

**Parallélisme**

**GÉNÉRALITÉS**

COURS 8 - COMMUNICATIONS ET ROUTAGE

ERIC GOUBAULT  
COMMISSARIAT À L'ÉNERGIE ATOMIQUE & CHAIRE ECOLE  
POLYTECHNIQUE/THALÈS  
SACLAY

LE 10 MARS 2010

- Topologie statique: réseau d'interconnexion fixe:
  - anneau, tore 2D,
  - hypercube, graphe complet etc.
- Topologie dynamique: modifiée en cours d'exécution (par configuration de *switch*)

3

---

**PLAN**

- Communications et routage
  - Généralités
  - Cas de l'hypercube (diffusion simple)
  - Cas du tore 2D (multiplication de matrices)
- Algorithmique sur ressources hétérogènes
  - Allocation et équilibrage de charges statique/dynamique
  - Cas de l'algorithme LU: sur anneau (1D) puis tore (2D)

**CARACTÉRISTIQUES**

Topologie	# proc.	$d^o$	diam.	# liens
Complet	$p$	$p - 1$	1	$\frac{p(p-1)}{2}$
Anneau	$p$	2	$\lfloor \frac{p}{2} \rfloor$	$p$
Grille 2D	$\sqrt{(p)}\sqrt{(p)}$	2, 3, 4	$2(\sqrt{(p)} - 1)$	$2p - 2\sqrt{(p)}$
Tore 2D	$\sqrt{(p)}\sqrt{(p)}$	4	$2\lfloor \frac{\sqrt{(p)}}{2} \rfloor$	$2p$
Hypercube	$p = 2^d$	$d = \log(p)$	$d$	$\frac{p \log(p)}{2}$

## INTÉRÊT

- Réseau complet: idéal pour le temps de communications (toujours unitaire)
- Mais: passage à l'échelle difficile! (prix du cablage, rajouter des nouveaux processeurs?)
- Anneau: pas cher, mais communications lentes, et tolérance aux pannes des liens faible
- Tore 2D, Hypercube: bons compromis entre les deux

5

## COMMUNICATIONS DANS UN HYPERCUBE

- Chemins et routage dans un hypercube
- Broadcast dans un hypercube

## NUMÉROTATION DES SOMMETS

Un  $m$ -cube est la donnée de:

- sommets numérotés de 0 à  $2^m - 1$
- il existe une arête d'un sommet à un autre si les deux diffèrent seulement d'un bit dans leur écriture binaire

$$\begin{array}{ccc} 00 & \overset{0}{-} & 01 \\ 1 & | & 1 \\ 10 & \overset{0}{-} & 11 \end{array}$$

7

## CHEMINS ET ROUTAGE

- $A, B$  deux sommets,  $H(A, B)$  leur distance de Hamming (le nombre de bits qui diffèrent dans l'écriture)
- Il existe un chemin de longueur  $H(A, B)$  entre  $A$  et  $B$  (récurrence facile sur  $H(A, B)$ )
- Il existe  $H(A, B)!$  chemins entre  $A$  et  $B$ , dont seuls  $H(A, B)$  sont indépendants (passent par des sommets différents)

## CHEMINS ET ROUTAGE

Un routage possible: on corrige les poids faibles d'abord,

- $A = 1011$ ,  $B = 1101$
- $A \text{ xor } B = 0110$  (ou exclusif bit à bit)
- $A$  envoie donc son message sur le lien 1 (c'est à dire vers 1001) avec entête 0100
- Puis 1001, lisant l'entête, renvoie sur son lien 2, c'est à dire vers  $1101 = B$ .

Egalement algorithme dynamique qui corrige les bits selon les liens disponibles

9

## PLONGEMENTS D'ANNEAUX ET DE GRILLES

Code de Gray  $G_m$  de dimension  $m$ , défini récursivement:

$$G_m = 0G_{m-1} \mid 1G_{m-1}^{rev}$$

- $xG$  énumère les éléments de  $G$  en rajoutant  $x$  en tête de leur écriture binaire
- $G^{rev}$  énumère les éléments de  $G$  dans l'ordre renversé
- $\mid$  est la concaténation (de "listes" de mots binaires)

## CODES DE GRAY: EXEMPLE

- $G_1 = \{0, 1\}$ ,
- $G_2 = \{00, 01, 11, 10\}$ ,
- $G_3 = \{000, 001, 011, 010, 110, 111, 101, 100\}$ ,
- $G_4 = \{0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1011, 1010, 1001, 1000\}$ .

