ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF8601: Systèmes informatiques parallèles (Automne 2014)
3 crédits (3-1.5-4.5)

EXAMEN FINAL

DATE: Mercredi le 17 décembre 2014

HEURE: 13h30 à 16h00

DUREE: 2H30

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Convertissez le programme suivant en code efficace écrit en assembleur Vectoriel MIPS. **(3 points)**

```
double a[N], b[N], c[N], d[N];
int i;

for(i = 0 ; i < N ; i++)
  if((a[i] * b[i]) != 0) d[i] = (a[i] + b[i]) * c[i];
```

b) Une unité centrale de traitement vectorielle est telle que décrite dans le livre et contient une de chacune des unités suivantes: i) addition et soustraction point flottant, ii) multiplication point flottant, iii) division point flottant, iv) opérations entières, v) opérations logiques, et vi) rangement et chargement. Ces unités en pipeline permettent de produire une nouvelle valeur à chaque cycle. Les instructions scalaires (e.g. L.D) prennent un cycle d'exécution chacune. Dans ces unités en pipeline, les additions et soustractions ont une profondeur de pipeline de 7, les multiplications de 12, les divisions de 18 et les chargements et rangements de 10. Calculez le temps requis pour l'exécution de la séquence d'instructions suivante. **(1 point)**

```
L.D      F0, Ra
L.D      F1, Rb
LV       V1, Rc
MULVS.D  V1, V1, F0
LV       V2, Rd
MULVS.D  V2, V2, F1
ADDVV    V3, V1, V2
MULVS.D  V3, V3, F0
SV       V3, Re
```

c) Les GPU sont constitués de plusieurs processeurs SIMD, chacun étant constitué de plusieurs éléments simples de calcul (thread processor). Est-ce que différents processeurs SIMD peuvent travailler sur différents programmes ou fonctions? Est-ce que différents éléments simples de calcul d'un même processeur SIMD peuvent travailler sur différents programmes ou fonctions? Expliquez. **(1 point)**

Question 2 (5 points)

a) Le programme suivant calcule PI par la méthode de Monte-Carlo. La fonction srand48 initialise le générateur de nombres aléatoires alors que la fonction drand48 fournit un nombre point flottant (double) entre 0 et 1. Ainsi, ce programme génère un point dans le carré -1 à 1 (longueur de 2), en x et y, et détermine si ce point est à l'intérieur du cercle de rayon 1,

ce qui devrait être le cas dans une proportion de PI (aire du cercle) / 4 (aire totale du carré). En faisant la somme du nombre de fois où c'est à l'intérieur, divisé par le nombre d'essais, on doit approximer PI / 4 et donc PI en multipliant ce résultat par 4. On vous demande de convertir ce programme en fonction kernel OpenCL qui fait un travail équivalent. Vous pouvez supposer que les fonctions srand48 et drand48 et les points flottants *double* sont disponibles en OpenCL. Vous pouvez laisser certaines opérations simples être faites par le programme qui appelle cette fonction, par exemple le calcul final de pi. Expliquez quels sont les paramètres d'entrée et de sortie de votre fonction kernel et comment vous suggérez de grouper le travail en *work item* et en groupes. **(3 points)**

```
int i, inside, nb = 100000000;
double x, y, pi;
long sum_pi = 0;

srand48(0);
for(i = 0; i < nb; i++) {
  x = drand48() + drand48() - 1.0;
  y = drand48() + drand48() - 1.0;
  inside = sqrt(x*x + y*y) <= 1 ? 1 : 0;
  sum_pi += inside;
}
pi = (sum_pi * (double)4.0) / nb;
```

