

Introduction à la programmation des ordinateurs parallèles

Ronan.Keryell@enst-bretagne.fr

—

Département Informatique

École Nationale Supérieure des Télécommunications de Bretagne

DEA Rennes 1 — ENSTBr

ISIA — ENSMP

21-24 mars 2005

Version 1.6

Extraire du parallélisme

1

Exemple de calcul de polynômes de vecteurs (livre « Initiation au parallélisme, concepts, architectures et algorithmes », Marc , Stéphane Ubéda & Frédéric Desprez)

```
pour i = 0 à n - 1 faire
    vv[i] = a + b.v[i] + c.v[i]2 + d.v[i]3 + e.v[i]4 + f.v[i]5 + g.v[i]6
fin pour
```

Calcul avec parallélisme de donnée (typique SIMD) \equiv faire en parallèle la même chose sur des données différentes :

```
pour i = 0 à n - 1 faire en parallèle
    vv[i] = a + b.v[i] + c.v[i]2 + d.v[i]3 + e.v[i]4 + f.v[i]5 + g.v[i]6
fin pour
```



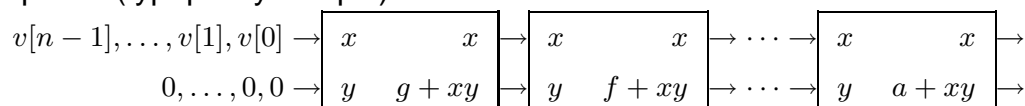
Découpage en tâches (typique MIMD) \equiv faire des choses différentes sur des données différentes :

```

pour  $i = 0$  à  $n - 1$  faire
  tâches parallèles
     $x = a + b.v[i] + c.v[i]^2 + d.v[i]^3$ 
    ||
     $y = e. + f.v[i] + g.v[i]^2$ 
    ||
     $z = v[i]^4$ 
  fin tâches parallèles
   $vv[i] = x + z.y$ 
fin pour
    
```



Pipeline (typique systolique) \equiv travail à la chaîne :



7 étages de pipeline (7 processeurs) traitant 1 flux de plusieurs données.



Parallélisme de données :

- Régularité des données
- Même calcul à des données distinctes

Parallélisme de contrôle :

- Fait des choses différentes

Parallélisme de flux : pipeline

- Régularité des données
- Chaque donnée subit séquence de traitements



Grain du parallélisme

Taille moyenne des tâches élémentaires (en nombre d'instruction, taille mémoire, temps)

Gros grain → grain fin :

- ▶ Programmes
- ▶ Procédures & fonctions
- ▶ Instructions
- ▶ Expressions
- ▶ Opérateurs
- ▶ Bits

Choix en relation avec l'architecture cible



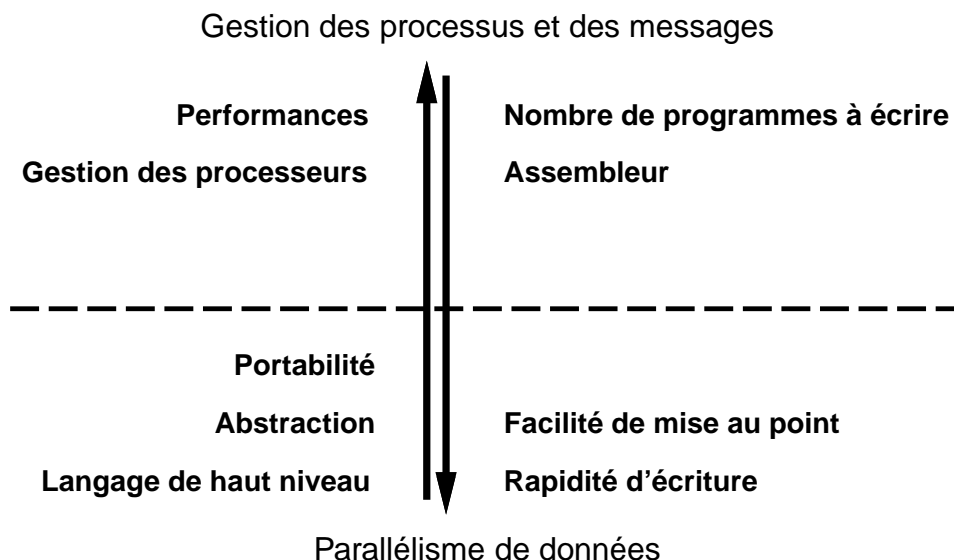
- Nombre d'opérations exécutées simultanément
- Idée du nombre de processeurs nécessaires
- Si moins de processeurs, repliement du parallélisme
- Peut varier au cours de l'exécution
- Si trop de processeurs : inactivité ↗



Principalement 2 niveaux :

- Gestion totale du parallélisme « à la main » au niveau des processeurs
- Langage de haut niveau abstrayant le parallélisme de la machine





Mode de couplage avec la mémoire

Rêve de l'utilisateur λ : ne voir qu'une grosse machine SISD
 \rightsquigarrow énorme mémoire globale.

