Exercices de programmation OpenMP

Exercice 1.

On veut écrire un programme parallèle qui transforme une image en niveaux de gris en son négatif :

- un pixel en niveaux de gris est représenté par un octet non signé *g*
- le négatif est calculé par *n* = 255 – *g*

Question 1.

Dans cette question, on considère qu'une image est représentée comme un vecteur de pixels (ligne par ligne).

Le programme séquentiel suivant réalise le traitement demandé :

```
#define SIZE 1024
unsigned char image[SIZE*SIZE] = {...};
unsigned char negatif[SIZE*SIZE];

int main(){
     for (int i=0; i<SIZE*SIZE; i++)
          negatif[i] = 255 - image[i];
}
```

Identifier les dépendances éventuelles entre les calculs à réaliser. Ecrire une version parallèle OpenMP de ce programme.

Question 2.

Dans cette question, on considère qu'une image est représentée comme une matrice à deux dimensions de pixels :

```
unsigned char image[SIZE][SIZE] ;
```

Ecrire un programme OpenMP qui réalise le traitement demandé.

Exercice 2.

On veut écrire un programme qui initialise un vecteur V de N entiers de la manière suivante :

- V[0] = V[1] = 1
- pour tout i, 2 ≤ i < N, V[i] = V[i-1] + V[i-2]

Identifier les dépendances éventuelles entre les calculs à réaliser. Ecrire une version parallèle OpenMP de ce programme.

Exercice 3.

On veut écrire un programme parallèle OpenMP qui calcule le produit de deux matrices A et B de dimensions NxN dans une matrice C de même dimensions.

Question 1.

Identifier les dépendances éventuelles entre les calculs à réaliser. Quels calculs peuvent être réalisés en parallèle (c'est-à-dire simultanément, par des threads différents) ?

Question 2.

Ecrire une version parallèle OpenMP de ce programme.

Exercice 4.

On considère un programme dont le rôle est de simuler la propagation de la chaleur dans une tige métallique :

- la tige est initialement à 0°C
- on place une de ses extrémités dans un four à 100°C et l'autre dans de la glace à 0°C
- le programme calcule les températures que l'on doit observer tout au long de la tige après stabilisation

Le calcul consiste à découper la tige en N segments et à itérer selon le temps, en supposant qu'à un instant donné, la température d'un segment est égale à la moyenne pondérée des températures du segment lui-même et de ses deux voisins immédiats. Le nombre d'itérations (TMAX) est choisi suffisamment grand pour espérer que la température de chaque segment converge.

Le programme séquentiel est le suivant :

```
#define N 2050
float T[N] = {100.0, 0.0, 0.0, …, 0.0};
float tmp[N] ;
tmp[0] = T[0] ; tmp[N-1] = T[N-1] ;

for (int t = 0 ; t < TMAX ; t++){
     for (int s = 1 ; s < N-1 ; s++)
          tmp[s] = T[s] ;
     for (int s = 1 ; s < N-1 ; s++)
          T[s] = (tmp[s-1] + 4*tmp[s] + tmp[s+1]) / 6;
}
```

Question 1.

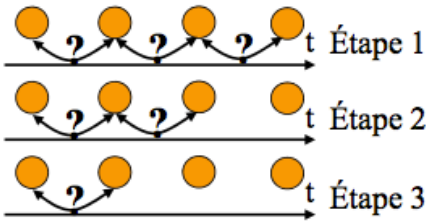
Identifier les dépendances éventuelles entre les calculs à réaliser. Quels calculs peuvent être réalisés en parallèle (c'est-à-dire simultanément, par des threads différents) ? Des synchronisations sont-elles nécessaires ?

Question 2.

Ecrire une version parallèle OpenMP de ce programme.

Exercice 5.

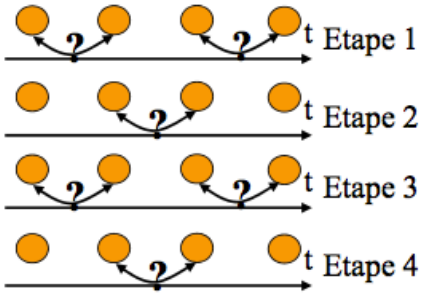
L'algorithme du tri-bulle compare toutes les paires d'éléments adjacents d'un vecteur et les échange si le premier est supérieur au deuxième :



Question 1.

Est-ce que cet algorithme vous semble parallélisable ? Pourquoi ?

Une autre version, de même complexité, compare (a) dans un premier temps, les paires dont le premier élément est de rang pair, (b) dans un second temps, celle dont le premier élément est de rang impair :



Question 2.

Ecrire un programme parallèle réalisant cet algorithme, en utilisant des directives OpenMP.

Exercice 6.

Ecrire un programme parallèle OpenMP qui cherche le maximum dans un vecteur de N éléments :

a) en passant par des maximums « partiels » (chaque thread recherche le maximum sur un sous-ensemble des éléments du vecteur)
b) en utilisant un maximum partagé unique, sans clause de réduction
c) en utilisant une clause de réduction appropriée

Exercice 7.

Ecrire un programme parallèle OpenMP qui compte le nombre d'entiers premiers entre 1 et N.
Une version séquentielle pourrait être :

```
int num_primes = 0;
for (int i = 1 ; i<N ;i++){
     bool is_prime = TRUE ;
     for (int j=2 ; j<i ; j++){
          if (i%j == 0){
               is_prime = FALSE ;
               break ;
          }
     }
     if (is_prime)
          num_primes++ ;
}
```

Exercice 8.

