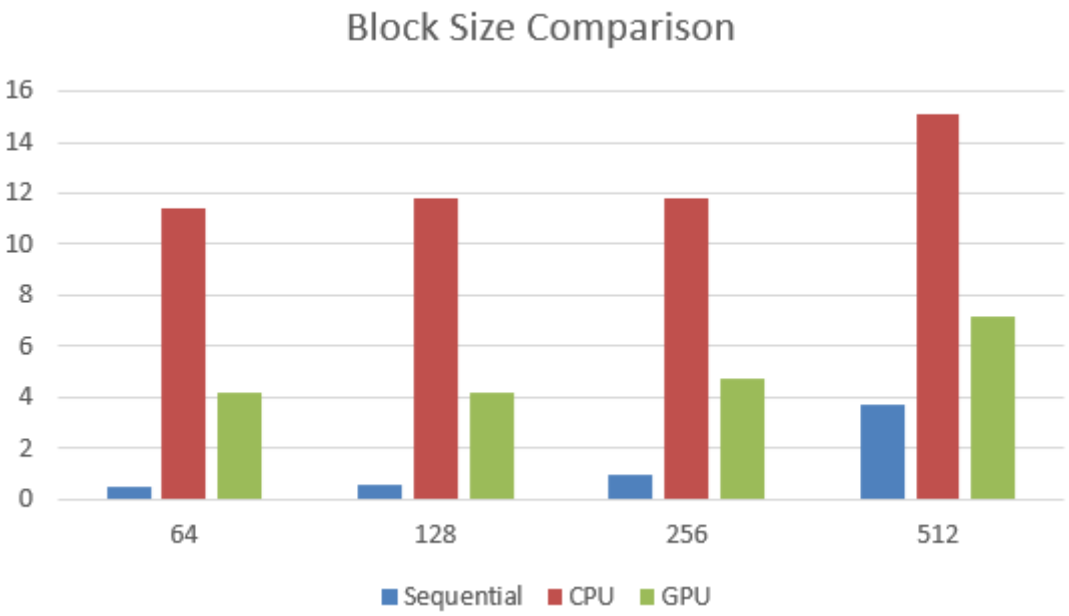
Experimente:

Block Size:

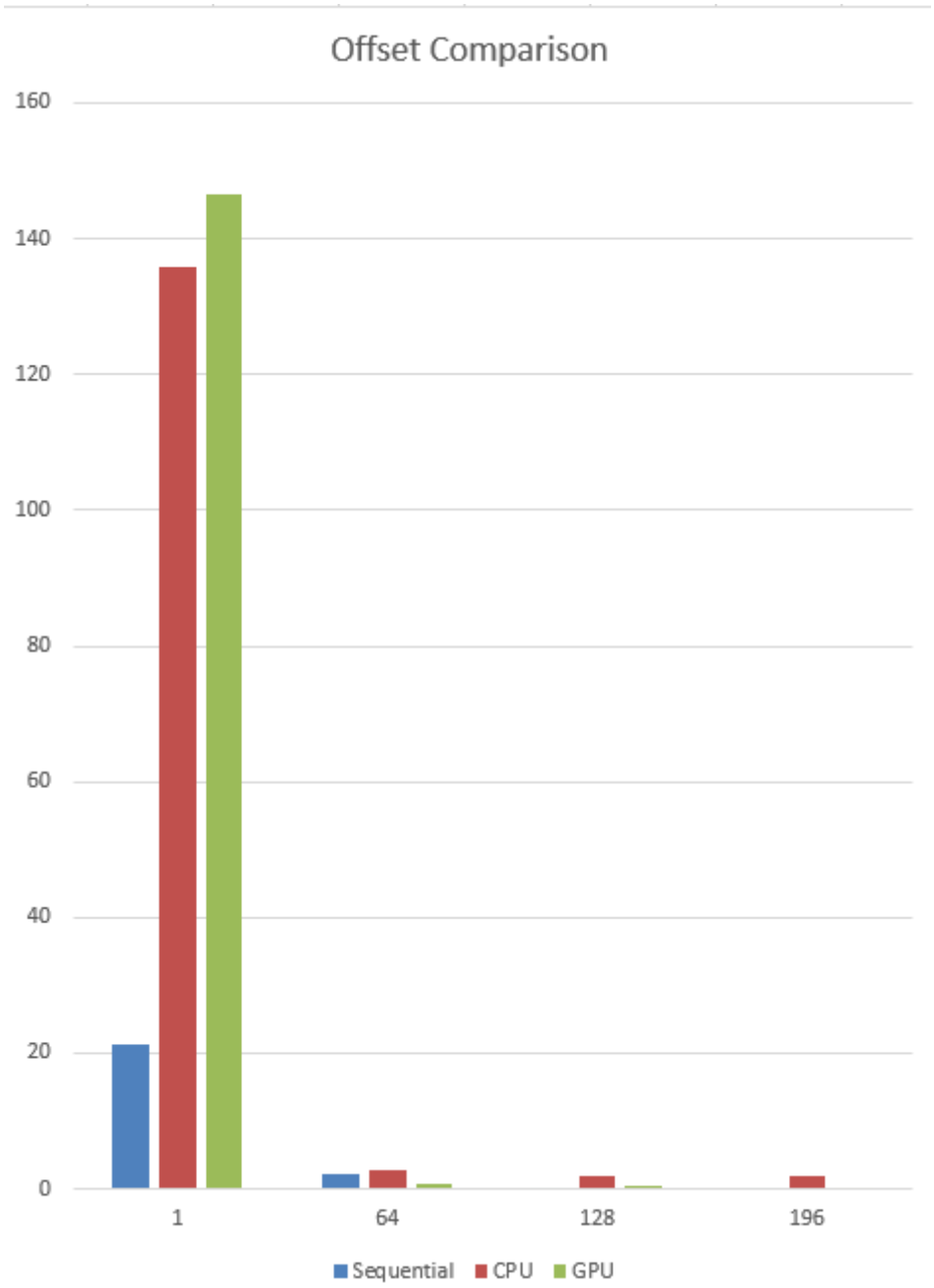
(Duration: 30s, Offset: 1, Cores 4/16)



Für jede Block-Größe sind die parallelen Ansätze deutlich langsamer als der sequentielle Ansatz, allerdings wird der Unterschied kleiner bei größerer Block-Größe. Der GPU CUDA-Ansatz funktioniert für die verwendeten Werte deutlich besser.

Offset:

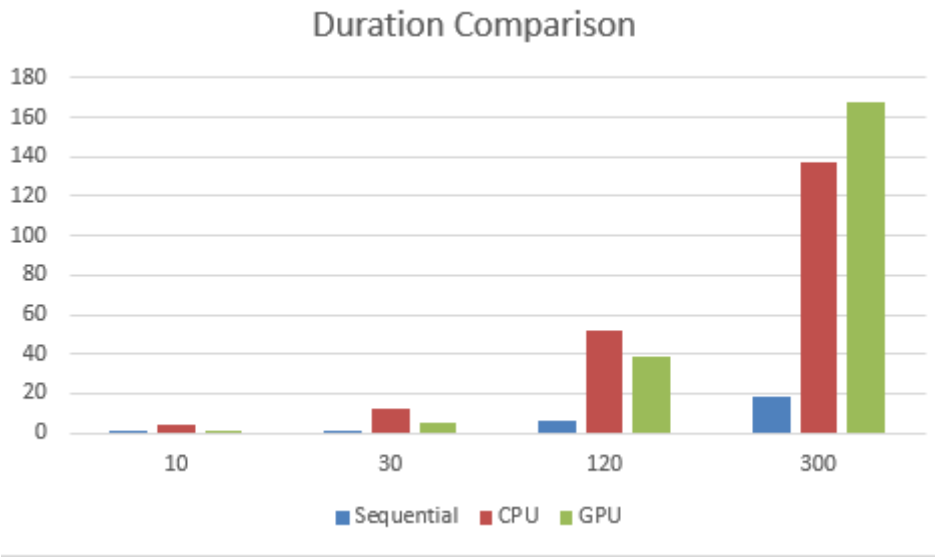
(Block Size: 256, Duration: 300s, Cores 4/16)



Anders als die anderen Experimente wurden diese mit einer größeren Datei (300s) durchgeführt, um den Unterschied besser darstellen zu können. Wie bei den Experimenten für Block Size, ist sequentielle Ansatz deutlich schneller. Anzumerken ist, dass das Experiment für einen Offset von 1 am längsten gedauert hat und der CPU-Ansatz hier performanter war. Dies ist vermutlich auf die begrenzte Memory meines Systems zurückzuführen.

