LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Gruppe 296 – Abgabe zu Aufgabe A210

Sommersemester 2022

Pascal Gaertner Saina Amiri Moghadam Charalampos Piotopoulos

1 Einleitung

Die Mandelbrot-Menge, benannt nach dem Benoit Mandelbrot, ist eine Menge komplexer Zahlen. Sie wird durch Iteration erzeugt, d.h. durch Wiederholung eines Prozesses, bis eine Bedingung erfüllt ist. Die Funktion der Mandelbrot-Menge ist ein quadratisches Polynom der Form $z_{i+1} = z_i^2 + c\,(i \geq 0)$, wobei c eine vom Benutzer festgelegte konstante Zahl ist. z_0 hingegen bestimmt den Startwert für die Iteration. Für unsere Projektübung haben wir den Startwert $z_0 = 0$ verwendet. Die Anwendung der Funktion liefert uns die neue Zahl z_1. Nun verwenden wir das Ergebnis der vorherigen Berechnung als Eingabe für die nächste. Die resultierende Folge, die durch diese Iteration erzeugt wird, kann je nach c zwei Beobachtungen aufzeigen. Die erste ist, dass wir mit jeder Iteration eine größere Zahl erhalten und die Folge gegen unendlich tendiert. Die andere Beobachtung ist, dass die Folge im absoluten Wert endlich bleibt, d.h. beschränkt ist. Hier haben wir zwei Beispiele, die beide Fälle veranschaulichen.

Im ersten Beispiel nehmen wir den Wert $2i$ für c.

$$
\begin{aligned}
z_0 &= 0 \\
z_1 &= 0^2 + 2i = 2i \\
z_2 &= (2i)^2 + 2i = -4 + 2i \\
z_3 &= (-4 + 2i)^2 + 2i = 12 - 14i \\
z_4 &= (12 - 14i)^2 + 2i = -52 - 334i
\end{aligned}
$$

Wir sehen, dass die Folge ins unendliche tendiert. Wenn wir jetzt i für c nehmen ändert sich das Benehmen der Folge.

$$
\begin{aligned}
z_0 &= 0 \\
z_1 &= 0^2 + i = i \\
z_2 &= i^2 + i = -1 + i \\
z_3 &= (-1 + i)^2 + i = -i \\
z_4 &= (-i)^2 + i = -1 + i
\end{aligned}
$$

Hier sehen wir das die Folge beschränkt ist.

Die Mandelbrot-Menge ist eine geometrische Darstellung der obigen Beobachtungen. Die Mandelbrot-Menge besteht aus allen Zahlen, deren Folge mit dem Startwert von 0 nicht ins Unendliche geht.

In unserem Projekt haben wir einen Algorithmus in C entwickelt, der die Mandelbrot-Menge berechnet, wenn der Benutzer manuelle Eingaben für den Wert von c, die Anzahl der Iterationen und andere Konfigurationen macht. Schließlich wird aus der berechneten Menge ein Bild erzeugt.

Diese Ausarbeitung beginnt mit der Beschreibung und Analyse des gewählten Lösungsansatzes. Dann wird die Korrektheit der Implementierung und des Ansatzes gezeigt und bewertet. Danach wird die Performanz der Implementierung analysiert und bewertet. Die Ausarbeitung schließt mit einer Zusammenfassung und einem Ausblick auf zukünftige Entwicklungen unseres Algorithmus.

2 Lösungsansatz

In diesem Kapitel wird der Lösungsansatz beschrieben und analysiert. Jede komplexe Zahl besteht aus einem Real- und einem Imaginärteil. Daher können wir die ursprüngliche Gleichung auf folgende Weise umformen.

$$z_n = (z_n)^2 + c$$
$$z_n = (Re_n + Im_n)^2 + Re_c + Im_c$$
$$z_n = Re_n^2 + 2Re_nIm_n + Im_n^2 + Re_c + Im_c$$

Wie die obige Darstellung zeigt, können wir die Berechnungen leicht in zwei Teile aufteilen. Den realen $z_n = Re_n^2 + Im_n^2 + Re_c$ und den imaginären Teil $z_n = 2Re_nIm_n + Im_c$. Da sich der Imaginärteil aus $i * n$ zusammensetzt und wir i^2 haben, können wir ihn auch als $z_n = Re_n^2 - n^2 + Re_c$ umschreiben.

