Heterogeneous Computing Abschlussprojekt

Einführung

Für mein Abschlussprojekt habe ich mich dafür entschieden an das zweite Aufgabenblatt anzuknüpfen und das Potenzial von GPU basierten Anwendungen anhand eines anderen Beispiels zu untersuchen. Der Maßgebliche Grund dafür war, dass ich persönlich große Probleme bei Setup von CUDA und den Kompatibilitäten verschiedener Libraries hatte und ich am Ende mehr Zeit damit verbracht habe, als mit den eigentlichen Aufgaben, was mich persönlich sehr gestört hat. Weiterhin waren meine Ergebnisse bezüglich der zweiten Aufgabe etwas enttäuschend für mich, da ich gehofft hatte maßgebliche Performance-Boosts durch die Verwendung der leistungsstarken GPUs zu finden, diese jedoch überhaupt nicht auftraten. Daher habe ich für mein Abschlussprojekt untersucht ob die Ergebnisse in dem Feld **Image Processing** anders aussehen. Im Rahmen dessen habe ich verschieden Image Filter in Python mit herkömmlichen Ansätzen und mit Cuda basierten Ansätzen implementiert und miteinander verglichen.

Implementierung

Die Implementierung meines Projekts erfuhr in Python. Implemtiert habe ich die Image Filer **Gaussian Blur**, **Sobel Edge Detection**, **Image Sharpening** und **Non-Local Means (NLM) Denoising**. Die GPU basierten Implementierungen verwenden die Python Library CuPy as CUDA-Schnittstelle, während die CPU Implementierungen aus NumPy und SciPy bestehen.

Der Grundsätzliche Ablauf des Codes beginnt mit einlesen eines Bildes. Anschließend werden einige Preprocessing-Schritte durchgeführt. Hierbei wird das Bild in Grayscale umgewandelt und neu Skaliert. Anschließend wird ein Filter auf das Bild angewendet (Oder auch mehrere, da manche Filter nur richtig funktionieren nachdem Gaussian Blur angewendet wurde). Die Filter werden für jedes Bild zweimal angewendet, einmal für jede der Beiden Implementierungen (GPU und CPU). Anschließend werden die Zeiten für beide Funktionen verglichen.

In der eigentlichen Main-Funktion wird dieser Prozess mehrmals für verschiedene Filter und verschiedene Skalierungen des Bildes wiederholt. Weiterhin wird Cuda zunächst „Aufgewärmt“, um den einmaligen Initialisierungsoverhead nicht mit aufzuzeichnen. Um die Daten zu speichern wird ein Pandas-Dataframe verwendet, der anschließend als Excel-Datei abgespeichert wird.

Resultate

Gaussian Blur

Scale Factor	GPU Approach	CPU Approach
0.5	0.001000643	0.032006502
1	0	0.173038721
2	0	0.738173962
4	0	3.466785908

Sobel Edge Detection

Scale Factor	GPU Approach	CPU Approach
0.5	0.001000166	0.297075033
1	0	1.145265102
2	0.000999928	4.521013737
4	0.001000166	19.28532457

Image Sharpening

Scale Factor	GPU Approach	CPU Approach
0.5	0.001000404	0.091011524
1	0.001000166	0.301066875
2	0.001000166	1.322297335
4	0.003000498	6.027351856

Non-Local Means Denoising

Scale Factor	GPU Approach	CPU Approach
0.125	447.5975692	33.77794003

Auswertung

Getestet wurde mit einem Bild der Größe 4096 x 4096 (Bei Faktor = 1).

Die Ergebnisse zeigen deutliche Unterschiede in der Laufzeit zwischen den verschiedenen Implementation für alle Filter. Wie erwartet zeichnen sich die GPU-Ansätze mit deutlich besseren Zeiten aus als die CPU-Ansätze für die Gaussian Blur, Edge Detection und Image Sharpening Filter. Hierbei ist vor allem interessant, dass die Zeit für die GPU quasi unabhängig von der Skalierung des Bildes ist. Die Laufzeit ist lediglich von dem initialen Overhead abhängig und würde vermutlich auch bei deutlich größeren Bildern ähnliche Zeiten vorweisen. Größere Bilder konnten bei mir jedoch leider aufgrund von Memory-Constraints nicht getestet werden. Die CPU-Ansätze hingegen zeigen deutliche Verlangsamungen um mehrere Faktoren auf bei größeren Skalar-Faktoren. Gerade bei dem Edge Detection Filter wird der Unterschied zwischen GPU- und CPU-Ansatz deutlich, wo bei einem Faktor von 4 des Bildes die CPU 19s brauchte, während die GPU innerhalb eines tausendstel einer Sekunde fertig wurde. Damit ist die GPU um einen Faktor von 19000x schneller gewesen für diesen Test.

Ein anderes Ergebnis wurde hingegen in meiner Implementierung von NLM Denoising vorgefunden. In diesem komplexeren Filter schnitt die CPU deutlich besser ab als die GPU.

Dank der deutlich höheren benötigten Rechenleistung die für diesen Filter nötig war, habe ich hierfür auch leider wenigere Tests machen können. Meine Vermutung bezüglich dieser Ergebnisse ist, dass es sich hier vermutlich eher um ein Hardwareproblem (zu wenig Memory)oder eine mangelnde Implementierung meinerseits handelt.

Insgesamt zeigen die Ergebnisse jedoch einen klaren Vorteil bei der Verwendung von GPU-basierten Implementierungen für Image Processing Filter.

Anmerkung: Die Bilder die in den Tests produziert wurden habe ich nicht mit hochgeladen (Außer das Original), da diese Teils zu große Dateigrößen hatten und auch aus meiner Sicht nicht wichtig waren (Außer zum Testen meinerseits ob die Filter funktionierten).