Heterogeneous Computing

Übung 1

Maximilian Späth

Mai 2024

1 WAV Analyse mittels Fast Fourier Transformation in Python

Zunächst hatte ich versucht mithilfe der KI ein Programm in Java anfertigen zu lassen. Da ich noch nicht viel ausprobiert hatte mit den Blockgrößen und den shift sizes war ich mir deren Auswirkung nicht bewusst, und scheine mein System mit dem JAVA Programm mit zu kleinem shifting maßlos überfordert zu haben. Da ich hier nicht wirklich auf ein Ergebnis kam, entschied ich mich ein Programm in Python schreiben zu lassen.
Hier hab ich mit einer Blockgröße (**B**) von 1024 und einer Größe für das Shifting (**S**) von 512 schnell Ergebnisse bekommen. Von diesem Punkt an habe ich **B** und **S** variiert, wobei ich **S** nur nach unten korrigiert habe. Entsprechend den Hinweisen im Aufgabenblatt habe ich mir die Hauptfrequenzen sowie die Amplitude der Blöcke ausgeben lassen.

1.1 Beobachtungen

Für bspw. ein **B** von 256 und einem **S** von 2, wird zunächst sehr lange gerechnet. Viele aufeinanderfolgende Blöcke haben eine Hauptfrequenz von *0Hz*, gefolgt von einer, ebenfalls langen jedoch verglichen kürzeren, Sequenz von Blöcken mit einer gleichbleibenden Hauptfrequenz != 0, die wenn ich den output so durchgehe für das gewählte **B** und **S** immer *172Hz* beträgt. Es scheint, als würden

viele meiner durch das kleine **S** entstandenen Blöcke, und ich interpretiere das jetzt einfach mal so, den gleichen Informationsgehalt haben, also wähle ich ein größeres **S**, sowie ein größeres **B**.
Für ein **B** von 512 und einem **S** von 16, werden die Sequenzen jeglicher Frequenzen (*0Hz* oder *!0Hz*) etwas kleiner, da ja weniger Blöcke berechnet werden und der Informationsgehalt einzelner Blöcke meiner Auffassung hier zunächst wächst. Die Hauptfrequenz $! = 0$ entspricht stets *86.13Hz*, und weicht davon nicht ab. Dem Internet entnehme ich das eine Hauptfrequenz von 0Hz entweder aufgrund einer fehlerhaften Analyse, einem DC-Offset, oder Hintergrundrauschen vorkommen kann. Da der Code an sich gut zu funktionieren scheint, ging ich mal rein intuitiv davon aus, dass die Hauptfrequenz von *0Hz* mit dem gewählten **B** und oder dem gewählten **S** zusammenhängt und es vielleicht aufgrund einer hohen Abtastrate Bereiche gibt, in denen kein Signal ausgegeben wurde, bzw. ein Schwellwert nicht übertreten wurde und daher *0Hz* in diesem Block die richtige Hauptfrequenz ist. Für eine Erhöhung von **B** auf 1024 und einem **S** von 16 erhielt ich nun eine weitaus kürzere wiederkehrende Sequenz von Blöcken mit einer Hauptfrequenz von *0Hz*. Die wiederkehrende Sequenz von Blöcken mit einer Hauptfrequenz $! = 0$ hingegen ist um einiges länger geworden, scheinbar überlagern sich die Blöcke nun vermehrt so, dass das Signal häufiger Teil des Blocks ist, die Hauptfrequenz betrug zu dem durchgängig *43.07Hz*. Die bisherigen Werte der Hauptfrequenzen $! = 0$ in meinen bisherigen Ansätzen vergleichend, scheint ein enger Zusammenhang zwischen diesen zu bestehen, da sie sich immerhin um den Faktor 2 unterscheiden [*43Hz , 86Hz, 172Hz*].
Ich bleibe also meiner Linie und meinem angeschlagenen Verständnis der Fourier Analyse treu und versuche weiter die Blöcke mit einer Hauptfrequenz von *0Hz* zu eliminieren und erhöhe das **B** auf 2048, bei einem unveränderten **S** von 16. Die Hauptfrequenzen sind jetzt alle durchgehend bei *43Hz*, genauer gesagt bei textit43.07Hz, wie auch in dem zuvor durchgeführten Durchlauf mit einm **B** von 1024 und einem **S** von 16. Die Amplitude liegt hier bei ungefähr 180.9 und variiert nur ganz leicht zwischen augenscheinlich 180.85 und 180.95. Ich behalte ein **B** von 2048 bei, und erhöhe das **S** auf 32 und erhalte weiterhin durchgehend *43.07Hz* während die Varianz in der Amplitude minimal ansteigt und augenscheinlich zwischen 180 und 181 mit einer Tendenz zu 181 liegt.