Wir durchlaufen dann jedes Pixel und berechnen ihre jeweilige Größe sowie ihre tatsächlichen X- und Y-Koordinaten unter Berücksichtigung der Größe des Bildes und der Auflösung. Dann führen wir die Berechnungen mit der gewählten Anzahl von Iterationen durch. Außerdem haben wir eine Bedingung implementiert, die die for-Schleife vorzeitig verlässt, wenn eine der Berechnungen größer als 2 ist. Dazu werden Real- und Imaginärteil quadriert, addiert und dann verglichen, ob sie kleiner als 4 sind. Wenn dies nicht der Fall ist, wird die Schleife frühzeitig verlassen und der Pixel wird als weiß gespeichert. Wenn die for-Schleife auf natürliche Weise beendet wird, malen wir das Pixel schwarz.

Eine weitere Lösung wurde erstellt, um unsere erste Lösung dagegen zu testen. Diese Lösung gruppiert 8 Pixel zusammen. Dann wird für jedes Pixel innerhalb einer Gruppe berechnet, ob es Teil der Mandelbrot-Menge ist oder nicht. Falls die Gesamtzahl der Pixel nicht durch 8 teilbar ist, werden die restlichen Pixel einzeln berechnet.

Wie später im Abschnitt über die Leistungsanalyse gezeigt wird, schneidet er bei Iterationen unter 5000 besser ab. Dies kann auf die Gruppierung innerhalb dieses Algorithmus zurückgeführt werden. Durch die Gruppierung verbringen wir weniger Zeit in einer for-Schleife und haben insgesamt weniger Sprungbefehle. Bei mehr Iterationen kann sich dies summieren und den Code schneller machen. Nach 5000 Iterationen scheitert das Programm jedoch, weshalb der andere Ansatz als Lösungsansatz gewählt wurde.

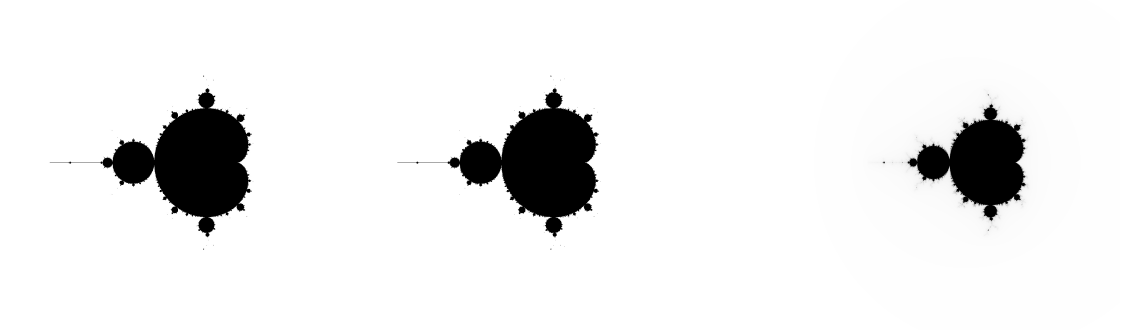
Eine weitere zukünftige Verbesserung könnte darin bestehen, den gesamten Prozess zu parallelisieren. Da die Berechnungen voneinander unabhängig sind, könnte man leicht alles parallel ausführen, um den Code noch schneller zu machen.

