Examen du cours I3
Programmation hybride et multi-cœurs

Vendredi 15 février 2019 - durée 3 heures
Supports de cours autorisés.

Dans la première partie (questions 1 à 3), le langage de programmation que vous utilisez pour répondre aux questions (pseudo-code, C, C++, fortran, OpenMP, MPI) importe peu et la syntaxe ne sera pas évaluée.
Du moment que vous indiquiez clairement (par des commentaires) ce que font les ligne de codes que vous ajoutez pour répondre aux questions.
Dans la deuxième partie (questions 4 à 8), la syntaxe des instructions Cuda devra être correcte.

Question nº 1 (2 points)

- Définir les notions de localité spatiale et localité temporelle.

On suppose que les coefficents des matrices sont rangés lignes par lignes (comme en C/C++). Ci-dessous, deux versions d'un pseudo-code qui calcule le produit matrice-vecteur :

```
1  input: matrix A, vector V
2  output: vector W
3
4  N ← A.nrows()
5  M ← A.ncolumns()
6
7  for i = 0 to N-1
8      W(i) ← 0.0
9      for j = 0 to M-1
10         W(i) ← W(i) + A(i,j) * V(j)
11     end
12 end
```

```
1  input: matrix A, vector V
2  output: vector W
3
4  N ← A.nrows()
5  M ← A.ncolumns()
6
7  for i = 0 to N-1
8      W(i) ← 0.0
9  end
10
11 for j = 0 to M-1
12     for i = 0 to N-1
13         W(i) ← W(i) + A(i,j) * V(j)
14     end
15 end
```

- Laquelle sera probablement plus rapide pour `n` et `m` grands ?
- Même question pour `n` et `m` petits ?
- Justifiez vos réponses.

Question n° 2 (5 points)

On veut paralléliser avec OpenMP, la fonction C++ :

```
1  void maxlocal(std::vector<double> & v,
2                std::vector<int> & imax,
3                std::size_t &nmax) {
4    std::size_t i, n = v.size();
5    std::size_t smax = imax.size();
6    nmax = 0;
7    for (i=1; i<n-1; i++)
8      if ((v[i-1]<v[i]) && (v[i]>v[i+1])) {
9        imax[nmax] = i;
10       nmax += 1;
11       if (nmax >= smax) break;
12     }
13 }
```

Le but de cette fonction est de calculer, sur les composantes d'un vecteur `v`, les indices des `maxima locaux` (i tels que $v_i > v_{i-1}$ et $v_i > v_{i+1}$).
On suppose que le nombre de maxima locaux est inférieur à la taille du vecteur d'entiers `imax` (dans lequel seront rangés les indices de maxima locaux).

La parallélisation de cette fonction en ajoutant une pragma simple suivant le modèle "OpenMP grain fin" est impossible (le compilateur refuse de compiler quand on ajoute la pragma), ou en tout cas compliquée.

- Expliquer pourquoi.
- Modifier la fonction en la parallélisant suivant le modèle "OpenMP grain grossier".
- La version parallèle fournit un résultat correct mais peut-être différent de celui de la version séquentielle. Quelle est cette différence ?
- Question optionnelle : Indiquer une méthode pour obtenir si possible un résultat identique.

Question nº 3 (4 points)
Programmation hybride MPI - OpenMP

On suppose avoir accès á une machine parallèle à N nœuds de calcul où chaque nœud est constitué d'un processeur multi-cœurs avec M cœurs.

On considère une matrice L triangulaire inférieure:

$$\begin{array}{ll} L(i,j) = 0 & \text{pour } i < j \\ L(i,j) \text{ est non nul} & \text{pour } i = j \\ L(i,j) \text{ est quelconque} & \text{pour } i > j \end{array}$$

Un peudo-code séquentiel pour le produit matrice-vecteur utilisant ce type de matrice, s'écrit:

```
1  input: matrix L, vector V
2  output: vector W
3
4  N ← V.n()
5
6  for i = 0 to N–1
7      for j = 0 to i
8          W(i) ← W(i) + L(i,j)*V(j)
9      end
10 end
```

Écrire un pseudo-code qui utilise une programmation hybride MPI-OpenMP pour effectuer ce calcul sur la machine parallèle.
On indiquera notamment dans le pseudo-code :

- Comment la matrice et les vecteurs sont distribués sur les différents nœuds.
- Comment est réparti le travail entre les différents cœurs d'un même nœud.
- Quand sont effectués les échanges de messages.

Il faudra répartir le mieux possible la charge de travail entre les nœuds et les cœurs.

Pour rappel, l'exactitude des syntaxes MPI et OpenMP se sera pas prise prise en compte dans l'évaluation de votre réponse.

Question n° 4 (1 point)

- Donner 2 différences d'architecture matérielle (liées au parallélisme) entre un CPU et un GPU.

Question n° 5 (2 points)

- Qu'est ce qu'un kernel CUDA ?
- Quel est le mot clé permettant de déclarer un kernel CUDA et quelle est la syntaxe spécifique pour lancer l'exécution d'un kernel CUDA ?
- Quelles sont les instructions pour définir le parallélisme utilisé par l'exécution d'un noyau CUDA ?

Question n° 6 (1 point)

- Quelles sont les différences principales dans la définition et l'exécution des noyaux Cuda et OpenCL ?

Question n° 7 (1 point)

- Expliquer ce qu'est la divergence de branches sur GPU.

  Quel est son inconvénient ?

Question n° 8 (4 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