UNIVERSITÉ DE MONTPELLIER
UNIVERSITÉ DE MONTPELLIER- FACULTÉ DES SCIENCES ET TECHNIQUES
PLACE EUGÈNE BATAILLON - 34095 MONTPELLIER CEDEX 5
M1 Informatique – 2021/2022

HAI720I – Examen n° 1

Programmation algorithmique efficace
[Durée 2h00 – aucun document autorisé]

Question 1 : Donner la définition de l'acronyme CPI ainsi que la formule associée en étant très précis.

Question 2 : Expliquer ce qui signifie ILP pour un processeur. Quelles sont les 4 techniques qui permettent généralement d'améliorer l'ILP dans un programme.

Question 3 : En considérant un processeur superscalaire capable d'exécuter une addition et une multiplication sur un `float` en un seul cycle. Étant donner une fonction qui nécessite de faire N_a additions et N_m multiplications sur des `float` qui n'ont aucune dépendance de données. Déterminer quelle sera la performance maximale de cette fonction sur ce processeur. Vous exprimerez votre résultat en pourcentage par rapport à la puissance crête et vous détaillerez les étapes de votre raisonnement.

Question 4 : En considérant un processeur superscalaire ayant les caractéristiques suivantes :

- un capacité de calcul de 8 opérations sur des `double` par cycle : 2 multiplications et 6 additions ;
- une fréquence d'horloge de 1KHz (1024 cycles/s)
- une bande passante maximale de 8 kB/~~cycle~~ (kiloOctets/cycle) avec la mémoire RAM ;
- un seul niveau de cache de taille 2MB (Méga Octets) avec des blocs de 64 octets.

On se donne une fonction `trmm` qui calcule $C = AB+C$ avec A, B et C des matrices triangulaires supérieures [1] de taille $n \times n$ contenant des `double`. Bien évidemment, on stocke uniquement, et de manière contigüe, les éléments pertinents des matrices, câd $n(n+1)/2$ entrées par matrice. Le nombre d'opérations sur des `double` pour exécuter la fonction `trmm` est $W(n) = n(n+1)(n+2)/3$.

(A) Faire le dessin du modèle *Roofline* pour le processeur considéré. Pour rappel, vous devrez mettre en abscisse l'intensité opérationnelle et en ordonnée la performance du système en flops. Votre dessin utilisera une échelle logarithmique en base 2 pour les abscisses et les ordonnées (préciser bien les unités de mesures). Vous identifierez également les zones représentant les parties *memory bound* et *compute bound*.

(B) Calculer une borne supérieure pour l'intensité opérationnelle de la fonction `trmm` en ne considérant que des *cold miss*. À partir de cette borne déduire pour quelle plage de valeurs de n la fonction `trmm` sera *memory bound*.

(C) Si l'on considère que les matrices A, B et C ne sont plus stockées de manière contigüe, et que toutes les entrées sont distantes d'au moins 64 octets. Quelle est la nouvelle plage de valeurs de n où la fonction `trmm` sera *memory bound*. Vous expliquerez votre raisonnement et votre calcul.

Question 5 : On considère une fonction `maxRowCol(maxrow,maxcol,A)` qui calcule l'ensemble des éléments maximaux par ligne et par colonne d'une matrice A de taille 8×8 stockée de manière contigue au format *row major*. Les vecteurs `maxrow` et `maxcol` sont tous deux de taille 8 et vérifient après l'appel de la fonction

$$\forall i,j, 0 \leq i,j < n, \quad \texttt{maxrow}[i] = \max(A[i,j] \tag{1}$$
$$\texttt{maxcol}[j] = \max(A[i,j] \tag{2}$$

1. toutes les entrées de la matrice sous la diagonale sont nulles

```
void maxRowCol(uint32_t maxrow[8], uint32_t maxcol[8], const uint32_t A[64]){
  for(size_t i=0;i<8;i++){
    uint32_t max=A[i*8]
    for(size_t j=1;j<8;j++)
      max = A[i*8+j]>max? A[i*8+j] : max ;
    }
    maxrow[i]=max;
  }
  for(size_t j=0;j<8;j++){
    uint32_t max=A[j]
    for(size_t i=1;i<8;i++)
      max = A[i*8+j]>max? A[i*8+j] : max ;
    }
    maxcol[j]=max;
  }
}
```

On considère un système d'exécution ayant les caractéristiques suivantes :

- il n'y a qu'un seul niveau de cache avec des blocs de taille 16 octets (bloc= ligne de cache) ;
- le cache est *fully associative*, *write-allocate/write-through* d'une taille totale de 96 octets ;
- le cache utilise une politique de remplacement de bloc *LRU : Least Recently Used* ;
- il y au maximum 8 registres et toutes les variables qui ne sont pas des tableaux seront dans ces registres.