Ich erhöhe das **S** testweise auf 128 und 256, erhalte jedoch keine sich stark unterscheidenden Ergebnisse von einem **S** von 32 die bezüglich der Amplitude die kleinste Varainz aufweist. Ich erhöhe ein weiteres Mal das **B** auf 4096 mit einem **S** von 64. Die Amplitude steigt hier auf 359, was grob einem Faktor von 2 gegenüber 180 entspricht die ich erhalten habe, als ich die Hälfte der Blocksize mit einem gleichen shifting probierte. Die Hauptfrequenz ist wieder *43.07Hz*.

1.2 Einschätzung

Nachdem ich mich mehrfach gewundert habe, warum ich ab einem bestimmten **B** und **S** als Hauptfrequenz nur noch *43.07Hz* erhalte, habe ich überlegt ob ich hier etwas richtiges ausprobiert habe und dies schon eine mögliche Antwort sein könnte. Immerhin ist rein akustisch in der Eingangs Datei auch ein durchgehend gleichbleibendes Signal zu hören, was vermutlich aber nicht zwingend darauf hinweisen muss das das Ausgangssignal auch einer einzigen Frequenz entstammt. Nun ja ich habe mich trotzdem mal dazu entschieden die Frequenz zu googlen und es stellt sich raus das 43Hz im Gamma Frequenzbereich liegt, was nach Wikipedia folgend definiert ist: „Eine Gamma-Welle oder ein Gamma-Rhythmus ist ein Muster neuronaler Schwingungen beim Menschen mit einer Frequenz zwischen 25 und 140Hz, wobei der 40-Hz-Punkt von besonderem Interesse ist“. Der 40Hz Punkt ist von besonderem Interesse und nachdem ich mich habe selber von ein paar 43Hz Schwingungen akustisch stimulieren lassen, gehe ich davon aus den Frequenzen Mix richtig entschlüsselt zu haben.

2 Speicherbedarfsanalyse der Python Implementierung

Um den Speicherbedarf zu analysieren, wurden die in der nachfolgenden Abbildung 1 aufgeführten Werte für **B** und **S** getestet und das python package *tracemalloc* verwendet. Für ein **B** von 2048 und einem **S** von 32 ergab sich ein Höchstwert von umgerechnet ca. 1.2726 Gibibytes(**GB**). Der meiste Speicher wird hier aber für das Spektrogram 2DArray belegt, dass, durch die gewählte Größe von Blöcken, dem shifting und dem Rückgabewert der diskreten Fourier Transformation, beliebig groß wird . Nachdem die Funktion von diesem Over-

Durchlauf	Blockgröße(**B**)	Shiftgröße(**S**)	Speicherverbrauch in **GB**
1.	2048	32	0.010165
2.	2048	64	0.00510731
3.	1024	16	0.020282

Abbildung 1: Ergebnisse der Speicherverbrauchs Analyse in Python, Speicherverbrauch ist in Gibibytes(**GB**) angegeben

head erlöst ist, resultiert ein Speicherverbrauch von 0.010165 **GB**. Für ein **B** von 2048 und einem **S** 64 resultiert ein Verbrauch von 0.00510731 **GB**, was der Hälfte des zuvor gemessenen Verbrauchs entspricht und kohärent ist mit der Verringerung der Anzahl der Blöcke um den Faktor 2. Für ein **B** von 1024 und einem **S** von 16 ist der Verbrauch 0.020282 **GB**.

3 Zwei Alternative Ansätze

Für die Aufgabe 3 wird der in Aufgabe 1 und 2 verwendete Ansatz zur Fourier Analyse mit Anwendung einen Hanning Fensters in Java und R übersetzt. Folgend werden zunächst die Ergebnisse der Java Implementierung dargestellt, gefolgt von der Darstellung der Ergebnisse einer R Implementierung. Der Code sowohl für Java als auch R wurde über Prompt-Engineering mit ChatGPT-4 erstellt.

3.1 Speicheranalyse FFT in Java

Die KI verwendet für die Umsetzung in Java das *javax.sound* package zum Einlesen und Arbeiten mit der WAV-Datei, sowie die *jtransform* Bibliothek um von dort die Funktionalität zur *Fast Fourier Transformation* nutzen zu können. Wie im python Code enthalten, wurde gefordert ebenfalls das Hanning Fenster anzuwenden. Entsprechend der Größen in Aufgabe 2 wird der Speicherverbrauch für die gleichen Größen für **B** und **S** bestimmt. Die Werte für **B** und **S** sowie die Ergebnisse für den Speicherverbauch können der nachstehenden Abbildung 2 entnommen werden.
Zur Berechnung des verwendeten Speichers der JVM wird ein Startwert durch

