Sorbonne Université et Polytech Sorbonne
Calcul haute performance : programmation et algorithmique avancées

Examen 9 février 2023, durée 2 heures (13h45-15h45)

Uniquement les notes de cours de la partie II sont autorisées

Il est impératif de rédiger la réponse des deux parties sur des copies différentes.

Partie I (sur 12,5 points)

QCM Une seule bonne réponse par question (5 points)

Bonne réponse +1 point, aucune réponse 0 point, mauvaise réponse −0.5 point.

1. Quelle est la valeur maximale prise par `blockDim.x*gridDim.x` lorsqu'on exécute `kernel<<<16,256>>>();` ?
   a) 4096
   b) 4080
   c) 3825
2. Supposons 64 blocks de un seul thread exécutant la ligne
   `printf("%i, ", blockIdx.x);`
   On obtient
   a) Obligatoirement : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63,
   b) Un affichage aléatoire, par exemple : 0, 1, 33, 54, ...
   c) Ni l'un ni l'autre
3. Dans quel cas on favorise l'utilisation d'une mémoire locked au lieu d'une mémoire Mapped sur le Host ?
   a) Lorsque le GPU n'a pas suffisamment d'espace sur sa mémoire globale.
   b) Lorsque les données sont peu utilisées par le GPU et très souvent changées par le CPU.
   c) Lorsque les données sont très souvent utilisées par le GPU et peu changées par le CPU.
4. Désigner la fausse affirmation
   a) La taille totale en octets des registres est plus importante que la taille en octets de la shared.
   b) Le nombre de registres par block dépend du nombre de blocks.
   c) Le nombre de registres par thread dépend du nombre de threads.
5. Le calcul du temps d'exécution en utilisant les `cudaEvent_t`
   a) Est recommandé pour optimiser les kernels.
   b) Requiert une synchronisation entre le Host et le Device.
   c) Est recommandé pour optimiser une application faisant intervenir le Host et le Device.

Problème : Multiplications de petites matrices (7,5 points)

Nous disposons d'un nombre $N = 2^{16}$ $(= 65536)$ de $d \times d$ matrices carrées $(A^n)_{1\leq n\leq N}$ et $(B^n)_{1\leq n\leq N}$ où ^{n} désigne un indice et pas une puissance. Nous souhaitons définir le kernel

`multiBatch_k(float *A, float *B, int d)`

d'une multiplication matricielle batch $(A^n \times B^n)_{1\leq n\leq N}$ efficace pour chaque $d = 1, 2, ..., 1024$.

Nous rappelons que pour deux matrices a et b de composantes respectives $(a_{i,j})_{1\leq i,j\leq d}$ et $(b_{i,j})_{1\leq i,j\leq d}$ où i parcourt l'indice de la ligne et j l'indice de la colonne, le produit $c = a \times b$ donne une matrice de composantes $(c_{i,j})_{1\leq i,j\leq d}$ où chaque $c_{i,j}$ resulte du produit scalaire du vecteur ligne $a_{i,1\leq k\leq d}$ avec le vecteur colonne $b_{1\leq k\leq d,j}$ i.e. $c_{i,j} = \sum_{k=1}^{d} a_{i,k}b_{k,j}$.

A l'appel du kernel `multiBatch_k`, nous supposons que les pointeurs `A` et `B` en entrée contiennent respectivement l'ensemble des matrices $(A^n)_{1\leq n\leq N}$ et $(B^n)_{1\leq n\leq N}$. Nous supposons ainsi que la mémoire globale du GPU est suffisante. Les valeurs sont disposées ligne après ligne ce qui donne par exemple : $A[0] = A^1_{1,1}$, $A[1] = A^1_{1,2}$, ..., $A[d-1] = A^1_{1,d}$, $A[d] = A^1_{2,1}$, ..., $A[d^2-1] = A^1_{d,d}$, $A[d^2] = A^2_{1,1}$, ..., $A[N(d^2-1)] = A^N_{d,d}$.

Nous supposerons aussi que le résultat de la multiplication est sauvegardé dans l'espace mémoire pointé par `A` (écrase la valeur de l'entrée).

