Tema II: Algoritmos Paralelos

- Método de resolución de un problema susceptible de ser simulado en un modelo de computación paralela.

- Un conjunto de procesos que pueden ser ejecutados concurrente y cooperativamente para resolver un problema.

Un problema se divide en subproblemas que pueden ser resueltos simultáneamente por procesadores independiente.

Diseño: conocer con detalle los modelos de computación paralela.

Corrección: tener presente unos parámetros que no aparecen en la computación secuencial (interacción de procesos, sincronización, comunicación de datos,...)

Análisis: precisar una serie de conceptos distintos a los usados en algoritmos secuenciales.

Durante la ejecución de un algoritmo paralelo:

- Interactúan diversos procesos por sincronización e intercambios de datos.

- Existen puntos en donde varios procesos se comunican.

Puntos de interacción: Puntos de la ejecución de un algoritmo paralelo donde se comunican varios procesadores (clasifica la ejecución de un proceso en varias $etapas$).

Principio: Admitiremos como aleatorio el tiempo que un proceso necesita para ejecutar una etapa.

Existencia de **tiempos de espera**: puede que dos procesadores que deben comunicarse no alcancen a la vez el punto de interacción.

Tiempo de ejecución de un algoritmo paralelo:

- Procesamiento básico (nº de operaciones)
- Bloqueado (tiempo de espera, tiempo em- empleado en sincronizaciones, ...)
- Otras tareas (iniciación variables globales, ...)

Tipos de sincronización

- **Explícita:** se detallan los puntos de interacción y existen tiempos de espera (*algoritmos sincronizados*).

- **Implícita:** No requieren tiempos de espera. La comunicación se realiza mediante la lectura dinámica de *variables globales* actualizadas, almacenadas en una memoria común (*algoritmos asincronizados*). Las variables globales se localizan en las llamadas *secciones críticas*.

Los algoritmos paralelos se clasifican:

- Según la estrategia usada para controlar la concurrencia: *centralizados* y *descentralizados* ó *distribuidos*.

- Según la *granularidad* (máxima cantidad de procesos que se pueden realizar antes de necesitar comunicarse con otros procesos).

- Según la topología de la red de comunicaciones.

Factor de aceleración, Coste y Eficiencia

Sea A un algoritmo *paralelo* diseñado para resolver un problema X en una máquina paralela con k procesadores.

$t(n)$ = tiempo de ejecución, en el caso peor, de un algoritmo óptimo (o del mejor algoritmo conocido) para resolver X.

$t(n, k)$ = tiempo de ejecución del algoritmo paralelo.

Recuento de pasos:

- Pasos *computacionales*: operación lógica o aritmética sobre unos datos.

- Pasos de *encaminamiento*: operación de comunicación entre procesadores.

Número de procesadores:

- Tienen un coste económico, así como su mantenimiento y sus interconexiones.

- En general, el número de procesadores, k, es función del tamaño del dato de entrada, n.

Factor de aceleración (*speed up*):

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

Coste: $C(n,k) = k \cdot t(n,k)$.

Eficiencia: $E(n,k) = \dfrac{t(n)}{C(n,k)}$

Algoritmo *óptimo*: $E(n,k) \in \theta(1)$.

Algoritmo *eficiente*: $E(n,k) \in \Omega(\dfrac{1}{\lg n})$.

El factor de aceleración puede ser:

- *Lineal*, si $S(n,k) = k$; es decir, si $E(n,k) = 1$ (ganancia en tiempo igual al número de procesadores).

- *Sub-lineal*, si $S(n,k) < k$; es decir, si $E(n,k) < 1$ (ganancia en tiempo inferior al número de procesadores).

- *Supra-lineal*, si $S(n,k) > k$; es decir, si $E(n,k) > 1$ (ganancia en tiempo superior al número de procesadores).

Todo algoritmo paralelo se puede simular en una máquina secuencial, ejecutando en serie los pasos paralelos (obviando problemas de comunicación).

