Heterogeneous Computing

Portfolio

Aufgabensammlung

Universität Trier
FB IV - Informatikwissenschaften
Professur für Systemsoftware und Verteilte Systeme

Semester: SoSe 2024

Name und Matrikelnummer:

Simon Szulik, 1474315

Inhaltsverzeichnis

1. Fourieranalyse der geheimnisvollen Wellenlängen

Die Fourieranalyse ist eine mathematische Methode, die es ermöglicht, komplexe periodische Funktionen in eine Reihe von einfachen sinusförmigen Wellen zu zerlegen. Sie wird häufig in der Signalverarbeitung und der Analyse periodischer Phänomene wie Schwingungen und Wellen angewendet, um deren Bestandteile und Frequenzanteile zu untersuchen. Durch die Anwendung der Fourieranalyse kann man komplexe Signale in ihre Grundbestandteile zerlegen und dadurch besser verstehen.

1.1 Implementierung

Für die Implementierung wurde Python verwendet, sowie die folgenden Bibliotheken:

- **scipy**: Zum Einlesen der Audio-Datei (.wav) & dem Anwendungen einer schnellen Fourier-Analyse.
- **numpy**: Für numerische Operationen und die Berechnung der einzelnen Frequenzen.
- **matplotlib**: Als visuelle Darstellungsmöglichkeit der Daten und zum Erstellen von Plots.
- **argparse**: Zum Parsen der Kommandozeilenargumente, um die Eingabeparameter wie Dateiname, Blockgröße und Ausgabetyp zu bestimmen.
- **memory profiler**: Zur Analyse und Auswertung des Speicherbedarfs.

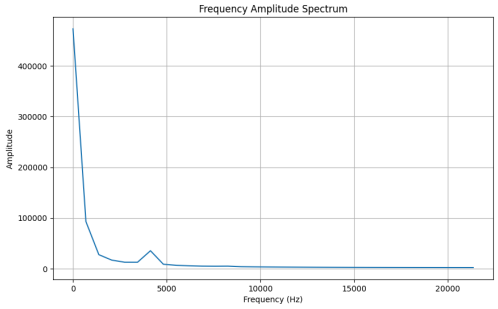
Im Beispiel 1.1 ist eine typische Ausführung des Skriptes zu sehen. Mit hilfe der Argumente lässt sich sowohl die Blockgröße, als auch die Art des Diagramms einfach bestimmen.

```
1  py.exe Fourier.py audio.wav —b 512 —o f —p 1
```

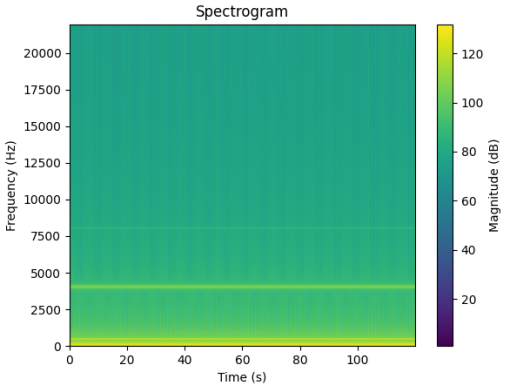
Beispiel 1.1: Skriptausführung in der Knsole

1.1.1 Ergebnisse

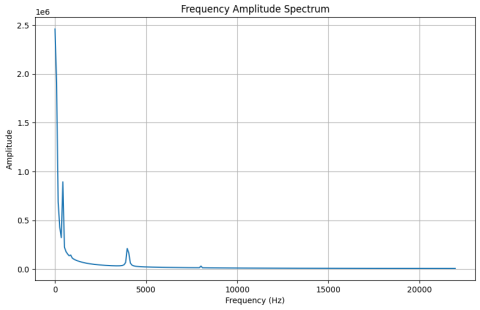
In diesem Abschnitt betrachten wir ein paar Beispiel Plots der Analyse. Bezüglich der Blockgröße gibt es einige Charakteristika die zu beachten sind. Behandelt man Signale mit schnellen und häufigen Änderungen wie Sprachaufnahmen, so können kleinere Blöcke (128-512) besser sein. Signale mit stabileren Frequenzen können Problemlos mit größeren Blocken analysiert werden. Um eine möglichst breite Spanne von Fällen abzudecken, befindet sich in Abbildung 1.2 eine Ansammlung von verschiedenen Plots der zu analysierenden Datei.



