Programmation parallèle
Cours 2 : modèles

Ahmad AUDI
ahmad.audi@ign.fr

Granularité – degré de parallélisation

- La **granularité** correspond à la taille des taches effectuées en parallèle
  Il existe trois types de granularité : fine, moyenne, grosse.
  - Gros grain de calcul
  - Grain de calcul moyen
  - Grain de calcul fin

- Le degré de parallélisation correspond aux nombres d'opérations que l'on peut effectuer en parallèle.
  - Plus le degré est important, plus on peut accélérer le calcul.
  - ATTENTION : le degré n'est pas constant au cours du déroulement d'un programme

Exemple

- Le produit de deux matrices de taille `N` : `C = A*B`

```
for i=1 to N
    for j=1 to N
       S=0
       for k=1 to N
           S=S+A(i,k)*B(k,j)
       C(i,j)=S
```

- Dans la multiplication de matrice, on peut paralléliser :
  - La boucle for `k=1` to `N` → granularité fine
  - La boucle for `j=1` to `N` → granularité moyenne
  - La boucle for `i=1` to `N` → granularité grosse

- Le choix de la méthode est fait en fonction de l'architecture de la machine et des performances des communications.

Un exemple de parallélisme de données

```
pour i=1,n
  parbegin
    a(i)=f(i) // calcul f(i) independant du tableau a
  parend
finpour
Pour i=1,n
  parbegin
    b(i)=a(i)+a(n-i+1)
  parend
finpour
```

Exemple (suite)

- Chacune des deux boucle peut être effectuée en parallèle mais la 1ère doit être terminée avant le début de la 2ème
- La deuxième boucle pose le problème de l'accès à `a(i)` et `a(n-i+1)`
- Ex : pour `i=1,b(1)` est calculé à partir de `a(i)` et `a(n)`
- Sur une machine SM, la programmation ne pose pas de problème
- L'écriture du programme correspond à un programme pour une machine SM. Le compilateur gère la répartition des données

Exemple (suite)

- Pour les machines DM, la première boucle se parallélise bien. Les données sont partagées et envoyées à chaque processeur.

- Avec par exemple une distribution par bloc, le code devient avec p processeurs numéroté de `0` à `p-1`, `p` diviseur de `n` :

```
//reception des donnes
...
TailleBloc = n/p
Pour i=1,TailleBloc
    a(i)=f(NumProc*TailleBloc+i)
finpour
```

Exemple (suite)

- Par contre, la 2ème boucle pose des problèmes dues au calcul de `a(i)+a(n-i+1)` :
- Solutions :
  1. Récupération par une communication de `a(i)+a(n-i+1)`, bilan :
     Un calcul = une communication

  2. Le processeur k reçoit (avant le calcul) la partie de `a` du processeur `p-k-1` et lui envoie sa partie de `a`
     modification importante du programme

Distribution des données : cas 1D

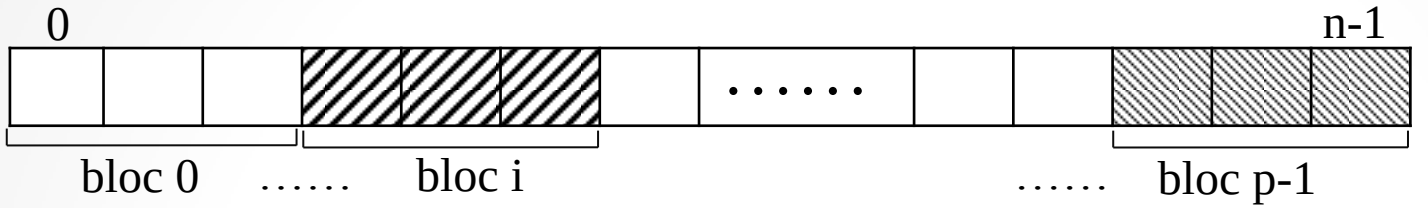
- La régularité des données permet de les distribuer facilement.
  Exemple : tableau `n×n` partagé sur `p` processeurs, chacun recevant `n/p`

- Les schémas de distribution les plus courant sont :
  - Distribution par bloc
  - Distribution cyclique (modulo)
  - Distribution par bloc cyclique

- Soit `t(n)`, un tableau de taille `n` et `p` le nombre de processeurs (`p` diviseur de `n`)
  Chaque processeur $(P_i)_{0 \leq i \leq p-1}$ reçoit des données $\{d_j\}_{0 \leq j < n}$

