

Introduction aux algorithmes et architectures parallèles

Chaire UNESCO

Calcul Numérique Intensif

TUNIS - Février 2004

Mohamed JEMNI

École Supérieure des Sciences et Techniques de Tunis

Bibliographie

- M. Cosnard & D. Trystram, Algorithmes et architectures parallèles, InterEditions, Paris, 1993.
- P. Kuonen, La Programmation parallèle, Notes de cours EPFL.
- M. J. Flynn, Computer Architecture, Pipelined and parallel processor design, Jones and Barlett Publishers, 1995.
- D. E. Culler and Cie, Parallel Computer Architecture, A Hardware/Software Approach, Morgan Kaufmann Pub. 1999.
- F.T. Leighton, Introduction to parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann Pub. 1992.

Bibliographie

- <http://www.top500.org/>
- <http://www.mpi-forum.org>
- <http://www.netlib.org/utk/papers/mpi-book/mpi-book.html>
- <http://www.openmp.org>
- www.netlib.org/scalapack/scalapack_home.html

Architectures Parallèles

Pourquoi?

- Approche classique : diminuer le temps de calcul en effectuant plus vite chaque (ensemble d') opération(s)
- En calcul parallèle : diminuer le temps de calcul en exécutant simultanément plusieurs opérations (tâches)

0. Introduction

Avantages

- amélioration des performances de calcul
- accroissement de la taille des problèmes à résoudre
- Résolution de nouveaux problèmes

Problèmes

- La remise en question des concepts d'algorithmique classique basés sur le principe de la machine séquentielle.
- Diversité des modèles d'architectures parallèles
- Difficulté de la programmation des machines parallèles.

Algorithmique Parallèle Pourquoi?

La conception, l'analyse et l'évaluation
d'algorithmes destinés à s'exécuter sur des
machines parallèles.

PLAN

- 1. Historique**
- 2. Classification**
- 3. Concepts de base de l'algorithmique parallèle**
- 4. Les sources du parallélisme**
- 5. Modèles de programmation Parallèle**
- 6. Techniques de Parallélisation**

1. Historique

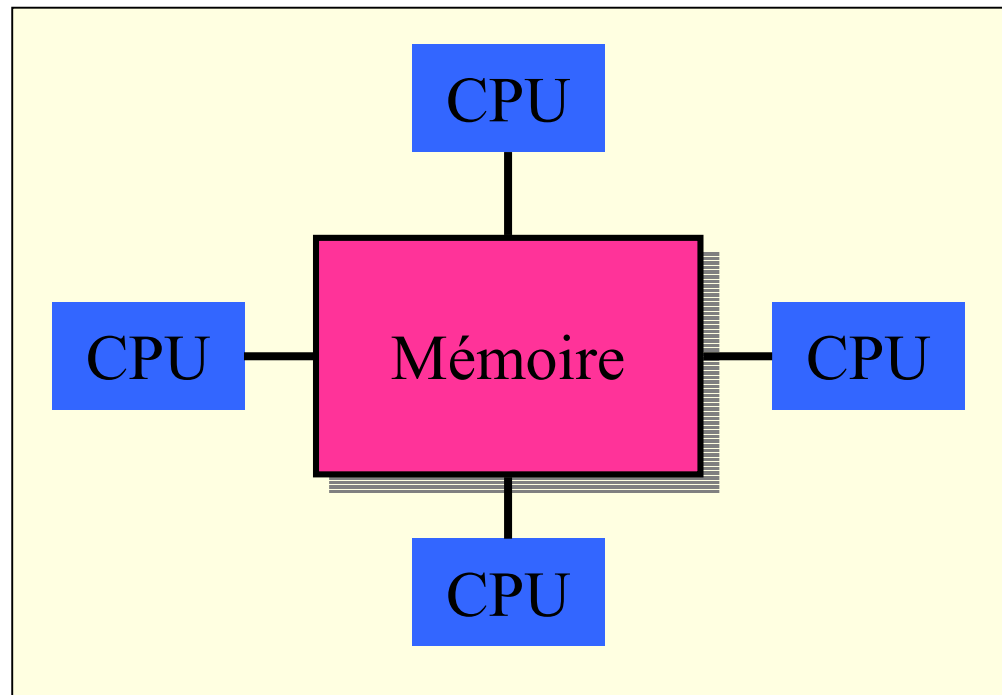
- Deux approches différentes:
 - les architectures multiprocesseurs
 - les architectures vectorielles
- La première machine parallèle (1972): la machine ILLIAC IV a été construite par Burroughs et livrée au NASA
 - ☞ Architecture : tableau de 8x8 processeurs connectés en tore de dimension 2.
- Le premier ordinateur vectoriel (1973) : le STAR 100 construit par CDC (Control Data Corporation)

1.1 Les ordinateurs vectoriels

- **Cray Research Inc**
 - CRAY-1 (en 1976) : 130 MFLOPS
 - CRAY-XMP (en 1982) : le premier Superordinateur vectoriel multiprocesseurs: 2 procs
 - CRAY-2 (en 1985) : Superordinateur vectoriel à 4 procs et qui avait une puissance théorique de 1950 MFLOPS.
- **Autres constructeurs : IBM, NEC, HITACHI, ...**
 - ➔ Énormes machines très coûteuses : le besoin d'architecture plus régulière, extensible et facile à réaliser!

1.2 Les ordinateurs multiprocesseurs

- à mémoire commune (shared memory):
plusieurs processeurs se partagent une même mémoire



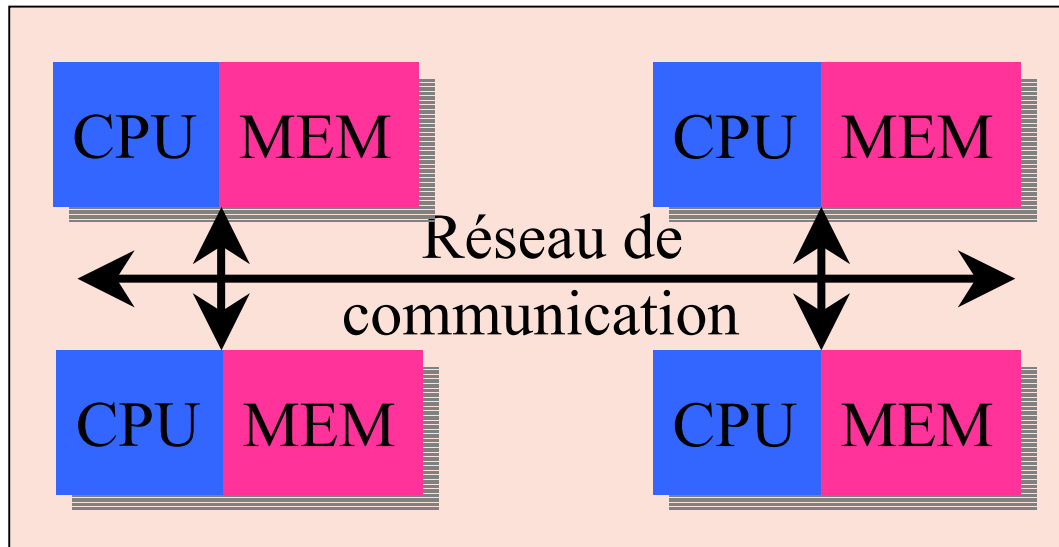
Machine à mémoire partagée

1.2 Les ordinateurs multiprocesseurs

- Exemples de machines à mémoire commune (années 80):
 - HEP en 1981
 - NYU en 1983
 - BBN Butterfly en 1985
 - IBM RP3 en 1987
- Technologie bien maîtrisée
- Programmation concurrente (pas de distribution de données)
- Pb: étranglement de la mémoire (nbre élevé de processeurs)
 - ☞ Limitation à qlqs dizaines de processeurs

1.2 Les ordinateurs multiprocesseurs

- à mémoire distribuée : chaque processeur possède sa propre mémoire et peut communiquer avec les autres processeurs.



Machine à mémoire distribuée

1.2 Les ordinateurs multiprocesseurs

- Exemples de machines à mémoire distribuée :

☞ Apparition du **transputer** en 1980 : un microprocesseur classique avec 4 liens spécialisés (intégrés sur sa puce) pour communiquer directement avec d'autres transputers.

En 1985 : Inmos Inc → machines T212, T44 et T800
Echec du T9000 annoncé pour 1994

1.2 Les ordinateurs multiprocesseurs

- Exemples de machines à mémoire distribuée :

☞ Machines basées sur les microprocesseurs de commerce

- Intel Corp:

1986 → IPSC/1: hypercube 32-128 μ proc Intel 80286/87

1988 → IPSC/2: hypercube (<128) μ proc Intel 80386/87

1990 → IPSC/i860 basée sur le μ proc Intel i860 et avait une performance de crête de 8000 MFLOPS.

▶ 1992 (Projet Touchstone de Intel) :

→ machine Paragon : grille 2048 processeurs i860 avec une performance de crête de 300 GFLOPS

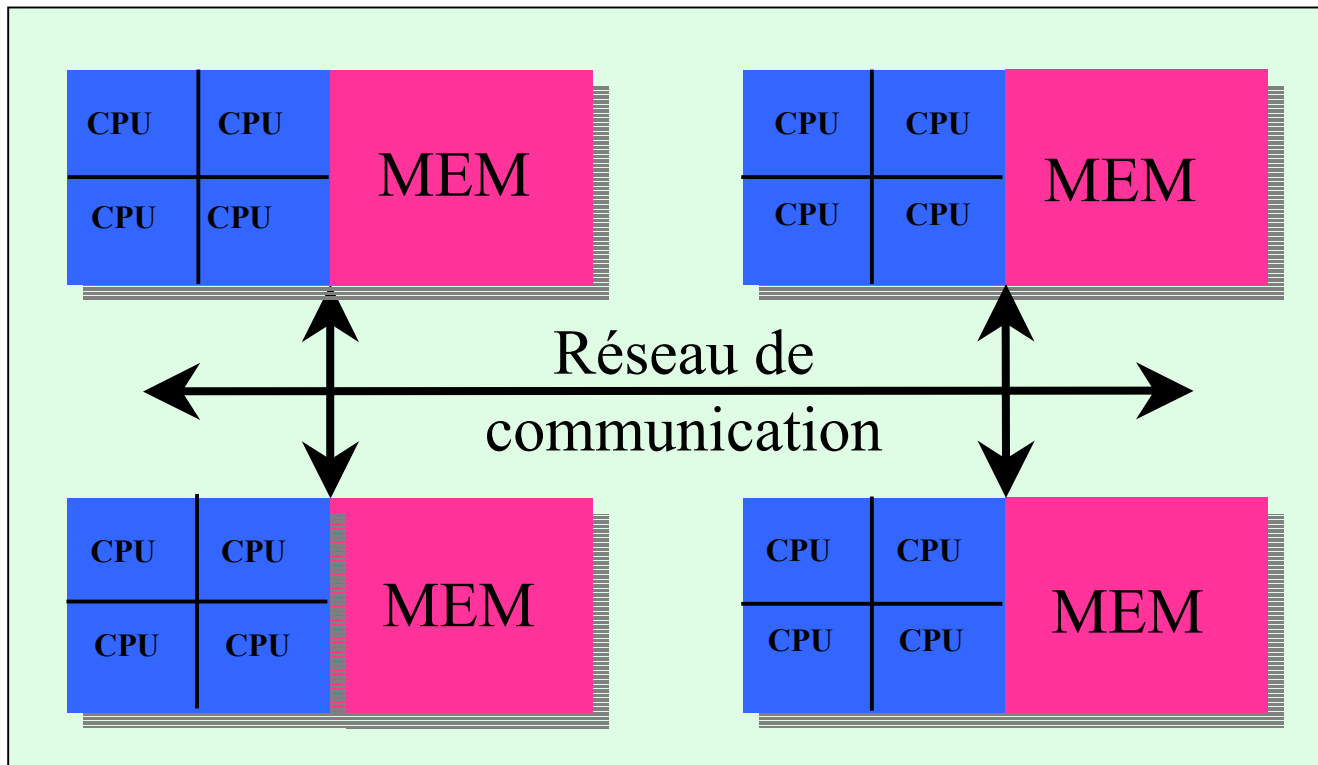
1.2 Les ordinateurs multiprocesseurs

- Thinking Machines : le connexionisme = connecter un nombre très élevé de processeurs extrêmement simples.
1980 → L'ICL DAP : tableau de 4096 procs 1-bit (MC =4K)
CM1 : grille de 65536 (256x256) processeurs 1-bit
1988 → CM2 : amélioration des procs

Et enfin en 1992 CM5 : massivement parallèle conçue pour atteindre une puissance > 1 TeraFLOPS !!!

