CPA – Computació Paral·lela

Grau en Enginyeria Informàtica

T2. Memòria Compartida. Disseny Bàsic d'Algoritmes Paral·lels

J. M. Alonso, P. Alonso, F. Alvarruiz, I. Blanquer,
J. Ibáñez, E. Ramos, J. E. Román

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Curs 2024/25

etsinf

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Contingut

1 Model de Memòria Compartida
- Model
- Detalls

2 Fonaments del Disseny d'Algoritmes Paral·lels
- Anàlisi de Dependències
- Graf de Dependències

3 Avaluació de Prestacions (I)
- Paràmetres Absoluts
- Prestacions en Memòria Compartida

4 Disseny d'Algoritmes: Descomposició en Tasques
- Descomposició de Domini
- Altres Descomposicions

Apartat 1

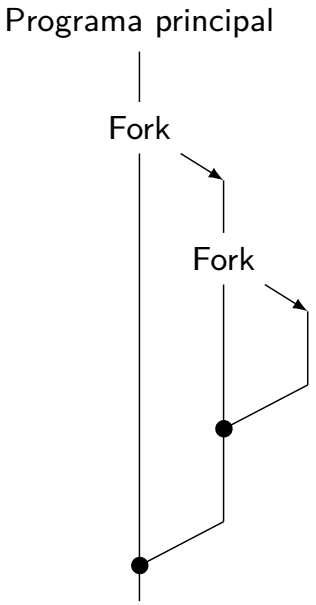
Model de Memòria Compartida

- Model
- Detalls

Processos Concurrents

Per a especificar processos concurrents és habitual utilitzar construccions de tipus *fork-join*

- *Fork* crea una nova tasca concurrent que comença a executar en el mateix punt en què estava la tasca pare
- *Join* espera a que acabe la tasca
- Exemple: crida al sistema `fork()` en Unix



Aquest esquema es pot implementar a nivell de:

- Processos del sistema operatiu (*processos pesats*)
- Fils (*processos lleugers*)

Model de Memòria Compartida

Característiques:

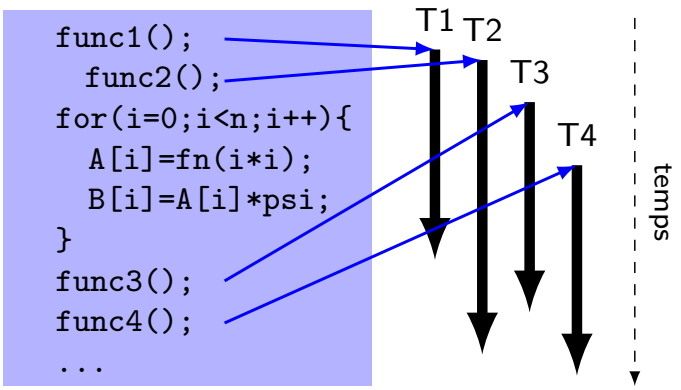
- Espai d'adreces de memòria únic per a tots
- Programació bastant similar al cas seqüencial
  - Qualsevol dada és accessible per qualsevol
  - No cal intercanviar dades explícitament
- Inconvenients
- L'accés concurrent a memòria pot donar problemes
  - S'ha de coordinar: semàfors, monitors, ...
  - Resultat impredictible si no es protegeixen bé els accessos a memòria
- Difícil controlar la localitat de dades (memòries cache)

Model de Fils

Aquest model està molt lligat al de memòria compartida

Fil (*thread*): flux d'instruccions independent que pot ser planificat per a execució pel sistema operatiu

- Un procés pot tenir múltiples fils d'execució
- Cada fil té dades "privades"
- Comparteixen recursos/memòria del procés
- Es requereix sincronització



OpenMP

Estandardització de fils portable

- Basat en directives de compilador
- Disponible en C/C++ i Fortran
- Portable/multi-plataforma (Unix, Windows)
- Fàcil d'usar: paral·lelització incremental

Algunes directives i funcions

- `#pragma omp parallel for`
- `omp_get_thread_num()`

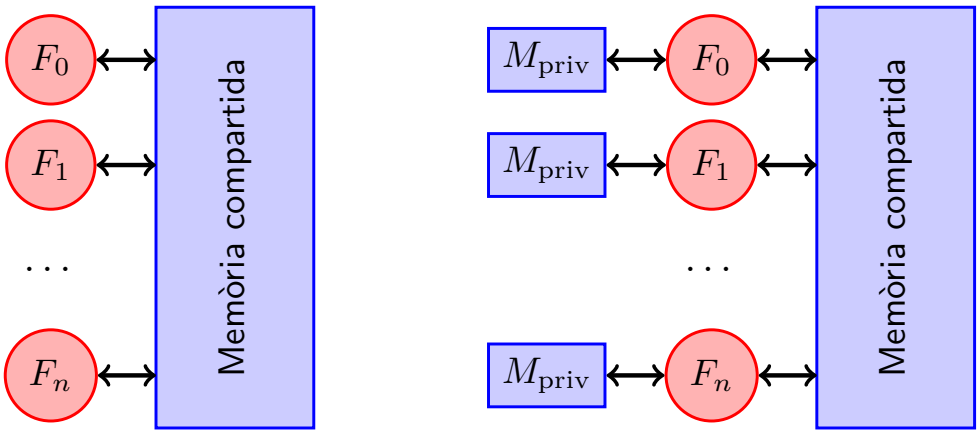
La creació i finalització de fils està implícita en algunes directives

