ECOLE POLYTECHNIQUE DE MONTREAL

Département de génie informatique et génie logiciel

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

EXAMEN FINAL

DATE: Mercredi le 18 décembre 2013

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) Un programme reçoit dans un vecteur *in* des informations lues de capteurs. Il doit effectuer une conversion et ajouter une calibration *cal* spécifique à chaque capteur. Convertissez ce programme en code efficace écrit en assembleur Vectoriel MIPS. **(3 points)**

```
double in[N], out[N], cal[N];
int i;

for(i = 0 ; i < N ; i++) out[i] = ((in[i] + cal[i]) * 1.8 + 32);
```

b) Le Xeon Phi peut traiter 512 bits en parallèle, sous la forme de 16 valeurs point flottant de 32 bits, ou 8 de 64 bits. Son pipeline d'exécution comporte 12 étages mais permet de produire un résultat de 512 bits à chaque cycle. Comment cela se compare-t-il avec l'architecture vectorielle MIPS étudiée, en termes de matériel requis, de performance et de bande passante requise venant de la mémoire? **(1 point)**

c) Vous désirez mettre sur votre liste de cadeaux un ordinateur pour le calcul parallèle. Vous hésitez entre un co-processeur de type GPU (e.g. AMD ou NVidia) ou un Intel Xeon Phi. Quels sont les principales différences architecturales et avantages de chacun? **(1 point)**

Question 2 (5 points)

a) On veut générer un histogramme pour la valeur des pixels dans une grande image. Le code en C pour effectuer ce travail est fourni. Proposez une implémentation efficace en OpenCL qui effectue ce travail. Fournissez le code pour la fonction de type kernel correspondante. Expliquez par ailleurs les arguments fournis pour l'appel de cette fonction. **(3 points)**

```
void Histogram(unsigned char image[4096][4096],
    unsigned int histogram[256])
{
  int i, j;

  for(i = 0; i < 4096; i++) {
    for(j = 0; j < 4096; j++) {
      histogram[image[i][j]]++;
    }
  }
}
```

b) Que fait la fonction `barrier(CLK_LOCAL_MEM_FENCE)`? Donnez un exemple de cas où son utilisation serait requise? **(1 point)**

c) Dans le travail pratique 2, vous deviez calculer une image avec le programme sinoscope en utilisant d'abord OpenMP et ensuite OpenCL. À partir de quelle superficie d'image valait-il mieux utiliser OpenCL et la carte graphique plutôt que le processeur avec OpenMP à 8 fils? Expliquez pourquoi. **(1 point)**

Question 3 (5 points)