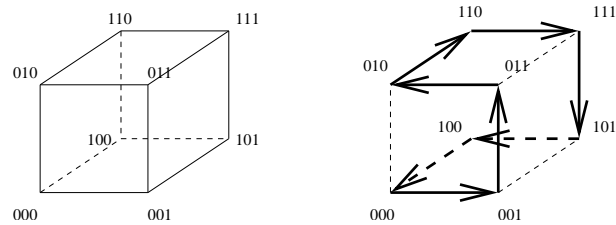
(imaginer la numérotation des processeurs sur l'anneau, dans cet ordre - un seul bit change à chaque fois)

11

## CODES DE GRAY: INTÉRÊT

- Permet de définir un anneau de  $2^m$  processeurs dans le  $m$ -cube grâce à  $G_m$
- Permet de définir un réseau torique de taille  $2^r \times 2^s$  dans un  $m$ -cube avec  $r + s = m$  (utiliser le code  $G_r \times G_s$ )

## CODES DE GRAY: EXEMPLE



13

## ARBRES COUVRANTS DE L'HYPERCUBE

- Primitives: `send(cube-link, send-adr, L)`,  
`receive(cube-link, recv-adr, L)`
- Les processeurs vont recevoir le message sur le lien correspondant à leur premier 1 (à partir des poids faibles)
- Et vont propager sur les liens qui précèdent ce premier 1
- Le processeur 0 est supposé avoir un 1 fictif en position  $m$
- Tout ceci en  $m$  phases,  $i = m - 1 \rightarrow 0$

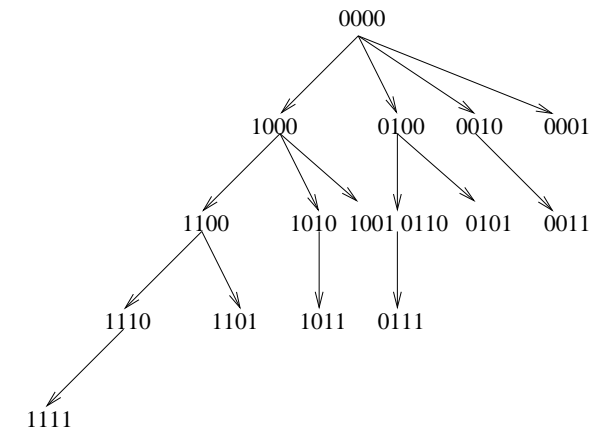
15

## DIFFUSION SIMPLE DANS L'HYPERCUBE

- on suppose que le processeur 0 veut envoyer un message à tous les autres
- algorithme naïf 1: le processeur 0 envoie à tous ses voisins, puis tous ses voisins à tous leurs voisins etc.: Très **redondant!**
- algorithme naïf 2: on utilise le code de Gray et on utilise la diffusion sur l'anneau

## ARBRES COUVRANTS DE L'HYPERCUBE

On construit à la fois un arbre couvrant de l'hypercube et l'algorithme de diffusion (ici  $m = 4$ ):



## EXEMPLE

Pour  $m = 4$ ,

- phase 3: 0000 envoie le message sur le lien 3 à 1000
- phase 2: 0000 et 1000 envoient le message sur le lien 2, à 0100 et 1100 respectivement
- ainsi de suite jusqu'à la phase 0

Améliorations possibles, voir poly

17

## ALGORITHMIQUE SUR GRILLE 2D

- Algorithme de Cannon
- Algorithme de Fox
- Algorithme de Snyder

## PRODUIT DE MATRICES SUR GRILLE 2D

- $C = C + AB$ , avec  $A$ ,  $B$  et  $C$  de taille  $N \times N$
- $p = q^2$ : on dispose d'une grille de processeurs en tore de taille  $q \times q$
- distribution des matrices par blocs:  $P_{ij}$  stocke  $A_{ij}$ ,  $B_{ij}$  et  $C_{ij}$