- El programador no s'ha de preocupar de fer *fork/join*

Model de Memòria amb Fils

Model simple: espai únic d'adreces

Model més realista: espai únic d'adreces, amb variables privades per a cada fil



Una pila de crides per a cada fil

- Algunes variables es creen en la pila (locals)
- Un fil no pot saber si la pila d'un altre fil està activa

Coordinació d'Accessos a Memòria

L'intercanvi d'informació entre fils es fa mitjançant lectura/escriptura de variables en memòria compartida

L'accés simultani pot produir una condició de carrera

- El resultat final pot ser incorrecte
- És de naturalesa no determinista

Exemple: dos fils volen incrementar la variable `i`

Seqüència amb resultat correcte:

H0 carrega `i` en un registre: `0`
H0 incrementa registre: `1`
H0 guarda el valor en `i`: `1`
H1 carrega `i` en un registre: `1`
H1 incrementa registre: `2`
H1 guarda el valor en `i`: `2`

Seqüència amb resultat incorrecte:

H0 carrega `i` en un registre: `0`
H1 carrega `i` en un registre: `0`
H0 incrementa registre: `1`
H1 incrementa registre: `1`
H0 guarda el valor en `i`: `1`
H1 guarda el valor en `i`: `1`

Exclusió Mútua i Sincronització

Com solucionar la condició de carrera?

Operacions atòmiques

- Forçar a que les operacions problemàtiques es realitzen de forma atòmica (sense interrupció)
- Instruccions especials del processador: *test-and-set* o *compare-and-exchange* (CMPXCHG en Intel)

Secció crítica

- Fragments de codi amb més d'una instrucció
- No permetre que hi haja més d'un fil executant-la
- Requereix mecanismes de sincronització: semàfors, etc.
- Pot aparèixer risc d'interbloqueig

Altre tipus de sincronització

- Barrera: esperen en un punt a que arriben tots
- Execució ordenada

Apartat 2

Fonaments del Disseny d'Algoritmes Paral·lels

- Anàlisi de Dependències
- Graf de Dependències

Paral·lelització d'Algoritmes

Paral·lelitzar un algoritme implica trobar tasques (parts de l'algoritme) concurrents (es poden executar en paral·lel)

Quasi sempre, hi ha dependències entre tasques

- Si una tasca només pot començar quan un altra ha acabat

```
a = 0
PER A i=0 FINS n-1
  a = a + x[i]
FPER
b = 0
PER A i=0 FINS n-1
  b = b + y[i]
FPER
PER A i=0 FINS n-1
  z[i] = x[i]/b + y[i]/a
FPER
PER A i=0 FINS n-1
  y[i] = (a+b)*y[i]
FPER
```

Exemple:

- Els dos primers bucles són independents entre si
- El tercer bucle utilitza els valors d'a i b, que es calculen en els dos bucles anteriors

Dependències de Dades

Es pot determinar si existeixen dependències entre dues tasques a partir de les dades d'entrada/eixida de cada tasca

Condicions de Bernstein:
Dues tasques T_i i T_j (T_i precedeix a T_j en seqüencial) són independents si

1. $I_j \cap O_i = \varnothing$
2. $I_i \cap O_j = \varnothing$
3. $O_i \cap O_j = \varnothing$

I_i i O_i representen el conjunt de variables llegides i escrites per T_i

Tipus de dependències:

- Dependència de flux (es viola la condició 1)
- Anti-dependència (es viola la condició 2)
- Dependència d'eixida (es viola la condició 3)

Dependències de Dades: Exemples

Dependència de flux

```
double a=3,b=5,c,d;
c = T1(a,b);
d = T2(a,b,c);
```

T_2 no pot començar fins que acabe T_1, perque llig la variable c, que és escrita per T_1

Anti-dependència

```
// T1,T2 modifiquen 3er argument
double a[10],b[10],c[10],y;
T1(a,b,&y);
T2(b,c,a);
```

T_2 no pot començar fins que acabe T_1, en cas contrari T_2 sobrescriuria el contingut de a que és entrada de T_1

Dependència d'eixida

```
// T1,T2 modifiquen 3er argument
double a[10],b[10],c[10],x[5];
T1(a,b,x);
T2(c,b,x);
```

Ambdues tasques modifiquen l'array x

Dependències de Dades en Bucles

Algunes es poden eliminar modificant l'algorisme

Codi amb dependència de flux

```
for (i=1; i<n; i++) {
  b[i] = b[i] + a[i-1];
  a[i] = a[i] + c[i];
}
```

