Exercices INF7440 : série #6

0. Parallélisme itératif à granularité fine

Soit le programme MPD 1 (défini de façon partielle). Ce programme, après avoir généré deux vecteurs de nombres entiers, doit effectuer la somme de ces deux vecteurs, puis imprimer le vecteur résultat.

Complétez ce programme de façon à ce que la somme des deux vecteurs se fasse à l'aide de *parallélisme itératif à granularité fine* (c'est-à-dire, le plus parallèle possible).

1. Parallélisme récursif

Même question que la précédente (compléter le programme MPD 1, défini de façon partielle). Toutefois vous devez maintenant compléter ce programme de façon à ce que la somme des deux vecteurs se fasse de façon *récursive et parallèle*.

2. Parallélisme itératif avec un nombre fixe de processus

Soit le programme (partiel) MPD 2, qui lui aussi calcule la somme de deux vecteurs. Complétez ce programme de façon à ce qu'il génère un nombre fixe `NBPROCS` de processus itératifs (mais créé de façon dynamique), chacun de ces processus devant traiter (de façon séquentielle et itérative) un sous-intervalle des tableaux — par ex., si `NBPROCS = 6` et `n = 60`, alors le premier processus traitera les éléments 1 à 10 du tableau, le deuxième les éléments 11 à 20, etc.

Vous pouvez (devez) évidemment introduire une ou des procédures auxiliaires.

Note : Pour simplifier, on suppose que le nombre d'éléments `n` est un multiple de `NBPROCS`. Des procédures `inf` et `sup` permettant de déterminer les bornes inférieure et supérieure de la tranche de tableau traitée par le `i`ème processus vous sont aussi fournies.

3. Découplage producteur/consommateur

Soit le programme MPD 3 présenté dans les notes de cours du chapitre "Le langage MPD" (p. 14). Ce programme, à l'aide de deux processus (créés de façon statique à l'aide de déclarations `process`), permet de générer une suite de valeurs et ensuite de vérifier si ces valeurs sont paires ou non.

Dans son état actuel, les deux processus sont fortement dépendants l'un de l'autre. Ainsi, le processus qui génère un nombre ne peut le faire que si le processus qui vérifie le nombre précédent a complété son travail. Cette version du programme conduit donc à une *alternance* stricte entre les deux processus (d'où un temps d'exécution strictement linéaire).

Modifiez le programme pour permettre une plus grande indépendance entre les deux processus, permettant ainsi au processus qui génère les nombres d'en générer *plusieurs* sans nécessairement devoir attendre que les éléments précédents aient été traités.

4. Parallélisme récursif

Soit le programme (partiel) MPD 3, qui détermine le nombre d'occurrences d'une clé dans un tableau.

Complétez ce programme de façon à ce que le travail se fasse de façon *récursive et parallèle*. Toutefois, la procédure récursive que vous devez définir ... ne doit pas être une *fonction*, c'est-à-dire, ne doit *pas* retourner un résultat par l'intermédiaire d'un `returns` ; elle doit plutôt modifier une variable globale appropriée à chaque fois que la clé est rencontrée (attention aux conflits d'accès).

Vous pouvez (devez) évidemment introduire une ou des procédures auxiliaires.

5. Parallélisme itératif

Soit le programme (partiel) MPD 4, qui trouve la valeur maximum dans un tableau (unidimensionnel) d'éléments. Complétez ce programme de façon à ce qu'il génère un nombre fixe `NBPROCS` de processus itératifs (crées de façon dynamique), chacun de ces processus devant traiter (de façon séquentielle et itérative) un sous-intervalle du tableau.

Pour simplifier, on suppose que le nombre d'éléments `n` est un multiple de `NBPROCS`.

