Examen du cours AMS-I03
Programmation hybride et multi-cœurs

Vendredi 14 février 2020 - durée 3 heures
Supports de cours autorisés.

La syntaxe des lignes de code que vous écrirez ne sera pas évaluée (oubli de ";" ou ordre des arguments dans l'appel des fonctions par exemple), par contre, bien insérer les pragmas et appeler les fonctions nécessaires.
Ajouter des lignes commentaires dans le code que vous écrivez.

Question de cours n° 1 (1 point)

Définir les notions de localité spatiale et localité temporelle. Donnez un exemple simple pour illustrer chacune de ces deux notions.

Partie 1 : parallélisation en mémoire partagée

Question de cours n° 2 (1 point)

Rappeler les différences principales entre la programmation OpenMP «grain fin» (fine-grain) et «gros grain» (coarse-grain).

Question n° 3 (5 points)

Soient A une matrice triangulaire inférieure ($A_{i,j} = 0$ si $j > i$) de taille $n \times n$, V et W deux vecteurs de taille n.
Le pseudo-code suivant calcule le produit matrice vecteur : $w = Av$, en tenant compte de la structure triangulaire inférieure de A:

```
1  input: matrix A, vector V, int n
2  output: vector W
3
4  for i = 0 to n-1
5      s ← 0.0
6      for j = 0 to i
7          s ← s + A(i,j)*V(j)
8      end
9      W(i) ← s
10 end
```

- Paralléliser suivant le modèle "OpenMP grain fin" ce code
- Expliquer pourquoi le parallélisme grain fin ne sera probablement pas optimal.
- Paralléliser suivant le modèle "OpenMP grain grossier" (ou gros grain) ce code en tenant compte de la structure de A

Partie 2 : parallélisation hybride

Question de cours n° 4 (1 point)

Dans le modèle de programmation hybride MPI-OpenMP, décrivez des avantages espérés par rapport à une programmation purement MPI et une programmation purement OpenMP.

Question n° 5 (5 points)

Le but de cet exercice est de calculer une valeur approchée de π par intégration numérique en modèle de programmation mixte MPI+OpenMP.

La machine de calcul dispose de N nœuds (machine à 1 processeur), chaque processeurs étant composé de M cœurs. Le nombre total de cœurs est donc de $N \times M$.

Soit la formule suivante utilisée pour calculer la valeur du nombre π :

$$\pi = \int_0^1 \frac{4}{1+x^2}\, dx$$

Le programme séquentiel suivant permet de calculer une valeur approchée de cette intégrale en utilisant la méthode du trapèze. Cette méthode simple consiste à remplir la surface sous la courbe par une série de petits rectangles. Lorsque la largeur des rectangles tend vers zéro, la somme de leur surface tend vers la valeur de l'intégrale (et donc vers π).

```
1  #include <iostream>
2
3  int main() {
4    int lNumSteps = 100000000;
5    double lStep = 1.0/lNumSteps;
6    double lSum = 0.0, x;
7
8    for (int i=0; i<lNumSteps ; ++i ) {
9      x = (i+0.5) * lStep;
10     lSum += 4.0/(1.0 + x*x);
11   }
```

```
12    double pi = lSum*lStep;
13
14    std::cout.precision(15) ;
15    std::cout << "pi = " << pi << std::endl;
16
17    return 0;
18  }
```

- Ajouter une parallélisation OpenMP (type "grain fin").
- Ajouter une parallélisation MPI pour obtenir une parallélisation hybride.

Partie 3 : parallélisation sur GPU

Question de cours n° 6 (2 points)

- Décrivez en quelques lignes, le modèle de parallélisme utilisé dans un GPU.
- Quelles sont les principales différences entre CUDA et OpenCL (mode de définition d'un noyau de calcul et exécution de ce noyau sur un GPU) ?

Question n° 7 (5 points)

Soit A une matrice "proche" de la matrice identité Id ($Id_{i,j} = 1$ si $i = j$ et 0 sinon). L'algorithme suivant calcule une approximation de la matrice inverse de A :

$$A^{-1} \approx Id + \lim_{k=1}^{K} (Id - A)^k$$

On suppose que cette approximation converge quand K tend vers ∞.

- Écrire un programme principal en C ou C++, qui implémente cet algorithme. Ce programme exécutera une série de noyaux Cuda en s'arrêtant dès que le dernier terme ajouté à la somme ci-dessus est inférieur à une quantité donnée.
- On fera attention à limiter le nombre de transferts entre la mémoire du CPU et celle du GPU.

Total : 20 points