Algorithmique parallèle

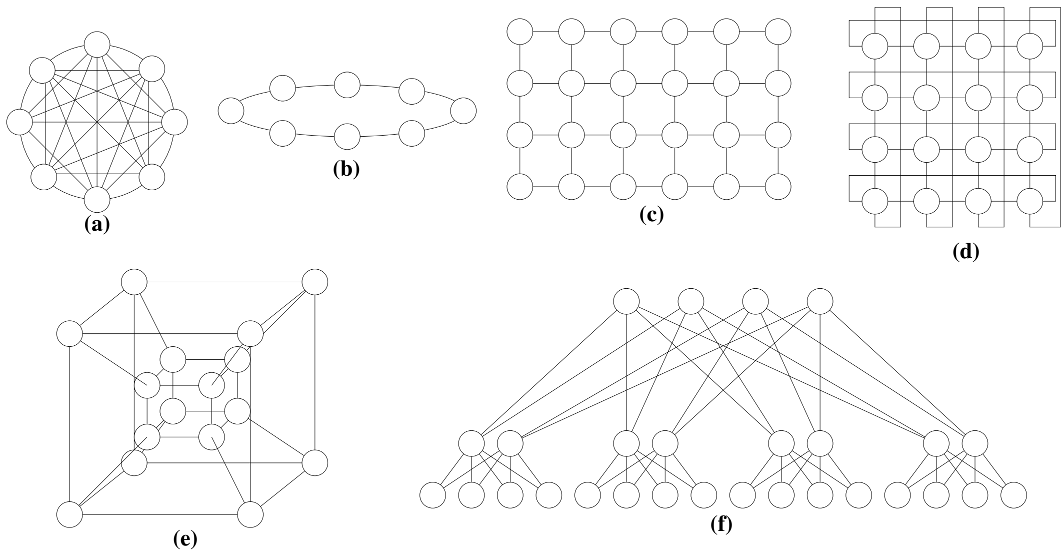
Plan

1. Communications et routage
2. Produit de matrices sur grille 2D
3. Algorithmique hétérogène

1 Réseaux d'interconnexion

Topologies

– Topologie statique : réseau d'interconnexion fixe
– Topologie dynamique : modifiée en cours d'exécution (configurer un *switch*).



– Borne de Moore sur le nombre de processeurs p d'une topologie statique de degré Δ :

$$p = 1 + \Delta + \Delta(\Delta - 1) + \Delta(\Delta - 1)^2 + \ldots + \Delta(\Delta - 1)^{D-1}$$

Topologie	**Nbre de proc.**	**Degré**	**Diamètre**	**Nbre de liens**
Réseau complet	p	$p-1$	1	$p(p-1)/2$
Anneau	p	2	$\lfloor p/2 \rfloor$	p
Grille 2D	$\sqrt{p}\sqrt{p}$	$2 \rightarrow 4$	$2(\sqrt{p}-1)$	$2p - 2\sqrt{p}$
Tore 2D	$\sqrt{p}\sqrt{p}$	4	$2\lfloor \sqrt{p}/2 \rfloor$	$2p$
Hypercube	$p = 2^d$	$d = \log(p)$	d	$p\log(p)/2$

2 Routage

Modèle "store-and-forward" (SF)

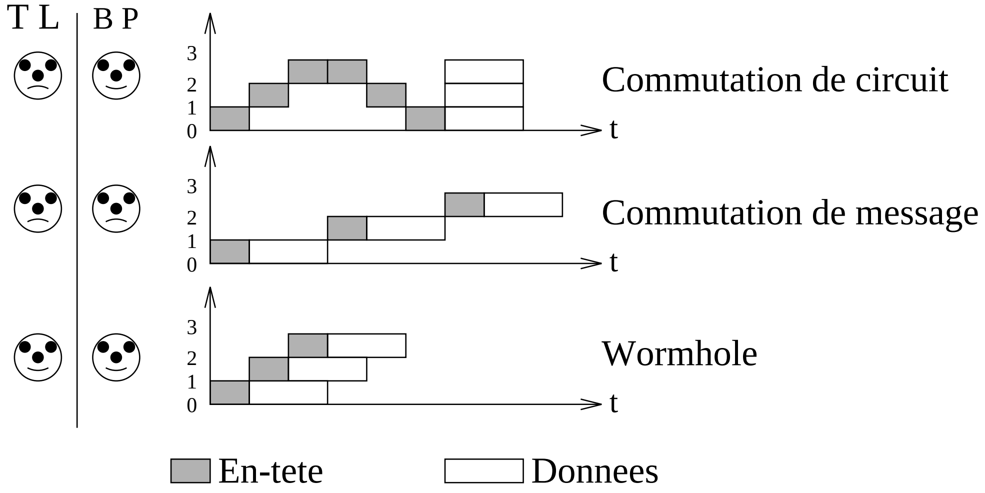
- Chaque processeur intermédiaire reçoit et stocke le message M avant de le ré-émettre en direction du processeur destination
- Modèle à commutation de message
- Première génération de machines
- Coût : $d(x, y)(\beta + L\tau)$
- Pipeline $\rightarrow$ $(\sqrt{(d(x, y) - 1))\beta} + \sqrt{L\tau})^2 = L\tau + o(\sqrt{L})$.