Luego

$$k \cdot t(n,k) \geq t(n)$$

Es decir,

$$S(n,k) \leq k$$

Ley de Amdahl (1967)

Sea una máquina secuencial ideal en la que cada instrucción necesita una unidad de tiempo para su ejecución.

Sea $t(n)$ el tiempo de ejecución, en el caso peor, de un algoritmo secuencial óptimo.

Sean $t_s(n)$ (resp. $t_p(n)$) el tiempo de ejecución de la parte **no** (resp. **sí**) paralelizable.

Entonces $t(n) = t_s(n) + t_p(n)$.

Sea $f(n) = \dfrac{t_s(n)}{t(n)}$

Ley de Amdahl: el mejor tiempo de ejecución en paralelo, con k procesadores, se dará cuando las $t_p(n)$ instrucciones paralelizables son ejecutadas de manera óptima (es decir, en tiempo $\dfrac{t_p(n)}{k}$)

Una cota superior del factor de aceleración:

$$\text{lim}_{k \to \infty} S(n,k) = \frac{1}{f(n)}$$

La ley de Amdhal no permite aceleraciones supra-lineales.

Primeros ejemplos

Ejemplo 1. Determinar si un número entero, x, pertenece a una lista **ordenada** $A[1,\ldots,n]$.

- Sabemos que $t(n) \in \theta(\lg n)$.

- En **SIMD - EREW** se tiene:

  ⋆ $t(n,k) = \lg k + \lg\frac{n}{k}$.
  ⋆ $S(n,k) = \frac{\lg n}{\lg k + \lg\frac{n}{k}}$.
  ⋆ $C(n,k) = k \cdot (\lg k + \lg\frac{n}{k})$.
  ⋆ $E(n,k) = \frac{\lg n}{k\cdot(\lg k + \lg\frac{n}{k})}$.

- En **SIMD - CREW** se tiene:

  ⋆ $t(n,k) = \lg\frac{n}{k}$.
  ⋆ $S(n,k) = \frac{\lg n}{\lg\frac{n}{k}}$.
  ⋆ $C(n,k) = k \cdot \lg\frac{n}{k}$.
  ⋆ $E(n,k) = \frac{\lg n}{k\cdot\lg\frac{n}{k}}$.

Ejemplo 2. Determinar si un número entero, x, pertenece a una lista numérica **arbitraria** $A[1,\ldots,n]$.

- Sabemos que $t(n) \in \theta(n)$.

- En **SIMD - EREW** se tiene:

  ⋆ $t(n,k) = \lg k + \frac{n}{k}$.
  ⋆ $S(n,k) = \frac{n}{\lg k + \frac{n}{k}}$.
  ⋆ $C(n,k) = k \cdot (\lg k + \frac{n}{k})$.
  ⋆ $E(n,k) = \frac{n}{k \cdot (\lg k + \frac{n}{k})}$.

- En **SIMD - CREW** se tiene:

  ⋆ $t(n,k) = \frac{n}{k}$.
  ⋆ $S(n,k) = \frac{n}{n/k} = k$.
  ⋆ $C(n,k) = k \cdot \frac{n}{k} = n$.
  ⋆ $E(n,k) = \frac{n}{n} = 1$.

Ejemplo 3. Sea el algoritmo secuencial:

para $i \leftarrow 1$ **hasta** 100 **hacer**

$A[i] \leftarrow 1$

Número de operaciones realizadas en cada paso del bucle:

- Asigno a i el valor 1.
- Comparo i con 100.
- Asigno a $A[i]$ el valor 1.

En total 300 operaciones, más 2 para salir.

Realicemos dicho **algoritmo en paralelo** con 100 procesadores.

Número de operaciones realizadas:

- Asigno un índice i a cada procesador.
- Asigno a $A[i]$ el valor 1.

Número de operaciones realizadas: 2.

Por tanto,

- $S(n, 100) = 302/2 \simeq 150.$
- $C(n, 100) = 200.$
- $E(n, 100) = 302/200 \simeq 1.5.$

La aceleración y la eficiencia de un algoritmo paralelo son funciones del tamaño del dato de entrada y del número de procesadores.