1.2 Les ordinateurs multiprocesseurs

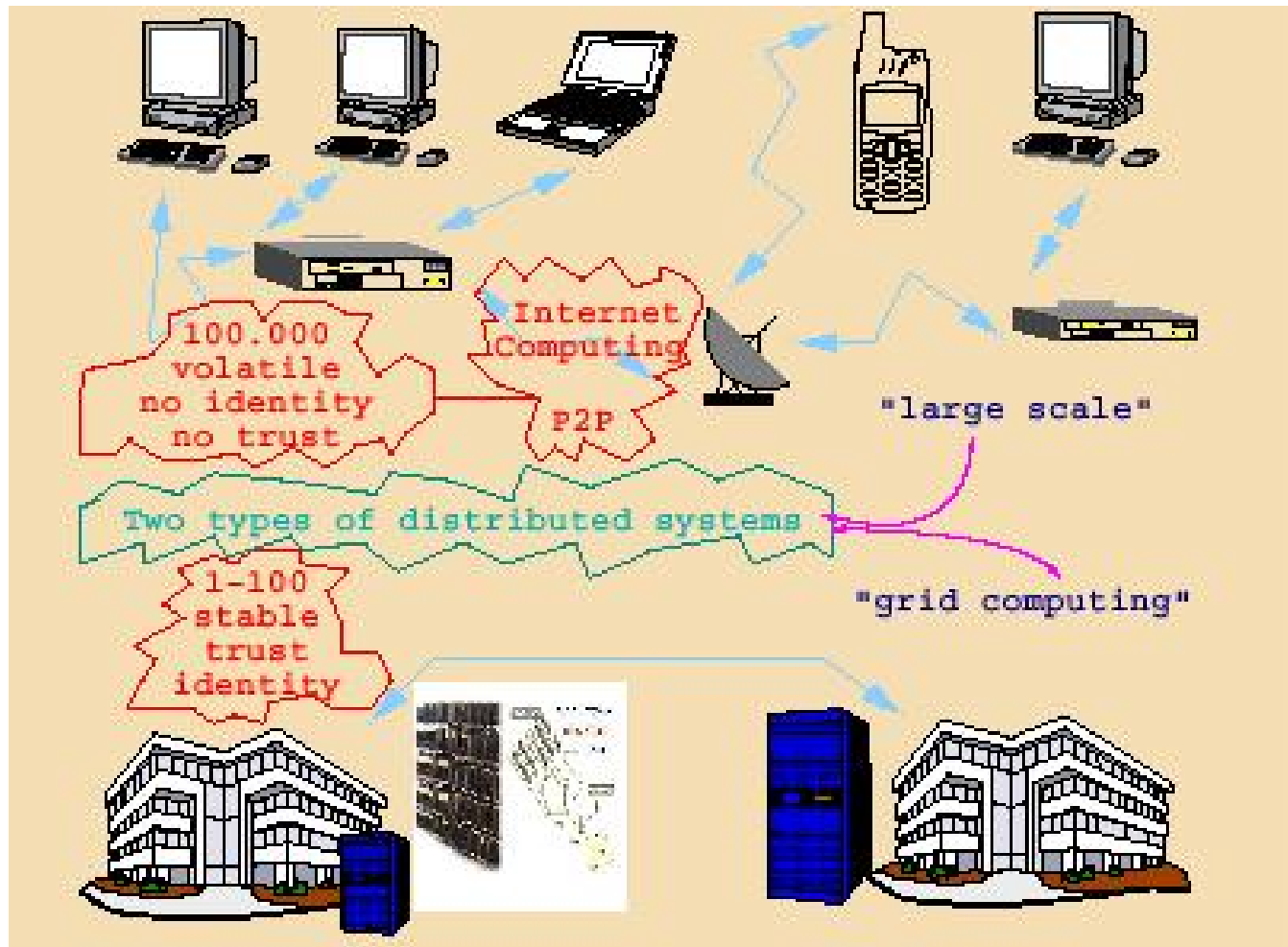
- hybrides : mémoire commune / mémoire partagée



Machines hybrides

1.2 Les ordinateurs multiprocesseurs

- Les grilles



PLAN

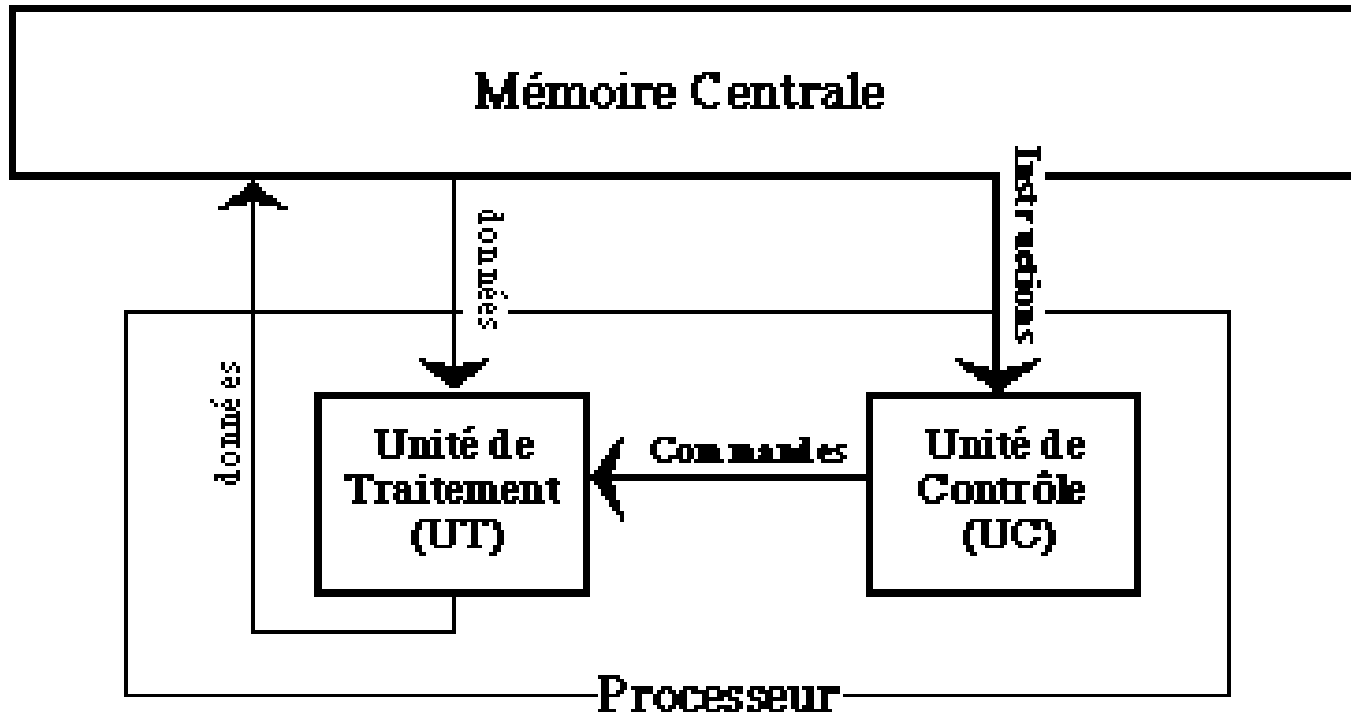
1. Historique
2. Classification
3. Concepts de base de l'algorithmique parallèle
4. Les sources du parallélisme
5. Modèles de programmation Parallèle
6. Techniques de Parallélisation

2. Classification des machines parallèles

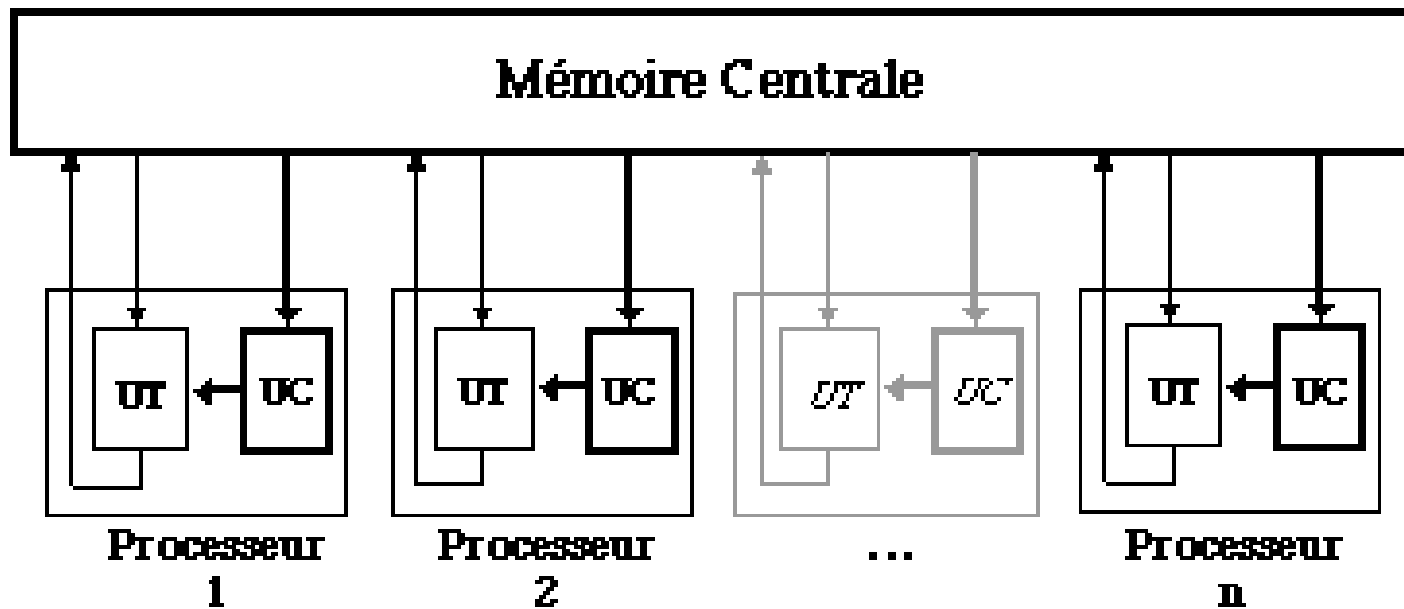
- **Flynn (1966)**

| | | |
|------------------------------|----------------|------------------|
| Flux de D onnées | S ingle | M ultiple |
| Flux d' I nstructions | | |
| S ingle | S ISD | S IMD |
| M ultiple | M ISD | M IMD |

