ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF8601: Systèmes informatiques parallèles (Automne 2012)
3 crédits (3-1.5-4.5)

CONTRÔLE PÉRIODIQUE

DATE: Mardi le 30 octobre 2012

HEURE: 9h30 à 11h20

DUREE: 1H50

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Vous avez accès à une grappe de 128 ordinateurs. Chaque ordinateur est constitué d'une partie électronique avec un taux de disponibilité de 0.999 et de deux disques redondants en miroir. Le taux de disponibilité de chaque disque est de 0.99. Vous avez conçu votre application parallèle pour décomposer le problème en 128 morceaux, faire calculer chaque morceau sur un ordinateur différent, et obtenir le résultat à la fin si tous les ordinateurs ont été opérationnels tout au long du calcul. Quelle est la probabilité que toute la grappe soit disponible à un instant donné? **(2 points)**

b) Une grappe de 1024 noeuds est utilisée pour une tâche de calcul qui se termine par une réduction (somme des résultats fournis par chacun des 1024 noeuds). Les 1024 noeuds sont connectés à un commutateur performant qui permet à n'importe quelle paire de noeuds de communiquer en même temps. Toutefois, un noeud donné ne peut envoyer et recevoir qu'un seul message par unité de temps. Propposez une organisation efficace pour effectuer l'opération de réduction. Combien d'unités de temps sont nécessaires pour envoyer les messages requis pour cette opération de réduction? **(2 points)**

c) Vous analysez un programme à paralléliser et trouvez qu'il est constitué d'une phase a) de préparation du problème, une phase b) de calcul matriciel, et une phase c) de regroupement des résultats. En mode sériel, ces phases prennent respectivement a: 10s, b: 4000s c: 20s. En parallélisant le programme sur 128 noeuds, la phase b se retrouve raccourcie par un facteur 128 mais il faut ajouter 20s pour envoyer les données en multi-diffusion aux 128 noeuds et 1s par noeud pour récupérer les résultats. Quel est le temps de calcul sériel? En parallèle? Quel est le facteur d'accélération. **(1 point)**

Question 2 (5 points)

a) Dans votre premier travail pratique, vous avez utilisé la librairie TBB pour paralléliser le problème du dragon, et vous avez étudié comment la librairie TBB découpait le problème en intervalles et assignait les fils d'exécution. Décrivez et expliquez comment la librairie décomposait votre problème du dragon et assignait le travail à plusieurs fils d'exécution. **(2 points)**

b) Vous devez compiler un très gros logiciel constitué de 5000 fichiers. La compilation de chaque fichier prend 0.5s de temps de processeur et 0.4s d'attente après les serveurs de fichiers en réseau. Votre ordinateur contient 8 processeurs. Puisqu'il est facile de paralléliser une telle compilation en répartissant les fichiers entre les processeurs, combien de processus parallèles (ou fils d'exécution) de compilation devrait-on utiliser au minimum pour accélérer le plus possible la compilation? Est-ce qu'il y a un problème à créer beaucoup plus de processus parallèles que ce minimum requis? **(2 points)**

c) Les ordinateurs du laboratoire bénéficient de la technologie hyperthread. Comment cela fonctionne-t-il et quel en est le bénéfice? Est-ce que cela affecte le choix du nombre de processus parallèles recommandés pour accélérer une tâche parallèle? **(1 point)**

Question 3 (5 points)

a) Le programe suivant s'exécute sur un ordinateur du laboratoire utilisé pour les travaux pratiques du cours INF8601. La variable sums est un vecteur d'entiers 32 bits de 64 entrées. Le temps d'exécution de ce programme (temps réel écoulé) pour nb_iter = 1 000 000 000 est, pour nb_thread = (1, 2, 4, 8, 16), de respectivement (2.57s, 3.41s, 4.60s, 3.67s, 2.85s). Comment expliquez-vous que le temps augmente puis diminue en fonction du nombre croissant de fils d'exécution parallèles? Ces ordinateurs contiennent 4 processeurs et une cache L1 avec des blocs de longueur 64 octets. **(2 points)**

```
  omp_set_num_threads(nb_thread);
  #pragma omp parallel for shared(sums)
  for(i = 0; i < nb_iter; i++) {
    sums[omp_get_thread_num()] += i % 2 + 1;
  }
}
```

