UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

M2 - Architecture et Programmation d'accélérateurs Matériels.

(APM 2022-2023)

TP2

Asynchronisme – Utilisation des Streams

julien.jaeger@cea.fr
adrien.roussel@cea.fr

Les objectifs de ce TP sont :
— Utilisation des Streams CUDA pour faire du calcul asynchrone
— Permettre le recouvrement des transferts mémoire par du calcul.

I Portage sur GPU (Stream par défaut)

Le code source principal se situe dans la section CODE du présent TP. Le programme représente le calcul d'une fonction exponentielle sur un espace 1D.

Q.1: Portez la fonction "`func`" sur GPU, en configurant une grille 1D qui contienne suffisamment de blocs et de threads pour couvrir l'ensemble du vecteur.

II Initialisation des Streams

Un flux (ou Stream) CUDA est une suite d'opération qui est exécutée selon l'ordre dans lequel elles sont appelées sur le GPU. Ce modèle de programmation est utilisé pour introduire de la concurrence sur le GPU :
- Les opérations CUDA venant de différent flux peuvent s'exécuter de manière concurrentes.
- Les opérations CUDA entre différents flux peuvent être interchangées.

Nous allons dans un premier temps créer et initialiser différents flux à partir du code réalisé lors de la question précédente.

Q.2: Allouez un ensemble `NSTREAMS` de Streams CUDA, et initialisez les.

Q.3: Sans spécifier de Stream sur lequel exécuter un noyau CUDA, quel stream est utilisé ? Quelle particularité a-t-il ?

Q.4: Modifiez le lancement du kernel CUDA pour que `NSTREAMS` instances de ce kernel soient exécutées sur des Streams différents.

Q.5: Qu'observez-vous au niveau des performances ? Pourquoi et comment les résoudre ?

Q.6: Utilisez la fonction cudaMemcpyAsync pour effectuer les copies mémoires vers et depuis le GPU. Quel effet a cette fonction ?
NB : N'oubliez pas de synchroniser le host et le device avant de faire la vérification du résultat avec cudaThreadSynchronize pour éviter toute corruption mémoire.

Q.7: A quoi sert la fonction cudaMallocHost ? Modifiez votre code en conséquence.

Q.8: Classer les fonctions et les appels de kernels en deux catégories : les synchrones et les asynchrones.

Q.9: En fixant `NSTREAMS = 2`, dessiner un schéma représentant les différentes tâches sur les flux d'exécution ainsi que leur dépendances.
Nous prendrons comme notation :
A $\rightarrow$ **B** : l'exécution de B ne peut commencer tant que celle de A n'est pas terminée.
A $\Rightarrow$ **B** : l'exécution de B ne peut se terminer tant que celle de A n'est pas terminée.

Liste des tâches :

Création S0	Création S1	Malloc Host	Thread Sync
requête cpy H→D [0,N[	requête cpy H→D [N,2N[	requête cpy D→H [0,N[	requête cpy D→H [N,2N[
cpy H→D [0,N[	cpy H→D [N,2N[	cpy D→H [0,N[	cpy D→H [N,2N[
requête K1 [0,N[	requête K1 [N,2N[	exécution K1 [0,N[	exécution K1 [N,2N[

III Mesure de temps

Q.10: Utiliser la fonction système `gettimeofday` pour mesurer le temps passé dans l'exécution du programme CUDA.

Q.11: Avec les fonctions `gettimeofday` et `cudaThreadSynchronize` mesurer le temps d'exécution des kernels uniquement (sans compter le temps des transferts mémoire etc...).

Q.12: Quel est l'impact (inconvénient) sur l'exécution générale du programme de cette dernière façon de mesurer ? Vous pouvez vous aider du graphe de dépendance précédent et l'enrichir.

Q.13: Comment mesurer le temps de chaque kernel avec la technique précédente ?

Q.14: Comment utiliser les événements (fonctions `cudaEvent*`) pour mesurer le temps de chaque kernel et le temps total de tous les kernels ? Commentaire par rapport à la façon précédente ?

IV Multiplication Matricielle

Q.15: Reprenez l'exercice de multiplcation matricielle du TP1 et transformez le code pour le rendre multi-streams. Dans ce cas, un stream s'occupera d'un paquet de lignes à traiter.