POLYTECHNIQUE MONTRÉAL

Département de génie informatique et génie logiciel

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

EXAMEN FINAL

DATE: Mardi le 17 décembre 2024

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 $n = 5$, le nombre d'éléments à traiter par un *work item*, et les matrices *a*, *b* et *c* de dimension 5×5, qui sont stockées en mémoire sous la forme d'un vecteur de 25 éléments. Les *work item* sont organisés en 2 dimensions et la taille du problème est de 5 dans chaque dimension. La taille des *work group* n'est pas précisée. Le contenu des matrices est de a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5}, b = {0, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 9, 8, 7, 6}, c = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}. Quelle est l'opération effectuée par cette fonction OpenCL? Quelle sera la valeur en sortie de $c[12]$? **(2 points)**

```
__kernel void mm(const int n, __global float *a,
    __global float *b, __global float *c)
{ int k;
  int i = get_global_id(0);
  int j = get_global_id(1);
  for (k = 0; k < n; k++)
    c[i*n+j] += a[i * n + k] * b[k * n + j];
}
```

b) Le programme OpenCL précédent, de la question 1 a), doit être appliqué à des matrices de grande taille et on vous demande de l'optimiser. Proposez jusqu'à 4 améliorations possibles (pas nécessairement applicables simultanément) qui pourraient permettre à ce programme d'être plus rapide. Justifiez. **(2 points)**

c) Plusieurs nouveaux processeurs présentent une intégration plus forte et supportent la mémoire virtuelle partagée entre le CPU et le GPU pour les applications. Quelles sont les conséquences d'une mémoire virtuelle partagée CPU-GPU sur la programmation d'applications qui utilisent le GPU, et sur leur performance? **(1 point)**

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, rank, size, n = 40, m = 7, chunk, start;
  int in[40], h[7], hl[7];
  for(i = 0; i < n; i++) in[i] = i % 7;
  for(i = 0; i < m; i++) hl[i] = 0;

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  chunk = n / size; start = chunk * rank;
  for(i = start; i < start + chunk; i++) hl[in[i]]++;
  printf("hl[%d] = %d\n", rank, hl[rank]);

  MPI_Reduce(hl, h, m, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
  if(rank == 0) {
    printf("h = {");
    for(i = 0; i < m; i++) printf("%d ", h[i]);
    printf("}\n");
  }
  MPI_Finalize();
}
```

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; o1[i] = o2[i] = 0; }
  MPI_Scatter(in, 1, MPI_INT, o1, 1, MPI_INT, 0, MPI_COMM_WORLD);
  for(i = 1; i < 4; i++) o1[i] = o1[0] + rank * i;
  MPI_Alltoall(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();
}
```

c) Quelle est la différence de comportement entre les fonctions MPI_Send, MPI_Isend et MPI_Bsend? **(1 point)**

Question 3 (5 points)

a) Vous compilez un programme avec les options de profilage de `gprof` et l'exécutez ensuite. Ce programme ne comporte qu'un seul fil d'exécution, qui est interrompu par `gprof` à chaque 1ms afin de noter la fonction dans laquelle se trouve le compteur de programme. De plus, à chaque appel, `gprof` note le décompte du nombre d'appels de chaque provenance (fonction appelante). L'information donnée dans le tableau qui suit est ainsi produite pendant l'exécution. Calculez pour chaque fonction le temps passé dans la fonction elle-même (self) et le temps passé dans la fonction elle-même plus ses enfants (self+childs), comme le ferait l'outil `gprof`. **(2 points)**