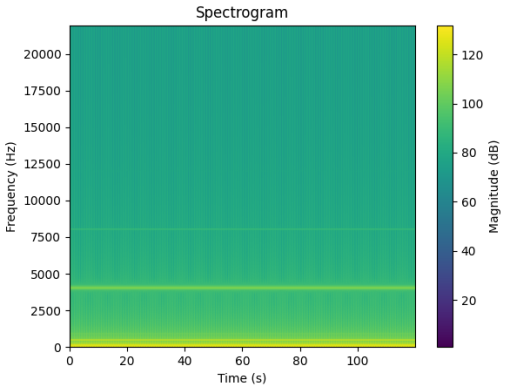
(a) Frequenz-Plot mit Blockgröße 64



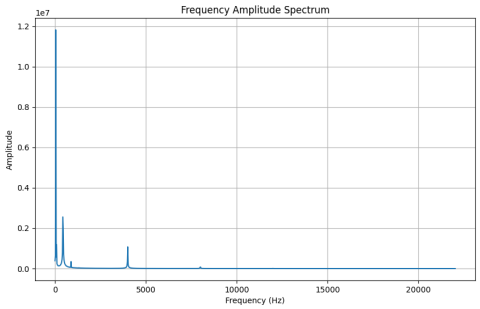
(b) Spektrogramm mit Blockgröße 64



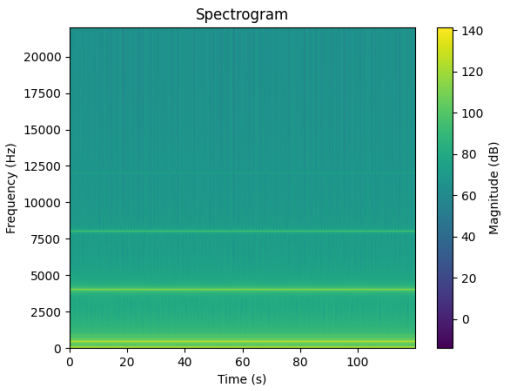
(c) Frequenz-Plot mit Blockgröße 512



(d) Spektrogramm mit Blockgröße 512



(e) Frequenz-Plot mit Blockgröße 2048



(f) Spektrogramm mit Blockgröße 2048

Abbildung 1.1: Beispielergebnisse

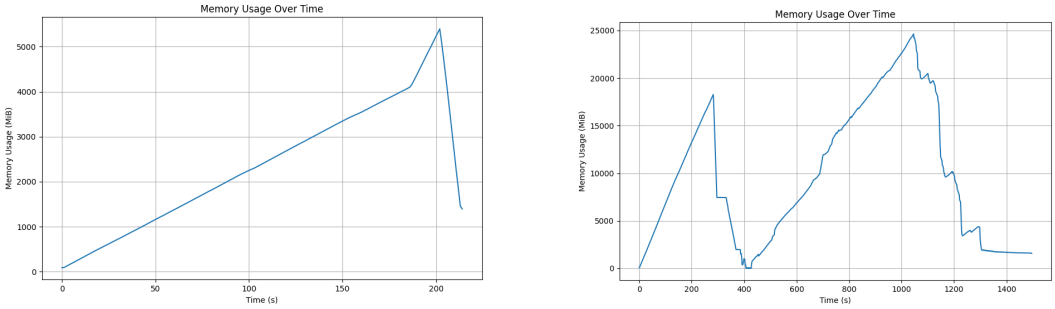
1.2 Speicherplatzbedarf

Für eine erste Speicheranalyse wurden die folgenden Spezifikationen unter einem Windows System verwendet:

Attribut	**Details**
Prozessor	AMD Ryzen 5 7500F
Architektur	ZEN 4
Kerne	6
Threads	12
Basis-Taktfrequenz	4.8 GHz
L3-Cache	32 MB
Fertigungstechnologie	65 Watt
Speicherunterstützung	DDR5
PCIe-Unterstützung	PCIe 5.0
Sockel	AM5

