POLYTECHNIQUE MONTRÉAL

Département de génie informatique et génie logiciel

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

CORRIGÉ DE L'EXAMEN FINAL

DATE: Vendredi le 22 décembre 2023

HEURE: 9h30 à 12h00

DUREE: 2H30

NOTE: Aucune documentation permise sauf un aide-memoire, 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) Le programme OpenCL suivant reçoit en argument une image en tons de gris (valeur entre 0 et 255 pour chaque point) et calcule en sortie le contenu du vecteur `histogram`. Le programme est appelé avec une taille globale de problème de 4000, et une image qui contient des lignes horizontales où chaque pixel a une valeur donnée par l'équation `image[i][j] = i % 256`. Tous les points de chaque rangée sont ainsi à la même valeur. Quel sera le contenu du vecteur `histogram` en sortie? **(2 points)**

```
__kernel void HistogramA(__global unsigned char image[4000][4000],
    __global unsigned int histogram[256])
{
  int i;
  int col = get_global_id(0);
  for(i = 0; i < 4000 ; i++) atomic_inc(histogram[image[i][col]]);
}
```

Chaque colonne contient la suite de 0, 1, 2, 3, ... jusqu'à 255, répétée 15 fois, puis la suite 0, 1, 2, 3, ... jusqu'à 159: {0, 1, 2, 3, ... 254, 255, (0, 1, 2, 3, ... 254, 255) x 14, 0, 1, 2, 3, ... 158, 159}. Le vecteur histogram contient donc 16 rangées à cette valeur x 4000 points identiques par rangée = 64000 pour les entrées de 0 à 159, et 15 x 4000 = 60000 pour les entrées de 160 à 255.

b) Deux amis vous proposent chacun une version différente du programme OpenCL HistogramA utilisé en a): `HistogramB` et `HistogramC`. i) Est-ce que les trois versions devraient donner le même résultat en sortie? Quelle devrait être la performance relative de chaque version (i.e. la plus rapide, la plus lente, entre les deux) et pourquoi? **(2 points)**

```
__kernel void HistogramB(__global unsigned char image[4000][4000],
    __global unsigned int histogram[256])
{
  int i;
  int row = get_global_id(0);
  for(i = 0; i < 4000 ; i++) atomic_inc(histogram[image[row][i]]);
}

__kernel void HistogramC(__global unsigned char image[4000][4000],
    __global unsigned int histogram[256])
{
  int i;
  int col = get_global_id(0), group_size = get_local_size(0);
  int chunk_size = 256 / group_size + 1;
  int start_chunk = get_local_id(0) * chunk_size;
  int end_chunk = min(256, start_chunk + chunk_size);
  unsigned int tmp_histogram[256];
  __local unsigned int local_histogram[256];

  for(i = 0; i < 256; i++) tmp_histogram[i] = 0;
  for(i = 0; i < 4000 ; i++) tmp_histogram[image[i][col]]++;
  for(i = start_chunk; i < end_chunk; i++) local_histogram[i] = 0;
```

```
  barrier(CLK_LOCAL_MEM_FENCE);
  for(i = 0; i < 256; i++)
    atomic_add(local_histogram[i], tmp_histogram[i]);
  barrier(CLK_LOCAL_MEM_FENCE);
  for(i = start_chunk; i < end_chunk; i++)
    atomic_add(histogram[i], local_histogram[i]);
}
```

Les trois versions font le même calcul. La version A est plus rapide que B car les accès de work item adjacents sont à des cases adjacentes, `image[i][col]` avec col qui change avec `get_global_id(0)` de 1 d'un work item au suivant. La version C est encore plus rapide que A, car elle fait beaucoup moins d'accès atomiques dans chaque work item, 256 accès atomiques locaux au work group et une fraction de 256 accès atomiques globaux, au lieu de 4000 accès atomiques en mémoire globale.

