Heterogeneous Computing Übung 2

Aufgabe 1:

In dieser Aufgabe wird eine WAV-Datei eingelesen, in Blöcke zerlegt, verschoben und die Frequenzanteile mittels FFT berechnet. Der mittlere Amplitudenwert jeder Frequenz wird ausgegeben, sofern dieser einen bestimmten Schwellwert überschreitet.

- **Python:** Hier wurde die Bibliothek numpy verwendet, um die FFT durchzuführen.
- **C:** Für die FFT-Berechnungen wurden sowohl FFTW als auch KISS FFT eingesetzt. FFTW zeigte im Multithreading nicht die gewünschte Performance, daher wurde KISS FFT ebenfalls getestet.

Aufgabe 2:

Diese Aufgabe beinhaltet die Generierung von WAV-Testdateien mit unterschiedlichen Signalformen zur Validierung der anderen Aufgaben. Der WAV-Header wird geschrieben und numpy wird zur Signalerzeugung verwendet.

Aufgabe 3:

In dieser Aufgabe wurden die FFT-Berechnungen parallelisiert, um die Leistung zu steigern.

- **Python:** Trotz der Einschränkung durch das „Fake“ Multithreading von Python, zeigte sich eine Performance-Steigerung von 30 %.
- **C:** Aufgrund der begrenzten Performance von Python-Multithreading, wurde die Implementierung in C vorgenommen. Dabei zeigte FFTW im Multithreading keine Geschwindigkeitsvorteile, sodass KISS FFT getestet wurde.
    - **KISS FFT:** Diese Bibliothek erwies sich als schneller und lieferte eine Geschwindigkeitssteigerung um den Faktor K/2, wobei K die Anzahl der CPU-Kerne ist. Bei 4 Kernen ist die Threaded KISS FFT also doppelt so schnell.
    - **Single-Process FFTW:** Diese Methode war jedoch schneller als alle anderen getesteten Methoden.

Aufgabe 4:

Für diese Aufgabe wurde in Python die Bibliothek Reikna verwendet, um ohne Kernel-Code die FFT auf der GPU durchzuführen. Obwohl dies die Korrektheit der Ergebnisse sicherstellte, war die Performance im Vergleich zu anderen Methoden deutlich langsamer. Deshalb wurde auch eine Implementierung in C getestet. Der Kernel-Code wurde von einer KI generiert und war nicht vollständig korrekt, aber deterministisch mit Abweichungen. Laut ClaudeAI handelt es sich um einen Cooley-Tukey-Algorithmus.

Testumgebung:

Zur Evaluierung der verschiedenen Ansätze wurden WAV-Dateien mit einer Dauer von 1 Stunde generiert. Die Blockgröße betrug 512, und der Versatz wurde jeweils verdoppelt (1, 2, 4, 8 …). Diese Einstellungen reichten aus, um auch größere WAV-Dateien zu simulieren und umfassend zu testen.