Problèmes (à défaut de SISD...) :

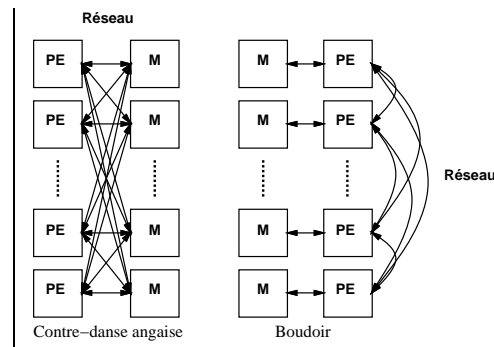
- sortance des portes logiques bornées (temps en $\mathcal{O}(\mathcal{M})$) = :
amplifications exponentielles nécessaires (temps en $\mathcal{O}(\log \mathcal{M})$)
- temps de propagation de l'ordre de $\frac{2}{3}c$ dans les fils
- on peut augmenter le débit mais pas diminuer la latence
- conflits d'accès à une même case par plusieurs PES



Solutions possibles :

- exploiter la localité des accès
- couper la mémoire en morceau

Mais comment relier processeurs et mémoire(s) ?



Couplage fort contre couplage faible.



Couplage fort \equiv mémoire partagée

Avantages :

- plus simple à programmer (mais non sans surprise...)
- logiciel plus simple : chaque PE voit toutes les autres mémoires
- découplage entre nombre de PEs et nombre de mémoires
 - ▶ beaucoup de bancs \rightsquigarrow fort débit
 - ▶ nombres premiers pour éviter des conflits, etc.
- mémoire maximale vue par **chaque** processeur, utile pour applications demandant beaucoup de mémoire



Inconvénients :

- optimisation du cas pire : « on ne sait rien et on fait le maximum »
- matériel complexe car débit élevé en continu
- cohérence mémoire délicate si cache
- difficile si beaucoup de PES

Mémoire souvent réalisée de manière structurée : bancs.

Pour grosses machines vectorielles ou les multiprocesseurs (mainframes ou stations de travail), BSP, à faible nombre de processeurs : CRAY Y-MP C916 : 1024 bancs pour 250 Go/s



Avantages :

- optimisation du meilleur cas : « on sait tout et on fait le minimum », en ayant tout localement
- rapide si placement des données correct
- « compilation » des communications : cache logiciel, fusion de messages
- assez simple même si beaucoup de PES



Inconvénients :

- si accès très dispersés : lent
- placement (distribution et alignement) important
- temporaires pour stocker les données communiquées
- nécessite un logiciel plus complexe.

Toutes machines SIMD sauf BSP : CM-2, MP-1, ILLIAC IV, OPSILA.

Machines MIMD avec beaucoup de PES : DELTA, CM-5, s.



Placement en mémoire

Éviter les conflits sur des accès typiques (lignes, colonnes, diagonales si possible) :

- $(i, j) \longrightarrow (ui + vj) \bmod N$ BSP
- $(i, j) \longrightarrow i \oplus j \bmod N$ STARAN
- $(i, j) \longrightarrow AI \oplus BJ \bmod N$ XOR scheme
- carrés magiques $N \times N$ avec des nombres de 1 à N

Problème : temps de calcul \rightsquigarrow faire simple...



MP : Multi-Processeur... à mémoire partagée $((MP)^2)$.

Au niveau système :

SMP

- : symétrique
Chaque processeur peut accéder au matériel d'E/S
(sémaphores...)

AMP

- : asymétrique
Seul 1 processeur accède au matériel. Les autres sous-traitent

SMP plus compliqué (SunOS 5) mais plus efficace que l'AMP
(SunOS 4) pour les E/S.



Le programmeur & la mémoire

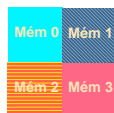
17

Multiplication de matrice :

$$B = A \times A$$

Modèle d'exécution et couplage mémoire

A & B



- Mémoire partagée :
- Mémoire distribuée sur les PEs :

Temporaires



Communication dans 1 temporaire (voire 2...) nécessaire pour
des calculs répartis aussi par bloc.



