Contents

Chaque processeur a ses propres données.

Il faut donc partitionner (découper) le problème sur les PE

Pour implémenter le modèle à échange de messages (MPI), on a les hypothèses suivantes, garanties par le système d'échange de message

- Chaque PE connait le nombre p de PE impliqués
- Chaque processeur reçoit un identifiant, my-rank, entre 0 et p - 1
- Le même programme est exécuté par chacun des PE (processeurs), mais ce programme est paramétré par p et my-rank.
- Il existe des primitives de communication qui permettent d'échanger des données entre les PE. (MPI_Send, MPI_Recv) (Système boîte aux lettres pour chaque PE)

0.0.0.1 Exemple de programme avec échange de messages

But: faire une transposition des données qui va inverser l'ordre du tableau...

On veut paralléliser cela: on découpe le tableau

Pseudo code MPI

```
// Créer les données dans chaque sous-vecteur
// A sera un vecteur de taille n/p
for i = 0 to n/p - 1
    A[i] = my_rank
```

```
// Créer les données dans chaque sous-vecteur
// A sera un vecteur de taille n/p
for i = 0 to n/p - 1
    A[i] = my_rank (n/p) + i

// Ces lignes de code ne necessitent aucune communication et les PE travaillent
// indépendamment

// Communication:
source = p - 1 - my_rank // Le PE symétrique
dest = source
MPI_Sendrecv(A, n/p, dest, B, n/p, source)
// n/p = taille, A: message, B: buffer pour recevoir le message

// Transposition locale:

for i = 0, n/p - 1
    A[i] = B[n/p - 1 - i]
```

0.0.1 Primitives de communication

Dans un système à échange de messages, il existe de nombreuses primitives qui réalisent optimalement des patterns d'échange de données

- Point-à-point (one-to-one) ex: MPI_Send(), MPI_Recv()
- Permutations: $P_i \rightarrow P_{f(i)}$ $\forall i$, et f est une permutation des p adresses...

  La fonction est bijective

  Par exemle la transposition vue au dessus....

  Chaque processeur envoie à son voisin $((my\_rank + 1)$ % p) et on reçoit du même processeur
- Broadcast, one for all diffusion

  Un processeur envoie la même donnée à tous les autres processeurs

  MPI_Bcast(...root....)
- Réduction (many to one) On fait une opération sur tout ce que les autres processeurs ont envoyé, comme par exemple un plus...
- Gather: un processeur reçoit des données de tous les atures et les stocke dans un tableau
- Scatter, distribution one-to-all personnalisée: contraire de Gather.

  Un processeur envoire une donnée différente à tous les autres processeurs
- Echange total, all-to-all:

  Chaque processeur envoie une donnée propre à tous les autres: une suite de broadcast multiple.
- Echange total personnalisé: chaque PE envoie un message différent vers un autre PE. C'est un scatter multiple
- Scan ou parallel prefix:

  Chaque processeur reçoit la somme des valeurs contenues dans tous les processeurs de rang inférieur à lui:

  $P_i = \sum_{j=0}^{i} a_j$

  On peut avoir n'importe quelle opération arithmétique ou logique.

  Beaucoup d'algorithmes parallèles utilisent les primitives scan.

Un algorithme parallèle par échange de message est une combinaison d'algorithmes local à chaque PE et de primitives de communication.

Echange de message: pour ou contre ?

- C'est un modèle clair et explicite de collaboration

- Mais c'est contraignant, peu intuitif et nécessite de repenser le problème

  Ex: partitionner les données

- Matériel relativement standard et simple.

  On contrôle plutôt bien ce qui se passe.

  Lien assez serré entre le programme et ce que fait la machine.

- Ce sont des systèmes scalables: on peut faire collaborer des millions de PE avec ce modèle (à la fois HW (hardware), et échange de messages)

- Coordination implicite des processeurs

  On ne peut pas recevoir un message avant qu'il n'ait été envoyé. On préserve la causalité (dépendance) entre donnée.