Nous distinguerons les cas : d petit ($1 \leq d \leq 3$), d moyen ($4 \leq d \leq 32$) et d grand ($33 \leq d \leq 1024$).

1. Pour d petit ($1 \leq d \leq 3$), nous associons un thread à chaque multiplication de deux matrices.

a. Donner "un bon" choix de nombres de threads et de nombres de blocks à lancer. (0,5 point)

b. Définir le kernel `multiBatch_k` qui permet de faire cette multiplication. (1,5 points)

c. Expliquer pourquoi cette solution n'est pas adaptée pour ($4 \leq d \leq 32$). (1 point)

2. Pour d moyen ($4 \leq d \leq 32$), nous associons d^2 threads à chaque multiplication de deux matrices. Lors du produit matriciel $c = a \times b$, le calcul des d coefficients $\{c_{i,k}\}_{1 \leq k \leq d}$ de chaque ligne i est parallélisé sur d^2 threads, d threads par coefficient pour calculer un produit scalaire parallélisé.

a. Expliquer comment peut-on choisir le nombre de threads et le nombre de blocks à lancer selon la valeur de d. (1 point)

b. Définir le kernel `multiBatch_k` qui permet de faire cette multiplication. (1,5 points)

c. Expliquer pourquoi cette solution n'est pas adaptée pour ($33 \leq d \leq 1024$). (1 point)

3. Pour d grand ($33 \leq d \leq 1024$), combien de threads proposez-vous par multiplication de deux matrices ? Justifier. (1 point)

Partie II (sur 7,5 points)

Stockage des matrices

Soit A une matrice carrée d'ordre n, avec nnz coefficients non nuls.

1. Écrire en pseudo-code l'algorithme calculant le produit transposé $A^T x = y$, où A est stockée dans le format CSR, x et y sont deux vecteurs de taille n.
2. On suppose maintenant que A est symmétrique et que seulement sa diagonale et sa partie triangulaire supérieur sont stockées. Si A est décomposée par blocs de lignes pour tirer profit du parallèlisme à mémoire distribuée :
   — que devient l'algorithme du produit matrice-vecteur $Ax = y$ si y n'est pas distribué, i.e., si y est connu en entier par tous les processus ?
   — que devient l'algorithme si maintenant y suit la même distribution que A ?
3. Pour chacune des trois matrices suivantes (de taille respective 5×5, 4×4 et 4×4) pour lesquelles on représente les coefficients non nuls par des croix, choisir avec une courte justification quel est le format de stockage le plus approprie entre DIA, COO et CSR.

$$A_1 = \begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix} \quad A_2 = \begin{bmatrix} \times & \times & \times & \\ & & & \\ & & & \\ \times & & \times & \times \end{bmatrix} \quad A_3 = \begin{bmatrix} \times & & & \\ & & & \times \\ & & \times & \\ & \times & & \times \\ \times & & & \end{bmatrix}$$

Méthodes de décomposition de domaine et préconditionnement

Soit $\Omega = [0;1]$. On s'intéresse au préconditionnement d'un système linéaire avec une méthode de Schwarz. On décompose Ω en deux sous-domaines $\{\Omega_i\}_{i \in \{1,2\}}$. On va supposer que $\Omega_1 = [0;1] \times [0;2h]$ et que $\Omega_2 = [0;1] \times [h;1]$, avec $h = \frac{1}{3}$.

1. A quoi correspond l'action de $\{R_i\}_{i \in \{1,2\}}$ (respectivement $\{R_i^T\}_{i \in \{1,2\}}$) sur un vecteur u (respectivement $\{u_i\}_{i \in \{1,2\}}$) ?
2. Écrire la matrice de restriction R_1 (plusieurs choix sont possibles).
3. Quel(s) paramètre(s) permet(tent) d'accélérer la convergence d'une méthode de Schwarz avec recouvrement ?
4. Justifier brièvement quel type de parallèlisme (à mémoire distribuée ou à mémoire partagée) est le plus approprié pour la méthode de Jacobi, respectivement Gauss–Seidel, par blocs.