POLYTECHNIQUE MONTRÉAL

Département de génie informatique et génie logiciel

Cours INF8601: Systèmes informatiques parallèles (Automne 2023)
3 crédits (3-1.5-4.5)

CONTRÔLE PÉRIODIQUE

DATE: Mercredi le 25 octobre 2023

HEURE: 9h30 à 11h20

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 processeur 64 bits qui possède un cache L1 pour les données de 64Kio, avec chaque bloc en cache qui fait 64 octets. Ce cache a une fonction de correspondance directe et une stratégie de remplacement de *bloc le moins utilisé récemment* (LRU). Tel que montré dans le code qui suit, une boucle de calcul accède à répétition, 3 fois, un vecteur de *n* entiers de 64 bits pour faire différents calculs dans des registres. i) Jusqu'à quelle valeur de *n* est-ce que le vecteur peut entrer entièrement dans le cache de données L1? ii) Lorsque le vecteur entre entièrement en cache, quel sera le taux de succès en cache de données? iii) Lorsque la taille du vecteur dépasse nettement la taille du cache, quel sera le taux de succès en cache pour l'accès aux données? **(2 points)**

```
for(i = 0; i < 3; i++)
  for(j = 0; j < n; j++) sum = sum ^ v[j];
```

b) On veut configurer une grappe de calcul pour effectuer un travail. Deux configurations possibles sont offertes: a) une grappe de 32 ordinateurs utilisés pendant 1 heure, b) une grappe de 16 ordinateurs utilisés pendant 2 heures. Chaque ordinateur a une probabilité de 0.95 d'être fonctionnel pour toute la durée d'une heure de calcul. Un serveur de fichiers est utilisé tout au long des calculs. Son unité centrale de traitement a aussi une probabilité de 0.95 d'être fonctionnelle pour toute la durée d'une heure, alors que son unité de disque, en RAID 5, requiert 4 disques fonctionnels sur 5, chaque disque ayant une probabilité de 0.8 d'être fonctionnel pendant toute la durée d'une heure. Dans les deux cas, tous les ordinateurs de la grappe ainsi que le serveur de fichiers (unité centrale de traitement et unité de disque) doivent être fonctionnels pendant toute la durée du calcul. i) Quelle est la probabilité que la grappe de calcul a, excluant le serveur de fichiers, soit fonctionnelle pendant 1 heure? ii) La grappe de calcul b pendant 2 heures? iii) Quelle est la probabilité que le serveur de fichiers soit fonctionnel pendant 1 heure? iv) Quelle est finalement la configuration qui a le plus de chances de fonctionner pendant toute sa période afin de terminer son calcul? **(2 points)**

c) Vous trouvez un vieil ordinateur 32 bits dans le recyclage qui est encore fonctionnel. Vous voulez déterminer la taille des blocs en cache pour les données à l'aide d'un petit programme qui accède à répétition les éléments d'un vecteur dans une boucle. Comment pourriez-vous le faire? Expliquez? **(1 point)**

Question 2 (5 points)

a) Considérez le code suivant utilisant les *pthreads* qui s'exécute. Quelle en est la sortie? **(2 points)**

```
void *create_child_threads(void *arg) {
  int id = *((int *)arg);
  pthread_t t;
  fprintf(stderr, "T%d ", id);
  if(id < 100) {
    id = id * 10;
    for(int i = 1; i <= 2; i++) {
      id++;
      pthread_create(&t, NULL, create_child_threads, &id);
```

```
      pthread_join(t, NULL);
    }
  }
  pthread_exit(NULL);
}
int main(int argc, char **argv) {
  int id = 0;
  create_child_threads(&id);
  return(0);
}
```

b) La fonction suivante `DoParallelScan` est appelée avec comme arguments `y` un vecteur de 10 `float` tous à 0, `x` un vecteur de 10 `float` {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0}, et n un entier qui vaut 10. i) Quel est le contenu de `y` après la première passe du `parallel_scan`? ii) Après la deuxième passe? iii) Quelle est la valeur retournée par la fonction à la fin? **(2 points)**

```
class Body {
  T sum; T* const y; const T* const x;
  Body( T y_[], const T x_[] ) : sum(0), x(x_), y(y_) {}
  T get_sum() const {return sum;}
  template<typename Tag>void operator()(const blocked_range<int>& r, Tag) {
    T temp = sum;
    for( int i=r.begin(); i<r.end(); ++i ) {
      temp = temp + x[i] * x[i];
      if(Tag::is_final_scan()) y[i] = temp;
    }
    sum = temp;
  }
  Body(Body& b, split) : x(b.x), y(b.y), sum(0) {}
  void reverse_join(Body& a) { sum = a.sum + sum; }
  void assign( Body& b ) {sum = b.sum; }
};
float DoParallelScan(T y[], const T x[], int n) {
  Body body(y,x);
  parallel_scan(blocked_range<int>(0,n), body);
  return body.get_sum();
}
```

c) Vous réalisez un programme avec la librairie `pthreads`. Au moment de créer chaque nouveau fil d'exécution, il faut spécifier la taille à utiliser pour sa pile d'exécution. Comment peut-on estimer l'espace à réserver pour la pile d'un fil d'exécution? **(1 point)**

