Heterogeneous Computing 2

Vorwort

Chronologie:
Die ursprüngliche Idee war, den Cooley-Tukey FFT auf CPU Sequentiell, parallel und auf der GPU umzusetzen. Die CPU-Lösungen haben auch auf Anhieb funktioniert (Dank an ChatGPT), allerdings war die Performance Verbesserung dabei nur etwa ein Faktor von 4 (bei 24 Kernen).

Anhand des Algorithmus bin ich nicht dahintergekommen, was diese nur mäßige Verbesserung bedingt, also habe ich mich zunächst der Implementierung des FFT mit CUDA gewidmet. Da ich mein Projekt in Java angelegt habe, hatte ich zunächst versucht JCUDA, Java Bindings von CUDA zu benutzen. Mein erstes Learning ist nicht JCUDA zu benutzen. Nachdem ich mich mehrere Tage mit JCUDA herumgeschlagen habe, habe ich zu C++ gewechselt. Die CPU-Implementierungen sind weiterhin in Java.

Da GPUs bestimmte Konzepte wie z.B. Rekursion nicht unterstützen hatte ich eine Iterative Variante des Cooley-Tukey Algorithmus geplant. Hier bin ich aber auf Fehler bezüglich gleichzeitiger Read/Write Operationen gestoßen. Ich habe mit verschiedenen Varianten experimentiert, bis hin zum starten eines einzigen Threads pro Block und einen Block pro Subtask in der Fourier-Analyse, sodass die relevanten Daten in die Block-Shared Memory gespeichert werden können (und dort schneller gelesen/geschrieben werden). Allerdings bestand hier immer noch das Problem, die Shared Memories der Blöcke am Ende zu Synchronisieren und zu einem Ergebnis zu verrechnen.

Nachdem ich so mehrere Tage wieder zu keinem Ergebnis gekommen bin, habe ich ich entschieden einen einfacheren Ansatz zu verfolgen. Also habe ich den grundlegenden Algorithmus für die Diskrete Fourier Transformation implementiert (der ja sehr langsam ist), und den Fokus darauf gelegt mehr die Performanceverbesserung zu beobachten und weniger die Korrektheit der DFT Ergebnisse. Bis heute sind die Ergebnisse der Tatsächlichen Analyse nicht wirklich aufschlussreich über das Signal (ich vermute wegen Leakage, da ich keine Windowing Funktion verwende). Aber: der gleiche Algorithmus läuft in allen Varianten, und die Performanceunterschiede entsprechen eher den erwarteten Werten.

1. CPU Implementierung

Den DFT habe ich entsprechend der Formel

$$\hat{a}_k = \sum_{j=0}^{N-1} \mathrm{e}^{-2\pi \mathrm{i} \cdot \frac{jk}{N}} \cdot a_j \quad \text{für} \quad k = 0, \ldots, N-1$$

Implementiert. Für die Blockweise Fourieranalyse muss für jeden Block jeder Frequenzbereich durch die Summe ausgerechnet werden. Dies resultiert in drei geschachtelten Schleifen und einer langsamen Ausführungsdauer.

Die **parallele** CPU Variante berechnet die Blöcke möglichst gleichzeitig. Es wird die Anzahl der vorhandenen Kerne ermittelt und für jeden Block ein Task an einen ExecutorService übergeben, der diese Tasks auf die Kerne verteilt. Diese Task beinhalten dann nur noch zwei geschachtelte Schleifen und geben als Ergebnis eine Liste mit den Amplituden ihres Blocks zurück, welche gesammelt, zusammengerechnet und aggregiert werden müssen. Das beinhaltet die Berechnung des Durchschnitts über alle Blöcke als auch eine Normalisierung der Resultate.

2. CUDA

Das CUDA-Programm ist eine einzige ausführbare Datei und enthält das Einlesen der Daten, den DFT und das Speichern der Ergebnisse als .txt zur Überprüfung. ChatGPT konnte bei Dingen wie dem Einlesen von WAV-Dateien gut helfen, bei der Umsetzung der Fourier-Analyse war es allerdings kaum eine Hilfe. Der DFT-Algorithmus ist der gleiche wie in den CPU-basierten Lösungen, damit die Ergebnisse sinnvoll verglichen werden können. Es wird für jeden Block, für den die Fourier Analyse durchgeführt wird, ein Thread angelegt. Jeder Thread ermittelt, für welchen Bereich des Inputfeldes er zuständig ist, führt den DFT durch und speichert die Ergebnisse per *atomicAdd* im Ouput-Feld. Dafür braucht es einige Hilfsfunktionen für Rechenoperationen mit komplexen Zahlen. Außerdem gibt es einen AvgKernel, der nur alle Durchschnitte im Output Array gleichzeitig berechnet. Wenn schon denn schon.

