ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF8601: Systèmes informatiques parallèles (Automne 2018)
3 crédits (3-1.5-4.5)

CONTRÔLE PÉRIODIQUE

DATE: Vendredi le 26 octobre 2018

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) Lors de votre premier travail pratique, vous avez analysé le comportement d'applications parallèles PThread et TBB. D'après les données obtenues, donnez une courte explication des différences entre PThread et TBB par rapport à leur fonctionnement propre. Combien de fils d'exécution sont démarrés? Comment se fait la répartition du calcul entre les fils d'exécution? Quels sont les appels systèmes impliqués dans la synchronisation des fils d'exécution? **(2 points)**

b) Vous disposez d'une grappe de calcul avec 64 noeuds. Chaque noeud est constitué d'une carte électronique et de deux disques en miroir (en redondance, il suffit que l'un des deux fonctionne). La probabilité de cesser de fonctionner pendant une période de 15 minutes est de .001 pour les cartes et de .01 pour les disques. Deux possibilités s'offrent. On peut avoir des tâches de 15 minutes ou d'une heure (tous les noeuds doivent être parfaitement fonctionnels pendant toute la période choisie, sinon le résultat est perdu). Etant donné le surcoût de mettre en place la tâche et de sauver les résultats à la fin, une tâche de 15 minutes fait 5 fois moins de travail qu'une tâche d'une heure. Quelle option permettra d'effectuer globalement plus de travail en moyenne? **(2 points)**

c) Vous analysez le temps passé dans les différentes fonctions d'un programme de calcul et obtenez les temps suivants: A 100s, B 80s, C 60s, D 120s, E 40s. Le temps passé dans les autres fonctions du programme est négligeable. Vous êtes en mesure de paralléliser les fonctions A, B et D. Quel sera le facteur d'accélération obtenu avec 8 threads roulant sur autant de coeurs? **(1 point)**

Question 2 (5 points)

a) Un ordinateur 32 bits, dont la mémoire est adressable à l'octet, possède 2 coeurs (P0 et P1). Chaque coeur a une mémoire cache de données de 4096 octets avec des blocs de 256 octets et une fonction de correspondance directe. La mémoire cache est initialement vide. Les accès suivants (R pour lecture ou W pour écriture) ont lieu sur les coeurs spécifiés aux adresses données. Comment se décomposent les 32 bits d'adresse en: décalage dans le bloc, numéro de bloc, et étiquette? Quel est le taux de succès en cache pour chacun des 2 coeurs? Quel est l'état (M, O, E, S ou I) de chacun des blocs présents dans les caches à la fin de ces accès? **(2 points)**

```
P0 R 0x111; P1 R x111; P0 R 0x1F0; P0 R 0x222;
P1 W 0x211; P1 R 0x2200; P0 R 0x0F0; P1 W 0x000;
```

b) Un ordinateur 64 bits, 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 12MiO calquée sur un fichier. La table de page doit donc être augmentée pour couvrir ces nouveaux 12MiO. Combien de noeuds devront être ajoutés à l'arbre pour couvrir ces 12MiO au minimum? Au maximum? **(2 points)**

c) 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`, `12 24` ou `12 24 42`?

```
Processeur 0                    Processeur 1
```

```
f1=12;
f2=24;
valid1=1;
f3=42;
valid2=2;
```

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

(1 point)

Question 3 (5 points)

a) Le programme suivant utilise la librairie TBB et est exécuté avec la commande `./qtbb 2014`. Donnez la sortie pour la ou les lignes qui commencent par a). Expliquez le patron des lignes en sortie qui commencent par b). **(2 points)**

```
// programme qtbb.c
#include "tbb/tbb.h"
using namespace tbb;

struct Process {
  float value;
  Process() : value(0) {}
  Process(Process& p, split) { value = 0; }
  void operator()(const blocked_range<float*>& v) {
    float m = 0, *pf;
    for(pf=v.begin(); pf!=v.end(); pf++) m = *pf > m ? *pf : m;
    value = m > value ? m : value;
    printf("b) %f\n", m);
  }
  void join(Process& r) {value =  r.value > value ? r.value : value;}
};

int main(int argc, const char* argv[]) {
  Process p;
  int i, n = atoi(argv[1]);
  float m, *v = new float[n];
  for(i = 0; i < n; i++) v[i] = (float)(i % 256 + i / 256);
  parallel_reduce( blocked_range<float*>(v, v+n), p);
  printf("a) %f\n", p.value);
}
```

b) Le programme suivant est utilisé pour estimer la valeur de certains paramètres d'un ordinateur. Deux exécutions sont faites (commande 1 et commande 2) en faisant varier à chaque fois un des paramètres.