Modèle "cut-through" (CT)

- Co-processeur de communication
- Coût $\beta + (d(x, y) - 1)\delta + L\tau$
- $\delta \ll \beta$: chemin calculé par matériel, partie logicielle dans β
- Deux protocoles :
  1. *Circuit-switching (CC)*
     - Chemin créé avant émission des premiers octets
     - Puis message acheminé directement entre source et destination
     - Immobilisation des liens
     - iPSC d'Intel
  2. *Wormhole (WH)*
     - Adresse du destinataire placée dans l'en-tête du message
     - Routage sur chaque processeur
     - Message découpé en petits paquets appelés *flits*
     - Blocage $\rightarrow$ flits stockés dans les registres internes des routeurs intermédiaires
     - Paragon d'Intel.

Comparaison

– CC et WH plus efficaces que SF : masquer la distance entre les processeurs communiquants
– WH construit sa route tandis que le message avance dans le réseau
– CC construit son chemin (et l'acquitte) avant l'envoi des premiers paquets de données.



TL = Temps de latence
BP = Bande passante.

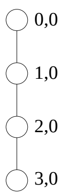
3 Produit de matrices sur grille 2D

Allocation des données sur les processeurs

- Algorithmes par blocs$\rightarrow$ augmenter le grain de calcul et améliorer l'utilisation des mémoires hiérarchiques
- Distribuer les données de manière cyclique $\rightarrow$ équilibrer la charge
- Distribution cyclique par blocs pour un vecteur de M composantes sur p processeurs, $0 \leq m < M$ et $0 \leq q < p$:

$$m \mapsto (q, b, i) = \left(\lfloor \frac{m \mod T}{r} \rfloor,\ \lfloor \frac{m}{T} \rfloor,\ m \mod r \right)$$

où r est la taille de bloc et $T = rp$.

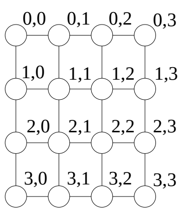


0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0
2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0
2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0
3,0	3,0	3,0	3,0	3,0	3,0	3,0	3,0	3,0	3,0

(a) r=3, s=10, P=4, Q=1

0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0
3,0	3,0	3,0	3,0	3,0	3,0	3,0	3,0	3,0	3,0
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0	2,0
3,0	3,0	3,0	3,0	3,0	3,0	3,0	3,0	3,0	3,0
0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0

(b) r=1, s=10, P=4, Q=1



0,0	0,0	0,0	0,1	0,1	0,1	0,2	0,2	0,2	0,3
0,0	0,0	0,0	0,1	0,1	0,1	0,2	0,2	0,2	0,3
0,0	0,0	0,0	0,1	0,1	0,1	0,2	0,2	0,2	0,3
1,0	1,0	1,0	1,1	1,1	1,1	1,2	1,2	1,2	1,3
1,0	1,0	1,0	1,1	1,1	1,1	1,2	1,2	1,2	1,3
1,0	1,0	1,0	1,1	1,1	1,1	1,2	1,2	1,2	1,3
2,0	2,0	2,0	2,1	2,1	2,1	2,2	2,2	2,2	2,3
2,0	2,0	2,0	2,1	2,1	2,1	2,2	2,2	2,2	2,3
2,0	2,0	2,0	2,1	2,1	2,1	2,2	2,2	2,2	2,3
3,0	3,0	3,0	3,1	3,1	3,1	3,2	3,2	3,2	3,3

