Maîtrise d'informatique
Module de Programmation parallèle

Examen

Philippe Marquet

Juin 2000

Tous documents autorisés
Durée : 3h

Ce sujet comporte deux parties. Dans une première partie nous reprenons des algorithmes déjà présentés en cours/TD/TP et les implantons sur une nouvelle architecture logicielle/matérielle. Dans la seconde partie, nous étendons ces algorithmes.

Important Plusieurs questions vous demandent d'expliquer/décrire/motiver l'énoncé d'un algorithme, la parallélisation d'un problème... Ne les négligez pas ; ces questions sont notées ! Bien entendu, des réponses concises et précises sont appréciées.

1 Programmation de l'IBM SP : OpenMP + MPI

Il s'agit d'implanter les deux problèmes du jeu de la vie et de la multiplication matrice/vecteur sur la nouvelle machine de l'université. Cette machine est constituée d'un ensemble de $N = 4$ nœuds connectés par un réseau dédié. À ce niveau, la machine est programmée en MPI. Chacun des nœuds de la machine est un multiprocesseur ($P = 16$ processeurs) à mémoire partagée. À ce niveau la machine est programmée en OpenMP.

Il s'agit donc de produire deux programmes OpenMP + MPI. Nous allons procéder en plusieurs étapes.

Multiplication matrice/vecteur On considère l'algorithme de multiplication matrice/vecteur faisant circuler le vecteur parmi les différents processeurs organisés en un anneau. On considère une matrice carrée de $n \times n$ valeurs. On suppose que les valeurs $\lambda = n/N$ et $\mu = \lambda/P$ sont entières. À chaque étape, on additionne dans le vecteur résultat la contribution sur les données locales de la matrice des valeurs du vecteur circulant. Une implantation efficace de cet algorithme doit considérer le recouvrement des communications par les calculs.

Jeu de la vie d'après J. Conway Il s'agit ici de simuler l'évolution d'une population de cellules réparties sur une grille carrée de taille $n \times n$. On considère que la grille est torique ; ainsi chaque cellule a huit cases voisines. On suppose aussi que les valeurs $\lambda = n/N$ et $\mu = \lambda/P$ sont entières. Les règles définissant l'évolution sont les suivantes :

- si une cellule vivante possède 2 ou 3 cellules voisines, elle continue à vivre ;
- une cellule meurt
  - d'isolement si elle a moins de 2 voisines,
  - d'étouffement si elle a plus de 3 voisines ;
- une cellule naît si exactement 3 cases voisines sont occupées.

L'évolution d'une population initiale se déroule jusqu'à ce que l'on observe une stabilisation de la population, ou jusqu'à ce que l'on ait atteint un nombre MAX d'itérations. De même que pour le

Id: xam00-juin.tex,v 1.7 2000/06/20 06:39:13 marquet Exp

problème précédent, une implantation efficace de cet algorithme doit considérer le recouvrement des communications par les calculs.

Exercice 1 (Quelle approche ?)
Dans le but de produire des programmes OpenMP + MPI, il est possible de procéder de deux manières : produire une version OpenMP puis l'enrichir d'un parallélisme MPI, ou produire une version MPI et l'enrichir d'un parallélisme OpenMP.

Choisissez une des deux approches et discutez (brièvement) ce choix.

Exercice 2 (OpenMP ou MPI)
Il s'agit maintenant de mettre en œuvre le choix précédent. Vous allez développer une version OpenMP ou une version MPI des algorithmes.

Question 2.1 Donnez les grandes lignes d'une parallélisation OpenMP ou MPI de la multiplication matrice vecteur puis du jeu de la vie.

Question 2.2 Donnez le code C/MPI ou C/OpenMP d'une version parallèle de ces deux programmes.

Exercice 3 (OpenMP et MPI)
Il s'agit de développer la version finale OpenMP + MPI des algorithmes.

Question 3.1 Explicitez les changements devant intervenir sur vos programmes développés à l'exercice précédent pour ajouter un second niveau de parallélisation.

En particulier précisez bien comment s'articule l'imbrication des deux parallélismes.

Question 3.2 Donnez le code C/OpenMP + MPI de ces deux programmes.

2 De $A \times X$ à Fox ou de Conway à l'équilibrage par pavage

Nous considérons des extensions des problèmes traités dans la section précédente. Nous utiliserons le paradigme de programmation à passage de messages que nous implanterons au choix avec PVM ou MPI sur un ensemble de N processeurs.

Multiplication de matrices L'algorithme de Fox pour la multiplication de matrices est décrit ci-après. On calcule $C = A \times B$, avec A, B, et C des matrices carrées de n^2 éléments (ou blocs d'éléments) notés a_{ij}, bij, et c_{ij}, avec $0 \leq i < n$ et $0 \leq j < n$. L'algorithme consiste à calculer et accumuler localement les termes de la somme :

$$c_{ij} = a_{i0}b_{0j} + a_{i1}b_{1j} + ... + a_{in}b_{nj}$$

en fournissant au processus responsable du bloc c_{ij} les données nécessaires. Pour cela on effectue n étapes telles qu'à la k^{e} étape :

- pour chaque ligne, on diffuse à tous les processus d'une ligne le bloc de A situé à la k^{e} position à droite du bloc diagonal ;
- pour chaque colonne, on permute circulairement vers le haut les blocs de B ;
- pour chaque bloc, on accumule dans c_{ij} le produit du bloc de A reçu par diffusion et du bloc de B reçu par permutation.

