7). \end{cases}$

Universidad de Alcalá

Difusión todos-a-todos (II)

- Vemos que la operación es completamente análoga en cuanto a costes a las anteriores.
- Podemos escribir, entonces,

$$\begin{aligned}\sum_{\ell=0}^{\log p-1}\left(t_0+2^\ell\frac{m}{r_\infty}\right) &= t_0\log p+\frac{m}{r_\infty}\sum_{\ell=0}^{\log p-1}2^\ell\\ &= t_0\log p+\frac{m}{r_\infty}(p-1).\end{aligned}$$

Comentario

Hemos considerado que la comunicación es *full duplex*.

Reducción todos-a-todos (I)

Se realiza una operación de reducción en donde cada nodo contribuye con su dato pero el resultado final acaba almacenado en todos los nodos: no existe un nodo raíz. Supongamos 8 nodos:

$$\text{Paso 1:} \quad \begin{cases} 1 \leftrightarrow 0, & 3 \leftrightarrow 2, & 5 \leftrightarrow 4, & 7 \leftrightarrow 6. \\ \sum_0^1 \leftrightarrow \sum_0^1, & \sum_2^3 \leftrightarrow \sum_2^3, & \sum_4^5 \leftrightarrow \sum_4^5, & \sum_6^7 \leftrightarrow \sum_6^7. \end{cases}$$

$$\text{Paso 2:} \quad \begin{cases} 2 \leftrightarrow 0, & 3 \leftrightarrow 1, & 6 \leftrightarrow 4, & 7 \leftrightarrow 5. \\ \sum_0^3 \leftrightarrow \sum_0^3, & \sum_0^3 \leftrightarrow \sum_0^3, & \sum_4^7 \leftrightarrow \sum_4^7, & \sum_4^7 \leftrightarrow \sum_4^7. \end{cases}$$

$$\text{Paso 3:} \quad \begin{cases} 4 \leftrightarrow 0, & 5 \leftrightarrow 1, & 6 \leftrightarrow 2, & 7 \leftrightarrow 3. \\ \sum_0^7 \leftrightarrow \sum_0^7, & \sum_0^7 \leftrightarrow \sum_0^7, & \sum_0^7 \leftrightarrow \sum_0^7, & \sum_0^7 \leftrightarrow \sum_0^7. \end{cases}$$

Universidad de Alcalá

Revisión: 1.10

Reducción todos-a-todos (II)

- Observemos que, al igual que en la reducción todos-a-uno, el tamaño del mensaje intercambiado en cada paso es siempre el mismo, es decir, es el tamaño del dato una vez reducido por la operación suma.
- Por tanto, el coste puede modelarse como $\log p$ comunicaciones de un mensaje de tamaño m:

$$\left(t_0 + \frac{m}{r_\infty}\right) \log p.$$

Comentario

Hemos ignorado los costes de la operación reductiva y hemos considerado que la comunicación es *full duplex*.

Difusión todos-con-todos en general

- Como resumen, obsérvese que en la red considerada, la difusión todos-con-todos presenta un tiempo de computación en que aparece, como mínimo, el término $(p-1)m/r_\infty$.