Laboratory for Parallel Programming
Prof. Dr. Felix Wolf

TECHNISCHE UNIVERSITÄT DARMSTADT

Zia Ul Huda, Sebastian Rinke

Systemnahe und Parallele Programmierung (WS 17/18)

Praktikum: CUDA

Die Lösungen müssen bis zum 13. Februar 2018, 13:30 Uhr, in Moodle submittiert werden. Anschließend müssen Sie Ihre Lösungen einem Tutor vorführen. Alle Programmieraufgaben müssen in C und mit CUDA auf dem Lichtenberg Cluster kompiliert und ausgeführt werden können. Alle Lösungen müssen zusammen in einer `tar` Datei eingereicht werden.

Machen Sie bitte in Ihrer Abgabe die Gruppennummer und die Namen der Mitglieder der Gruppe ersichtlich. Dies kann zum Beispiel direkt in der Lösung oder durch eine separate Textdatei als Teil Ihrer Abgabe erfolgen.

In diesem Praktikum geht es um die C CUDA Programmierung. Im Folgenden sollen Sie einen Algorithmus implementieren der die Kanten von Objekten in einem Bild detektiert. Das Problem der Kantendetektion findet breite Anwendung im Gebiet der Computer Vision. Die Eingabe für Ihren Algorithmus ist ein Farbbild im RGB Format. Das heißt, die Farbe von jedem Pixel wird durch sein Rot-, Grün- und Blauwert bestimmt. Wir beschreiben nun die einzelnen Schritte Ihres Algorithmus:

1) Farben in Graustufen Konvertieren. Es gibt verschiedene Methoden die drei RGB Farbintensitäten auf einen einzigen Grauwert abzubilden. Beispiele sind der Durchschnitt der drei Farbwerte, Entsättigung, und Dekomposition. In diesem Praktikum verwenden wir eine Technik basierend auf der Farbmetrik um die ursprünglich wahrgenommene Helligkeit des Farbbildes im Graustufenbild zu erhalten. Der Grauwert eines Pixels wird demnach wie folgt aus den RGB Farbwerten berechnet:

$$Grau = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B \tag{1}$$

2) Weichzeichnen. In diesem Schritt verringern wir den Kontrast des Graustufenbildes mit einem Gauss Filter um das Bildrauschen, d.h. Störungen die keinen Bezug zum Bildinhalt haben, zu verringern. Dazu nutzen wir eine 3×3 Konvolutionsmatrix. Konvolution (Faltung) bedeutet hier das gewichtete Addieren jedes Pixels zu seinen Nachbarpixeln, wobei die Gewichte in der Konvolutionsmatrix stehen.

3) Kantendetektion. Der letzte Schritt ist das Anwenden eines Sobel Filters um die Kanten im Graustufenbild zu erkennen. Der Filter nutzt die folgenden zwei Konvolutionsoperatoren:

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \tag{2}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \tag{3}$$

Dabei steht $*$ für das Anwenden der Konvolutionsmatrix auf jedes Pixel im Bild A (unser Bild nach dem Weichzeichnen). Der neue Grauwert eines Pixels berechnet sich als:

$$G = \sqrt{G_x^2 + G_y^2} \tag{4}$$

Das finale Bild zeigt schließlich die Kanten vom ursprünglichen Eingabebild.

Von den oben beschriebenen Schritten befindet sich eine sequentielle Implementierung in der Datei

`template.cu`. In dieser Datei implementieren Sie bitte auch den Algorithmus mit CUDA. In Ihrer CUDA Implementierung soll in jedem Kernel jeder Thread ein Pixel verarbeiten. Im Folgenden beschreiben wir die einzelnen Aufgaben für die Implementierung.

Hinweise: Zum Kompilieren oder Ausführen eines CUDA Programms auf dem Lichtenberg Cluster müssen sie das CUDA Modul laden: `module add cuda`. Kompilieren können sie bereits auf den Login Knoten. Zum Ausführen des CUDA Progamms muss jedoch ein Batch Job submittiert werden. Nutzen sie dafür die Datei `batch_job.sh` als Beispiel.

Aufgabe 1

(2 Punkte) Berechnen Sie die Dimensionen des *Grids* in Abhängigkeit von `BLOCK_SIZE` und den Dimensionen des Eingabebildes. Das Grid dient als Eingabe für alle Kernelaufrufe.

Aufgabe 2

(4 Punkte) Allokieren Sie zwei Puffer im Speicher auf dem *Device* um Bilder während der Verarbeitung zu speichern. Initialisieren Sie jedes Byte der Puffer mit dem Wert `0`.

Aufgabe 3

(4 Punkte) Allokieren Sie einen Puffer für das Eingabebild auf dem Device und kopieren Sie das Eingabebild in diesen Puffer.

Aufgabe 4

(11 Punkte) Implementieren Sie den Kernel `cuda_grayscale` der das farbige Eingabebild in ein Graustufenbild konvertiert. Rufen Sie den Kernel entsprechend im Programm auf.

Aufgabe 5

(4 Punkte) Die Funktion `cuda_applyFilter` wendet eine 3×3 Konvolutionsmatrix auf ein Pixel an und wird von den Kernels in den nächsten Aufgaben verwendet. Implementieren Sie diese Funktion.

Aufgabe 6

(11 Punkte) Implementieren Sie den Kernel `cuda_gaussian` um das Graustufenbild mit einem Gauss Filter weich zu zeichnen. Rufen Sie den Kernel entsprechend im Programm auf.

Aufgabe 7

(11 Punkte) Implementieren Sie den Kernel `cuda_sobel` der einen Sobel Filter auf das weichgezeichnete Graustufenbild anwendet. Rufen Sie den Kernel entsprechend im Programm auf.

Aufgabe 8

(6 Punkte) Kopieren Sie jedes Bild was durch die Anwendung eines Kernels entsteht zurück in den Hauptspeicher und speichern Sie es in einer Datei damit Sie die Korrektheit des Kernels überprüfen können.

Aufgabe 9

(2 Punkte) Geben Sie alle allokierten Puffer im Device Memory wieder frei.