Fonction	échantillons	appels	appelants
main	4	1	
f1	1	3	main: 3
f2	3	5	main: 2, f1: 3
f3	8	8	f1: 5, f2: 3
f4	9	6	f1: 1, f2: 2, f3: 3
f5	7	7	f2: 4, f4: 3
f6	10	3	f3: 3

b) Un outil de mesure comme Heap Profile mémorise le contenu de la pile d'appels (la fonction courante et celles dans le chemin d'appel) ainsi que le nombre d'octets alloués, à chaque appel pour allouer de la mémoire (e.g. malloc et new). Voici les échantillons récoltés. Calculez le nombre d'octets alloués dans chaque fonction elle-même (self) et dans chaque fonction elle-même plus ses enfants (self+childs). **(2 points)**

```
(main), 32 octets
(main, f1, f2, f3), 96 octets
(main), 64 octets
(main, f2), 48 octets
(main, f3, f4, f5), 24 octets
(main, f1, f3, f5), 40 octets
(main, f3, f5, f6), 48 octets
(main, f1, f5), 16 octets
(main, f2, f4, f6), 56 octets
(main, f1, f3, f5), 72 octets
```

c) Vous suspectez un problème dans votre programme, relié à l'accès à une matrice qui occasionne un trop grand nombre de fautes de cache, et qui en ralentit l'exécution. Quel serait un bon outil pour diagnostiquer un tel problème? Expliquez comment fonctionne l'outil en question. Comment interagit-il avec l'exécution de votre programme pour prendre des mesures, quel traitement fait-il sur ces mesures, et quelle présentation vous fait-il des résultats afin de vous aider à comprendre d'où vient le problème? **(1 point)**

Question 4 (5 points)

a) Un programme de tri parallèle doit trier N nombres en ordre croissant, sur M processus qui se trouvent sur autant de noeuds dans une grappe d'ordinateurs. Les N nombres ont déjà été répartis également en M fichiers d'entrée, un pour chacun des M noeuds. Pour commencer, i) chaque processus lit ses nombres et les trie en ordre croissant. Ensuite, ii) chaque processus prend 10 nombres dans sa liste de nombres triés, en position 0, M/9, 2M/9... M-1, et envoie cette sous-liste au processus racine (rank == 0). iii) Le processus racine fusionne en ordre croissant les sous-listes des M processus afin de produire une liste qui contient 10M nombres. iv) Le processus racine extrait ensuite les nombres qui divisent cette liste en M intervalles afin de les utiliser comme pivots, en position 0, 10, 20, ... M-1, et il envoie cette liste de pivots à chaque processus. v) Chaque processus envoie alors ses nombres aux autres processus, en envoyant au processus de rang i tout nombre qui tombe dans l'intervalle i, selon les pivots reçus du processus racine. vi) Chaque processus trie les nombres reçus et écrit le résultat dans son fichier de sortie. La liste complète triée est alors disponible en lisant un à la suite de l'autre les M fichiers de sortie, un par noeud. Est-ce que cet algorithme produira effectivement un fichier trié à la fin? Combien d'opérations de comparaison, en fonction de N et M, est-ce que chaque étape prendra sur chaque processus? Pour simplifier la question, on ne considère pas les envois de message, les accès en mémoire, ou les lectures et écritures, seulement les comparaisons. **(2 points)**

b) Un programme OpenMP s'exécute sur m processeurs. Il doit mettre une matrice de taille $n \times n$ à la puissance 250. Comment pourriez-vous réaliser cette opération? Ne composez pas un programme, mais expliquez les opérations arithmétiques qu'un tel programme réaliserait. Combien d'opérations d'addition et de multiplication est-ce que chacun des m processeurs devra effectuer? **(2 points)**

c) La solution d'un ensemble d'équations représenté par $b = Ax$ peut se faire par la méthode itérative de Jacobi avec l'équation $x_{k+1} = D^{-1}(b - Ox_k)$, où on calcule x à l'itération k+1 à partir de x à l'itération k, et où D est la diagonale de A et O est A - D. Si b et x sont de dimension n et A de $n \times n$, combien d'additions/soustractions et de multiplications, en fonction de n, est-ce que le calcul d'une itération requiert? **(1 point)**

Le professeur: Michel Dagenais