LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Gruppe 146 – Abgabe zu Aufgabe A217
Wintersemester 2022/23

Giancarlo Calvache Vlad-Alexandru Marin Emre Öztas

1 Einleitung

Diese Ausarbeitung befasst sich mit der computergestützten Darstellung des *Burning-Ship* Fraktals, einer modifizierten Version des *Mandelbrot* Fraktals. Fraktale[4] sind geometrische Objekte und Muster, die auf allen Größenskalen *selbstähnlich* sind und deren Komplexität nie zu Ende geht. Der Begriff wurde 1975 vom Vater der fraktalen Geometrie *Benoit Mandelbrot* ins Leben gerufen und leitet sich vom Lateinischem Wort Fractus, im Deutschen "gebrochen", ab. Die Geometrie der Frakale wurde als Antwort auf die Frage eingeführt, wie man die allgegenwärtige Komplexität der in der Natur vorkommenden Gegenstände, wie die Äste von Bäumen, Blütenstrukturen oder Küstenlinien, in einer geometrischen Formulierung schematisieren kann.
Oftmals, entstehen Darstellungen von Fraktalen durch einfache Rekursions- oder Iterationsvorschriften, welche ein komplexes Muster erzeugen. Die Art und Weise, wie ein Fraktal vom Computer erzeugt wird, ist dabei eng gekoppelt mit seiner mathematischen Definition. Daher, sind Fraktale lediglich eine verbildlichte Darstellung einer mathematischen Rekursionsformel. Im Fall des *Burning-Ship* Fraktals verwenden wir einen *Escape-Time-Algorithmus*, um zu berechnen, ob ein Punkt sich in der *Burning-Ship*-Menge befindet oder nicht.
Eine Implementierung auf Basis von 32-bit single-precision floating points kann nur eine sehr begrenzte Genauigkeit in der Darstellung erzielen. Um eine höhere Genauigkeit zu realisieren, muss die Genauigkeit und damit die Speichergröße der verwendeten floating points gleichermaßen vergrößert werden, was zu einem geringeren Durchsatz in der parallelen Verarbeitung führt. Aufgrund der beschränkten Registergrößen von Rechnern, ist eine unendlich genaue Berechung von Fraktalen daher impraktikabel, zumindest wenn die Berechung nicht auf Kosten von Performanz und Speichernutzung des Rechners geht. Eine Möglichkeit die Berechung der Iterationsvorschrift zu erleichtern, ist es die Berechung aufzuteilen und jede parallele Verarbeitungseinheit mit der SIMD-Erweiterung zu optimieren. Diese Praxis taucht in moderner Software für die Generierung von Fraktalen auf, würde allerdings den Umfang des Projektes sprengen, da die Arithmetik der erweiterten Floating Points aus den genannten Gründen, angepasst werden müsste.
Vor diesem Hintergrund ist die Zielsetzung für das Projekt, eine Implementierung für die Iterationsvorschrift des *Burning-Ship* Fraktals zu schreiben, welche anhand Parallelität mittels SSE-CPU-Erweiterungen optimiert ist, und ein Rahmenprogramm zu entwickeln, durch welches das Fraktal in einer festgelegten Auflösung als verlustlose Bilddatei generiert werden kann. Hierzu verwenden wir das kompressionsfreie Format BMP[3][2]

mit einer Farbtiefe von 8-bit. Darüberhinaus, wird das Rahmenprogramm durch den Zusatz erweitert, das Fraktal bis zu 10000-Fach vergrößert erzeugen zu können, um die meist verborgenen Muster innerhalb von Fraktalen ansatzweise sichtbar zu machen. Die Entscheidungen und Erwägungen, welche wir für die Implementierung getroffen haben, werden in den nächsten Abschnitten näher erläutert.

2 Lösungsansatz

Zur Menge des *Burning-Ship*-Fractals gehören alle Punkte $c = \Re(c) + i * \Im(c) = (\Re(c), \Im(c))$ der Komplexen Ebene im Raum von $[-2, 2]^2 \in \mathbf{R}^2$, die nicht divergieren, wenn die folgende Iterationsvorschrift mit $z_0 = 0$; k-Fach angewendet wird:

$$z_{n+1} = (\mid \Re(z_n) \mid + \mid \Im(z_n) \mid)^2 + c \tag{1}$$

Wir bezeichnen später c als die Position des Pixel auf einem Bild der Dimension $m \times n$, welches wir erzeugen.

