Maîtrise d'informatique
Module de Programmation parallèle

Interrogation

Philippe Marquet

Avril 1999

Tous documents autorisés
Durée : 1h30

1 Coloriage de régions en MPL

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 fournir une image des régions. Une région est définie comme une zone continue de pixels voisins (dans les 4 directions Nord, Est, Sud, Ouest).

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
a	b	c	c
a	a	a	c
a	d	c	c

0	0	2	2
0	5	6	6
0	0	0	6
0	13	6	6

Exercice 1
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 :

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

Exercice 2
Donnez l'algorithme et le code MPL de renumérotation des régions. (On pourra utiliser la fonction de bibliothèque `reduceMin ()` ; *cf.* polycopié page 34.)

Id: xam99-interro.tex,v 1.2 1999/08/27 13:16:04 marquet Exp

2 Préfixes parallèles à l'aide de processus légers

Le programme des figures 1 et 2 est une tentative de calcul de toutes les sommes partielles d'un tableau `v0` par un ensemble de processus légers.

Exercice 3
Complétez la fonction `main ()` de ce programme.

Exercice 4
Exhibez un exemple d'exécution sur 2 processeurs de la fonction `scanAdd ()` qui ne produit pas le résultat attendu.

Exercice 5
Corrigez la fonction `scanAdd ()`.

```
#include <stdio.h>
#include <pthread.h>
#include "barrier.h"

#define NBPROC 8

/* Valeurs necessaires a la definition de la machine  virtuelle */
char Activite[NBPROC] ;
pthread_t Tids[NBPROC] ;

/* Les valeurs dans la memoire de cette machine */
int v0[NBPROC], v1[NBPROC], v2[NBPROC], v3[NBPROC] ;

/* Rendez-vous entre les threads */
barrier_t rdv ;

/*
  Retourne dans res le parallel prefix de la valeur v.
  Cette fonction doit etre appele par NBPROC threads "simultanement".
  Le parametre me identifie le thread appelant parmi les NBPROCS threads.
  0 <= me < NBPROC.
  */
void
scanAdd (int me, int *v, int *res)
{
  int i ;

  res[me] = Activite[me] ? v[me] : 0 ;

  for (i=1 ; i<NBPROC ; i*=2) {
    barrier_wait (rdv) ;
    if (me >= i)
      res[me] += res [me-i] ;
  }

  if (!Activite[me]) res[me] = 0 ;
}
```

Fig. 1 – *Programme principal (1/2)*

```
/*
  Code d'un thread, i.e. un processeur virtuel.
  */
void *
a_thread (void *arg)
{
  int me = (int) arg ;
  scanAdd (me, v0, v1) ;
}

void
main (void)
{
  int i, j ;
  int iproc ;
  pthread_attr_t attr ;

  /* initialisation */
  rdv = barrier_init () ;
  init_value (&v) ;
  init_activite () ;

  /* on lance les NBPROCS threads */
  /* ... A COMPLETER ... */

  /* On affiche le resultat */
  print_value (v1) ;

  exit (0) ;
}
```

FIG. 2 – *Programme principal (2/2)*

```
/* A barrier */
typedef struct *barrier_t ;

/* Create & initialize a barrier */
barrier_t barrier_init (void) ;

/* Destroy a barrier */
void barrier_destroy (barrier_t barrier) ;

/* Wait until the required number of threads enter the barrier */
void barrier_wait (barrier_t barrier) ;
```

FIG. 3 – *Bibliothèque* `barrier`