Durchlauf	Blockgröße(**B**)	Shiftgröße(**S**)	Speicherverbrauch in **GB**
1.	2048	32	0.01004894
2.	2048	64	0.00543848
3.	1024	16	0.00084613

Abbildung 2: Ergebnisse der Speicherverbrauchs Analyse in Java, Speicherverbrauch ist in Gibibytes(**GB**) angegeben

Runtime.totalMemory() - Runtime.freeMemory() berechnet. Für ein **B** von 2048 und ein **S** von 32 resultiert ein Speicherverbrauch von 0,01004894 **GB**, welcher leicht kleiner ist als jener der Python Implementierung. Für ein **B** von 2048 und ein **S** von 64 resultiert ein Speicherverbrauch von 0,00543848 **GB**, welcher leicht über dem Verbrauch der Python Implementierung liegt. Für ein **B** von 1024 und ein **S** von 16 resultiert ein Speicherverbrauch von 0,00084613 **GB**, der so viel geringer ist als jener der Python Implementierung dass dies für mich keinen Sinn macht. Die Anzahl der Blöcke ist nach Testausgaben die gleiche, das **S** ist ebenfalls das gleiche, und die beiden Durchläufe zuvor zeigen zumindest einen ähnlichen Speicherbedarf. Ich bin daher davon ausgegangen, dass das vielleicht irgendwie mit dem Garbage Collector zusammenhängen könnte und habe mal versucht, häufiger während der Fourier Analyse den verbrauchten Speicher zu sichern, jedoch ohne einen anderen Speicherbedarf zu erhalten. Ich habe daher leider keine Eklärung dafür, warum der Speicherbedarf bei einem **B** von 1024 und einem **S** von 16 bei Java so viel geringer sein soll als bei Python, zumal grundlegend zu erwarten wäre, dass die Java Implementierung generell mehr Speicher benötigt. Jedoch ist es aufgrund der JVM und dem Garbage Collector nicht ganz trivial den verbrauchten Speicherplatz zuverlässig zu bestimmen und so hoffe ich auf konsistentere Ergebnisse in der folgenden Umsetzung mit R.

3.2 Speicheranalyse FFT in R

Die Ergebnisse der Implementierung in R können der Abbildung 3 entnommen werden. Um den Speicherverbrauch zu analysieren habe ich die Bibliotek *profmem* genutzt, und den entsprechenden Befehl auf die berechnende Funktion angewendet um so den Speicherbedarf zu bestimmen. Während der Analyse

Durchlauf	Blockgröße(**B**)	Shiftgröße(**S**)	Speicherverbrauch in **GB**
1.	2048	32	0.38575825
2.	2048	64	0.38575825
3.	1024	16	0.38704208

Abbildung 3: Ergebnisse der Speicherverbrauchs Analyse in Java, Speicherverbrauch ist in Gibibytes(**GB**) angegeben

werden keine Ergebnisse gepsiechert, aus einem mir unerfindlichen Grund unterscheiden sich die Werte für den Speicherverbauch für die unterschiedlichen GRößen kaum, und sind die bisher am höchsten gemessenen. Ich weiß daher an diesem Punkt leider auch nicht was ich vergleichen soll, da es keinen Sinn macht, dass alle Durchläufe mit R gleich viel Speicherplatz benötigen, obwohl sich die Anzahl zu berechnender Blöcke stark unterscheidet.

3.3 Zusammenfassung

Abschließend kann ich sagen, dass ich der Implementierung mit Python am meisten vertraue. Diese ging verhältnismäßig einfach von der Hand, ließ sich einfach warten und erweitern. Die Implementierung in Java war schon etwas schwieriger, und der Vergleich der Ergebnisse von Java mit derer von Python scheint für mich aufgrund des Garage Collectors der JVM zu hinken, da prinzipiell zu erwarten wäre, dass der gleiche Aufwand in Java mehr Speicherplatz benötigt als in Python, da Java ganze Objekte im Arbeitsspeicher hält. Die Implementierung in R war am wenigsten sinnvoll. Die Speicheranalyse fällt ohne jeden Grund immens hoch aus und unterschiedet sich unahängig der großen unterschiede zu berechnender Blöcke kaum. ich kann nur davon ausgehen, dass die verwendete Bibliothek profmem sich anders verhält als ich es annehme, da keine großen Arrays speicher technisch alloziert werden, und auch diese müssten sich ja zwischen den Durchläufen stark unterscheiden. Bezüglich der Analyse der Hauptfrequenzen bei den verschiedenen Blockgrößen und gewählten Größen für das Shifting erhalte ich zumindest die gleiche Frequenz von 43.07Hz, während die Amplitude hier jedoch in den drei Implementierungen unterschiedlich hoch ist. Wenn in folgenden Aufgaben weiterhin die Fourier Analyse eine Rolle spielt,

werde ich von meiner Python Analyse aus weiterarbeiten.