La iteració `i` modifica `a[i]` que és llegida per la iteració `i+1`

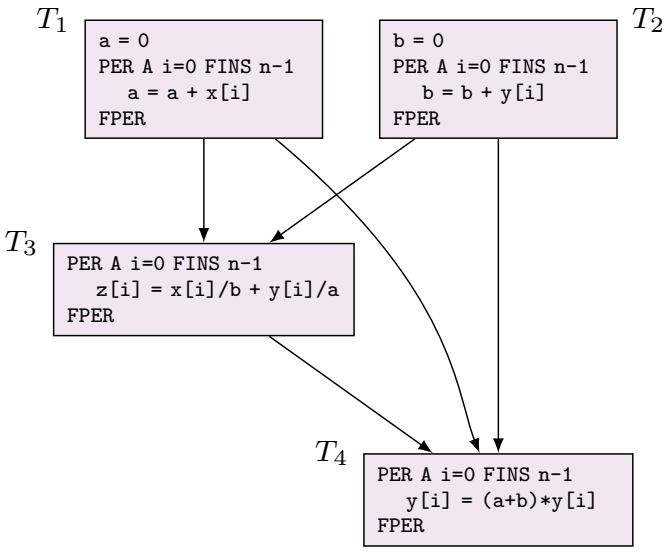
Eliminació de la dependència mitjançant *esbiaixat del bucle*:

Codi sense dependències

```
b[1] = b[1] + a[0];
for (i=1; i<n-1; i++) {
  a[i] = a[i] + c[i];
  b[i+1] = b[i+1] + a[i];
}
a[n-1] = a[n-1] + c[n-1];
```


Paral·lelització d'Algoritmes: Exemple

```
a = 0
PER A i=0 FINS n-1
  a = a + x[i]
FPER
b = 0
PER A i=0 FINS n-1
  b = b + y[i]
FPER
PER A i=0 FINS n-1
  z[i] = x[i]/b + y[i]/a
FPER
PER A i=0 FINS n-1
  y[i] = (a+b)*y[i]
FPER
```



Dependències de flux: $T_1 \to T_3$, $T_2 \to T_3$, $T_1 \to T_4$, $T_2 \to T_4$
Anti-dependències: $T_2 \to T_4$, $T_3 \to T_4$

Disseny d'Algoritmes Paral·lels: Idea General

Bàsicament dues fases:

1. Descomposició en tasques
   - Requereix una anàlisi detallada del problema
     $\longrightarrow$ Graf de Dependències de Tasques

2. Assignació de tasques
   - Quin fil/procés executa cada tasca
   - A voltes implica agrupar vàries tasques

Habitualment hi ha vàries estratègies possibles de paral·lelització
- Usar una descomposició o altra pot tindre gran impacte en les prestacions
- Hi ha que intentar maximitzar el grau de concurrència

Graf de Dependències de Tasques

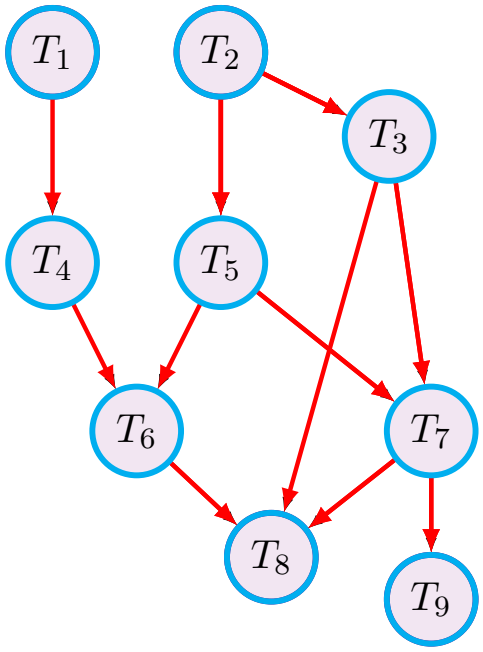
Abstracció utilitzada per a expressar les dependències entre les tasques i el seu relatiu ordre d'execució
- Es tracta d'un graf acíclic dirigit (GAD)
- Els nodes representen tasques (poden tindre associat un cost)
- Les arestes representen les dependències entre tasques

Definicions:
- Longitud d'un camí: suma de los costos c_i dels nodes que el componen
- Camí crític: el més llarg entre un node inicial i un final
- Màxim grau de concurrència: major nombre de tasques que poden executar-se al mateix temps
- Grau mitjà de concurrència: $M = \sum_{i=1}^{N} \frac{c_i}{L}$

  (N = nodes totals, L = longitud del camí crític)

Grafs de Depències de Tasques: Exemple

Graf amb $N = 9$ tasques (suposem que totes tenen cost $c_i = 1$)

