




CentraleSupélec



Mineure CalHau1


## Algorithmique distribuée

**Stéphane Vialle**




université  
PARIS-SACLAY


ÉCOLE DOCTORALE  
Sciences et technologies  
de l'information  
et de la communication (STIC)



RISEGrid  
Research Institute for Intelligent Systems and Grids




lorraine



Grand Est  
ALSACE CHAMPAGNE-ARDENNE LORRAINE

Stephane.Vialle@centralesupelec.fr  
<http://www.metz.supelec.fr/~vialle>



CentraleSupélec

## Algorithmique Distribuée

- 1. Produit de matrices sur un anneau de  $P$  processeurs**
2. Produit de matrices sur tore de  $P = q \times q$  processeurs
3. N-corps sur un anneau de  $P$  processeurs
4. Quick-sort sur hypercube de  $P = 2^d$  processeurs

CentraleSupélec

## Produit de matrices denses sur anneau

# Algorithme distribué

**Problème à résoudre :**  
 A, B, C :  $n \times n = N$  éléments

$$C = A \cdot B \quad c_{ij} = \sum_{k=1}^n (a_{ik} \cdot b_{kj}) \quad O(\text{Nbr d'opérations}) = O(N^{3/2})$$

**Comment répartir les données ?**

Pb du **partitionnement**

Calculs déterministes localisés :

- Partitionnement statique

Calculs déterministes NON localisés :

- Duplication ?
- **Circulation ?**

CentraleSupélec

## Produit de matrices denses sur anneau

# Algorithme distribué

**Partitionnement sur un anneau de processeurs :**

- A partitionnée en blocs de lignes
- B et C partitionnées en blocs de colonnes
- **Circulation de A**
- B et C statiques

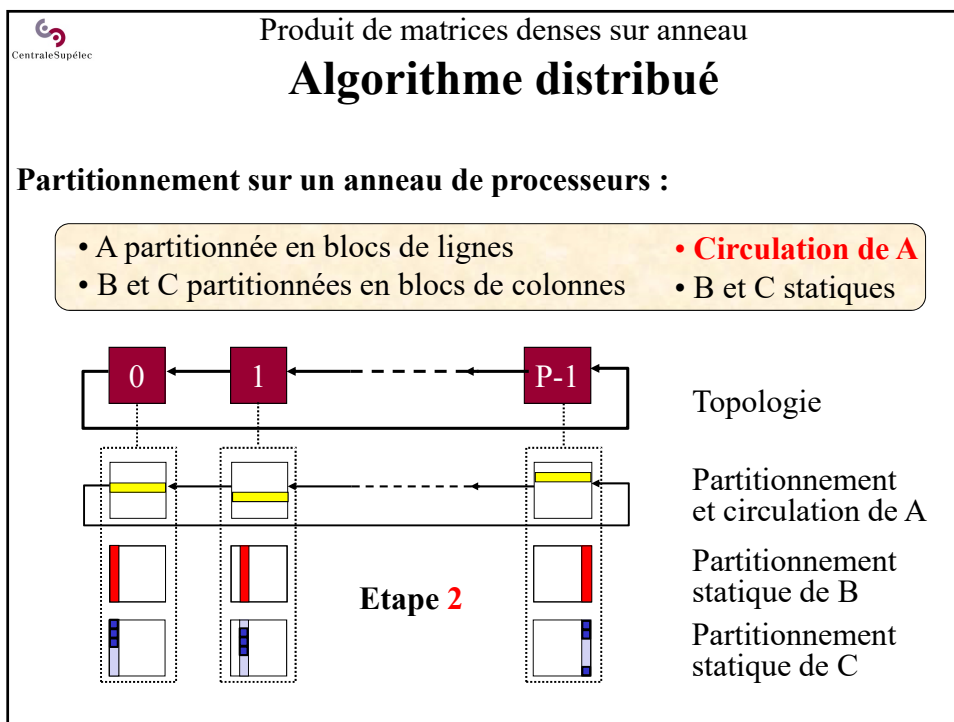
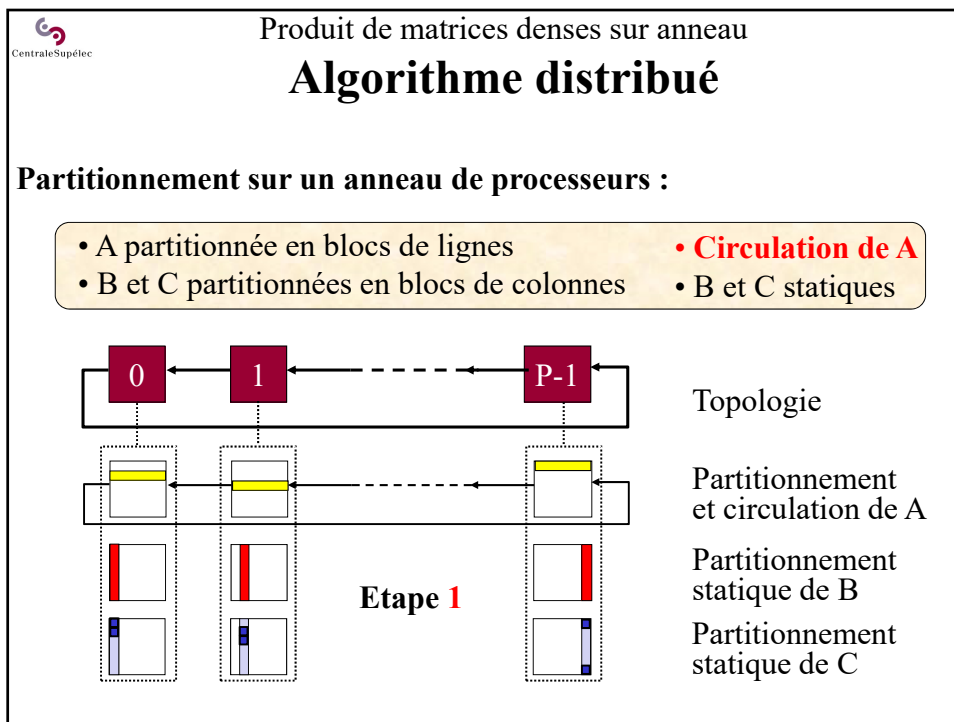
**Etape 0**  
(état initial)

Topologie

Partitionnement et circulation de A

Partitionnement statique de B

Partitionnement statique de C



CentraleSupélec

Produit de matrices denses sur anneau

## Algorithme distribué

**Partitionnement sur un anneau de processeurs :**

- A partitionnée en blocs de lignes
- B et C partitionnées en blocs de colonnes
- **Circulation de A**
- B et C statiques

**Résultats à la fin des P étapes :**

Topologie

Partitionnement statique de C

**Bilan :**

- Chaque PC a calculé un bloc de colonnes de C
- Les  $P$  PC ont travaillé en parallèle
- **Calcul de tous les blocs de colonnes en parallèle**, en  $P$  étapes

CentraleSupélec

Produit de matrices denses sur anneau

## Algorithme distribué

**Partitionnement sur un anneau de processeurs :**

- A partitionnée en blocs de lignes
- B et C partitionnées en blocs de colonnes
- **Circulation de A**
- B et C statiques

**Déroulement de l'algorithme sur PE-2, avec  $P = 4$  :**

Étape 0 C

Étape 1 C

Étape 2 C

Étape 3 C

CentraleSupélec

Produit de matrices denses sur anneau

## Algorithme distribué

Stratégies d'implantation sur un anneau de P processeurs :

<pre>// Sans recouvrement for (step=0; step&lt;P; step++)   calcul();   barrier(); // si besoin   circulation();   barrier(); // si besoin }</pre>	<pre>// Avec recouvrement for (step=0; step&lt;P; step++)   thread{ calcul(); }   thread{ circulation(); }   joinThreads();   permutBuff(); }</pre>
--	---

↓

- Concevoir l'algorithme avec des barrières de (re)synchronisation potentielles
- Selon le mécanisme de communication utilisé :
  - Implanter des barrières explicites : synchronisation forte ou bien
  - Se contenter de la synchro des comms : synchronisation relaxée

CentraleSupélec

Produit de matrices denses sur anneau

## Algorithme distribué

Stratégies d'implantation sur un anneau de P processeurs :

<pre>// Sans recouvrement for (step=0; step&lt;P; step++)   calcul();   barrier(); // si besoin   circulation();   barrier(); // si besoin }</pre>	<pre>// Avec recouvrement for (step=0; step&lt;P; step++)   thread{ calcul(); }   thread{ circulation(); }   joinThreads(); // barrier   permutBuff(); }</pre>
--	--

↓

- Concevoir l'algorithme avec une barrière de (re)synchronisation et implanter la barrière de (re)synchronisation !
- Permuter les buffers de calcul et de communication (quasi-obligatoire)