- ¿Cuál será el número ideal de procesadores para un tamaño prefijado?

- ¿Cuál será el menor tamaño que debe tener un problema, prefijado el número de procesadores?

- Determinar la relación entre el número, k, de procesadores y el tamaño, n, de un problema para conseguir una eficiencia prefijada.

Ejemplo 4. Supongamos que un algoritmo paralelo tiene como factor de aceleración $S(n,k) = \sqrt{n}$ y como eficiencia $E(n,k) = \sqrt{n}/k$.

Fijado $E(n,k) = c$ $(0 \leq c \leq 1)$, para conservar la eficiencia debe verificarse que $n = c^2 \cdot k^2$.

Este algoritmo no debería ser utilizado en máquinas *masivamente paralelas*.

Ejemplo 5. Consideremos el algoritmo secuencial obvio para multiplicar dos matrices (lectura de datos en disco, operaciones y escritura de resultados en disco).

Número de operaciones: $n^2 + n^3$.

Un algoritmo paralelo **SIMD-EREW**, con k procesadores, necesita un tiempo de ejecución $n^2 + \frac{n^3}{k}$.

Luego $E(n,k) = \dfrac{n^2 + n^3}{k \cdot (n^2 + \frac{n^3}{k})}$

Fijado $E(n,k) = c$ $(0 \leq c \leq 1)$, para conservar la eficiencia se debe verificar que $n = \dfrac{c \cdot k - 1}{1 - c}$.

En máquinas masivamente paralelas, un crecimiento lineal del número de procesadores implicaría un crecimiento lineal del tamaño del dato de entrada.

Instrucciones

Pseudocódigo secuencial:

- $variable \leftarrow expresion$.
- **si** $condicion$ **entonces** $instruccion$.
- **si** $condicion$ **entonces** $instruccion_1$ **si no** $instruccion_2$.
- **mientras** $condicion$ **hacer** $instrucciones$.
- **para** $variable \leftarrow valor-inicial$ **hasta** $variable \leftarrow valor - final$ **hacer** $instrucciones$.
- $etiqueta:\ instruccion$
- **ir a** $etiqueta$.
- **devolver** $expresion$.

Pseudocódigo paralelo.

- Los algoritmos paralelos trabajan con operaciones secuenciales y/o paralelas.

- Las operaciones secuenciales se describen en pseudocódigo secuencial.

- Los operaciones paralelas pueden ser:

  ⋆ Aquellas que expresan la tarea(s) que debe(n) ejecutar cada procesador.

  para cada P_i tal que $S(i)$ **hacer** Q.

  En donde Q es la tarea a realizar por los P_i que satisfagan la condición $S(i)$.

  ⋆ Aquellas que permiten la comunicación entre procesadores

  bifurcación
  proceso$_1$
  . . .
  proceso$_n$
  unión

  lock
  ⋮
  unlock

  También se puede notar el comienzo de una sección crítica:

  sección crítica
  ⋮

Formato de la comunicación entre procesadores: se basa en que cada procesador posee una memoria privada a donde llegan los mensajes de procesadores vecinos.

$$t_i \Longleftarrow x_j$$

El procesador P_i comunica a P_j el valor x_j que queda almacenado en la variable local t_j de P_j.

Problema 1. Dadas las listas numéricas $A[1,\ldots,n]$ y $B[1,\ldots,n]$, hallar $C[1,\ldots,n]$ tal que $C[i] = A[i] + B[i]$, $(1 \leq i \leq n)$.

- Una cota inferior del problema, en el caso peor, es $\theta(n)$.

- Diseñamos un algoritmo paralelo en una **PRAM** con k procesadores $(k \geq n)$.

procedimiento $suma_{PRAM}$ (A, B, C)

para cada P_i tal que $1 \leq i \leq n$ **hacer**

$C[i] \leftarrow A[i] + B[i]$

Se verifica:

- $t(n) = n$.
- $t(n, k) = 1$.
- $S(n, k) = \dfrac{t(n)}{t(n, k)} = n$.
- $C(n, k) = k \cdot t(n, k) = k$.
- $E(n, k) = \dfrac{t(n)}{C(n, k)} = \dfrac{n}{k}$.