3 Korrektheit

In diesem Kapitel wird die Korrektheit unserer Implementierung analysiert. Unsere Implementierung liefert als Ausgabe eine .ppm Datei, wo das erzeugte Mandelbrot Set dargestellt wird. Damit wir die Korrektheit dieser Bilder analysieren können, brauchen wir andere Bilder von Mandelbrot Sets mit denselben Parametern um sie mit unserer Ausgabe 1a zu vergleichen. Dafür werden zum einen Bilder unserer Vergleichsimplementierung (V1) 1b (weitere Informationen dazu im Kapitel Performanzanalyse), sowie auch Bilder erzeugt vom Mandelbrot Viewer des MIT[1] 1c verwendet. Wie man hier auf den Bildern sehen kann, sind alle drei Bilder fast identisch. Die wenigen Unterschiede der Bilder vom Mandelbrot Viewer des MIT entstehen aus dem Grund, dass bei deren Implementierung das Bild beliebig vergrößert werden kann, was bei unseren Implementierungen noch nicht der Fall ist. Trotzdem ist die Struktur dieselbe und die gleichen Pixel wurden in alle Implementierungen als Teil des Mandelbrot-Sets interpretiert und gefärbt. Die Bilder wurden mit folgenden Parametern erzeugt:

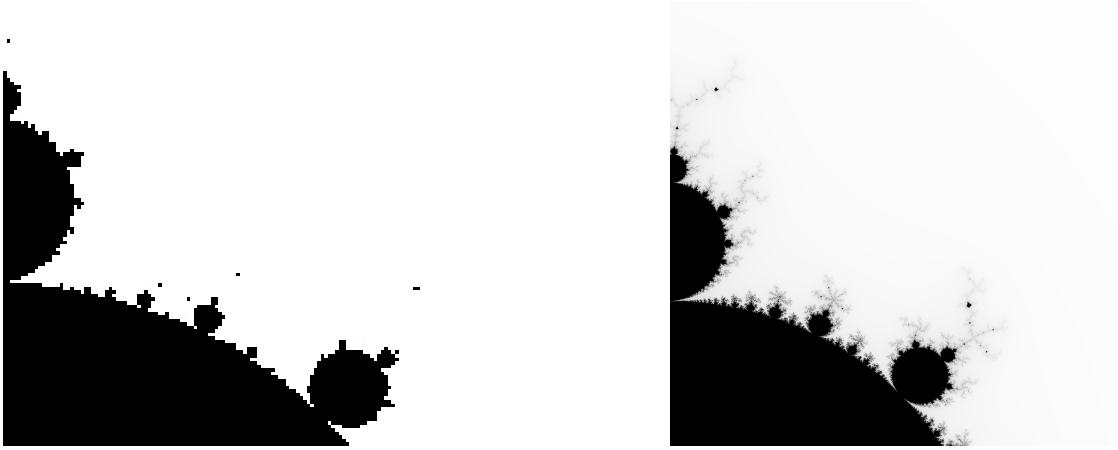
- width = 1000
- height = 1000
- Anzahl maximaler Iterationen n = 255

In der Abbildung 2 sind die vergrößerte Bilder zu sehen, wo wieder der Unterschied in Schärfe zu sehen ist, aber trotzdem die gleiche Struktur.



(a) V0 (b) V1 (c) Mandelbrot Viewer

Abbildung 1: Vergleich zwischen V0, V1 und Mandelbrot Viewer



(a) V0 (b) Mandelbrot Viewer

Abbildung 2: Vergleich zwischen V0 und Mandelbrot Viewer vergrößert

Außerdem kann man in den folgenden Bildern 3 den Einfluss, den die Anzahl an maximalen Iterationen n hat auf das erzeugte Bild. Bei sehr wenigen Iterationen ist das Bild logischerweise kein wirkliches Mandelbrot, sondern ähnelt nur die Struktur von einem. Je mehr mögliche Iterationen, desto schärfer ist das Bild, bis zu einer Iterationenzahl, die zu einem Bild mit einer schärferen Struktur führt.