CentraleSupélec Produit de matrices denses sur anneau

## Modélisation de perfs sur anneau théorique

**Performances sans recouvrement :**

- Temps séquentiel :
 
$$T_{seq} = N \cdot (2 \cdot \sqrt{N} - 1) t_{flop}$$

$$T_{seq} \approx 2 \cdot N \cdot \sqrt{N} t_{flop}$$
- Temps parallèle :
 
$$t_{1-calc} = \frac{\sqrt{N}}{P} \cdot \frac{\sqrt{N}}{P} \cdot (2 \cdot \sqrt{N} - 1) t_{flop}$$

$$t_{1-calc} \approx 2 \cdot \frac{N \cdot \sqrt{N}}{P^2} t_{flop}$$

**Modèle de comm :**

- $t_{com}(q) = t_s + q \cdot t_w$
- toutes les comms en parallèles

- Temps parallèle (circulation) :
 
$$t_{1-circ} = t_s + \sqrt{N} \cdot \frac{\sqrt{N}}{P} t_w$$

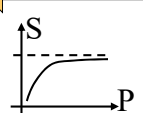
$$t_{1-circ} \approx \frac{N}{P} t_w$$

Speed up :

$$S^{no} = \frac{T_{seq}}{T_{par}^{no}} \approx P \cdot \frac{1}{1 + \frac{P t_w}{2 \cdot \sqrt{N} t_{flop}}}$$

$$T_{par}^{no} \approx P \cdot (t_{1-calc} + t_{1-circ})$$

$$T_{par}^{no} \approx 2 \cdot \frac{N \cdot \sqrt{N}}{P} t_{flop} + N t_w$$



CentraleSupélec Produit de matrices denses sur anneau

## Modélisation de perfs sur anneau théorique

**Comparaison calculs-communications (sans recouvrement) :**

- Calculs :
 
$$T_{par}^{calc} \approx 2 \cdot \frac{N^{3/2}}{P} t_{flop}$$

$$P \text{ fixé} \Rightarrow O(T_{par}^{calc}) = O(N^{3/2})$$
- Circulation :
 
$$T_{par}^{circ} \approx N t_w$$

$$O(T_{par}^{circ}) = O(N)$$

$O(T_{par}^{calc}) > O(T_{par}^{circ})$

Avec  $P$  fixé, quand  $N \uparrow$  : les calculs deviennent prépondérants  
 → le surcoût des comms devient négligeable

Et le speedup devient parfait :

$$S^{no} \approx P \cdot \frac{1}{1 + \frac{P \cdot t_w}{2 \cdot \sqrt{N} \cdot t_{flop}}} \xrightarrow{N \rightarrow \infty} P$$

« Bon » problème pour un TP !

CentraleSupélec  
Produit de matrices denses sur anneau

## Modélisation de perfs sur anneau théorique

**Performances avec recouvrement :**

- Temps séquentiel :
 
$$T_{seq} = N \cdot (2 \cdot \sqrt{N} - 1) t_{flop}$$

$$T_{seq} \approx 2 \cdot N \cdot \sqrt{N} t_{flop}$$
- Temps parallèle :
 
$$t_{1-calc} = \frac{\sqrt{N}}{P} \cdot \frac{\sqrt{N}}{P} \cdot (2 \cdot \sqrt{N} - 1) t_{flop}$$

$$t_{1-calc} \approx 2 \cdot \frac{N \cdot \sqrt{N}}{P^2} t_{flop}$$

$$t_{1-circ} = t_s + \sqrt{N} \cdot \frac{\sqrt{N}}{P} t_w$$

$$t_{1-circ} \approx \frac{N}{P} t_w$$

$T_{par}^{ov} \approx P \cdot \max(t_{1-calc}, t_{1-circ}) \cdot \alpha$

- A  $P$  fixé :
 
$$\exists N_0 / N > N_0 \Rightarrow t_{1-calc} > t_{1-circ}$$

$T_{par}^{ov} \approx P t_{1-calc} \cdot \alpha$   
 $T_{par}^{ov} \approx 2 \cdot \alpha \cdot \frac{N \cdot \sqrt{N}}{P} t_{flop}$

CentraleSupélec  
Produit de matrices denses sur anneau

## Modélisation de perfs sur anneau théorique

**Performances avec recouvrement :**

- Temps séquentiel :
 
$$T_{seq} = N \cdot (2 \cdot \sqrt{N} - 1) t_{flop}$$

$$T_{seq} \approx 2 \cdot N \cdot \sqrt{N} t_{flop}$$
- Temps parallèle :
 
$$\exists N_0 / N > N_0 \Rightarrow$$

$$T_{par}^{ov} \approx 2 \cdot \alpha \cdot \frac{N \cdot \sqrt{N}}{P} t_{flop}$$

• Speed up :

$$S_{ov} = \frac{T_{seq}}{T_{par}^{ov}} \approx P \cdot \frac{1}{\alpha}$$

Remarque:

$$\alpha = \alpha(N, P) \xrightarrow{N \rightarrow \infty} 1$$

encore:  $S(P) \xrightarrow{N \rightarrow \infty} P$

Rappel :

$$S^{no} = P \cdot \frac{1}{1 + \frac{P t_w}{2 \cdot \sqrt{N} t_{flop}}}$$

→ **Si le recouvrement est techniquement possible,**  
alors son exploitation doit améliorer les perfs.

## Algorithmique Distribuée

1. Produit de matrices sur un anneau de  $P$  processeurs
2. **Produit de matrices sur tore de  $P = q \times q$  processeurs**
3. N-corps sur un anneau de  $P$  processeurs
4. Quick-sort sur hypercube de  $P = 2^d$  processeurs

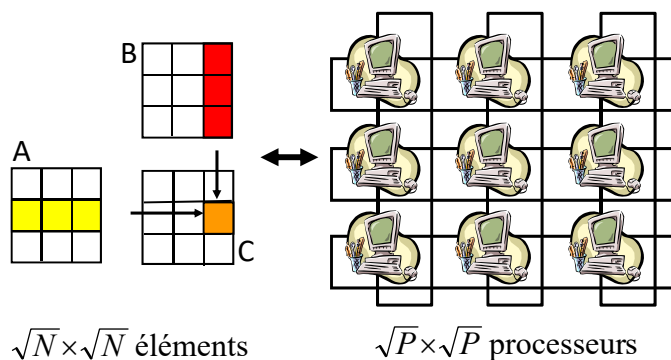
## Produit de matrices sur tore de $P$ proc Algorithme distribué

$A, B, C : n \times n = N$  éléments

$C = A \cdot B$

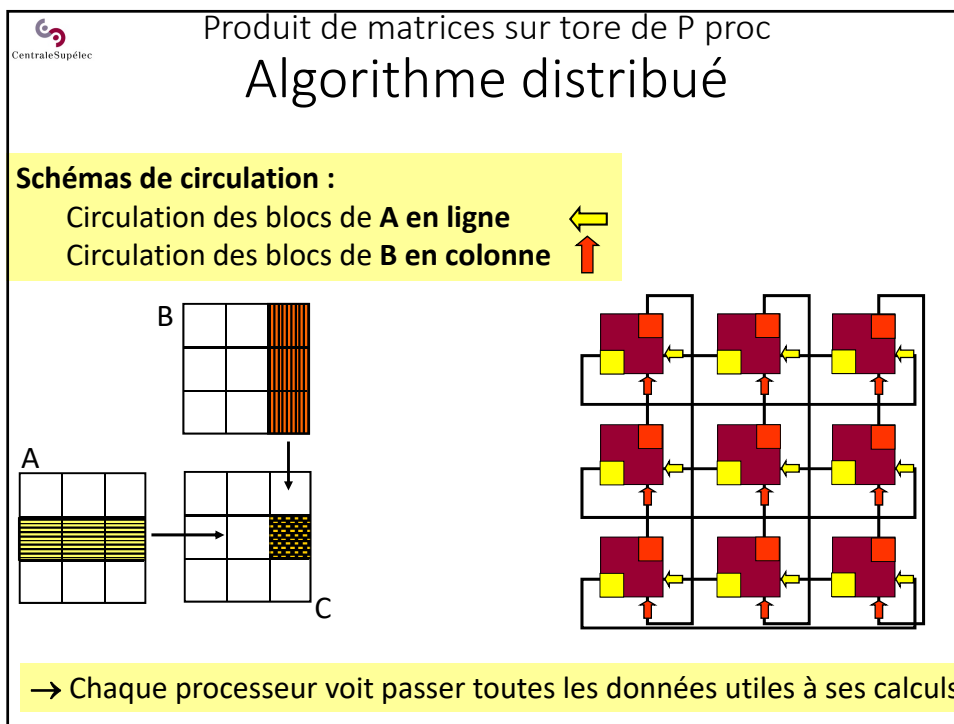
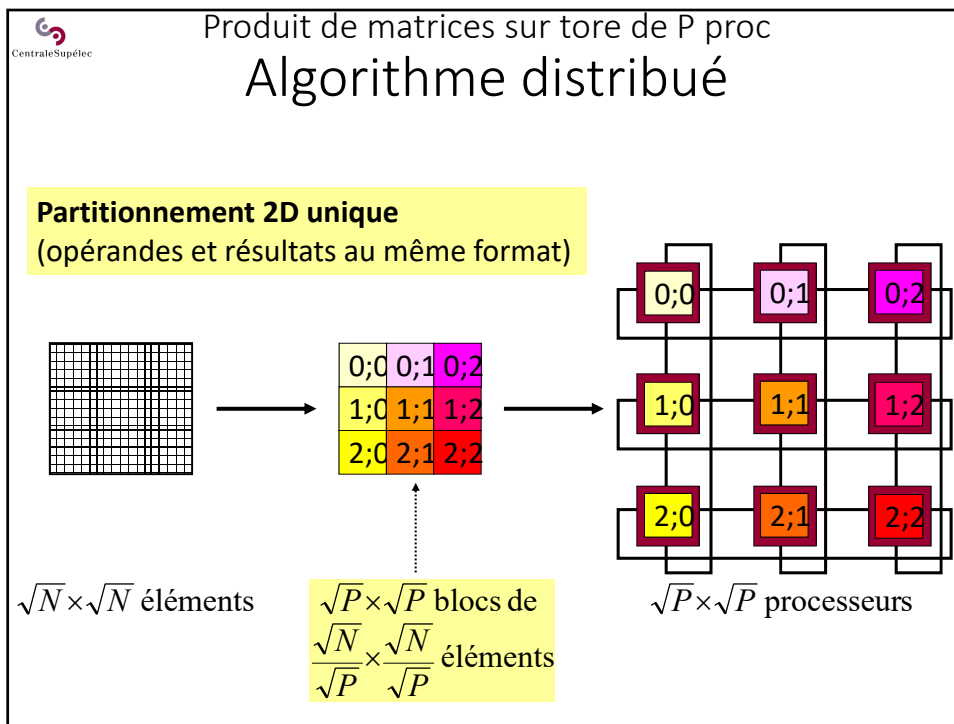
$$c_{ij} = \sum_{k=1}^n (a_{ik} \cdot b_{kj})$$