(c) r=3, s=3, P=4, Q=4

0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3	0,0	0,1
1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3	1,0	1,1
2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3	2,0	2,1
3,0	3,1	3,2	3,3	3,0	3,1	3,2	3,3	3,0	3,1
0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3	0,0	0,1
1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3	1,0	1,1
2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3	2,0	2,1
3,0	3,1	3,2	3,3	3,0	3,1	3,2	3,3	3,0	3,1
0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3	0,0	0,1
1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3	1,0	1,1

(d) r=1, s=1, P=4, Q=4

(a) et (b) réseau linéaire, (c) et (d) grille

Implémentation dans une bibliothèque

1. **Version centralisée** Routines appelées en supposant les données sur la machine hôte. Résultats stockés sur la machine hôte.
   + minimise le nombre de routines à écrire
   + choisir la distribution des données la plus adaptée au noyau de calcul
   - coût prohibitif si on enchaîne plusieurs appels
2. **Version distribuée** Données déjà distribuées sur les processeurs, résultats distribués également.
   + *scalable*
   + routines de redistribution des données
   - compromis à trouver entre *coût de redistribution + coût de l'algorithme avec rangement adapté* et *coût de l'algorithme avec rangement non adapté*.

Produit de matrices sur grille 2D

- $C = C + AB$, avec A, B et C de taille $N \times N$
- $p = q^2$, grille cible = tore de taille $q \times q$
- Distribution des matrices par blocs : P_{ij} stocke A_{ij}, B_{ij} et C_{ij}.

Algorithme de Cannon

$$\begin{matrix} C00 & C01 & C02 & C03 \\ C10 & C11 & C12 & C13 \\ C20 & C21 & C22 & C23 \\ C30 & C31 & C32 & C33 \end{matrix} = \begin{matrix} A00 & A01 & A02 & A03 \\ A11 & A12 & A13 & A10 \\ A22 & A23 & A20 & A21 \\ A33 & A30 & A31 & A32 \end{matrix} \; \mathrm{x} \; \begin{matrix} B00 & B11 & B22 & B33 \\ B10 & B21 & B32 & B03 \\ B20 & B31 & B02 & B13 \\ B30 & B01 & B12 & B23 \end{matrix}$$

/* mouvements de données avant les calculs */
Rotations de A et B

/* calcul du produit de matrice */
pour $k = 1$ à $\sqrt{P}$ en // faire
   LocalProdMat(A, B, C)
   Rotation verticale de B
   Rotation horizontale de A
finpour

/* mouvements de données après les calculs */
Rotations de A et B

Algorithme de Fox

$$\begin{matrix} C00 & C01 & C02 & C03 \\ C10 & C11 & C12 & C13 \\ C20 & C21 & C22 & C23 \\ C30 & C31 & C32 & C33 \end{matrix} \;+\!=\; \begin{matrix} A00 & A00 & A00 & A00 \\ A11 & A11 & A11 & A11 \\ A22 & A22 & A22 & A22 \\ A33 & A33 & A33 & A33 \end{matrix} \;\mathrm{x}\; \begin{matrix} B00 & B01 & B02 & B03 \\ B10 & B11 & B12 & B13 \\ B20 & B21 & B22 & B23 \\ B30 & B31 & B32 & B33 \end{matrix}$$

$$\begin{matrix} C00 & C01 & C02 & C03 \\ C10 & C11 & C12 & C13 \\ C20 & C21 & C22 & C23 \\ C30 & C31 & C32 & C33 \end{matrix} \;+\!=\; \begin{matrix} A01 & A01 & A01 & A01 \\ A12 & A12 & A12 & A12 \\ A23 & A23 & A23 & A23 \\ A30 & A30 & A30 & A30 \end{matrix} \;\mathrm{x}\; \begin{matrix} B10 & B11 & B12 & B13 \\ B20 & B21 & B22 & B23 \\ B30 & B31 & B32 & B33 \\ B00 & B01 & B02 & B03 \end{matrix}$$

/* pas de mouvements de données avant les calculs */

/* calcul du produit de matrices */
Diffuser la première diagonale de A
pour $k = 1$ à $\sqrt{P} - 1$ en // faire
   LocalProdMat(A, B, C)
   Rotation verticale de B
   Diffuser la diagonale suivante de A
finpour
en // faire
   LocalProdMat(A, B, C)
   Rotation verticale de B
fin

