Maîtrise d'informatique
Module de programmation parallèle

TD[1] : Introduction au parallélisme de données

Philippe Marquet

Avril 2002

Exercice 1 (Arithmétique sur des grands entiers)
On se propose d'utiliser la MasPar pour réaliser des opérations arithmétiques sur des entiers pouvant comporter jusqu'à plusieurs milliers de chiffres. Nous travaillons en MPL.

Dans le cadre d'une première évaluation, nous décidons de représenter un tel `ENTIER` par une valeur `plural` en distribuant un chiffre par processeur élémentaire.

Question 1.1 Donner le schéma de code MPL réalisant la multiplication d'un `ENTIER` par un `int` puis d'un `ENTIER` par un `ENTIER`.

Question 1.2 Proposer un implantation plus complète. Deux aspects sont à considérer : l'efficacité et la possibilité de manipuler des `ENTIER` comportant plus de chiffres que le nombre de processeurs de la machine.

Exercice 2 (Calcul 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 ;
```

[1]Plusieurs des exercices proposés ici sont issus de sujets d'interrogation ou d'examen des années précédentes

Id: td-dp.tex,v 1.8 2006/03/07 18:15:40 marquet Exp

```
  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 ) ;
}
```

Question 2.1 Proposer une parallélisation du programme de calcul de π sur une machine de type MasPar de 128×128 processeurs programmée en MPL.

Exercice 3 (Composantes connexes d'une image)
Soit une image couleurs (par exemple 256 couleurs) de taille 128×128. Cette image est stockée dans une variable

```
plural char image ;
```

à raison d'un pixel par processeur de la machine.

Le but est de déterminer les composantes connexes de cette image. Ces composantes connexes définissent des régions de l'image. Une région est définie comme une zone continue de pixels voisins (dans les 4 directions Nord, Est, Sud, Ouest) de même couleur.

Les régions sont identifiées par des entiers distincts.

On veut en résultat une variable

```
plural int regions ;
```

qui indique pour chaque pixel, donc pour chaque processeur, la région dans laquelle il est situé.

Exemple pour la grille 4×4 de gauche (les couleurs sont identifiées par des lettres), on obtient les régions de la partie droite (à une renumérotation près) :

```
a  a  b  b        0   0  2  2
a  b  c  c        0   5  6  6
a  a  a  c        0   0  0  6
a  d  c  c        0  13  6  6
```

Question 3.1 (Identification des régions) Donnez l'algorithme et le code MPL d'une fonction qui identifie les régions d'une image donnée.

On procède maintenant à une normalisation de la numérotation des régions telle que les régions soient identifiées par des entiers successifs à partir de 0. Sur notre exemple, on obtient (à une permutation près des numéros) :

```
0  0  1  1
0  2  3  3
0  0  0  3
0  4  3  3
```

Question 3.2 (Normalisation des régions) Donnez l'algorithme et le code MPL de renumérotation des régions. (On pourra utiliser la fonction de bibliothèque `reduceMin ()` ; *cf.* polycopié page 34.)

Question 3.3 (Complexité) Donnez les complexités des algorithmes des questions 3.1 et 3.2.

Exercice 4 (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`).

Question 4.1 (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*. »

Question 4.2 (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*. »

Exercice 5 (Shearsort)
Dans le principe, le *shearsort* étend le tri pair-impair sur une grille 2-D.

Soient $n \times n$ valeurs à trier. On considère une grille de $n \times n$ processeurs virtuels (n est une valeur entière paire). On trie alternativement les lignes et les colonnes : les lignes sont triées aux itérations impaires et les colonnes sont triées aux itérations paires. Les lignes sont triées :

- dans l'ordre croissant pour les lignes paires (de 0 à $n - 2$) ;
- dans l'ordre décroissant pour les lignes impaires (de 1 à $n - 1$).

Les colonnes sont triées dans l'ordre croissant de haut en bas. On obtient le résultat ($n \times n$ valeurs triées) « en serpent » au bout de $2 \log n + 1$ itérations. La figure 1 donne un exemple d'exécution du tri de 16 valeurs.

Soit une MasPar 128 × 128 processeurs. On applique l'algorithme du shearsort pour trier les 128 × 128 valeurs contenues dans une variable `plural int val`.

Question 5.1 Donnez l'algorithme d'une phase de tri des 128 lignes (phase impaire du shearsort). Explicitez en particulier en quoi les traitements des lignes paires et impaires peuvent ou ne peuvent pas être réalisés simultanément.

Question 5.2 Donnez le code MPL implantant l'algorithme de l'exercice précédent.

Question 5.3 Discutez de l'implantation de l'algorithme shearsort pour trier les $128p \times 128p$ valeurs contenues dans une variable `plural int val [p][p]`.

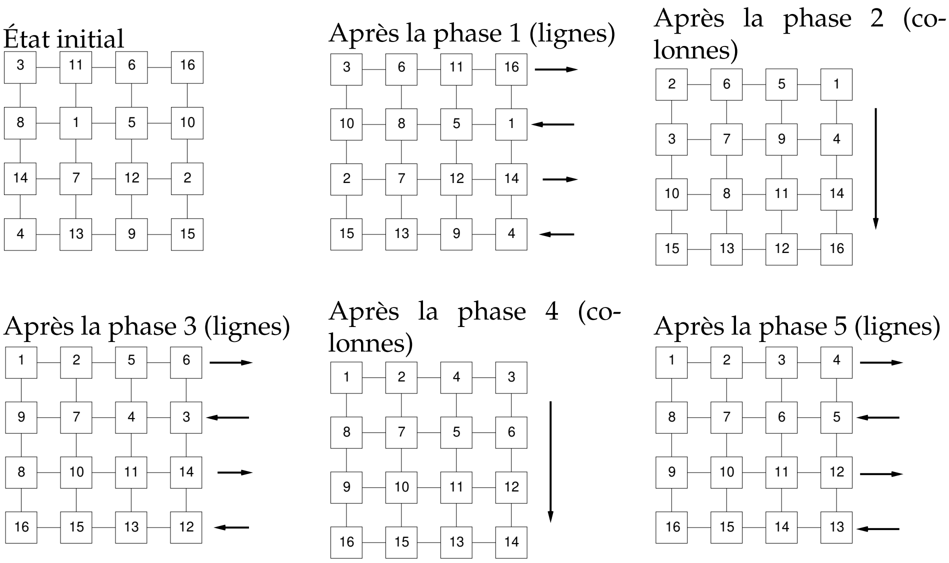

FIG. 1 – Exemple d'exécution du shearsort pour $n = 4$