POLYTECHNIQUE MONTRÉAL

Département de génie informatique et génie logiciel

Cours INF8601: Systèmes informatiques parallèles (Automne 2024)
3 crédits (3-1.5-4.5)

CONTRÔLE PÉRIODIQUE

DATE: Mercredi le 23 octobre 2024

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 ordinateur multi-coeur 32 bits, dont la mémoire est adressable à l'octet, possède un cache de données de 16Kio, avec des ensembles de 2 blocs, pour chacun de ses 8 coeurs. Chaque bloc en cache contient 64 octets. Ce cache est initialement vide. i) Combien de blocs et d'ensembles contient le cache de chaque coeur de processeur? ii) Comment se décomposent les 32 bits d'adresse en: décalage dans le bloc, numéro d'ensemble et étiquette? Les adresses suivantes sont accédées en séquence, en lecture (R) ou en écriture (W), sur les processeurs spécifiés (P0 à P7). iii) Donnez l'état (M, E, S ou I) des blocs non vides du cache après chaque accès. **(2 points)**

```
P0 W 0x32B 7;P3 W 0x32B 5;P7 W 0x32B 9;P1 R 0x32B;P2 R 0x43B;
```

b) Trois serveurs de données redondants, chacun contenant une unité de disque RAID de 5 disques, sont utilisés pour alimenter une grappe de calcul. Chaque serveur a une probabilité de panne de 0.1 pour son électronique, de 0.15 pour l'électronique de l'unité de disque RAID et de 0.2 pour chaque disque dans les unités de disque RAID. Un serveur fonctionne en autant que son électronique, l'électronique de l'unité de disque RAID et au moins 3 disques sur les 5 sont opérationnels. i) Quelle est la probabilité globale de panne pour 1 serveur? ii) Pour les 3 serveurs simultanément? **(2 points)**

c) Un programme qui s'exécute sur un seul fil d'exécution (thread) a une fraction parallélisable de 0.8. Quel est le facteur d'accélération qu'il sera possible d'obtenir si on l'exécute sur un processeur avec de nombreux coeurs (e.g. 64 coeurs) avec 8 fils d'exécution? Avec 16 fils d'exécution? **(1 point)**

Question 2 (5 points)

a) Le programme suivant crée un certain nombre de fils d'exécution POSIX. A chaque fil créé, un message est imprimé avec un identificateur qui indique la position de ce fil dans l'arborescence de création de fils d'exécution. i) Quel est le nombre de fils d'exécution créés à l'aide de pthread_create par ce programme? ii) Donnez une sortie possible de ce programme? **(2 points)**

```
void *create_child_threads(void *arg) {
  int id = (uintptr_t)arg, level = 0;
  pthread_t t;
  fprintf(stderr, "T%d ", id);
  for(int tmp = id; tmp > 0; level++) tmp = tmp / 10;
  if(level <= 2) {
    int nb_child = 3 - level;
    for(int i = 0; i < nb_child; i++)
      pthread_create(&t,NULL,create_child_threads,(void *)((uintptr_t)(id*10+i+1)));
  }
  pthread_exit(NULL);
}

int main(int argc, char **argv) {
  create_child_threads((void *)((uintptr_t)0));
  return(0);
}
```

b) Un ordinateur 64 bits, semblable au x86-64, dont la mémoire est adressable à l'octet, utilise des tables de pages à 4 niveaux (en fait un arbre de noeuds, chacun occupant une page, à 4 niveaux) et des pages de 4KiO. Les 16 bits les plus significatifs de l'adresse sont inutilisés. Un programme ajoute à son espace adressable une zone de 20MiO calquée sur un fichier. La table de page doit donc être augmentée pour couvrir ces nouveaux 20MiO. i) Comment se décompose l'adresse en ses différents champs (décalage dans la page, index à chaque niveau dans la table de pages...). ii) Combien de noeuds devront être ajoutés à l'arbre pour couvrir ces 20MiO au minimum? Au maximum? **(2 points)**

c) Pour un ordinateur qui utilise des adresses de 64 bits, comme l'architecture x86-64, chaque entrée dans la table de page a une longueur de 64 bits. Cependant, pour chaque entrée, on veut mémoriser l'adresse physique de la page (adresse de 64 bits) ainsi que plusieurs bits de statut comme les permissions (valide, lecture, écriture, exécution) pour cette page. Comment peut-on placer toute cette information dans une entrée de 64 bits? **(1 point)**