/* pas de mouvements de données après les calculs */

Algorithme de Snyder

$$
\begin{array}{cccc}
\mathbf{C00} & C01 & C02 & C03 \\
C10 & \mathbf{C11} & C12 & C13 \\
C20 & C21 & \mathbf{C22} & C23 \\
C30 & C31 & C32 & \mathbf{C33}
\end{array}
=
\begin{array}{cccc}
A00 & A01 & A02 & A03 \\
A10 & A11 & C12 & A13 \\
A20 & A21 & A22 & A23 \\
A30 & A31 & A32 & A33
\end{array}
\ \mathrm{x}\
\begin{array}{cccc}
B00 & B10 & B20 & B30 \\
B01 & B11 & B21 & B31 \\
B02 & B12 & B22 & B32 \\
B03 & B13 & B23 & B33
\end{array}
$$

$$
\begin{array}{cccc}
C00 & \mathbf{C01} & C02 & C03 \\
C10 & C11 & \mathbf{C12} & C13 \\
C20 & C21 & C22 & \mathbf{C23} \\
\mathbf{C30} & C31 & C32 & C33
\end{array}
=
\begin{array}{cccc}
A00 & A01 & A02 & A03 \\
A10 & A11 & C12 & A13 \\
A20 & A21 & A22 & A23 \\
A30 & A31 & A32 & A33
\end{array}
\ \mathrm{x}\
\begin{array}{cccc}
B01 & B11 & B21 & B31 \\
B02 & B12 & B22 & B32 \\
B03 & B13 & B23 & B33 \\
B00 & B10 & B20 & B30
\end{array}
$$

/* mouvements de données avant les calculs */
transposer la matrice B

/* calcul du produit de matrices */
en // faire
   LocalProdMat(A, B, C)
   Rotation verticale de B
finpour
pour $k = 1$ à $\sqrt{P} - 1$ en // faire
   Somme globale sur les lignes pour $C_{i,(i+k-1) mod \sqrt{P}}$
   LocalProdMat(A, B, C)
   Rotation verticale de B
finpour
Somme globale sur les lignes pour $C_{i,(i+\sqrt{P}-1) mod \sqrt{P}}$

/* mouvements de données après les calculs */
transposer la matrice B

4 Algorithmique hétérogène

Allocation statique de tâches

– $t_1, t_2, \ldots, t_p$ temps de cycle des processeurs
– B tâches identiques et indépendantes
– Principe : $c_i \times t_i = Constante$

$$c_i = \frac{\frac{1}{t_i}}{\sum_{k=1}^{p} \frac{1}{t_k}} \times B$$

– Allocation optimale :

Distribue$(B, t_1, t_2, \ldots, t_p)$
{ Initialisation : calcule c_i tels que $c_i \times t_i \approx Constante$
et $c_1 + c_2 + \ldots + c_p \leq B$ }
do $i = 1, p$
$\quad c_i = \left\lfloor \frac{\frac{1}{t_i}}{\sum_{i=1}^{p} \frac{1}{t_i}} \times B \right\rfloor$
{ Incrémenter itérativement les c_i
qui minimisent le temps d'exécution tant que $\sum_{k=1}^{p} c_i < B$}
while $\sum_{i=1}^{p} c_i < B$ do
$\quad$ trouve $k \in \{1, \ldots, p\}$ tel que $t_k \times (c_k + 1) = \min\{t_i \times (c_i + 1)\}$
$\quad c_k = c_k + 1$
Retourne$(c_1, c_2, \ldots, c_k)$

Algorithme incrémental

– Allocation optimale pour tout nombre de tâches entre 1 et B

– Programmation dynamique avec $t_1 = 3$, $t_2 = 5$, et $t_3 = 8$:

Nombre d'atomes	c_1	c_2	c_3	Coût	Proc. sélectionné	Allocation σ
0	0	0	0		1	
1	1	0	0	3	2	$\sigma[1] = 1$
2	1	1	0	2.5	1	$\sigma[2] = 2$
3	2	1	0	2	3	$\sigma[3] = 1$
4	2	1	1	2	1	$\sigma[4] = 3$
5	3	1	1	1.8	2	$\sigma[5] = 1$
6	3	2	1	1.67	1	$\sigma[6] = 2$
7	4	2	1	1.71	1	$\sigma[7] = 1$
8	5	2	1	1.87	2	$\sigma[8] = 1$
9	5	3	1	1.67	3	$\sigma[9] = 2$
10	5	3	2	1.6		$\sigma[10] = 3$

