PPS-ET5 - Cours de programmation GPU

TP 2 : Programmation CUBLAS

Stéphane Vialle, CentraleSupélec & LISN, *Stephane.Vialle@centralesupelec.fr*

8 janvier 2025

Objectifs du TP

Ce TP a pour objectif de pratiquer la programmation d'un GPU au sein d'un nœud de calcul CPU+GPU en utilisant des appels de bibliothèque : il consiste à implanter un produit de matrices denses sur un GPU en utilisant la bibliothèque CUBLAS.

On expérimentera différents types de données et de précision des calculs, et on comparera les précisions des résultats obtenus et les vitesses de calculs mesurées.

Plate-forme de développement

Les machines utilisées seront principalement celles du cluster **Tx** du **DCE** de CentraleSupélec :

- **Tx** : chaque machine contient un CPU Intel XEON quad-core hyperthreadés, et un GPU NVIDIA RTX-2080 Ti (architecture Turing)

L'environnement CUDA C et C++ est disponible sur chaque machine (et donc le compilateur "**nvcc**" et les drivers pour utiliser le GPU sont disponibles).

Vous utiliserez les comptes de TP **ppsgpu_*i*,** où ***i*** est une valeur entière entre **1** et **15**. Utilisez le même numéro que pour les TP sur clusters de CPU.

Connexion aux serveurs GPU du DCE

A travers *vscode* ou directement dans un *shell* :

- Connectez vous à **chome.metz.supelec.fr** avec votre login **ppsgpu_*i***

- Pendant le TP :
  - UN SEUL des deux membres du binôme alloue un serveur GPU :

    ```
    srun -N 1 --exclusive --reservation=CodeResa --pty bash
    ```

  - L'AUTRE membre du binôme se connecte directement à la machine allouée par son binôme à l'aide d'un simple ssh :

    ```
    ssh .....
    ```

- Après le TP (si besoin) :
  - Pour obtenir un serveur **Tx** (RTX 2080 Ti) pour 2h :

    ```
    srun -N 1 --exclusive -p gpu_inter -C tx --pty bash
    ```

- Pour obtenir une machine **tx** précise (RTX 2080 Ti) pour 2h **si elle est libre** : utiliser l'option -w

  `srun -N 1 --exclusive -p gpu_inter -w tx11 –pty bash`

  Remplacez tx11 par la machine sh utilisée en TP

Attention :

- Si vous perdez le lien avec la machine allouée et que vous devez en réallouez une : le second membre du binôme devra se connecter aussi à cette nouvelle machine (et ne pas rester connecté sur la précédente !).

- Si la commande **`srun`** (lancée sur chome.metz.supelec.fr) n'aboutit plus : c'est probablement qu'une session « zombie » existe toujours associée à votre compte.
  - Il faut l'identifier avec la commande : **`mysrun`**
    Qui vous donnera son *Id*
  - Puis la tuer avec la commande : **`scancel`** *Id*

Tournez SVP →

Travail à effectuer

Remarques préliminaires :

Le squelette de programme que vous utiliserez contient un code de produit de matrices denses en OpenMP et CUDA.

- La partie OpenMP est complète, et est destinée à permettre de vérifier les résultats obtenus en CUDA.
- La partie CUDA est en partie développée (les kernels K0 et K1 du TP1 sont disponibles), mais il vous reste à compléter la suite du fichier **gpu.cu**.
- Le squelette est compilable et contient une aide intégrée (./MatrixProduct -h)

Pour valider vos codes vous compilerez en **Simple Précision et comparerez vos résultats à ceux des kernels K0 ou K1 du premier TP** :

- Laissez "**#-DDP**" dans le Makefile

  → le type "T_real" devient alors le type "float (sur 32 bits)"
- Les résultats de vos premiers pseudo-kernels appelant les CUBLAS sur les CUDA-Cores doivent être très proches de ceux des kernels K0 et K1 du premier TP et proches de ceux du CPU en kernel -cpu-k 1 (erreur de l'ordre de 1E-5).

Pour faire vos mesures de performances vous compilerez aussi en **Simple Précision :**

- La simple précision est adaptée aux capacités des GeForce GTX1080, RTX2080, RTX3080 et RTX3090.

1 - Implantation d'un appel CUBLAS "gemm" suivi du lancement d'un kernel de transposition

a. Continuez à compléter le squelette de programme du TP1 si vous avez réussi à faire les kernels K0 et K1.