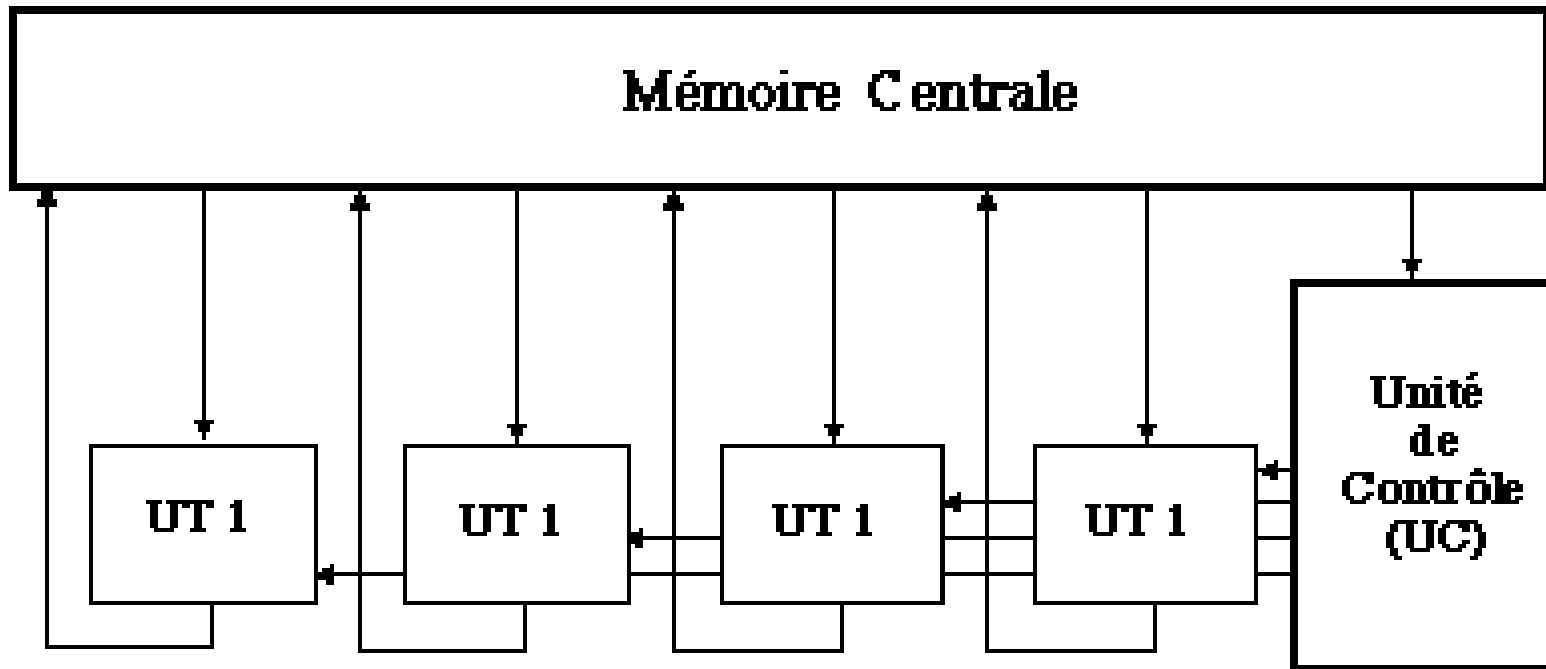
2.1 SISD



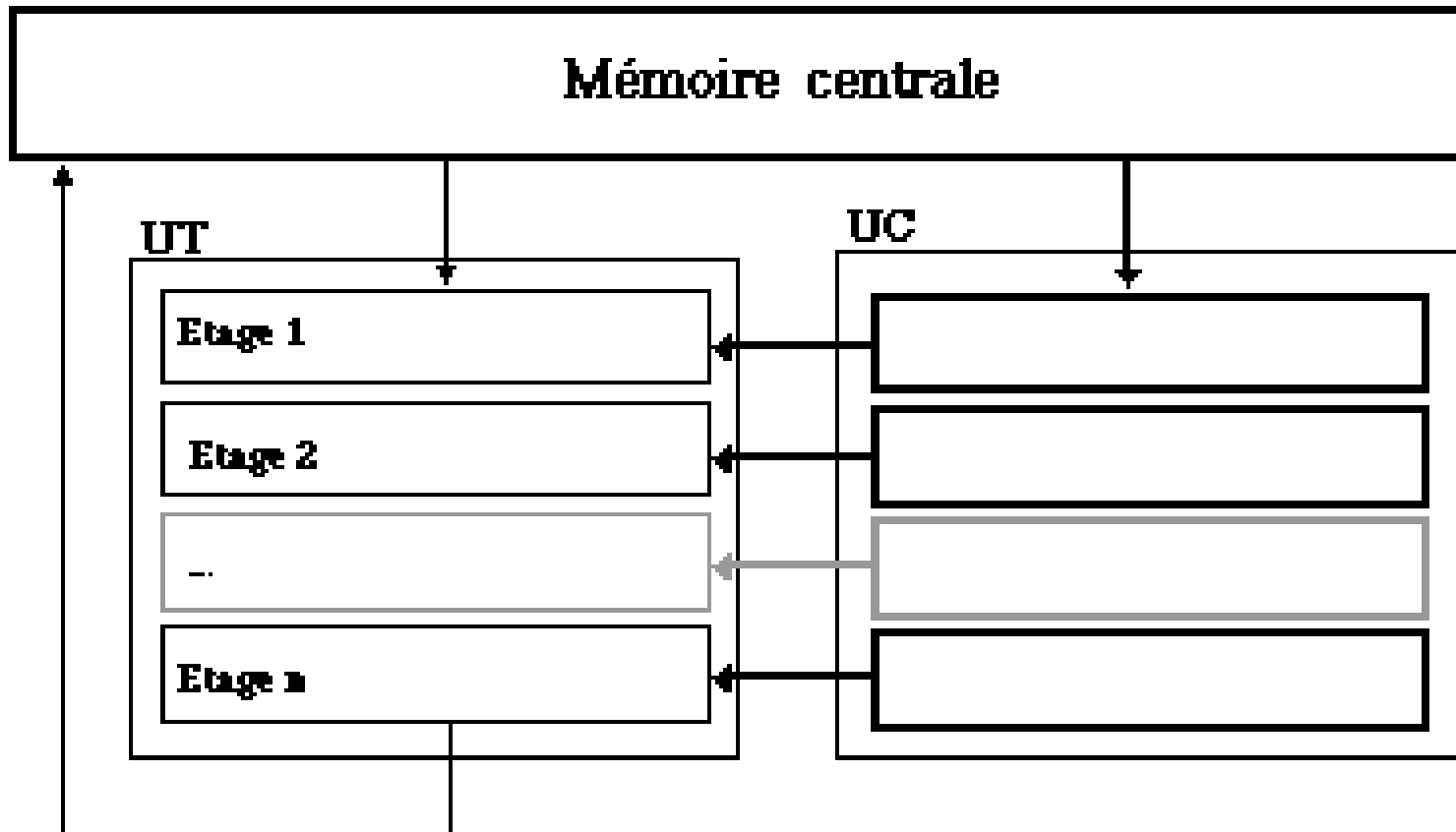
2.2 MIMD



2.3 SIMD



2.4 PIPELINE



2.5 Le modèle PRAM

- **Définition:**

Une Parallel Random Access Machine (PRAM) est un ensemble de processeurs séquentiels indépendants, des RAM, qui possèdent chacun une mémoire privée et qui communiquent entre eux en utilisant une mémoire globale qu'ils se partagent

2.5 Le modèle PRAM

- Opérations fondamentales sur les PRAM:

Considérons une PRAM composée de p processeurs P_0, \dots, P_{p-1} et de m positions mémoire $M_0 \dots M_{m-1}$. Chaque processeur peut faire:

Lire(M_j)

Calculer(f)

Ecrire(M_j)

Toutes ces opérations atomiques s'effectuent de manière synchrone.

2.5 Le modèle PRAM

- Variantes du modèle PRAM:

4 variantes selon les accès à la mémoire commune:

- PRAM-EREW : Exclusive Read Exclusive Write
- PRAM-CREW : Concurrent Read Exclusive Write
- PRAM-ERCW : Exclusive Read Concurrent Write
- PRAM-CRCW : Concurrent Read Concurrent Write

Le cas CR ne pose pas de problème

→ Problème pour CW

2.5 Le modèle PRAM

- Stratégies possibles pour CW:

Conflit: dans le cas où plusieurs processeurs peuvent écrire le contenu de la même case mémoire

→ 3 modèles pour résoudre ce conflit:

- Modèle commun : tous les procs doivent écrire la même valeur.
- Modèle arbitraire : un quelconque proc écrit sa valeur
- Modèle prioritaire : les procs sont classés suivant un ordre de priorité, seul le plus prioritaire accède à la mémoire.

PLAN

1. Historique
2. Classification
3. Concepts de base de l'algorithmique parallèle
4. Les sources du parallélisme
5. Modèles de programmation Parallèle
6. Techniques de Parallélisation

3. Concepts de base

- L'accélération:

Considérons un algorithme qui s'exécute sur un ordinateur parallèle comportant p processeurs (identiques) en un temps T_p , et soit T_1 son temps d'exécution séquentiel.

L'accélération est définie par le rapport: $S_p = T_1/T_p$

Généralement on a : $1 \leq S_p \leq p$

3. Concepts de base

- L'efficacité

L'efficacité d'un algorithme parallèle est le rapport:

$$E_p = S_p/p$$

3. Concepts de base

Machine séquentielle

- dupliquer la vitesse du processeur
⇒ diviser le temps de calcul par 2

Machine parallèle

- dupliquer le nombre de processeurs
⇒ ~~diviser le temps d'exécution par 2~~

3. Concepts de base

- Loi d'Amdhal

$$T_1 = T_{\text{seq}} + T_{\text{par}}$$

$$S_p \leq (T_s + T_{\text{par}}) / (T_s + T_{\text{par}}/p)$$

Donc
$$S_p \leq T_1 / T_{\text{seq}}$$

Exemple: si 10% du programme est séquentiel alors le facteur d'accélération sera inférieur à 10 quel que soit p.

