CPA – Computació Paral·lela

Grau en Enginyeria Informàtica

T3. Pas de Missatges. Disseny Avançat 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 Pas de Missatges
- Model
- Detalls

2 Esquemes Algorítmics
- Paral·lelisme de Dades
- Paral·lelisme de Treballs

3 Avaluació de Prestacions (II)
- Temps Paral·lel
- Paràmetres Relatius

4 Disseny d'Algoritmes: Assignació de Tasques
- El Problema de l'Assignació
- Estratègies d'Agrupament i Replicació

5 Esquemes d'Assignació
- Esquemes d'Assignació Estàtica

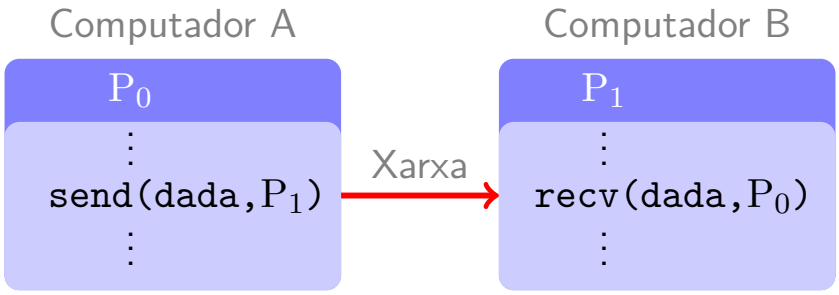
Apartat 1

Model de Pas de Missatges

- Model
- Detalls

Model de Pas de Missatges

- Les tasques gestionen el seu espai de memòria privat
- S'intercanvien dades a través de missatges
- La comunicació sol requerir operacions coordinades (p.e. enviament i recepció)
- Programació laboriosa / control total de la paral·lelització



MPI: Message Passing Interface

Creació de Processos

El programa paral·lel es compon de diferents processos

- Solen correspondre's amb processos del S.O.
- Normalment un procés per processador
- Cadascun té un identificador o índex (enter)

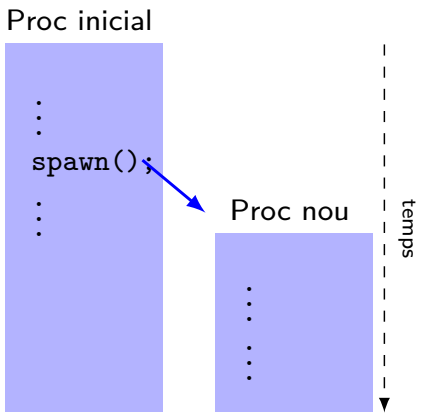
La creació de processos pot ser:

Estàtica: a l'inici del programa

- En línia de comandos (`mpiexec`)
- Existeixen durant tota l'execució
- És el més habitual

Dinàmica: durant l'execució

- Primitiva `spawn()`



Comunicadors

Els processos s'organitzen en grups

- Per a operacions col·lectives, com l'enviament 1 a tots
- Es defineixen mitjançant índexs o amb operacions de conjunts (unió, intersecció, etc.)

Concepte més general: Comunicador = grup + context

- La comunicació en un comunicador no pot interferir amb la d'un altre
- Útil per a aïllar la comunicació dins d'una llibreria
- Es defineixen a partir de grups o altres comunicadors
- Comunicadors predefinits:
  - Món (*world*): format per tots els processos creats per `mpiexec`
  - Propi (*self*): format per un sol procés

Operacions Bàsiques d'Enviament/Recepció

L'operació més comuna és la comunicació punt a punt

- Un procés envia un missatge (`send`) i un altre el rep (`recv`)
- Cada `send` ha de tenir un `recv` aparellat
- El missatge és el contingut d'una o més variables

```
/* Procés 0 */
x = 10;
send(x,1);
x = 0;
```

```
/* Procés 1 */
recv(i,0);
```

L'operació `send` és segura des del punt de vista semàntic si es garanteix que el procés 1 rep el valor que tenia `x` abans de l'enviament (10)

Existeixen diferents modalitats d'enviament i recepció

Example: Suma de Vectors

$x = v + w$, $v \in \mathbb{R}^n$, $w \in \mathbb{R}^n$, $x \in \mathbb{R}^n$

- Suposem $p = n$ processos
- Inicialment `v`, `w` estan en P_0, i el resultat `x` ha d'emmagatzemar-se en P_0

```
SUB suma(v,w,x,n)
distribuir(v,w,vl,wl,n)
sumapar(v,w,vl,wl,x,xl,n)
combinar(x,xl,n)
```

```
SUB distribuir(v,w,vl,wl,n)
EN CADA P(i), i=0 FINS A n-1
  SI i == 0
    PER A j=1 FINS A n-1
      send(v[j],j)
      send(w[j],j)
    FPER
  SI NO
    recv(vl,0)
    recv(wl,0)
  FSI
```

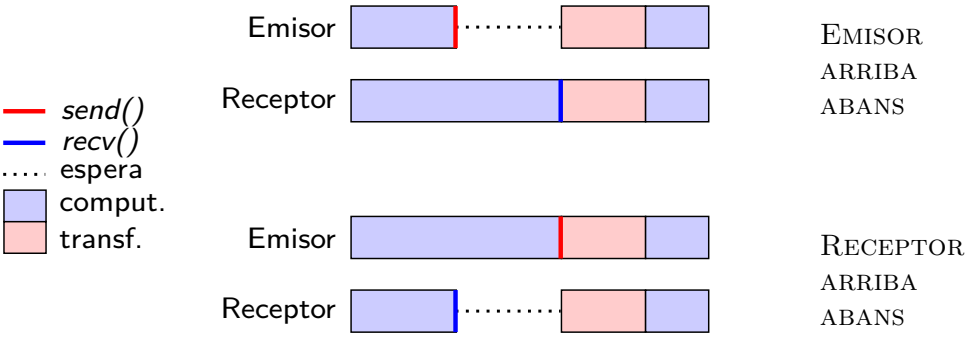
```
SUB sumapar(v,w,vl,wl,x,xl,n)
EN CADA P(i), i=0 FINS A n-1
  SI i == 0
    x[0] = v[0] + w[0];
  SI NO
    xl = vl + wl
  FSI
```

```
SUB combinar(x,xl,n)
EN CADA P(i), i=0 FINS A n-1
  SI i == 0
    PER A j=1 FINS A n-1
      recv(x[j],j)
    FPER
  SI NO
    send(xl,0)
  FSI
```