Programme MPD 1 Programme pour somme de deux vecteurs

```
resource sommeVecteurs()
  # Lecture et verification du nombre d'elements a generer et a traiter
  int n; getarg(1, n);
  if ( n <= 0 ) { write( "*** Erreur: n <= 0 " ); stop(1); }

  int a[n], b[n], c[n];

  # Generation (aleatoire) des elements d'un tableau.
  procedure generer( ref int a[*], int n ) {
    for [i = 1 to n] { a[i] = int(random(1, n)); }
  }

  # Impression des elements d'un tableau.
  procedure imprimer( int a[*], int n ) {
    for [i = 1 to n-1] { writes( a[i], ", " ); }
    write( a[n] );
  }


  # Partie a completer
  procedure somme( int a[*], int b[*], ref int c[*], int n )
  ...


  # Programme principal

  # Generation aleatoire des elements
  generer(a, n); generer(b, n);

  # Calcul de la somme
  somme( a, b, c, n );

  # Impression des resultats
  write( "c = a + b" );
  writes( "a:: " ); imprimer( a, n );
  writes( "b:: " ); imprimer( b, n );
  writes( "c:: " ); imprimer( c, n );
end
```

Programme MPD 2 Programme pour somme de deux vecteurs avec parallélisme itératif (nombre fixe de processus créés de façon dynamique)

```
resource sommeVecteurs()
  const int NBPROCS = 6;

  # Lecture et verification du nombre d'elements a generer et a traiter.
  int n; getarg(1, n);
  if ( n <= 0 | (n % NBPROCS) != 0 ) {
    write( "*** Erreur: n <= 0 | ( n %", NBPROCS, ") != 0" ); stop(1);
  }

  int a[n], b[n], c[n];

  # Generation (aleatoire) des elements d'un tableau.
  ...

  # Impression des elements d'un tableau.
  ...

  # Bornes inferieure et superieur pour la partie de tableau traitee
  # par le processus numero i.
  procedure inf( int i, int n ) returns int r
  { r = (i-1) * (n / NBPROCS) + 1; }
  procedure sup( int i, int n ) returns int r
  { r = i * (n / NBPROCS); }



  # Partie a completer
  procedure somme( int a[*], int b[*], ref int c[*], int n )
  ...



  # Programme principal

  # Generation aleatoire des elements
  ...

  # Calcul de la somme
  somme( a, b, c, n );

  # Impression des resultats
  ...
end
```

Programme MPD 3 Programme pour déterminer le nombre d'occurrences d'une clé dans un tableau avec parallélisme récursif

```
resource trouverNbOccurrences()
  # Lecture et verification du nombre d'elements a generer et a traiter
  int n; getarg(1, n);
  if ( n <= 0 ) { write( "*** Erreur: n <= 0" ); stop(1); }

  int a[n];

  # Variable globale pour le nombre total d'occurrences.
  int nbOccs = 0;
  ...

  # Generation (aleatoire) des elements du tableau a.
  ...

  # Impression des elements du tableau a.
  ..



  procedure nbOccurrences( int cle, int a[*], int i, int j )
  # Partie a completer
  ...



  #
  # Programme principal
  #

  # Generation aleatoire des elements
  generer(a, n);

  # Cle a rechercher
  int cle; getarg(2, cle);

  # Recherche de l'element
  nbOccurrences( cle, a, 1, n );

  # Impression des resultats
  writes( "a:: " ); imprimer( a, n );
  write( "La cle", cle, "apparait", nbOccs, "fois" );
end
```

Programme MPD 4 Programme pour trouver l'élément maximum dans un tableau avec parallélisme itératif (nombre fixe de processus créés de façon dynamique)

```
resource trouverMaximum()
  const int NBPROCS = 6;

  # Lecture et verification du nombre d'elements a generer et a traiter
  ...

  int a[n];

  # Variable globale pour le nombre total d'occurrences.
  int maxGlobal;
  ...


  # Generation (aleatoire) des elements du tableau a.
  ...

  # Impression des elements du tableau a.
  ...

  procedure inf( int i, int n ) returns int r
  { r = (i-1) * (n / NBPROCS) + 1; }
  procedure sup( int i, int n ) returns int r
  { r = i * (n / NBPROCS); }

  procedure trouverMax( int a[*], int n )
  # Partie a completer
  ...


  #
  # Programme principal
  #

  # Generation aleatoire des elements
  ...

  # Recherche de l'element maximum
  trouverMax( a, n );

  # Impression des resultats
  writes( "a:: " ); imprimer( a, n );
  write( "La valeur maximum est", maxGlobal );
end
```