Tabelle 1.1: Spezifikationen des AMD Ryzen 5 7500F

Wie erwartet und in Abbildung 1.2 zu erkennen, erhöht sich mit steigender Blockgröße auch der benötigte RAM-Verbrauch. Dies geschieht aufgrund der immer größer werdenden Vektoren, die benötigt werden, um die Daten zu speichern. Bei 32 GB verfügbarem RAM ist eine Blockgröße von 512 das Maximum, bevor der Prozess aufgrund des Speicherplatzbedarfs abbricht.



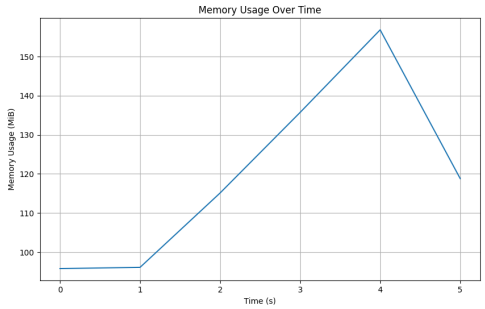
(a) Speicherbedarf mit Blockgröße 64 (b) Speicherbedarf mit Blockgröße 512

Abbildung 1.2: Speicherplatzbedarf mit Schrittweite 1

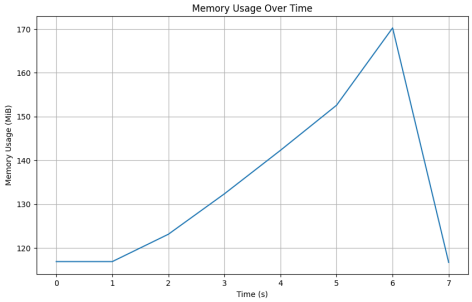
Um dennoch Zugriff zu größeren Blockgrößen und deren Speicherverbauch zu haben, gibt es einige Lösungsansätze. Eine effektive Methode zur Reduktion des Speicherverbrauchs besteht darin, die Präzision der Daten zu verringern. Audiodaten werden oft in 64-Bit-Gleitkommazahlen (float64) gespeichert, die jedoch mehr Speicherplatz beanspruchen als notwendig. Wenn die Präzision ausreicht, kann die Konvertierung der Daten in 32-Bit-Gleitkommazahlen (float32) den Speicherbedarf halbieren. Dies kann durch einfache Typumwandlung erreicht werden und führt zu einer signifikanten Reduktion des Speicherverbrauchs, ohne die Qualität der Analyse erheblich zu beeinträchtigen.
Ein weiterer Ansatz zur Reduktion des Speicherverbrauchs besteht darin, nur die notwendigen Informationen zu speichern. Anstatt alle FFT-Blöcke zu speichern, können beispielsweise nur die durchschnittlichen Magnituden oder die Spitzenwerte der

Frequenzen aufgezeichnet werden. Dies reduziert die Menge der gespeicherten Daten erheblich, da nur aggregierte oder besonders wichtige Informationen behalten werden. Diese Methode eignet sich besonders, wenn die detaillierten Daten der einzelnen Blöcke nicht benötigt werden.
Eine erweiterte Methode zur FFT-Berechnung besteht darin, die Analyse in einem sliding Window durchzuführen und die Ergebnisse zwischendurch in temporären Dateien zu speichern. Diese Chunk-basierte Verarbeitung, teilt die Audiodaten in kleinere Abschnitte auf, die in den verfügbaren RAM passen. Nach der Verarbeitung eines Chunks werden die Ergebnisse gespeichert und der nächste Chunk wird geladen. Diese kontinuierliche Verarbeitung verhindert, dass der gesamte Speicher belegt wird und ermöglicht dennoch eine detaillierte Analyse der Audiodaten.
Da es sich bei der analysierten WAV-File allerdings um keine Sprachaufnahme, sondern um einen kontinuirlichen Frequenzmix handelt, ist eine feine zeitliche Auflösung nicht unbedingt erforderlich. Ein sinvoller Lösungsansatz um den Speicherverbrauch zu deutlich zu reduzieren wäre somit eine größere Schrittweite. Anstatt die Blöcke um jeweils nur einen Schritt zu verschieben, können sie um mehrere Schritte verschoben werden. Dies reduziert die Anzahl der berechneten Blöcke und somit auch die Menge der gespeicherten Daten.