$O(\text{Nbr d'opérations}) = O(N^{3/2})$



Quels  
partitionnement  
et circulation ?





CentraleSupélec

## Produit de matrices sur tore de P proc Algorithme distribué

**Problème de séquençement :**  
 besoin **simultané** des blocs  $A_{i,k}$  et  $B_{k,j}$

$C = A \cdot B$       $c_{ij} = \sum_{k=1}^n (a_{ik} \cdot b_{kj})$

Seuls les processeurs de la première diagonale sont satisfaits !

→ Modifier le partitionnement initial

CentraleSupélec

## Produit de matrices sur tore de P proc Algorithme distribué

**Démarche :**

- On conserve le principe du partitionnement en 2D
- On conserve le schéma de circulation (A en ligne, B en colonne)
- Mais on cherche un partitionnement initial adapté ...
- ... on place les données de PE(0;0), et on *propage les contraintes*

CentraleSupélec

## Produit de matrices sur tore de P proc Algorithme distribué

Les schémas de circulation imposent :

- les blocs de A sur la première ligne de processeurs
- les blocs de B sur la première colonne de processeurs

Circulation en colonne vers le haut

Circulation en ligne vers la gauche

CentraleSupélec

## Produit de matrices sur tore de P proc Algorithme distribué

L'algorithme de produit matriciel impose :

- les blocs de B sur la première ligne de processeurs
- les blocs de A sur la première colonne de processeurs

Blocs compatibles  $\left( A_{i,k} \text{ et } B_{k,j} \right)$

Blocs compatibles  $\left( A_{i,k} \text{ et } B_{k,j} \right)$

CentraleSupélec

## Produit de matrices sur tore de P proc Algorithme distribué

Les schémas de circulation imposent :

- les blocs de A sur les lignes de processeurs restantes
- les blocs de B sur les colonnes de processeurs restantes

On vérifie si les nouveaux couples sont compatibles...

... Oui! On a toujours :

$$\left( A_{i,k} \text{ et } B_{k,j} \right)$$

Blocs compatibles

CentraleSupélec

## Produit de matrices sur tore de P proc Algorithme distribué

**Finalement :**

- Partitionnement fixé sur tous les processeurs
- Tous les couples de blocs sont compatibles à la première étape
- Tous les couples de blocs sont compatibles après chaque circulation

Colonne 0 décalée de 0 cran vers le haut  
Colonne 1 décalée de 1 cran vers le haut  
Colonne 2 décalée de 2 cran vers le haut

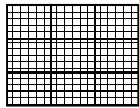
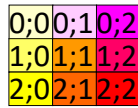
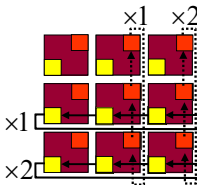
Généralisation ...

Puis Calcul complet en  $\sqrt{P}$  étapes

CentraleSupélec

## Produit de matrices sur tore de P proc Algorithme distribué

**Algorithme de Canon :**  
Produit de matrices de  $\sqrt{N} \times \sqrt{N}$  éléments sur un tore de  $\sqrt{P} \times \sqrt{P}$  procs.

- 1 – Partitionnement des matrices en  $\sqrt{P} \times \sqrt{P}$  blocs de  $\frac{\sqrt{N}}{\sqrt{P}} \times \frac{\sqrt{N}}{\sqrt{P}}$  éléments
 

- 2 – Pré-décalages initiaux des blocs sur le tore de proc :
  - ligne  $i$  ( $[0; \sqrt{P}-1]$ ) : décalage de  $i$  crans  $\leftarrow$ ,
  - colonne  $j$  ( $[0; \sqrt{P}-1]$ ) : décalage de  $j$  crans  $\uparrow$ .
- 3 -  $\sqrt{P}$  étapes :  $\sqrt{P} \times \{$  calculs : somme partielle de  $C(i,j)$  ;  
circulations :  $A(i,j)$  1 cran  $\leftarrow$ , et  $B(i,j)$  1 cran  $\uparrow$  ;  $\}$
- 4 – Dernière circulation ou post décalages de A et B si nécessaire.

CentraleSupélec

## Produit de matrices sur tore de P proc Algorithme distribué

**Pseudo-Code de l'algorithme de Canon sur chaque processeur :**

```

main ()
{
  partitionnement ();
  predecalage ();
  barriere ();
  for (step=0; step < SQRT_P-1; step++) {
    multiplication_locale ();
    circulation_ligne_colone ();
    barriere ();
  }
  multiplication_locale ();
  postdecalage ();
}

```

Post-décalage :

- nécessaire si on souhaite réutiliser les matrices pour des opérations variées (+, x, ...)

CentraleSupélec

Produit de matrices sur tore de P proc

## Modélisation des performances

**1 - Performances avec messages bloquants et sans recouvrement :**

*pré-décalages;  $\sqrt{P}$  étapes de calcul et  $\sqrt{P}-1$  circulations; postdécalage*

Hyp : Toutes les communications peuvent se faire en parallèle  
Modélisation en «  $t_s + Q \cdot t_w$  »

- Pré et Post décalages :  $t_{pre-decal} \approx 2 \cdot (\sqrt{P}-1) \cdot \frac{N}{P} t_w$   
- dans un vrai tore unidirectionnel :  $t_{post-decal} \approx 2 \cdot (\sqrt{P}-1) \cdot \frac{N}{P} t_w$
- dans un cluster :  $t_{pre-decal} \approx 2 \cdot \frac{N}{P} t_w$   
 $t_{post-decal} \approx 2 \cdot \frac{N}{P} t_w$

CentraleSupélec

Produit de matrices sur tore de P proc

## Modélisation des performances

**1 - Performances avec messages bloquants et sans recouvrement :**

*pré-décalages;  $\sqrt{P}$  étapes de calcul et  $\sqrt{P}-1$  circulations; postdécalage*

- Pré et post décalage :  $t_{pre-decal} \approx 2 \cdot \frac{N}{P} t_w$  et  $t_{post-decal} \approx 2 \cdot \frac{N}{P} t_w$
- Circulation :  $t_{1-circ} \approx 2 \cdot \frac{N}{P} t_w$
- Calcul :  $t_{1-calc} \approx 2 \cdot \frac{N^{3/2}}{P^{3/2}} t_{flop}$

Donc sur un cluster, avec des msgs bloquants :  $T_{par}(P) \approx 2 \cdot \frac{N^{3/2}}{P} t_{flop} + 2 \cdot (\sqrt{P}+1) \frac{N}{P} t_w$

CentraleSupélec

Produit de matrices sur tore de P proc

## Modélisation des performances

**1 - Performances avec messages bloquants et sans recouvrement :**

*pré-décalages;  $\sqrt{P}$  étapes de calcul et  $\sqrt{P}-1$  circulations; postdécalage*