19

## AU DÉPART:

$$\begin{pmatrix} C_{00} & C_{01} & C_{02} & C_{03} \\ C_{10} & C_{11} & C_{12} & C_{13} \\ C_{20} & C_{21} & C_{22} & C_{23} \\ C_{30} & C_{31} & C_{32} & C_{33} \end{pmatrix} \leftarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix} \times \begin{pmatrix} B_{00} & B_{01} & B_{02} & B_{03} \\ B_{10} & B_{11} & B_{12} & B_{13} \\ B_{20} & B_{21} & B_{22} & B_{23} \\ B_{30} & B_{31} & B_{32} & B_{33} \end{pmatrix}$$

PRINCIPE DE L'ALGORITHME DE CANNON

“Preskewing”:

$$\begin{pmatrix} C_{00} & C_{01} & C_{02} & C_{03} \\ C_{10} & C_{11} & C_{12} & C_{13} \\ C_{20} & C_{21} & C_{22} & C_{23} \\ C_{30} & C_{31} & C_{32} & C_{33} \end{pmatrix} = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{11} & A_{12} & A_{13} & A_{10} \\ A_{22} & A_{23} & A_{20} & A_{21} \\ A_{33} & A_{30} & A_{31} & A_{32} \end{pmatrix} \times \begin{pmatrix} B_{00} & B_{11} & B_{22} & B_{33} \\ B_{10} & B_{21} & B_{32} & B_{03} \\ B_{20} & B_{31} & B_{02} & B_{13} \\ B_{30} & B_{01} & B_{12} & B_{23} \end{pmatrix}$$

21

PRINCIPE DE L'ALGORITHME DE CANNON (2)

Rotations sur  $A$  et  $B$ :

$$\begin{pmatrix} C_{00} & C_{01} & C_{02} & C_{03} \\ C_{10} & C_{11} & C_{12} & C_{13} \\ C_{20} & C_{21} & C_{22} & C_{23} \\ C_{30} & C_{31} & C_{32} & C_{33} \end{pmatrix} = \begin{pmatrix} A_{01} & A_{02} & A_{03} & A_{00} \\ A_{12} & A_{13} & A_{10} & A_{11} \\ A_{23} & A_{20} & A_{21} & A_{22} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix} \times \begin{pmatrix} B_{10} & B_{21} & B_{32} & B_{03} \\ B_{20} & B_{31} & B_{02} & B_{13} \\ B_{30} & B_{01} & B_{12} & B_{23} \\ B_{00} & B_{11} & B_{22} & B_{33} \end{pmatrix}$$

PRINCIPE DE L'ALGORITHME DE CANNON

```
/* diag(A) sur col 0, diag(B) sur ligne 0 */
Rotations(A,B); /* preskewing */
```

```
/* calcul du produit de matrice */
forall (k=1; k<=sqrt(P)) {
  LocalProdMat(A,B,C);
  VerticalRotation(B,upwards);
  HorizontalRotation(A,leftwards); }

```

```
/* mouvements des donnees apres les calculs */
Rotations(A,B); /* postskewing */
```

23

PRINCIPE DE L'ALGORITHME DE FOX

- Pas de mouvements de données initiales
- Diffusions horizontales des diagonales de  $A$  (décalées vers la droite)
- Rotations verticales de  $B$  (de bas en haut)

PRINCIPE DE L'ALGORITHME DE FOX

$$\begin{pmatrix} C_{00} & C_{01} & C_{02} & C_{03} \\ C_{10} & C_{11} & C_{12} & C_{13} \\ C_{20} & C_{21} & C_{22} & C_{23} \\ C_{30} & C_{31} & C_{32} & C_{33} \end{pmatrix} + = \begin{pmatrix} A_{00} & A_{00} & A_{00} & A_{00} \\ A_{11} & A_{11} & A_{11} & A_{11} \\ A_{22} & A_{22} & A_{22} & A_{22} \\ A_{33} & A_{33} & A_{33} & A_{33} \end{pmatrix} \times \begin{pmatrix} B_{00} & B_{01} & B_{02} & B_{03} \\ B_{10} & B_{11} & B_{12} & B_{13} \\ B_{20} & B_{21} & B_{22} & B_{23} \\ B_{30} & B_{31} & B_{32} & B_{33} \end{pmatrix}$$