b) Les programmes suivants s'exécutent sur deux processeurs à mémoire partagée afin de jouer à Roche, Papier, Ciseaux. Chaque processeur choisit un chiffre entre 0 et 2, attend que le choix de l'autre processeur soit prêt et vérifie s'il a gagné, auquel cas il émet un son pour signifier qu'il a gagné. Est-ce que ce programme donnera le résultat attendu sur un ordinateur à cohérence séquentielle? Avec ordonnancement total des écritures? Avec un ordonnancement partiel des écritures? Avec un ordonnancement faible? Quelles sont les barrières mémoire à insérer (et où) afin d'assurer un comportement correct du programme dans tous les cas? **(2 points)**

```
(initialement choix0 = -1; pret0 = 0, choix1 = -1; pret1 = 0)
Processeur 0                        Processeur 1

choix0 = choisir(0,2);              choix1 = choisir(0,2);
pret0 = 1;                          pret1 = 1;
while(pret1 == 0);                  while(pret0 ==0);
if(gagne(choix0, choix1)            if(gagne(choix1, choix0)
{ jouer_musique();                  { jouer_musique();
}                                   }
```

c) Dans les systèmes de mémoire cache avec cohérence par répertoire, il faut stocker pour chaque bloc en cache la liste des processeurs qui en possèdent une copie, par exemple pour leur envoyer un message d'invalidation en cas d'écriture. Une manière naïve de réaliser cela serait d'avoir une matrice d'octets avec une rangée pour chaque bloc de mémoire centrale et une colonne pour chaque processeur. Chaque octet permet d'indiquer si un bloc se trouve sur un processeur donné. Est-ce organisé de cette manière dans les processeurs courants? Comment peut-on réaliser ce répertoire de manière à prendre beaucoup moins d'espace? **(1 point)**

Question 4 (5 points)

a) Le programme suivant effectue la multiplication de deux matrices contenant des valeurs calculées par les fonctions fn1 et fn2. Le temps requis pour exécuter fn1 et fn2 varie beaucoup selon la valeur de leurs deux arguments. Ce programme ne fonctionne pas à la vitesse espérée et on fait appel à vos services d'expert. Proposez des améliorations qui permettent d'accélérer le plus possible ce programme parallèle sans changer le résultat final dans la matrice c. **(2 points)**

```
int i, j, k;
double a[NRA][NCA], b[NCA][NCB], c[NRA][NCB];

#pragma omp parallel shared(a,b,c) private(i,j,k)
{ #pragma omp for
  for (i=0; i<NRA; i++)
    for (j=0; j<NCA; j++)
      a[i][j]= fn1(i,j);
  #pragma omp for
  for (i=0; i<NCA; i++)
    for (j=0; j<NCB; j++)
      b[i][j]= fn2(i,j);
  #pragma omp for
  for (i=0; i<NRA; i++)
    for (j=0; j<NCB; j++)
      c[i][j]= 0;
  #pragma omp for
  for(j=0; j<NCB; j++)
    for (i=0; i<NRA; i++)
      for (k=0; k<NCA; k++)
        c[i][j] += a[i][k] * b[k][j];
```

b) Pour chacun des 4 outils suivants expliquez leur utilité, leur coût en performance, et un exemple de problème pour lequel il pourrait poser un diagnostic efficacement: profileur de fonction Valgrind/Callgrind, outil de vérification Valgrind/Helgrind, profileur de compteur de performance OProfile, traceur LTTng. **(2 points)**

c) Un programme parallèle sur 64 processeurs doit trouver le chemin le plus court pour un voyageur de commerce qui doit visiter un grand nombre de villes. Chaque fil d'exécution, après avoir évalué la longueur d'un chemin, vérifie si ce chemin est plus court que le meilleur chemin trouvé jusqu'à présent. Pour ce faire, le programme peut utiliser des mutex de lecture-écriture ou utiliser la technique RCU afin de lire et éventuellement remplacer la valeur du chemin le plus court jusqu'à présent. Lequel serait le plus performant? Pourquoi? Proposez une troisième solution possiblement plus intéressante? **(1 point)**

Le professeur: Michel Dagenais