Université de Limoges
Faculté des Sciences et Techniques

Année Universitaire 2016 - 2017
version du 18 octobre 2017

Parallélisme 2
Application de filtre parallélisé

Auteurs

Baron Marin
Daviaud Jérémy
Nicot Fabien

Master Informatique, semestre 2

Responsable : Bonnefoi Pierre-François

Introduction

Dans le cadre de l'UE de parallélisme nous avons dû réaliser plusieurs programmes afin de réaliser une application de filtre de manière parallélisé. Nous avons d'abord réalisé une version séquentielle afin de servir de base pour les mesures de temps. Nous avons donc aussi dû réaliser une version parallèle avec openMP et deux versions avec CUDA dont une avec optimisation des conflits d'accès mémoire entre les threads.

1 Structure du projet et compilation

Concernant la structure du projet nous avons stocké les headers dans le dossier include et les fichiers c sont dans le dossier commonlib. Les fichiers contenant le code de chaque questions est contenu dans un dossier de la même lettre que celle de la question.

Nous utilisons un makefile principal ainsi que des sous-makefiles pour chaque question du projet afin de faire le lien entre les librairies et compiler le programme. Le makefile sans paramètre exécute la première règle c'est à dire compiler tous les programmes. Sinon il compile le programme choisi par l'utilisateur en paramètres (parmi "qa", ..., "qe").

2 Traitement de l'image

Pour réaliser un programme pour appliquer un filtre sur une image, nous avons utilisé une structure Kernel pour stocker les informations du filtre. Il contient une matrice d'entiers correspondant aux valeurs du filtre, la taille du filtre, mais aussi le facteur de division.

Nous appliquons le filtre sur tout les pixels de l'image, sauf ceux qui sont aux bords car on accéderait à des pixels en dehors de l'image en appliquant le filtre. Nous recopions juste les pixels qui sont aux bords.

3 OpenMP

Pour la version en openMP de l'application de filtre nous avons parallélisé les 2 boucles for qui parcours chaque pixel de l'image grâce à l'instruction "pragma omp parallel for". Les variables étant en début de block (ce n'est pas du C99), on ne peut pas faire de déclaration à la boucle. Nous avons aussi fait attention de bien déclarer les indices des boucles for en private pour éviter que les autres threads les modifient.

4 CUDA

Nous avons dû réaliser deux versions de notre programme, la première pour la question 3 afin de faire fonctionner le programme d'application de filtre avec CUDA. Pour la question

4 nous avons dû optimiser le programme de la question 3 le maximum possible à l'aide de l'utilisation de la mémoire partagée.

Pour le premier programme nous avons utilisé un thread par pixel de l'image, mais aussi le fait qu'un bloc s'occupe d'une ligne et il y a autant de blocs que de ligne. Nous avons utilisé cette répartition car nous utilisons une image qui ne comporte que 500 pixels en x ce qui convient pour la plupart des devices.

Pour le deuxième programme, nous avons d'abord stocké les informations concernant le filtre, nous avons aussi stocké les pixels de la même ligne dans la mémoire partagée. Nous avons réalisé une version où tous les pixels utilisés sont dans la mémoire partagée, la vitesse d'exécution du kernel diminue car on stocke trop de données dans la mémoire partagée, nous avons donc pas retenu cette deuxième version pour la question 5.

5 Les relevés de temps

Voici les mesures de temps que nous avons réalisés pour les différents exercices et pour différentes variables d'environnement.

Question	**Info mesuré**	**Temps mesuré**
Question 1 :	Séquentiel	103.564000ms
Question 2 :	OpenMP 1 threads	105.827985ms
Question 2 :	OpenMP 2 threads	53.158706ms
Question 2 :	OpenMP 4 threads	41.570381ms
Question 2 :	OpenMP 8 threads	35.063853ms
Question 2 :	OpenMP 16 threads	62.473234ms
Question 3 :	CUDA Simple	15.6 ms
Question 4 :	CUDA Opti	6.9 ms
Question 5 :	CUDA Simple * 100	1243.8 ms (12.4 ms)
Question 5 :	CUDA Opti * 100	598.3 ms (6 ms)
Question 6 :	CUDA Simple * 1000	12363.4 ms (12.4 ms)
Question 6 :	CUDA Opti * 1000	5621.8 ms (5.6 ms)

TABLE 1 – Mesure des temps d'exécution des différents programmes. GPU : GTX 1060 6GB ; CPU : i7-4770K 8T-4GHz ; 16Go DDR3 1600MHz Dual Channel