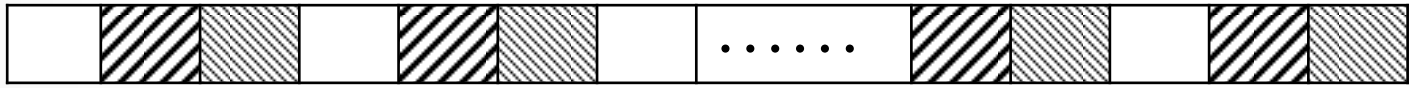
Distribution des données : cas 1D (suite)

- Par bloc :



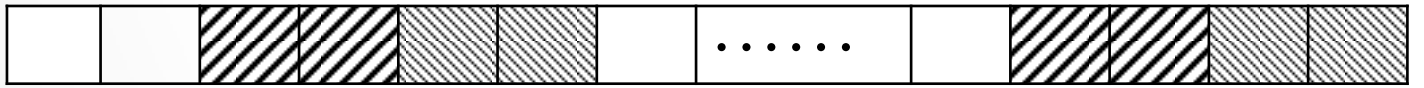
Le bloc i contient les données $i*(n/p) \leq j < (i+1)*n/p$

- Cyclique :



Le bloc i contient les données $j \bmod p = i$

- Par bloc cyclique :