La commande et les temps d'exécution (temps CPU en mode usager) sont donnés en commentaire. A la lueur de ces résultats, que pouvez-vous conclure sur la taille des blocs en cache? Sur la taille de la mémoire cache (de quel niveau L1, L2 ou L3)? Sur la taille de la mémoire centrale? **(2 points)**

```
# commande 1 et résultat
$ for i in 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26;
    do /usr/bin/time --format="%U" ./cache $i 0 32; done
1.97 1.96 1.96 1.95 1.97 1.96 1.97 1.97 2.06 2.13 2.15 2.15 2.18 2.19 2.19

# commande 2 et résultat
$ for i in 0 1 2 3 4 5 6 7 8 9 10;
    do /usr/bin/time --format="%U" ./cache 24 $i 32; done
1.92 2.48 4.15 8.24 17.61 25.36 32.61 35.63 38.31 41.90 38.33

# cache.c:
int main(int argc, char **argv)
{ uint64_t i, j, inc, sum = 0, one = 1;
  static uint32_t *data;
  uint64_t size = one<<atoi(argv[1]),
    stride = one<<atoi(argv[2]), iter = one<<atoi(argv[3]);

  data = malloc(size * sizeof(uint32_t));
  for(i = 0; i < size; i++) data[i] = i ^ 16777215;

  for(i = 0 ; i < iter; i += size / stride)
    for(j = 0; j < size; j += stride) sum += data[j];
  return(sum);
}
```

c) Vous possédez un ordinateur à 4 coeurs. Vous programmez une application de calcul parallèle avec la librairie pthread. Combien de thread devriez-vous utiliser pour paralléliser le calcul?

(1 point)

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(2);
  #pragma omp parallel private(thread)
  { printf("a)\n");
    for(int i = 0; i < 4; 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,2)
    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) Le programe suivant s'exécute sur un ordinateur avec 2 coeurs physiques hyperthread qui donnent ainsi 4 coeurs logiques. La variable s est un vecteur d'entiers 32 bits de 64 entrées. La commande indiquée en commentaire a été utilisée pour rouler ce programme avec un nombre croissant de threads de 1 à 32. Le temps d'exécution de ce programme (temps réel écoulé) imprimé en sortie est fourni à la suite de la commande en commentaire. Comment expliquez-vous cette évolution du temps d'exécution en fonction du nombre de threads, et en particulier que 2 threads parallèles prennent plus de temps que 1 thread? Est-ce que ce programme comporte un bogue de performance? Comment pourrait-on le corriger? **(2 points)**

```
// $ for i in 1 2 4 8 16 32; do /usr/bin/time --format="%E" ./qomp2018-b $i; do
// 10.44 13.66 25.81 27.71 22.38 8.13



int main(int argc, char **argv)
{ uint32_t s[64], pos;
  omp_set_num_threads(atoi(argv[1]));
  #pragma omp parallel private(pos) shared(s)
  { pos = omp_get_thread_num();
    #pragma omp for
    for(int i = 0; i < 2000000000; i++) {
      s[pos] += (i * pos) % (pos + 1);
    }
  }
}
```

c) Une variable est accédée en écriture par plusieurs threads en parallèle et doit être protégée. Deux versions de code sont proposées afin d'éviter la corruption de cette variable. Est-ce que les deux versions sont correctes? Laquelle sera la plus efficace? **(1 point)**

```
// Version 1
#pragma omp critical
global_sum += local_sum;
// Version 2
#pragma omp atomic update
global_sum += local_sum;
```

Le professeur: Michel Dagenais