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)

EXAMEN FINAL

DATE: Lundi le 11 décembre 2017

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) L'ordinateur le plus rapide du monde, selon le site top500.org, Sunway TaihuLight, contient 40 960 processeurs SW26010 de 256 coeurs de traitement à 1.45GHz plus 4 coeurs de gestion, pour 10 649 600 coeurs au total. Sa puissance de calcul, pour un banc d'essai simple très facile à paralléliser, est de 93 014 Tera Flop/s. Calculez le nombre d'opérations point flottant par coeur et par cycle. Suggérez une ou des configurations possibles et probables pour les processeurs (nombre d'unités arithmétiques parallèles, nombre de cycle par instruction) qui expliqueraient une telle performance. **(2 points)**

b) Une unité centrale de traitement vectorielle est telle que décrite dans le livre et contient une de chacune des unités suivantes: i) addition, soustraction et comparaison 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. LD, DRSL) prennent un cycle d'exécution chacune. Dans ces unités en pipeline, les additions, soustractions et comparaisons point flottant ont une profondeur de pipeline de 10, les multiplications point flottant de 15, les divisions point flottant de 22 et les chargements et rangements de 12. Calculez le temps requis pour l'exécution de la séquence d'instructions suivante. **(2 points)**

```
1  LV      V1, R2
2  LV      V2, R3
3  SLTV.D  V1, V2
4  ADDV.D  V3, V1, V2
5  SGEV.D  V1, V2
6  MULV.D  V3, V1, V2
7  SV      R4, V3
```

c) Dans une grille de noeuds de calcul, comme celle du produit MRG de Red Hat, il peut arriver qu'un noeud soit surchargé ou doive être arrêté, et qu'une de ses tâches doive être déménagée sur un autre noeud. Dans d'autres cas, un noeud peut tomber en panne subitement et ses tâches en cours d'exécution sont perdues. Comment le service de gestion des tâches de la grille réagit-il, et quels mécanismes utilise-t-il, lorsqu'un noeud doit être libéré? Lorsqu'un noeud tombe en panne subitement? **(1 point)**

Question 2 (5 points)

a) On vous demande d'écrire la fonction OpenCL PolyMax afin de trouver la valeur maximale dans un grand vecteur v qui contient 4Gi (2^{32}) entiers longs non signés. Le résultat doit être placé dans la valeur de retour globale res. Le GPU utilisé contient 64 processeurs SIMD de 64 coeurs chacun, pour un total de 4096 ALU parallèles. Ecrivez la fonction OpenCL PolyMax afin d'effectuer efficacement cette tâche. Expliquez le choix que vous aurez fait de la quantité de travail effectuée dans chaque *work item* et de la taille des *work group*. **(3 points)**

```
__kernel void PolyMax(__global const unsigned long *v,
    __global unsigned long *res)
{
// Que faire?
}
```

b) Dans le travail pratique 2, vous deviez calculer une image avec le programme sinoscope en utilisant d'abord OpenMP et ensuite OpenCL. Lequel permettait d'obtenir la meilleure performance. Expliquez comment chacun des différents facteurs suivants peuvent affecter la performance relative des deux solutions: i) nombre de coeurs du CPU, ii) nombre de coeurs de la carte graphique, iii) taille de l'image, iv) mémoire virtuelle partagée ou non entre le CPU et le GPU? **(1 point)**

c) Un programme OpenCL appelle la fonction *clEnqueueNDRangeKernel* pour faire exécuter un kernel. Lorsque la fonction retourne *CL_SUCCESS*, est-ce que cela veut dire que le kernel s'est exécuté avec succès? Sinon, comment peut-on savoir à quel moment l'exécution du kernel s'est terminée et s'il n'y a pas eu d'erreur? **(1 point)**

Question 3 (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 à l'écran? Est-ce que le programme comporte une erreur? Expliquez. **(2 points)**

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

int main (int argc, char *argv[])
{
  int i, j, rank, size, dest, data, max = 0, res;
  int v[] = {2, 4, 8, 6, 12, 20, 18, 16, 32, 29, 31, 22};
  MPI_Status s;
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  if(rank == 0) {
    for(i = 0; i < sizeof(v) / sizeof(int); i++) {
      dest = (i % (size - 1)) + 1;
      MPI_Send(v + i, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
    }
  } else {
```