2.1 Das Rahmenprogramm

Der Benutzer soll in der Lage sein, einen exakten Punkt auf der *Burning-Ship*-Menge zu fixieren und eine Vergrößerung an der Stelle vorzunehmen. Dies wird durch die beiden Programm-Optionen *-s<Real>,<Imag>* und *-r<Vergrößerungsfaktor>* umgesetzt, wobei alle Argumente dieser Optionen Gleitkommazahlen sind, die eine Präzision von 64-Bit in der Mantisse haben, wodurch etwa 15 Nachkommastellen in Dezimaldarstellung für die Berechnung verwendet werden können. Alle möglichen Eingaben für -s sind auf den oben genannten Raum beschränkt und die Eingaben für -r sind durch das Intervall $[4.0, 1e-15]$ begrenzt. Darüberhinaus soll der Benutzer ebenfalls, die Dimension des Ausgabebildes bestimmen können, was durch die Option *-d<Breite,Höhe>* erfolgt. Da verwertbare Ergebnisse entstehen sollen, wird eine Dimension des Ausgabebildes auf das Intervall $[100, 8000]$ in Pixeln beschränkt, da die Angabe zu kleiner Dimensionen dazu führt, dass das Ergebnis der Iteration nicht wahrnehmbar ist und zu große Dimensionen dazu, dass Bilder mit einer Speichergroße von mehreren Gigabytes erzeugt werden. Zu weiteren Programmargumenten gehören -h bzw. –help, wodurch ein Hilfstext zur Benutzung des Programmes angezeigt wird, -o zum setzten des Namen der Ausgabedatei, -B<Anzahl der Wiederholungen> durch Benchmarks, mit der Angegebenen Anzahl wiederholt werden. Die Option -V<Index> gibt an, welche Implementierung verwendet wird, um das Fraktal zu erzeugen. Es sind die folgenden Implementierungen vordefiniert, welche dem Index nach, beginnend mit Null, sortiert sind:

1. burning_ship — Ohne Optimierungen
2. burning_ship_V1 — SIMD
3. burning_ship_V2 — AVX

Die Option -n<Natürliche Zahl> gibt an, für wie viele Iterationen k die Folge $(z_n)_k$ berechnet wird.

2.2 Umsetzung der Iterationsvorschrift

Um zu berechnen ob ein Punkt c auf der Komplexen Ebene gegen unendlich konvergiert oder nicht, oder anders gesagt ob der Punkt die Eigenschaft der Burning-Ship-Menge nicht erfüllt, müssen wir eine Eigenschaft der Komplexen Zahlen finden, welche uns erlaubt diese Invariante effizient zu überprüfen. Ein Beweis [1] zeigt, dass es ausreicht, für Punkte $c_{x,y}$ in einem Bereich der Komplexen Zahlen zu zeigen, dass $c_{x,y}$ genau dann sich nicht in der *Burning-Ship* Menge befindet, wenn ab einer Iteration k gilt, dass $\|z_k\| > 2$. Für ein c mit $\mid \Re(c) \mid > 2$ oder $\mid \Im(c) \mid > 2$ ist dies automatisch der Fall, weshalb wir nur eine Umgebung $c \in [-2, 2]^2$ betrachen müssen. Dazu wandeln wir die Iterationsvorschrift um, in dem wir die Gleichung in (1) auflösen und sie nach dem Real- und Imaginärteil aufschreiben:

$$\Re(z_{n+1}) = \Re(z_n)^2 - \Im(z_n)^2 + \Re(c) \tag{2}$$

$$\Im(z_{n+1}) = 2 \mid \Re(z_n)\Im(z_n) \mid + \Im(c) \tag{3}$$

Diese beiden Gleichungen werden benötigt, um die Wurzelberechung innerhalb der Vektornorm zu umgehen und stattdessen zu Berechnen ob $\|z_k\|^2 > 4$. Wir bezeichnen jedes Pixel auf dem Ausgabebild als $p_{x,y} \in [0, Breite] \times [0, Hoehe]$. Da die Iterationsfunktion des *Burning-Ship* Fraktals die Farbstufe des Pixels anhand dem oben genannten Raum berechnet d.h. $\Phi_{burning\_ship}(p_{x,y}, n) : ([-2, 2]^2) \times n \rightarrow [0, 1]$, sodass die Koordinate, welche vergrößert werden soll, sich bei Eingabe genau im Mittelpunkt des Bildes befindet, müssen wir zwei Projektionen $\pi_x(p_{x,y}) : [0, Breite] \rightarrow [-2, 2] \cdot R$ und $\pi_y(p_{x,y}) : [0, Hoehe] \rightarrow [-2, 2] \cdot R$ definieren, welche die Dimensionen des Bildes in eine geeignete Form umwandeln bevor sie der Iterationsfunktion übergeben werden. $R \in [1, 10^{-6}]$ ist hierbei der Vergrößerungsfaktor des Bildes. Dadurch erhalten wir das folgende Programm, welches das *Burning-Ship* Fraktal mit Angabe der Dimension des Bildes, dem Vergrößerungsfaktor und der Anzahl an Interationen pro Pixel berechnet:

2.3 Färbung und Schreiben in einem BMP-Datei

Wir haben eine Farbentabelle erstellt, die im Header der BMP-Datei gelagert wird. Diese Tabelle entsteht aus 16 Farben. Nachdem wir die Anzahl von Iterationen für jede Pixel berechnet haben, skalieren wir diese Nummern im Bereich [0,15]. Die erhaltene Index dient dafür, dass wir für jede Pixel eine Farbe aus der Tabelle zuordnen. Zunächst wird eine BMP-Datei erstellt, die aus dem vordefinierten Header und die Farbenindexe aller Pixeln entsteht.

3 Genauigkeit

Wir haben eine Testfunktion implementiert, die die Ausgabe der gegebenen Parameter und Implementierung berechnet. Gleichzeitig wird die SISD-Implementierung mit der gleichen Parameter geruft. Die Ergebnisse werden pixelweise verglichen und das Verhältnis von unterschiedlichen Pixel zu alle Pixel wird ausgegeben. Des Weiteren haben wir die Implementierungen mit verschiedenen Eingaben testet. 1

Abbildung 1: Größte burning ship mit Parametern -d 800,800 -n 40 -r 4

d	r	s	n	SIMD	AVX
800,800	1.0	0,0	40	14.116094%	14.116094%
800,800	1.0	0,0	30	0.270625%	0.270625%
800,800	1.0	0,0	50	12.767344%	12.767344%
1000,1000	1.0	0.0	40	13.974400%	13.974400%
800,800	4.0	0,0	40	16.977344%	16.977969%
800,800	0.1	0,0	40	0%	0%
800,800	0.009	-1.77001035,-0.05000005	40	48.806250%	48.881719%

Tabelle 1: Fehlerquotioent der Implementierungen

Die Tabelle zeigt den Fehlerquotienten der Implementierungen von SIMD und AVX. Es scheint, dass der Fehlerquotient von verschiedenen Faktoren abhängt, wie z.B. den Werten für d, r, s und n. In einigen Fällen haben beide Implementierungen einen ähnlichen Fehlerquotienten, wie bei 800,800 für d und 40 für n (14.116094% für SIMD und 14.116094% für AVX). In anderen Fällen ist der Fehlerquotient Null, wie bei 800,800 für d und 0.1 für r. Allgemein scheint AVX einen geringfügig niedrigeren Fehlerquotienten zu haben als SIMD.

4 Performanzanalyse

Wir haben mit verschiedenen Eingaben Laufzeitmessungen durchgeführt. Alle Messungen, die in diesem Unterabschnitt erläutert werden, sind auf einem System mit einem Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz (8 CPUs), 4.2GHz Prozessor, 16384MB Arbeitsspeicher, Windows 10 Home 64-bit, 256KB L1, 1MB L2 und 8MB L3.