On considère un programme séquentiel suivant :

```
float A[N+2][N+2], diff=0 ;

int main(){
     read(N) ;
     Initialiser(A) ;
     Calculer(A) ;
}
void Calculer(float **A){
     char fini=0;
     float diff=0, temp;
     while (!fini) {
          diff = 0;
          for (int i=1; i<=N; i++)
               for (int j=1; j<=N; j++){
                    temp = A[i][j];
                    A[i][j]=0.2*(A[i][j]+A[i][j-1]+A[i-1][j]+A[i][j+1]+A[i+1][j]);
                    diff = diff + abs(A[i][j] – temp);
               }
          if (diff /(N*N) < SEUIL)
               fini = 1;
     }
}
```

A chaque itération de la boucle externe, chaque élément de la matrice A est remplacé par une moyenne pondérée de lui-même et de ses 4 voisins immédiats (Nord, Sud, Est et Ouest). Les mises à jour se font dans la matrice A elle-même (et non pas dans une matrice intermédiaire). Lorsqu'il calcule la nouvelle valeur d'un élément, le programme calcule aussi la différence entre cette nouvelle valeur et l'ancienne. Si la moyenne des différences observées sur l'ensemble des éléments est inférieure à un seuil prédéfini, le calcul se termine.

Le code séquentiel fait apparaître des dépendances entre les calculs d'une même itération de la boucle externe : un élément est calculé avec les nouvelles valeurs de ses voisins Nord et Ouest et avec les anciennes valeurs de ses voisins Sud et Est. On suppose que l'on sait que l'on peut ignorer ces dépendances (induites par la formulation séquentielle) et qu'on peut se permettre de calculer les éléments dans un ordre différent (par exemple, calculer un point avec l'ancienne valeur de son voisin Ouest ou la nouvelle valeur de son voisin Est), sans que cela nuise à la convergence de l'algorithme.

Ecrire une version parallèle OpenMP de ce programme.

Exercice 9.

On considère le programme suivant qui détermine les maximums locaux d'un vecteur (un maximum local est un élément du vecteur qui est supérieur à l'élément précédent et à l'élément suivant) :

```
float tab[N] ;
float max[N], ;
int nb_max_locaux = 0 ;

for (i=1; i<N-1; i++){
    if ((tab[i-1] < tab[i]) && (tab[i+1]<tab[i])) {
      max[nb_max_locaux]=tab[i];
      nb_max_locaux++;
    }
}
```

Proposer une parallélisation OpenMP de ce programme,

Exercice 10.

Dans les problèmes de type *N-bodies*, on considère un espace dans lequel évoluent un ensemble de corps qui interagissent les uns avec les autres.

Ce modèle est utilisé dans de nombreux domaines, comme par exemple l'astrophysique : le but est d'étudier le déplacement des étoiles dans le temps, ce déplacement étant lié aux forces de gravitation.
Si l'on considère une paire d'étoiles dans ce système, chacune applique à l'autre une force $\vec{f} = \frac{G.m_1.m_2.(\overrightarrow{x_1}-\overrightarrow{x_2})}{|\overrightarrow{x_1}-\overrightarrow{x_2}|^3}$ où *G* est une constante, m_1 et m_2 sont les masses respectives des deux étoiles, et $\overrightarrow{x_1}$ et $\overrightarrow{x_2}$ leurs vecteurs position.
Le temps est discrétisé, et, à chaque pas de temps, on calcule les forces qui s'appliquent à chacune des étoiles, puis on met à jour les données qui la concernent (position, vitesse et accélération).

La solution de base consiste à :

- calculer, à chaque étape, les forces subies par chaque étoile de la part de chacune des autres étoiles
- faire la somme de toutes les forces subies par chaque étoile
- calculer, à partir des forces subies, les nouvelles positions, vitesse et accélération de chaque étoile

et ce sur un ensemble de pas de temps (itérations) donné.

Le programme séquentiel est le suivant :

```
for (int  t=0 ; t<NB_PAS ; t++){
     for (int i=0 ; i<N ; i++) {
          force[i] = 0 ;
          for (int j=0 ; j<N ; j++)
               force[i] = force[i] + interaction(data, i,j) ;
     }
     for (int i=0 ; i<N ; i++)
          data[i] = update(data, force, i) ;
}
```

où `data[i]` est une structure de données qui comprend la position, la vitesse et l'accélération de l'étoile `i`, `interaction(i,j)` est une fonction qui renvoie la force appliquée par l'étoile `j` à l'étoile `i` (cette fonction lit `data[i]` et `data[j]` mais ne les modifie pas) et update(data,force,i) est une fonction

qui calcule les nouvelles position, vitesse et accélération de l'étoile `i` en fonction de la force qu'elle subit.

Question 1.

Analyser les dépendances entre calculs et écrire une version parallèle OpenMP de ce code.

Question 2.

L'algorithme séquentiel ne tient pas compte du fait que les forces sont symétriques : si $\overrightarrow{f_{ij}}$ représente la force appliquée par l'étoile *i* à l'étoile *j*, on a $\overrightarrow{f_{ij}} = -\overrightarrow{f_{ji}}$. Il n'est donc pas nécessaire de calculer séparément l'intensité de ces deux forces.

Pour prendre en compte cette observation, on pourrait modifier l'algorithme séquentiel de calcul des forces de la manière suivante :

```
for (int  t=0 ; t<NB_PAS ; t++){
     for (int i=0 ; i<N ; i++) {
          force[i] = 0 ;
          for (int j=0 ; j<i ; j++){
               float f = interaction(data, i,j) ;
               force[i] = force[i] + f ;
               force[j] = force[j] - f
     }
     for (int i=0 ; i<N ; i++)
          data[i] = update(data, force, i) ;
}
```

Ecrire une version parallèle OpenMP de ce code.

Quels sont les inconvénients de cette version ? Proposer une solution pour les atténuer.