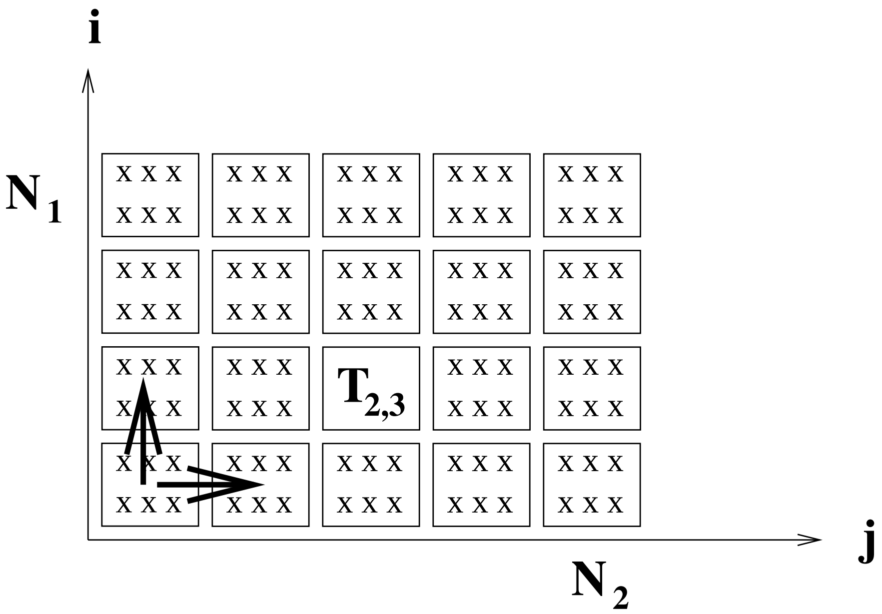
Distribue$(B, t_1, t_2, \ldots, t_p)$
$\{$ Initialisation : aucune tâche à distribuer. $m = 0$ $\}$
do $i = 1, p$
  $c_i = 0$
$\{$ Construit itérativement l'allocation $\sigma\}$
do $m = 1, B$
  trouve $k \in \{1, \ldots, p\}$ tel que $t_k \times (c_k + 1) = \min\{t_i \times (c_i + 1)\}$
  $c_k = c_k + 1$
  $\sigma[m] = k$
Retourne$(\sigma, c_1, c_2, \ldots, c_k)$

Exemple d'application

– Domaine de calcul rectangulaire

– Vecteurs de dépendance

$$\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$$



Allocation pérodique

– p processeurs de temps de cycle $t_1, t_2, \ldots, t_p$

– Allocation de N colonnes par tranches de B colonnes

– Calculer les c_i et trier les $c_i t_i$

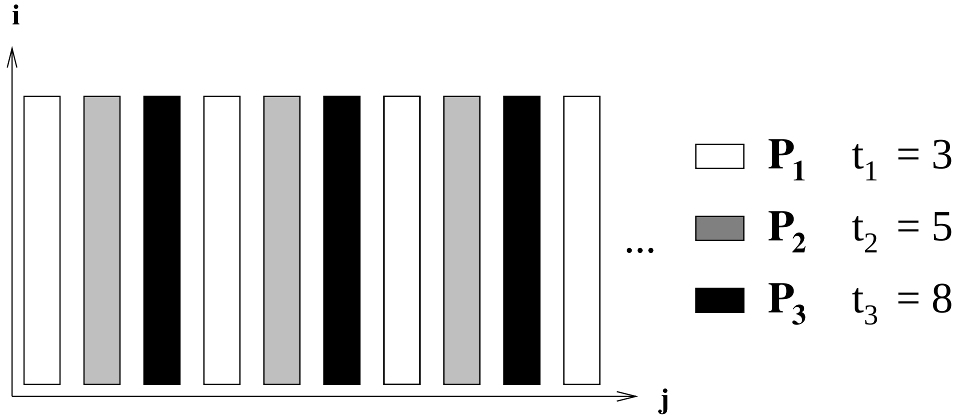
– Allouer, dans cet ordre, c_i colonnes consécutives à P_i

