Maîtrise d'informatique
Module de Programmation parallèle

Examen

Philippe MARQUET

Juin 2002

Tous documents autorisés
Durée : 3h

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 Sauna mixte

Au sein d'un établissement, un unique sauna peut être aussi bien utilisé par les hommes que par les femmes ; mais pas en même temps. De plus, le sauna ne peut contenir plus de N personnes.

Il s'agit de proposer une solution pour réguler les accès au sauna. Dans un premier temps nous allons esquisser une solution à l'aide d'un moniteur de Hoare puis nous l'implanterons en terme de processus légers.

Exercice 1 (Moniteur de Hoare)

- Décrivez les variables d'état du moniteur, et en particulier les conditions nécessaires à son implantation.
- Donnez les entrées d'un moniteur de Hoare pour le contrôle des entrées au sauna.
- Explicitez les choix de priorité que vous seriez amenés à faire. □

Exercice 2 (Processus légers)
Donnez le code d'une implantation du moniteur défini. Les hommes et les femmes étant implantées comme des processus légers faisant appel à des fonctions implantant les entrés du moniteur. On utilisera l'API `pthread` d'une bibliothèque de gestion de processus légers. □

2 Calcul data-parallèle de π par intégration

On peut approximer la valeur de

$$\pi = \int_0^1 \frac{4}{1+x^2} \quad dx$$

par

$$w. \sum_{x=\frac{w}{2}, 1-\frac{w}{2}, w} \frac{4}{1+x^2}$$

(Méthode des rectangles ; w est la largeur de chacun des rectangles.) Le programme suivant calcule ainsi une valeur approchée de π :

```
#include <stdio.h>
#include <time.h>

#define f(A) (4.0/(1.0+A*A))

const int n = 1 << 24 ; /* 1024 * 128 * 128 */

int main ()
{
  int i ;
  double w, x, sum, pi ;

  w = 1.0 / n ;
  sum = 0.0 ;
  for (i=1 ; i<=n ; i++) {
    x = w * ((double)i-0.5) ;
    sum += f(x) ;
  }
  pi = w * sum ;

  printf( "computed pi = %24.16g\n", pi ) ;
}
```

Exercice 3
Proposez une parallélisation de ce programme de calcul de π sur une machine de type MasPar de 128×128 processeurs programmée en MPL. □

3 Réduction/diffusion sur une grille de processus communicants

Soit une grille de $N \times N$ processus communicants. Chacun des processus connaît les processus à sa droite, à sa gauche et au dessus et dessous de lui. Les quatre processus des coins de la grille n'ont que deux voisins ; les autres processus du bord de la grille ont trois voisins. Les primitives utiles sont :

`xrank()` et `yrank()` retournent l'indice de colonne et ligne du processus, entre 0 et $N-1$;

`size()` retourne la valeur de N ;

`send(dest, buf, length)` envoie (de manière non bloquante) le buffer `buf` de `length` éléments au processus désigné par `dest` qui peut prendre une des quatre valeurs `north`, `south`, `east`, `west`.

`recv(sender, buf, length)` range dans le buffer `buf` les `length` éléments reçus du processus désigné par `sender` qui peut prendre une des valeurs `north`, `south`, `east`, `west` ou la valeur `any` pour une réception depuis un quelconque des processus voisins. Cette réception est bloquante.

Il s'agit de réaliser la réduction/diffusion par un opérateur binaire d'une valeur détenue localement par chacun des processus à l'ensemble des autres processus. Si on considère par exemple la réduction/diffusion résultant de l'addition, à la terminaison de l'algorithme, tous les processus détiennent la somme des valeurs détenues initialement par chacun.

Il s'agit donc de fournir la fonction SPMD suivante :

```
typedef struct elem_s elem_t;
elem_t allreduce (elem_t operand, elem_t (*op)(elem_t, elem_t));
```

`op()` est une fonction à deux paramètres de type `elem_t` retournant une valeur de ce même type.

Une solution élémentaire est de considérer au sein de la grille un chemin de communication de chacun des processus vers un processus désigné ; par exemple le processus $(0, 0)$. Cet ensemble de chemins forme un arbre dont le processus $(0, 0)$ est la racine. Chacun des processus étant à une distance d'au plus $2N - 2$ du processus $(0, 0)$, l'arbre est de hauteur $2N - 2$.

Cet arbre est utilisé pour produire la valeur de la réduction en ce processus $(0, 0)$. Ce même arbre est ensuite utilisé pour diffuser cette valeur depuis ce processus $(0, 0)$. Nous avons un algorithme de réduction/diffusion en $4N - 4$ étapes de communications.

Deux améliorations de cet algorithme sont possibles. Une première amélioration (exercices 4 et 5) considère un « meilleur » processus centralisateur que le processus $(0, 0)$. Une seconde amélioration (exercice 6) augmente le réseau de communications pour travailler sur une grille torique. Les exercices 7 et 8 comparent les deux approches.

Afin de simplifier les choses, on suppose maintenant que la valeur N est impaire : $N = 2P+1$.

Exercice 4 (Algorithme de réduction/diffusion en $4P$ étapes de communications)
En remarquant que le processus (P, P) est à une distance d'au plus $2P$ de chacun des autres processus, il est possible de réaliser l'opération réduction/diffusion en $4P$ étapes de communications.

Décrivez un algorithme de réduction/diffusion sur une telle grille de processus. □

Exercice 5 (Implantation de la réduction/diffusion)
Donnez le code C de la fonction `allreduce()` utilisant les primitives ci-dessus réalisant une implantation de votre algorithme de réduction/diffusion en $4P$ étapes de communications. □

Exercice 6 (Grille torique)
On travaille maintenant sur une grille torique ; tous les processus ont donc maintenant exactement quatre voisins.

Si tous les processus calculent en parallèle et simultanément les réductions partielles des valeurs qu'ils détiennent et que les valeurs circulent entre les processus, une fois que les valeurs ont transitées par tous les processus sur une première dimension, on dispose des réductions selon cette dimension sur l'ensemble des processus. On itère alors le procédé sur la seconde dimension.

Détaillez cet algorithme et donnez le code C d'une implantation de la fonction `allreduce()` selon cet algorithme. En particulier, mettez en valeur un possible recouvrement des communications par des calculs. □

Exercice 7 (Comparaison)
Comparez les deux algorithmes, en particulier en terme de temps d'exécution, volume de communications et volume de calcul. □

Exercice 8 (Et si la réduction ne réduit plus...)
Étendez la comparaison précédente au cas où le produit d'une réduction par la fonction `op()` est d'une taille égale à la somme des tailles des opérandes de l'opération (par exemple la concaténation). □

Fin du sujet