Wir haben in allen Optimierungsstufen getestet. Abbildung 2, die mit -O3 kompiliert wurde, zeigt, die durchschnittliche Laufzeite von drei verschiedenen Implementierungen, in Abhängigkeit von dem maximalen Anzahl von Iterationen. Abbildung 3 ist dasselb, aber diesmal mit -O0 kompiliert. Abbildung 5 stellt die durchschnittliche Performanze der drei Implementierungen beim Small Ship, abhängig von den Dimensionen dar. Die Eingaben sind so gewählt, damit die Performanze bei verschiedenen Bildgrößen und maximale Iterationsschritte deutlich darzustellen

Wenn man den Graphen untersucht, kann man sagen, dass die erste Implementierung langsam ist und eine längere Zeit benötigt, um eine bestimmte Aufgabe auszuführen. Die zweite Implementierung ist schneller als die erste, aber nicht so schnell wie die dritte. Die dritte Implementierung ist die schnellste von allen dreien und benötigt die kürzeste Zeit, um eine Aufgabe auszuführen. Dieses Ergebnis ist erwartet, da V0 1 Pixel, V1 4 Pixel und V2 8 Pixel pro Iterationsschritt überabeitet.

Es ist wichtig zu beachten, dass die Geschwindigkeit einer Implementierung von verschiedenen Faktoren abhängt, wie z.B. der Effizienz des Algorithmus, dem verwendeten Prozessor und dem verwendeten Arbeitsspeicher. Trotzdem kann die schnelle Implementierung ein klarer Vorteil sein, insbesondere wenn es um große Datenmengen