Enviament amb Sincronització

En el mode síncron, l'operació `send` no acaba fins que l'altre procés ha efectuat el `recv` corresponent

- A més de la transferència de dades, els processos se sincronitzen
- Requereix un protocol perquè emissor i receptor sàpien que pot començar la transmissió (és transparent al programador)



Modalitats d'Enviament/Recepció

Enviament amb buffer/enviament síncron

- Un *buffer* emmagatzema una còpia temporal del missatge
- El `send` amb buffer finalitza quan el missatge s'ha copiat de memòria del programa a un buffer del sistema
- El `send` síncron no finalitza fins que s'inicia el `recv` corresponent en l'altre procés

Operacions bloquejants/no bloquejants

- Al finalitzar la crida a `send` bloquejant es segur modificar la variable que s'envia
- Al finalitzar la crida a `recv` bloquejant es garanteix que la variable conté el missatge
- Les no bloquejants simplement inicien l'operació

Finalització de l'Operació

En les operacions no bloquejants cal poder determinar la finalització

- En el `recv` per a poder llegir el missatge
- En el `send` per a poder sobrescriure la variable

El `send` i `recv` no bloquejants ens donen un número d'operació *req*

Primitives:

- `wait(req)`: el procés es bloqueja fins que ha acabat l'operació *req*
- `test(req)` indica si ha finalitzat o no
- `waitany` i `waitall` quan hi ha diverses operacions pendents

Es pot usar per a solapar comunicació i càlcul

Selecció de Missatges

L'operació `recv` requereix un identificador de procés *id*

- No conclou fins que arriba un missatge d'*id*
- S'ignoren els missatges procedents d'altres processos

Per a més flexibilitat, es permet usar un "comodí" per a rebre de qualsevol procés

A més, s'usa una etiqueta (*tag*) per a distingir entre missatges

- També permet comodí per a indicar qualsevol etiqueta

Exemple: `recv(z,any_src,any_tag,status)` acceptarà el primer missatge que entre

- La primitiva `recv` té un argument `status` on apareix l'emissor i l'etiqueta
- Els missatges no seleccionats no es perden, queden en una "cua de missatges"

Problema: Interbloqueig

Un mal ús de `send` i `recv` pot produir interbloqueig

Cas de comunicació síncrona:

```
/* Procés 0 */
send(x,1);
recv(y,1);
```

```
/* Procés 1 */
send(y,0);
recv(x,0);
```

- Tots dos queden bloquejats en l'enviament

Cas d'enviament amb buffer:

- L'exemple anterior no causaria interbloqueig
- Pot haver-hi altres situacions amb interbloqueig

```
/* Procés 0 */
recv(y,1);
send(x,1);
```

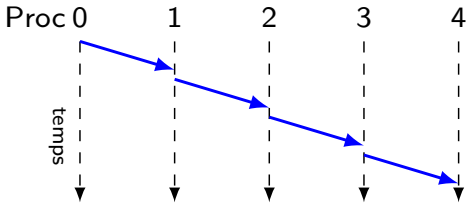
```
/* Procés 1 */
recv(x,0);
send(y,0);
```

Possible solució: intercanviar l'ordre d'un d'ells

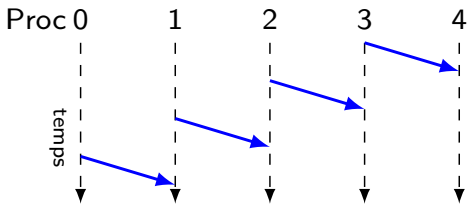
Problema: Serialització

Cada procés ha d'enviar una dada al seu veí dret

```
/* Procés i */
recv(y,i-1);
send(x,i+1);
```




```
/* Procés i */
send(x,i+1);
recv(y,i-1);
```



Possibles solucions:

- Protocol parells-imparells: els processos parells fan una variant, els imparells l'altra
- `send` o `recv` no bloquejant
- Operacions combinades: `sendrecv`

Comunicació Col·lectiva

Les operacions col·lectives involucren a tots els processos d'un comunicador (en molts casos, un d'ells té un paper destacat – procés arrel)

- Sincronització (`barrier`): tots els processos esperen a que arriben els demés
- Moviment de dades: un o més envien a un o més
- Reduccions: a més de comunicar es realitza un càlcul sobre les dades

Aquestes operacions poden realitzar-se amb comunicació punt a punt, però és recomanable usar la primitiva corresponent

- Existeixen diversos algoritmes per a cada cas (lineal, arbre)
- La solució òptima sol dependre de l'arquitectura (topologia de la xarxa)

Comunicació Col·lectiva: Tipus

- Difusió un a tots
  - Tots reben el que té el procés arrel
- Reducció tots a un
  - Operació dual a la difusió
  - Les dades es combinen mitjançant un operador associatiu
- Repartiment (*scatter*)
  - L'arrel envia un missatge individualitzat a cadascun
- Recollida o concatenació (*gather*)
  - Operació dual al repartiment
  - Similar a la reducció però sense operar
- Difusió tots a tots
  - p difusions simultànies, amb procés arrel diferent
  - Al final, tots emmagatzemen totes les dades
- Reducció tots a tots
  - Operació dual a la difusió tots a tots

Apartat 2

Esquemes Algorítmics

- Paral·lelisme de Dades
- Paral·lelisme de Treballs

Paral·lelisme de Dades / Particionat de Dades

En algoritmes amb moltes dades que es tracten de forma similar (típicament algoritmes matricials)

- En memòria compartida, es paral·lelitzen els bucles (cada fil opera sobre una part de les dades)
- En pas de missatges, es realitza un particionat de dades explícit

En pas de missatges pot ser desaconsellable paral·lelitzar

- El volum de computació ha de ser almenys un ordre de magnitud major que el de comunicacions
  - ✗ Vector-vector: cost $\mathcal{O}(n)$ enfront de $\mathcal{O}(n)$ comunicació
  - ✗ Matriu-vector: cost $\mathcal{O}(n^2)$ enfront de $\mathcal{O}(n^2)$ comunicació
  - ✓ Matriu-matriu: cost $\mathcal{O}(n^3)$ enfront de $\mathcal{O}(n^2)$ comunicació
- Sovint les dades estan ja distribuïdes

Cas 1: Producte Matriu-Vector

Solució en pas de missatges ($p = n$ processadors)