c) Lors des travaux pratiques, vous avez utilisé le langage de programmation OpenCL pour effectuer des calculs sur un GPU conçu par la compagnie AMD. Est-ce que le langage OpenCL peut aussi être utilisé pour effecturer des calculs sur les coeurs de CPU de la compagnie AMD? La plupart des utilisateurs de GPU conçus par la compagnie NVIDIA utilisent le langage de programmation CUDA. Pourraient-ils aussi utiliser le langage OpenCL sur ces GPU? **(1 point)**

Le langage OpenCL est un standard qui est supporté sur la plupart des plates-formes, incluant les GPU des compagnies AMD, NVIDIA, Intel et ARM. Un programme OpenCL peut aussi s'exécuter sur les coeurs de CPU de l'hôte.

Question 2 (5 points)

a) Le programme MPI suivant s'exécute sur 4 noeuds (MPI_Comm_size retourne 4). Donnez une sortie possible produite par ce programme? **(2 points)**

```
int main (int argc, char *argv[])
{
  int i, j, nb_row = 4, nb_col = 4, rank, size, m;
  int a[4][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {8, 7, 6, 5}, {4, 3, 2, 1}};
  int b[4][4] = {{0, 1, 0, 2}, {0, 3, 0, 4}, {0, 5, 0, 6}, {0, 7, 0, 8}};
  int c[4][4];
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  for(i = 0; i < nb_row; i++) {
    for(j = 0; j < nb_col; j++) {
      m = a[i][rank] * b[rank][j];
      MPI_Reduce(&m, &(c[i][j]), 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    }
  }

  if(rank == 0) {
```

```
    printf("C = {");
    for(i = 0; i < nb_row; i++) {
      printf("\n      ");
      for(j = 0; j < nb_col; j++) printf("%d ", c[i][j]);
    }
    printf("\n    }\n");
  }
  MPI_Finalize();
}
```

```
C = {
      0 50 0 60
      0 114 0 140
      0 94 0 120
      0 30 0 40
    }
```

b) Le programme MPI suivant s'exécute sur 4 noeuds (MPI_Comm_size retourne 4). Donnez une sortie possible produite par ce programme? **(2 points)**

```
int main (int argc, char *argv[])
{
  int size, rank, i, in[4], o1[4], o2[4];
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  for(i = 0; i < 4; i++) { in[i] = i * i + rank * i; o1[i] = o2[i] = 0; }
  MPI_Alltoall(in, 1, MPI_INT, o1, 1, MPI_INT, MPI_COMM_WORLD);
  MPI_Allgather(o1, 1, MPI_INT, o2, 1, MPI_INT, MPI_COMM_WORLD);
  printf("%d: in={%d, %d, %d, %d};\n", rank, in[0], in[1], in[2], in[3]);
  printf("%d: o1={%d, %d, %d, %d};\n", rank, o1[0], o1[1], o1[2], o1[3]);
  printf("%d: o2={%d, %d, %d, %d};\n", rank, o2[0], o2[1], o2[2], o2[3]);
  MPI_Finalize();
}
```

```
0: in={0, 1, 4, 9};
0: o1={0, 0, 0, 0};
0: o2={0, 1, 4, 9};
1: in={0, 2, 6, 12};
1: o1={1, 2, 3, 4};
1: o2={0, 1, 4, 9};
2: in={0, 3, 8, 15};
2: o1={4, 6, 8, 10};
2: o2={0, 1, 4, 9};
3: in={0, 4, 10, 18};
3: o1={9, 12, 15, 18};
3: o2={0, 1, 4, 9};
```

c) En plus des fonctions MPI_Send et MPI_Receive, on retrouve dans la librairie MPI les fonctions MPI_Send_init et MPI_Receive_init. En quoi ces fonctions diffèrent des premières, et dans quel cas est-ce plus avantageux de les utiliser? **(1 point)**

Ces fonctions permettent de préparer une requête pour un envoi ou une réception, qui sont alors exécutés avec la fonction MPI_Start sur cette requête. La même requête peut être répétée à plusieurs reprises, en appelant à nouveau MPI_Start sur cette même requête. Ceci est avantageux lorsqu'un envoi ou une réception sont effectués plusieurs fois avec les mêmes paramètres. Divers calculs internes associés à cette requête n'ont alors pas à être répétés à chaque appel à MPI_Send ou MPI_Receive. Ils ne sont faits qu'une seule fois, lors de l'appel à MPI_Send_Init ou MPI_Receive_Init.