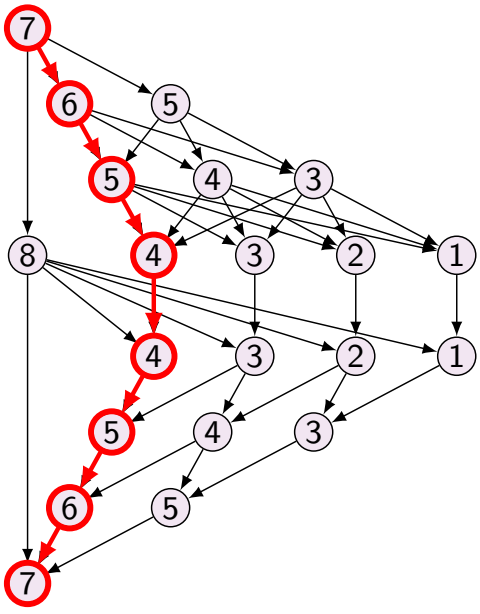


Nodes inicials: T_1, T_2
Nodes finals: T_8, T_9
Camins:
- $T_1 - T_4 - T_6 - T_8$ (longitud 4)
- $T_2 - T_5 - T_6 - T_8$ (longitud 4)
- $T_2 - T_5 - T_7 - T_8$ (longitud 4)
- $T_2 - T_3 - T_8$ (longitud 3)
- $T_2 - T_3 - T_7 - T_8$ (longitud 4)
- $T_2 - T_5 - T_7 - T_9$ (longitud 4)
- $T_2 - T_3 - T_7 - T_9$ (longitud 4)

Camí crític: $L = 4$
Concurrència:
- Grau màxim: 3
- Grau mitjà: $M = \sum_{i=1}^{9} \frac{1}{4} = 2.25$

Grafs de Dependències de Tasques: Exemple

Graf amb $N = 21$ tasques (s'indica el cost c_i en cada tasca)



Camí crític
$L = 7 + 6 + 5 + 4 + 4 + 5 + 6 + 7 = 44$

$$M = \sum_{i=1}^{N} \frac{c_i}{L} = \frac{7 + 6 + 5 + 5 + \cdots}{44} = 2$$

Exemple de Descomposició en Tasques

Donats m polinomis

$$P_i(x) = a_{i,0} + a_{i,1}x + a_{i,2}x^2 + \cdots + a_{i,n}x^n, \;\; i = 0 : m-1$$

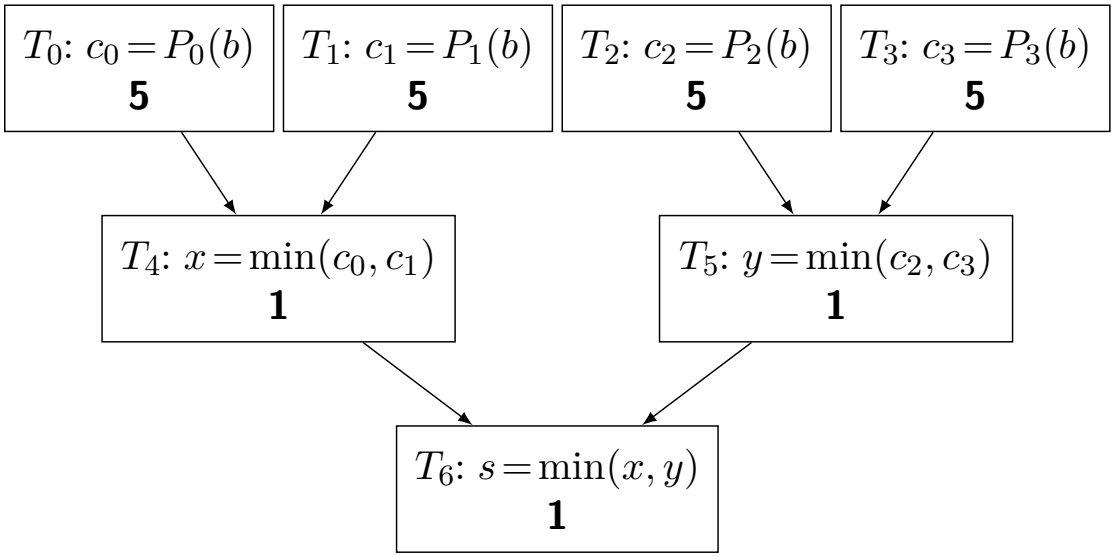
i un valor b, calcular

$$s = \min_{i=0:m-1} \{P_i(b)\},$$

Possible descomposició en tasques:

