ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

Cours INF8601: Systèmes informatiques parallèles (Automne 2016)
3 crédits (3-1.5-4.5)

EXAMEN FINAL

DATE: Jeudi le 22 décembre 2016

HEURE: 9h30 à 12h00

DUREE: 2H30

NOTE: Toute documentation permise, calculatrice non programmable permise

Ce questionnaire comprend 4 questions pour 20 points

Question 1 (5 points)

a) Un programme en C a été converti en assembleur Vectoriel MIPS. Le code source en C a été égaré. On vous demande de donner la section de programme en C la plus simple possible à laquelle pourrait correspondre le programme Vectoriel MIPS qui suit. **(2 points)**

```
        LD      R1, N
        LD      R2, a
        LD      R3, b
        LD      R4, c
        ANDI    R5, R1, #63
        DSRL    R6, R1, #6
        ADDI    R6, R6, #1
        MTC1    VLR, R5
loop:   LV      V1, R2
        LV      V2, R3
        SLTV.D  V1, V2
        ADDV.D  V3, V1, V2
        CVM
        SGEV.D  V1, V2
        MULV.D  V3, V1, V2
        CVM
        SV      R4, V3
        MTC1    VLR, #64
        ADDI    R2, R2, #64 * 8
        ADDI    R3, R3, #64 * 8
        ADDI    R4, R4, #64 * 8
        ADDI    R6, R6, #-1
        BNEZ    R6, loop
```

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 8, les multiplications de 16, les divisions de 24 et les chargements et rangements de 12. Calculez le temps requis pour l'exécution de la séquence d'instructions suivante. **(2 points)**

```
1   LV      V3, R3
2   LV      V4, R4
3   SUBV.D  V5, V2, V1
4   SUBV.D  V6, V3, V4
5   ADDV.D  V1, V5, V6
```

```
6   SUBV.D   V3, V5, V6
7   DIVVS.D  V2, V1, F1
8   DIVVS.D  V4, V3, F1
9   SV       V2, R5
10  SV       V4, R6
```

c) Plusieurs nouveaux systèmes avec CPU et GPU intégrés, de compagnies comme AMD et Intel, offrent une mémoire virtuelle partagée entre les CPU et GPU. Quel est l'impact de ce changement? Donnez au moins deux conséquences, nouvelles possibilités ou opérations qui ne sont plus nécessaires. **(1 point)**

Question 2 (5 points)

a) La fonction OpenCL suivante est exécutée pour 1023 *work items*. Le vecteur d'entiers `v` contient (1, 2, 3, 4, 5..., 1024). Quelle est la valeur des paramètres `s` et `d` à la fin de l'exécution de cette fonction OpenCL. **(2 points)**

```
__kernel void sumo(__global long *s, __global long *d,
    __global const long *v)
{
  __local long local_s = 0;
  __local long local_d = 0;
  int i = get_global_id(0);
  atomic_add(local_s, v[i]);
  atomic_add(local_d, v[i+1] - v[i]);
  barrier(CLK_LOCAL_MEM_FENCE);
  if(get_local_id(0) == 0) {
    atomic_add(*s, local_s);
    atomic_add(*d, local_d);
  }
}
```

b) Une fonction OpenCL fait un traitement simple sur les éléments d'une matrice. On vous propose deux versions. Laquelle sera la plus performante? Pourquoi? On vous suggère de changer pour avoir chaque *work item* qui traite une rangée complète de la matrice. Quel impact cela aurait-il sur le nombre de *work item*? Selon quel critère doit-on choisir d'avoir plus de *work item* plus courts, ou moins de *work item* mais plus longs? **(2 points)**

```
__kernel void MatrixA(const __global float* input,
    uint matrix_width, uint matrix_height,
    __global float* output)
{
  int i = get_global_id(0);
```

```
  int j = get_global_id(1);
  int pos = j * matrix_width + i;
  if(input[pos] < 0) output[pos] = 0;
  else output[pos] = input[pos];
}

__kernel void MatrixB(const __global float* input,
    uint matrix_width, uint matrix_height,
    __global float* output)
{
  int j = get_global_id(0);
  int i = get_global_id(1);
  int pos = j * matrix_width + i;
  if(input[pos] < 0) output[pos] = 0;
  else output[pos] = input[pos];
}
```

c) Sur les nouvelles architectures avec CPU et GPU intégrés, de compagnies comme AMD et Intel, on trouve maintenant des queues de commandes en mode usager (user-level queues) pour donner du travail au GPU. Quel est l'impact de cette nouvelle fonctionallité sur le coût d'exécution d'une commande, comme l'exécution d'une fonction (kernel en OpenCL)? **(1 point)**