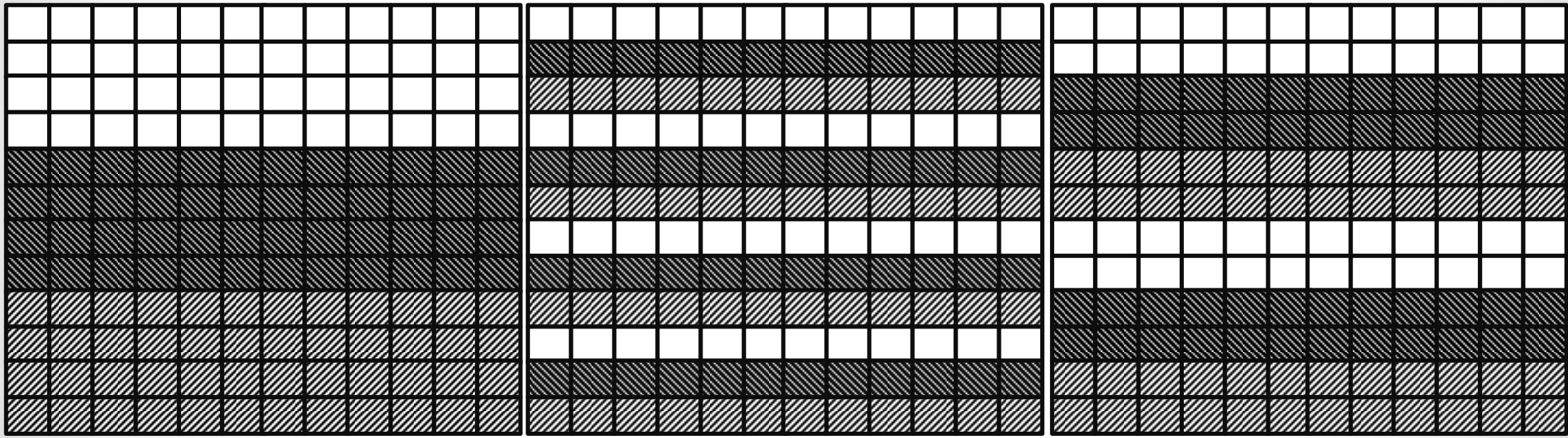


Le bloc i contient les données $j/b \bmod p = i$

Répartition par bloc de taille b, b diviseur de n/p

Distribution 1D de données 2D

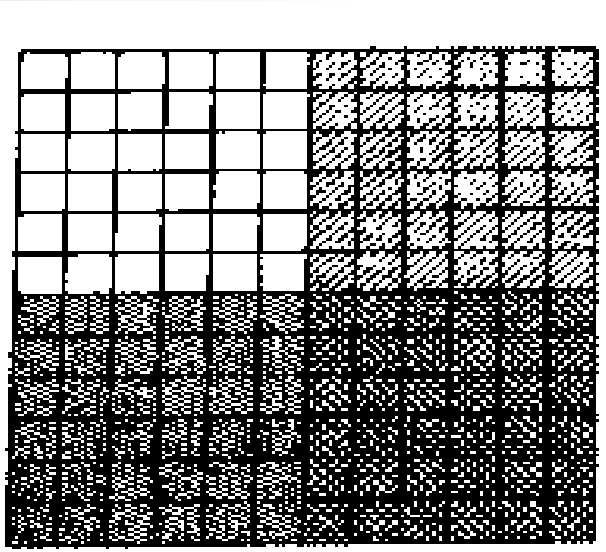
processeurs $(P_i)_{0 \leq i \leq p-1}$



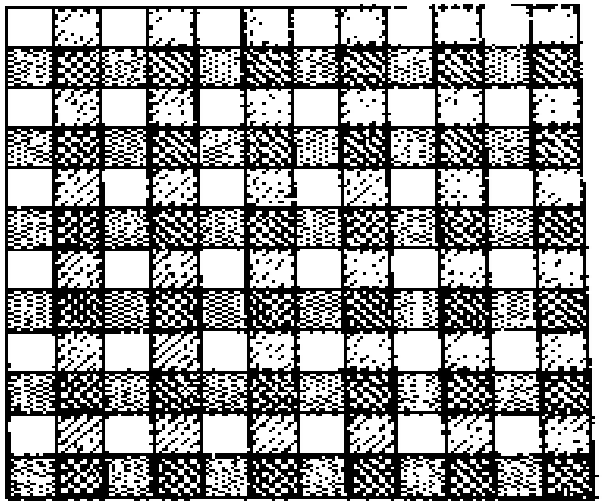
Bloc | Cyclique | Bloc cyclique

Distribution 2D de données 2D

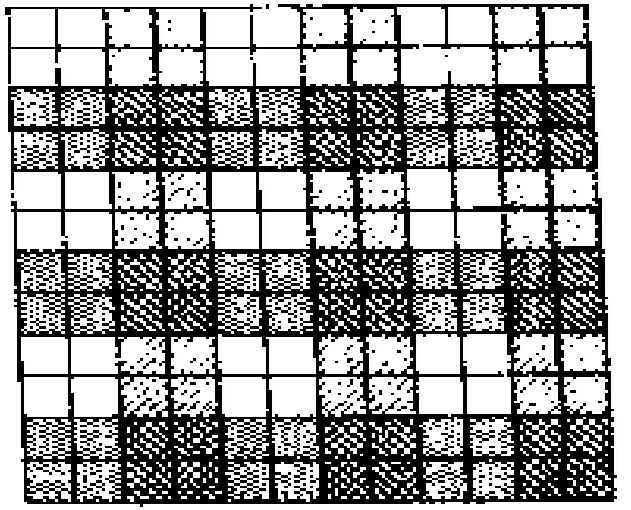
processeur $(P_i)_{0\leq i \leq q-1, 0 \leq i \leq q-1}$ (avec $q^2 = p$)



Bloc



Cyclique



Bloc cyclique

Comment obtenir un programme parallèle efficace ?

Les grands principes :

- **Localité des données** : *répartir les données de sorte que chaque processeur dispose localement d'un maximum de données à traiter*
  → très important pour machines DM : réduction des communications

- **Équilibrage de charge** *(load balancing) : attribuer au mieux les charges de calcul en fonction des caractéristiques de chaque processeur, afin de limiter les périodes d'inactivité des processeurs*
  → machines homogènes : la charge de calcul doit être la même pour chaque processeur

- **Recouvrement des communications par le calcul**

Equilibrage de charge

Charge de calcul prédicable → équilibrage de charge statique

- Données réguliers présentant toutes un même cout de calcul
  → distribution bloc, cyclique, bloc cyclique
- Données régulières présentant des coûts de calcul différents
  → utilisation d'une fonction de Coût + distribution boc, cyclique, bloc cyclique …

Charge de calcul non prédicable → équilibrage de charge dynamique
(exemple : fractale de Mandelbrot)

- Modèle maître-ouvrier
- Modèle auto-régulé

Modèles de conception

Fork-join
Parbegin-parend

- Le paradigme «fork» permet de scinder un fil d'exécution en deux
  - un fil parent
  - un fil enfant
- Le parent peut ensuite attendre la fin de l'exécution de son enfant à l'aide du «join»
- Le «parbegin» / «parend» est similaire, mais avec plusieurs fils d'exécution
- Utilisé dans OpenMP

Modèles de conception

Single Program Multiple Data (SPMD)

- Modèle SIMD : un seul programme appliqué sur de multiples données
- La différence avec le SIMD pure est que les fils d'exécution sont asynchrones au lieu de synchrones
  - implique souvent le besoin d'utiliser des mécanismes de synchronisation
- Approche populaire utilisée par les programmes MPI
  - passage de messages