Question 3 (5 points)

a) Un outil de mesure de performance, comme CPU Profile, échantillonne le contenu de la pile d'appels (la fonction courante et celles dans le chemin d'appel) à un intervalle de temps régulier de 1ms. Lorsqu'un échantillon se répète, ce qui arrive souvent, un facteur multiplicatif est ajouté, plutôt que de sauver le même chemin d'appel plusieurs fois dans le fichier de sortie. Voici les échantillons récoltés, chacun suivi du nombre de fois qu'il a été rencontré. Calculez le temps passé dans chaque fonction elle-même (self) et dans chaque fonction et ses appels (self+childs). **(2 points)**

```
(main, f1), 9
(main, f2), 7
(main), 6
(main, f1, f2, f4, f5, f6), 5
(main, f1, f3), 11
(main, f2, f4, f6), 8
(main, f1, f2, f3, f4), 3
(main, f1, f3, f5), 10
```

L'exécution complète a pris 59ms, puisque le nombre total d'échantillons est de 59 et qu'ils sont pris à chaque 1ms. Pour chaque fonction, le temps self est représenté par le nombre d'échantillons où la fonction se trouve sur le dessus de la pile. Le temps self+childs est donné par le nombre d'échantillons où la fonction se retrouve quelque part sur la pile.

fonction	self	self+child
main	6ms	9+7+6+5+11+8+3+10 = 59ms
f1	9ms	9+5+11+3+10 = 38ms
f2	7ms	7+5+8+3 = 23ms
f3	11ms	11+3+10 = 24ms
f4	3ms	5+8+3 = 16ms
f5	10ms	5+10 = 15ms
f6	5+8=13ms	5+8 = 13ms

b) Un outil de vérification des accès et de l'allocation de mémoire, semblable à Memcheck, instrumente toutes les lectures (read, adresse, nombre d'octets) et écritures (write, adresse, nombre d'octets) en mémoire, de même que les allocations (malloc, adresse, nombre d'octets) et libérations (free, adresse). Les

nombres d'octets et adresses sont donnés en hexadécimal. Voici la trace de l'information récoltée, en ordre chronologique, pendant l'exécution d'un petit programme. Quelles erreurs d'utilisation de la mémoire ou fuites de mémoire à la fin peut-on en déduire? Dans chaque cas, expliquez le problème et donnez le numéro de la ligne correspondante dans la trace. **(2 points)**

```
01: malloc, 0x40000000, 0x10
02: malloc, 0x40000040, 0x20
03: write, 0x40000050, 0x4
04: read, 0x40000050, 0x8
05: write, 0x40000004, 0x4
06: malloc, 0x40000060, 0x20
07: malloc, 0x40000080, 0x20
08: write, 0x40000000, 0x8
09: read, 0x40000000, 0x8
10: free, 0x40000040
11: free, 0x40000060
12: read, 0x40000084, 0x8
13: write, 0x40000084, 0x8
14: free, 0x40000060
15: read, 0x40000050, 0x4
16: free, 0x40000080
```

Pour chaque accès en écriture, on vérifie que la mémoire a été allouée. Pour chaque accès en lecture, on vérifie que la mémoire est allouée et a été initialisée. Pour chaque free, on vérifie que la mémoire était allouée. Finalement, on vérifie une fois le programme terminé que chaque section allouée a été désallouée.