- Suposem que inicialment `v`, `A` estan en P_0
- El resultat `x` ha d'emmagatzemar-se en P_0

```
SUB matvec(A,v,x,n,m)
distribuir(A,Al,v,n,m)
mvlocal(A,v,xl,n,m)
combinar(xl,x,n)
```

```
SUB distribuir(A,Al,v,n,m)
EN CADA P(i), i=0 FINS A n-1
  SI i == 0
    PER A j=1 FINS A n-1
      enviar(A[j,:],j)
      enviar(v[:],j)
    FPER
    Al = A[0,:]
  SI NO
    rebre(Al,0)
    rebre(v[:],0)
  FSI
```

```
SUB mvlocal(Al,v,xl,n,m)
EN CADA P(i), i=0 FINS A n-1
  xl = 0
  PER A j=0 FINS A m-1
    xl = xl + Al[j] * v[j]
  FPER
```

```
SUB combinar(xl,x,n)
EN CADA P(i), i=0 FINS A n-1
  SI i == 0
    x[0] = xl
    PER A j=1 FINS A n-1
      rebre(x[j],j)
    FPER
  SI NO
    enviar(xl,0)
  FSI
```


Paral·lelisme de Tasques

En casos en què es generen més tasques que processos, o la resolució d'una tasca genera noves tasques

- L'assignació estàtica no és viable o té problemes de càrrega desequilibrada
- Assignació dinàmica: es van assignant a mesura que els processos queden ociosos

---

Sol implementar-se mitjançant un esquema asimètric: mestre-treballadors

- El mestre porta compte de les tasques fetes/per fer
- Els treballadors reben les tasques i notifiquen al mestre quan les han acabat

En alguns casos és possible una solució simètrica: treballadors replicats

Mestre-Treballadors

Exemple: Fractals amb pas de missatges (`np` processos)

Mestre

```
count=0; row=0;
for (k=1; k<np; k++) {
  send(row, k, data_tag);
  count++; row++;
}
do {
  recv({r,color}, slave, res_tag);
  count--;
  if (row<max_row) {
    send(row, slave, data_tag);
    count++; row++;
  }
  else
    send(row, slave, end_tag);
  display(r,color);
} while (count>0);
```

Treballadors

```
recv(y, master, src_tag);

while(src_tag == data_tag) {
  /*
   * compute row colors
   */
  send( {y,color}, master,
       res_tag);
  recv(y, master, src_tag);
}
```

`count` representa el nombre de processos amb tasca assignada

Es processen `max_row` línies de la imatge (independents)

Apartat 3

Avaluació de Prestacions (II)

- Temps Paral·lel
- Paràmetres Relatius

Temps d'Execució Paral·lel

Temps que tarda un algorisme paral·lel en p processadors

- Des que comença el primer fins que acaba l'últim

Es descompon en temps aritmètic i de comunicacions

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

t_a correspon a tots els temps de càlcul

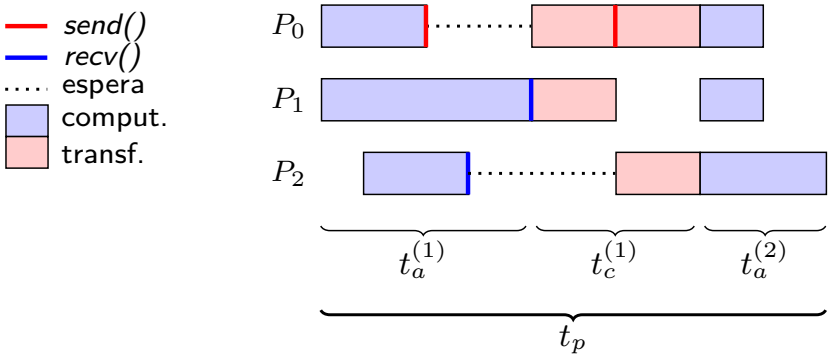
- Tots els processadors calculen concurrentment
- És com a mínim igual al màxim temps aritmètic

t_c correspon a temps associats a transferència de dades

- En memòria distribuïda t_c=temps d'enviament de missatges
- En memòria compartida t_c=temps de sincronització

Temps d'Execució Paral·lel: Components

Ex.: pas de missatges amb tres processos, P_0 envia P_1 a i P_2



En la pràctica:

- No hi ha separació clara entre fases de càlcul i comunicació (P_1 no ha d'esperar)
- A voltes es pot solapar comunicació i càlcul (amb operacions no bloquejants, per exemple P_2)

$$t_p = t_a + t_c - t_{\text{overlap}}$$

t_{overlap}: temps de solapament

Modelatge del Temps de Comunicació

Suposant pas de missatges, P_0 i P_1 en nodes diferents amb connexió directa

Temps necessari per a enviar un missatge de n bytes: $t_s + t_w n$

- Temps d'establiment de la comunicació, t_s
- Ample de banda, w (màxim nombre de bytes per seg.)
- Temps d'enviament d'1 byte, $t_w = 1/w$

En la pràctica és més complicat:

- Xarxa commutada, de latència no uniforme, col·lisions, ...

Recomanacions:

- Agrupar diversos missatges en un (n gran, t_s únic)
- Evitar moltes comunicacions simultànies

En memòria compartida, les consideracions són diferents

Exemple: Producte Matriu-Vector (1)

$x = A \cdot v$, $A \in \mathbb{R}^{n \times n}$, $v \in \mathbb{R}^n$, $x \in \mathbb{R}^n$

Temps seqüencial:
$t(n) = 2n^2$ flops

Paral·lelització amb $p = n$ processadors

Temps paral·lel en memòria compartida:
$t(n,p) = 2n$ flops

Temps paral·lel en pas de missatges:

- `distribuir`: $2 \cdot (n-1) \cdot (t_s + t_w \cdot n)$
- `mvlocal`: $2n$ flops
- `combinar`: $(n-1) \cdot (t_s + t_w \cdot 1)$

$t(n,p) = 3 \cdot (n-1) \cdot t_s + (n-1) \cdot (2n+1) t_w + 2n$ flops

$\boxed{t(n,p) \approx 3nt_s + 2n^2 t_w + 2n \text{ flops}}$

Exemple: Producte Matriu-Vector (2)