Por tanto:

- Tiempo ejecución paralelo $\leq$ Tiempo ejecución óptimo secuencial.
- Si $k = n$, los costes son similares.
- Si $k = n$, este algoritmo paralelo es óptimo.

Problema 2. Hallar el elemento máximo de la lista numérica $A[1,\ldots,n]$.

- Una cota inferior del problema, en el caso peor, es $\theta(n)$.
- Un algoritmo óptimo secuencial:

procedimiento $maximo_{sec.}$ $(A[1,\ldots,n])$

$max \leftarrow A[1]$

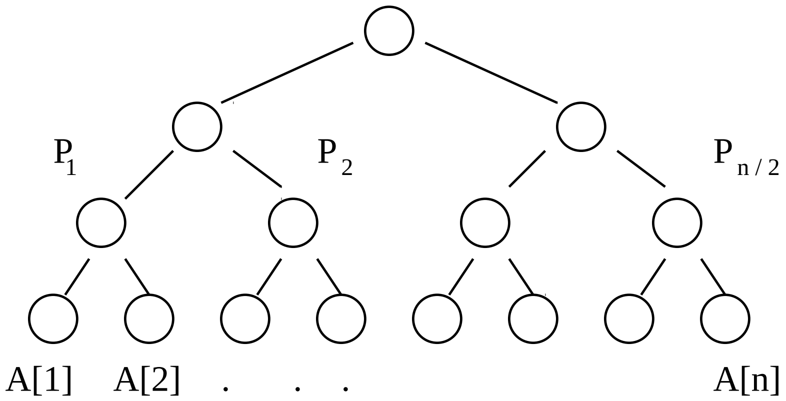
para $i \leftarrow 2$ **hasta** n **hacer**

si $A[i] > max$ **entonces**

$max \leftarrow A[i]$

devolver max

Paralelizamos la resolución del problema.



Implementamos la idea en una **PRAM–EREW** con $k = n - 1$ procesadores (n potencia de 2).

Dato de entrada: en la memoria compartida, en una lista $L[1, \ldots n-1, n, \ldots 2n-1]$ tal que $A[i] = L[n+i-1]$ $(1 \leq i \leq n)$.

procedimiento $maximo_{PRAM}$ $(A[1, \ldots, n])$

  para $i \leftarrow \lg n$ **hasta** 1 **hacer**

    para cada P_j tal que $2^{i-1} \leq j \leq 2^i - 1$ **hacer**

$$L[j] \leftarrow \max\{L[2j], L[2j+1]\}$$

  devolver $L[1]$

Se verifica:

- $t(n) = n$.
- $t(n,k) = \lg n$.
- $S(n,k) = \dfrac{n}{\lg n}$.
- $C(n,k) = k \cdot \lg n$.
- $E(n,k) = \dfrac{n}{k \cdot \lg n} \in \theta(\dfrac{1}{\lg n})$.

El algoritmo es eficiente pero **no** es óptimo.

Vamos a establecer la corrección del algoritmo.

Para cada j $(1 \leq j \leq n-1)$, sea

$$D_j = \{L[s] : \ L[s] \text{ descendiente de } L[j]\}.$$

Para cada i tal que $1 \leq i \leq \lg n$ sea

$$\theta(i) \equiv \forall j \ (2^{i-1} \leq j \leq 2^i - 1 \Longrightarrow \text{máx } D_j = L[j])$$

Teorema 1: *La fórmula θ es un invariante del bucle.*

Demostración: Por inducción acotada descendente sobre i.

■

Teorema 2: *Tras ejecutar el algoritmo, para cada i tal que $1 \leq i \leq \lg n$, si $L[j]$ es un nodo de profundidad i, entonces $L[j] \leq \text{máx } A[1, \ldots, n]$.*

Demostración: Por inducción acotada descendente sobre i.

