HC Übung 2

Aufgabe 1:

Umgesetzt durch die Funktion **sequential.**

Diese besitzt die vier geforderten Parameter:

- **wav_path**
  - Der Link zur .wav-Datei
- **block_size**
  - Die Größe der Blocks
- **offset**
  - Der Versatz zwischen den Blocks
- **threshold**
  - Der Schwellwert für die Ausgabe der Amplitudenmittelwerte

Zunächst werden in der Funktion die Eingabeparameter **block_size** und **offset** überprüft und ob diese den Anforderungen entsprechen, ansonsten wird eine Fehlermeldung ausgegeben.

Anschließen wird die .wav-Datei ausgelesen. Falls diese Stereo ist, konvertieren wir diese zu Mono. Als nächstes werden die Daten der .wav-Datei entsprechend der **block_size** und des **offset** in Blöcke aufgeteilt. Für jeden Block wird die FFT mithilfe der Funktion **perform_fft_on_block** ausgeführt. Anschließend werden die Ergebnisse der Blöcke zusammengeführt und die Amplitudenmittelwerte berechnet. Im letzten Schritt werden die Frequenzen und die Amplitudenmittelwerte die größer als der Threshold sind ausgegeben.

Aufgabe 2:

Umgesetzt durch die Funktion **generate_wav_file.**

Diese Funktion besitzt fünf Parameter:

- **filename**
  - Name der zu erstellenden .wav-Datei
- **frequency**
  - Die Frequenz der Sinus-Welle
- **amplitude**
  - Die Amplitude der Sinus-Welle
- **duration**
  - Die Dauer des Audio-Signals
- **sample_rate**
  - Die Sample-Rate in Hertz.

Zunächst wird eine Zeitachse für das Signal erstellt. Anschließend wird anhand der angegebenen Amplitude und Frequenz und der Zeitachse ein Signal erstellt. Im nächsten Schritt wird sichergestellt, dass das Signal sich zwischen dem Range -1 und 1 befindet. Nun wird das Signal in das 16-Bit PCM Format konvertiert bevor es auf eine .wav-Datei gespeichert wird.

Aufgabe 3:

Diese Aufgabe wurde durch die Funktion **parallel_cpu** mit der Hilfsfunktion **process_blocks** umgesetzt. Die Funktion besitzt dieselben Parameter wie **sequential** aus Aufgabe 1, mit dem zusätzlichen Parameter **num_cores**, der zu benutztende Anzahl an Kernen festlegt:

- **wav_path**
  - Der Link zur .wav-Datei
- **block_size**
  - Die Größe der Blocks
- **offset**
  - Der Versatz zwischen den Blocks
- **threshold**
  - Der Schwellwert für die Ausgabe der Amplitudenmittelwerte
- **num_cores**
  - Anzahl zu den verwendenden Kernen

Der Großteil der Funktion funktioniert ähnlich wie die Funktion **sequential.** Der wesentliche Unterschied besteht darin, dass die Daten zunächst in eine Anzahl an Chunks aufgeteilt werden (gegeben durch die Anzahl an Kernen definiert durch **num_cores**). Diese Chunks werden mithilfe des ProcessPoolExecutor auf eine Anzahl Unterprozesse aufgeteilt, die die FFT-Funktion in der Hilfsfunktion **perform_fft_on_block_parallel** ausführen. Anschließend werden die Ergebnisse zusammengeführt. Der Rest der Funktion ist identisch zu Aufgabe 1.

Aufgabe 4:

Diese Aufgabe wurde durch die Funktion **parallel_gpu** umgesetzt. Die Parameter sind dieselben wie bei Aufgabe 3, nur das statt **num_cores**, die Variable **num_gpus** heißt.

- **wav_path**
  - Der Link zur .wav-Datei
- **block_size**
  - Die Größe der Blocks
- **offset**
  - Der Versatz zwischen den Blocks
- **threshold**
  - Der Schwellwert für die Ausgabe der Amplitudenmittelwerte
- **num_gpus**
  - Anzahl zu den verwendenden Kernen

Der Hauptunterschied zur Aufgabe 3, ist das die Berechnungen der FFT verteilt über mehrere GPU Kerne erfolgt. Dies wird mithilfe der library Cupy umgesetzt, die FFT mit Cuda unterstützt. Für die FFT wird eine Hilfsfunktion **process_blocks_on_gpu** verwendet, die mithilfe von CUDA Streams die FFT für jeden Block durchführt.