Versió per a $p < n$ proc. (distribució per blocs de files)

```
SUB matvec(a,v,x,n,p)
distribuir(a,aloc,v,n,p)
mvlocal(a,aloc,v,x,n,p)
combinar(x,n,p)
```

```
SUB distribuir(a,aloc,v,n,p)
EN CADA P(i), i=0 FINS A p-1
  nb = n/p
  SI i == 0
    aloc = a[0:nb-1,:]
    PER A j=1 FINS A p-1
      send(a[j*nb:(j+1)*nb-1,:],j)
      send(v[:] ,j)
    FPER
  SI NO
    recv(aloc,0)
    recv(v,0)
  FSI
```

```
SUB mvlocal(a,aloc,v,x,n,p)
EN CADA P(pr), pr=0 FINS A p-1
  nb = n/p
  PER A i=0 FINS A nb-1
    x[i] = 0
    PER A j=0 FINS A n-1
      x[i] += aloc[i,j] * v[j]
    FPER
  FPER
```

```
SUB combinar(x,n,p)
EN CADA P(i), i=0 FINS A p-1
  nb = n/p
  SI i == 0
    PER A j=1 FINS A p-1
      recv(x[j*nb:(j+1)*nb-1],j)
    FPER
  SI NO
    send(x[0:nb-1],0)
  FSI
```


Exemple: Producte Matriu-Vector (3)

Paral·lelització amb $p < n$ processadors

Temps paral·lel en pas de missatges:

- `distribuir`:
  $(p-1)\cdot\left(t_s + t_w \cdot \frac{n^2}{p}\right)+(p-1)\cdot(t_s+t_w\cdot n) \approx 2pt_s+n^2t_w+pnt_w$
- `mvlocal`: $2\frac{n^2}{p}$ flops
- `combinar`: $(p-1)\cdot(t_s + t_w \cdot n/p) \approx pt_s + nt_w$

$$\boxed{t(n,p) \approx 3pt_s + (n^2+pn)t_w + 2\frac{n^2}{p} \text{ flops}}$$

Paràmetres Relatius

Els paràmetres relatius serveixen per a comparar un algorisme paral·lel amb un altre

- Speedup: $S(n, p)$
- Eficiència: $E(n, p)$

Normalment s'apliquen en l'anàlisi experimental, encara que el speedup i l'eficiència es poden obtenir en l'anàlisi teòrica

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

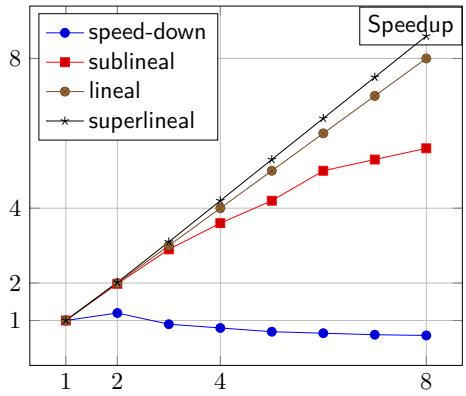
La eficiència mesura el grau d'aprofitament que un algorisme paral·lel fa d'un computador paral·lel

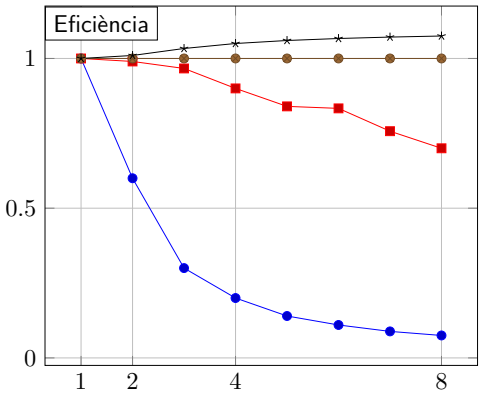
$$E(n, p) = \frac{S(n, p)}{p}$$

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

Speedup: Casos Possibles

$S(n,p) < 1$	$1 < S(n,p) < p$	$S(n,p) = p$	$S(n,p) > p$
"Speed-down"	Cas sublineal	Cas lineal	Cas superlineal
L'algorisme paral·lel és més lent que l'algorisme seqüencial	L'algorisme paral·lel és més ràpid que el seqüencial, però no aprofita tota la capacitat dels processadors	L'algorisme paral·lel és el més ràpid possible, aprofita els processadors al 100%	Situació anòmala, l'algorisme paral·lel té menor cost que el seqüencial





Exemple: Producte Matriu-Vector

Temps seqüencial: $t(n) = 2n^2$ flops

Paral·lelització per files ($p = n$ processadors)

En memòria compartida:
$t(n,p) = 2n$
$S(n,p) = n$
$E(n,p) = 1$

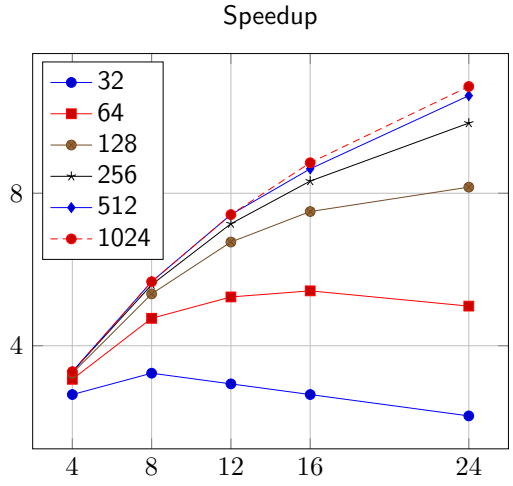
En pas de missatges:
$t(n,p) = 2n^2 t_w + 3nt_s + 2n$
$S(n,p) \to 1/t_w$
$E(n,p) \to 0$

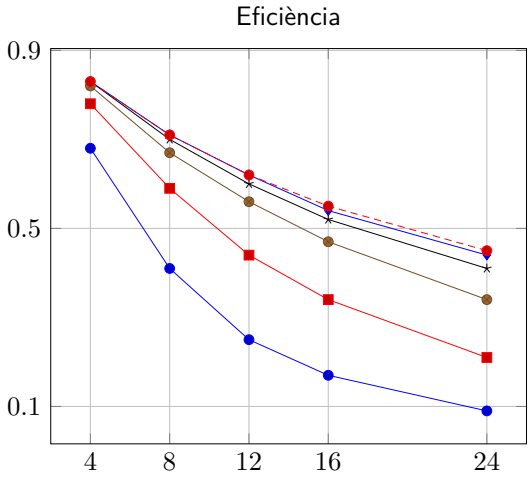
Paral·lelització per blocs de files ($p < n$ processadors)