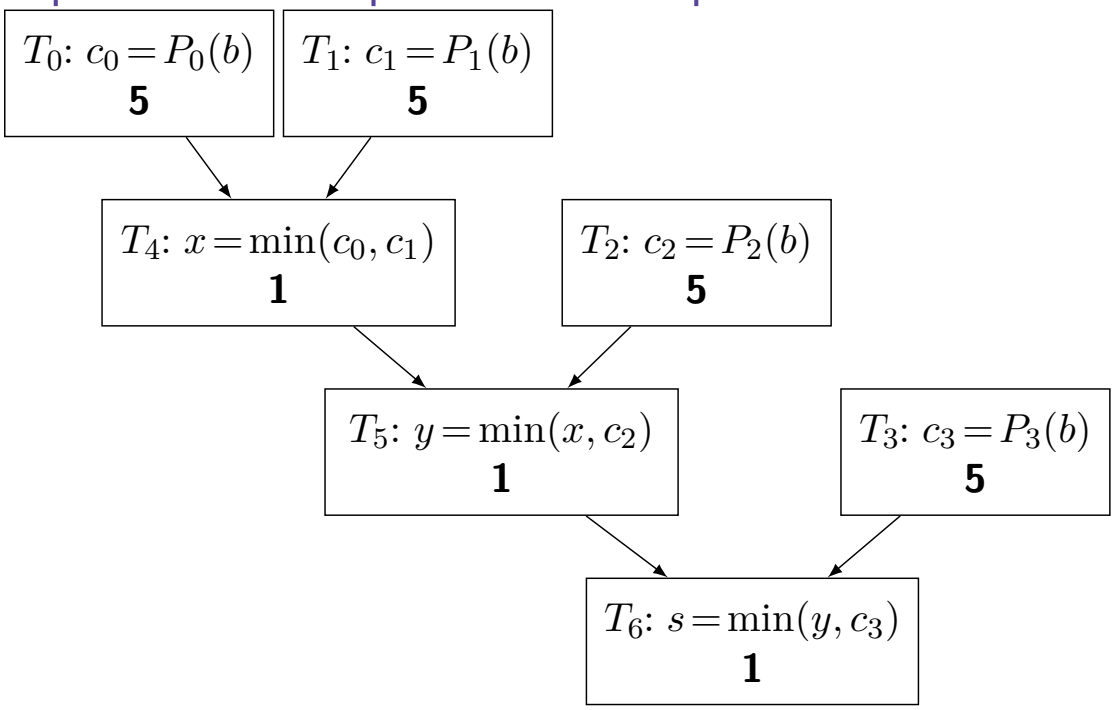
- Una tasca per cada avaluació de polinomi
  $\rightarrow$ independents entre sí
- Varies tasques per a calcular valors mínims de dos en dos (recursivament)

Exemple de Descomposició en Tasques: Graf 1



$$L = 7, \quad M = \frac{5+5+5+5+1+1+1}{7} = 3.28$$

Exemple de Descomposició en Tasques: Graf 2

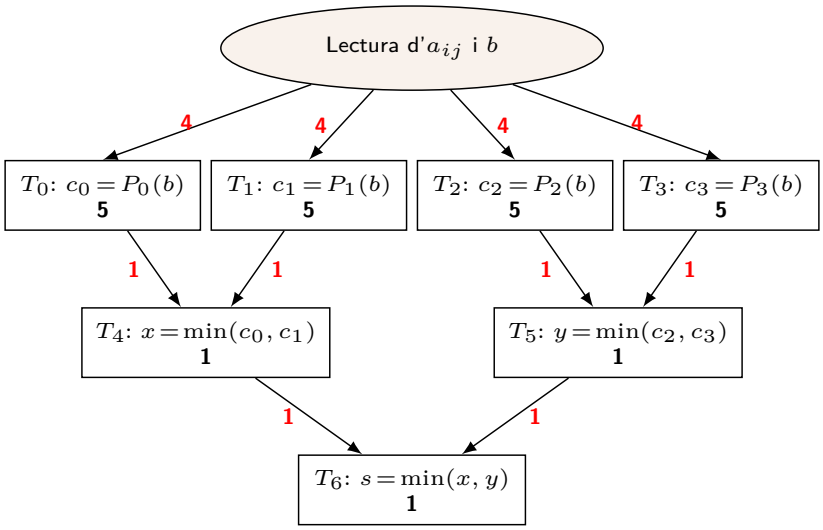


$$L = 8, \quad M = \frac{5+5+5+5+1+1+1}{8} = 2.875$$

Graf amb Comunicació

A voltes el graf incorpora informació relativa a la comunicació

- Possibilitat d'afegir nodes auxiliars (sense cost)
- Aristes amb pes: denoten la comunicació entre tasques (valor proporcional a la quantitat de dades)



Apartat 3

Avaluació de Prestacions (I)

- Paràmetres Absoluts
- Prestacions en Memòria Compartida

Avaluació de Prestacions

El principal objectiu en la computació paral·lela és augmentar les prestacions

- És fonamental conèixer com es comporten les diferents parts d'un programa paral·lel
- És també important saber com es comportarà davant canvis en el nombre de processadors i la grandària del problema

En aquest apartat es presenten diverses mesures i tècniques per a detectar on un programa paral·lel baixa el seu rendiment i comparar-ho amb implementacions seqüencials i altres configuracions

Tipus d'Anàlisi

Anàlisi a priori

- Realitzat abans de la implementació del programa sobre el pseudocodi o el disseny del programa
- Independent de la màquina en la qual s'executa
- Permet identificar la millor opció a l'hora d'implementar un programa
- Permet determinar la grandària dels problemes adequada i les característiques del hardware adequat

Anàlisi a posteriori

- Realitzat sobre una implementació i màquina específica i utilitzant un conjunt de dades d'entrada
- Permet analitzar colls de botella i detectar condicions no observades en el disseny

Anàlisi Teòrica

El cost s'analitza en funció de la grandària del problema: n

En molts casos el cost només depèn de n: $t(n)$

Però en ocasions, davant un mateix n, pot haver-hi un comportament diferent en funció de les dades d'entrada