Bien sûr toute la mémoire sera libérée implicitement à la fin du programme, mais il est mieux de la désallouer explicitement au moment où elle n'est plus utilisée. Ceci rend disponible cette région mémoire pour une réallocation, et permet aussi plus facilement de vérifier la cohérence des allocations dynamiques de mémoire dans le programme.

A la ligne 4, on lit 8 octets à l'adresse 0x40000050 alors que seulement 4 ont été initialisés à cette adresse à la ligne 3.

A la ligne 12, on lit 8 octets qui n'ont jamais été initialisés.

A la ligne 14, on désalloue l'adresse 0x40000060 alors qu'elle l'a déjà été à la ligne 11

A la ligne 15, on lit l'adresse 0x40000050 alors que ces octets ont été désalloués à la ligne 10.

A la fin du programme, l'adresse 0x40000000, allouée à la ligne 1, n'a jamais été désallouée et ceci constitue donc une fuite de mémoire.

c) Pour chacun des deux cas suivants d'ennuis avec un logiciel, quel serait un bon outil à suggérer pour trouver le problème: i) un problème de corruption est présent et on suspecte un accès incorrect à la mémoire comme un débordement de vecteur en C/C++, et ii) le temps écoulé est très long, mais le temps en exécution en mode usager est normal, si bien que le problème semble se situer dans l'interaction avec le système d'exploitation ou avec les autres tâches. **(1 point)**

Dans le premier cas, un outil comme Address Sanitizer, ou encore Valgrind Memcheck (plus lent mais un peu plus complet), permettra de vérifier la cohérence des accès en mémoire et de détecter la plupart des accès incorrects, comme les débordements en mémoire. En effet, Address Sanitizer maintient une carte des cases mémoire allouées, et vérifie avant chaque accès en mémoire s'il est à une case allouée. De plus,

Address Sanitizer maintient un espace tampon entre les objets alloués en mémoire, afin justement de créer des accès à des zones non allouées lorsqu'on déborde la mémoire allouée à un objet.

Pour le second cas, le problème est probablement au niveau de la contention pour les ressources dans le système d'exploitation (CPU préempté par d'autres tâches, fautes de pages en raison d'utilisation abusive de la mémoire par certains processus, contention au niveau des disques...). Un outil de traçace au niveau du système d'exploitation, comme perf, ftrace ou LTTng, sera très utile pour comprendre ce qui se passe et diagnostiquer de tels problèmes. Il serait aussi possible dans certains cas d'avoir une idée du problème en regardant diverses métriques dans /proc sur l'utilisation de la mémoire, les fautes de pages, la longueur de la queue des tâches prêtes à s'exécuter...

Question 4 (5 points)

a) Quelle est la sortie produite par ce programme? Comment se nomme l'opération bien connue effectuée sur le vecteur v dans ce programme? **(2 points)**

```
void seqprfsum(int *u, int m)
{ int s = 0; for(int i = 0; i < m; i++) { u[i] += s; s = u[ i ]; }}

void parprfsum (int *x, int n, int *z)
{
  #pragma omp parallel
  { int i, j, me = omp_get_thread_num ( ),
        nth = omp_get_num_threads( ), chunksize = n / nth,
        start = me * chunksize;
    seqprfsum(&x[start], chunksize);

    #pragma omp barrier
    #pragma omp single
    { for(i = 0; i < nth - 1; i++) z[i] = x[(i+1) * chunksize - 1];
      seqprfsum(z ,nth - 1);
    }
    if(me > 0 ) for(j = start; j < start + chunksize; j ++) x[j] += z[me - 1];
  }
}

int main(int argc, char **argv)
{ int v[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
  int sp[4];

  omp_set_num_threads(4);
  parprfsum(v, 16, sp);
  for(int i = 0; i < 16; i++) printf("%d ", v[i]);
  printf("\n");
}
```