Sinon :

Récupérez et compilez le nouveau squelette de programme OpenMP+CUDA

cp ~vialle/PPS-GPU/MatrixProduct-CUDA-cublas-enonce-2.zip .

b. Dans les deux cas chargez les Openblas :

- Ajoutez la bibliothèque Openblas à votre configuration (il existe plusieurs versions des « blas », chaque utilisateur charge donc celle qu'il veut) en utilisant la commande **module**.
  - Sur une machine **Tx** :

    ```
    module load openblas/0.3.26/gcc-13.1.0-openmp
    ```
- Compilez votre squelette de programme:

  ```
  make
  ```
- Exécutez la commande

  `./MatrixProduct -h` pour voir les détails de fonctionnement de l'application.

c. **Implantez le pseudo-kernel K4 dans le fichier 'gpu.cu' pour calculer C = AxB :**
   - utilisez la routine **CUBLAS_GEMM(...)** (voir main.h), il s'agit du renommage de cublasSgemm ou de cublasDgemm, selon que l'on compile en simple ou double précision (-DDP dans le Makefile),
   - utilisez la macro **CHECK_CUBLAS_SUCCESS** (voir gpu.h) pour lancer vos appels CUBLAS et tester leurs codes de retour,
   - et lancez une grille de blocs du **kernel TransposeKernel_v1**, associé à des blocs carrés de BLOCK_SIZE_XY_KT1 x BLOCK_SIZE_XY_KT1 threads, pour transposer le résultat de l'appel CUBLAS. Lancez une grille de blocs adaptée à toutes les tailles de matrices (SIZE x SIZE).
   - Testez votre implantation sur une matrice de **4096x4096 DOUBLE** (option -DDP active dans le 'Makefile' et changement de SIZE dans le 'main.h').

     Obtenez-vous les mêmes valeurs qu'avec les BLAS sur CPU : MatrixProduct -t CPU -cpu-k 1 -cpu-nt 4, (sur Tx) ?

d. **Mesurez les performances du pseudo-kernel K4 obtenues sur une matrice de 4096x4096 FLOAT éléments** (#-DDP dans le Makefile) **:**
   - Continuez à remplir le fichier Excel de saisi des perfortmances comme au TP : complétez-le au fur et à mesure du TP2.
   - Comparez aux meilleures performances obtenues sur CPU multi-coeurs en OpenMP avec un kernel de même niveau : kernel 1 (BLAS sur CPU). Mais recherchez le nombre de threads idéal pour le kernel BLAS sur le CPU 4 cœurs physiques des Tx :

     ./MatrixProduct -t CPU -cpu-k 1 -cpu-nt 4

     ./MatrixProduct -t CPU -cpu-k 1 -cpu-nt 8

     Normalement, un thread par cœur physique devrait être l'optimum pour des kernel BLAS (s'ils sont bien installés).
   - Calculez le speedup « GPU-K4 *vs* best-config-CPU »

2 - Implantation d'un appel CUBLAS "gemm" suivi d'un appel CUBLAS "geam"

a. Créez le pseudo-kernel K5 dans le fichier 'gpu.cu' :
   - Utilisez la routine **CUBLAS_GEMM(...)** (voir main.h)

     et la routine **CUBLAS_GEAM(...)** (voir main.h)
   - Testez et validez votre implantation sur une matrice de **4096x4096 FLOAT**

     Vérifiez que vous obtenez les mêmes valeurs qu'avec le pseudo-kernel K4.

b. Mesurez les performances obtenues sur une matrice de 4096x4096 FLOAT
   - Comparez-les aux performances obtenues avec le pseudo-kernel K4
   - Comparez-les aux performances de la meilleure configuration sur le CPU

3 - Implantation d'un unique appel CUBLAS "gemm" permettant d'obtenir C = AxB

a. Créez le pseudo-kernel K6 dans le fichier 'gpu.cu' :
   - Utilisez la routine **CUBLAS_GEMM(...)** (voir main.h) **pour qu'elle calcule C = AxB à elle seule**

- Testez et validez votre implantation sur une matrice de **4096x4096 FLOAT**
  Vérifiez que vous obtenez les mêmes valeurs qu'avec le pseudo-kernel K5

b. Mesurez les performances obtenues sur une matrice de 4096x4096 FLOAT

- Comparez-les aux performances obtenues avec le pseudo-kernel K5
- Comparez-les aux performances de la meilleure configuration sur le CPU

4 - Utilisation des TensorCores avec des types de données et de calculs "standard"

a. Implantez le pseudo-kernel K7 dans le fichier 'gpu.cu' pour exploiter les TensorCores :

- Utilisez la routine **cublasGemmEx(...)**

  Utilisez le type **T_CUBLAS_real** défini dans 'main.h'

  Utilisez le type **T_CUBLAS_COMPUTE_real** défini dans 'main.h'

  Indiquez l'algorithme **CUBLAS_GEMM_DEFAULT_TENSOR_OP**

- Testez et validez votre implantation sur une matrice de **4096x4096 FLOAT**
  Vérifiez que vous obtenez les mêmes valeurs qu'avec le pseudo-kernel K6.