PLAN

1. Historique
2. Classification
3. Concepts de base de l'algorithmique parallèle
4. Les sources du parallélisme
5. Modèles de programmation Parallèle
6. Techniques de Parallélisation

4. Les sources du parallélisme

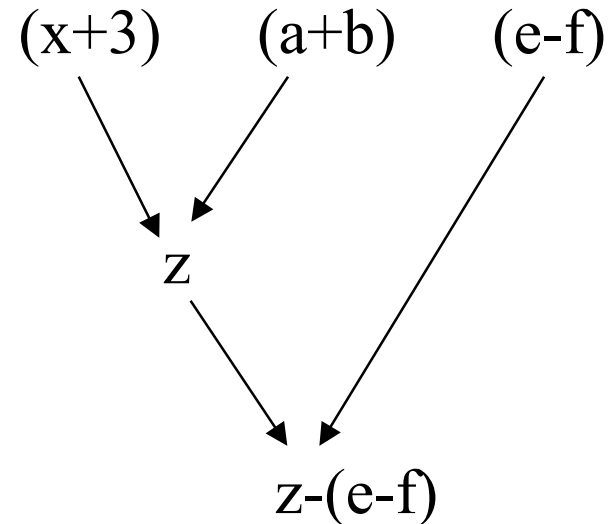
4.1 Le parallélisme de contrôle

- découle de la constatation naturelle d'opérations (tâches) simultanées
- **Exemple** : $(x+3)*(a+b)-(e-f)$

Calculer

- $(x+3)$ et $(a+b)$ et $(e-f)$
- puis $(x+3)*(a+b)$ (noté z)
- puis $z-(e-f)$

⇒ 3 processeurs
3 unités de temps



4. Les sources du parallélisme

- Généralement :
 - construire un graphe de dépendance
 - trouver le bon compromis entre le degré du parallélisme et le temps total d'exécution

- Exemple 1:

a,b,c,d 4 vecteurs

$$\forall i \in [1..N] \quad d(i) = a(i) - b(i)$$

$$c(i) = c(i-1) + a(i) + b(i) \quad /* \quad c(0) = 0 \quad */$$

4. Les sources du parallélisme

/ Programme séquentiel en ADA */*

```
PROCEDURE exemple1 IS
-- Déclarations ... --
BEGIN
  c (0) := 0;
  FOR i IN 1..n LOOP
    d(i) := a(i) - b(i)
    c(i) := c(i-1) + a(i) + b(i)
  END FOR;
END exemple1;
```

Les opérations $a(i) - b(i)$ et $a(i) + b(i)$ sont **indépendantes**

4. Les sources du parallélisme

/ Parallélisme de contrôle */*

```
PROCEDURE exemple1 IS
-- Déclarations ... --
BEGIN
  c (0) := 0;
  FOR i IN 1..n LOOP
    PARBEGIN
      d(i) := a(i) - b(i)
      t(i) := a(i) + b(i)
    END PAR;
    c(i) := c(i-1) + t(i)
  END FOR;
END exemple1;
```

4. Les sources du parallélisme

- **Exemple 2:** somme de deux vecteurs $A=B+C$

N : taille des vecteurs,

P : nombre des processeurs

Programme pour la machine CRAY T3D,
Fortran et PVM

Parameter (P=128)

Parameter (N=1024)

Parameter (**Size=N/P**)

Parameter (Master=0)

Real A(N), B(**Size**), C(**Size**)

Integer MyPE, Ok

...

4. Les sources du parallélisme

```
Call pvmfmyTid(MyPE)
DO I = 1, Size
  A(I) = B(I) + C(I)
ENDDO
IF (MyPE .EQ. Master) THEN
  DO I = 1, P-1
    Call pvmfRecv (I,1,Ok)
    Call pvmfUnpack (real8, A(I*Size),Size,1,Ok)
  ENDDO
ELSE
  Call pvmfInitsend (pvmRaw,Ok)
  Call pvmfPack (real8, A(1), Size, 1, Ok)
  Call pvmfSend (Master,1,Ok)
ENDdIF
```

4. Les sources du parallélisme

4.2 Le parallélisme de données :

- répéter une action (programme) sur des données similaires
- les programmes d'algèbre linéaire font intensivement des opérations portant sur des grandes matrices.

⇒ diviser ces grands volumes de données en morceaux qui seront traités par différentes unités de calcul.

Typiquement: les ordinateurs vectoriels

- **Exemple 3:**

produit de deux matrices $n \times n$ $C=A*B$

4. Les sources du parallélisme

-- Programme séquentiel en ADA --

```
... N : CONSTANT integer := 1024;
PROCEDURE Produit_Mat (A,B: in Matrice; C: out matrice) IS
BEGIN
FOR i IN 1 .. N LOOP
  FOR j IN 1 .. N LOOP
    C(i,j) := 0.0;
    FOR k IN 1..N LOOP
      C(i,j) := C(i,j) + A(i,k)*B(k,j);
    END FOR;
  END FOR;
END FOR;
END Produit_Mat;
```

→ temps d'exécution : $O(N^3)$

4. Les sources du parallélisme

-- Programme vectorisé --

```
PROCEDURE Produit_Mat (A,B: in Matrice; C: out matrice) IS
TYPE Vecteur IS ARRAY (1..N) OF float;
T : Vecteur;
BEGIN
FOR i IN 1 .. N LOOP
  FOR j IN 1 .. N LOOP
    T(1:N) := Vmul ( A(i,1:N), B(1:N,j) )
    C(i,j) := Vsum(T(1:N))
  END FOR;
END FOR;
END Produit_Mat;
```

→ temps d'exécution $O(N^2)$ avec des registres de taille N

Pb: taille des registres

4. Les sources du parallélisme

```
-- Code pour pour machine MIMD a mémoire partagée --  
... N : CONSTANT integer := 1024;  
PROCEDURE Produit_Mat (A,B: in Matrice; C: out matrice) IS  
BEGIN  
FORALL i IN 1 .. N LOOP  
    FORALL j IN 1 .. N LOOP  
        C(i,j) := 0.0;  
        FOR k IN 1..N LOOP  
            C(i,j) := C(i,j) + A(i,k)*B(k,j);  
        END FOR  
    END ALL;  
END ALL;  
END Produit_Mat;
```

→ temps d'exécution $O(N)$ avec N^2 processeurs

Pb: étranglement de la mémoire

4. Les sources du parallélisme

-- Code pour pour machine MIMD-DM à N^2 processeurs --

Processeur IS p;

N : CONSTANT integer := 1024;

....

PROCEDURE Produit_Mat (A,B: in Vecteur; C: out float) IS

BEGIN

 C := 0.0;

 FOR k IN 1..N LOOP

 C := C + A(k)*B(k);

 END FOR;

....

END Produit_Mat;

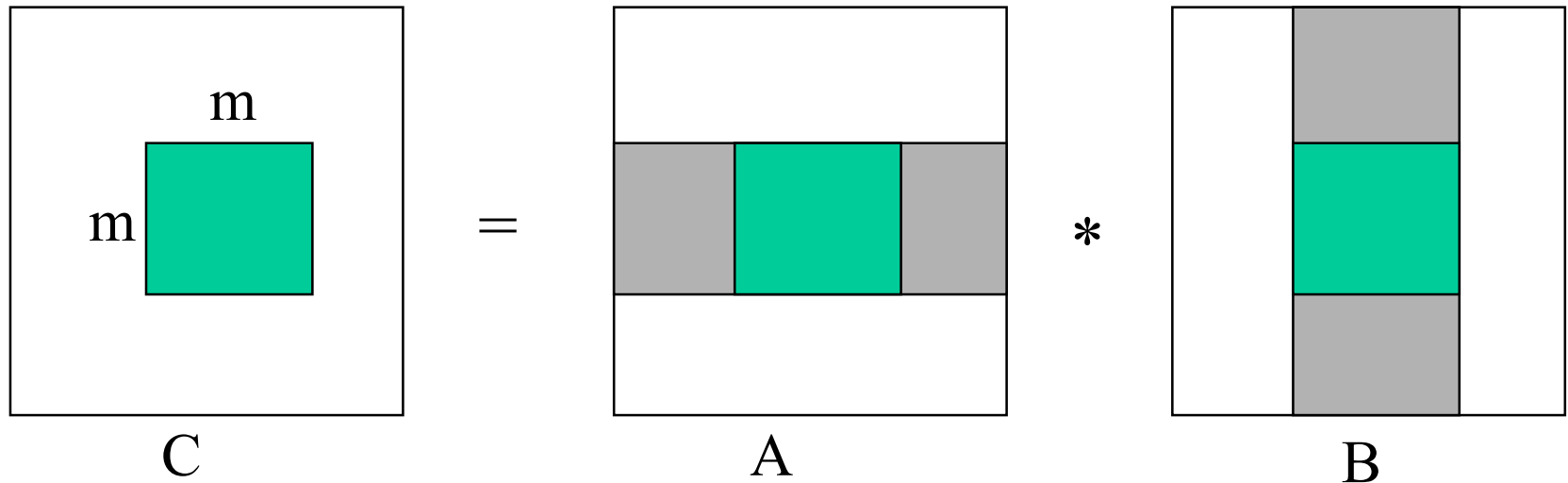
Chaque processeur doit posséder une copie locale
de la ligne i de A et la colonne j de B

4. Les sources du parallélisme

- Pb: $P < N^2 \Rightarrow$ chaque proc. calcule m elts ($C(1..m,j)$)
- $P < N \Rightarrow$ chaque proc. calcule au moins une ligne de C
 \Rightarrow une copie locale de toute la matrice B

très coûteuse en mémoire et communications
(si B est résultat d'un autre calcul)

4. Les sources du parallélisme



- Chaque proc calcule $m \times m$ elts, $m = N/\sqrt{P}$
- **Première solution:** m ligne de A + m colonnes de B
 - \Rightarrow pas de communications
 - \Rightarrow Qté de mémoire pour stocker A et $B = 2(\sqrt{P})N^2$
Exple: si $P = 256$, 16 copies de A et de B

4. Les sources du parallélisme

Pbs:

- Occupation mémoire
 - Communication (cas d'enchaînement de plusieurs calculs)
-
- **De préférence:** une seule copie de A et B avec une répartition qui minimiserait le volume de communication durant le calcul

4. Les sources du parallélisme

- **Deuxième solution** : répartir A et B de la même façon que C

⇒ non nécessité de redistribution des données
(cas de succession de calcul)

⇒ Communication pour chercher les parties manquantes

$$\text{Volume de comm.} = 2(mN - m^2) = 2N^2((\sqrt{P}) - 1)/P$$

chaque proc calcule m^2 elts ($N-1$ additions + N multip pour 1 elt)

$$\text{Rapport comm/calcul} = 2(\sqrt{P} - 1)/(2N - 1)$$

Exple: $N=1024$ et $P = 256$, le rapport = 0.016

4. Les sources du parallélisme

- L'exploitation du parallélisme de données sur des architectures MIMD à mémoire distribuée nécessite principalement une **distribution judicieuse** des données afin de minimiser le rapport communications/calculs.

- **Pb. Ouvert:** distribution automatique des données

4. Les sources du parallélisme

4.3 Le parallélisme de flux

Principe : fonctionnement selon le schéma du travail à la chaîne
(en mode pipeline)

typiquement: appliquer à des données une fonction F tq

$$F(x) = F_1(F_2(F_3(\dots F_p(x)\dots)))$$

Sur N données:

$$\text{accélération} = NP/(N+P)$$

Parallélisme limité : borne max de l'accélération est P

PLAN

1. Historique
2. Classification
3. Concepts de base de l'algorithmique parallèle
4. Les sources du parallélisme
- 5. Modèles de programmation Parallèle**
6. Techniques de Parallélisation

5. Modèles de programmation parallèle

Quatre approches courantes de la programmation parallèle

1. Parallélisme implicite
2. Utilisation de bibliothèques
3. Développer un langage parallèle nouveau
4. Étendre un langage séquentiel existant

Pb: la programmation parallèle n'est jamais totalement indépendante de l'architecture sous-jacente.