(A) Expliquer en quelques lignes les termes : *cold miss*, *capacity miss* et *conlict miss*

(B) Donner le nombre de lignes de cache de ce système.

(C) Donner le nombre exact de défauts de cache de l'exécution de la fonction `mawRowCol` sur ce système. Expliquer en détail votre calcul.

(D) Quelle doit être la taille minimale du cache pour que l'exécution de la fonction `mawRowCol` sur ce système ne génère que des *cold miss*. Expliquer votre réponse.

(E) Donner le code d'une nouvelle version de la fonction `maxRowCol` permettant de ne faire que des *cold miss* avec un cache de 96 octets. Aide : une seule double boucle est suffisante. Vous expliquerez votre algorithme.

Question 6 : Nous souhaitons effectuer un traitement d'image en niveau de gris avec le langage CUDA sur GPU. Pour cela nous allons utiliser le kernel `traiteImage` donné ci-cessous avec une image de 600×800 pixels (800 pixels sur l'axe horizontal et 600 pixels sur l'axe vertical de l'image). On a donc $m = 600$ et $n = 800$.

```
__global__ void traiteimage(float* Img_in, float* Img_out, int m, int n){

   int row = blockIdx.y*blockDim.y + threadIdx.y;
   int col = blockIdx.x*blockDim.x + threadIdx.x;

   if ((row < m) and (col<n))
     Img_out[row*n+col] = 2*Img_in[row*n+col];
}
```

On décide d'appeler ce kernel avec une grille contenant des blocs de taille 16×16. Cela signifie que chaque bloc est organisé comme un tableau à deux dimension 16×16 de threads.

(A) Donner un exemple valide de l'appel de ce kernel avec une image de taille 600×800. Vous pourrez considérer que les images nécessaires à cet appels sont définies par la déclaration **float** *d_In, *d_Out;

(B) Si l'on considère que nos images 600×800 d'entrée et de sortie sont stockée dans la RAM du CPU (de manière non-unifiée), quelles sont les opérations nécessaires avant de pouvoir appeller le kernel `traiteImage`. Expliciter l'appel aux instructions CUDA correspondantes en considérant que les images sont accessibles dans la RAM CPU via les pointeurs **float** *h_In, *h_Out;

(C) Combien de *warps* seront générés lors de l'appel du kernel `traiteImage` sur des images de 600×800 pixels. Vous expliquerez votre calcul.

(D) Parmi tous les warps générés, combien auront de la divergence de thread ? Expliquer votre raisonnement.

(E) Si on appelle ce kernel avec une image de taille 800×600, est-ce que le nombre de warp qui ont de la divergence reste identique. Si votre réponse est non, donner la nouvelle valeur.

Question 7 : On souhaite implanter le traitement d'image vu à la question précédente en utilisant des instructions SIMD. Nous considérons ici un processeur fictif capable d'utiliser des registres SIMD 256-bits (AVX) et d'effectuer en 1 seul cycle : 1 load, 1 store et 2 additions avec des registres SIMD. Complèter le code ci-dessous pour effectuer le traiment d'image avec la meilleure performance possible. Vous expliquerez pourquoi les performances sont optimales. On considèrera uniquement les options de compilation suivantes : `-O2 -march=native`.

```
// on considère ces deux fonctions comme existantes
void readImageFromFile(float* Img, size_t m, size_t n, string filename);
void writeImageToFile(float* Img, size_t m, size_t n, string filename);

int main(int argc, char** argv){
  size_t m=600,n=800;
  float *Img_in, *Img_out;
  Img_in  = new float[m*n];
  Img_out = new float[m*n];
  readImageFromFile(Img_in, m, n, argv[1]);

  // CODE À DEFINIR

  writeImageToFile(Img_out, m, n, argv[2]);

return 0;
```

Aide : les signatures des fonctions intéressantes sont :

- *__m256 _mm256_load_ps(__m256* mem);*
- *__m256 _mm256_loadu_ps(__m256* mem);*
- *__m256 _mm256_store_ps(__m256* mem, __m256 b);*
- *__m256 _mm256_storeu_ps(__m256* mem, __m256 b);*
- *__m256 _mm256_add_ps (__m256 a, __m256 b);*