- Cost del cas més favorable
- Cost del cas més desfavorable
- Cost mitjà
  Mitjanant els temps de cadascuna de les possibles entrades per la probabilitat que aquestes apareguen

En la pràctica, s'usen cotes asimptòtiques (inferior i superior)

Concepte de Flop

Flop: *floating point operation* - unitat de mesura per a:

- Cost dels algoritmes
- Rendiment dels computadors (flop/s)

1 flop = cost d'una operació elemental en coma flotant (producte, suma, divisió, resta)

- Considerem menyspreable el cost de les operacions en aritmètica entera
- El cost d'altres operacions en coma flotant s'avaluarà sobre la base del Flop
  $\rightarrow$ per exemple, una arrel quadrada igual a 8 flops

Suposa una unitat de mesura del cost independent de la màquina (el temps que tarda un flop varia d'un processador a un altre)

Notació Asimptòtica

Notació $\mathcal{O}$

- Permet acotar superiorment, excepte constants i asimptòticament, la forma en què creix una funció
- En la pràctica es correspon amb el terme d'ordre superior de l'expressió del cost sense considerar el seu coeficient
  - Exemple: la multiplicació matriu per vector és $\mathcal{O}(n^2)$

Notació o (o-xicoteta)

- Té en compte a més el coeficient de major ordre
- Adequat quan comparem dos algoritmes que tenen el mateix ordre $\mathcal{O}$
  - Exemple: el producte de matriu triangular per vector pot realitzar-se mitjançant l'algorisme convencional amb cost $o(2n^2)$ o mitjançant un algorisme optimitzat $o(n^2)$

Paràmetres per a Avaluar les Prestacions

Paràmetres Absoluts

- Permeten conèixer el cost real que tenen els algoritmes paral·lels
- Suposen la base per al càlcul dels paràmetres relatius que permeten comparar algoritmes
- Són els més importants en problemes de temps real

Paràmetres Relatius

- Permeten comparar els algoritmes paral·lels entre si i amb les versions seqüencials
- Proporcionen informació del grau d'aprofitament dels processadors

Paràmetres Absoluts

- Temps d'execució d'un algorisme seqüencial: $t(n)$
- Temps d'execució d'un algorisme paral·lel: $t(n, p)$
  - Temps aritmètic: $t_a(n, p)$
  - Temps de comunicacions: $t_c(n, p)$
- Cost total: $C(n, p)$
- *Overhead*: $t_o(n, p)$

Notació:

- Quan la talla del problema és sempre n, sense ambigüitat, s'ometrà, per exemple: $t(p)$
- A voltes usarem subíndexs en comptes de funcions: t_p, C_p

Temps d'Execució

Temps que tarda a executar-se l'algorisme seqüencial (en un sol processador, $t(n)$) o l'algorisme paral·lel (en p processadors, $t(n,p)$)

- El cost a priori es mesurarà en flops
  - Només es tindrà en consideració el nombre d'operacions en coma flotant
- Experimentalment el cost es mesurarà en segons

Expressions útils per al càlcul del cost computacional:

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

Cost Computacional: Exemples

```
PER A i=1 FINS A n
  PER A j=1 FINS A n
    x = x + a[i,j]
  FPER
FPER
```

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

```
PER A i=1 FINS A n
  PER A j=i FINS A n
    x = x + 3.0*a[i,j]
  FPER
FPER
```

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

```
PER A i=1 FINS A n
  PER A j=i FINS A n
    PER A k=i FINS A n
      x = x + a[i,k]
    FPER
  FPER
FPER
```

$$t(n) = \sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=i}^{n} 1 \approx \sum_{i=1}^{n}\sum_{j=i}^{n}(n-i) \approx$$
$$\sum_{i=1}^{n}(n^2 - 2ni + i^2) = \sum_{i=1}^{n} n^2 - 2n\sum_{i=1}^{n} i +$$
$$\sum_{i=1}^{n} i^2 \approx n^3 - \frac{2n^3}{2} + \frac{n^3}{3} = \frac{n^3}{3} \text{ flops}$$

Cost Total i *Overhead*

L'execució d'un algorisme paral·lel sol implicar un temps extra respecte de l'algoritme seqüencial

El cost total paral·lel comptabilitza el total de temps emprat en un algorisme paral·lel

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

L'*overhead* indica quin és el cost afegit respecte a l'algorisme seqüencial

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

Speedup i Eficiència

El *speedup* indica el guany de velocitat que aconsegueix l'algorisme paral·lel pel que fa a un algorisme seqüencial

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

Cal indicar a què es refereix $t(n)$

- Pot ser el millor algorisme seqüencial conegut
- Pot ser l'algorisme paral·lel executat en 1 processador

La eficiència mesura el grau d'aprofitament que un algorisme paral·lel fa d'un computador paral·lel