- Temps parallèle sur cluster :  $T_{par}(P) \approx 2 \cdot \frac{N^{3/2}}{P} t_{flop} + 2(\sqrt{P}+1) \cdot \frac{N}{P} t_w$
- Temps séquentiel :  $T_{seq} \approx 2 \cdot N^{3/2} t_{flop}$
- Speed up sur cluster avec msgs bloquants :  $S(P) \approx P \cdot \frac{1}{1 + (\sqrt{P}+1) t_w / \sqrt{N} t_{flop}}$

Donc :  $S(P) \xrightarrow[N \rightarrow \infty]{} P$       On retrouve l'asymptote idéale

CentraleSupélec

Produit de matrices sur tore de P proc

## Modélisation des performances

**2 - Performances avec msgs **non** bloquants et sans recouvrement :**

*pré-décalages;  $\sqrt{P}$  étapes de calcul et  $\sqrt{P}-1$  circulations; postdécalage*

Hyp : Toutes les communications peuvent se faire en parallèle  
Modélisation en «  $t_s + Q \cdot t_w$  »

- Pré et Post décalages :
  - dans un vrai tore unidirectionnel :
 
$$t_{pre-decal} \approx (\sqrt{P}-1) \cdot \frac{N}{P} t_w$$

$$t_{post-decal} \approx (\sqrt{P}-1) \cdot \frac{N}{P} t_w$$
  - dans un cluster :
 
$$t_{pre-decal} \approx \frac{N}{P} t_w$$

$$t_{post-decal} \approx \frac{N}{P} t_w$$

CentraleSupélec

Produit de matrices sur tore de P proc

## Modélisation des performances

**2 - Performances avec msgs **non** bloquants et sans recouvrement :**

*pré-décalages;  $\sqrt{P}$  étapes de calcul et  $\sqrt{P}-1$  circulations; postdécalage*

- Pré et post décalage :  $t_{pre-decal} \approx \frac{N}{P} t_w$  et :  $t_{post-decal} \approx \frac{N}{P} t_w$
- Circulation :  $t_{1-circ} \approx \frac{N}{P} t_w$
- Calcul :  $t_{1-calc} \approx 2 \cdot \frac{N^{3/2}}{P^{3/2}} t_{flop}$

Donc sur un cluster, avec des msgs non-bloquants :

$$T_{par}(P) \approx 2 \cdot \frac{N^{3/2}}{P} t_{flop} + (\sqrt{P} + 1) \frac{N}{P} t_w$$

CentraleSupélec

Produit de matrices sur tore de P proc

## Modélisation des performances

**2 - Performances avec msgs **non** bloquants et sans recouvrement :**

*pré-décalages;  $\sqrt{P}$  étapes de calcul et  $\sqrt{P}-1$  circulations; postdécalage*

- Temps parallèle sur cluster :  $T_{par}(P) \approx 2 \cdot \frac{N^{3/2}}{P} t_{flop} + (\sqrt{P} + 1) \frac{N}{P} t_w$
- Temps séquentiel :  $T_{seq} \approx 2 \cdot N^{3/2} t_{flop}$
- Speed up sur cluster avec msgs non-bloquants :  $S(P) \approx P \cdot \frac{1}{1 + (\sqrt{P} + 1) t_w / (2 \cdot \sqrt{N} t_{flop})}$

Donc :  $S(P) \xrightarrow{N \rightarrow \infty} P$  On retrouve l'asymptote idéale



CentraleSupélec

Produit de matrices sur tore de P proc

## Modélisation des performances

**3 - Performances avec msgs non bloquants et avec recouvrement :**

*pré-déc;  $\sqrt{P}-1$  calculs et  $\sqrt{P}-1$  circulations; 1 calcul et postdec*

Hyp : Toutes les communications peuvent se faire en parallèle  
Modélisation en «  $t_s + Q \cdot t_w$  »

- Pré et Post décalages :
 
$$t_{pre-decal} \approx (\sqrt{P}-1) \cdot \frac{N}{P} t_w$$
 - dans un vrai tore unidirectionnel :
 
$$t_{post-decal} \approx (\sqrt{P}-1) \cdot \frac{N}{P} t_w$$
- dans un cluster :
 
$$t_{pre-decal} \approx \frac{N}{P} t_w$$

$$t_{post-decal} \approx \frac{N}{P} t_w$$

CentraleSupélec

Produit de matrices sur tore de P proc

## Modélisation des performances

**3 - Performances avec msgs non bloquants et avec recouvrement :**

*pré-déc;  $\sqrt{P}-1$  calculs et  $\sqrt{P}-1$  circulations; 1 calcul et postdec*

- Pré et post décalage :  $t_{pre-decal} \approx \frac{N}{P} t_w$  et :  $t_{post-decal} \approx \frac{N}{P} t_w$
- Circulation :  $t_{1-circ} \approx \frac{N}{P} t_w$
- Calcul :  $t_{1-calc} \approx 2 \cdot \frac{N^{3/2}}{P^{3/2}} t_{flop}$

$$T_{par}(P) \approx t_{pre-dec} + (\sqrt{P}-1) \alpha \cdot \max(t_{1-calc}; t_{1-circ}) + \alpha \cdot \max(t_{1-calc}; t_{post-dec})$$

$$T_{par}(P) \approx \frac{N}{P} t_w + \sqrt{P} \alpha \cdot \max\left(2 \cdot \frac{N^{3/2}}{P^{3/2}} t_{flop}; \frac{N}{P} t_w\right)$$

CentraleSupélec

## Produit de matrices sur tore de P proc Modélisation des performances

**3 - Performances avec msgs non bloquants et avec recouvrement :**

*pré-déc;  $\sqrt{P}-1$  calculs et  $\sqrt{P}-1$  circulations; 1 calcul et postdec*

Rappel : Calculs prépondérants, donc :

- Temps parallèle sur cluster :  $\exists N_0 / N > N_0 \Rightarrow T_{par} \approx 2 \cdot \alpha \cdot \frac{N^{3/2}}{P} t_{flop} + \frac{N}{P} t_w$
- Temps séquentiel :  $T_{seq} \approx 2 \cdot N^{3/2} t_{flop}$
- Speed up sur cluster avec msgs non-bloquants et recouvrement :  $\exists N_0 / N > N_0 \Rightarrow S(P) \approx P \cdot \frac{1}{\alpha + t_w / (2 \sqrt{N} t_{flop})}$

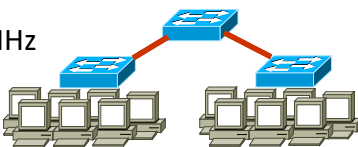
Hyp:  $\alpha = \alpha(N, P) \xrightarrow{N \rightarrow \infty} 1$     Donc:  $S(P) \xrightarrow{N \rightarrow \infty} P$     **On retrouve l'asymptote idéale**

CentraleSupélec

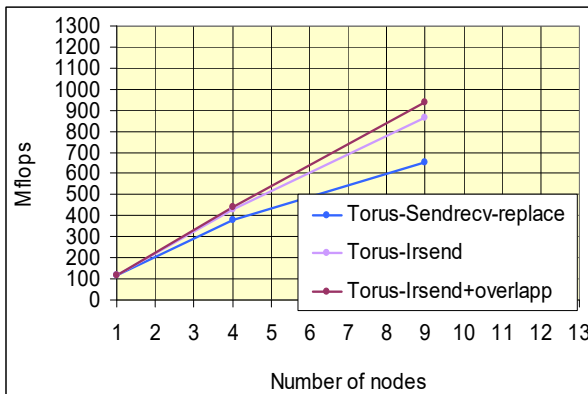
## Produit de matrices sur tore de P proc Modélisation des performances

**Expérimentation du produit de matrices en tore sur cluster de PC**

- Cluster « Beowulf »
- 3 implants. en MPI (voir chap MPI)
- 12 x P3 - 700MHz
- **Fast Ethernet**
- 3 switches



- Performances correctes
- L'envoi de messages non-bloquant est efficace.
- Le recouvrement est encore plus efficace, il gomme le passage à 3 switches.