25

PRINCIPE DE L'ALGORITHME DE FOX (2)

$$\begin{pmatrix} C_{00} & C_{01} & C_{02} & C_{03} \\ C_{10} & C_{11} & C_{12} & C_{13} \\ C_{20} & C_{21} & C_{22} & C_{23} \\ C_{30} & C_{31} & C_{32} & C_{33} \end{pmatrix} + = \begin{pmatrix} A_{01} & A_{01} & A_{01} & A_{01} \\ A_{12} & A_{12} & A_{12} & A_{12} \\ A_{23} & A_{23} & A_{23} & A_{23} \\ A_{30} & A_{30} & A_{30} & A_{30} \end{pmatrix} \times \begin{pmatrix} B_{10} & B_{11} & B_{12} & B_{13} \\ B_{20} & B_{21} & B_{22} & B_{23} \\ B_{30} & B_{31} & B_{32} & B_{33} \\ B_{00} & B_{01} & B_{02} & B_{03} \end{pmatrix}$$

PRINCIPE DE L'ALGORITHME DE FOX

`/* pas de mouvements de donnees avant les calculs */`

`/* calcul du produit de matrices */`

`broadcast(A(x,x));`

`forall (k=1; k<sqrt(P)) {`

`LocalProdMat(A,B,C);`

`VerticalRotation(B,upwards);`

`broadcast(A(k+x,k+x)); }`

`LocalProdMat(A,B,C);`

`VerticalRotation(B,upwards);`

`/* pas de mouvements de donnees apres les calculs */`

27

PRINCIPE DE L'ALGORITHME DE SNYDER

- Transposition préalable de  $B$
- Sommes globales sur les lignes de processeurs (des produits calculés à chaque étape)
- Accumulation sur les diagonales (décalées à chaque étape vers la droite) de  $C$  des résultats - représentées en gras dans les transparents ci-après
- Rotations verticales de  $B$  (de bas en haut)

PRINCIPE DE L'ALGORITHME DE SNYDER

$$\begin{pmatrix} \mathbf{C00} & C_{01} & C_{02} & C_{03} \\ C_{10} & \mathbf{C11} & C_{12} & C_{13} \\ C_{20} & C_{21} & \mathbf{C22} & C_{23} \\ C_{30} & C_{31} & C_{32} & \mathbf{C33} \end{pmatrix} + = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix} \times \begin{pmatrix} B_{00} & B_{10} & B_{20} & B_{30} \\ B_{01} & B_{11} & B_{21} & B_{31} \\ B_{02} & B_{12} & B_{22} & B_{32} \\ B_{03} & B_{13} & B_{23} & B_{33} \end{pmatrix}$$

29

PRINCIPE DE L'ALGORITHME DE SNYDER (2)

$$\begin{pmatrix} C_{00} & \mathbf{C01} & C_{02} & C_{03} \\ C_{10} & \mathbf{C11} & \mathbf{C12} & C_{13} \\ C_{20} & C_{21} & \mathbf{C22} & \mathbf{C23} \\ \mathbf{C30} & C_{31} & C_{32} & \mathbf{C33} \end{pmatrix} + = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix} \times \begin{pmatrix} B_{01} & B_{11} & B_{21} & B_{31} \\ B_{02} & B_{12} & B_{22} & B_{32} \\ B_{03} & B_{13} & B_{23} & B_{33} \\ B_{00} & B_{10} & B_{20} & B_{30} \end{pmatrix}$$

ANALYSE DES ALGORITHMES

PRINCIPE DE L'ALGORITHME DE SNYDER

```
/* mouvements des donnees avant les calculs */
Transpose(B);
/* calcul du produit de matrices */
forall (i) {
  LocalProdMat(A,B,C);
  VerticalRotation(B,upwards); }
forall (k=1;k<sqrt(P)) {
  GlobalSum(C(i,(i+k-1) mod sqrt(P)));
  LocalProdMat(A,B,C);
  VerticalRotation(B,upwards); }
GlobalSum(C(i,(i+sqrt(P)-1) mod sqrt(P)));
/* mouvements des donnees apres les calculs */
Transpose(B);
```

31

Selon les modèles (CC, SF, WH): voir poly!