(a) n = 10 (b) n = 10000

Abbildung 3: Vergleich n = 10 und n = 10000

4 Performanzanalyse

In diesem Kapitel wird die Performanz unserer Implementierung gemessen und mit unserer Vergleichsimplementierung V1 verglichen und anschließend analysiert. Die Vergleichsimplementierung ist basiert auf die Idee so wenig Iterationen wie möglich in den for-Schleifen zu erzeugen. Dazu werden zuerst anstatt einem Pixel pro Iteration acht Pixel pro Iteration berechnet und anschließend die verbliebenen Pixel. Die Performanz wurde anhand der benötigten Zeit für verschiedene representative maximale Iterationen n gemessen. Die Messungen wurden in einem Apple Macbook M1 mit 8 (4 Performance und 3 efficiency) Prozessorkernen mit 8GB RAM und 256GB SSD Laufwerkspeicher auf Mac OS 12.3.1 durchgeführt. Die Zeiten wurden von dem Mittelwert von 20 Messungen ausgerechnet. Die Messungen wurden mit der Option -B1 und folgenden Parametern durchgeführt:

- width = 1000
- height = 1000
- resolution = 4.0

Diese Zeiten dienen zum Vergleich zwischen den 2 Implementierungen. Die Zeiten der zwei Implementierungen sind in der Tabelle 1 und im Graphen 4 zu sehen.

n	V0[s]	V1[s]
10	0,052272	0,058026
100	0,142985	0,137066
500	0,450873	0,425736
1000	0,829926	0,782611
5000	3,891612	3,761461
10000	7,794141	7,220338
20000	15,266170	14,492853

Tabelle 1: Zeitmessungen V0 und V1

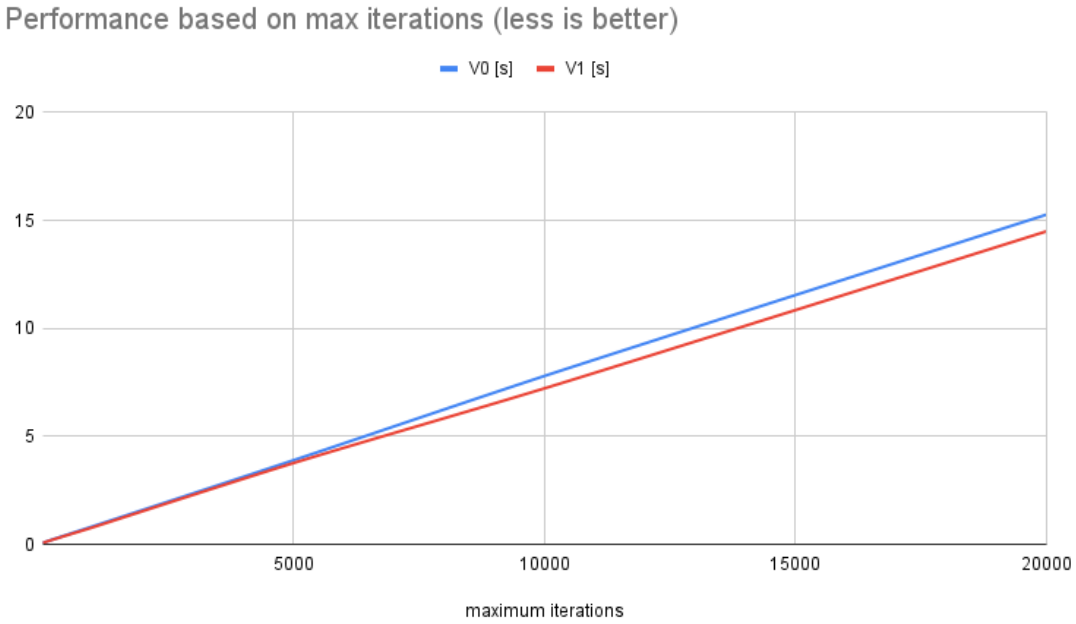


Abbildung 4: Zeitmessungen V0 und V1 anhand der Iterationen n (weniger ist besser)

Von diesen Messungen sieht man, dass die Vergleichsimplementierung ein bisschen schneller ist und mit zunehmenden Iterationen wird es noch schneller als unsere Implementierung. Aber wenn man im Graph 5 schaut, merkt man, dass diese Implementierung bei einer Höhe und Breite >4500 nicht mehr funktioniert und einen Segmentation Fault auslöst. Deswegen haben wir uns auch entschieden die ein bisschen langsamere aber dafür zuverlässige Alternative zu verwenden. Die maximalen Iterationen hier sind 255. Die anderen Parameter sind gleich.

