Master 1 Informatique UNSA
UE « Parallélisme et Répartition », - Epreuve écrite individuelle
10 Décembre 2009, durée 2h
Tous documents autorisés

F. Baude & F. Huet

Exercice 1 (barème approximatif : 6 points)

– Algorithme parallèle 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 nb
#Precondition n >=1
#Postcondition
# nb = 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

Le but de l'exercice est de donner une version séquentielle, puis 2 versions PRAM, et de comparer au passage les complexités.

1. Décrire en pseudo-langage (impératif) une version itérative séquentielle. Donner la complexité en temps notée Tseq(n).
2. Donner 2 versions parallèles pour ce problème :

a. une « pure » PRAM, ce qu'on appelle aussi « parallélisme itératif »

b. l'autre, également exprimée en PRAM, mais fondée sur une approche « Diviser pour Régner », que l'on appelle aussi communément « parallélisme récursif ».
Pour chacun des 2 cas, expliquer bien la démarche sous-jacente que vous avez suivie et précisez le modèle de PRAM utilisée. Donner la complexité en temps Tpar(n) et le nombre de processeurs PRAM nécessaires. Puis sa complexité « en travail ». Est-il optimal, et si non, dire s'il peut être rendu optimal et comment.

Exercice 2 (barème approximatif : 1,5 point par question, soit 4,5 points)

– Quelques questions « en vrac » !

1. Que signifie « SMP » , et donnez au moins un exemple de machine « SMP ». En fonction de la taille (nombre de CPUs) d'une telle machine, expliquez quelle architecture de communication est adaptée (bus type Ethernet, réseau cross-bar, réseau multi-étage, …)
2. L'accélération *(speedup)* maximale obtenue avec un algorithme parallèle traitant un problème de dimension fixe est limitée par sa partie séquentielle et non par le nombre de processeurs. Vrai ou faux ? Justifiez votre réponse
3. Dans OpenMP, il est possible de paralléliser une boucle for. Expliquez comment s'effectue cette parallélisation. Y-a-t'il des contraintes à respecter pour qu'une boucle for soit parallélisable ?

Exercice 3 (barème approximatif : 5 points)

– Algorithmique distribuée/répartie sur topologie de processeurs non totalement connectée

Soit un anneau de processeurs, numérotés entre 0 et p-1 (nombre de processeurs connus par la fonction Tot_proc_num()). Adr est un tableau de taille p * L octets, présent sur chaque processeur.

1. .

a. Que fait cet algorithme, lorsque exécuté en parallèle par chaque processeur. Que contient adr à la fin, sur chaque processeur ?

b. Quel comportement (quelle sémantique) doit avoir la primitive d'envoi *Send* pour que cet algorithme ne bloque (*deadlock*) pas ?

c. Quelle est la durée d'exécution parallèle de cet algorithme (l'exprimer en fonction de L et de p, ainsi que de *tw*, le cout de transfert par octet et *ts*, le cout de démarrage d'un envoi de message)? Quelle est la quantité de messages/octets qui circulent en même temps sur le réseau (ceci nous donnant une mesure d'occupation des liens du réseau)

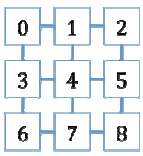
```
q := my_num();
p := Tot_proc_num() ;
adr[q] := my_adr ;// pointeur my_adr représente une info de longueur=L dans la mémoire de chaque processeur
Pour i=1 à p-1 Faire
        Send(adr[q-i+1], (q+1) mod p) // envoi au suivant dans l'anneau
        Receive(adr[q-i], (q-1) mod p) // réception de mon prédécesseur dans l'anneau
```

2. Donner une autre solution pour le même problème résolu par l'algorithme ci-dessus, mais en supposant une topologie de communication plus riche que l'anneau, cad en hypercube de dimension n, avec log(p)=n, contenant p processeurs. Vous répondrez aux mêmes questions que celles posées dans 1.c

Exercice 4 (barème approximatif : 4,5 points)

– Calculs itératifs sur topologie en grille avec MPI

Le but de cet exercice est d'écrire une application distribuée MPI effectuant un calcul itératif synchronisé. Chaque processus possède une valeur *v* et effectue une série d'opérations dessus. Une fois que tous les processus ont effectué leur calcul, alors, ils commencent une nouvelle itération. Une fois un nombre *max* d'itérations atteint, le calcul s'arrête. Les processus seront organisés suivant une topologie de grille 2D, c.a.d que chaque processus a au plus 4 voisins (Nord, Sud, Est, Ouest), comme indiqué dans le schéma ci dessous



1. Sachant que la grille 2D est carrée, que pouvez-vous dire du nombre de processus impliqués dans le calcul ?
2. On considère un processus de rang i, indiquez le pseudo code C/MPI permettant de calculer le rang de ses voisins Nord, Sud, Est et Ouest, si ils existent.
3. Donnez le pseudo code C/MPI permettant de donner sa valeur *v* à ses voisins et de récupérer la leur. Vous devez faire attention à ce que votre code ne *deadlock* pas.
4. Pourquoi le code suivant est-il suffisant pour faire un calcul itératif synchronisé ? Autrement dit, un processus peut-il calculer l'itération *k* alors qu'un autre calcule *k+10* par exemple ?

```
// i est le numéro de l'Itération en cours
for (i=0;i<max;i++) {
     calculLocalSurV();
    //code écrit aux questions 2, 3
}
```