b) Dans la fonction OpenCL qui suit, on veut faire une copie locale du vecteur a, local_a, afin d'accélérer la suite. Cette copie est répartie entre les *work item* du groupe, de sorte que chaque item copie sa part du vecteur (1024 / local_size éléments). Une fois cette copie terminée, le gros des calculs, chaque calcul pouvant utiliser tous les éléments du vecteur local_a, doit commencer. Le chargé de laboratoire vous suggère qu'il faut ajouter un énoncé de synchronisation après la copie locale et avant les vrais calculs. Est-ce vrai? Quel énoncé devriez-vous mettre? Quel est son effet? Que serait le problème qui pourrait se produire sans cet énoncé de synchronisation? **(1 point)**

```
__kernel void compute(__global float *a, __global float *b) {
  int local_id = get_local_id(0);
  int local_size = get_local_size(0);
  __local float local_a[1024];

  /* Copie locale: chaque work item copie localement
     1024 / local_size éléments */
  for(i = local_id; i <= 1024; i += local_size) {
    local_a[i] = a[i];
  }

  /* Vrais calculs: une fois la copie de a complète,
```

```
      le calcul commence */
  ...
```

c) Dans le travail pratique 2, vous avez comparé la performance de OpenMP et de OpenCL pour différents problèmes (images de différentes tailles). Pour quel genre de problème utiliseriez-vous plutôt OpenMP ou plutôt OpenCL? Donnez un exemple. **(1 point)**

Question 3 (5 points)

a) La fonction suivante fait l'intégration numérique par la méthode trapézoïdale d'une fonction f. Ecrivez un programme MPI qui appelle cette fonction afin de réaliser cette intégration en parallèle sur un grand intervalle (qui sera séparé en petits intervalles à traiter sur chaque noeud). Le début et la fin de l'intervalle à traiter ainsi que l'incrément (step) vous seront fournis dans des variables globales auxquelles votre programme peut faire référence: `begin, end, step`. Votre programme doit imprimer le résultat final sur le noeud 0. **(2 points)**

```
float integre_trapeze(float begin, float end, float step) {
  float resultat = 0;
  float x;
  int i;

  resultat = (f(begin) + f(end))/2.0;
  x = begin;
  for (x = begin + step; x < end; x += step) {
    resultat += f(x);
  }
  resultat *= h;
  return resultat;
}
```

b) Le programme suivant s'exécute sur 4 noeuds (MPI_Comm_size retourne 4). Donnez le contenu des deux lignes (in et o1) imprimées sur chaque noeud? **(2 points)**

```
int main (int argc, char *argv[])
{
  int size, rank, i, in[4], o1[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 + 2 * rank * i + rank * rank; o1[i] = 0;
  }
```

```
  MPI_Allgather(in + rank, 1, MPI_INT, o1, 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]);
  MPI_Finalize();
}
```

c) Dans le cadre du travail pratique 3, à la question 3.4, vous avez tracé un graphique de l'accélération selon le nombre de processeurs, avec une courbe pour chaque taille d'image. Décrivez l'allure de ces différentes courbes. Est-ce que l'accélération en fonction du nombre de processeurs croit plus vite pour les petites ou les grandes images? Comment expliquez-vous cela? **(1 point)**

Question 4 (5 points)

a) L'outil Helgrind de Valgrind permet de détecter les courses, un des problèmes les plus importants dans les programmes parallèles multithread. Donnez un exemple d'accès à une variable partagée par plus d'un thread en décrivant le cas où il y a une course et le cas où des primitives de synchronisation empêchent une course. Expliquez comment Helgrind peut distinguer les deux et détecter le cas où la course est présente. **(2 points)**

b) L'outil gcov peut indiquer le nombre de fois que chaque bloc de code est exécuté. L'outil gprof fournit un profil du temps d'exécution pour chaque section du programme. Les deux peuvent donc donner une bonne indication des points chauds d'un programme. Quels sont les avantages de gprof par rapport à gcov pour évaluer le temps passé dans chaque section d'un programme? **(1 point)**

c) Les outils OProfile ou perf peuvent, tout comme gprof, fournir un profile du temps d'exécution d'un programme. Quels sont les avantages de ces programmes par rapport à gprof? En quoi sont-ils plus versatiles? **(1 point)**

d) La virtualisation permet, entre autres, de partager une machine physique entre plusieurs machines virtuelles. Ce partage est-il utile pour le calcul parallèle? En quoi les machines virtuelles peuvent-elles être utiles pour le calcul parallèle? **(1 point)**

Le professeur: Michel Dagenais