UE : Dimensionnement et évaluation des architectures (I5AISE51)
Sujet de TD

Institut National des Sciences Appliquées de Toulouse
P.-E. Hladik, pehladik@insa-toulouse.fr

—

TD 2 : allocation mémoire, kernel et performances

Version bêta (4 janvier 2021)

1 Le b.a.-ba du CUDA

Vous allez reprendre le code d'addition de vecteur pour vous exercer.

Objectif 1.1

- utiliser cudaMemcpy, cudaMalloc,
- lancer un kernel.

(1.1) Travail à faire : Addition vectorielle

1. Récupérer le code source `addvec.cu`.
2. Observer le code et modifiez le pour qu'il compile et fasse le travail correctement.
3. Compiler, exécuter et observer.
4. Modifier le code pour prendre en considération une taille quelconque des vecteurs (et non pas simplement un multiple de 512).

(1.1) Comment faire : recopie de la mémoire

La documentation de `cudaMemcpy` est disponible sur
https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html

(1.2) Comment faire : Lancement d'un kernel

Le lancement d'un kernel se fait comme un appel normal à une fonction en ajoutant des paramètres de configuration d'exécution entre <<< et >>> avant les arguments de la fonction. Le premier argument donne le nombre de blocs et le second le nombre de threads par bloc.

2 Performance

Objectif 2.1

— comparer les performances d'un code C et d'un code CUDA

(2.1) Travail à faire : La première classe

1. Récupérer le code `addVecWithoutKernel.cu`, complier et exécuter.
2. Ajouter dans le code de `addVec.cu` des mesures de performance (temps de chargement de la mémoire et temps d'exécution) et afficher les.
3. Compiler et exécuter les deux versions (C et CUDA) de l'addition vectorielle.
4. Comparer les performances. A-t-on gagné quelque chose ?

(2.1) Comment faire : Création et horodatage d'un événemet CUDA

On pourrait utiliser les timers de l'OS sur le CPU, mais pendant que le noyau GPU fonctionne, il se peut que nous effectuions des calculs de manière asynchrone sur l'hôte. Pour résoudre cela on va utiliser des `cudaEvent_t` (événements CUDA).

Un événement dans CUDA est un horodatage GPU qui est enregistré à un moment spécifié par l'utilisateur. L'API est relativement facile à utiliser et ne comporte que deux appels : la création d'un événement et son horodatage. Par exemple, au début d'une séquence de code, nous demandons au moteur d'exécution CUDA d'enregistrer l'heure actuelle. Nous le faisons en créant puis en enregistrant l'événement :

```
cudaEvent_t start;
cudaEventCreate(&start);
cudaEventRecord(start, 0);
```

Le second argument de `cudaEventRecord` restera un mystère, pour les curieux [a].

a. https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__EVENT.html

(2.2) Comment faire : Synchronisation

Malheureusement, il y a un problème de synchronisation. La partie la plus délicate de l'utilisation des événements est due au fait que certains appels dans CUDA C sont asynchrones. C'est excellent du point de vue des performances, car cela signifie que nous pouvons calculer quelque chose sur le GPU et le CPU en même temps, mais cela rend la mesure du temps délicate.

Vous devez imaginer les appels à `cudaEventRecord()` comme une instruction pour enregistrer l'heure actuelle placée dans la file d'attente du GPU. Par conséquent, notre événement ne sera pas réellement enregistré tant que le GPU n'aura pas tout terminé avant l'appel à cudaEventRecord(). C'est précisément ce que nous voulons pour que notre événement d'arrêt mesure l'heure correcte. Mais nous ne pouvons pas lire en toute sécurité la valeur de l'événement d'arrêt avant que le GPU n'ait terminé son travail précédent et enregistré l'événement d'arrêt. Heureusement, nous disposons d'un moyen de demander au CPU de se synchroniser sur un événement, la fonction API d'événement `cudaEventSynchronize()` :

```
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start, 0);
// do something
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);
```

(2.3) Comment faire : Calcul du temps écoulé entre deux événements

Il est possible de connaître le temps en ms séparant deux évènement avec la méthodes `cudaEventElapsedTime`. Par exemple :

```
float elapsedTime;
cudaEventElapsedTime(&elapsedTime, start, stop);
printf("Time %3.1f ms\n", elapsedTime);
```

3 Connaître sa machine (épisode 2)

Message à caractère informatif

Ceci est un exercice bonus si vous vous ennuyez.

Objectif 3.1

— Afficher de nouvelles caractéristiques de sa machine

(3.1) Travail à faire : Lecture des paramètres du GPU

1. Compléter `properties.cu` pour afficher le nombre maximum de threads par bloc, la taille maximum des dimensions d'un grid et d'un block.