Number of nodes	Torus-Sendrecv-replace (Mflops)	Torus-Irsend (Mflops)	Torus-Irsend+overlapp (Mflops)
1	100	100	100
2	200	200	200
3	300	300	300
4	400	400	400
5	450	500	500
6	500	600	600
7	550	700	700
8	600	800	800
9	650	900	900

CentraleSupélec

## Produit de matrices sur tore de P proc Modélisation des performances

**Comparaison Anneau/Tore pour le produit de matrices :**

	Anneau	Tore
Bloquant	$T_{par}(P) \approx 2 \cdot \frac{N^{3/2}}{P} t_{flop} + N t_w$	$T_{par}(P) \approx 2 \cdot \frac{N^{3/2}}{P} t_{flop} + 2(\sqrt{P} + 1) \frac{N}{P} t_w$
Non-bloquant Recouvrement	$T_{par} \approx 2 \alpha \cdot \frac{N^{3/2}}{P} t_{flop}$	$T_{par} \approx 2 \alpha \cdot \frac{N^{3/2}}{P} t_{flop} + \frac{N}{P} t_w$

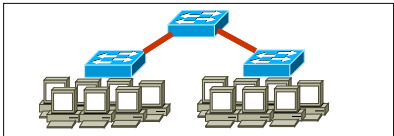
**Bilan :**

- bloquant → P = 4 : tore plus lent, P = 9 : tore plus rapide  
Le tore est plus rapide pour P > 7
- non bloquant → L'anneau est toujours plus rapide !  
Même temps de boucle calcul-circulation, mais prédécalage à faire en plus sur le tore!

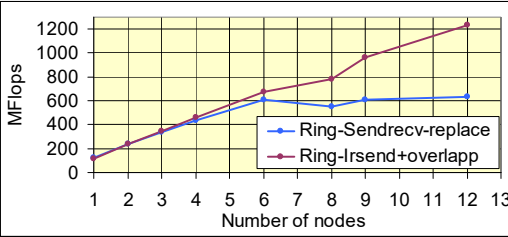
CentraleSupélec

## Produit de matrices sur tore de P proc Modélisation des performances

**Tests sur cluster « Beowulf » :**

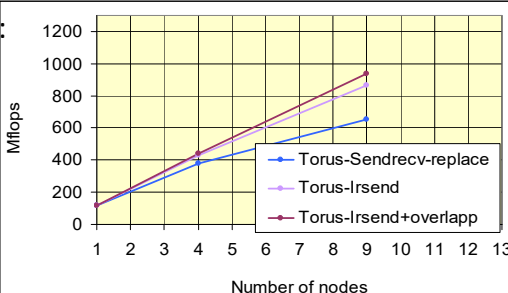


12 x P3 - 700MHz  
Fast Ethernet & 3 switches



Number of nodes	Ring-Sendrecv-replace (MFlops)	Ring-Irsend+overlapp (MFlops)
1	100	100
2	200	200
3	300	300
4	400	400
6	500	600
8	550	700
9	550	800
12	550	1000
13	550	1100

- non-bloquant + recouvrement : (presque) pas de différence → normal !
- bloquant (sans recouvrement) :  
P= 4 : tore plus lent  
P= 9 : tore plus rapide  
→ normal !



Number of nodes	Torus-Sendrecv-replace (MFlops)	Torus-Irsend (MFlops)	Torus-Irsend+overlapp (MFlops)
1	100	100	100
2	200	200	200
3	300	300	300
4	400	400	400
6	500	500	500
8	550	600	600
9	550	700	800
12	550	800	1000
13	550	800	1000

## Algorithmique Distribuée

1. Produit de matrices sur un anneau de  $P$  processeurs
2. Produit de matrices sur tore de  $P = q \times q$  processeurs
- 3. N-corps sur un anneau de  $P$  processeurs**
4. Quick-sort sur hypercube de  $P = 2^d$  processeurs

Problème des N-Corps sur anneau de processeurs

## Définition

### Calcul des trajectoires de $N$ corps en attraction mutuelle:

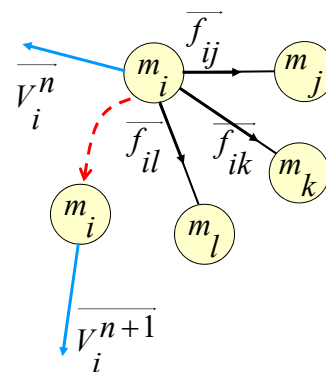
- $N$  molécules en interaction électrostatique
- $N$  planètes en interaction gravitationnelle
- ...

À  $t_n$  :  $N$  triplets  $(\vec{P}os_i^n, \vec{V}_i^n, \vec{a}_i^n)$

$$\text{Avec: } \vec{a}_i^n = \sum_{j \neq i} \left( \frac{G \cdot m_j}{(d_{ij}^n)^2} \cdot \frac{\vec{P}os_j^n - \vec{P}os_i^n}{d_{ij}^n} \right)$$

$$\text{D'où: } \vec{V}_i^{n+1} = \vec{V}_i^n + \vec{a}_i^n \cdot \delta t$$

$$\vec{P}os_i^{n+1} = \vec{P}os_i^n + \vec{V}_i^n \cdot \delta t + \vec{a}_i^n \cdot (\delta t)^2$$



CentraleSupélec

## Problème des N-Corps sur anneau de processeurs

### Définition

**Dans sa forme initiale :**

- Tous les corps interagissent avec tous les autres  
→ *Long range data interaction*  
→  $O(N^2)$  : plus complexe qu'un produit de matrices ( $O(N^{3/2})$ )

**Dans sa forme usuelle :**

- On ignore les influences trop lointaines (trop faibles) : on se limite à un *rayon d'influence*  
→ perte de précision, mais :  $\alpha \cdot N^2 \cdot t_{1\text{-influence}} \rightarrow \alpha \cdot n^2 \cdot t_{1\text{-influence}}$   
avec :  $n \ll N$

**Dans tous les cas :**

Le problème des N-corps reste un problème très lourd ( $O(N^2)$ ) qui sature très vite les ordinateurs → besoin de parallélisation

CentraleSupélec

## Problème des N-Corps sur anneau de processeurs

### Algorithme distribué

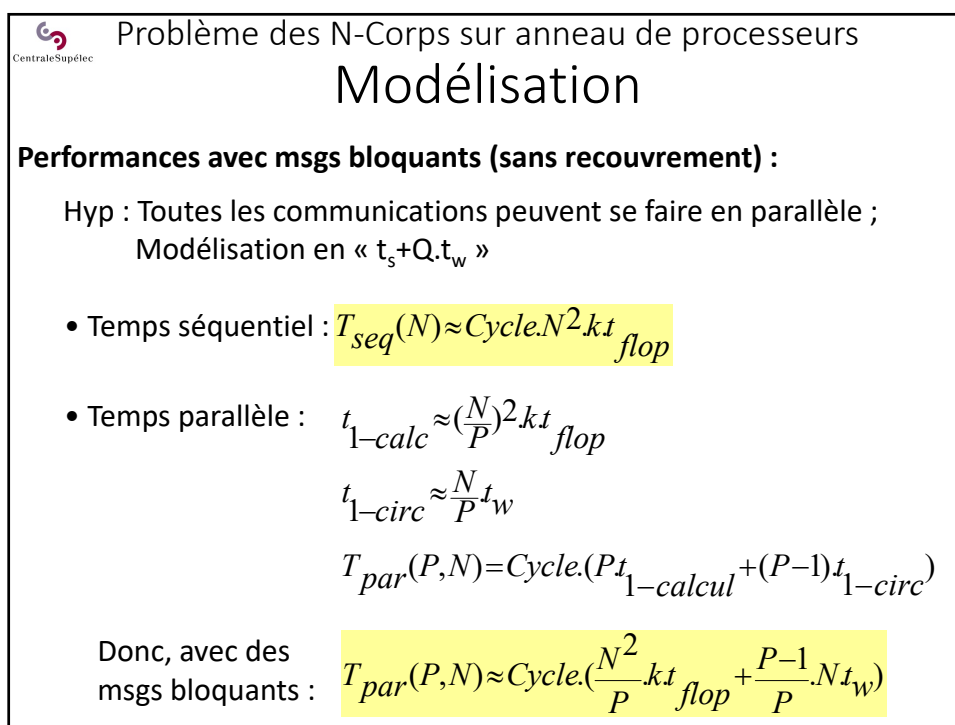
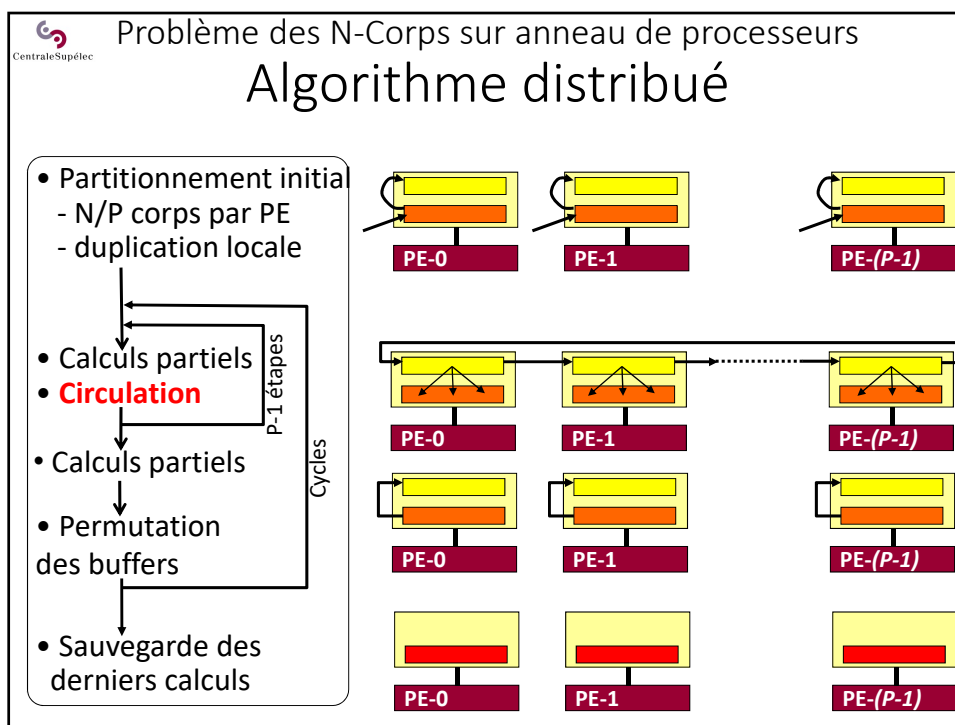
**Principe de parallélisation :**


