Algorithmique parallèle
Bibliothèque MPI

Formation d'ingénieurs de l'Institut Galilée MACS-2

Philippe d'Anfray, Xavier Juvigny
Philippe.d-Anfray@renater.fr, xavier.juvigny@onera.fr

24 Novembre 2006

Calculateurs MIMD à mémoire distribuée

Ces calculateurs sont constitués de nœuds reliés par des liens de communication. Chaque nœud comprend -au moins- un processeur de calcul et une mémoire locale. L'ensemble des liens forme le réseau de communication de la machine. Celle-ci est caractérisée par :

- le nombre (éventuellement très grand) de nœuds ;
- la nature (processeur, taille mémoire) des nœuds ;
- les caractéristiques (notamment débit) des liens ;
- la topologie du réseau de communication.

Les nœuds sont indépendants les uns des autres (typiquement chacun possède sa propre horloge et son OS) et il n'y a pas de contrôle centralisé.

Dans la pratique ces machines sont souvent des grappes (*clusters*) de PCs que nous pourrons émuler en déployant des logiciels *ad hoc* sur un réseau de stations.

Modèle de programmation

C'est un modèle de tâches communicantes. Le programmeur lance sur les nœuds des programmes indépendants qui travaillent sur des données locales mais peuvent communiquer entre-eux par un mécanisme d'échange de messages. Ce mécanisme est asynchrone (cf. système de boîtes aux lettres).

Dans le cas général, un programme peut être vu comme un graphe dont les sommmets sont les tâches de calcul et dont les arêtes représentent les communications qui traduisent les nécessaires échanges d'informations et synchronisations entre tâches.

Dans la pratique, pour paralléliser une application on procède souvent à un découpage des données du problème à traîter, l'application est alors qualifiée de

SPMD. Le même code (*Single Program*) s'exécute sur chacun des nœuds de la machine en traitant des données différentes (*Multiple Data*). Cette méthode perrmet de décrire des programmes parallèles indépendants du nombre de nœuds de la machine.

Tout l'art consiste à répartir équitablement les tâches sur les nœuds de calcul (équilibrage) en minimisant les communications afin que le coût de contrôle du parallélisme ne devienne pas prédominant. Pour cela il faut aussi veiller à la taille des tâches qui seront exécutées en parallèle (granularité).

Outil de programmation

Nous utiliserons la bibliothèque MPI (*Message Passing Interface*) déployé sur les stations de travail ou PC en réseau de l'université. MPI existe en version Fortran, C et C++.

La bibliothèque MPI fournit les primitives du modèle :

– accès à l'environnement, *qui suis-je ?* parmi *combien de nœuds ?* ;
– envoi et reception de messages ;
– synchronisation.

On y trouve aussi de nombreuses fonctions de haut niveau : diffusions, réductions, mouvements de données...

Quelques exemples d'algorithmes distribués

Nous nous proposons d'étudier et de programmer quelques exemples simples d'algorithmes distribués. Pour chaque exemple, décrire la stratégie et l'algorithme local (papier) puis effectuer la programmation en C avec MPI.

1-Faire tourner un "jeton"

Un problème courant dans les applications distribuées est celui du contrôle : *qui peut ou doit faire quelque chose ?*. Une solution consiste à utiliser un "jeton" qui peut circuler entre les nœuds de la machine. Le nœud qui possède le jeton a alors certains droits et peut travailler sans risque de conflit avec les autres processeurs.

Ici nous essaierons de faire tourner le jeton (symbolisé par un entier) sur "un anneau" de processeurs. Le processus à décrire est le suivant : le nœud 0 lit une valeur, l'envoie au nœud 1 qui le transmet à 2 etc...jusqu'a ce qu'il revienne à 0 qui l'imprime et termine le processus.

Attention à bien écrire le même programme pour chacun des nœuds, c'est en testant l'environnement (*qui suis-je ? parmi combien ?*) que l'on décide de ce que l'on fait.

2-La diffusion

Nous nous intéressons maintenant à la diffusion (*broadcast*) d'une information par un nœud (appelé la "racine") à tous les autres. Si la racine diffuse l'information en envoyant successivement des messages aux autres nœuds, cela risque d'être coûteux (méthode naïve). En revanche, si tous les nœuds participent à un processus global on ira beaucoup plus vite.

Programmer néanmoins la méthode dite naïve puis, en raisonnant sur le cas particulier d'un cube expliciter un algorithme permettant de diffuser l'information de façon optimale (délai d'exécution, 3 échanges de messages au lieu de 7 en cas d'envoi successifs). Le programmer quand la racine est le `nœud 0`. Généraliser à un nœud racine quelconque, à un opérateur quelconque.

3-La réduction

Une réduction est l'application d'un opérateur à des données distribuées sur les nœuds. Par exemple, la somme ou le max de variables réelles dont un exemplaire est présent dans chaque mémoire lobale. Le résultat de la réduction est récupéré sur un nœud (la racine) jouant un rôle particulier.

En utilisant le résultat de la question précédente, trouver et expliciter un algorithme permettant d'effectuer, toujours pour le cas particulier du cube, une opération de réduction amenant le résultat sur un nœud donné. Le programmer quand la racine est le `nœud 0` et quand l'opérateur est le "max" de variables réelles. Généraliser à un nœud racine quelconque.

4-"Squelette" d'une application

On se propose enfin d'écrire le squelette d'une application parallèle `SPMD` :

1. Le nœud 0 lit des données et les diffuse aux autres nœuds.
2. Synchronisation, début des mesures de temps.
3. . . .calculs . . .pour l'instant on ne fait rien. . .
4. Fin des mesures de temps ; calcul du temps "global" d'exécution (max des temps "locaux").
5. Récupération et impression (dans le bon ordre) par le nœud 0 des résultats des autres nœuds.

5-"Bonus" : la réduction diffusion

Il est souvent utile de diffuser le produit d'une réduction pour que tous les nœuds aient le résultat. Il est possible d'enchaîner les deux opérations en choisissant un nœud racine mais d'autres optimisations sont possibles.

Expliciter et programmer un algorithme optimal effectuant simultanément ces opérations sur une structure hypercube. Regarder d'abord le cas du "segment" puis du "carré" et généraliser.