5. Modèles de programmation parallèle

5.1. Parallélisme implicite

Principe:

C'est le compilateur qui transforme le programme séquentiel en un programme parallèle équivalent.

Avantages:

- simplicité vis-à-vis du programmeur
- récupération de la base énorme de programmes séquentiels
- indépendance par rapport à l'architecture de la machine

5. Modèles de programmation parallèle

Problèmes :

- restriction à la parallélisation de structures régulières (nids de boucles)
- le résultat de la parallélisation dépend fortement de la façon avec laquelle le programme séquentiel est écrit.
- limites des compilateurs paralléliseurs

5. Modèles de programmation parallèle

5.2. Utilisation de bibliothèques

Principe: utiliser un langage séquentiel classique conjointement avec une bibliothèque de primitives parallèles pour les routines très couramment utilisées.

↪ ne pas trop perturber les habitudes des programmeurs.

5. Modèles de programmation parallèle

Exemples:

- **ScaLapack** : version parallèle de Lapack
(routines de base pour la résolution de syst d'équations linéaires)
 - **PBLAS**: version parallèle (Basic Linear Algebra Subprograms)
- ↪ Approche essentiellement limitée aux gros calculs scientifiques du type résolution de systèmes d'équations linéaires.

5. Modèles de programmation parallèle

5.3. Étendre un langage parallèle

Principe: un langage séquentiel classique est étendu avec des primitives permettant l'expression explicite du parallélisme.

Exples: Fortran90, HPF

↳ évite la redéfinition complète d'un langage et du compilateur

Généralement, les fabricants de machines parallèles fournissent une librairie de primitives parallèles avec différents langages (C et/ou Fortran).

Exemples:

- PVM (parallel Virtual Machine)
- MPI (Message passing Interface)

5. Modèles de programmation parallèle

- Fortran90

⇒ apporte des améliorations au langage Fortran77.

⇒ possède des primitives qui permettent d'exprimer le parallélisme de données.

Exemples: MAXVAL, SUM, MAXLOC,

Ces primitives permettent au compilateur Fortran90 de générer du code parallélisé.

5. Modèles de programmation parallèle

↪ toutes les opérations binaires ou unaires peuvent être faites sur des tableaux

Exemple:

```
real A(10,20), B(10,20)
logical L(10,20)
A = A + 1.0
A = SQRT (A)
L = A .EQ. B
A(1,1:5) = B(1,1:9:2) + B(1,2:10:2)
```

Pb:

```
do i = 1,100
  C(i,i) = 0.0
enddo
```

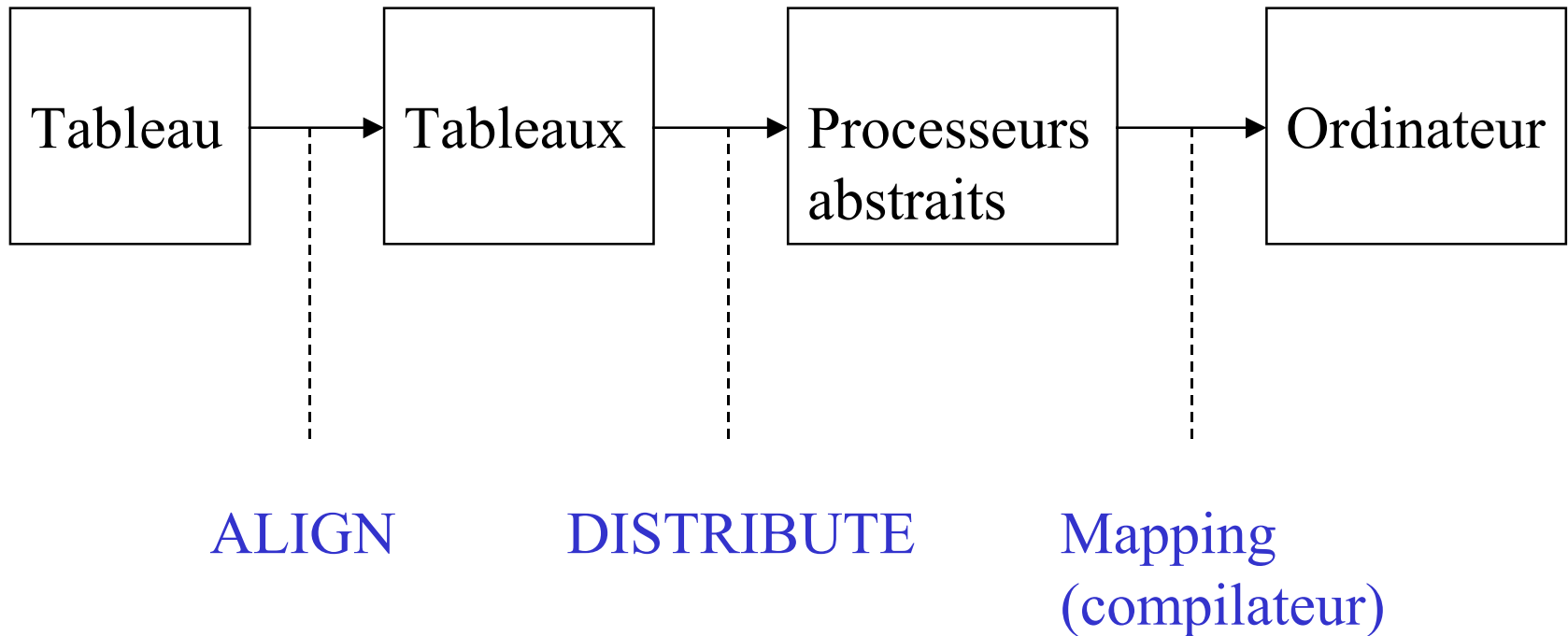
5. Modèles de programmation parallèle

- HPF (High Performance Fortran)

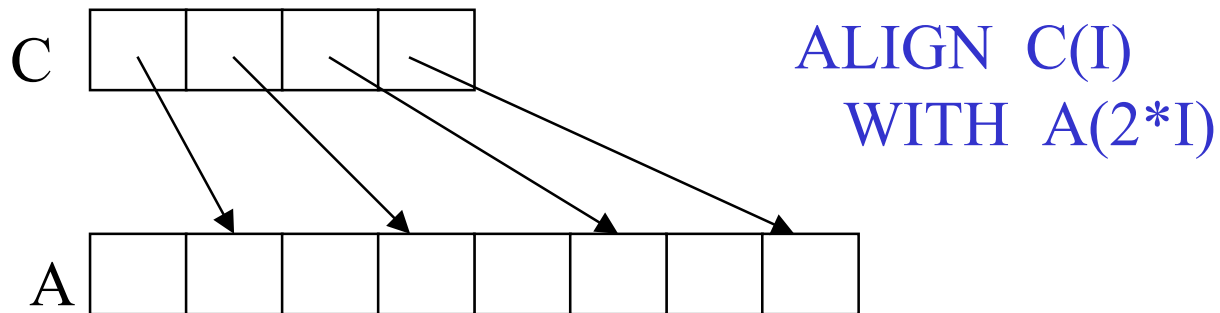
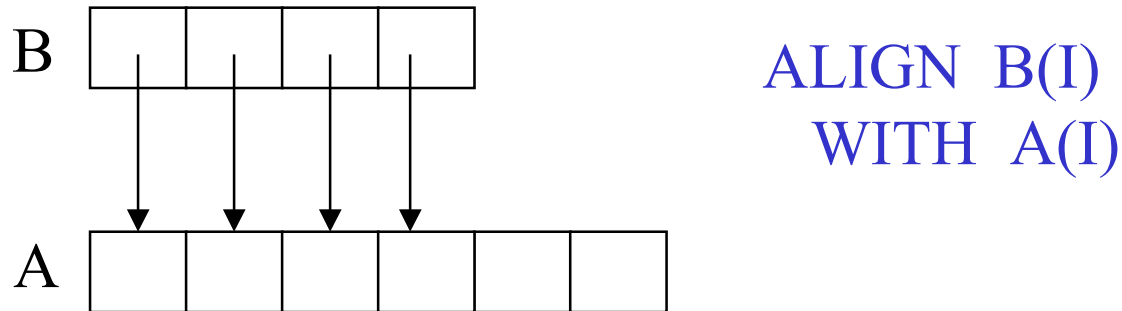
↳ extension du langage Fortran90 avec:

- des directives pour la distribution des données
- l'instruction FORALL
- la directive INDEPENDENT
- de nouvelles fonctions prédéfinies

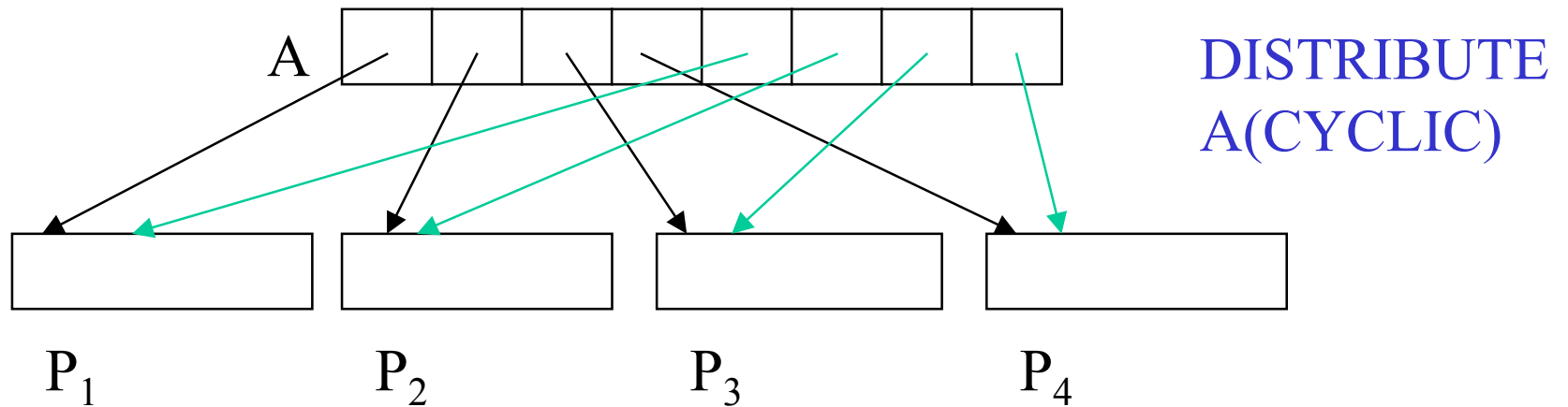
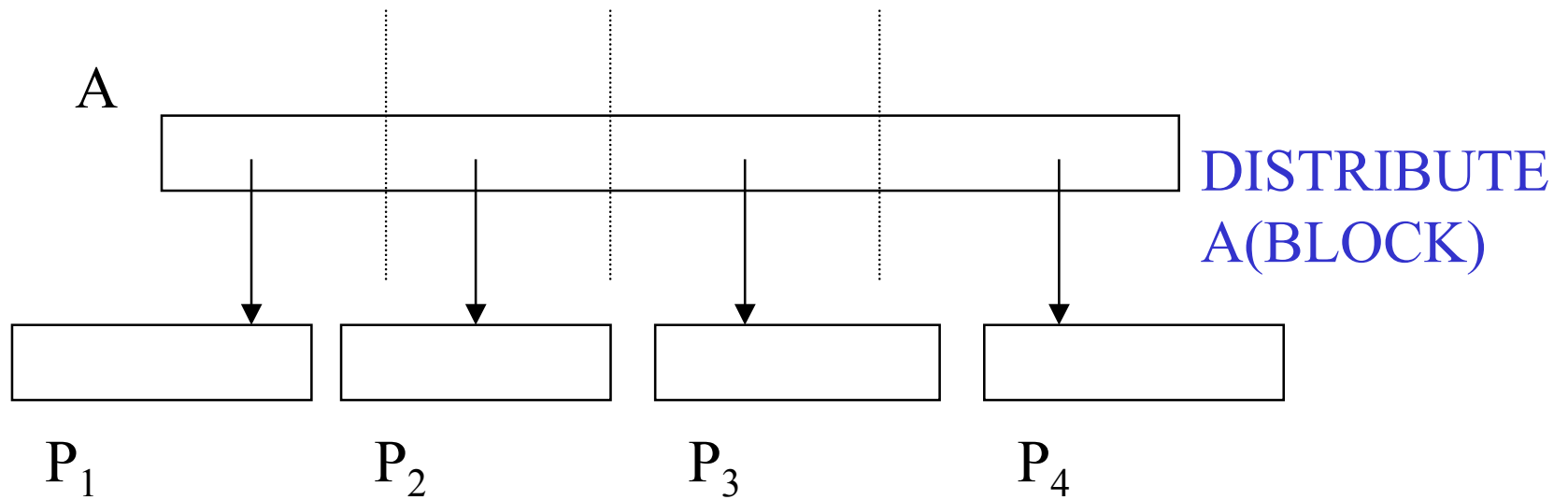
5. Modèles de programmation parallèle