En pas de missatges:
$t(n,p) = 3pt_s + (n^2 + pn)t_w + 2\frac{n^2}{p}$
$S(n,p) \to \frac{2p}{pt_w+2}$
$E(n,p) \to \frac{2}{pt_w+2}$

Variació de Prestacions

- Normalment l'eficiència disminueix a mesura que s'incrementa el nombre de processadors
- L'efecte és menys acusat per a grandàries de problema més grans





Llei d'Amdahl

Moltes vegades una part del problema no es pot paral·lelitzar
$\rightarrow$ La Llei d'Amdahl mesura el speedup màxim assolible

Donat un algorisme seqüencial, descomponem $t(n) = t_s + t_p$

- t_s és el temps de la part intrínsecament seqüencial
- t_p és el temps de la part perfectament paral·lelitzable (es pot resoldre amb p processadors)

El temps mínim paral·lel assolible serà $t(n,p) = t_s + \frac{t_p}{p}$

Speedup màxim:

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

Apartat 4

Disseny d'Algoritmes: Assignació de Tasques

- El Problema de l'Assignació
- Estratègies d'Agrupament i Replicació

Assignació de Tasques

- La fase de descomposició ha donat lloc a un conjunt de tasques
- Tenim un algorisme paral·lel abstracte *independent* de la plataforma hardware i *ineficient*
- És necessari *adaptar* la descomposició obtinguda a una arquitectura particular

---

L'assignació de tasques o planificació de tasques consisteix a determinar

- en quines unitats de processament i
- en quin ordre

s'executarà cada tasca

Processos i Processadors

- Procés: Unitat lògica de còmput capaç d'executar tasques computacionals
- Processador: Unitat hardware que realitza càlculs

Un algorisme paral·lel es compon de processos que executen tasques

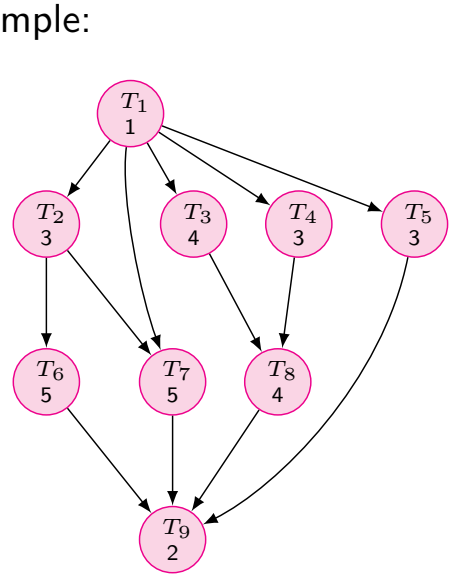
- L'assignació estableix correspondència entre tasques i processos en la fase de disseny
- La correspondència entre processos i processadors es fa al final i probablement en execució

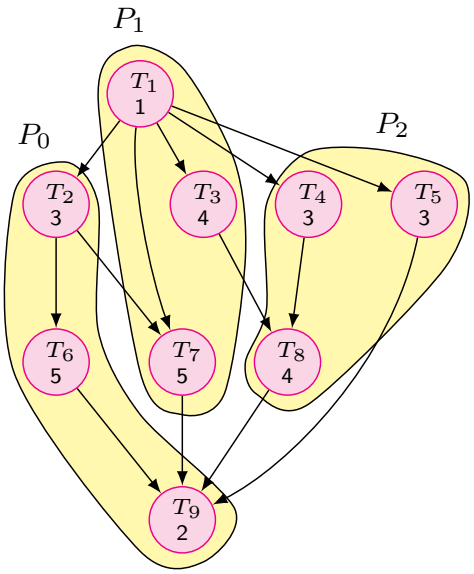
El Problema de l'Assignació. Exemple (1)

Assignació: Establir correspondència tasca–procés i seleccionar l'ordre d'execució

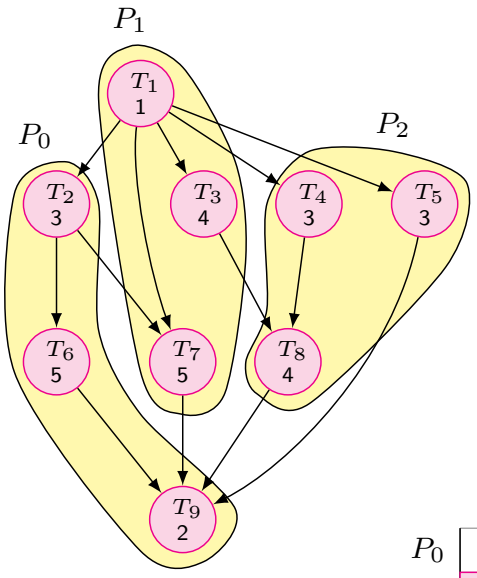
Sol incloure també l'agrupament previ d'algunes tasques

Exemple:

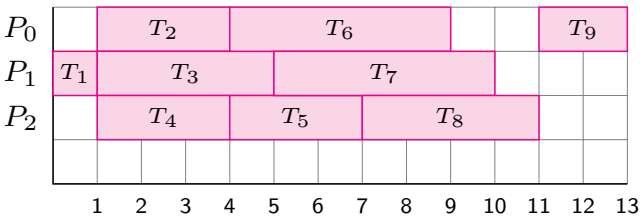




El Problema de l'Assignació. Exemple (2)



Ordre d'execució de les tasques segons l'assignació realitzada



Objectius de l'Assignació

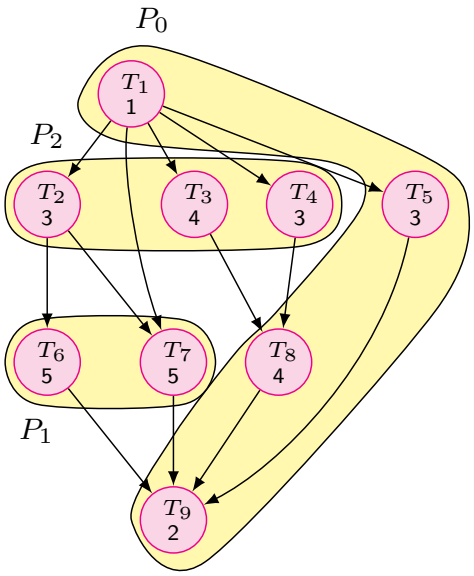
Objectiu: Minimitzar el temps d'execució

