SI4 Programmation Parallèle

Epreuve écrite

5 mai 2022, durée 2h30

1 page A4 recto/verso manuscrite autorisée

2 copies séparées (Exercice 4 à part)

Exercice 1 – (barème approximatif : 5 points)

Algorithmique et Programmation en PRAM, OpenMP et MPI

Voici un code PRAM (très simple) impliquant p processeurs d'une machine PRAM

```
(1)For each Processor j, 0 ≤ j < p in parallel do :
        allocate a local array Res of size p; write its own data into B[j] and in Res[j]
(2)For k=1 to p – 1 in sequential do:
        (3)For each Processor j, 0 ≤ j < p in parallel do:
                Res[(j+k) mod p] = B[(j+k) mod p]
EndFor
```

1. Déterminer dans quelle variante de PRAM est décrit cet algorithme
2. Que fait cet algorithme, et quelle est sa complexité en temps de calcul PRAM (dans la variante de PRAM utilisée) ?
3. Traduire l'algorithme donné dans le rectangle au dessus, en pseudo OpenMP
4. Recommencez, mais en vue de coder en MPI en vous restreignant à une topologie virtuelle en anneau uni-directionnel. Pour cela, expliquer les changements fondamentaux à apporter à l'algorithme, puis donner le code MPI correspondant. Exprimer son coût total en temps parallèle, en supposant qu'un message de longueur L (ici L=1) s'envoie en Beta + L * tau. Donner aussi le nombre total de messages échangés entre les processeurs voisins de l'anneau.

Exercice 2 (barème approximatif : 1 point par question, soit 2 points)

– Quelques questions « en vrac » !

1. Peut-il y avoir une certaine réticence vis-à-vis de « toujours plus de CPUs » concernant une machine parallèle que l'on désire utiliser pour résoudre un problème en parallèle. Servez-vous du principe exprimé par la loi d'Amdhal pour justifier votre réponse.
2. Expliquer brièvement la loi de Moore et quelles sont ses limites actuelles

Exercice 3 (bareme approximatif : 8 points)

– Algorithme pour le calcul du nombre d'inversions

Soit A un tableau (vecteur) de n nombres entiers. On veut concevoir une procédure qui permet de déterminer le nombre d'inversions d'éléments adjacents de A, c.a.d. le nombre d'éléments (immédiatement) adjacents qui ne sont pas correctement ordonnés. Voici une spécification en pseudo-langage

```
procedure nbInversions (tab d'entiers A, int n) returns int nbI
#Precondition n >=1
#Postcondition
# nbI = SUM( 1<= i <n tel que A[i]>A[i+1])
```

Quelques exemples :

- nbInversions ([1,1,1], 3) → 0
- nbInversions ([1,2,2], 3) → 0
- nbInversions ([10,9,8], 3) → 2
- nbInversions ([1,2,3,4,3,2,1,0,1], 9) → 4

- nbInversions ([8,7,6,4,5,3,1,2,3], 9) → 5
- nbInversions ([9,8,7,6,5,1,2,4,3], 9) → 6
- nbInversions ([9,8,7,6,5,1,4,3], 8) → 6

Le but de l'exercice est de donner une version séquentielle, puis des versions parallèles et de comparer les complexités.

1. Décrire en pseudo-langage (impératif) une version itérative séquentielle. Donner la complexité en temps notée Tseq(n).
2. Donner le principe d'une version parallèle récursive, c'est-à-dire par approche « divide and conquer » (sans l'écrire), qui suppose pour simplifier que la taille du vecteur (et donc des sous-vecteurs) est une puissance de 2
3. Donner le principe d'une autre version parallèle, cette fois plus appropriée pour une PRAM, toujours en supposant que la taille du vecteur est une puissance de 2 pour simplifier, en expliquant bien la démarche sous-jacente suivie et en précisant le modèle de PRAM utilisée, sachant que l'entier final *nbI* est en mémoire partagée, accessible en Read et en Write par tous les processeurs de la PRAM.
Bonus +2 points: écrire l'algorithme précis, avec tous les indices bien positionnés
Donner la complexité en temps Tpar(n) et le nombre de processeurs PRAM nécessaires.
Puis son coût « en travail ». Est-il efficace, et si non, dire s'il peut être rendu optimal en travail, et comment.
4. Une variante autour de ce problème (4 points). Au lieu de déterminer nbI (le nombre d'inversions) on veut déterminer pour chaque élément a de A quel est le nombre des éléments non correctement ordonnés qui précèdent l'élément a dans le tableau A

Quelques exemples :

- nbInversionsTab ([1,1,1], 3) → (0,0,0)
- nbInversions Tab([1,2,2], 3) → (0, 0, 0)
- nbInversions Tab([10,9,8], 3) → (0,1,2)
- nbInversions Tab([1,2,3,4,3,2,1,0,1], 9) → (0,0,0,0,1,2,3,4,4)
- nbInversions Tab([8,7,6,4,5,3,1,2,3], 9) → (0,1,2,3,3,4,5,5,5)
- nbInversions Tab([9,8,7,6,5,1,4,3], 8) → (0,1,2,3,4,5,5,6)
- nbInversions Tab([9,8,7,6,5,1,2,4,3], 9) → (0,1,2,3,4,5,5,5,6) : prenons le cas du dernier élément, 3 ; de son point de vue, il y a bien 6 éléments (ceux soulignés) qui le précèdent qui ne sont pas correctement ordonnés par ordre croissant
- nbInversions Tab([5,2,3,6,7,8,9,7,3,2,4,3,2,3,2,1]) → (0,1,1,1,1,1,1,2,3,4,4,5,6,6,7,8) pour un vecteur A de dimension 16.

a) Sans encore écrire ni une version séquentielle, ni une ou deux versions parallèles pour cette nouvelle procédure nbInversionsTab, expliquer en quoi elle diffère du problème initial.

b) Concernant une version PRAM, expliquer qu'on peut reposer sur le traditionnel Prefix mais avec une opération plus élaborée que juste une seule somme de valeurs entre voisins « dans l'arbre binaire ». Pour illustrer cette nécessité de sophistiquer les opérations réalisées, utiliser par exemple les deux derniers exemples fournis ayant 8 valeurs dans A ou en ayant 16. Il y a besoin de modifier la phase de montée et celle de descente.

c) Décrire l'opération à réaliser dans chacune des 3 phases de l'algorithme préfix, en indiquant avec quelles valeurs opérer. Hint : vous commencez avec des 0 à chaque feuille de votre arbre binaire, qui représente le cumul des valeurs non correctement ordonnées. En plus du cumul qui va être accumulé, en remontant dans l'arbre, vous devez conserver de manière bien distincte à chaque nœud de l'arbre, un 0 ou un 1 (selon résultat du test) qu'il faudra additionner à ce qui monte ou à ce qui descendra. En gardant cette sorte de « retenue » dans une variable bien distincte, vous serez alors à même de réaliser correctement la phase de descente de la méthode Prefix.

Exercice 4 (bareme approximatif : 5 points) SUR UNE COPIE SEPAREE, MERCI – GPGPU

1. Dans CUDA il y a la notion de Thread Block et Grid. Expliquez brièvement à quoi elles correspondent
2. Étant donné un tableau unidimensionnel de taille N et un Thre!ad Block en 1D de taille 16, quelle est la Grid nécessaire pour traiter tout le tableau ?
3. Écrivez en pseudo-code le kernel permettant d'élever chaque élément d'un tableau de taille N passé en paramètre au carré.