5. Modèles de programmation parallèle



5. Modèles de programmation parallèle



5. Modèles de programmation parallèle

•Exemple

```
!HPF$ PROCESSORS P(4)
!HPF$ ALIGN B(I+2) WITH A(I)
!HPF$ ALIGN C(2*I) WITH A(I)
!HPF$ DISTRIBUTE A(BLOCK) ONTO P
!HPF$ FORALL (I=1:N) A(I) = B(I+2) + C(2*I)
```

...

```
!HPF$ FORALL (I=1:N) X(I,I) = 0.0
!HPF$ FORALL (I=1:N,J=1:N,I<J) Y(I,J) = 0.0
```

...

```
!HPF$ INDEPENDENT
  DO I = 1,n
    A(f(i)) = B(I)
  ENDDO
```

5. Modèles de programmation parallèle

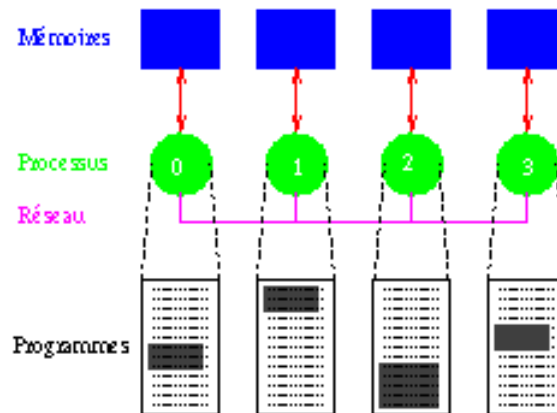
- MPI (Message Passing Interface)
- Bibliothèque standard de primitives de programmation des machines parallèles à mémoire distribuée
- Exprimer explicitement le parallélisme de données
- Gérer explicitement toutes les communications par envoi/réception de messages.
- Plus de 200 primitives

5. Modèles de programmation parallèle

- MPI (Message Passing Interface) <http://www.mpi-forum.org>
 - Bibliothèque standard de primitives de programmation des machines parallèles à mémoire distribuée
 - Exprimer explicitement le parallélisme de données
 - Gérer explicitement toutes les communications par envoi/réception de messages.
 - Plus de 200 primitives

5. Modèles de programmation parallèle

- ➔ Modèle de programmation par **échange de messages** :
- chaque processus exécute éventuellement des parties différentes d'un programme ;
 - toutes les variables du programme sont privées et résident dans la mémoire locale de chaque processus ;
 - une donnée est échangée entre deux ou plusieurs processus via un appel, dans le programme, à des primitives particulières.

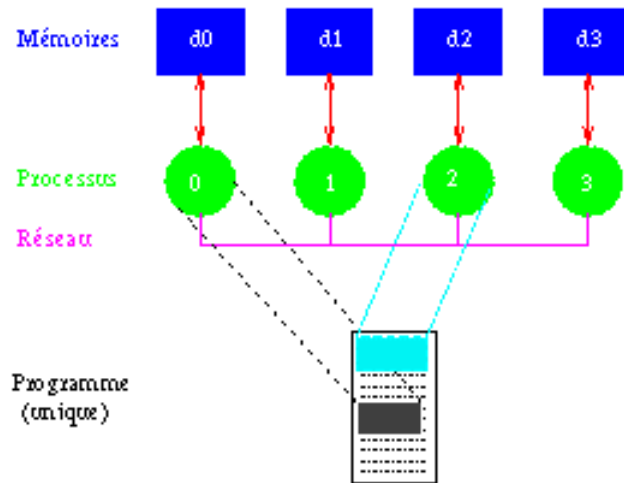


5. Modèles de programmation parallèle

➔ Modèle d'exécution SPMD :

Single Program, Multiple Data

le même programme s'exécute pour tous les processus



5. Modèles de programmation parallèle

Programme MPI

- Style de programmation **SPMD**

Un fichier source -> un exécutable qui sera lancé plusieurs fois

Point d'entrée du fichier :

fonction C *main*

-MPI = une bibliothèque qu'il faut initialiser

MPI_Init() : au début de l'exécution du programme

MPI_Finalize() : à la fin de l'exécution du programme

5. Modèles de programmation parallèle

Un premier exemple

Programme HelloWorld : fichier hello.c

```
1. #include <mpi.h>
2. #include <stdio.h>
3. int main (int argc, char** argv) {
4. MPI_Init( &argc, &argv );
5. printf("HelloWorld\n ");
6. MPI_Finalize();
7. return 0;
8. }
```

Compilation :

```
mpicc -o hello hello
```

Sur l'IBM-SP

```
mpcc -o hello hello.c
```

5. Modèles de programmation parallèle

Lancement : mpirun

Commande :

```
mpirun -np <#nodes>
```

Sur l'IBM-SP :

```
poe hello -procs <#nodes> -msp_api MPI
```

Trace d'exécution sur 1 noeud

```
> mpirun -np 1 hello  
> HelloWorld
```

Trace d'exécution sur 3 noeuds

```
>mpirun -np 3 hello  
>HelloWorld  
>HelloWorld  
>HelloWorld
```

5. Modèles de programmation parallèle

Communicateur MPI

- Au lancement d'un programme MPI

Tous les processus appartient au même groupe

C'est le **communicateur MPI**

- **MPI_COMM_WORLD** : nom du communicateur au lancement

Chaque processus possède un identificateur dans ce groupe

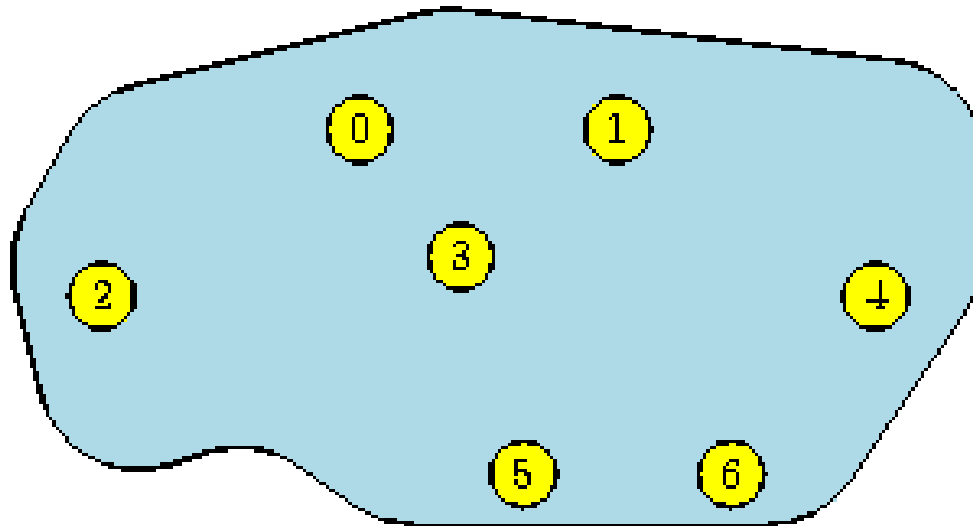
C'est le rang du processus :

- valeur de 0 à #nodes-1

5. Modèles de programmation parallèle

Illustration

`mpirun -np 7 hello`



`MPI_COMM_WORLD`

5. Modèles de programmation parallèle

HelloWorld bis

```
1. #include <mpi.h>
2. #include <stdio.h>
3. int main (int argc, char** argv) {
4. int rank;
5. int size;
6. MPI_Init( &argc, &argv );
7. MPI_Comm_rank( MPI_COMM_WORLD, &rank );
8. MPI_Comm_size( MPI_COMM_WORLD, &size );
9. printf("HelloWorld, my rank is %i in group of
size %i\n", rank, size);
10. MPI_Finalize();
11. return 0;
12. }
```

5. Modèles de programmation parallèle

Traces possibles d'exécution

- Traces d'exécution sur 3 noeuds

>mpirun -np 3 hello

> HelloWorld, my rank is 0 in group of size 3

> HelloWorld, my rank is 1 in group of size 3

> HelloWorld, my rank is 2 in group of size 3

> mpirun -np 3 hello

> HelloWorld, my rank is 2 in group of size 3

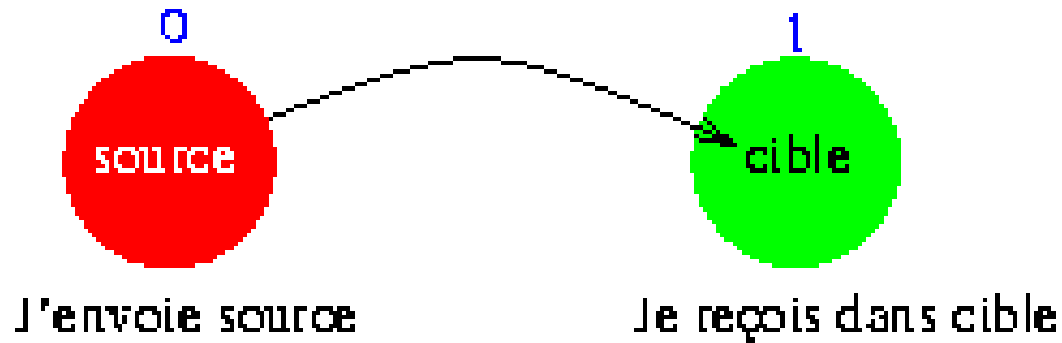
> HelloWorld, my rank is 0 in group of size 3