Question 3 (5 points)

a) La fonction suivante `CheckVector` est appelée avec comme argument `v` un vecteur de 30 `float` {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 0.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 0.0, 21.0, 22.0, 33.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 0.0}, et `size` un entier qui vaut 30. Expliquez brièvement ce que fait le programme. Quelle est la valeur retournée par la fonction à la fin? **(2 points)**

```
struct Count {
  int value; float *v;
  Count() : value(0) {}
  Count( Count& c, split ) { v = c.v; value = 0; }

  void operator()( const blocked_range<int>& r ) {
    int temp_count = value;
    for(int i = r.begin(); i != r.end(); i++ )
      if(v[i] < v[i - 1]) temp_count++;
    value = temp_count;
  }
  void join( Count& rhs ) {value += rhs.value;}
};

int CheckVector(float v[], int size) {
  Count c; c.v = v;
  parallel_reduce( blocked_range<int>(1, size), c);
  return c.value;
}
```

b) La section de code suivante s'exécute sur un ordinateur avec ordonnancement faible. Les variables f1, f2, f3, valid1 et valid2 sont initialement à 0. Quelles barrières devrait-on insérer, et où, pour avoir comme seules sorties possibles: `0 0 0` , `22 34` ou `22 34 42` ? **(2 points)**

```
Processeur 0                    Processeur 1

f1 = 22;                          if(valid1) {
f2 = 34;                            if(valid2) {
valid1 = 1;                           printf("%d %d %d\n", f1, f2, f3);
f3 = 42;                            } else {
valid2 = 1;                           printf("%d %d\n", f1, f2);
                                    } else {
                                      printf("0 0 0\n");
                                    }
```

c) Pour implémenter un verrou, il est possible de faire une boucle sur un échange atomique. On échange le contenu du verrou avec la valeur 1. Si l'échange donne 1, le verrou était pris et on boucle jusqu'à ce que la valeur 0 soit obtenue, ce qui indique que le verrou était libre. Une implémentation plus efficace est d'abord de boucler sur une lecture du verrou pour attendre que le verrou soit libre, et ensuite essayer un échange atomique. Le code suivant illustre cette différence. Pourquoi est-ce plus efficace? **(1 point)**

```
        DADDUI R2, R0, #1
lockit: EXCH R2, 0(R1)
        BNEZ R2, lockit
```

```
lockit: LD R2, 0(R1)
        BNEZ R2, lockit
        DADDUI R2, R0, #1
        EXCH R2, 0(R1)
        BNEZ R2, lockit
```

Question 4 (5 points)

a) Le programme OpenMP suivant est exécuté. Donnez les lignes qui seront générées en sortie. **(2 points)**

```
int main(int argc, char **argv)
{ int nb_thread, thread, p, s;
  omp_set_num_threads(3);
  #pragma omp parallel private(thread, nb_thread)
  { printf("a)\n");
    for(int i = 0; i < 3; i++) {
      nb_thread = omp_get_num_threads();
      thread = omp_get_thread_num();
      printf("b) thread %d / %d, i = %d\n", thread, nb_thread, i);
    }
    printf("c) thread %d / %d\n", thread, nb_thread);
    #pragma omp for schedule(static,1)
    for(int i = 0; i < 4; i++) {
      nb_thread = omp_get_num_threads();
      thread = omp_get_thread_num();
      printf("d) thread %d / %d, i = %d\n", thread, nb_thread, i);
    }
  }
}
```

b) On vous fournit la section de code suivante en OpenMP. Les matrices `double` a[NRA][NCA] et b[NCA][NCB] ont été initialisées aux valeurs du problème et la matrice c[NRA][NCB] et le vecteur d[ND] `double` ont été initialisés à 0. i) Vérifiez que le code est correct (déterministe en présence de fils d'exécution parallèles), et suggérez au besoin un correctif. ii) Proposez au moins deux améliorations, par rapport au programme avec votre correctif inclus, qui permettront d'obtenir le même résultat mais plus efficacement. Expliquez brièvement chaque amélioration apportée. **(2 points)**

```
int main(int argc, char **argv) {
  #pragma omp parallel for shared(a,b,c,d) private(i,j,k)
  for (int i=0; i<NRA; i++)
    for(int j=0; j<NCB; j++)
      for(int k=0; k<NCA; k++) {
        c[i][j] += a[i][k] * b[k][j];
        d[(int)(c[i][j]) % ND]++;
      }
}
```

c) La commande `#pragma omp barrier` effectue un rendez-vous entre tous les fils d'exécution de la région parallèle. Comment peut-on implémenter efficacement un rendez-vous en mémoire partagée? **(1 point)**

Le professeur: Michel Dagenais