Sujets de TD n° 1

On considère l'approche suivante pour paralléliser un programme :

1. Parmi tous les calculs réalisés dans le programme, chercher ceux qui peuvent être faits en parallèle (sont indépendants les uns des autres)
   NB : si la version séquentielle du programme ne présente pas de parallélisme possible, vérifier qu'elle n'introduit pas de fausses dépendances. Il peut être nécessaire de modifier la version séquentielle pour faire apparaître du parallélisme.
2. Une fois tous les calculs parallélisables identifiés, imaginer une répartition de ces calculs sur un nombre de threads visé. Différents objectifs peuvent être considérés :
   - minimiser les communications/synchronisations nécessaires entre threads pour obtenir un résultat correct
   - équilibrer la charge entre threads
   - faciliter l'écriture ultérieure du programme parallèle
3. Ecrire la version parallèle (threads POSIX, OpenMP, MPI, …)

Dans cette série d'exercices, on s'intéresse aux deux premières étapes.

Exercice 1.

On considère un programme qui, à partir d'une matrice A à deux dimensions (N lignes et P colonnes), calcule la moyenne des éléments de A et génère une matrice B de mêmes dimensions telle que :

- B[i][j] = 1 si A[i][j] est supérieur à la moyenne des éléments de A
- B[i][j] = 0 sinon

Une version séquentielle de ce programme est donné ci-dessous :

```
int main(){
     int A[N][P], B[N][P] ;
     int somme = 0 ;
     float moyenne ;
     for (int i=0 ; i<N ; i++)
          for (int j=0 ; j<P ; j++)
               somme = somme + A[i][j] ;
     moyenne = somme / (N*P) ;
     for (int i=0 ; i<N ; i++)
          for (int j=0 ; j<P ; j++)
               if (A[i][j] >= moyenne)
                    B[i][j] = 1 ;
               else
                    B[i][j] = 0 ;
}
```

Question 1.

Pour chaque nid de boucles, déterminer si les itérations peuvent être exécutées en parallèle (représenter quelques itérations sur un graphe de dépendance).

Question 2.

Si ce n'est pas le cas, proposer une transformation du programme qui facilite la parallélisation. Comment peut-on agglomérer les calculs en N threads ? En N/4 threads ? Quelles communications/synchronisations sont nécessaires ?

Exercice 2.

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 des températures des deux segments voisins. Le nombre d'itérations (TMAX) est choisi suffisamment grand pour espérer que la température de chaque segment converge. Le programme séquentiel est le suivant :

```
int temp[N] = {100, 0, 0, 0, 0, …, 0};

/* calcul de d'évolution des températures */
for (int t = 0 ; t < TMAX ; t++){
     for (int s = 1 ; s < N-1 ; s++)
          temp[s] = (temp[s-1] + temp[s+1]) / 2;
}
```

NB : on ne calcule pas les éléments `temp[0]` et `temp[N-1]` qui conservent les valeurs 0 et 100.

Question 1.

Identifier les dépendances entre les calculs imposées par la version séquentielle en représentant deux itérations de la boucle t sur un graphe de dépendance, avec N=4.

Question 2.

On suppose qu'on peut ignorer, dans la version parallèle, les dépendances existant dans la version séquentielle (calcul de `temp[s]` après celui de `temp[s-1]` et avant celui de `temp[s+1]`) sans que ce soit un obstacle à la convergence des résultats. Toutefois, le calcul d'un segment à l'itération t doit se faire avec les valeurs des segments voisins calculées à l'itération t ou t-1.

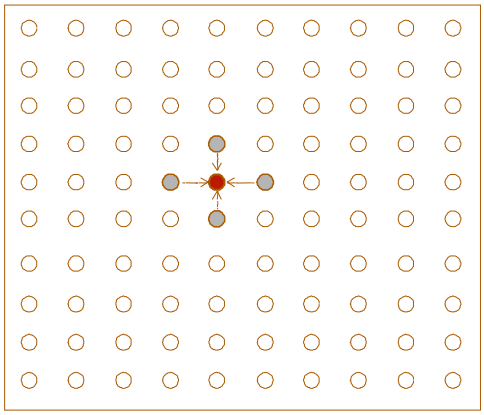
Proposer une agglomération en (N-2)/64 threads dans ce cas, en identifiant les communications/synchronisations à assurer.

Exercice 3.

On considère un programme qui résout une équation différentielle partielle sur une grille, en utilisant la méthode des différences finies. Il travaille sur un tableau à deux dimensions (grille) de $(n+2)\times(n+2)$ éléments.

