POLYTECHNIQUE MONTRÉAL

Département de génie informatique et génie logiciel

Cours INF8601: Systèmes informatiques parallèles (Automne 2021)
3 crédits (3-1.5-4.5)

CONTRÔLE PÉRIODIQUE

DATE: Jeudi le 28 octobre 2021

HEURE: 13h45 à 15h35

DUREE: 1H50

NOTE: Aucune documentation permise sauf un aide-mémoire, préparé par l'étudiant, qui consiste en une feuille de format lettre manuscrite recto verso, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Un programme s'exécute sur un ordinateur avec une mémoire centrale de 8GiO et une mémoire cache de données de 64KiO. La taille des blocs en cache est de 128 octets. La section de programme suivante s'exécute sur cet ordinateur. Chaque entier occupe 8 octets sur cet ordinateur 64 bits. Dans ce programme, les entiers du vecteur `buffer` sont accédés en boucle. Quel sera le taux de succès en mémoire cache, pour les accès au vecteur `buffer`, si la constante `STRIDE` vaut 1? Si elle vaut 2? **(2 points)**

```
// i, sum et buffer[1000000] sont des entiers, le cache est vide
for(i = 0; i < 1000000; i += STRIDE) sum += buffer[i];
```

b) Une grappe de calcul parallèle contient 8 noeuds. Chaque noeud est constitué d'une carte mère, qui a une probabilité de disponibilité de 0.9, et d'un système de deux disques redondants en miroir. Un noeud est disponible si la carte mère et le système de disques sont disponibles. Le système de disques est disponible si au moins 1 des 2 disques est disponible. Chaque disque a une probabilité de disponibilité de 0.75. Quelle est la probabilité que les 8 noeuds soient disponibles simultanément? **(2 points)**

c) Un programme s'exécute sur un coeur et prend 100s. La fraction parallélisable de ce programme est de 0.95. Que deviendra le temps d'exécution de ce programme s'il s'exécute en parallèle sur 32 coeurs? Sur 64 coeurs? **(1 point)**

Question 2 (5 points)

a) Un ordinateur 64 bits à mémoire virtuelle utilise une table de pages à 3 niveaux et des pages de 64KiO. Chaque noeud dans la table de pages occupe une page. Les 9 bits les plus significatifs de l'espace adressable sont inutilisés. Quelle est la taille d'une entrée dans la table de page? Combien d'entrées sont contenues dans un noeud de la table de pages? Comment se décompose l'adresse de 64 bits en bits inutilisés, index de niveau 0, index de niveau 1, index de niveau 2 et décalage dans la page? **(2 points)**

b) Le programme TBB suivant s'exécute. Quelle sera la sortie affichée sur `cout`? Expliquez! **(2 points)**

```
struct Comp {
  float value;
  Comp() : value(0) {}
  Comp(Comp& c, split) { value = 0; }
  void operator()(const blocked_range<float*>& r) {
    float temp = value;
    for(float* a=r.begin(); a!=r.end(); ++a) temp += *a;
    value = temp;
  }
  void join(Comp& rhs) {value += rhs.value;}
};

int main(int argc, char *argv[]) {
```

```
  Comp comp; float array[1000];
  for(int i = 0; i < 1000; i++) array[i] = i % 10;
  parallel_reduce(blocked_range<float*>(array, array+1000),comp);
  cout << "Value: " << comp.value << "\n";
}
```

c) Que permet de spécifier la fonction `pthread_attr_setstacksize` de la librairie pthread? Qu'est-ce qui détermine la taille requise pour cette valeur? **(1 point)**

Question 3 (5 points)

a) Un programme multi-thread s'exécute sur 2 coeurs en parallèle. Afin d'aller plus vite, il n'utilise pas de verrou. Un thread veut mettre à jour des valeurs, `data1_a` et `data1_b` puis `data2_a` et `data2_b`, et indiquer lorsqu'elles sont valides en mettant à 1 `valid1` et `valid2`, respectivement. Le programme proposé suit. Doit-on ajouter des barrières mémoire pour avoir un comportement correct (i.e., ne pas lire les données avant qu'elles n'aient été mises à jour)? Si oui, modifiez le programme de chacun des deux coeurs en ajoutant les barrières mémoire minimales requises. **(2 points)**

```
(initialement toutes les variables sont à 0)
Coeur 0                          Coeur 1

data1_a = 16;                    if(valid1) {
data1_b = 14;                      new1 = data1_a + data1_b;
data2_a = 5;                     }
data2_b = 7;                     if(valid2) {
valid1 = 1;                        new2 = data2_a - data2_b;
valid2 = 1;                      }
```

b) Sur certains processeurs, les écritures à des adresses différentes vers la mémoire centrale peuvent être réordonnancées (Partial Store Order). Expliquez quels mécanismes matériels sur un processeur peuvent expliquer que de tels réordonnancements sont possibles? **(2 points)**

c) Quelle est la différence entre les protocoles de cohérence de cache MESI et MOESI? En quoi consiste l'état qui diffère entre les deux? **(1 point)**

Question 4 (5 points)

a) Le programme OpenMP suivant est exécuté. Donnez les lignes qui seront générées en sortie. Pour chaque ligne, dites si le nombre affiché peut changer d'une exécution à l'autre. Laquelle des 3 boucles sera la plus rapide? La plus lente? Pourquoi **(2 points)**

```
int main(int argc, char **argv)
{ int i, sum = 0;
  omp_set_num_threads(4);
```

```
  #pragma omp parallel for
  for(i = 0; i < 1000000; i++) sum += 1;
  printf("sum = %d\n", sum);
  sum = 0;
  #pragma omp parallel for
  for(i = 0; i < 1000000; i++) {
    #pragma omp atomic update
    sum += 1;
  }
  printf("atomic sum = %d\n", sum);
  sum = 0;
  #pragma omp parallel for
  for(i = 0; i < 1000000; i++) {
    #pragma omp critical
    sum += 1;
  }
  printf("critical sum = %d\n", sum);
}
```

b) Le programme OpenMP suivant est exécuté. Donnez les lignes qui seront générées en sortie. Expliquez si l'ordre entre les lignes, ou leur contenu, peuvent changer d'une exécution à l'autre et comment. **(2 points)**

```
int main(int argc, char **argv)
{ int t = -1, nbt = -1;
  printf("a)\n");
  omp_set_num_threads(4);
  #pragma omp parallel private(t, nbt)
  { nbt = omp_get_num_threads();
    t = omp_get_thread_num();
    printf("b) thread %d / %d\n", t, nbt);
    #pragma omp for schedule(static,1)
    for(int i = 0; i < 7; i++) {
      printf("c) thread %d / %d, i = %d\n", t, nbt, i);
    }
    #pragma omp single
    printf("d) thread %d / %d\n", t, nbt);
    #pragma omp masked
    printf("e) thread %d / %d\n", t, nbt);
  }
  printf("f) thread %d / %d\n", t, nbt);
}
```

c) Un processeur contient 32 coeurs. Vous désirez exécuter un programme parallèle multi-thread sur ce processeur. Le programme effectue surtout des calculs et fait un peu d'entrées-sorties. Combien de threads suggérez-vous d'utiliser? Pourquoi? **(1 point)**

Le professeur: Michel Dagenais