■

Corolario: *El algoritmo antes descrito es correcto.*

Paradigma del árbol binario completo

Problema genérico: dados n elementos y una operación binaria asociativa entre ellos, calcular la composición de todos por la operación.

Mediante el método del árbol binario completo se resuelve el problema con $\frac{n}{2}$ procesadores necesitando un tiempo de ejecución $\theta(\lg n)$.

Este paradigma puede ser utilizado para realizar computaciones sobre n items, $A[1,\ldots,n]$, si ésta se puede describir como combinación de las computaciones sobre $A[1,\ldots,\frac{n}{2}]$ y $A[\frac{n}{2}+1,\ldots n]$.

Paralelismo acotado y no acotado

Tiempo de ejecución:

- Pasos computacionales.
- Pasos de encaminamiento..

Algoritmos paralelos que tiene:

- *Paralelismo acotado*.
- *Paralelismo no acotado*.

Algoritmo paralelo que tiene un paralelismo *polinomialmente acotado*: existe un polinomio $p(x)$ tal que $k(n) \leq p(n)$, para cada n (tamaño dato entrada).

Teorema de Brent (1974). *Sea A un algoritmo paralelo con tiempo de ejecución t. Si A realiza m operaciones computacionales, p procesadores pueden ejecutar A en tiempo $O(t + \frac{m-t}{p})$.*

Cotas inferiores en Paralelismo

Problema con n entradas y una salida, que se obtiene aplicando operaciones binarias asociativas entre las entradas.

Secuencialmente se necesitan, al menos, $n-1$ operaciones (si intervienen todas las entradas).

Modelizamos el problema mediante los árboles binarios etiquetados de tipo B_k.

- Definición recursiva de los árboles binarios etiquetados de tipo B_k.

Todo árbol etiquetado de tipo B_k modeliza un proceso de computación:

- Las hojas representan las entradas.
- Los nodos internos representan las operaciones binarias.
- La profundidad proporciona el número de pasos computacionales.
- La etiqueta de un nodo indica el nivel de altura.

- En cada nivel, $k =$número máximo de nodos.

Notemos

- $N(k,h) =$ máximo nº de hojas de un árbol de tipo B_k de profundidad h.

- $r(k,n) =$ menor profundidad de un árbol de tipo B_k con n hojas..

Se verifica:

- $N(k,0) = 1$.

- $N(k,h+1) = N(k,h) + \min\{k, N(k,h)\}$.

- $N(k,h-1) < n \leq N(k,h) \Longrightarrow r(k,n) = h$.

- $N(k,h) = 2^{\min\{\lceil \lg k \rceil, h\}} + k \cdot \max\{0, h - \lceil \lg k \rceil\}$.

- $r(k,n) = \min\{\lceil \lg k \rceil, \lceil \lg n \rceil\} +$
$$\max\{0, \lceil (n - 2^{\lceil \lg k \rceil})/k \rceil\}$$

Cota inferior del problema con n entradas y k procesadores: $r(k,n)$

La clase NC

La **C**lase de **N**ick (Nicholas Pippenger, 1979): "tratabilidad" paralela.

¿Qué problemas pueden ser resueltos de manera más *eficiente* usando algoritmos paralelos, en lugar de cualquier algoritmo secuencial?

- *Un problema X pertenece a* **NC** *si existe un algoritmo* **PRAM** *que resuelve X, con $O(p(n))$ procesadores en tiempo $O(q(\lg n))$ ($p(x)$ y $q(x)$ polinomios, y n el tamaño del dato de entrada)*.

Adecuación de esta definición:

- Diferencia entre n^3 y 2^n perceptible para $n = 20$. En cambio, entre $\lg^3 n$ y $\sqrt{n}$ la diferencia se nota a partir de $n = 10^{12}$.

- Si $p(x)$ o $q(x)$ tienen grado elevado, el algoritmo no será implementable en la práctica.

- Es una clase que, "generalmente", no depende del modelo de Computación Paralela.