Ce programme fait une somme préfixe. Dans une première passe, il calcule localement la somme préfixe dans chaque morceau. Ensuite, il fait une somme préfixe entre les morceaux pour calculer, pour chaque

morceau, la somme associée aux morceaux précédents. Dans une seconde passe, pour chaque morceau, il ajoute la somme des morceaux précédents à chaque élément. La sortie est la suivante:

```
0 1 3 6 10 15 21 28 36 45 55 66 78 91 105 120
```

b) Vous proposez de réaliser un programme parallèle pour solutionner un système d'équations linéaires, $b = Ax$, par la méthode itérative de Jacobi. Soit A la matrice des coefficients, b le vecteur des valeurs, et x le vecteur des variables, on peut définir D la matrice diagonale de A, et O la matrice en enlevant la diagonale de A ($O = A - D$). La solution à l'itération $k+1$ est donnée en fonction de la solution à l'itération k par $x_{k+1} = D^{-1}(b - Ox_k)$. On vous demande de démontrer comment fonctionne la méthode itérative de Jacobi en l'appliquant manuellement sur les données suivantes, avec le vecteur x de solution initiale (i.e. pour x_0) tel que spécifié. Donnez la valeur obtenue pour le vecteur x après la première itération (x_1). **(2 points)**

```
A = np.array([
    [4, 2, 1, 1],
    [3, 4, 1, 0],
    [3, 0, 5, 1],
    [1, 2, 1, 4]
])
b = np.array([5, 0, 10, 8])
x_init = np.array([1, 1, 1, 1])
```

Il suffit de créer les matrices D et O et d'appliquer l'équation.

```
[0, 2, 1, 1] [1]   [4]   | [ 5]   [4]   [ 1]
[3, 0, 1, 0] [1] = [4]   | [ 0] - [4] = [-4]
[3, 0, 0, 1] [1]   [4]   | [10]   [4]   [ 6]
[1, 2, 1, 0] [1]   [4]   | [ 8]   [4]   [ 4]

[1/4,   0,   0,   0]   [ 1]   [0.25]
[  0, 1/4,   0,   0] x [-4] = [  -1]
[  0,   0, 1/5,   0]   [ 6]   [ 1.2]
[  0,   0,   0, 1/4]   [ 4]   [   1]
```

c) On veut paralléliser la multiplication de grandes matrices, C = AB, de dimension $n \times n$, sur m noeuds à l'aide de la librairie MPI. Est-il préférable de diviser le travail en donnant à calculer à chaque noeud pour la matrice résultat C: i) n/m rangées, ou ii) une sous-matrice de $(n/\sqrt{m}) \times (n/\sqrt{m})$? Pourquoi? Combien d'additions et de multiplications chaque noeud devrait-il faire dans chaque cas? **(1 point)**

Le nombre de calculs à faire dans chaque cas sera le même. En effet, pour chaque élément, il faut faire le produit scalaire du vecteur de n éléments de la rangée correspondante avec le vecteur de n éléments de la colonne correspondante, soit n multiplications et $n-1$ additions. Dans le premier cas, chaque noeud fera le calcul sur n/m rangées de n éléments, soit n^2/m éléments. Dans le second cas, le calcul sera fait sur une sous-matrice de $(n/\sqrt{m}) \times (n/\sqrt{m}) = n^2/m$ éléments, soit exactement le même nombre. L'avantage de la répartition de ii) est que chaque noeud a besoin de moins de données. En effet, dans le premier cas, chaque noeud doit contenir n/m rangées de n éléments de A et toute la matrice B (n^2 éléments), alors que dans le second cas, chaque noeud doit avoir $n/\sqrt{m}$ rangées de A et $n/\sqrt{m}$ colonnes de B, pour un total de $2n^2/\sqrt{m}$ éléments. Par exemple, pour n = 1000 et m = 100, cela donnerait 1010000 éléments pour i) et 200000 pour ii).

Le professeur: Michel Dagenais