À la fin des n étapes, chaque bloc c_{ij} contient le résultat et les blocs b_{ij} sont revenus à leur place initiale. La figure 1 illustre les premières étapes de l'algorithme.

$$\begin{pmatrix} c_{00} & c_{01} & c_{02} & c_{02} \\ c_{10} & c_{11} & c_{12} & c_{12} \\ c_{20} & c_{21} & c_{22} & c_{22} \\ c_{30} & c_{31} & c_{32} & c_{32} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{00} & a_{00} & a_{00} \\ a_{11} & a_{11} & a_{11} & a_{11} \\ a_{22} & a_{22} & a_{22} & a_{22} \\ a_{33} & a_{33} & a_{33} & a_{33} \end{pmatrix} \times \begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{02} \\ b_{10} & b_{11} & b_{12} & b_{12} \\ b_{20} & b_{21} & b_{22} & b_{22} \\ b_{30} & b_{31} & b_{32} & b_{32} \end{pmatrix}$$

$$\begin{pmatrix} c_{00} & c_{01} & c_{02} & c_{02} \\ c_{10} & c_{11} & c_{12} & c_{12} \\ c_{20} & c_{21} & c_{22} & c_{22} \\ c_{30} & c_{31} & c_{32} & c_{32} \end{pmatrix} += \begin{pmatrix} a_{01} & a_{01} & a_{01} & a_{01} \\ a_{12} & a_{12} & a_{12} & a_{12} \\ a_{23} & a_{23} & a_{23} & a_{23} \\ a_{30} & a_{30} & a_{30} & a_{30} \end{pmatrix} \times \begin{pmatrix} b_{10} & b_{11} & b_{12} & b_{12} \\ b_{20} & b_{21} & b_{22} & b_{22} \\ b_{30} & b_{31} & b_{32} & b_{32} \\ b_{00} & b_{01} & b_{02} & b_{02} \end{pmatrix}$$

FIG. 1 – *Les premières étapes de la multiplication matricielle par la méthode Fox*

Équilibrage par pavage Il s'agit d'améliorer les performances de la simulation du jeu de la vie. On suppose travailler dans un environnement non dédié (à notre application : d'autres programmes, d'autres utilisateurs partagent les ressources). Ainsi les différents processeurs utilisés peuvent être considérés comme ayant des vitesses différentes et pouvant évoluer au cours du temps. Notre simulation comportant une forte synchronisation entre les différents processeurs, il est intéressant d'équilibrer la charge proportionnellement à la vitesse des processeurs. L'algorithme d'équilibrage de la charge par pavage est décrit ci-après.

Cet algorithme divise l'ensemble des processeurs en un ensemble de petits sous-domaines disjoints de processeurs : les *tuiles*. Au sein de chaque tuile, un équilibrage *parfait* de la charge est réalisé. Afin de propager cet équilibre à l'ensemble du système, le pavage des processeurs en tuiles est décalé pour la phase d'équilibrage suivante.

Pour un anneau de processeurs, une tuile consiste en un groupe de deux processeurs « voisins » ; une de ces tuiles contient alternativement au cours du temps un processeur de rang pair k et le processeur $k+1$ puis un processeur de rang impair k et son voisin de droite comme illustré à la figure 2.

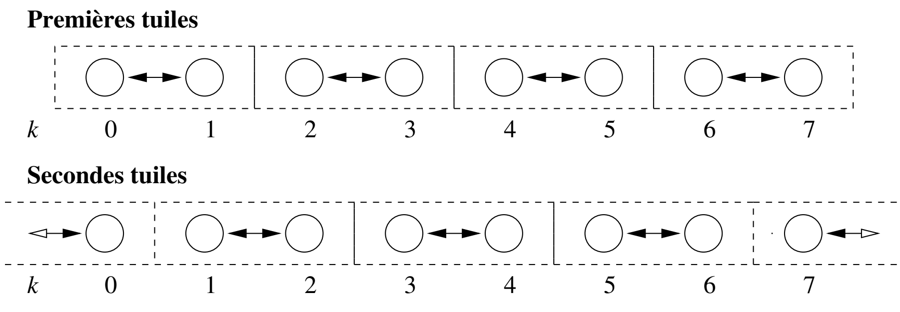

FIG. 2 – *Le pavage en tuiles d'un anneau de processeurs*

L'équilibrage parfait au sein d'une tuile est le suivant : soit la tuile consituée des processeurs k et $k+1$. On définit les vitesses des processeurs comme le rapport du nombre d'éléments traités lors d'une phase de la simulation (la charge s_k) par le temps de ce traitement : $v_k = s_k/t_k$. On équilibre la charge proportionnellement à cette vitesse ; on cherche donc les deux nouvelles charges s'_k et s'_{k+1} telles que :

$$\frac{s'_k}{v_k} = \frac{s'_{k+1}}{v_{k+1}}$$

il faut aussi que la charge totale reste constante :

$$s_k + s_{k+1} = s'_k + s'_{k+1}$$

On en déduit que :

$$s'_k = \frac{s_k + s_{k+1}}{1 + \frac{v_{k+1}}{v_k}}$$

Exercice 4
Traitez au choix un des deux problèmes : multiplication de matrices ou équilibrage par pavage.

Question 4.1 Décrivez comment vous parallélisez le problème choisi.

Question 4.2 Donnez le code source d'une version parallèle du problème choisi.

3 Annexe

Pourvu que vous en définisiez précisément le comportement, vous pouvez introduire nombre de fonctions utiles qui ne sont pas en rapport direct avec les aspects centraux des développements à réaliser ici. Par exemple :

`long gettimer()` retourne le temps écoulé depuis le dernier appel à `starttimer()`.

`typedef .. cellule_t` est le type des cellules d'une population.

`cellule_t next_cell(cellule_t c, short nbv)` retourne la nouvelle valeur d'une cellule `c` en fonction de son nombre de voisins `nbv`.