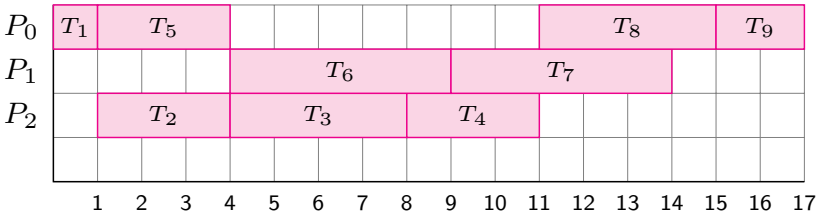
Composició del temps d'execució d'un algorisme paral·lel i estratègies de minimització:

- Temps de computació: Maximitzar concurrència assignant tasques independents a processos diferents
- Temps de comunicació: Assignar tasques que es comuniquen molt al mateix procés
- Temps d'oci: Minimitzar les dues causes d'ociositat:
  - Desequilibris de càrrega: es busca equilibrar els càlculs i les comunicacions entre processos (diagrama anterior)
  - Espera: es busca minimitzar l'espera de tasques que no estan preparades

Objectius de l'Assignació. Exemple

Exemple d'assignació amb càrrega equilibrada però amb més *esperes*





Estratègies Generals d'Assignació (1)

Assignació estàtica o planificació determinista: Les decisions d'assignació es prenen abans de l'execució. Passos:

1. S'estima el nombre de tasques, el seu temps d'execució i els costos de comunicació
2. S'agrupen tasques en altres majors per a reduir cost de comunicació
3. S'associen tasques a processos

El problema d'assignació estàtica òptima és *NP-complet* per al cas general[1]

Avantatges de l'assignació estàtica:

- No afig cap sobrecàrrega en temps d'execució
- Disseny i implementació són més senzills

[1] No es coneix cap algorisme amb temps polinomial que solucione el problema

Estratègies Generals d'Assignació (2)

Assignació dinàmica: El repartiment del treball computacional es realitza en temps d'execució

Aquest tipus d'assignació s'empra quan:

- Les tasques es generen dinàmicament
- La grandària de les tasques no es coneix a priori

En general, les tècniques dinàmiques són més complexes. El principal desavantatge és la sobrecàrrega induïda deguda a

- La transferència d'informació de càrrega i de treball computacional entre els processos
- La presa de decisions per a moure càrrega entre processos (es realitza en temps d'execució)

Avantatge: No és necessari conèixer el comportament a priori, són flexibles i apropiades per a arquitectures heterogènies

Agrupament (1)

L'agrupament s'utilitza per a reduir el nombre de tasques a fi de:

- Limitar costos de creació i destrucció de tasques
- Minimitzar els retards deguts a la interacció entre tasques (accés a dades locals en lloc de remotes)

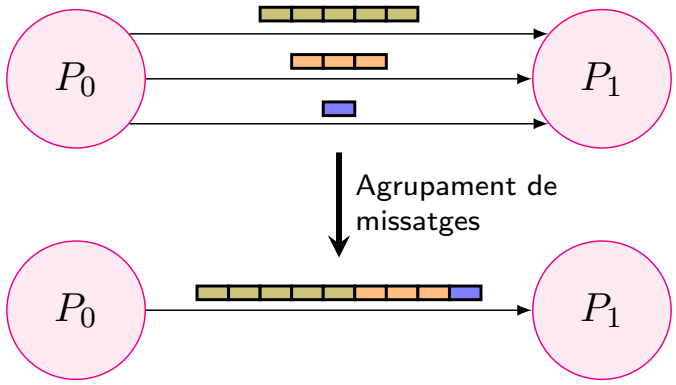
Estratègies d'agrupament:

- *Minimització del volum de dades transferides*. Distribució de tasques basada en blocs de dades (algoritmes matricials), agrupament de tasques no concurrents (grafs de tasques estàtics), emmagatzematge temporal de resultats intermedis (ex., producte escalar de dos vectors)
- *Reducció de la freqüència d'interaccions*. Minimitzar nombre de transferències i augmentar el volum de dades a transferir en cadascuna

Agrupament (2)

Reducció de la freqüència d'interaccions

- En *pas de missatges* significa reduir el nombre de missatges (latència) i augmentar la grandària d'aquests



- En *memòria compartida* significa reduir el nombre de fallades de cache

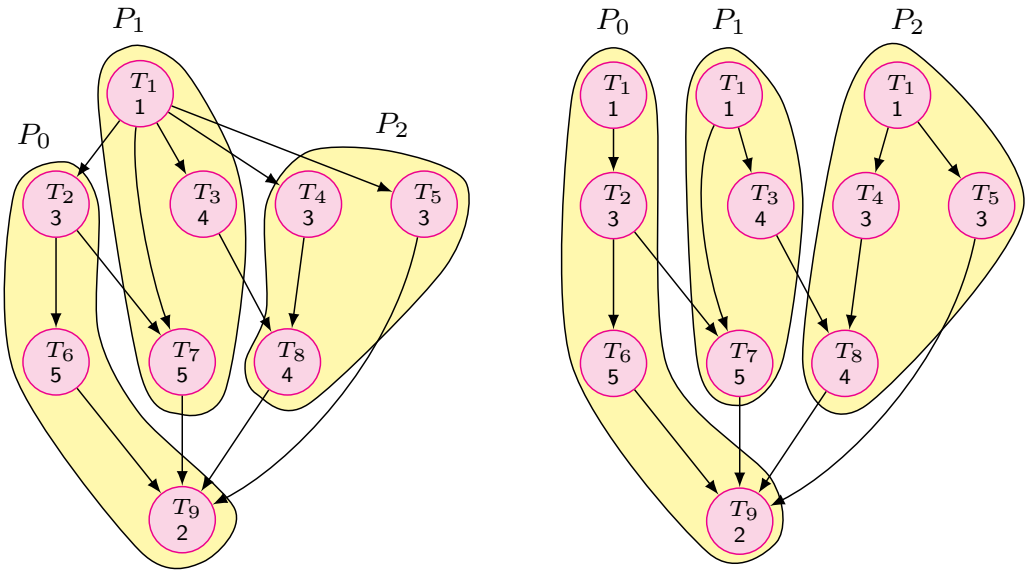
Replicació

La replicació implica que part dels càlculs o dades del problema no es reparteixen sinó que són realitzats o manejats per tots