a) Le programme suivant doit s'exécuter en 21 processus sur autant de noeuds. Il décompose un vecteur en morceaux de taille semblable et demande à chaque processus d'additionner un nombre à chaque élément dans son morceau, en plus de faire la somme de ses éléments. Pour chacun des 21 processus, dites combien d'éléments font partie du morceau de vecteur à traiter (`mysize` tel qu'imprimé sur chaque noeud)? Le programme bloque avant la fin et reste en attente. Quel est le problème? **(2 points)**

```
#define SIZE 100000000
float  v[SIZE];

int main (int argc, char *argv[])
{
  int i, j, rank, size, offset, chunksize, mysize, extra;
  double mysum = 0.0, sum = 0.0;
  MPI_Status s;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  chunksize = SIZE / size;
  extra = SIZE % size;
  if(rank == 0) {
    for(i=0; i < SIZE; i++) {v[i] =  i; sum += v[i]; }
    offset = chunksize;
    for (i = 1; i < size; i++) {
      mysize = i < (size - extra) ? chunksize : chunksize + 1;
      MPI_Send(&offset, 1, MPI_INT, i, 1, MPI_COMM_WORLD);
      MPI_Send(&mysize, 1, MPI_INT, i, 2, MPI_COMM_WORLD);
      MPI_Send(&v[offset], mysize, MPI_FLOAT, i, 3, MPI_COMM_WORLD);
      offset = offset + mysize;
    }
    offset = 0; mysize = chunksize;
  }
  else {
```

```
    MPI_Recv(&offset, 1, MPI_INT, 0, 1, MPI_COMM_WORLD, &s);
    MPI_Recv(&mysize, 1, MPI_INT, 0, 2, MPI_COMM_WORLD, &s);
    MPI_Recv(&v[offset], mysize, MPI_FLOAT, 0, 3, MPI_COMM_WORLD, &s);
    printf("%d: mysize = %d\n", rank, mysize);
  }

  for(i = offset; i < offset + mysize; i++) {
    v[i] = v[i] + i; mysum = mysum + v[i];
  }
  MPI_Reduce(&mysum, &sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

  if(rank == 0) {
    for(i = 1; i < size; i++) {
      MPI_Recv(&offset, 1, MPI_INT, i, 1, MPI_COMM_WORLD, &s);
      MPI_Recv(&mysize, 1, MPI_INT, i, 2, MPI_COMM_WORLD, &s);
      MPI_Recv(&v[offset], mysize, MPI_FLOAT, i, 2, MPI_COMM_WORLD, &s);
    }
    printf("Sum = %f\n",sum);
  } else {
    MPI_Send(&offset, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
    MPI_Send(&mysize, 1, MPI_INT, 0, 2, MPI_COMM_WORLD);
    MPI_Send(&v[offset], mysize, MPI_FLOAT, 0, 3, MPI_COMM_WORLD);
  }
MPI_Finalize();
}
```

b) Le programme suivant s'exécute sur 4 noeuds (MPI_Comm_size retourne 4). Donnez le contenu des deux lignes (o1 et o2) imprimées sur chaque noeud? **(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 + rank * rank; o1[i] = o2[i] = 0; }
  MPI_Allgather(in, 1, MPI_INT, o1, 1, MPI_INT, MPI_COMM_WORLD);
  MPI_Alltoall(in, 1, MPI_INT, o2, 1, MPI_INT, MPI_COMM_WORLD);
  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) Dans le cadre du travail pratique 3, vous avez mesuré la performance des fonctions de communication de type bloquant (blocking), avec tampon (buffered) et asynchrone

(async). Expliquez comment fonctionne chaque type, et décrivez et expliquez les résultats obtenus pour ces types de communication dans le cadre du travail pratique 3? **(1 point)**

Question 4 (5 points)

a) Vous voulez évaluer le temps d'exécution pris par les différentes parties d'un programme afin de l'optimiser. Trois options s'offrent à vous: l'outil de profilage `gprof` qui instrumente les entrées de fonction et échantillonne le compteur de programme à chaque miliseconde, l'outil de trace `lttng` avec l'entrée et la sortie de chaque fonction qui ont été instrumentées, et l'outil `Perftools CPU profile` qui échantillonne le contenu de la pile d'exécution à chaque miliseconde. Comparez ces trois options en termes de surcoût ajouté au programme et d'informations obtenues (complétude, fiabilité, précision...). **(2 points)**

b) Comment l'outil lockdep peut-il détecter la possibilité d'un interblocage, même si celui-ci ne s'est pas produit pendant l'exécution sous analyse? **(1 point)**

c) Quel outil permet efficacement d'estimer les fautes de cache, causées par les différentes sections d'un programme en exécution, avec un surcoût ajouté au programme extrêmement faible? Expliquez? **(1 point)**

d) En calcul parallèle, l'efficacité du traitement est un facteur souvent primordial. Néanmoins, l'utilisation de virtualisation, par exemple avec OpenStack, est de plus en plus populaire pour le calcul parallèle, en dépit de son surcoût qui peut facilement atteindre 10%. Quel en est l'intérêt qui compense pour ce surcoût? **(1 point)**

Le professeur: Michel Dagenais