Modèles de conception

Pipeline

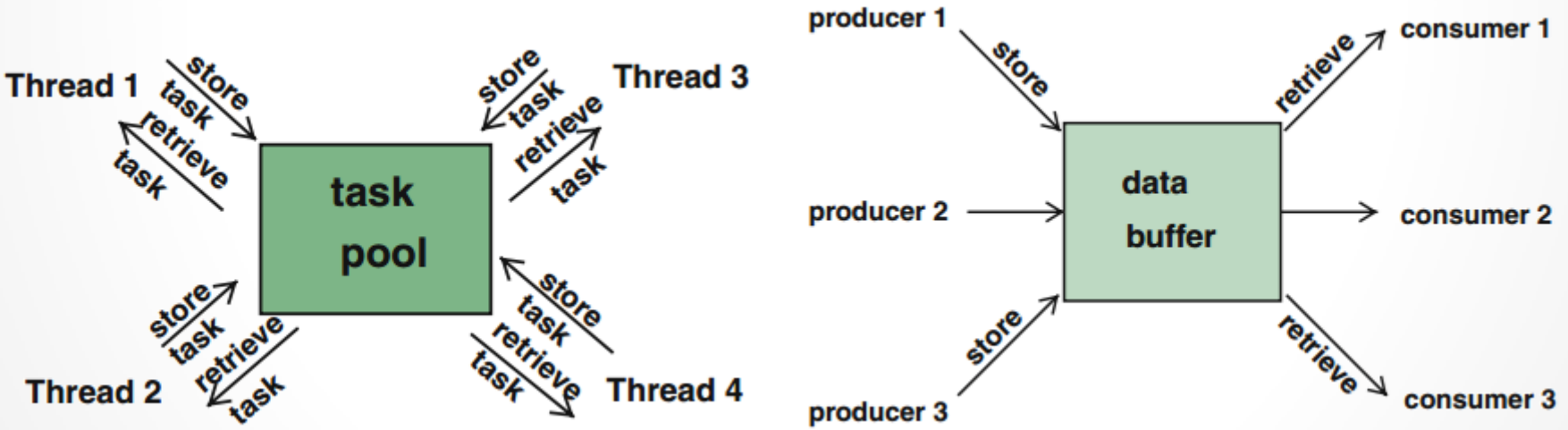
- Comme pour les architectures matérielles, la sortie d'un fil d'exécution peut servir d'entrée à un autre fil d'exécution
  - en enfilant plusieurs fils de la sorte, on crée ainsi un pipeline de fils d'exécution
  - implique que les fils d'exécution doivent être ordonnés
  - et que le problème peut être décomposé en un séquence de tâches appliquées sur des données distinctes

Modèles de conception

Groupe de travailleurs Producteurs-consommateurs

- Dans le paradigme du «groupe de travailleurs», les tâches attendent dans une file et les travailleurs (les fils d'exécution) exécutent des tâches tant que la file n'est pas vide

- Pour le paradigme «producteurs-consommateurs», il y a deux types de fils d'exécution: les producteurs qui produisent des données et les consommateurs qui les consomment



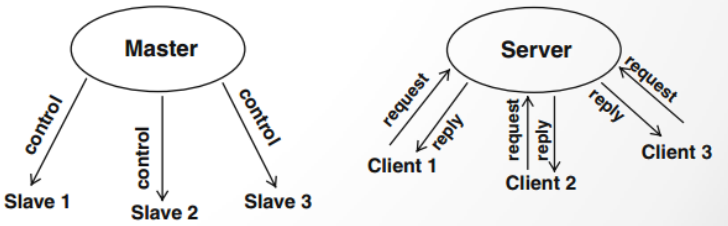
Modèles de conception

Modèle maître-ouvrir (maître-esclave)

- Le maitre connaît les données et le travail
- Un ouvrier attend du maître soit une demande de calcul (l'ouvrier exécute le calcul et retourne le résultat), soit un ordre de fin

- Cette solution a cependant des limites :
  - La mémoire locale du maître doit pouvoir charger toutes les données
  - 2 envois de messages pour 1 calcul → nécessité d'une grande granularité de calcul forte pour une bonne efficacité
  - S'il y a trop d'ouvriers, le maître peut être un goulet d'étranglement

- Avantages :
  - L'équilibrage de charge peut se faire en fonction de l'hétérogénéité du matériel, ou de son occupation partielle (par d'autres utilisateurs)



Modèles de conception

Modèle auto-régulé

Principe de « vol de travail » (work stealing) :

- Chaque processeur gère sa propre liste de travaux à effectuer
- Si la liste de travail d'un processeur est vide, il récupère une partie de la liste des autres processeurs
  - + meilleure gestion mémoire
  - + tous les processeurs participent au calcul
  - - difficulté de programmation