Consideraciones:

- Existen problemas de la clase **NC**.
- $\mathbf{NC} \subseteq \mathbf{P}$.
- No se sabe si $\mathbf{P} = \mathbf{NC}$.
- Se intuye que todas las inclusiones siguientes son estrictas: $\mathbf{NC} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$.

Teorema 1. Sean X e Y problemas tales que $X \leq^p Y$ e $Y \in \mathbf{NC}$. Entonces, $X \in \mathbf{NC}$.

Al igual que la conjetura $\mathbf{P} = \mathbf{NP}$ se estudia a partir de los problemas **NP**–completos, la conjetura $\mathbf{NC} = \mathbf{P}$ se estudia a través de los problemas **P**–completos.

Teorema 2. Si X es **P**–completo y $X \in \mathbf{NC}$, entonces $\mathbf{P} = \mathbf{NC}$.

Anexo

La inducción proporciona un potente instrumento de demostración de teoremas en Matemáticas.

En Algorítmica, la inducción:

- Es una herramienta básica para analizar el comportamiento asintótico de funciones.

- Es útil para probar propiedades combinatorias de ciertas estructuras de datos.

- Proporciona, en algunos casos, el diseño de algoritmos que resuelven un determinado problema.

- Permite probar la corrección parcial de ciertos algoritmos, siguiendo un esquema general.

- Se utiliza implícitamente en el test de parada de algoritmos cuya prueba necesita el principio de buena fundamentación de los naturales.

Teorema (de inducción débil o completa). *Sea $\varphi(n)$ una "propiedad"(fórmula del lenguaje de la Aritmética). Supongamos que se verifica:*

- $\varphi(0)$.
- $\forall n \in \mathbf{N}\ (\varphi(n) \to \varphi(n+1)$.

Entonces se tiene que $\forall n \in \mathbf{N}\ ((\varphi(n))$.

Teorema (de inducción fuerte). *Sea $\varphi(n)$ una "propiedad". Supongamos que se verifica:*

- $\varphi(0)$.
- $\forall n \in \mathbf{N}\ [\forall k \in \mathbf{N}\ (k < n \to \varphi(k)) \to \varphi(n)]$

Entonces se tiene que $\forall n \in \mathbf{N}\ ((\varphi(n))$.

Versiones "acotadas".

- (Ascendente). Sea $n \in \mathbf{N}$ tal que
  $\varphi(0) \wedge \forall i\ (0 \leq i < n \wedge \varphi(i) \to \varphi(i+1))$

  Entonces $\forall i\ (0 \leq i \leq n)\ (\varphi(n))$.

- (Descendente). Sea $n \in \mathbf{N}$ tal que
  $\varphi(n) \wedge \forall i\ (0 < i \leq n \wedge \varphi(i) \to \varphi(i-1))$

  Entonces $\forall i\ (0 \leq i \leq n)\ (\varphi(n))$.

Análogas versiones para la inducción fuerte.

Meta-inducción sobre la estructura de un algoritmo

Sea A un algoritmo genérico cuya estructura es: $I; B; F$ en donde I y F son conjuntos de instrucciones de inicialización y finalización y B es un bucle WHILE o FOR (denominado principal).

Para la corrección parcial de A es suficiente hallar una fórmula, φ, en las variables del algoritmo, que satisfaga las condiciones siguientes:

1. φ es verdadera inmediatamente antes del comienzo de la ejecución del bucle principal (esta fase juega el papel del *caso base*).

2. φ es un invariante del bucle principal (esta fase juega el papel del *paso inductivo*).

3. La veracidad de φ al finalizar la ejecución debe expresar que el algoritmo resuelve el problema para el que ha sido diseñado.

El procedimiento de inducción se utiliza en los pasos (1) y (2) para justificar que φ es verdadera tras la ejecución del bucle principal del algoritmo.

La prueba de que φ es un invariante del bucle principal se hace por inducción "ordinaria".

Este esquema de demostración de la corrección parcial se puede considerar como un método de prueba por meta-inducción.