Master 1 Informatique UNSA
UE « Parallélisme et Répartition »,
Epreuve écrite individuelle
21 Novembre 2008, durée 2h
Tous documents autorisés

Exercice 1 – (barème approximatif : 9 points)

Conception d'un programme pour le calcul de PI, en OpenMP, puis en MPI

Voici un code C séquentiel qui permet de calculer la valeur de PI (3.14….) (on l'a vérifié, il approxime bien ce nombre, et même avec par exemple i=5, il donne une valeur qui commence bien par 3.14).

```
static long num_steps = 100000;
double step;
void main()
{       int i; double x,pi,sum=0.0;
        step = 1.0 / (double) num_steps;
        for (i=1;i<=num_steps;i++){
                x= (i-0.5)*step;
                sum = sum + 4.0 / (1.0 + x*x);
        }
        pi = step * sum
}
```

1. Concevoir une version parallèle, donner un algorithme en pseudo – code en utilisant l'approche OpenMP
2. On veut distribuer ce code sur plusieurs machines, écrivez en pseudo-code une version utilisant MPI.

Vous prendrez bien soin de décrire votre démarche et d'expliquer les problèmes particuliers qui se posent selon que vous utilisez l'une ou l'autre approche, et la manière dont vous les résolvez.

3. Supposons et ce pour chacun des 2 cas, que nous disposons d'un ensemble de processeurs tous homogènes (même puissance) : dites ce que vous préconisez pour découper puis ordonnancer (« scheduler ») les différentes taches parallèles issues de votre implantation.
4. Refaire la même question, mais en supposant que cet ensemble est constitué de processeurs hétérogènes, et que vous ne connaissez pas précisément la puissance de chacun (parce que par exemple, ces processeurs sont partagés avec d'autres utilisateurs qui ont lancé leurs calculs en même temps que les vôtres…)

Exercice 2 (bareme approximatif : 3 points)

– Mesures de performance

Un algorithme séquentiel de décomposition d'une matrice carrée $\boldsymbol{n \times n}$ en une matrice triangulaire supérieure et une matrice triangulaire inferieure a un cout de $\boldsymbol{n^2 . (n-1) / 3}$. Sur une architecture parallèle ayant $\boldsymbol{p}$ processeurs, le temps nécessité par l'algorithme parallèle est de $\boldsymbol{(n^3 - n)/(2p)}$.

1. Quel est le facteur d'accélération (Speed-Up) de cet algorithme
2. Donner la formule d'efficacité de cet algorithme parallèle ? Que vaut-elle quand n tend vers l'infini ? (dans la suite de l'exercice, cette valeur sera appelée : *l'Efficacité Maximale*). Est-

ce que cette *Efficacité Maximale* est la meilleure possible ? Qu'est-ce qui pourrait expliquer que ce ne soit pas le cas ?

3. Vous pouvez donc définir à partir de quelles valeurs de n l'efficacité est pratiquement égale à l'efficacité maximale : par pratiquement, on veut dire que l'efficacité obtenue est au moins de 95% de l'*Efficacité Maximale*.

Exercice 3 (bareme approximatif : 4 points)

– Algorithmique de communication sur anneau de processeurs

Soit un anneau de processeurs. Nous avions déjà donné en TD des algorithmes pour diffuser un message d'un processeur vers tous les autres (diffusion ou broadcast) et pour la diffusion personnalisée (scatter).

Dans cet exercice, on vous demande de donner un autre algorithme, pour cette fois, réaliser un échange total : chaque processeur a une information qu'il veut diffuser à tous les autres. Concevez un algorithme qui tire au mieux partie de la topologie en anneau, et qui bien évidemment, soit le plus rapide possible.

Exercice 4 (bareme approximatif : 4 points)

– MYSTERE !

Soit L une liste chaînée.

Le but de l'exercice est d'expliquer ce que fait cet algorithme PRAM, sachant que chaque élément de la liste est soit négatif (N), soit positif (P).

1. Commencer par le faire tourner sur la liste à 8 éléments
   (e1,P)→(e2,P)→(e3,N)→(e4,P)→(e5,N)→(e6,P) →(e7,P) →(e8,P) →NIL
   Le symbole → matérialisant le suivant dans la liste.
2. Dire dans les différentes phases de l'algorithme quelle variante de PRAM doit-on supposer (EREW, CREW, ou CRCW) ?

Dans l'algorithme ci-dessous sur chaque élément « i », suivant(i) et négatif(i) représentent des pointeurs vers un élément dans la liste (ou vers NIL), et Fini(i) représente un booléen, à Faux initialement.

```
Pour tout i en parallèle faire :
        Si suivant(i) = Nil ou Signe(suivant(i)) == N
        Alors
                Fini(i) = Vrai
                négatif(i)=suivant(i)
        FinSi
Tant que il existe un nœud i tel que Fini(i) = Faux faire
        Pour tout i en parallèle faire :
                Si Fini(i) = Faux
                Alors
                        Fini(i)=Fini(suivant(i))
                        Si Fini(i) = Vrai
                        Alors
                                négatif(i) = négatif(suivant(i))
                        FinSi
                        suivant(i)=suivant(suivant(i))
                FinSi
```