Question 3 (5 points)

a) Le programme MPI suivant s'exécute sur 4 noeuds (MPI_Comm_size retourne 4). Quelle est la sortie produite par ce programme à l'écran? **(2 points)**

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
  int i, j, rank, size, chunk, start, end, tmp, res;
  int v[] = {11, 9, 7, 5, 3, 1, 10, 8, 6, 4, 2, 0};
  MPI_Status status;
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  chunk = sizeof(v) / sizeof(int) / size;
  start = rank * chunk;
  end = start + chunk;
```

```
  for(i = start; i < end; i++) {
    for(j = i; j < end; j++) {
      if(v[j] < v[i]) {
        tmp = v[i]; v[i] = v[j]; v[j] = tmp;
      }
    }
  }

  printf("Node %d: %d\n", rank, v[start]);
  MPI_Reduce(v+start,&res,1,MPI_INT,MPI_MIN,0,MPI_COMM_WORLD);
  if(rank == 0) printf("Result: %d\n", res);
  MPI_Finalize();
}
```

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

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
  int size, rank, i, i1[4], i2[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++) i1[i] = rank * i + i*i;
  MPI_Alltoall(i1,1,MPI_INT,i2,1,MPI_INT,MPI_COMM_WORLD);
  for(i = 0; i < 4; i++) i2[i] = i2[i] * rank;
  MPI_Alltoall(i2,1,MPI_INT,o1,1,MPI_INT,MPI_COMM_WORLD);
  printf("%d: i1={%d,%d,%d,%d}\n",rank,i1[0],i1[1],i1[2],i1[3]);
  printf("%d: i2={%d,%d,%d,%d}\n",rank,i2[0],i2[1],i2[2],i2[3]);
  printf("%d: o1={%d,%d,%d,%d}\n",rank,o1[0],o1[1],o1[2],o1[3]);
  MPI_Finalize();
}
```

c) Expliquez la différence entre les fonctions MPI_Send, MPI_Bsend et MPI_Isend. **(1 point)**

Question 4 (5 points)

a) Vous compilez un programme avec les options de profilage de gprof et l'exécutez ensuite. L'information suivante est produite pendant l'exécution. Sur 12 échantillons du compteur de

programme récoltés à intervalle régulier d'une seconde, 6 sont dans la fonction C, 3 dans D, 2 dans A et 1 dans B. Le nombre d'appels de chaque fonction (avec la provenance des appels entre parenthèses) est `main`: 1 (pas d'appelant), A:1 (main:1), B:2 (A:2), C:3 (A:3), D:6 (B:4, C:2), E:2 (A:1, B:1). Calculez pour chaque fonction le temps passé dans la fonction elle-même et le temps passé dans la fonction elle-même plus ses enfants, comme le ferait l'outil `gprof`. **(2 points)**

b) Pour chacun des 4 cas suivants, suggérez un outil qui serait particulièrement approprié pour découvrir ou cerner le problème. Expliquez comment cet outil permet de détecter ou cerner chaque problème. i) Un programme de calcul à paralléliser prend trop de temps en raison de certaines fonctions qui ne sont pas assez optimisées ou qui pourraient être parallélisées, on veut identifier ces fonctions. ii) Un programme fait par un étudiant de première année est bourré de problèmes comme l'accès à des variables non initialisées, des débordements de vecteurs et des libérations prématurées de mémoire. iii) Un programme s'exécute avec une bonne performance en moyenne et produit un résultat correct mais présente, à de rares occasions, des latences trop élevées pour accomplir certaines tâches. iv) Un programme présente tout probablement des problèmes de faux partage en mémoire cache, et on veut identifier à quel endroit ceci se produit dans le programme. **(2 points)**

c) Pour le deuxième travail pratique, vous avez repris un algorithme sériel afin de le paralléliser avec OpenMP. Après avoir simplement ajouté des directives OpenMP parallel for, il y avait des sources de bruits dans la sortie générée. Quelle en était la cause? Est-ce que ces problèmes affectaient aussi la performance? **(1 point)**

Le professeur: Michel Dagenais