INF5170 — Programmation parallèle
Examen intra (Automne 2000)

Durée: 13h30 – 16h30 (3 heures) **Documentation autorisée:** Toute documentation personnelle.

1. Instructions atomiques (10 pts)

Indiquez ce qui sera (ou ne sera pas) imprimé par chacun des programmes suivants. Si plusieurs réponses sont possibles, indiquez-les toutes.

a.
```
int x = 1, y = 1;

co  < x = x + y; >
//  < y = 0; >
//  < x = x - y; >
oc

printf( "x = %d, y = %d\n", x, y );
```

b.
```
int x = 1;

co  < await (x > 0) x = x + 2; >
//  < await (x > 0) x = x - 3; >
//  < await (x < 0) x = x + 5; >
oc

printf( "x = %d\n", x );
```

2. Parallélisme itératif statique (15 pts)

En utilisant la notation d'Andrews, écrivez une procédure `tableaux_egaux` qui va permettre de déterminer si deux tableaux sont égaux, c'est-à-dire, contiennent les mêmes éléments aux mêmes positions. Vous pouvez évidemment introduire des procédures auxiliaires.

Le travail devra être fait en utilisant une approche avec parallélisme statique, donc où le nombre de processus créés est fonction du nombre de processeurs (`NUM_NODES`) et où chaque processus est un processus itératif. Pour déterminer l'intervalle attribué à un processus `i`, vous devez utiliser les fonctions suivantes:

```
int inf( int i, int n ){ return(  i    * n / NUM_NODES     ); }
int sup( int i, int n ){ return( (i+1) * n / NUM_NODES - 1 ); }
```

L'interface de la procédure principale sera la suivante:

```
bool tableaux_egaux( int a[], int b[], int n );
/* n = taille des tableaux a et b
   resultat = (a[0] == b[0]) && (a[1] == b[1]) && ... && (a[n-1] == b[n-1])
 */
```

3. Parallélisme récursif (20 pts)

[10] a) Les fonctions (dans la notation d'Andrews) présentées à la Figure 1 permettent de calculer la factoriel `n!` d'un nombre entier non-négatif définie comme suit:

```
0!  = 1
n!  = 1 * 2 * ... * n-1 * n
```

Complétez la fonction `fact` de façon à ce qu'elle génère du parallélisme récursif en utilisant une approche diviser-pour-régner bien équilibrée (qui génère, dans le cas *récursif*, deux sous-problèmes de tailles équivalentes). Notez bien le rôle des paramètres `i` et `n`, tel qu'indiqué dans l'en-tête de la fonction. Pour simplifier, vouz pouvez supposer que $n=2^k$.

[10] b) Écrivez la version Threaded-C de la procédure `fact`.

```
int factoriel( int n )
{
  assert( n >= 0 );

  if ( n == 0 )
    return( 1 );
  else
    return( fact(1, n) );
}



int fact( int i, int n )
/* i = borne inferieure des elements a multiplier
   n = nombre d'elements a multiplier
   resultat = i * i+1 * i+2 * ... * i+n-1
 */
{
  /* Code a completer. */









}
```

Figure 1: Fonctions pour calculer la factoriel d'un nombre

4. Recherche de la valeur maximum d'un tableau (20 pts)

Les procédures présentés à la Figure 2 permettent de déterminer la valeur maximum d'un tableau d'entiers `elems` de taille `n`. On suppose que les éléments sont non-négatifs et, donc, que `0` est plus petit ou égal à n'importe quel élément du tableau.

[15] a) Écrivez la version Threaded-C de la procédure `trouver_max`. Vous pouvez utiliser n'importe quelle primitive ou opérations de librairie vues en classe. Toutefois, votre mise en oeuvre Threaded-C doit *réfléter le plus fidèlement possible* la version de la notation d'Andrews.

[5] b) Combien de communications seront requises, dans la version Threaded-C, pour chaque appel de `trouver_max`, *en incluant* la communication initiale pour créer l'activation de la procédure `trouver_max` et celle pour indiquer la fin de la procédure, et en supposant que chaque activation est créée sur un processeur différent du processus maître (`max`)?

```
/* Processus execute sur les differents processeurs. */

void trouver_max( int elems[], int n, int *max_global )
/* elems = adresse de depart de la portion de tableau a fouiller
   n = nombre d'elements de la partie de tableau a fouiller
   max_global = addresse de la variable qui doit etre modifiee
                s'il faut mettre a jour le maximum global
 */
{
  int i;
  int mon_max = 0;

  for [i = 0 to n-1] {
    if( elems[i] > mon_max )
      mon_max = elems[i];

  < if( mon_max > *max_global )
      *max_global = mon_max;    >
}


int borne_inf( int i, int n ){ return( i * n / NUM_NODES ); }


int max( int elems[], int n )
{
   int i;
   int max_global = 0;

   assert( n % NUM_NODES == 0 );

   co[i = 0 to NUM_NODES-1] {
     trouver_max( &elems[borne_inf(i, n)], n/NUM_NODES, &max_global );
   }
   return( max_global );
}
```

Figure 2: Procédures, dans la notation d'Andrews, pour trouver le maximum dans un tableau