- Détourner chaque accès mémoire et éventuellement faire une communication : *run-time resolution*
Solution de dernier recours car très lente...
- Bien délimiter et isoler les zones à problèmes : insérer les communications pour les données qui manquent en local.
- Gérer le problème au niveau page du système d'exploitation (MMU pour faire de la mémoire virtuellement partagée) \rightsquigarrow transparent au programmeur
Problème lors du partage des pages en écriture : chaque PE va transférer la page lors de chaque écriture \rightsquigarrow ping-pong de page pour 1 octet...

De manière générale, plus le problème est irrégulier, les communications aussi...



Programmation SPMD pour mémoire distribuée

- But : programme qui s'exécute sur plusieurs processeurs
- Pas d'accès simple aux données des autres processeurs
- Distribuer les données et faire suivre les calculs pour exprimer du parallélisme
- Simplification possible : règle des écritures locales (*owner-compute rule*)
 $A = F(B, C)$
F est calculé sur le nœud de la machine où est stocké A



```
forall( $i, \tilde{L}i \leq 0$ )
     $X(\tilde{S}_{Xi}) = f(Y(\tilde{S}_{Yi}), Z(\tilde{S}_{Zi}), \dots)$ 

devient

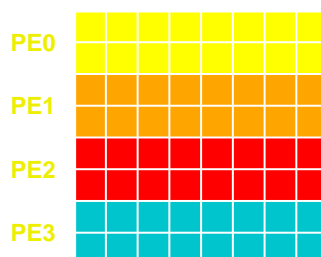
prog(p)
forall( $i, \tilde{L}i \leq 0$ )
    send $_{Y(\tilde{S}_{Yi})}(p)$ 
    send $_{Z(\tilde{S}_{Zi})}(p)$ 
    receive $_{Y(\tilde{S}_{Yi})}(p)$ 
    receive $_{Z(\tilde{S}_{Zi})}(p)$ 
    if own $_{X(\tilde{S}_{Xi})}(p)$ 
         $X(\tilde{S}_{Xi}) = f(Y(\tilde{S}_{Yi}), Z(\tilde{S}_{Zi}), \dots)$ 
```



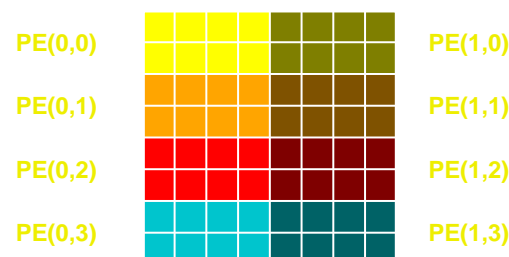
Distribution par bloc

21

Exemple de tableau 2D



Bloc 1D



Bloc 2D

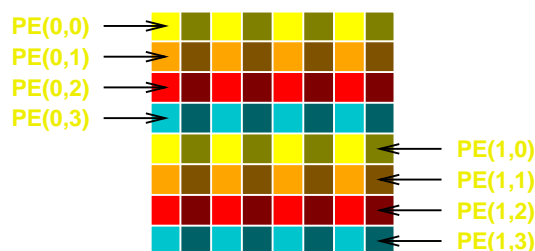
- Minimise les communications si accès de voisinage : calcul de différences finies,...



Exemple de tableau 2D



Bloc 1D



Bloc 2D

- Favorise la répartition de la charge au dépend des communications de voisinage : calcul de fractales,...

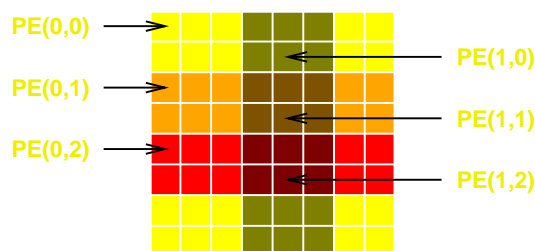


Distribution bloc-cyclique

Exemple de tableau 2D



Bloc 1D



Bloc 2D

- Compromis entre les 2 approches précédentes : calcul sur des matrices triangulaires,...
- Tous les mélanges sont possible par dimension de processeur, y compris la non distribution



Problème : faire la différence entre une donnée locale et une donnée distante \rightsquigarrow code compliqué et lent