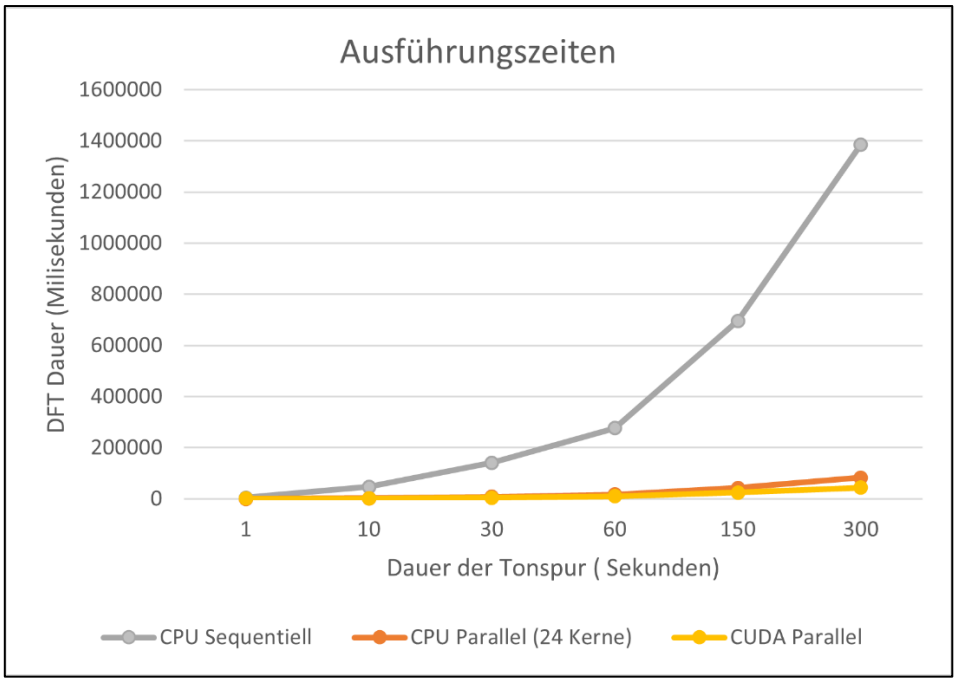
3. Sonstiges

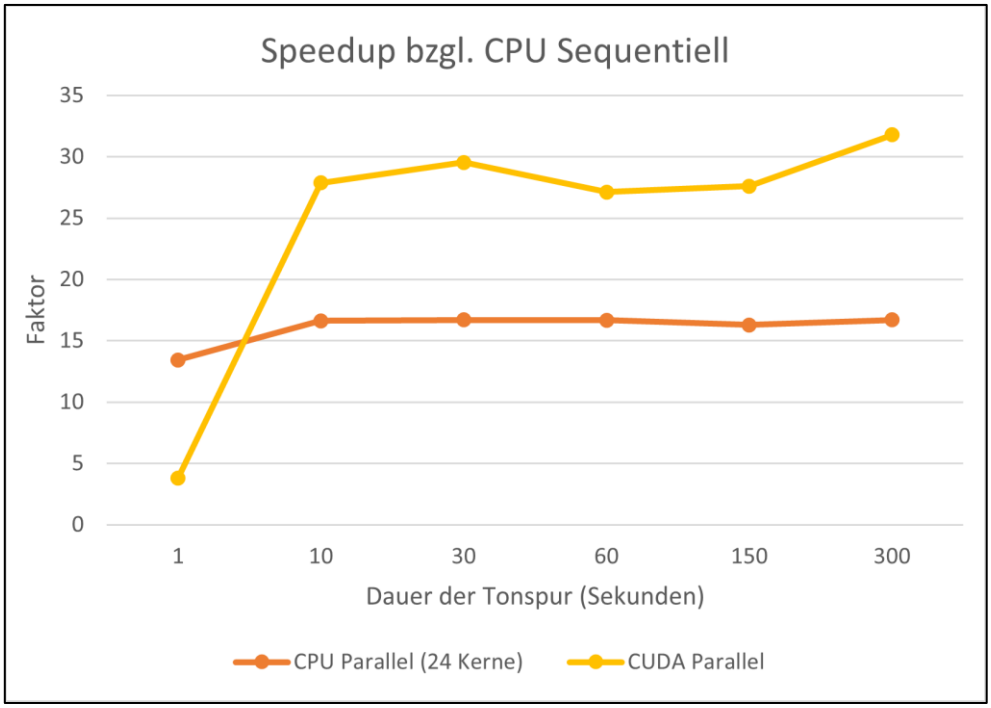
WavFileReader:
Klasse mit Funktionen für den Umgang mit den Soundfiles. Es gibt sowohl die Möglichkeit, WAV-Dateien zu erstellen, Dateien zu lesen und als Arrays komplexer Zahlen darzustellen sowie Ergebnis-Arrays als .txt. Datei zu speichern. Dies dient zur Überprüfung der Ergebnisse.

Plotter.py:
Einfaches Python Skript dass die Ergebnisse des DFT Darstellt. Das ist vor allem dazu da, um zu überprüfen ob alle Implementierungen das gleiche Ergebnis liefern. Die Ergebnisse der Fourier Analyse selbst sind wie gesagt nicht wirklich aufschlussreich, aber über die unterschiedlichen Implementierungen konsistent und in ihren Ausführungszeiten vergleichbar.

Experimente

Ich habe die Experimente mit monotonen WAV-Dateien unterschiedlicher Länge (1sec, 10sec, 30sec, 60sec, 150sec, 300sec) durchgeführt. Da der gleiche Algorithmus in allen Implementierungen läuft sind die Geschwindigkeitsunterschiede sehr vergleichbar.





Die CPU-Parallele Lösung mit 24 Kernen erreicht einen Speedup von einem Faktor ~15. Die CUDA-Implementierung erreicht einen Faktor von ~30, wobei dieser Faktor besser wird wenn die SoundDatei länger ist. Eine Anleitung zur Ausführung ist in der README.

Eine persönliche Anmerkung: Vielen Dank ihre Kulanz, die Sie mir bei dieser Aufgabe gezeigt haben, das weiß ich sehr zu schätzen. Die Ergebnisse der Fourier Analyse selber finde ich nicht zufriedenstellend, wohl aber die Performance-Unterschiede, die in etwa meinen Erwartungen entsprechen. Ich kann zumindest behaupten, dass ich mich sehr Intensiv mit CUDA und den Eigenheiten der GPU-Programmierung beschäftigt habe.