- Replicació de dades: consisteix a copiar dades d'accés comú en els diferents processos a fi de reduir les comunicacions
  - En *memòria compartida* és implícit ja que solament afecta a la memòria cache
  - En *memòria distribuïda* pot comportar una millora considerable del rendiment i simplificació del disseny
- Replicació de còmput i comunicació: consisteix a repetir un càlcul en cadascun dels processos en cas que aquest càlcul siga menor que el cost de transferència i càlcul, tant seqüencial com paral·lel

Replicació. Exemple

Exemple de replicació de còmput i comunicacions. Siga el següent graf, suposant que les comunicacions tenen un cost. Repliquem la tasca T1



Apartat 5

Esquemes d'Assignació

- Esquemes d'Assignació Estàtica

Esquemes d'Assignació Estàtica

Esquemes estàtics per a descomposició de domini

- Se centren en estructures de dades de gran grandària
- L'assignació de tasques a processos consisteix a repartir les dades entre els processos
- Principalment dos tipus:
  - Distribució de matrius per blocs
  - Divisió estàtica de grafs

Esquemes sobre grafs de dependències estàtics

- Se solen obtenir mitjançant descomposició funcional del flux de dades o descomposició recursiva

Distribucions de Matrius per Blocs

En computació matricial sol succeir que el càlcul d'un element depèn d'elements contigus (localitat espacial)

- L'assignació fa correspondre porcions contigües (blocs) del domini de dades (matriu)

Distribucions per blocs més usuals:

1. Distribució unidimensional per blocs d'un vector
2. Distribució unidimensional per blocs de files d'una matriu
3. Distribució unidimensional per blocs de columnes d'una matriu
4. Distribució bidimensional per blocs d'una matriu

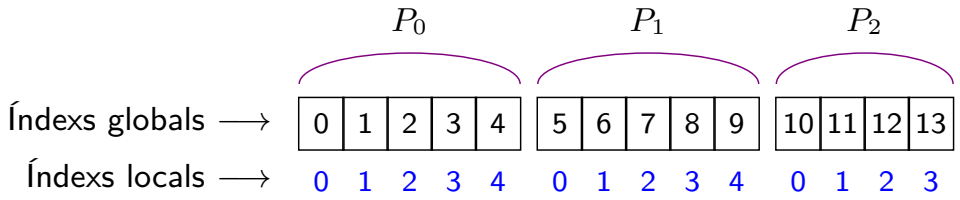
També vorem les variants cícliques

Distribució Unidimensional per Blocs

L'índex global i s'assigna al procés $\lfloor i/m_b \rfloor$ on $m_b = \lceil n/p \rceil$ és la grandària de bloc

L'índex local és $i \mod m_b$ (residu de la divisió entera)

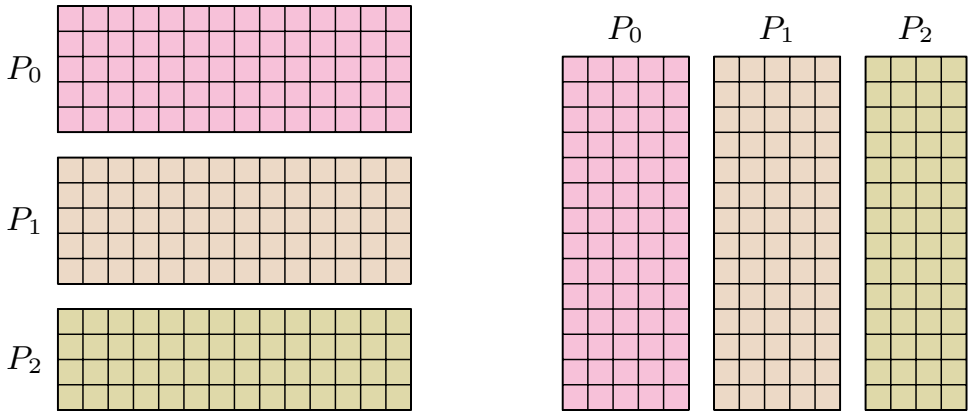
Exemple: per a un vector de 14 elements entre 3 processos



$m_b = \lceil 14/3 \rceil = 5$
Cada procés té m_b elements (excepte l'últim)

Distribució Unidimensional per Blocs. Exemple

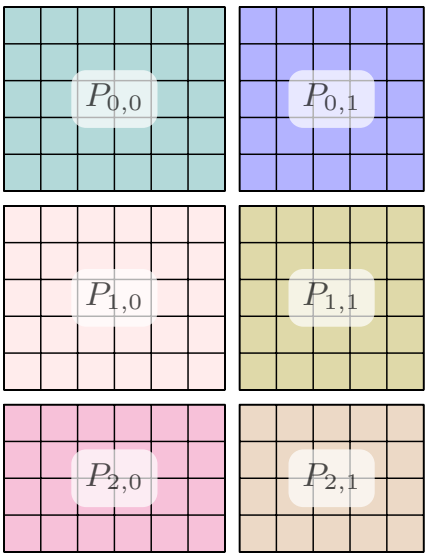
Exemple per a una matriu bidimensional de 14×14 entrades entre 3 processos per blocs de files i blocs de columnes



Cada procés posseeix $m_b = \lceil n/p \rceil$ files (o columnes)

Distribució Bidimensional per Blocs

Exemple de distribució bidimensional per blocs per a una matriu de $m \times n = 14 \times 11$ entre 6 processos organitzats en una malla lògica de 3×2



Cada procés posseeix un bloc de $m_b \times n_b = \lceil m/p_m \rceil \times \lceil n/p_n \rceil$, on p_m i p_n són la primera i segona dimensió de la malla de processos, respectivament (3 i 2 en l'exemple)

Exemple: Diferències Finites (1)

Càlcul iteratiu sobre una matriu $A \in \mathbb{R}^{n \times n}$

- Al començar té un valor donat $A^{(0)}$
- En l'iteració k-èsima ($k = 0, 1, \dots$) s'obté un nou valor $A^{(k+1)} = \left(a_{i,j}^{(k+1)}\right)$, $i, j = 0, \dots, n-1$, on

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - \Delta t \left(\frac{a_{i+1,j}^{(k)} - a_{i-1,j}^{(k)}}{0.1} + \frac{a_{i,j+1}^{(k)} - a_{i,j-1}^{(k)}}{0.02} \right)$$