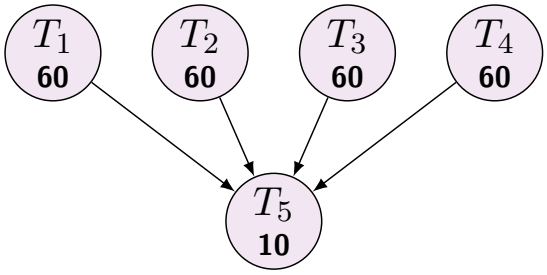
$$E(n,p) = \frac{S(n,p)}{p}$$

Sol expressar-se en tant per cent (o tant per 1)

Exemple d'Anàlsi de Prestacions Bàsic

Suposem aquest graf de dependencies

(en aquest exemple, el cost no depén de n)



Suposem que el alg. seqüencial realitza T_1, T_2, T_3, T_4, T_5

Temps seqüencial: $t_1 = 60 + 60 + 60 + 60 + 10 = 250$

Temps paral·lel per a $p = 4$, on T_1, T_2, T_3, T_4 s'executen concurrentment: $t_p = 60 + 10 = 70$

Speedup i eficiència:

$$S_p = \frac{t_1}{t_p} = \frac{250}{70} = 3.57 \qquad E_p = \frac{S_p}{p} = \frac{3.57}{4} = 0.89$$

Quin serà l'speedup per a $p = 2$, $p = 3$ i $p > 4$?

Com Obtindre Bones Prestacions

Idealment, per a p processadores tenim un *speedup* igual a p (eficiència igual a 1)

De què depén que ens apropem més o menys?

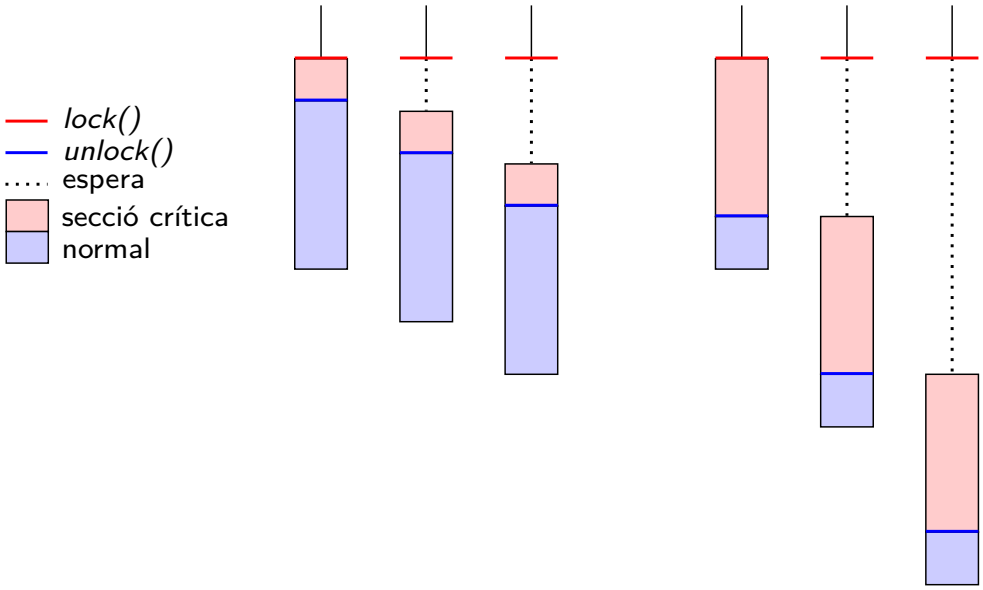
- Diseny de la paral·lelització apropiat
  - Repartiment de la càrrega equilibrat
  - Minimitzar temps en què els processadors estan ociosos
  - *Overhead* mínim possible
- Aspectes específics de l'arquitectura on s'executa
  - Distints en memòria compartida o pas de missatges
  - El temps d'accés a les dades no es considera en l'anàlisi teòrica del cost però és molt important en les arquitectures actuals

Sincronització: Eficiència

La sincronització pot tenir impacte negatiu en l'eficiència

La secció crítica ha de ser el més xicoteta possible

- En cas contrari es produeix una "serialització"



Igualment, hi ha que evitar barreres en tant que siga possible

Apartat 4

Disseny d'Algoritmes: Descomposició en Tasques

- Descomposició de Domini
- Altres Descomposicions

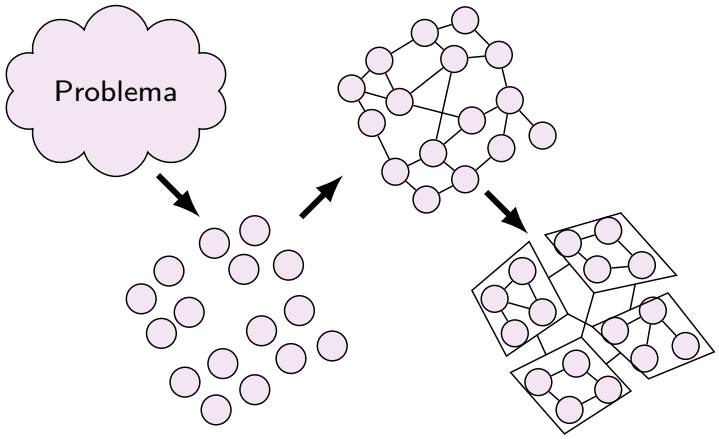
Disseny d'Algoritmes Paral·lels

El disseny d'algoritmes paral·lels presenta una complexitat molt major que en el cas seqüencial