b. Mesurez les performances obtenues sur les TensorCores avec une matrice de 4096x4096 FLOAT

- Comparez-les performances obtenues maintenant avec le pseudo-kernel K6 à celles obtenues précédemment

c. Mesurez les performances obtenues sur les TensorCores avec une matrice de 4096x4096 DOUBLE

- Recompilez en double précision (-DDP dans le Makefile)
- Obtenez-vous les mêmes résultats que le pseudo-kernel K6 ?
- Comparez-les performances obtenues avec celles du pseudo-kernel K6

5 - Utilisation des TensorCores avec des types de calculs en précision réduite

a. Cette question ne sera valable que pour une compilation en Simple Précision

b. Implantez le pseudo-kernel K8 dans le fichier 'gpu.cu' pour exploiter les TensorCores :

- Utilisez la routine **cublasGemmEx(...)**

  Utilisez le type de données : **CUDA_R_32F**

  Utilisez successivement les types de calcul en précision réduite :

  - CUBLAS_COMPUTE_32F_FAST_TF32
  - CUBLAS_COMPUTE_32F_FAST_16F
  - CUBLAS_COMPUTE_32F_FAST_16BF

  Indiquez l'algorithme **CUBLAS_GEMM_DEFAULT_TENSOR_OP**

c. Mesurez les performances obtenues sur les TensorCores avec une matrice de 4096x4096 FLOAT

- Pour les 3 types de calcul en précision réduite notez les résultats obtenus (notamment les écarts avec la version CPU) et les performances obtenues (exécutez chaque test plusieurs fois)
- Retenez le type de calcul le plus performant qui donne des résultats "encore acceptables"
- Comparez-les performances obtenues maintenant avec le pseudo-kernel K6 à celles obtenues précédemment

6 - Evaluation du verrouillage de la mémoire sur la vitesse des transferts de données CPU-GPU

a. **Suspendez les transferts à partir ou vers des données CPU "vérouillées en mémoire" (*pinned*) dans les fonctions *gpuInit* et *gpuFinalize* :**
   - Commentez le verrouillage en mémoire des matrices A, B et C sur le CPU (ne changez par leur allocation/déclaration) dans *gpuInit*
   - Commentez leur déverrouillage dans *gpuFinalize* (qui a lieu après la fin des transferts et des calculs)

b. **Vérifiez que vos transferts mémoires ne déclenchent pas de messages d'erreur et vérifiez que vous obtenez toujours les mêmes résultats** (comparez-les à ceux d'un kernel CPU ou aux valeurs obtenues précédemment sur GPU).

c. **Mesurez les performances obtenues sur une matrice de 4096x4096 FLOAT**
   - Re-mesurez les performances obtenues avec les kernels 4, 5, 6, 7 et 8
   - Remplissez **une copie** du deuxième tableau de l'onglet "K4-8 CUBLAS" du fichier Excel
   - Observez-vous des changements significatifs sur la Bw mesurée et sur la vitesse de calcul globale du produit de matrices sur un serveur Tx ?

d. **Réinstallez le verrouillage des données sur CPU :** décommentez le verrouillage et le déverrouillage de la mémoire dans les fonctions *gpuInit* et *gpuFinalize*

7 - Refaites vos tests sur la machine John9 ou John10 équipées d'un RTX3090

a. Depuis chome.metz.supelec.fr faites un simple ssh john9 ou ssh john10

b. Recompilez sur la machine **John** en **Simple Précision**

c. Relancez chacun de vos kernels (notamment les kernels exploitant les Tensor Cores)

d. Rassemblez les performances dans une **nouvelle copie** des tableaux de l'onglet "K4-8 CUBLAS" du fichier Excel

e. Est-ce que les Tensor Cores apportent un gain de performance sur un RTX3090 (architecture "Ampere") ?

f. Refaites une mesure de performances sans verrouillage de la mémoire CPU. Est-ce que la Bw et les performances globales sont plus fortement impactées sur un serveur John que sur un seveur Tx ?

Document à rendre après le TP2 :

Vous rendrez un SEUL document par binôme ou monôme : un fichier PDF

- Il comportera en haut de la première page :
  - vos nom et prénoms
  - la date du TP,
  - le titre ("TP2 de programmation CUBLAS").
- Il contiendra en général :
  - une description des pseudos-kernels réalisés (codes et explications),
  - vos mesures de performances (Texec et/ou GigaFlops, SpeedUp...) sur GPU, sous forme de tableaux ou de figures,
  - votre analyse des performances obtenues.
- **Taille maximale de vos rapports : 8 pages**
- **Date limite de remise de vos rapports : <voir date annoncée en cours>**
- **Remise par email à Stephane.Vialle@centralesupelec.fr**