## ALGORITHMIQUE HÉTÉROGÈNE

Allocation statique de tâches:

- $t_1, t_2, \dots, t_p$  temps de cycle des processeurs
- $B$  tâches identiques et indépendantes
- Principe:  $c_i \times t_i = \text{constante}$ , d'où:

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

33

## ALLOCATION STATIQUE OPTIMALE

```
Distribute(B,t1,t2,...,tn)
/* initialisation: calcule ci */
for (i=1;i<=p;i++)
  c[i]=...
/* incrementer iterativement les ci minimisant le temps */
while (sum(c[])<B) {
  find k in {1,...,p} st t[k]*(c[k]+1)=min{t[i]*(c[i]+1)};
  c[k] = c[k]+1;
return(c[]);
```

## ALGORITHME INCRÉMENTAL

- Allocation optimale pour tout nombre d'atomes entre 1 et  $B$
- Programmation dynamique avec  $t_1 = 3$ ,  $t_2 = 5$  et  $t_3 = 8$

35

## ALGORITHME INCRÉMENTAL

```
Distribute(B,t1,t2,...,tp)
/* Initialisation: aucune tache a distribuer m=0 */
for (i=1;i<=p;i++) c[i]=0;
/* construit iterativement l'allocation sigma */
for (m=1;m<=B;m++)
  find(k in {1,...,p} st t[k]*(c[k]+1)=min{t[i]*(c[i]+1)})
  c[k]=c[k]+1;
  sigma[m]=k;
return(sigma[],c[]);
```

### EXEMPLE

#atomes	$c_1$	$c_2$	$c_3$	cout	proc.sel.	alloc. $\sigma$
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	$\sigma[5] = 1$
...						
9	5	3	1	1.67	3	$\sigma[9] = 2$
10	5	3	2	1.6		$\sigma[10] = 3$

37

### RESSOURCES HÉTÉROGÈNES ET ÉQUILIBRAGE DE CHARGES

LU hétérogène:

- A chaque étape, le processeur qui possède le bloc pivot le factorise et le diffuse
- Les autres processeurs mettent à jour les colonnes restantes
- A l'étape suivante le bloc des  $b$  colonnes suivantes devient le pivot
- ainsi de suite: la taille des données passe de  $(n-1) \times b$  à  $(n-2) \times b$  etc.

### ALLOCATION STATIQUE ÉQUILIBRANT LES CHARGES 1D

- Solution 1: on redistribue les colonnes restant à traiter à chaque étape entre les processeurs: pb, coût des communications
- Solution 2: trouver une allocation statique donnant un équilibrage de charges à chaque étapes

39

### ALLOCATION STATIQUE ÉQUILIBRANT LES CHARGES 1D

- Distribution de  $B$  tâches sur  $p$  processeurs de temps de cycle  $t_1, t_2$  etc.  $t_p$  telle que
- pour tout  $i \in \{2, \dots, B\}$ , le nombre de blocs de  $\{i, \dots, B\}$  que possède chaque processeur  $P_j$  soit approximativement inversement proportionnel à  $t_j$

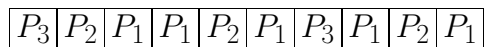
ALLOCATION 1D

- Allouer périodiquement un motif de largeur  $B$  les blocs de colonnes
- $B$  est un paramètre, par exemple si la matrice est  $(nb) \times (nb)$ ,  $B = n$  (mais meilleur pour le recouvrement calcul communication si  $B \ll n$ )
- Utiliser l'algorithme précédent en sens inverse: le bloc de colonne  $1 \leq k \leq B$  alloué sur  $\sigma(B - k + 1)$
- Cette distribution est quasi-optimale pour tout sous-ensemble  $[i, B]$  de colonnes

41

EXEMPLE

$n = B = 10, t_1 = 3, t_2 = 5, t_3 = 8$  le motif sera:



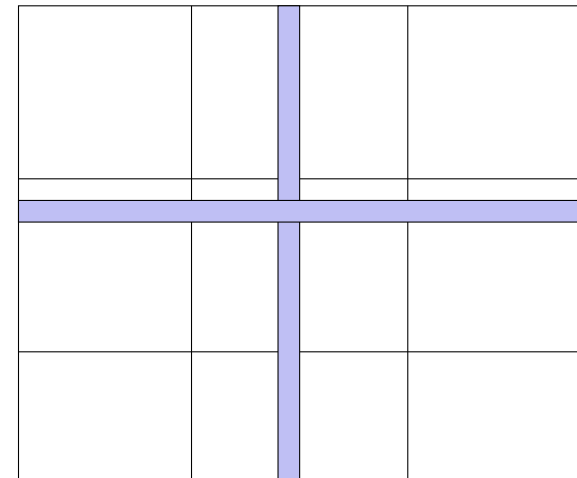
ALLOCATION STATIQUE 2D

Exemple: 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

43

MULTIPLICATION DE MATRICES - CAS HOMOGÈNE



MULTIPLICATION - CAS INHOMOGÈNE “RÉGULIER”