- Chaque processeur « gère » l'évolution de N/P corps.
- On fait circuler les coordonnées courante des N corps sur l'anneau.
- A chaque étape : chaque processeur accumule l'influence des N/P corps dont il reçoit les coordonnées sur ses N/P corps.
- Après P étapes, chaque processeur calcule les nouvelles coordonnées de ses N/P corps.

**Réalisation :** On utilise une **partition fixe** et une **partition circulante**

Coord. courantes des N corps :  
partition circulante par bloc de N/P

Influences subies par les N/P corps :  
partition fixe par bloc de N/P




CentraleSupélec  Problème des N-Corps sur anneau de processeurs

## Modélisation

**Performances avec msgs bloquants (sans recouvrement) :**

- Accélération :
 
$$S(P,N) \approx P \cdot \frac{1}{1 + \frac{(P-1)t_w}{N.k.t_{flop}}}$$
 Donc :  $S(P) \xrightarrow{N \rightarrow \infty} P$   
 On retrouve l'asymptote idéale
- Rapport Calculs/Communications :
 
$$\frac{\text{Calculs}}{\text{Comms}} = \frac{O(N^2)}{O(N)} = O(N)$$

Très bon rapport ! Encore meilleur que celui du produit de matrices  
 En augmentant N les performances finiront toujours par être bonnes !

CentraleSupélec  Problème des N-Corps sur anneau de processeurs

## Modélisation

**Performances avec msgs non-bloquants et avec recouvrement :**

Hyp : Toutes les communications peuvent se faire en parallèle ;  
 Modélisation en «  $t_s + Q.t_w$  »

- Temps séquentiel :  $T_{seq}(N) \approx \text{Cycle} \cdot N^2 \cdot k.t_{flop}$
- Temps parallèle :
 
$$T_{par}(P,N) = \text{Cycle} \left( \alpha \cdot (P-1) \cdot \text{Max}(t_{1-calc}, t_{1-circ}) + t_{1-calc} \right)$$

$$\exists N_0 / N > N_0 \Rightarrow T_{par}(P,N) \approx \text{Cycle} \cdot (\alpha \cdot (P-1) + 1) \cdot \frac{N^2}{P^2} \cdot k.t_{flop}$$

Donc, avec des msgs non-bloquants et du recouvrement :  $\exists N_0 / N > N_0 \Rightarrow T_{par}(P,N) \approx \text{Cycle} \cdot \alpha \cdot \frac{N^2}{P} \cdot k.t_{flop}$

CentraleSupélec

Problème des N-Corps sur anneau de processeurs

## Modélisation

**Performances avec msgs non-bloquants et avec recouvrement :**

- Accélération :

$$S(P,N) \approx \frac{\text{Cycle} \cdot N^2 \cdot k \cdot t \text{ flop}}{\text{Cycle} \cdot \frac{N^2}{P} \cdot k \cdot t \text{ flop}}$$

$$S(P,N) \approx P \cdot \frac{1}{\alpha}$$

Hyp:  $\alpha = \alpha(N,P) \xrightarrow{N \rightarrow \infty} 1$       Donc:  $S(P) \xrightarrow{N \rightarrow \infty} P$

On retrouve (encore) l'asymptote idéale

CentraleSupélec

## Algorithmique Distribuée

1. Produit de matrices sur un anneau de P processeurs
2. Produit de matrices sur tore de  $P = q \times q$  processeurs
3. N-corps sur un anneau de P processeurs
4. **Quick-sort sur hypercube de  $P = 2^d$  processeurs**



CentraleSupélec

## Quick-sort sur hypercube de processeurs

### Algorithme séquentiel

**Principe :**

- Tri récursif
- Diviser-pour-régner
- Séparation des données selon des valeurs *pivots*

**Performances :**

pire cas :  $\left(\frac{N \cdot (N-1)}{2}\right) t_{comp} \rightarrow O(N^2)$

moyen cas (Knuth) :  $\rightarrow O(2 \cdot N \cdot \log(N))$

meilleur cas :  $\left(N \cdot \log_2(N) - 2 \cdot N + 1\right) t_{comp} \rightarrow O(N \cdot \log(N))$

CentraleSupélec

## Quick-sort sur hypercube de processeurs

### Algorithme séquentiel

**Implantation séquentielle :**

```
void QuickSort(double *A, int q, int r)
{
    int s, i;
    double pivot;

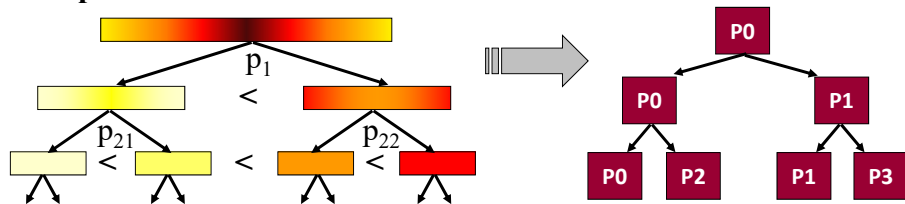
    if (q < r) {
        /* Partitionnement */
        pivot = A[q]; /* Choix du pivot ... */
        s = q;
        for (i = q+1; i <= r; i++) {
            if (A[i] <= pivot) {
                s = s+1;
                exchange(A, s, i);
            }
        }
        exchange(A, q, s);
        QuickSort(A, q, s-1); /* Appels recursifs */
        QuickSort(A, s+1, r);
    }
}
```

CentraleSupélec

## Quick-sort sur hypercube de processeurs

### Parallélisation naïve

**Principe :**



On crée un nouveau processus et on utilise un nouveau processeur à chaque appel récursif (en réutilisant les anciens).

**Bilan :**

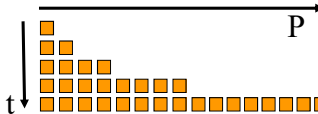
- Simple, facile à implanter en mémoire partagée
- **Mais inefficace !**

CentraleSupélec

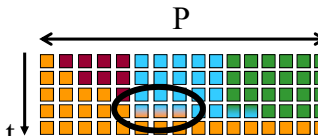
## Quick-sort sur hypercube de processeurs

### Parallélisation naïve

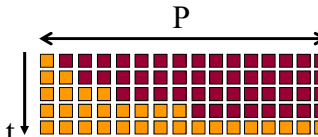
**1 - Faiblesse conceptuelle :**  
 Algorithme pas pleinement parallèle  
 → Trouver plus parallèle !



**2 - Faiblesse de mise en œuvre :**  
 Ressources (CPU) partagées :  
 → Temps de création dynamique et de re-répartition de processus



Ressources (CPU) allouées au lancement de l'application :  
 → Gaspillage !

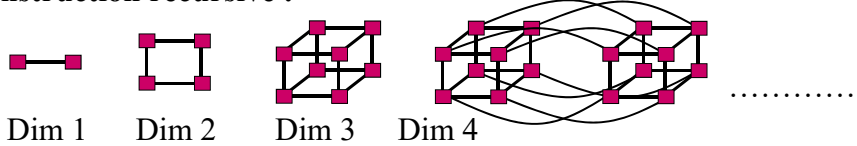


CentraleSupélec

## Quick-sort sur hypercube de processeurs

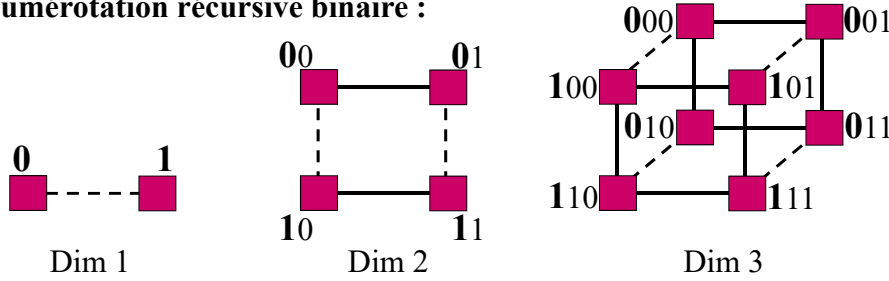
### Propriétés des hypercubes

- **Construction récursive :**



Dim 1    Dim 2    Dim 3    Dim 4    .....

