PPS-ET5 - Cours de programmation GPU

TP 3 : Programmation CUDA ShM

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

04 février 2025

Objectifs du TP

Ce TP a pour objectif de pratiquer la programmation optimisée d'un GPU en utilisant la *shared memory* de chaque Stream-Multiprocessor. On étudiera notamment la mise au point de synchronisations minimales mais suffisantes des threads (au sein d'un bloc), puis le traitement des *boundaries* pour la mise au point de kernel génériques.

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** :

a. 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 :**

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

1 - Implantation d'une grille "2D" de blocs "2D" et du kernel K2 utilisant la *shared memory*

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-shm-enonce-3.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 kernel K2 et sa grille de blocs de threads dans le fichier 'gpu.cu'** pour que:
   - chaque thread calcule un élément complet de la matrice C = AxB,
   - un bloc de threads soit un bloc 2D carré qui exploite la *shared memory*
   - les accès à la mémoire globale soient coalescents en lecture et en écriture
   - votre programme fonctionne pour des tailles de matrices telles que : SIZE = q.BLOCK_SIZE_XY
   - Testez votre implantation sur une matrice de 4096x4096 FLOAT (#-DDP dans le '**Makefile**'), et vérifiez que vous obtenez les mêmes valeurs qu'avec le kernel K1 sur GPU

c. **Mesurez les performances du kernel K2 obtenues sur une matrice de 4096x4096 FLOAT** éléments (#-DDP dans le Makefile).
   - Récupérez le **fichier Excel** de saisi des résultats, et complétez-le au fur et à mesure du TP.
   - Faites varier la taille de vos blocs 2D carrés de threads, et mesurer les performances obtenues pour des blocs de 1x1 à 32x32 threads.
   - **Est-ce que les performances obtenues semblent conforme à la théorie ? pourquoi ?**

2 - Calcul du nombre d'accès en mémoire globale

a. Calculez le nombre d'accès à la mémoire globale (en lecture et en écriture) en utilisant la *shared memory* (kernel K2), avec des matrices de SIZExSIZE éléments et des blocs de BSXYxBSXY.
   - en comptant les accès demandés par les *threads*
   - en comptant les accès "en râteau" réalisés par les *warps*, fonction de la coalescence des accès : un accès en râteau pouvant ramener 32 données (coalescence parfaite) ou seulement quelques données, ou même une seule en cas de non-coalescence ou de coalescence dégénérée.

b. En déduire le nombre d'accès à la mémoire globale durant tout le produit de matrices :
   - sous forme d'accès demandés par l'ensemble des *threads*
   - sous forme d'accès "en râteau" réalisés par l'ensemble des *warps*

c. Calculez le gain obtenu en nombre d'accès par rapport au kernel K1 (kernel 2D sans *shared memory*)
   - en comptant les accès demandés par l'ensemble des *threads*
   - en comptant les accès "en râteau" réalisés par l'ensemble des *warp*

d. Faites l'application numérique du gain pour des matrices de 4096x4096 et des blocs de 32x32 éléments.
   - est-ce que les gains de performances mesurés expérimentalement vont dans le sens des gains théoriques en nombre d'accès "en râteau" réalisés par les *warps* ?

3 - Implantation d'une grille "2D" de blocs "2D" et du kernel K3 utilisant la *shared memory* pour une utilisation GENERIQUE

a. Implantez un kernel K3 en généralisant votre kernel K2 pour qu'il s'applique à des matrices de taille quelconque

- Etudier les différents problèmes de *boundaries* qui se posent
- Testez le fonctionnement de votre code sur des matrices de 4096x4096 FLOAT, et comparez les résultats à ceux du kernel K1
- PUIS testez votre programme sur des matrices de 4097x4097 FLOAT, et comparez les résultats à ceux du kernel K1

b. Mesurez les performances de votre kernel K3

- Mesurez les performances sur des matrices de 4096x4096 FLOAT pour des blocs de 32x32
- Complétez le fichier Excel
- Comparez les aux performances du kernel K2 : notez-vous un impact des *boundaries* ?

Document à rendre après le TP3 :

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 ("TP3 de programmation CUDA ShM").
- Il contiendra en général :
  - une description de 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**