- 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  $t_{i,j}$

45

EQUILIBRAGE DE CHARGE PARFAIT

- Uniquement quand la matrice des temps de cycle  $T = (t_{i,j})$  est de rang 1
- Exemple, rang 2,  $P_{2,2}$  est partiellement inactif:

	1	$\frac{1}{2}$
1	$t_{11} = 1$	$t_{12} = 2$
$\frac{1}{3}$	$t_{21} = 3$	$t_{22} = 5$

- Exemple, rang 1, équilibrage parfait:

	1	$\frac{1}{2}$
1	$t_{11} = 1$	$t_{12} = 2$
$\frac{1}{3}$	$t_{21} = 3$	$t_{22} = 6$

PROBLÈME

Il faut optimiser:

- Objectif *Obj1*:

$$\min_{\sum_i r_i=1; \sum_j c_j=1} \max_{i,j} \{r_i \times t_{i,j} \times c_j\}$$

- Objectif *Obj2* (dual):

$$\max_{r_i \times t_{i,j} \times c_j \leq 1} \left\{ \left( \sum_i r_i \right) \times \left( \sum_j c_j \right) \right\}$$

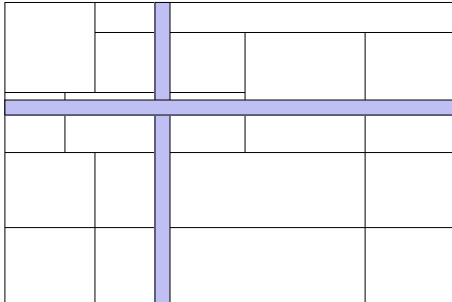
47

RÉGULARITÉ?

- En fait, 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 NP-complet
- Conclusion: l'équilibrage 2D est très difficile!

PARTITIONNEMENT LIBRE

où comment faire avec  $p$  (premier) processeurs??



49

PARTITIONNEMENT LIBRE

- $p$  processeurs de vitesses  $s_1, s_2, \dots, s_n$  de somme 1 (normalisées)
- Partitionner le carré unité en  $p$  rectangles de surfaces  $s_1, s_2, \dots, s_n$
- Surface des rectangles  $\Leftrightarrow$  vitesses relatives des processeurs
- Forme des rectangles  $\Leftrightarrow$  minimiser les communications

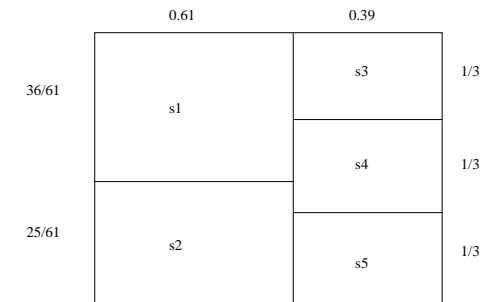
GÉOMÉTRIQUEMENT

- Partitionner le carré unité en  $p$  rectangles d'aires fixées  $s_1, s_2, \dots, s_p$  afin de minimiser
  - soit la somme des demi-périmètres des rectangles dans le cas des communications séquentielles
  - soit le plus grand des demi-périmètres des rectangles dans le cas de communications parallèles
- Problèmes NP-complets

51

EXEMPLE

- $p = 5$  rectangles  $R_1, \dots, R_5$
- Aires  $s_1 = 0.36, s_2 = 0.25, s_3 = s_4 = s_5 = 0.13$



## EXEMPLE

- Demi-périmètre maximal pour  $R_1$ , approximativement 1.2002
  - 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
  - borne absolue inférieure  $\sum_{i=1}^p 2\sqrt{s_i} = 4.36$  atteinte lorsque tous les rectangles sont des carrés