- **Numérotation récursive binaire :**



Dim 1    Dim 2    Dim 3

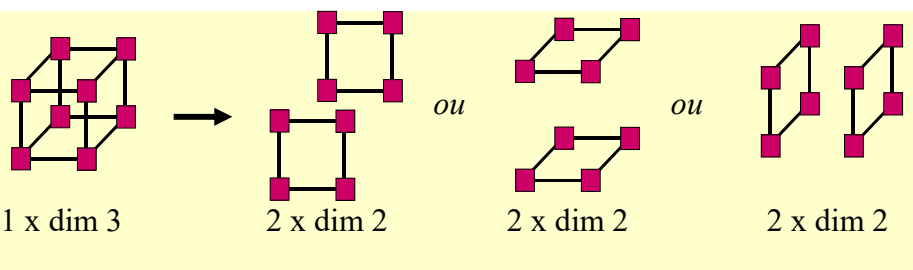
CentraleSupélec

## Quick-sort sur hypercube de processeurs

### Propriétés des hypercubes

- **Décomposition en sous-hypercube :**

1 hypercube de dimension  $n$  coupé par un hyper-plan donne 2 sous-hypercubes de dimension  $n-1$  :



1 x dim 3    2 x dim 2    ou    2 x dim 2    ou    2 x dim 2

→ Les algorithmes de routages restent les mêmes dans toutes les parties de l'hyper-cube

CentraleSupélec

## Quick-sort sur hypercube de processeurs

### Propriétés des hypercubes

- **Distance entre nœuds :**  
Le nombre de bits différents dans les adresses de deux nœuds donne leur distance (si la numérotation a suivi la construction récursive)

Ex : ■ A : 000, B : 011  
 $d(M,N) = 2$

Ex : ■ M : 110, N : 001  
 $d(M,N) = 3$

→ Les algorithmes de routages seront à base de calculs simples (et rapides)

CentraleSupélec

## Quick-sort sur hypercube de processeurs

### Algorithme sur hypercube

**Objectif :**

- **Tous les processeurs doivent travailler tout le temps !**
- S'inspirer d'un tri séquentiel rapide ( $O(N \cdot \log(N))$ ) : quick-sort

→ Trouver un schéma de parallélisation avec comm. optimisées (peu de comm, ou comm locales)

**Idee : Quick-sort : algo récursif ↔ topologie récursive : Hyper-cube**

CentraleSupélec

## Quick-sort sur hypercube de processeurs Algorithme sur hypercube

**Init :** Chaque nœud charge N/P données en local

**Etape 0 :**

- Choix de  $2^0$  pivots pour les  $2^0$  hypercubes de dimension d-0
- Séparation selon le pivot sur chaque nœud de l'hypercube
- Echange des listes inférieures et supérieures en dimension d-0
- Fusions locales des listes conservées et des listes reçues

Communication avec voisin en dimension d-0 :  
seul le bit d-0 diffère

CentraleSupélec

## Quick-sort sur hypercube de processeurs Algorithme sur hypercube

**Détails de l'étape 0 sur PE-000 et PE-100 :**

Séparations locales

Echanges de sous-listes

Fusion des sous-listes conservées et reçues

CentraleSupélec

## Quick-sort sur hypercube de processeurs Algorithme sur hypercube

**Etape 1 :**

- Choix de  $2^1$  pivots pour les  $2^1$  hypercubes de dimension d-1
- Séparation selon un pivot sur chaque nœud des  $2^1$  hypercubes
- Echange des listes inférieures et supérieures en dimension d-1
- Fusions locales des listes conservées et des listes reçues

Communication avec voisin en dimension d-1 : seul le bit d-1 diffère

CentraleSupélec

## Quick-sort sur hypercube de processeurs Algorithme sur hypercube

**Etape 2 :**

- Choix de  $2^2$  pivots pour les  $2^2$  hypercubes de dimension d-2
- Séparation selon un pivot sur chaque nœud des  $2^2$  hypercubes
- Echange des listes inférieures et supérieures en dimension d-2
- Fusions locales des listes conservées et des listes reçues

Communication avec voisin en dimension d-2 : seul le bit d-2 diffère

CentraleSupélec

## Quick-sort sur hypercube de processeurs

### Algorithme sur hypercube

**Etape finale :**  
 Chaque processeur tri en local sa liste finale  
 → ex : quick-sort local et séquentiel sur chaque processeur

CentraleSupélec

## Quick-sort sur hypercube de processeurs

### 1<sup>ère</sup> implantation

```

HcubeQuickSort(double *A, int d)
{
  int i;                               /* Compteur de dimension. */
  double *Ainf, *Asup, *Abuf;          /* Tables de données. */
  double x;                             /* Pivot. */
  for (i = d-1; i >= 0; i--) { /* Pour chaque dimension: */
    x = choix_pivot(me,i);             /* - Partitionnement local*/
    partitioner(A,x,Ainf,Asup);
    if (me & exp2(i) == 0) { /* - Communications */
      asend(Asup,me | exp2(i));
      recv(Abuf,me | exp2(i));
      union(Ainf,Abuf,&A);
    } else {
      asend(Ainf,me & ~exp2(i));
      recv(Abuf,me & ~exp2(i));
      union(Abuf,Asup,&A);
    }
  }
  QuickSortSequentiel(A);             /* Tri final des donnes */
}                                       /* locales */

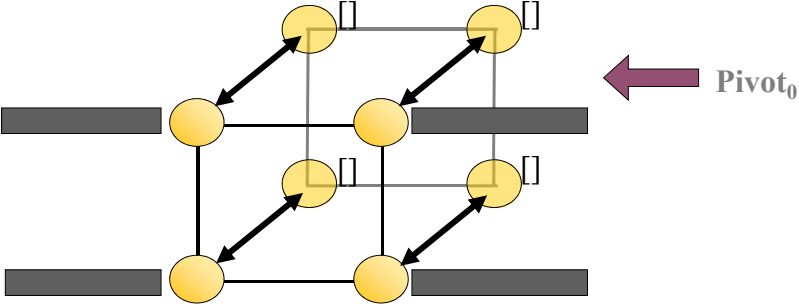
```

CentraleSupélec

## Quick-sort sur hypercube de processeurs 1<sup>ère</sup> implantation

**Faiblesse de ce premier hyper-quicksort :**

**Si mauvais choix de pivot : déséquilibre définitif !**  
→ processeurs surchargés et processeurs à vide



→ Soigner le choix du pivot.

CentraleSupélec

## Quick-sort sur hypercube de processeurs Algorithme optimisé

**Choix de pivot optimisé :**


→ Pivot idéal : élément médian →

**En séquentiel :** on approxime l'élément médian, ex :

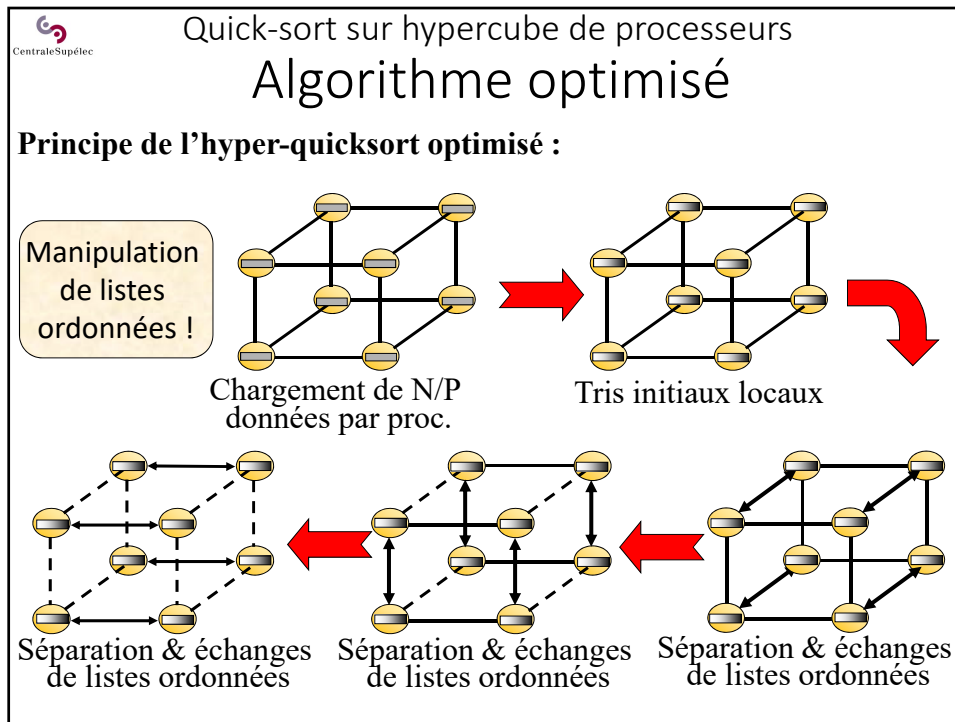
- l'élément médian des 5 premiers,
- l'élément médian d'un échantillonnage, ....