Illustration

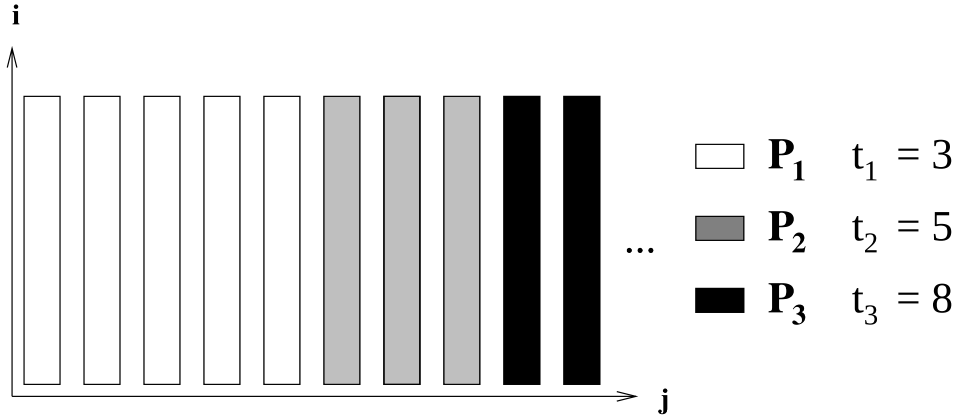
– 3 processeurs, $t_1 = 3$, $t_2 = 5$, et $t_3 = 8$, $B = 10$

– $T_{seq} \approx 3N$

– Glouton $\rightarrow$ solution cyclique de période B, $T \approx \frac{8}{3}N$



– Périodique $\rightarrow c_1 = 5$, $c_2 = 3$ et $c_3 = 2$, $T \approx 1.6N$



– $T_{opt} \approx \frac{120}{79}N = 1.52N$:
$B = 79 \rightarrow c_1 = 40$, $c_2 = 24$, $c_3 = 15 \rightarrow c_1t_1 = c_2t_2 = c_3t_3$
$\Rightarrow$ équilibrage parfait.

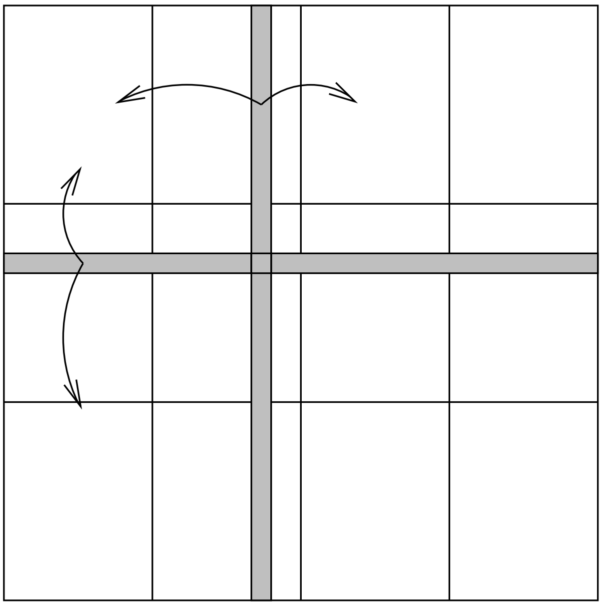
Retour au balayage d'image

– 3 processeurs de temps de cycle $t_1 = 1$, $t_2 = 2$ and $t_3 = 4$

– Période $B = 7$ calculée pour un équilibrage de charge optimal.

$$
\begin{array}{lllllllllllll}
P_1 & \boxed{0} & \boxed{1} & \boxed{3} & \boxed{6} & \boxed{10} & \boxed{14} & \boxed{18} & \boxed{22} & \boxed{26} & \boxed{30} & \cdots \\
P_1 & \boxed{2} & \boxed{4} & \boxed{7} & \boxed{11} & \boxed{15} & \boxed{19} & \boxed{23} & \boxed{27} & \boxed{31} & \cdots \\
P_1 & \boxed{5} & \boxed{8} & \boxed{12} & \boxed{16} & \boxed{20} & \boxed{24} & \boxed{28} & \boxed{32} & \cdots \\
P_1 & \boxed{9} & \boxed{13} & \boxed{17} & \boxed{21} & \boxed{25} & \boxed{29} & \boxed{33} & \cdots \\
P_2 & \boxed{14} & \boxed{18} & \boxed{22} & \boxed{26} & \boxed{30} & \boxed{34} & \cdots \\
P_2 & \boxed{20} & \boxed{24} & \boxed{28} & \boxed{32} & \boxed{36} & \cdots \\
P_3 & \boxed{26} & \boxed{30} & \boxed{34} & \boxed{38} & \cdots \\
\cdots & \cdots
\end{array}
$$