- Concurrència (implica comunicació i sincronització)
- Assignació de dades i codi a processadors
- Accés simultani a dades compartides
- Escalabilitat, per a un nombre creixent de processadors

Els principals passos en el disseny són:

- Descomposició en tasques
- Assignació de tasques



Descomposició en Tasques

Tasca: cadascuna de les unitats de computació definides pel programador que potencialment poden ser executades en paral·lel

- El procés de dividir un càlcul/programa en tasques es denomina descomposició

Granularitat

- La descomposició pot ser de gra fi o gra gros
- Normalment es fa una descomposició de gra fi i posteriorment s'agrupen en tasques més grans

Tècniques de Descomposició

- Descomposició del domini
- Descomposició funcional dirigida pel flux de dades
- Descomposició recursiva
- Altres: descomposició exploratòria, descomposició especulativa, enfocaments mixts

Descomposició del Domini

En el cas de grans estructures de dades regulars

- Es divideixen les dades en parts de grandària similar (subdominis)
- A cada subdomini se li associa una tasca, la qual realitzarà les operacions necessàries sobre les dades del subdomini

Sol utilitzar-se quan és possible aplicar el mateix conjunt d'operacions sobre les dades de cada subdomini

La descomposició pot ser:

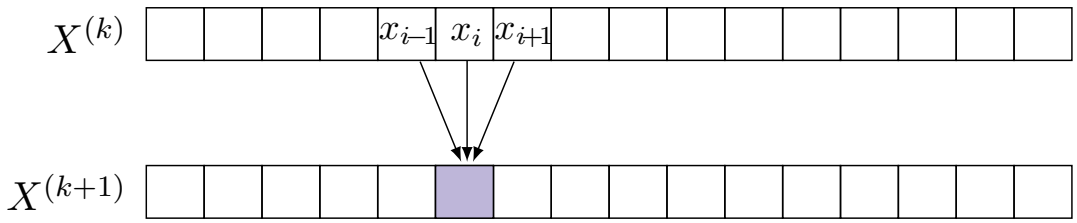
- Centrada en les dades d'eixida
- Centrada en les dades d'entrada
- Centrada en les dades intermèdies
- Descomposició basada en blocs (algoritmes matricials)

D. D. Centrada en les Dades d'Eixida

Cada component de les dades d'eixida es pot calcular de forma independent de la resta

Exemple: dissenyar un algorisme paral·lel iteratiu que calcule la successió de vectors $X^{(0)}, X^{(1)}, \ldots, X^{(k)}, X^{(k+1)}, \ldots \in \mathbb{R}^n$, on $X^{(0)}$ és un vector conegut i la resta s'obtenen així:

$$x_i^{(k+1)} = \frac{x_{i-1}^{(k)} - x_i^{(k)} + x_{i+1}^{(k)}}{2}, \quad i = 0, \ldots, n-1$$
$$x_{-1}^{(k)} = x_{n-1}^{(k)}, \quad x_n^{(k)} = x_0^{(k)}$$



D. D. Centrada en les Dades d'Entrada

Exemple: Producte escalar de vectors

$$\left.\begin{array}{l} x = [x_0, x_1, \ldots, x_{n-1}] \\ y = [y_0, y_1, \ldots, y_{n-1}] \end{array}\right\} \quad \Rightarrow \quad x \cdot y = \sum_{i=0}^{n-1} x_i y_i$$

Suposant p tasques i n divisible entre p, llavors la tasca i-èsima $(i = 0, \ldots, p-1)$ calcularia

$$\sum_{j=i\frac{n}{p}}^{(i+1)\frac{n}{p}-1} x_j y_j$$

Finalment, hi hauria tasques addicionals per a acumular les sumes parcials en la suma global

Descomposició Funcional

La descomposició funcional dirigida pel flux de dades s'utilitza quan

- La resolució del problema es pot descompondre en fases
- En cada fase s'executa un algorisme diferent

Se solen seguir els següents passos:

1. S'identifiquen les diferents fases
2. A cada fase se li assigna una tasca
3. S'analitzen els requisits de dades per a cadascuna de les tasques
   - Si el solapament de dades entre diferents tasques és mínim i el flux de dades entre elles és relativament xicotet, la descomposició estarà completa
   - Si no ocorre açò, seria necessari analitzar un altre tipus de descomposició

Descomposició Recursiva

És un mètode per a obtenir concurrència en problemes que poden resoldre's mitjançant la tècnica de divideix i venceràs

1. Dividir el problema original en dos o més subproblemes
2. Al seu torn aquests subproblemes es divideixen en dos o més subproblemes i així successivament fins a finalitzar el procés usant cert criteri de parada
3. Les dades obtingudes es combinen adequadament per a obtenir el resultat final

Pot implementar-se de diferents formes:

- Treballadors replicats amb borsa de tasques
- Algoritme recursiu

Descomposició Recursiva

Exemple: Ordenació Quicksort