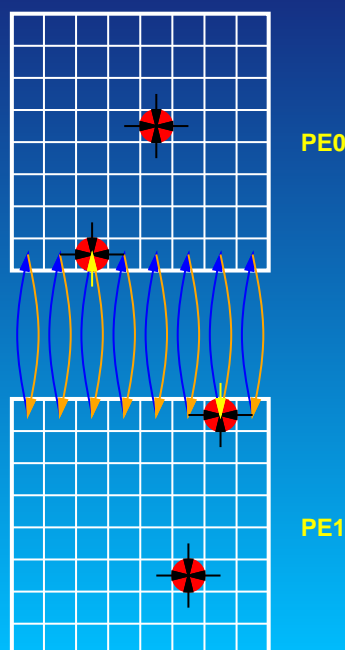
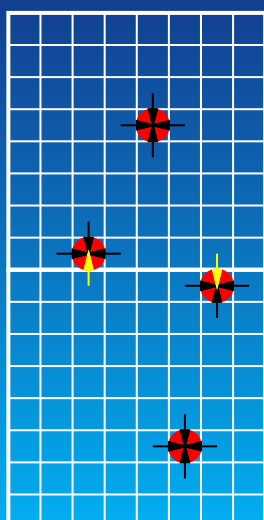
- Idée : agrandir chaque zone locale pour stocker les données distantes nécessaires
- Rajouter des communications pour récupérer les valeurs des bords
- Éventuellement recouvrir calcul sur l'« intérieur » avec communication sur les bords
- Possibilité de diminuer les communications en faisant du calcul redondant : étendre le domaine de recouvrement d'autant

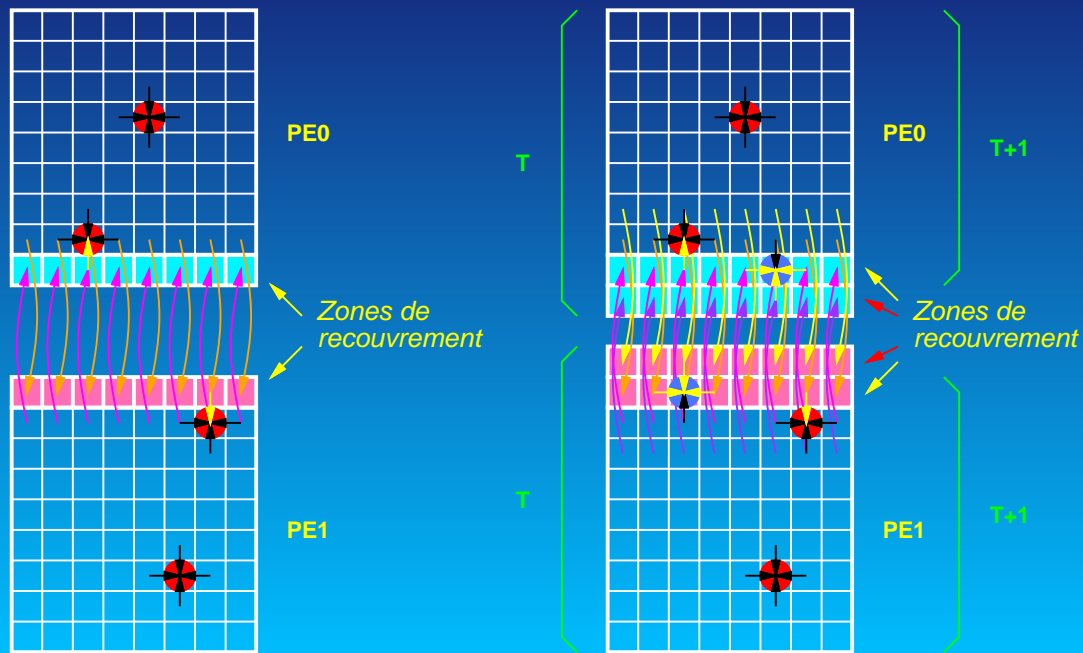
Exemple : $x_{i,j} = (y_{i-1,j} + y_{i+1,j} + y_{i,j-1} + y_{i,j+1})/4$ distribué en bloc sur 2 processeurs



Distribution sans recouvrement

25





Langages à passage de messages

- Définition de processus communicants
- Bibliothèques pour échanger des données entre processus
- Parallélisme de contrôle

Langages à parallélisme de données

- Opérations qui s'appliquent à un ensemble de données (addition de matrice, produits scalaires,...)
- plus haut niveau d'abstraction (pas de processus,...)
- Manipulation d'objets familiers : tableaux