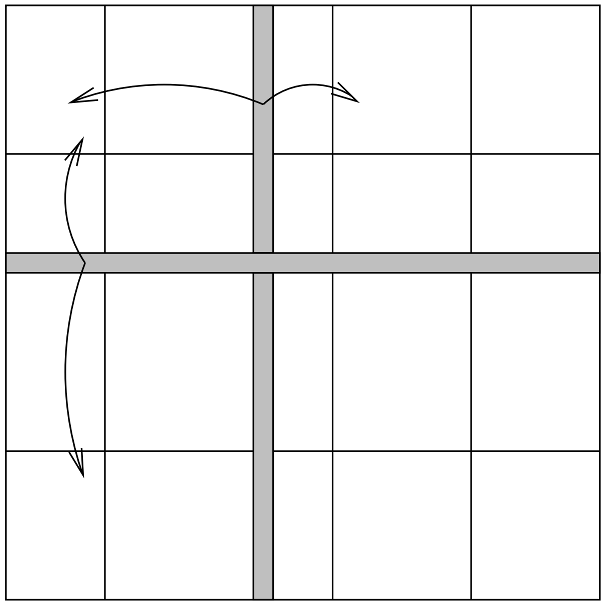
Multiplication de matrices sur grille homogène



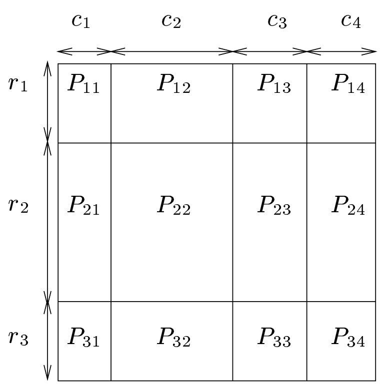
- Algorithme par blocs de ScaLAPACK
- Double diffusion horizontale et verticale
- S'adapte au cas de matrices et grilles
- Aucune redistribution initiale des données.

Multiplication de matrices sur grille hétérogène

– Allouer des rectangles de tailles différentes aux processeurs, en fonction de leur vitesse relative

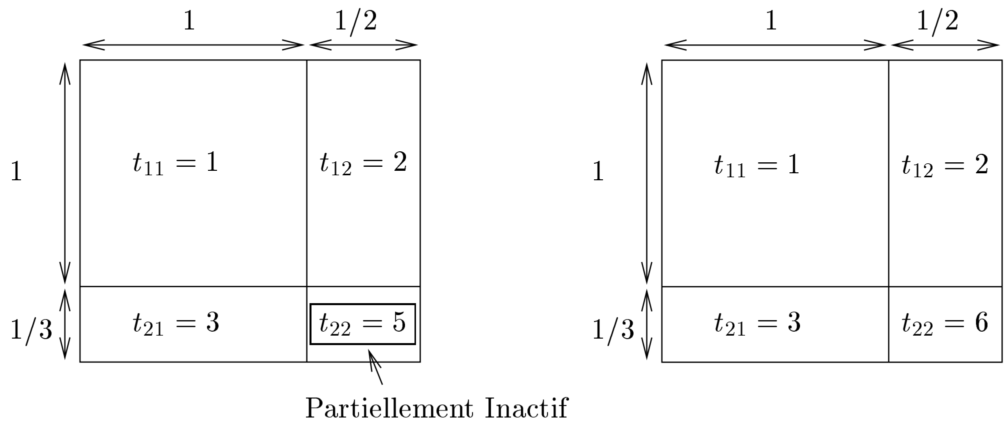


– $p \times q$ processeurs $P_{i,j}$ de temps de cycle $t_{i,j}$



Equilibrage de charge parfait

– Matrice $T = (t_{ij})$ de rang 1



Optimisation

– Objectif Obj_1 : $\min_{(\sum_i r_i=1;\sum_j c_j=1)} \max_{i,j}\{r_i \times t_{ij} \times c_j\}$