> HelloWorld, my rank is 1 in group of size 3

5. Modèles de programmation parallèle

Concepts de l'échange de messages

Si un message est envoyé à un processus, celui-ci doit ensuite le recevoir



5. Modèles de programmation parallèle

La donnée d'un message MPI

- Les données dans un message envoyé ou reçu sont décrites par un triplet : *(address, count, datatype)*, où :
 - *address* : l'adresse (un pointeur)
 - *count* : le nombre d'éléments de type datatype
 - *datatype* : un nom de type MPI pour les données
- Exemples de type MPI prédéfini (non exhaustif) et association avec C

| C/C++ | MPI |
|------------------------|-----------------------------|
| char, unsigned char | MPI_CHAR, MPI_UNSIGNED_CHAR |
| int, unsigned int | MPI_INT, MPI_UNSIGNED_INT |
| long, unsigned long | MPI_LONG, MPI_UNSIGNED_LONG |
| float, double | MPI_FLOAT, MPI_DOUBLE |
| packet d'octet (void*) | MPI_PACKED |

Les tags des messages MPI

- Tous les messages envoyés sont accompagnés d'une étiquette ou tag (*valeur entière*) qui sert au récepteur à identifier un message
- Le processus récepteur doit spécifier qu'il attend la réception d'un message en précisant un tag :
 1. En donnant une valeur entière
 2. En donnant la valeur `MPI_ANY_TAG` qui correspond à toute les valeurs

5. Modèles de programmation parallèle

La primitive de base : MPI_Send

• *MPI_Send (buffer, count, datatype, dest, tag, comm)*

Envoi un message de donnée (*buffer, count, datatype*) au noeud de rang *dest*, avec le tag *tag* dans le communicateur *comm*.

• **Spécification MPI : quand cette fonction retourne**

- la donnée peut alors être réutilisée
- la donnée a été communiquée à MPI pour envoi,
- MAIS la donnée n'est peut être pas encore reçue par le processus récepteur

• **Entre deux processus, les communications dans MPI sont FIFO**

5. Modèles de programmation parallèle

Exemple : MPI_Send

-Exemple : envoi d'un entier au processus de rang 0

1. `int integer = 31415;`
2. `int tag = 37;`
3. `MPI_Send(&integer, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);`

-Exemple : envoi d'un tableau de « float »

1. `float array [256]; /* plus initialisation */`
2. `int tag = 39;`
3. `MPI_Send(array, 128, MPI_FLOAT, 0, tag, MPI_COMM_WORLD);`

5. Modèles de programmation parallèle

La primitive de base : MPI_Recv

MPI_Recv (buffer, count, datatype, dest, tag, comm, status)

Réception un message de donnée (*buffer, count, datatype*) provenant du noeud de rang *dest*, avec le tag *tag* dans le communicateur *comm*, l'état de la réception est stocké dans *status*

Spécification MPI : quand cette fonction **retourne**

- La donnée est reçue et peut alors être utilisée
- Le tableau reçu peut être de taille moindre que *count*
- Le nombre d'éléments du tableau reçu est stocké dans *status*

5. Modèles de programmation parallèle

Exemple : MPI_Recv

-Exemple : réception d'un entier du noeud de rang 2

1. `int integer;`
2. `int tag = 37;`
3. `MPI_Status status;`
4. `MPI_Recv(&integer, 1, MPI_INT, 2, tag, MPI_COMM_WORLD, &status);`

-Exemple : réception d'un tableau de « float »

1. `float array [256];`
2. `int tag = 39;`
3. `MPI_Recv(array, 128, MPI_FLOAT, 2, tag, MPI_COMM_WORLD, &status);`

5. Modèles de programmation parallèle

Modes de Communication

1. *standard* : l'utilisateur peut modifier la donnée dès le retour du sous-programme. Il est à la charge de MPI d'effectuer ou non une copie temporaire du message. Si c'est le cas, l'envoi se termine avant que la réception ne soit postée.

2. *synchronous* : l'envoi du message ne se termine que si la réception a été postée et la lecture du message terminée. C'est un envoi couplé avec la réception.

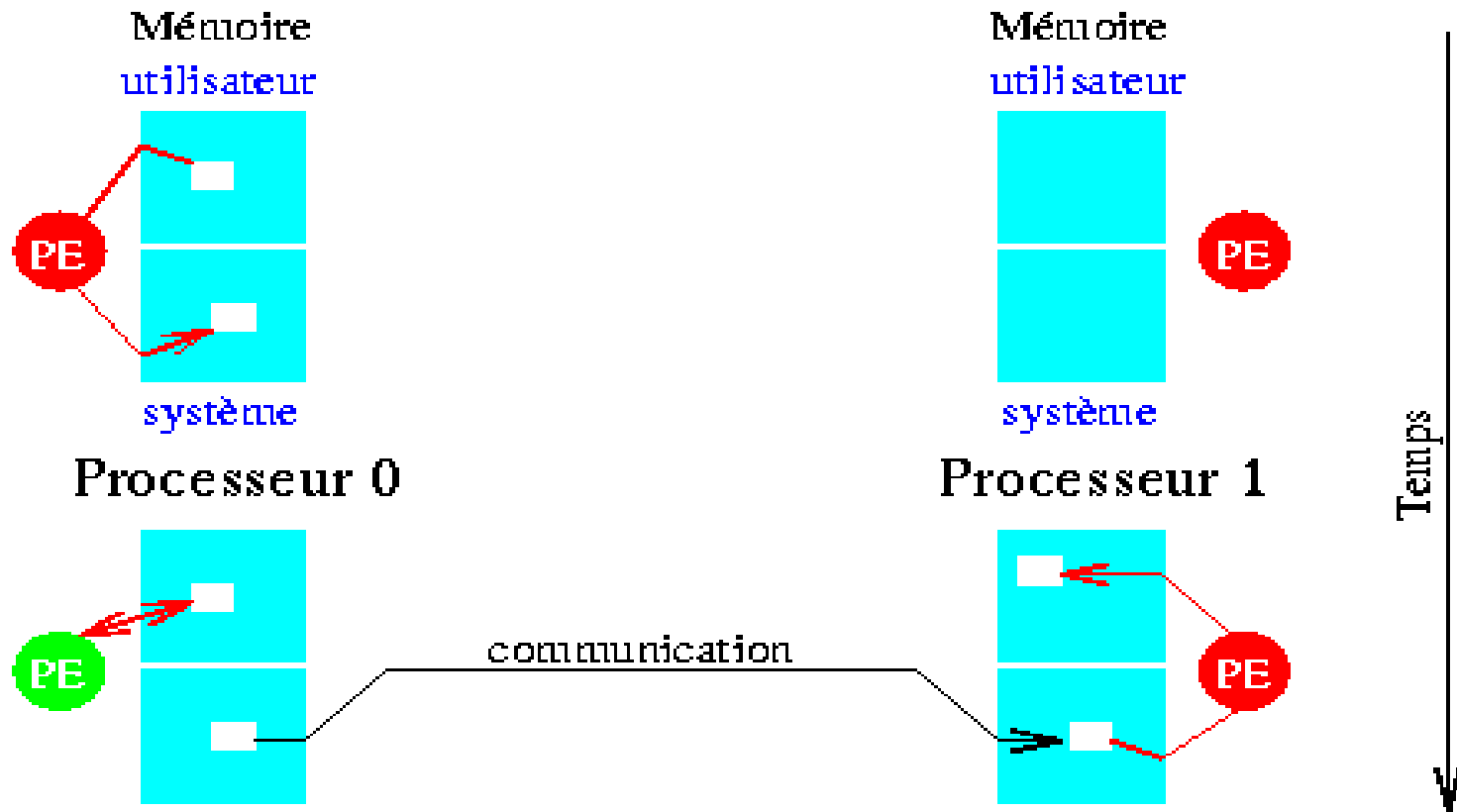
3. *buffered* : il est éventuellement à la charge du programmeur d'effectuer une copie temporaire du message. L'envoi est ainsi découplé de la réception.

4. *ready* : l'envoi du message ne peut commencer que si la réception a été postée auparavant (ce mode est intéressant pour les applications clients-serveurs).

5. Modèles de programmation parallèle

Exemple

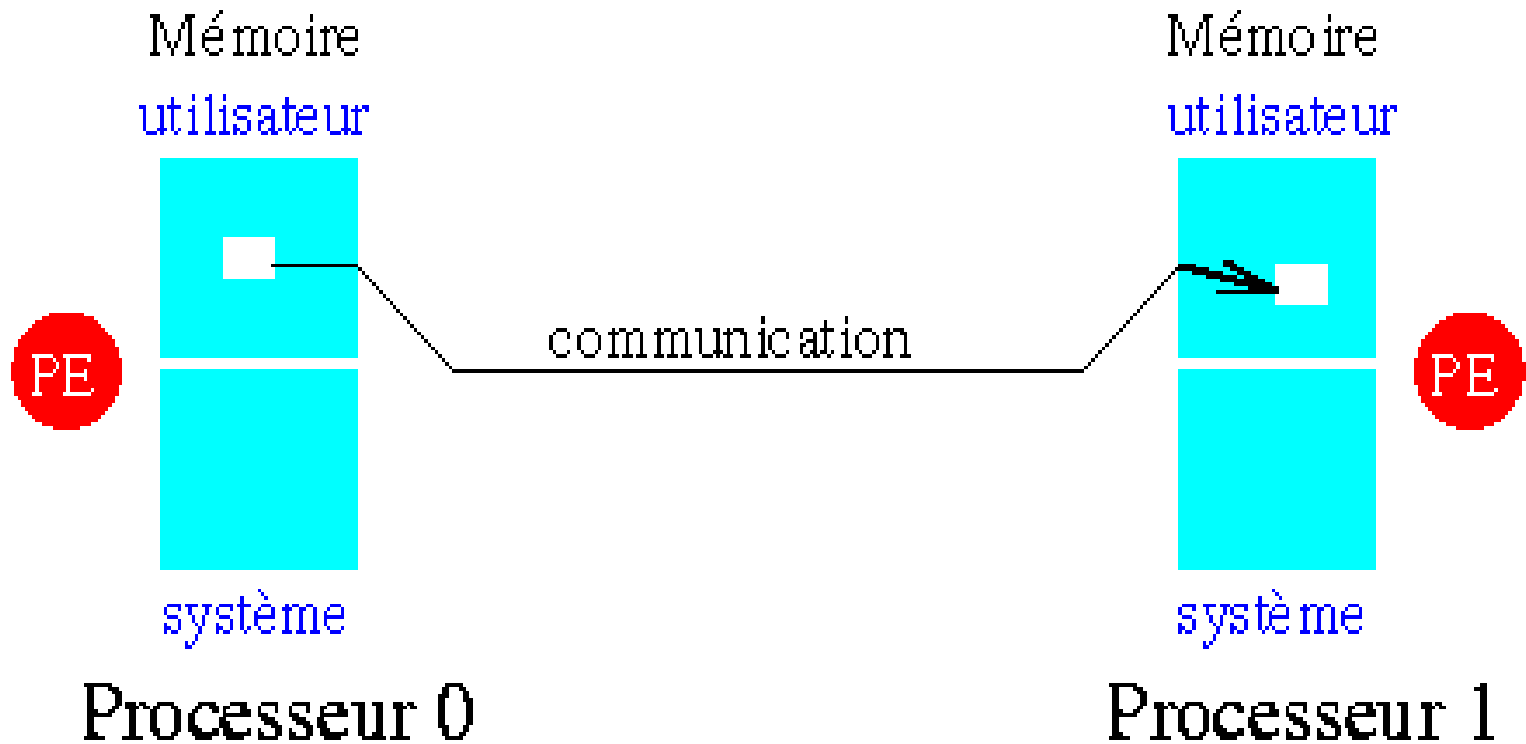
Envoi non bloquant avec copie temporaire du message : MPI_IBSEND()



5. Modèles de programmation parallèle

Exemple

Envoi synchrone bloquant : MPI_SSEND()



5. Modèles de programmation parallèle

- OpenMP

<http://www.openmp.org>

- Modèle de programmation parallèle standard pour machines MIMD-SM
- Portabilité des programmes sur les machines MIMD- SM
- Étendre C, C++ et Fortran par un ensemble de primitives et de directives de parallélisme SPMD, partage et privatisation de données, de synchronisation ...

