8INF856
Programmation sur architectures parallèles
Devoir 2

(À remettre au plus tard lundi le 27 octobre 2014)

- **Les étudiants sont invités à travailler en équipe de 2 ou 3.**
- **Vous serez évalué pour la rectitude et la qualité de vos réponses.**

1. Nous avons vu en classe l'exemple suivant qui calcule de façon parallèle le produit de deux matrices carrés d'ordre n en utilisant le langage Cilk:

```
P-Square-Matrix-Multiply(A,B,n)
  parallèle for i=1 to n do
      parallèle for j=1 to n do
         C[i,j]=0
         for k=1 to n do
            C[i,j] = C[i,j] + A[i,k]*B[k,j]
   return C
```

   Nous avons vu que le travail de cet algorithme est $\Theta(n^3)$ et la durée est $\Theta(n)$.

   (a) Modifiez cet algorithme afin d'obtenir une durée de $\Theta(\log n)$.

   (b) Indiquez quel est le travail de l'algorithme modifié.

2. Dans cet exercice vous devez implémenter l'algorithme récursif parallèle de fusion que nous avons vu en classe ainsi que l'algorithme séquentiel. Vos programmes devront être écrits en C et, pour le programme parallèle, vous devrez utiliser OpenMP. Votre programme doit lire un entier n en entrée afin d'initialiser un tableau T avec les n premiers nombres pairs suivis des n premiers nombres impairs. Ce sont ces deux moitiés de tableaux que vous devez fusionner. Vous pouvez supposer que n est une puissance de deux. Testez vos programmes et indiquez à partir de quelle valeur de n votre solution parallèle est plus efficace que l'algorithme séquentiel (ne comptez pas le temps nécessaire pour initialiser T.)

   Je vous demande de procéder de la façon suivante:

   - Dans votre répertoire maison sur `dim-linuxmpi1.uqac.ca` , vous devez créer un répertoire dont le nom est Devoir2
   - Vos deux programmes devront être dans des fichiers d2s.c (séquentiel) et d2p.c (parallèle).
   - Vos programmes doivent pouvoir être compilés sans faute à l'aide de la commande gcc. Pour votre programme parallèle utilisez la commande suivante: `gcc -fopenmp d2p.c`
   - Vous devez me faire parvenir par courriel un document contenant les informations suivantes:
     (a) Le nom des coéquipiers
     (b) Le nom d'usager et le mot de passe du compte ou se trouve le programme
     (c) La réponse au numéro 1 et 3a ainsi que l'analyse des programmes des numéros 2 et 3.

3. Étant donné un tableau de n entiers $T[0 \cdots n-1]$, on veut construire un second tableau $S[0 \cdots n-1]$ tel que $S[i] = \sum_{j=0}^{i} T[j]$. L'algorithme récursif parallèle suivant permet de résoudre ce problème avec une durée $T_\infty(n) = T_\infty(n/2) + \Theta(\log n) = \Theta(\log^2 n)$:

```
SP1(T, n)
   if (n==1) return
   parallel for i=0 to n/2 -1 do
       S[i]=T[2*i] + T[2*i+1]
   SP1(S, n/2)
   parallel for i=0 to n/2 -1 do
       T[2*i+1]=S[i]
       T[2*i+2]=S[i]+T[2*i+2]
```

   (a) Écrivez une version itérative de cet algorithme. La durée de votre solution doit être $\Theta(\log n)$.

   (b) Implémentez les deux versions de l'algorithmes avec OpenMP et comparez leur temps d'exécution. Vos deux programmes doivent être placé dans le répertoire Devoir2 sous le nom SP1.c et SP2.c, respectivement.

Remarque: Vous aurez besoin d'utiliser la directive `#pragma omp task` afin d'implémenter la récursion avec openmp. Je vous suggère aussi d'utiliser la fonction `omp_get_wtime()` pour déterminer le temps d'exécution d'un thread.