ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF8601: Systèmes informatiques parallèles (Automne 2017)
3 crédits (3-1.5-4.5)

CONTRÔLE PÉRIODIQUE

DATE: Vendredi le 27 octobre 2017

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) 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 0.05 pour l'électronique de l'unité de disque RAID et de 0.15 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 4 disques sur les 5 sont opérationnels. Quelle est la probabilité globale de panne pour 1 serveur? Pour les 3 serveurs simultanément? **(2 points)**

b) Lors du premier travail pratique, vous avez calculé le ratio $(user + sys)/elapsed$ à partir des chiffres dans le répertoire *results* pour l'exécution de *dragonizer*. Que représente chacune de ces trois composantes? Que signifie une valeur plus faible pour ce ratio? **(1 point)**

c) Si vous aviez à paralléliser le calcul pour une fractale dont le temps de calcul par unité de surface est très variable (contrairement à la fractale du dragon, dont le temps de calcul est proportionnel à la taille des intervalles), quelle librairie utiliseriez-vous entre *pthreads* et *TBB*? Pourquoi? **(1 point)**

d) Tous les thread d'exécution d'un même processus partagent le même espace mémoire. Dans ce contexte, comment le système peut-il fournir à chaque thread sa propre pile et ses propres variables spécifiques (*Thread Local Storage*)? **(1 point)**

Question 2 (5 points)

a) Un ordinateur multi-coeur 32 bits, dont la mémoire est adressable à l'octet, possède une cache de données de 32Kio, avec des ensembles de 4 blocs, pour chacun de ses 8 coeurs. Chaque bloc en cache contient 64 octets. Cette cache est initialement vide. 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). Donnez l'état des cases non vides (M, E, S ou I) après chaque accès. **(2 points)**

```
P0 W 0x22B 7;P3 W 0x22B 5;P7 W 0x22B 9;P1 R 0x22B;P2 R 0x33B;
```

b) Un ordinateur multi-coeur 64 bits, dont la mémoire est adressable à l'octet, utilise des tables de pages à 3 niveaux et des pages de 16Kio. Chaque noeud de la table de pages entre dans une page. Montrez comment l'adresse se décompose en décalage dans la page, index pour chacun des 3 niveaux dans la table, et s'il y a lieu les bits inutilisés. Décomposez l'adresse virtuelle 0x0123456789ABCDEF en ces différentes composantes (décalage, index0, index1...). Décrivez l'information typique contenue dans chaque entrée de 64 bits de la table de page. **(2 points)**

c) Plusieurs périphériques permettent un accès direct à la mémoire (DMA). Le système peut ainsi envoyer une commande à l'unité de disque pour lire certains secteurs et copier leur contenu à une certaine adresse physique en mémoire centrale qui correspond à un tampon d'entrée-sortie. Lorsque l'opération est terminée, l'unité de disque le signale au système d'exploitation par le biais d'une interruption. Que doit alors faire le système d'exploitation pour assurer la cohérence avant de laisser l'unité centrale de traitement accéder au contenu du tampon qui vient d'être rempli par l'unité de disque via un accès direct? **(1 point)**

Question 3 (5 points)

a) Une image en couleur est représentée par un vecteur de *taille* pixels. Chaque pixel est une structure de donnée contenant 3 octets, les champs *rouge*, *vert* et *bleu* chacun représentant l'intensité pour la couleur primaire correspondante. On veut compter pour une image les pixels qui sont plutôt rouges, verts ou bleus. Ainsi, si l'intensité rouge d'un pixel est plus grande que le vert ou le bleu, le pixel est compté comme rouge. En cas d'égalité de puissance pour 2 ou 3 couleurs pour un pixel, votre programme peut choisir comme il le veut entre ces 2 ou 3 couleurs. Ecrivez un programme, utilisant la librairie TBB, qui offre la fonction *void CompteCouleur(struct Pixel image[], int taille, int *rouge, int *vert, int *bleu)* permettant d'effectuer le calcul du nombre de pixels de chaque couleur. L'argument image contient un vecteur de pixels alors que l'argument taille contient la taille de ce vecteur. La fonction CompteCouleur doit calculer et retourner dans les arguments correspondants le nombre de pixels plutôt rouges, verts ou bleus. **(3 points)**

b) Un programme OpenMP est utilisé par une compagnie qui produit du matériel promotionnel et change très souvent de logo. Ainsi, un des thread ajoute régulièrement un nouveau logo alors que tous les thread doivent utiliser le plus récent logo pour générer du matériel. Il en résulte un accès concurrent à la variable *nb_version* et au vecteur *logo*[] entre le thread 0 et les autres thread. Parmi les possibilités pour synchroniser correctement ces accès concurrents, vous avez le choix entre des sections critiques, des opérations atomiques ou un accès sans synchronisation mais avec des barrières mémoire. Quel est le nom de la directive OpenMP à utiliser pour chacune de ces 3 possibilités? Parmi ces 3 possibilités, lesquelles sont applicables ici et laquelle devrait permettre la meilleure parallélisation? Expliquez. **(2 points)**

```
/* Normalement, créer un nouveau logo */
int CreateLogo() { return 42; }

int main(int argc, char **argv)
{ int N = 5000, nb_version = 0;
  int logo[N+1];

  logo[nb_version] = CreateLogo();
  nb_version++;

  #pragma omp parallel for shared(nb_version,logo)
  for(int i = 0; i < N; i++) {
    if(omp_get_thread_num() == 0) {
      logo[nb_version] = CreateLogo();
      nb_version++;
    }
    int v = nb_version - 1;
    int current_logo = logo[v];
    /* Faire quelque chose avec le logo */
  }
}
```

Question 4 (5 points)

a) On vous fournit la section de code suivante en OpenMP. Dans un premier temps, vérifiez que le code est correct (déterministe en présence de thread parallèles), et suggérez au besoin un correctif. Ensuite, vous devez proposer plusieurs améliorations qui permettront d'obtenir le même résultat mais plus efficacement. Fournissez une nouvelle version de cette section de programme et expliquez en une ou deux lignes de texte chaque amélioration apportée. **(3 points)**

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

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

b) Le programme OpenMP suivant s'exécute et produit une sortie sur stdout. Donnez une sortie possible produite par l'exécution de ce programme. Est-ce que la sortie peut changer d'une exécution à l'autre? Les valeurs de *p* et *s* affichées sur la ligne c) sont-elles toujours égales? Expliquez. **(2 points)**

```
int main(int argc, char **argv)
{ int nb_thread, thread, p, s;
  printf("a)\n");
  omp_set_num_threads(3);
  #pragma omp parallel private(thread, p) shared(s)
  { printf("b)\n");
    #pragma omp for schedule(static,1) ordered
    for(int i = 0; i < 8; i++) {
      nb_thread = omp_get_num_threads();
```

```
        thread = omp_get_thread_num();
        p = s = i;
        printf("c) thread %d, p = %d, s = %d\n", thread, p, s);
      }
      #pragma omp single
      printf("d) thread %d / %d\n", thread, nb_thread);
      #pragma omp master
      printf("e) thread %d / %d\n", thread, nb_thread);
      #pragma omp critical
      printf("f) thread %d / %d\n", thread, nb_thread);
    }
    printf("g) thread %d / %d\n", thread, nb_thread);
  }
```

Le professeur: Michel Dagenais