5. Modèles de programmation parallèle

- Exemple:

```
...  
#pragma parallel  
#pragma local (i)  
#pragma shared (a,b,c)  
#pragma pfor  
for (i=1;i<=1000;i++)  
{  
    b(i) = i*i;  
    a(i) =c(i)- 2*b(i);  
}  
...
```

PLAN

- 1. Historique**
- 2. Classification**
- 3. Concepts de base de l'algorithmique parallèle**
- 4. Les sources du parallélisme**
- 5. Modèles de programmation Parallèle**
- 6. Techniques de Parallélisation**

6. Techniques de Parallélisation

1. Parallélisation des instructions simples

→ Problème de dépendance : 3 types de dépendances

- **Dépendance de flux (ou producteur consommateur)**

S1: $x := y + 1$

S2: $z := x * b$

on note $S1 \delta S2$

- **Antidépendance (ou consommateur producteur)**

S1: $y := x + 3$

S2: $x := z * b$

on note $S1 \overline{\delta} S2$

- **Dépendance de sortie (ou producteur producteur)**

S1: $x := c + d$

S2: $x := z * b$

on note $S1 \delta^\circ S2$

6. Techniques de Parallélisation

- **Relation de Bernstein**

Deux instructions S1 et S2 sont en dépendance

\Leftrightarrow

$$(E1 \cap S2) \cup (S1 \cap E2) \cup (S1 \cap S2) \neq \emptyset$$

E_i = ensemble des entrées de l'instruction S_i

S_i = ensemble des sorties de l'instruction S_i

6. Techniques de Parallélisation

2. Parallélisation des nids de boucles

Objectif :

Paralléliser les différentes itérations d'un nid de boucles

Méthodologie:

- * Analyse des dépendances
- * Restructuration du programme initial

6. Techniques de Parallélisation

Exemple: (distribution de boucles)

```
Do i = 1,n
S1:   a(i) = 2*b(i)
S2:   c(i) = d(i)*a(i+1)
End Do
```

S2(i) δ S1(i+1)



```
Doall i = 1,n
  c(i) = d(i)*a(i+1)
End Doall
Doall i = 1,n
  a(i) = 2*b(i)
End Doall
```

6. Techniques de Parallélisation

Exemple:

```
Do i = 1,n
S1:    a(3*i-1) = 2*b(i)
S2:    c(i) = d(i)*a(6*i+1)
End Do
```

$3*i - 6*j = 2$ n'admet pas de solutions

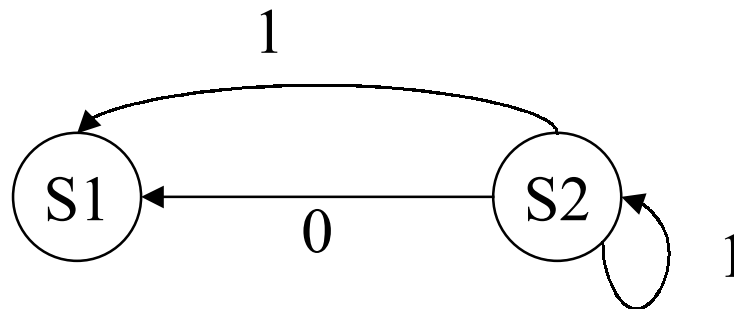


```
Doall i = 1,n
S1:    a(3*i-1) = 2*b(i)
S2:    c(i) = d(i)*a(6*i+1)
End Doall
```

6. Techniques de Parallélisation

Exemple: (Allen et Kennedy)

```
Do i = 1,n
  Do j = 1,m
S1:      a(i,j) = b(i,j)
S2:      b(i,j) = b(i,j) + a(i,j-1)
  End Do
End Do
```



Graphe de dépendance
réduit

6. Techniques de Parallélisation

Résultat:

```
DOALL i = 1,n
  DOALL j = 1,m
    a(i,j) = b(i,j)
  ENDDOALL
  DOALL j = 1,m
    b(i,j) = b(i,j) + a(i,j-1)
  ENDDOALL
ENDDOALL
```

6. Techniques de Parallélisation

- Le modèle polyédrique :

Principe

Programme source



Polytope source



Polytope destination



Programme destination

Transformation espace/temps

6. Techniques de Parallélisation

- Exemple

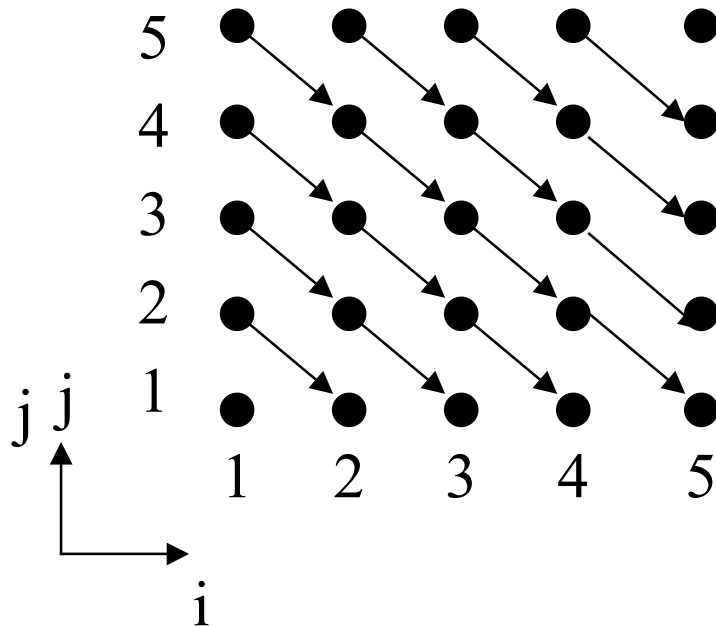
Do i = 1,5

Do j = 1,5

$a(i,j)=a(i,j)*a(i+1,j-1)$

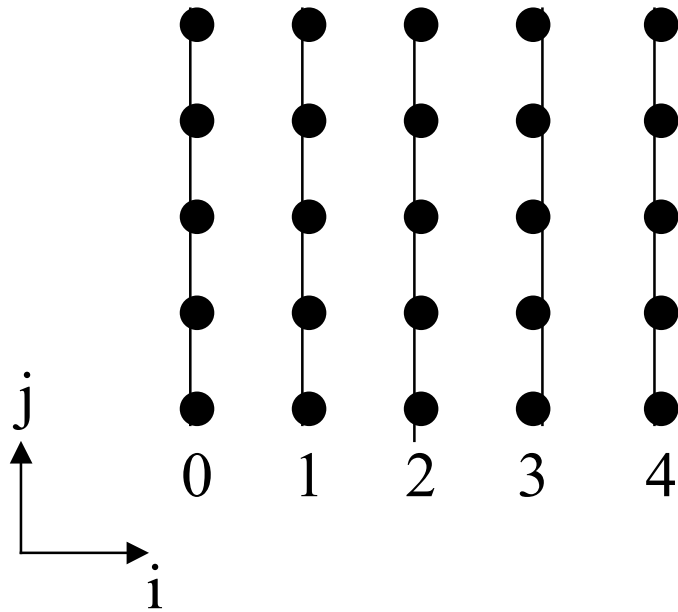
End Do

End Do

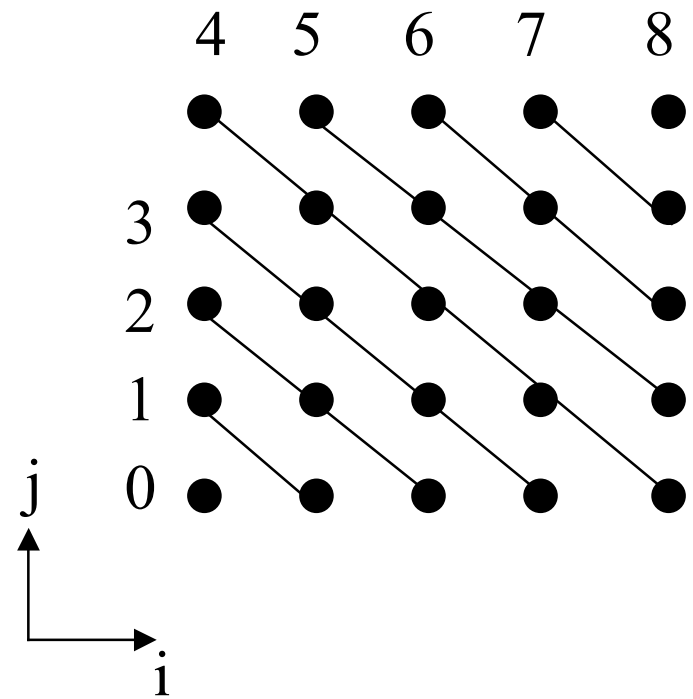


6. Techniques de Parallélisation

$$t(i,j) = i$$



$$p(i,j) = i+j$$



6. Techniques de Parallélisation

Polytope source : $Ax \leq b$

$$\begin{array}{c} \left| \begin{array}{cc} -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{array} \right| \quad \left| \begin{array}{c} i \\ j \end{array} \right| \leq \left| \begin{array}{c} 0 \\ 0 \\ 4 \\ 4 \end{array} \right| \\ \\ T = \left| \begin{array}{cc} 1 & 0 \\ 1 & 1 \end{array} \right| \quad T^{-1} = \left| \begin{array}{cc} 1 & 0 \\ -1 & 1 \end{array} \right| \end{array}$$

Polytope destination : $AT^{-1}y \leq b$

$$\begin{array}{c} \left| \begin{array}{cc} -1 & 0 \\ 1 & -1 \\ 1 & 0 \\ -1 & 1 \end{array} \right| \quad \left| \begin{array}{c} t \\ p \end{array} \right| \leq \left| \begin{array}{c} 0 \\ 0 \\ 4 \\ 4 \end{array} \right| \quad \begin{array}{l} (0 \leq t \leq 4) \\ (t \leq p \leq t+4) \end{array} \end{array}$$

6. Techniques de Parallélisation

Programme destination:

$(i,j) = T^{-1}(t,p)$

DO t=0,4

 DOALL p = t, t+4

$a(t,p-t) = a(t,p-t) * a(t+1,p-t-1)$

 END DOALL

END DO

6. Techniques de Parallélisation