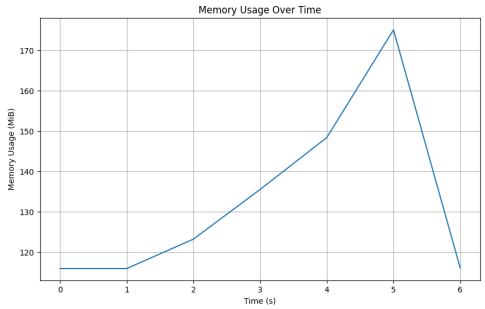
Ein simpler Ansatz die Schrittweite zu varrieren wäre es, sie gleich der angewendeten Blockgröße zu setzen. Das Ergebniss wären hierbei, dass ausschließlich sich nicht überlappende Blöcke betrachtet werden. Abbildung 1.3 beinhaltet eine Speicheranalyse mit genau diesen Einstellungen. Leicht zu erkennen ist, das sowohl der benötigte Speicherplatz und die Berechnungszeit sich um ein vielfaches verkleinert hat. Dadurch können wir weitaus größere Blockgrößen betrachten.



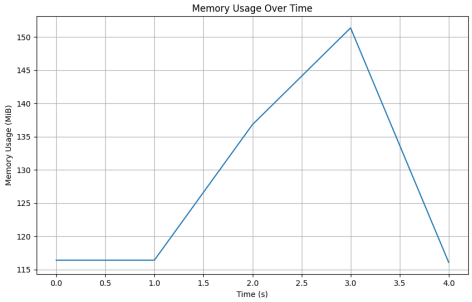
(a) Speicherbedarf mit Blockgröße 64



(b) Speicherbedarf mit Blockgröße 512



(c) Speicherbedarf mit Blockgröße 1024



(d) Speicherbedarf mit Blockgröße 2048

Abbildung 1.3: Speicherplatzbedarf mit Blockgroßer Schrittweite

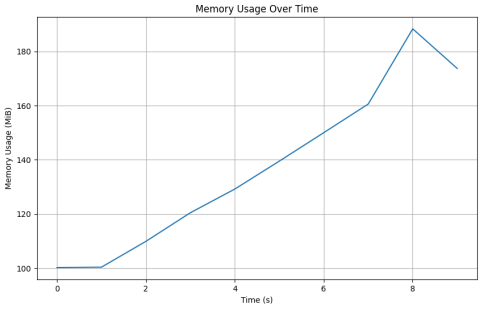
1.3 Variationen

Als Variation, wurde ein andere Prozessor verwendet, um die Unterschiede zwischen einem AM und BGA Sockeln festzustellen, als auch unterschiedliche Betriebssysteme.

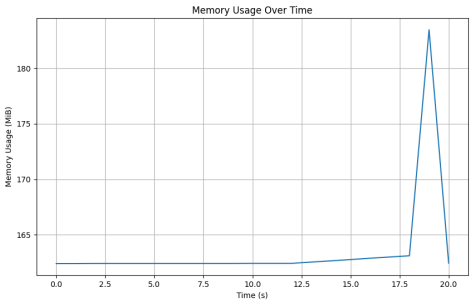
Attribut	**Details**
Prozessor	Intel Core i5-8250U
Architektur	Kaby Lake R
Kerne	4
Threads	8
Basis-Taktfrequenz	1.6 GHz
L3-Cache	6 MB
Fertigungstechnologie	14nm
Speicherunterstützung	DDR4-2400, LPDDR3-2133
PCIe-Unterstützung	PCIe 3.0
Sockel	BGA 1356