– Objectif Obj_2 : $\max_{r_i \times t_{ij} \times c_j \leq 1}\{(\sum_i r_i) \times (\sum_j c_j)\}$

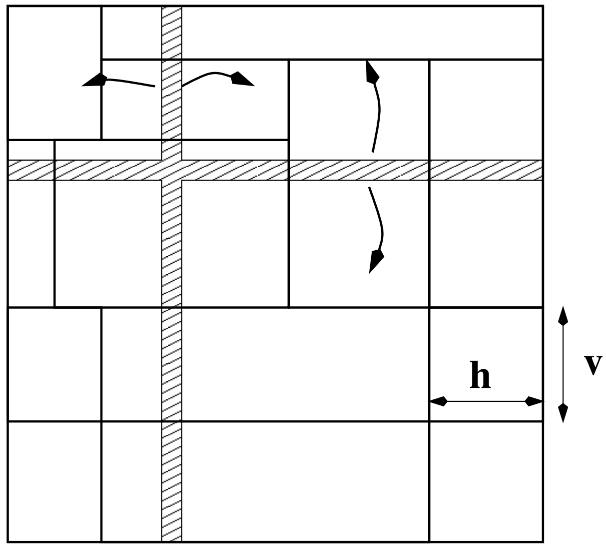
– La position des processeurs dans la grille n'est pas une donnée du problème : toutes les permutations sont possibles, et il faut chercher la meilleure

– Problème est NP-complet

– Difficulté de l'équilibrage de charge 2D !

Partitionnement libre

– Avec 13 processeurs ?



– p processeurs de vitesses $s_1, s_2, \ldots, s_p$ normalisées : $\sum_{i=1}^{p} s_i = 1$

– Partitionner le carré unité en p rectangles de surfaces $s_1, s_2, \ldots, s_p$

– Surface des rectangles $\Longleftrightarrow$ vitesses relatives des processeurs

– Forme des rectangles $\Longleftrightarrow$ minimiser les communications

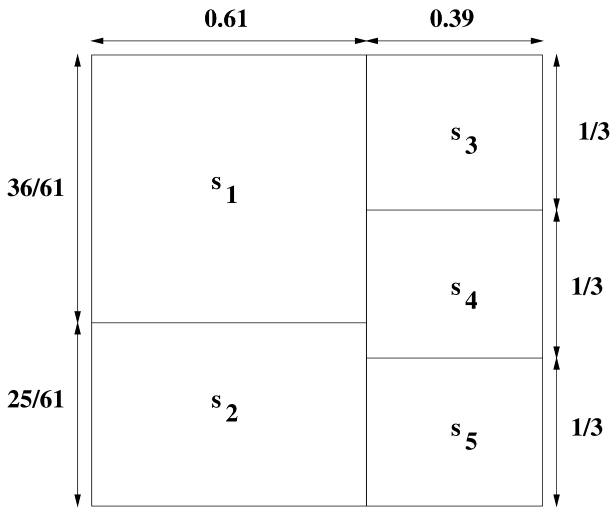
– Géométriquement : partitionner le carré unité en p rectangles d'aires fixées $s_1, s_2, \ldots, s_p$ (tels que $\sum_{i=1}^{p} s_i = 1$) afin de minimiser
  – soit la somme des demi-périmètres des rectangles dans le cas de communications séquentielles
  – soit le plus grand des demi-périmètres des rectangles dans le cas de communications parallèles

– Problèmes NP-complets, of course.

Exemple

– $p = 5$ rectangles $R_1, \ldots, R_5$

– Aires $s_1 = 0.36$, $s_2 = 0.25$, $s_3 = s_4 = s_5 = 0.13$



– Demi-périmètre maximal pour R_1, approximativement 1.2002
$\rightarrow$ borne inférieure absolue $2\sqrt{s_1} = 1.2$ atteinte lorsque le plus grand rectangle est un carré (pas possible ici)

– Somme des demi-périmètres = 4.39
$\rightarrow$ borne absolue inférieure $\sum_{i=1}^{p} 2\sqrt{s_i} \approx 4.36$ atteinte lorsque tous les rectangles sont des carrés (pas possible ici).