```
    for(i = 0; i < sizeof(v) / sizeof(int); i++) {
      dest = (i % (size - 1)) + 1;
      if(dest == rank) {
        MPI_Recv(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &s);
        if(data > max) max = data;
      }
    }
    fprintf(stderr, "Result: rank = %d, max = %d\n",rank,max);
    MPI_Reduce(&max,&res,1,MPI_INT,MPI_MAX,0,MPI_COMM_WORLD);
  }

  if(rank == 0) fprintf(stderr, "Reduced Result: %d\n", res);
  fprintf(stderr, "Program end, rank = %d\n", rank);
  MPI_Finalize();
}
```

b) Le programme suivant s'exécute sur 5 noeuds (MPI_Comm_size retourne 5). i) Donnez une sortie possible produite par ce programme à l'écran. ii) Le même résultat pourrait être obtenu avec les fonctions Send et Recv plutôt que ISend et IRecv utilisées, en quoi ISend et IRecv peuvent-elles être plus efficace pour un programme parallèle? iii) Expliquez la fonction du 5ème argument de la fonction Irecv qui est présentement à 0. Que serait le résultat si cette valeur était plutôt à 1? **(3 points)**

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

int main (int argc, char *argv[])
{
  int i, rank, size, prev, next, val, newval;
  MPI_Status s[2];
  MPI_Request r[2];
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  prev = rank - 1;
  next = (rank + 1) % size;
  val = rank * rank;

  for(i = 0; i < 6; i++) {
    MPI_Irecv(&newval,1,MPI_INT,prev,0,MPI_COMM_WORLD,&r[0]);
    MPI_Isend(&val,1,MPI_INT,next,0,MPI_COMM_WORLD,&r[1]);
    MPI_Waitall(2, r, s);
    val = newval;
```

```
    }
    printf("Rank %d, value %d\n", rank, val);

MPI_Finalize();
}
```

Question 4 (5 points)

a) Un fil d'éxécution d'un programme s'exécute et, à chaque 1ms, le fil est interrompu très brièvement pour prendre un échantillon avec le compteur de programme et le contenu de la pile d'appel. Ces échantillons sont éventuellement copiés vers un fichier. Chaque échantillon contient donc l'adresse d'exécution courante ainsi que l'adresse dans toutes les fonctions appelantes. A la fin de l'exécution, ces échantillons sont traités pour convertir chacune de ces adresses en identificateur de la fonction dans laquelle elle se trouve. Pour une exécution donnée, voici les échantillons recueillis (les échantillons sont séparés par des ";" avec pour chacun à gauche la fonction associée au compteur de programme et à droite les fonctions retrouvées dans l'ordre en remontant la pile d'appel. Calculez pour chaque fonction retrouvée dans les échantillons (main, A, B, C, D, E, F) le temps passé dans la fonction elle-même (self) et le temps passé dans la fonction et ses enfants (self+childs). **(2 points)**

```
D C A main; D C A main; E C A main; F D B main;
D B main; D B main; C A main; C B main; main
```

b) Un programme s'exécute à raison de 1G instruction / seconde (ou 1000 MIPS). Les blocs de base du programme ont une longueur moyenne de 10 instructions. Chaque appel de fonction exécute en moyenne environ 100 instructions. La fonctions mcount, utilisée pour compter le nombre d'appels et leur provenance pour une fonction, nécessite l'exécution de 8 instructions. La prise d'un échantillon du compteur de programme demande 1000 instructions et s'exécute à chaque 1ms si cela est activé. i) Quel sera le surcoût d'utiliser un outil comme gcov qui ajoute l'exécution d'une instruction pour chaque bloc de base? ii) Quel sera le surcoût d'utiliser un outil comme oprofile qui échantillonne le compteur de programme à chaque 1ms? iii) Quel sera le surcoût d'un outil comme gprof qui compte le nombre d'appels de chaque fonction et leur provenance, en plus d'échantillonner le compteur de cycle à chaque 1ms? **(2 points)**

c) Donnez un exemple de problème de performance qui ne pourrait être diagnostiqué facilement avec gprof mais qui pourrait aisément l'être avec oprofile. **(1 point)**

Le professeur: Michel Dagenais