Tabelle 1.2: Spezifikationen des Intel Core i5-8250U

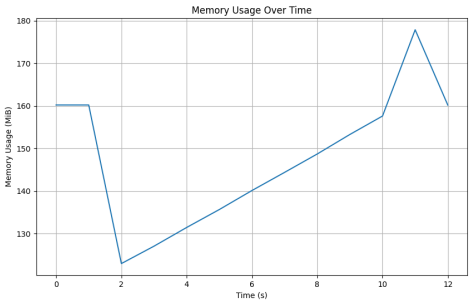
Auf dem oben beschriebenen Prozessor war sowohl unter Linux als auch unter Windows bereits bei kleinsten Blockgrößen ein Engpass festzustellen. Die Plots in Abbildung 1.5 zeigen die Ergebnisse der Speicheranalyse unter Linux. Ähnlich zum Prozessor aus den vorherigen Aufgaben, benötigen wir hier nur auch für größere Blöcke aufgrund der Blockgroßen Schrittweite nur wenige Sekunden für die Berechnung.



(a) Speicherbedarf mit Blockgröße 64



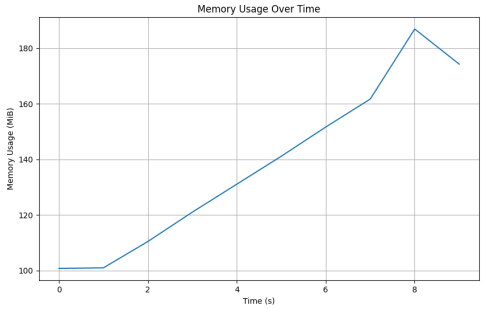
(b) Speicherbedarf mit Blockgröße 512



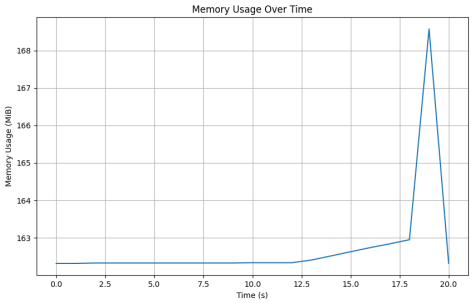
(c) Speicherbedarf mit Blockgröße 2048

Abbildung 1.4: Speicherplatzbedarf mit Blockgroßer Schrittweite unter Linux

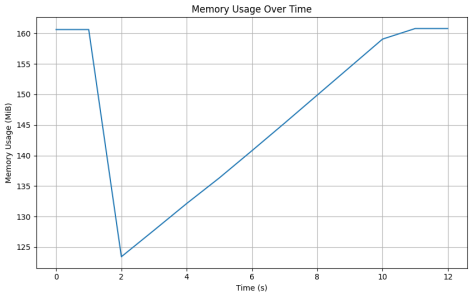
Führen wir die gleichen Tests auf gleichem Prozessor unter Windows durch, so scheint die Anwendung unter Windows etwas weniger Speicher zu verbrauchen. Diese Unterschiede können auf eine Vielzahl von Faktoren zurückzuführen sein, die sowohl in den Betriebssystemen als auch in deren spezifischer Implementierung und Verwaltung von Ressourcen liegen. Windows und Linux haben jeweils ihre eigenen Methoden zur Verwaltung von Speicher, die unterschiedliche Auswirkungen auf die Effizienz der Speichernutzung haben können. Windows verwendet ein Speichermanagementsystem, das häufig auf die Bedürfnisse von Desktop-Anwendungen optimiert ist. Es nutzt Techniken wie den "Heap Manager", der Speicher effizient verwaltet und Fragmentierung minimiert. Windows tendiert dazu, aggressiver ungenutzten Speicher zurückzufordern und Prozesse priorisiert zu behandeln, die eine hohe Leistung erfordern.



(a) Speicherbedarf mit Blockgröße 64



(b) Speicherbedarf mit Blockgröße 512



(c) Speicherbedarf mit Blockgröße 2048

Abbildung 1.5: Speicherplatzbedarf mit Blockgroßer Schrittweite unter Windows

2. Nächste Übung

2.1 Aufgabe x

3. Nächste Übung

3.1 Aufgabe x

Literaturverzeichnis