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° 2

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

Question 1 : Expliquer en quelques lignes ce qu'est le problème de *memory aliasing*. Vous illustrerez ce problème en proposant un exemple de code en C++.

Question 2 : Modifier le code de la fonction `scal` donnée ci-dessous pour supprimer le problème potentiel de *memory aliasing*. Cette fonction doit modifier le tableau T en multipliant chacun de ses éléments par une même valeur a . Vous devez proposer deux solutions.

```
1 void scal (int* T, size_t n, const int& a){
2 for (size_t i=0;i<n;i++)
3    T[i]*= a;
4 }
```

Question 3 : On considère le code CPU suivant :

```
1  #include <immintrin.h>
2  int main(){
3     double T[]={1,4,8,16,0,0,0,0};
4    __m256d a0, a1;
5      a0= _mm256_load_pd(T);
6    for (size_t i=0;i<3;i++){
7      a1= _mm256_load_pd(T+i+1);
8      a1= _mm256_add_pd(a0,a1);
9      _mm256_store_pd(T+i+1,a1);
10   }
11
12   return 0;
13 }
```

(A) Expliquer pourquoi le code ci-dessous fait un erreur de segmentation à l'execution.

(B) En considérant que l'erreur de segmentation est résolue, expliquer ce que calcule cette fonction.

Question 4 : On se donne un processeur superscalaire qui à les caractéristique suivantes :

- 2 multiplication par cycle, exécutées sur les ports 1 et 2
- 1 addition (ous soustraction) par cycle, exécutée sur le port 1

On se donne les trois fonctions ci-dessous qui calculent toutes le même résultat.

```
1
2 void f1(const float *A, float *R, size_t N){
3   for (size_t i=0;i<N;i+=1){
4     float a=A[i];
5     R[i]= 1 - a*a*a*a;
6   }
7 }
```

```
8  void f2(const float *A, float *R, size_t N){
9    for (size_t i=0;i<N;i+=1){
10     float a=A[i];
11     float x=1-a;
12     float y=1+a;
13     R[i]= (1+a*a) * x * y;
14   }
15 }
16 void f3(const float *A, float *R, size_t N){
17   for (size_t i=0;i<N;i+=1){
18     float a=A[i];
19     float x=a*a;
20     R[i]=  (1 - x) * (1 + x);
21   }
22 }
```

On considère également que le compilateur ne fait aucune optimisation sur l'exécution des instructions (les instructions sont exécutées tel que décrit dans le code); les opérations entières pour la gestion de la boucle seront à ignorer dans la suite.

(A) Pour chacune des fonctions, donner le nombre de cycles minimum que prend son exécution sur notre processeur sachant que l'on ne considère pas les dépendances de données (par exemple, a+b*c contient 2 opérations indépendantes). Vous justifierez votre réponse.

(B) Faite le même calcul, mais cette fois-ci en considérant les dépendances de données. Vous justifierez votre réponse, par exemple en utilisant un DAG pour représenter une itération de la boucle.

(C) On modifie notre processeur pour qu'il puisse faire 2 additions par cycle sur les ports 1 et 2; et 1 multiplication par cycle sur le port 1 (on nommera cette configuration *beta*). Quel est le nombre minimum de cycles pour chacune des fonctions en considérant les dépendances de données. Vous justifierez votre réponse.

(D) On propose de modifier la configuration *beta* de notre processeur pour que les 2 additions par cycle s'exécutent sur le port 2 et un nouveau port, identifé 3. La multiplication reste exécutée sur le port 1. Est-ce que cela change l'analyse de la question précédente ? si oui expliquer pourquoi.

(E) Pensez-vous qu'il soit possible de trouver une configuration de processeur superscalaire, avec autant de ports d'exécution et d'addition ou multiplication par cycle que vous voulez, qui améliore les performances de la configuration *beta*. Vous justifierez votre réponse.