- UNIX : |, ssh, socket, NFS, mémoire partagée, mmap(),...
- CSP, OCCAM (processus communicants)
- ADA
- Java
- Langages classiques (C, Fortran) avec bibliothèques de communication (MPI, PVM, P4, NX/2,...)
- Classes C++ pour camoufler les messages



- Langage + beaucoup de *packages* incluant le parallélisme entre autre
- Parallélisme de tâche/contrôle
`new class(...).start()`
`Runtime.getRuntime().exec(...)`
RMI : Remote Method Invocation
Ex : les threads en JDK1.2 avec Solaris 2.6 s'exécutent sur différents processeurs
- Mémoire partagée : accès à des variables visibles en commun
- Mémoire distribuée : communication \equiv envoyer un objet `Serializable` ou `Externalizable` dans un Stream avec *pipe* ou *socket*



- Fortran 90 : extension aux opérations sur les tableaux
- CM-Fortran (TMC), MPP-Fortran (Cray), HPF,... : directives de placement
- C* (TMC), MPL (MasPar), PompC, HyperC,... : directives de placement



Traducteurs

Facilitent la vie du programmeur :

- Vectoriseur : transforme un programme « séquentiel » (Fortran) en programme avec des instructions vectorielles
- Paralléliseur : transforme un programme « séquentiel » (Fortran) en programme(s) pour machine parallèle

Évite le parallélisme explicite mais pas toujours efficace.

Nécessite de l'aide... Directives (Cray, HPF,...).



- Réseaux de la recherche rapide (RENATER, VTHD++, ABILENE,...)
- Mondialisation des infrastructures
- Exploiter les « jachères de calcul » disponibles
- Protocoles plus sexy et interopérables (XML)
- Mode des Web services

~> Grilles de calcul



- Besoins de stocker et partager des données sur plusieurs sites
- Recherche et indexation
- Multiprotocole (HTTP vu comme un pareparefeu...)
- Identifiants uniques indépendant des protocoles ou adresses (NAT,...)
- Construction d'une infrastructure entre pairs au dessus du réseau réel
 - ▶ Nœuds de routage
 - ▶ Nœuds de stockage
 - ▶ Mandataires (*proxy*)
- Exemple : JXTA de Sun en Java, OceanStore,...



Majorité des ordinateurs de la planète ne font qu'attendre l'utilisateur... ☺↔ Des PFLOPS potentiels gratuits

- SETI@home : remplace les économiseurs (useurs ?) d'écran pour rechercher de signaux extra-terrestres
- Décryphon : bio-informatique
- Concours de cassage d'algorithmes de cryptographie



Grilles de calcul

- Idée offrir des ressources de calcul semblable au service d'électricité
- Faire des requêtes de ressources à un supercalculateur virtuel
- Mutualisation des ressources pour utiliser des puissances de calcul inabordable autrement
- Exemple : Globus2 vu comme un service Web
- Problèmes : latences considérables, débits souvent faibles, identifications globales (PKI),...
- Besoin d'une algorithmique adaptée à très gros grain : couplage de code différents (transport + modèle biologique,...)



Liste des transparents

- 0 Titre
- 1 Extraire du parallélisme
- Introduction**
- 4 Type de parallélisme
- 5 Grain du parallélisme
- 6 Degré du parallélisme
- 7 Programmation des machines parallèles
- Programmation**
- 8 Compromis à trouver...
- 9 Mode de couplage avec la mémoire
- La mémoire**
- 11 Couplage fort \equiv mémoire partagée
- 13 Couplage faible \equiv mémoire distribuée
- 15 Placement en mémoire
- 16 SMP & AMP
- 17 Le programmeur & la mémoire

- 18 Simulation d'une mémoire partagée
- 19 Programmation SPMD pour mémoire distribuée
- 21 Distribution par bloc
- 22 Distribution cyclique
- 23 Distribution bloc-cyclique
- 24 Recouvrements (overlaps)
- 25 Distribution sans recouvrement
- 26 Distribution avec recouvrement
- 27 Langages parallèles
- 27 Langages à passage de messages
- 27 Langages à parallélisme de données
- Langages**
- 28 Exemples de langages à parallélisme de contrôle
- 29 Java
- 30 Exemples de langages à parallélisme de données
- 31 Traducteurs
- 32 Passage à l'échelle
- 33 Stockage pair à pair
- 34 Calcul distribué mondial en client-serveur
- 35 Grilles de calcul