Maîtrise d'informatique
Module de Programmation parallèle

Examen

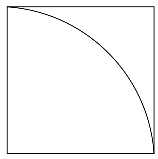
Philippe Marquet

Juin 1999

Tous documents autorisés
Durée : 3h
Barème : 6/4/5/5

1 Calcul probabiliste de π

Soit un carré de coté 1 contenant un quart de disque de rayon 1 sur l'un des ses coins (figure ci-contre). La probabilité qu'un point du carré appartienne au quart de disque est de



$$\frac{\text{Aire du quart de disque}}{\text{Aire du carré}} = \frac{\pi}{4}.$$

À partir de cette remarque, il s'agit de construire un programme de calcul de π sur une ferme de N machines en utilisant PVM.

Ce programme s'exécute jusqu'à ce qu'une saisie clavier l'interrompe ; il affiche alors la valeur calculée de π.

On suppose définies les fonctions suivantes :

- La fonction `getkey()` est bloquante en attente d'une saisie au clavier.
- La fonction `keystroke()` retourne une valeur booléenne indiquant si une touche a été frappée depuis le début du programme ou depuis l'appel précédent. Cette fonction n'est pas bloquante.
- La fonction `random()` retourne un réel aléatoire dans l'intervalle $[0, 1]$.
- (On ne sait jamais...) La fonction `distance()` retourne la distance d'un point à l'origine ; elle prend deux paramètres (abscisse et ordonnée du point).

Exercice 1 (Version PVM)
Développer une version utilisant des tâches PVM monothreadées. En particulier :

1. Donner un algorithme séquentiel du calcul de π. Cet algorithme sera implanté dans une fonction qui pourra être réutilisée dans la version parallèle.
2. Décrire l'architecture de l'application parallèle (rôle des différents processus, structures de données locales à chacun des processus, type de messages...).
3. Étudier précisément la terminaison du programme. Préciser comment et quand les résultats partiels sont récupérés et comment l'application se termine.

Exercice 2 (Version PVM multithreadée)
Développer une version PVM multithreadée du programme. Préciser en particulier l'architecture des processus légers définis au sein de chacune des tâches PVM et les structures de données partagées entre ces processus légers.

Id: xam99-juin.tex,v 1.5 1999/08/27 13:16:04 marquet Exp

2 Équilibrages de charges data-parallèles

Soit une « pseudo-MasPar » de `nproc` processeurs organisés en une grille de $1 \times$ `nproc`. Chacun des processeurs de la machine est en charge d'un travail représenté par la valeur de la variable :

```
plural int w
```

Le but des algorithmes d'équilibrage est de répartir au mieux cette charge sur les processeurs. Pour ce faire, on peut utiliser les deux fonctions de manipulation de la charge :

```
void send (plural int dest, plural int size)
```

qui transfert `size` charges de travail du processeur courant au processeur `dest`, et

```
void receive (plural int from, plural int size)
```

qui transfert `size` charges de travail du processeur `from` au processeur courant.

Différents algorithmes d'équilibrage de la charge peuvent être mis en œuvre. On distingue les algorithmes utilisant des schémas de communication irréguliers (`router`) et les ceux utilisant des schémas de communication réguliers (`xnet`).

Exercice 3 (Une stratégie d'équilibrage à l'aide de communications régulières)
Soit W la charge totale détenue par les $P =$ `nproc` processeurs,

$$W = \bar{w}P + r \qquad \text{avec} \qquad 0 \leq r < P.$$

Les valeurs $\bar{w}$ et r peuvent être calculées en fonction de la variable `w`. On équilibre la charge de telle manière que les r premiers processeurs contiennent chacun $\bar{w} + 1$ charges ; et les $P - r$ processeurs restant $\bar{w}$.

Le processeur k peut calculer w_Σ, le nombre de charges qui doivent être détenues par lui et ses voisins de gauche à l'équilibre :

$$w_\Sigma = (k+1)\bar{w} + \min(k+1, r).$$

À partir de cette valeur w_Σ et du nombre de charges effectivement détenues par le processeur et ses voisins de gauche, il est possible d'atteindre l'équilibre de la charge en effectuant des transferts de charges entre voisins.

Mettre en œuvre l'algorithme ainsi défini en MPL sur la « pseudo-MasPar. » Il est bien entendu exclu d'utiliser les communications via le « *global router.* »

Exercice 4 (Une stratégie d'équilibrage à l'aide de communications irrégulières)
L'algorithme proposé maintenant équilibre la charge entre des couples de processeurs. On apparie les processeurs deux à deux. Ces deux processeurs s'échangent des charges afin de posséder tous deux la même charge (à une valeur près).

Contrairement à l'exercice précédent, on atteindra pas un équilibre de la charge parfait sur toute la machine, mais une meilleure répartition de cette charge.

Afin de répartir cette charge au mieux, on réalise les appariements de processeurs de telle manière à associer le processeur le plus chargé avec le processeur le moins chargé, le deuxième processeur le plus chargé avec le deuxième processeur le moins chargé...

Un première étape est donc d'identifier le rang de chacun des processeurs en fonction de leur charge.

Le k^e processeur le plus chargé (noté processeur k^+) et le k^e le moins chargé (noté k^-) doivent ensuite pouvoir s'identifier mutuellement. La seule information détenue en commun par ces deux processeurs est la valeur de k. Le processeur k va donc servir de lieu de rendez-vous entre ces deux processeurs : k^+ dépose sont numéro de processeur sur k ; k^- vient l'y chercher et peut ainsi établir l'échange de charges entre k^+ et k^-.

Mettre en œuvre l'algorithme ainsi défini en MPL sur la « pseudo-MasPar. » Il est bien entendu exclu d'utiliser les communications via le réseau « *xnet.* »