Question 3 (5 points)

a) Un ordinateur adressable à l'octet, semblable au x86-64, utilise des adresses de 64 bits et des pages de 4Kio. Les 16 bits les plus significatifs de l'adresse virtuelle sont inutilisés, et des tables de pages à

4 niveaux sont utilisées. Chaque noeud dans l'arbre qui constitue la table de pages occupe une page. Un processus commence avec 2Mio, utilisés pour son code exécutable et le monceau au tout début de son espace adressable, et 2Mio, utilisés à la toute fin de son espace adressable pour la pile. i) Quelle est la taille d'une entrée dans un noeud de la table de page? ii) Combien d'entrées a-t-on dans chaque noeud de la table de pages? iii) Comment se décomposent en différents champs les 64 bits d'une adresse virtuelle afin d'accéder l'arbre qui constitue la table de pages? iv) Combien de noeuds existent présentement dans la table de pages pour ce processus qui utilise 2Mio au début et 2Mio à la fin de son espace adressable? **(2 points)**

b) Un fil d'exécution sur le coeur 0 place des valeurs mises à jour dans les variables `data_1_a` et `data_1_b`, et change ensuite `data_ready_1` à 1 pour indiquer que ces variables sont prêtes. Il fait la même chose avec les variables `data_2_a` et `data_2_b`, et change ensuite `data_ready_2` à 1. Initialement, toutes les variables sont à 0. Avec la librairie standard du langage C, des barrières mémoire doivent être ajoutées pour s'assurer que ces valeurs ne sont utilisées sur le coeur 1 que si elles ont été mises à jour. A cette fin, une barrière mémoire a été ajoutée sur le fil d'exécution du coeur 0. Doit-on aussi ajouter des barrières mémoire sur le fil d'exécution du coeur 1? i) A quel endroit? ii) Quelles sont les valeurs finales possibles pour `new1` et `new2` si les bonnes barrières mémoire ont été ajoutées? **(2 points)**

```
Coeur 0                          Coeur 1

data_1_a = 25;          if(data_1_ready) {
data_1_b = 7;             new1 = data_1_a + data_1_b;
data_2_a = 22;          }
data_2_b = 11;
cmm_smp_wmb();          if(data_2_ready) {
data_1_ready = 1;         new2 = data_2_a + data_2_b;
data_2_ready = 1;       }
```

c) La plupart des circuits de mémoire cache utilisent les protocoles MESI ou MOESI pour en assurer la cohérence. Quel état les différencie? Donnez un exemple concret de comportement différent possible, selon qu'une cache utilise l'un ou l'autre protocole. **(1 point)**

Question 4 (5 points)

a) Le programme OpenMP suivant est exécuté. Donnez une version possible des 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) thread %d / %d\n", t, nbt);
  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,2)
    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);
}
```

b) Dans le programme OpenMP suivant, les éléments des vecteurs b et c sont initialisés à 0.0, et la matrice a est initialisée avec des données à traiter. Le programme sera exécuté sur un ordinateur 64 bits avec les blocs de cache L1 d'une taille de 128 octets. La valeur de nb est un paramètre de configuration que l'usager doit choisir. On veut optimiser la performance de ce programme tout en s'assurant d'un comportement correct (toujours le même comportement attendu, pas de course non déterministe). i) Quelle valeur de nb recommandez-vous et pourquoi? ii) Est-ce que le programme comporte une course, et si oui laquelle? iii) Serait-il avantageux de changer l'ordre des boucles, j sur NCA d'abord et i sur NRA imbriquée ensuite? iv) Suggérez une optimisation possible. **(2 points)**

```
double a[NRA][NCA], b[NRA], c[NCA];
int nb;

int main(int argc, char **argv)
{
  int i, j;
  omp_set_num_threads(8);
  #pragma omp parallel for schedule(static,nb) shared(a,b,c) private(i,j)
  for (i = 0; i < NRA; i++) {
    for (j = 0; j < NCA; j++) {
      b[i] += a[i][j];
      c[j] += a[i][j];
} } }
```

c) Le calcul matriciel suivant est effectué en parallèle à l'aide de OpenMP. Est-ce que i, j, k est l'ordre d'imbrication le plus efficace pour les trois boucles? Sinon, quel est-il? Expliquez. **(1 point)**

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

Le professeur: Michel Dagenais