Question 5 : On se donne une fonction `matmulAddT(C,A,B,N)` qui pour des matrices A, B, C de taille $N \times N$ effectue le calcul $C = (A + A^T) \times B$, où A^T représente la matrice transposée de A.

```
1  void matmulAddT(float *C, const float* A, const float* B, size_t N){
2    for(size_t i=0;i<N;i++)
3      for(size_t j=0;j<N;j++){
4        C[i*N+j]=0.;
5        for(size_t k=0;k<N;k++){
6          C[i*N+j] += ( A[i*N+k] + A[k*N+i] ) * B[k*N+j];
7        }
8      }
9  }
```

(A) Calculer l'intensité arithmétique **exacte** $I(N)$ de cette fonction.

(B) Donner une borne supérieure pour l'intensité opérationnelle $Q(N)$ de cette fonction. Vous détaillerez l'ensemble de vos calculs menant à cette borne.

(C) Donner les caractéristiques précises du modèle *Ideal Cache Model* vue en cours.

(D) En considérant ce modèle, faite l'analyse du nombre de défaut de cache de cette fonction. Vous utiliserez les variables B et M pour représenter la taille d'un bloc et la taille du cache en octets. Votre analyse doit traiter les cas : *cold miss* uniquement et celui intégrant des *capacity miss*, en précisant les valeurs de N possibles pour chacun de ces cas.

(E) Pensez-vous qu'il soit possible d'obtenir une meilleure complexité en cache, lorsque les données ne tiennent pas toutes dans le cache, simplement en inversant l'ordre des boucles. Si oui donner l'ordre des boucles choisi et justifier votre réponse.

(F) Nous souhaitons améliorer cette fonction en utilisant la technique de découpage en blocs. Pour cela, notre fonction prendra en paramètre un entier b représentant la taille des blocs $b \times b$ utilisés. On prendra comme hypothèse que N est un multiple de b. Donner le code complet de cette nouvelle fonction *matmulAddT_blocked*.

(G) Quel est la condition sur la valeur de l'entier b pour qu'il y ait un nombre de défauts de cache optimal.

(H) Parmis les 4 propositions données ci-dessous, laquelle correspond à la complexité en cache $MT(N)$ de la fonction *matmulAddT_blocked* (en considérant celle-ci correctement écrite). Vous justifierez votre choix.

- R1 : $O(N^2/M)$
- R2 : $O(N^3b/M)$
- R3 : $O(N^2/b^2$
- R4 : $O(N^3b/B)$

Question 6 : Quizz GPU

1. Si l'on souhaite allouer un tableau de `v` entiers dans la mémoire globale CUDA, quel serait la bonne expression pour le second argument dans l'appel de la fonction `cudaMalloc` ?
   (A) `n`
   (B) `v`
   (C) `n * sizeof(int)`
   (D) `v * sizeof(int)`
   (E) `sizeof(v)`
2. Si l'on souhaite allouer un tableau de `n` nombre flottants, dans la mémoire globale CUDA, au travers de la variable `float* dA`, quelle serait la bonne expression pour le premier argument dans l'appel de la fonction `cudaMalloc` ?
   (A) `n`
   (B) `v`
   (C) `n * sizeof(int)`
   (D) `v * sizeof(int)`
   (E) `sizeof(v)`
3. Si l'on souhaite copier les mille premiers éléments d'un tableau `hA` contenant des `float`, dont les données sont stockées dans la mémoire globale du GPU, vers une zone mémoire du CPU, identifiée par le pointeur `float* dA`, quel est le bon appel à effectuer :
   (A) `cudaMemcpy(1000, hA, dA, cudaMemcpyHostToDevice);`
   (B) `cudaMemcpy(hA, dA, 1000, cudaMemcpyHostToDevice);`
   (C) `cudaMemcpy(dA, hA, 1000, cudaMemcpyDeviceToHost);`
   (D) `cudaMemcpy(dA, hA, 4000, cudaMemcpyHostToDevice);`
   (E) `cudaMemcpy(4000, hA, dA, cudaMemcpyDeviceToHost);`