M2 - Architecture et Programmation d'accélérateurs Matériels.

(APM 2022-2023)

TD2

Prise en main de CUDA
Approximation de π
par la méthode de Monte Carlo

julien.jaeger@cea.fr
mael.martin@cea.fr

Les objectifs de ce TD sont :
— Configuration d'une grille simple
— Écriture d'un premier kernel CUDA
— Allocation mémoire sur l'accélérateur

I Description du problème

Le calcul de π est un problème fréquemment étudié pour l'apprentissage du parallélisme. L'une des méthodes les plus courantes est la méthode approchée de type *Monte Carlo*.

On construit un carré de côté $c = 1$ et de centre O. En ce même centre, nous traçons un cercle de rayon $r = 1$. Pour comprendre la procédure d'approximation de π, nous avons besoin de connaitre l'aire de ce cercle via la formule $A = \pi r^2$. Dans notre cas, nous obtenons $A = \pi$.

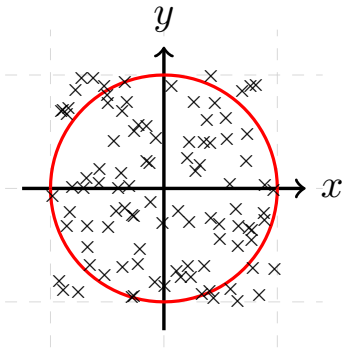

FIGURE 1 – Distribution aléatoire de points dans un carré unitaire.

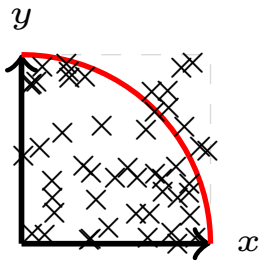

FIGURE 2 – Quart de cercle servant de cible pour le calcul de π

FIGURE 3 – Calcul d'approximation de π par la méthode de Monte Carlo

Intéressons nous uniquement à un quart de la figure. Imaginons maintenant

que l'on joue à un jeu de fléchettes, et que l'on prend cette figure pour cible. La méthode de Monte Carlo consiste à prendre un nombre infini de fléchettes et de les lancer de manière aléatoire en direction de la cible. Par conséquent, le nombre de fléchettes se trouvant dans le cercle tendra vers l'aire du quart de cercle, soit

$$P \simeq \frac{A}{4} = \frac{\pi}{4}.$$

En pratique, P se calcule également de la manière suivante :

$$P = \frac{\text{nb fléchettes dans le cercle}}{\text{nb total de fléchettes}}.$$

On se retrouve donc avec l'approximation de π telle que :

$$\pi \simeq 4 \times \frac{\text{nb fléchettes dans le cercle}}{\text{nb total de fléchettes}}.$$

Pour finir cet algorithme, il nous reste donc à vérifier après un tir si la fléchette se trouve dans le quart de cercle de la cible. Un point a de coordonnées (x, y) se trouve à l'intérieur du quart de cercle si la distance qui le sépare de l'origine $O(0, 0)$ est inférieure ou égale à 1, soit $x^2 + y^2 \leq 1$ (cf. théorème de Pythagore).

II Transposition du problème

1 Séquentielle

Q.1: A l'aide de la description du problème dans la section précédente, écrire un programme séquentiel en C qui permet de calculer l'approximation de π par la méthode de Monte Carlo.

2 Multi-thread

Q.2: Proposer une version parallèle du code précédent écrite avec OpenMP. Pour ce faire, chaque thread lancera un nombre TRIALS_PER_THREAD de fléchettes et stockera localement son résultat partiel de π dans un tableau accessible par tous les threads. La moyenne des différentes valeurs trouvées sera ensuite calculée sur le thread principal et donnera ainsi la valeur finale estimée de π.

III Prise en main avec CUDA

Les méthodes de type Monte Carlo se prêtent très bien aux accélérateurs de calculs type GPGPU car chaque thread exécute les mêmes suites d'instructions, sur des données différentes. Nous allons voir maintenant comment porter ce programme sur un accélérateur de ce type en utilisant le SDK CUDA.

1 Configuration de la grille

Q.3: Pour ce problème, nous allons utiliser une grille 1D pour lancer un grand nombre de threads en simultané. Proposer une configuration de kernel qui s'exécutera sur 512 blocs contenant chacun 256 threads.

2 Allocation mémoire

Q.4: A l'instar de la version OpenMP, nous allons calculer un résultat de π par thread CUDA. Ce résultat doit être stocké en mémoire sur le GPU pour pouvoir être ensuite utilisé côté hôte pour faire la moyenne des valeurs de π trouvées. A l'aide des fonctions du SDK CUDA, allouer un vecteur sur la mémoire du device qui permettra de stocker chaque résultat partiel.

Q.5: Les résultats partiels doivent être manipulés par l'hôte après l'exécution des threads pour faire la moyenne de ceux-ci. Toujours à l'aide des fonctions du SDK CUDA, donner la fonction qui permet de copier les données de l'accélérateur vers le CPU, ainsi que sa position relative dans le code.

3 Écriture du kernel

Q.6: Passons maintenant à l'écriture du kernel qui s'exécutera sur l'accélérateur de calculs. Pour chaque lancer de fléchette, un thread CUDA doit générer deux nombres aléatoires : x pour l'ordonnée, et y pour l'abscisse. A l'aide de la description du problème, écrire le kernel CUDA correspondant. Un thread doit lancer TRIALS_PER_THREAD fléchettes, compter le nombre de fléchettes qui se trouve dans le quart de cercle, et ainsi en déduire une approximation de π. Cette valeur approchée sera ensuite stockée dans une case du vecteur allouée précédemment sur le GPU.
NB : La génération de nombres aléatoires avec CUDA se fait via la bibliothèque *Curand*. Les fonctions utiles à cette question sont définies dans le header curand_kernel.h. L'initialisation de la graine se fait avec la méthode :
`unsigned int curand_init(unsigned long long seed, unsigned long long sequence, unsigned long long offset, curandState_t state)`.
La génération d'un nombre aléatoire dans l'intervalle $[0, 1]$ se fait ensuite avec la méthode :
`float curand_uniform(curandState_t state)`.