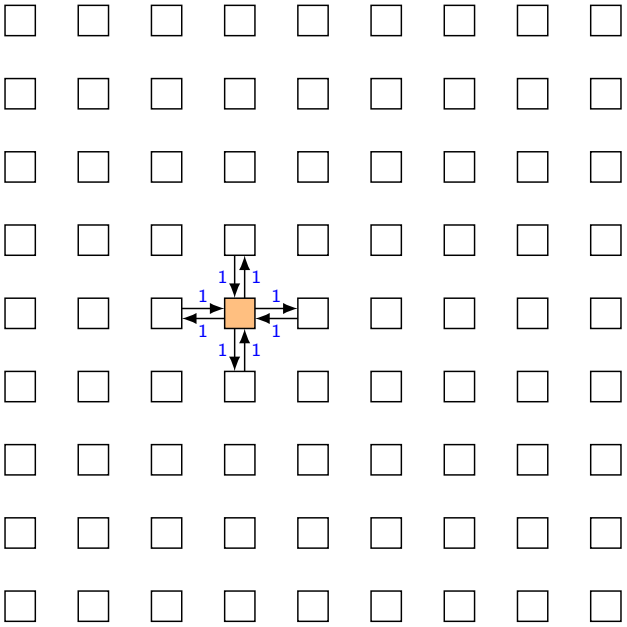
i determinades condicions de contorn

Vorem a continuació l'esquema de comunicacions de l'algoritme per a diferents distribucions (per a $n = 9$)

Exemple: Diferències Finites (2)

Sense agrupament

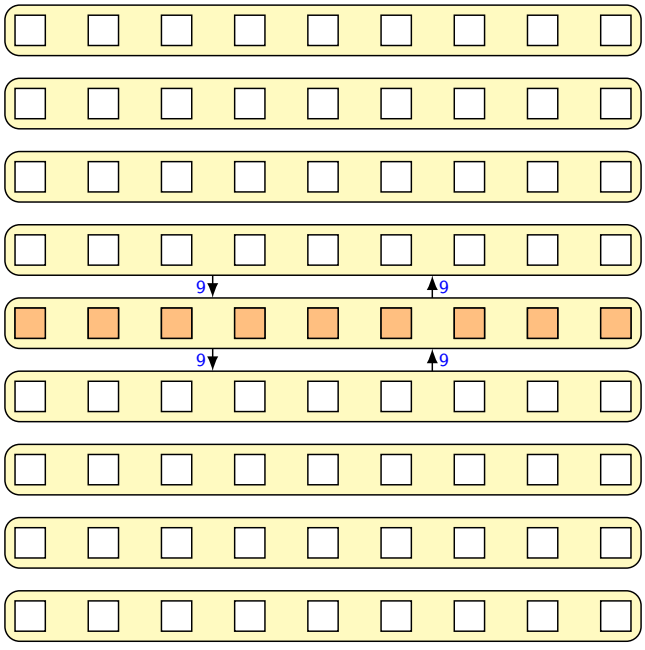
- 4 enviaments per tasca (1 element cadascun)
- 288 enviaments totals, 288 elements transferits



Exemple: Diferències Finites (3)

Agrupament unidimensional

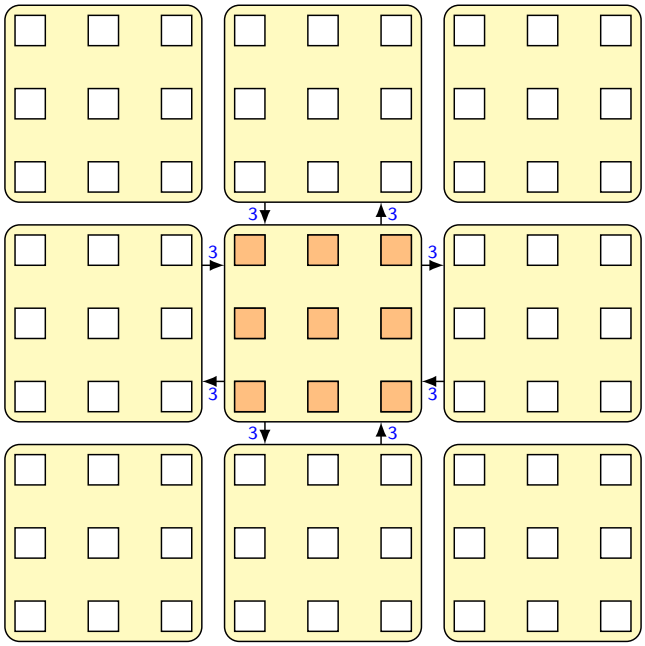
- 2 enviaments per tasca (9 elements cadascun)
- 16 enviaments totals, 144 elements transferits



Exemple: Diferències Finites (4)

Agrupament bidimensional:

- 4 enviaments per tasca (3 elements cadascun)
- 24 enviaments totals, 72 elements transferits



Efecte Volum-Superfície

L'agrupament millora la localitat

- Redueix el volum de comunicacions
- Interessa l'agrupament amb més computació i menys comunicacions

Efecte volum-superfície

- La càrrega de computació augmenta proporcionalment al nombre d'elements assignats a la tasca (volum en matrius 3D)
- El cost de comunicacions augmenta proporcionalment al perímetre de la tasca (superfície en matrius 3D)

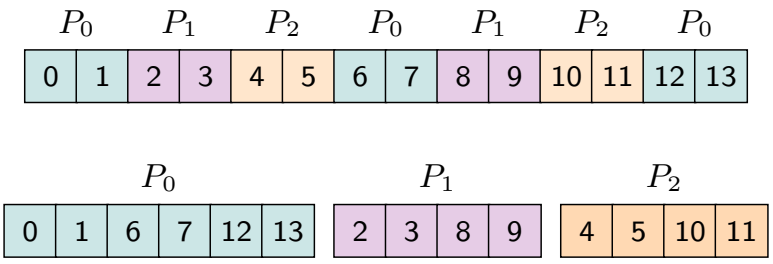
Aquest efecte augmenta amb el nombre de dimensions de la matriu

Distribucions Cícliques

Objectiu: equilibrar la càrrega durant tot el temps d'execució

- Major cost de comunicació al reduir-se la localitat
- Es combina amb els esquemes per blocs
- Ha d'existir un equilibri entre repartiment de càrrega i costos de comunicació: grandària adequada de bloc

Distribució cíclica unidimensional (grandària de bloc 2):



S'aplica igualment a matrius (per files o columnes)

Distribució Cíclica Bidimensional

Exemple de distribució cíclica per blocs bidimensional:
Matriu de 8×11 elements en blocs de 2×3 en una malla de 3×4 processos