Duration:

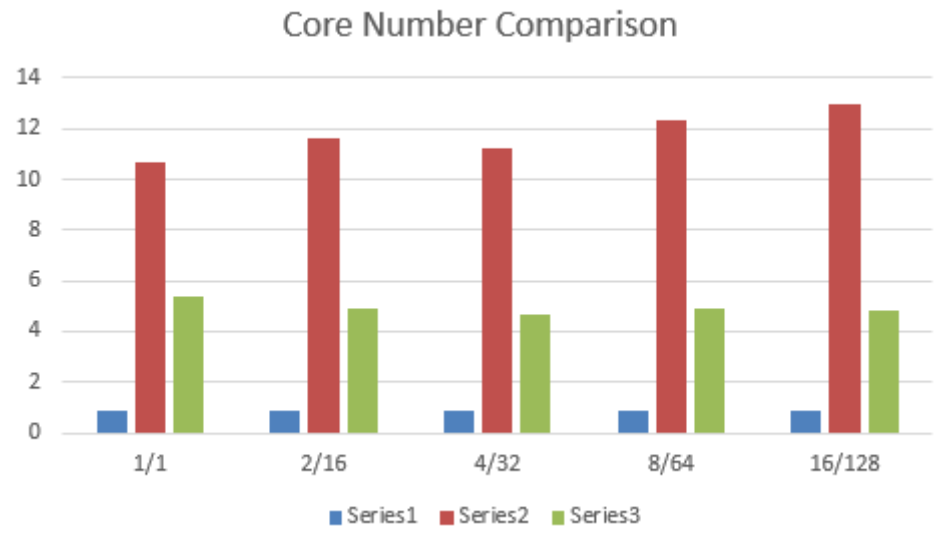
(Block Size: 256, Offset: 1, Cores 4/16)



Die Experimente für verschiedene Dauer zeigen wie bereits bei den Experimenten zu Offset, dass größere Audio-Dateien zu einer besseren Performance des CPU-Ansatzes führen, relativ zum GPU-Ansatz. Jedoch bleibt der sequentielle Ansatz der Beste.

Cores:

(Block Size: 256, Offset: 1, Duration: 30s)



Trotz unterschiedlicher Anzahl an verwendeten Kernen, ändert sich die Rechenzeit kaum für die CPU- und GPU-Ansätze. Dies lässt darauf schließen, dass der Overhead einen maßgeblichen Zeitaufwand beinhaltet und die eigentlichen parallelen Berechnungen sehr schnell sind, jedoch das Programm insgesamt verlangsamen.

Finale Anmerkungen:

Keiner meiner parallelen Implementationen performt besser als ein einfacher sequentieller Ansatz. Dies liegt vermutlich am Overhead, der Verhältnismäßig so viel Zeit in Anspruch nimmt,

dass die Ersparnisse durch die parallele Rechnung der FFT irrelevant sind. Grund hierfür könnte sein, dass die Ausgaben meiner Implementation erst nach einem Zusammensetzen der einzelnen FFT-Teile erfolgt. Möglicherweise würde eine direkte Ausgabe in den parallelisierten Abschnitten zu einer besseren Performanz führen. Da allerdings auch mit einer Kern-Anzahl von 1 die beiden Ansätze eine deutlich längere Zeit brauchen, liegt der Overhead vermutlich eher in der Initialisierung. Andere Gründe für die verschiedenen Zeiten könnte z.B. die Limitation von Python als Programmiersprache sein. Es wäre interessant die Ergebnisse mit einem in z.B. Java umgeschriebenen Code zu vergleichen.