Le calcul se fait en plusieurs itérations. A chaque itération, le programme traite tous les points et remplace chaque valeur par une moyenne pondérée d'elle-même et de ses 4 voisines immédiates (Nord, Sud, Est et Ouest). Les mises à jour se font dans la grille elle-même (et non pas dans une grille intermédiaire), de sorte que, si on traite la grille ligne par ligne, en partant depuis le coin supérieur gauche, 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. 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, on dit que la solution a convergé et le calcul se termine à la fin de l'itération en cours.



Le code séquentiel de ce programme est donné ci-dessous.

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

int main(){
     read(n) ;
     Initialiser(A) ;
     Calculer(A) ;
}

void Calculer(float **A){
     int i, j, fini=0;
     float diff=0, temp;
     while (!fini) {
          diff ← 0;
          for (i=0; i<n; i++)
               for (j=0; 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;
     }
}
```

Question 1.

Représenter les dépendances existant entre les calculs des différents éléments de la matrice.

Question 2.

Pour les programmes structurés en nids de boucles, une manière simple d'identifier la concurrence consiste à considérer chacune des boucles et à déterminer si ses itérations peuvent être exécutées en parallèle. Peut-on décomposer le noyau de résolution d'équations en tâches parallèles à partir de cette méthode ?

Question 3.

Lorsque les boucles ne peuvent pas être parallélisées simplement, on doit examiner plus finement les dépendances entre calculs pour pouvoir extraire de la concurrence.
Peut-on, à partir du schéma de dépendances, définir des ensembles de points calculables en parallèle ? Quels sont les inconvénients de cette solution ?

Une autre possibilité pour paralléliser un programme consiste à exploiter la connaissance que l'on a du problème, en prenant des libertés par rapport au programme séquentiel. Dans le cas du noyau de résolution d'équations, la méthode de calcul utilisée (méthode de Gauss-Seidel) n'est pas une méthode exacte : elle itère jusqu'à convergence des résultats. Aussi, on peut se permettre de calculer les points à jour dans un ordre différent, du moment que chaque point est mis à jour suffisamment souvent. Par exemple, si on accepte d'utiliser les valeurs de l'itération précédente pour les voisins Nord et Ouest (methode de Jacobi), le système convergera quand même, mais peut-être plus lentement. C'est cette méthode qu'on considère dans la suite de l'exercice.

Question 4.

On suppose que n (largeur et hauteur de la grille) est un multiple de p^2, nombre de processeurs disponibles, et on suppose que l'on veut créer autant de threads qu'il y a de processeurs. On veut attribuer les tâches (calculs de points) aux threads de manière statique et équitable (chaque thread aura donc $n \times n/p^2$ points à calculer).
Pour chacune des répartitions suivantes, évaluer le coût des communications (nombre d'accès à des données produites par un autre thread) :

- allocation de n/p^2 lignes à chaque thread, de manière entrelacée (i.e. le thread i traite les lignes i, $i+p^2$, $i+2p^2$, ...)
- allocation de n/p^2 lignes consécutives à chaque thread (i.e. le thread i traite les lignes $i \times p^2$ à $(i+1) \times p^2-1$)
- allocation d'une sous-grille carrée, de dimensions $n/p \times n/p$ à chaque thread

Pourquoi est-il souhaitable de limiter le nombre de communications ? Quelle est la répartition la plus efficace ?

Exercice 4.

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 en pas, et, à chaque pas, 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.

L'algorithme séquentiel est le suivant :

```
pour t=0 à nb_pas-1 faire
     début
          pour i=0 à N-1 faire
               début
                    force_subie[i] ← 0 ;
                    pour j=0 à N-1 faire
                         force_subie[i] ←
                          force_subie[i] + Interaction(i,j) ;
               fin
          pour i=0 à N-1 faire
               début
                    position[i] ← … ;
                    vitesse[i] ← … ;
                    accélération[i] ← … ;
               fin
     fin
```

Question 1.

A partir de l'analyse de l'algorithme séquentiel, définir des tâches indépendantes.

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 :

```
pour i=0 à N-1 faire
     début
          force_subie[i] ← 0 ;
          pour j=0 à i-1 faire
               début
                    force ← Interaction(i,j) ;
                    force_subie[i] ← force_subie[i] + force;
                    force_subie[j] ← force_subie[j] - force;
               fin
     fin
```

Quels sont les inconvénients de cette modification en ce qui concerne la définition de tâches indépendantes ?