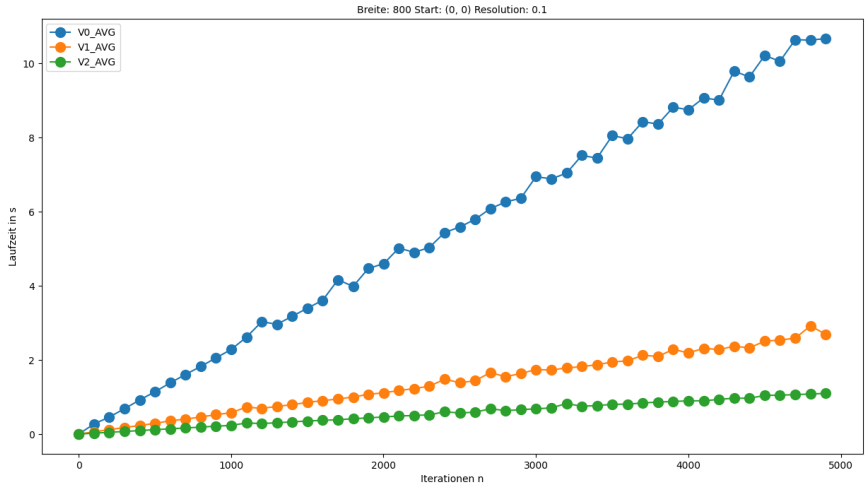

Abbildung 2: Progessive n mit -B5 und -O3

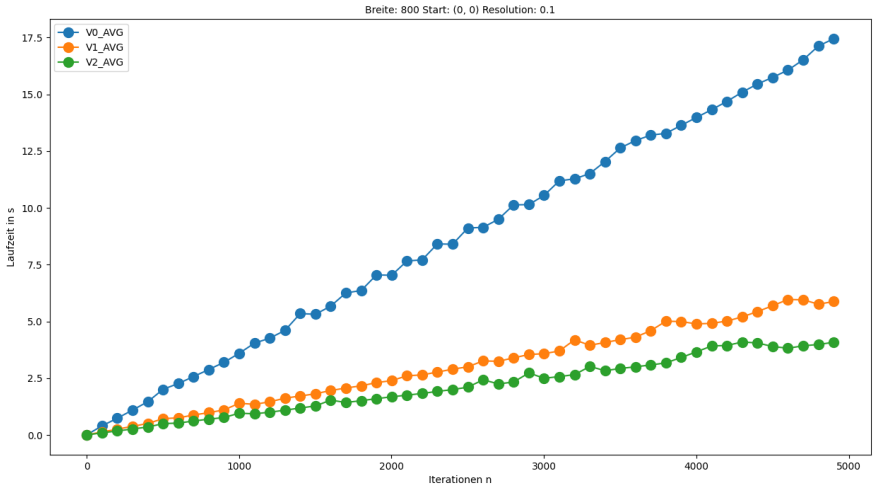

Abbildung 3: Progessive n mit -B5 und -O0

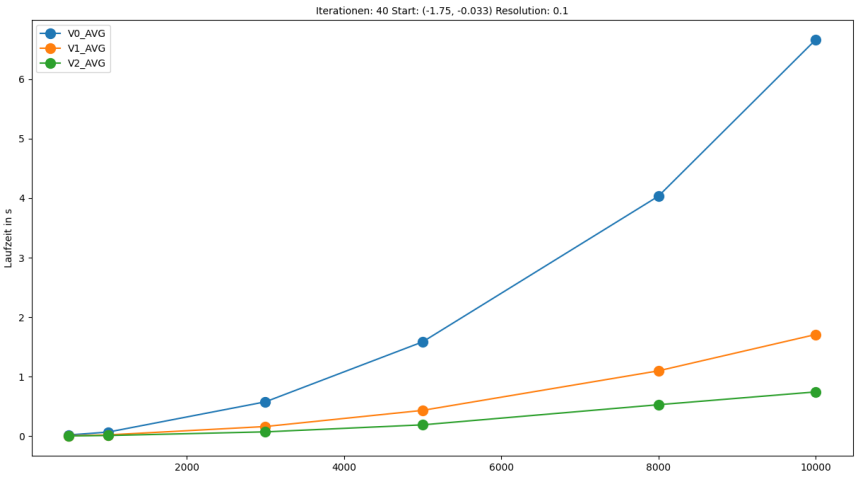

Abbildung 4: Small Ship mit -B5 und -O3

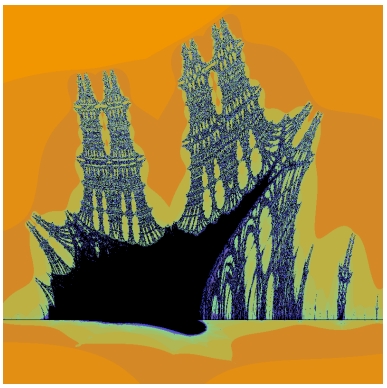

Abbildung 5: Small Ship

oder komplexe Aufgaben geht.

5 Zusammenfassung und Ausblick

In diesem Projekt wurde eine Funktion, die den Burning Ship Fraktal erstellt, implementiert. Die Funktion erhält als Parameter den Startpunkt des Ausschnitts, die Breite, die Höhe, die gewünschte Auflösung und die maximale Anzahl der Iterationen. Außerdem wird ein Zeiger auf einen Speicherbereich übergeben, der groß genug ist, um die berechneten Bitmap-Daten des Ergebnisses aufzunehmen. Die Funktion ist so konzipiert, dass sie eine innvolle Farbpalette verwendet, um das Ergebnis darzustellen. In diesem Projekt wurden insgesamt drei Implementierungen erstellt: eine normale Implementierung, eine SIMD-Implementierung und eine AVX-Implementierung. Die haben verschiedene Laufzeitverhältnisse und Genaugkeiten haben.

Eine mögliche Erweiterung des Projektes könnte die Verwendung von GPU-Beschleunigung sein, um die Berechnungsgeschwindigkeit und Effizienz zu verbessern. Es ist auch wichtig, alternative Lösungen zu untersuchen, um Nachteile, wie Korrektheit von V1 und V2 der aktuellen Implementierungen zu erkennen und gegebenenfalls zu verbessern. Zum Beispiel könnte eine alternative Lösung eine höhere Auflösung bei gleichbleibender Geschwindigkeit und Effizienz ermöglichen. Darüber hinaus kann die Vergleichbarkeit der drei Implementierungen weiter untersucht werden, um die beste Option für bestimmte Anwendungsfälle zu bestimmen.

Literatur

[1] "run away to infinity"criterion. `https://users.math.yale.edu/public_html/People/frame/Fractals/`, visited 2023-1-4.

[2] Bitmapinfoheader structure (wingdi.h), November 2022. `https://learn.microsoft.com/en-us/windows/win32/api/wingdi/ns-wingdi-bitmapinfoheader`, visited 2022-12-23.

[3] Uday Hiwarale. Bits to bitmaps: A simple walkthrough of bmp image format, October 2019. `https://medium.com/sysf/bits-to-bitmaps-a-simple-walkthrough-of-bmp-image-format-765dc6857393`, visited 2022-12-23.

[4] B. B. Mandelbrot. *Fractal Geometry: what it is, and what does it do?* Mathematics Department, Yale University, New Haven, Connecticut 06520, U.S.A., 1989. `https://users.math.yale.edu/mandelbrot/web_pdfs/fractalGeometryWhatIsIt.pdf`, visited 2023-1-4.