**En parallèle :**

- on trie initialement les éléments locaux  
→ on prend l'élément médian ! LocalTab[N/P/2]
- on suppose une distribution homogène sur tous les PEs  
→ le pivot idéal d'un PE est un très bon pivot pour tous !







CentraleSupélec

## Quick-sort sur hypercube de processeurs

### 2<sup>ème</sup> implantation

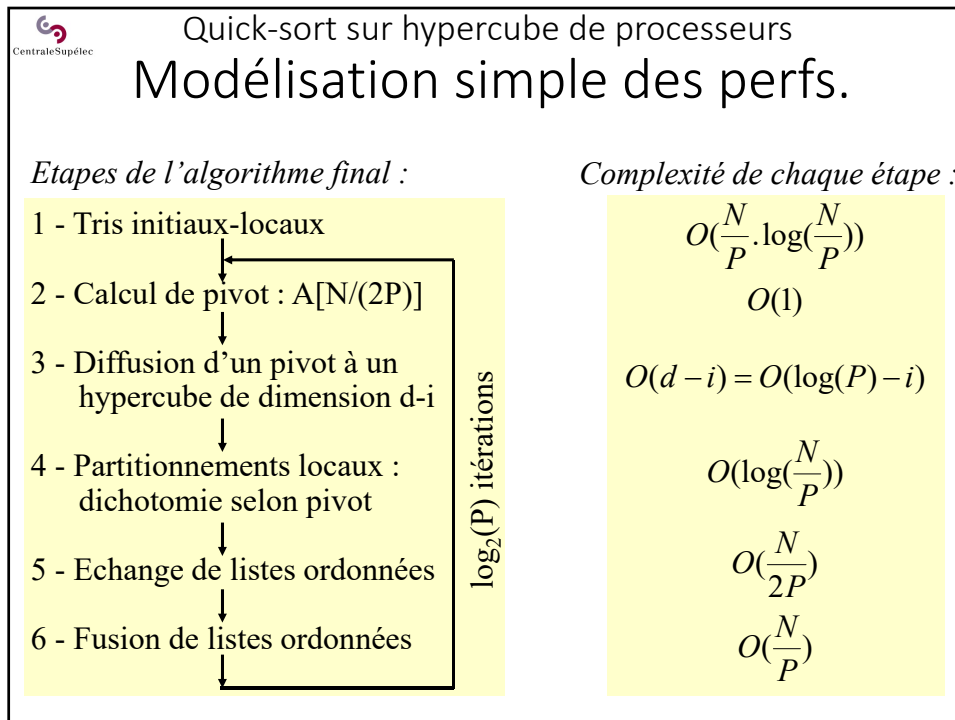
**Implantation de l'hyper-quicksort optimisé :**

```

HcubeQuickSort(double *A, int d)
{
  int i;
  double *Ainf, *Asup, *Abuf;
  double x;
  QuickSortSequentiel(A);
  for (i = d-1; i >= 0; i--) {
    x = choix_pivot(me,i);
    partitioner(A,x,Ainf,Asup);
    if (me & exp2(i) == 0) {
      asend(Asup,me | exp2(i));
      recv(Abuf,me | exp2(i));
      union_ordonne(Ainf,Abuf,&A);
    } else {
      asend(Ainf,me & ~exp2(i));
      recv(Abuf,me & ~exp2(i));
      union_ordonne(Abuf,Asup,&A);
    }
  }
}

```

*/\* Compteur de dimension. \*/*  
*/\* Tables de données. \*/*  
*/\* Pivot. \*/*  
*/\* Tri initial des données locales. \*/*  
*/\* Pour chaque dimension: \*/*  
*/\* - Partitionnement local \*/*  
*/\* - Communications \*/*



CentraleSupélec

### Quick-sort sur hypercube de processeurs Modélisation simple des perfs.

$$O(T_{par}(N, P)) =$$

$$O\left(\frac{N}{P} \cdot \log\left(\frac{N}{P}\right)\right) +$$

$$O(\log(P)) + O\left(\sum_{i=0}^{\log_2(P)-1} (\log_2(P)-i)\right) + O(\log(P) \cdot \log\left(\frac{N}{P}\right)) +$$

$$O(\log(P) \cdot \frac{N}{2P}) + O(\log(P) \cdot \frac{N}{P})$$

$$O(T_{par}(N, P)) = O\left(\frac{N}{P} \cdot \log\left(\frac{N}{P}\right)\right) +$$

$$O(\log(P)) + O\left(\frac{\log^2(P)}{2}\right) + O(\log(P) \cdot \log\left(\frac{N}{P}\right)) + O(\log(P) \cdot \frac{N}{P})$$

CentraleSupélec

Quick-sort sur hypercube de processeurs  
Modélisation simple des perfs.

$$O(T_{par}(N, P)) = O\left(\frac{N}{P} \cdot \log\left(\frac{N}{P}\right)\right) + O(\log(P)) + O\left(\frac{\log^2(P)}{2}\right) + O(\log(P) \cdot \log\left(\frac{N}{P}\right)) + O(\log(P) \cdot \frac{N}{P})$$

↓

**Pour  $P$  fixé** (sur une machine parallèle donnée) :

↓

$$O(T_{par}(N)) = O(N \cdot \log(N)) + O(\log(N)) + O(N)$$

↓

$$O(T_{par}(N)) = O(N \cdot \log(N))$$

**Idem tri séquentiel !!**

CentraleSupélec

Quick-sort sur hypercube de processeurs  
Modélisation simple des perfs.

**Bilan de la modélisation rapide de l'hyper-quick-sort:**

$$O(T_{par}(N)) = O(N \cdot \log(N))$$

- Le tri séquentiel initial des données locales masque encore tout !  
→ Tri séquentiel  $\cong$  Hyper-Quicksort !
- Raisonner plus finement sur des ordres de grandeurs est délicat !  
ex :  $O(N/P \cdot \log(N/P)) + O(\log(P)) <?> O(N \cdot \log^2(P))$

**→ Faire une modélisation plus précise – ne pas se contenter des ordres de grandeur : calculer les TEMPS d'exécution**

**Rappel :** Une parallélisation réelle sur une machine à  $P$  processeurs :

- Ne change pas la complexité du problème
- Divise juste le temps par une valeur fixe (le speed up!)

CentraleSupélec

Quick-sort sur hypercube de processeurs

## Modélisation détaillée des perfs.

$$T_{par}(N, P) = \alpha \cdot \frac{N}{P} \cdot \log\left(\frac{N}{P}\right) \cdot t_{comp} + \frac{(\log(P)-1) \cdot \log(P)}{2} \cdot (t_s + t_w) + \log(P) \cdot \log\left(\frac{N}{P}\right) \cdot t_{comp} + \log(P) \cdot \left(t_s + \frac{N}{2 \cdot P} \cdot t_w\right) + \log(P) \cdot \frac{N}{P} \cdot t_{comp}$$

$$T_{par}(N, P) = \alpha \cdot \frac{N}{P} \cdot \log\left(\frac{N}{P}\right) \cdot t_{comp} + \log(P) \cdot \log\left(\frac{N}{P}\right) \cdot t_{comp} + \log(P) \cdot \frac{N}{P} \cdot t_{comp} + \frac{(\log(P)-1) \cdot \log(P)}{2} \cdot (t_s + t_w) + \log(P) \cdot \left(t_s + \frac{N}{2 \cdot P} \cdot t_w\right)$$

CentraleSupélec

Quick-sort sur hypercube de processeurs

## Bilan de la modélisation des perfs.

**Démarche :**

- 1 - Modélisation du temps d'exécution de l'hyper-quicksort :**
  - hyp sur le réseau d'interconnexion des processeurs
  - modélisation de base des communications
  - hyp sur la distribution des données et la qualité des pivots
$$T_{par}^{hqs}(N, P) = \dots$$
- 2 - Comparaison aux temps d'exécution d'autres tris parallèle :**

$$T_{par}^{hqs}(N, P) < ? > T_{par}^{bs}(N, P)$$
- 3 - Confrontation à l'expérimentation**

**Problèmes :** beaucoup d'hypothèses → manque de précision

- **Modéliser pour classifier les solutions,**
- Expérimenter pour connaître les temps d'exécution

## Algorithmique distribuée

**Fin**