Breite & Höhe	V0[s]	V1[s]
100	0,004211	0,005041
500	0,094416	0,087099
1000	0,263578	0,246931
1500	0,544145	0,507961
2500	1,434412	1,355402
3000	2,033422	1,943677
4000	3,617822	3,389774
4500	4,554346	(failed)
5000	5,598512	(failed)

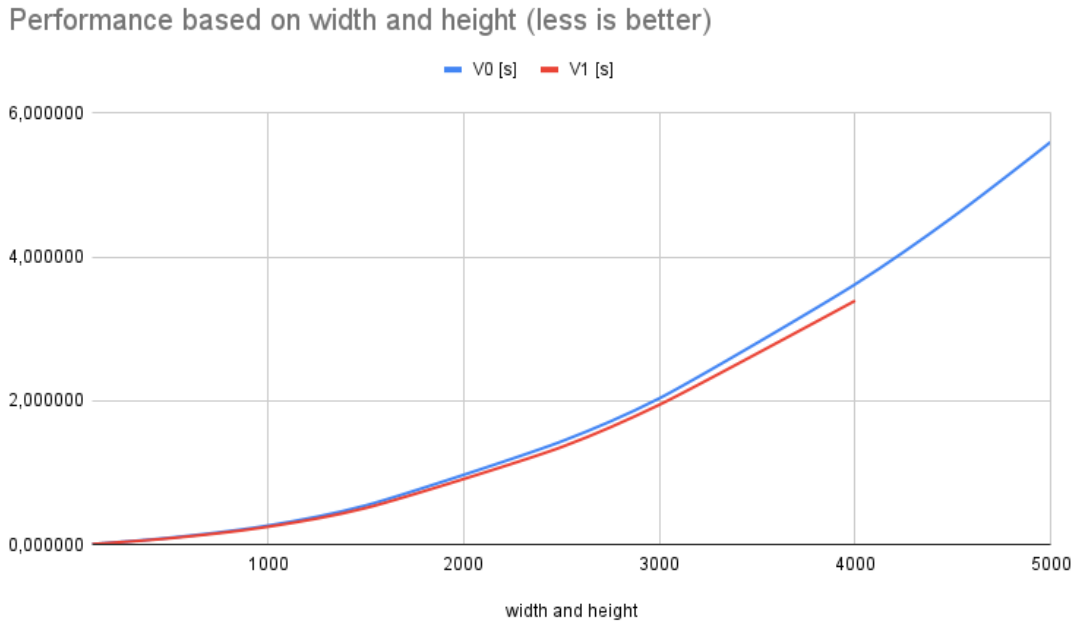


Abbildung 5: Vergleich zwischen V0 und V1 anhand von Höhe und Breite (weniger ist besser)

5 Zusammenfassung und Ausblick

Zusammenfassend wird im vorliegenden Projekt die Mandelbrot-Menge bei einem beliebigen Wert von c untersucht, einen Algorithmus in C entwickelt, der die Mandelbrot-Menge berechnet und als eine .ppm Datei grafisch dargestellt.

Folgende Fragestellungen lagen der Untersuchung zugrunde: In dem theoretischen Teil, wie kann man komplexe Zahlen in Graph darstellen und was sind die möglichen

Abbruchbedingungen? In der Implementierung, wie kann man die Performanz den Algorithmus verbessern?

In der endgültigen Implementierung wird der Realteil in Richtung der x-Achse und der Imaginärteil in Richtung der y-Achse (=imaginäre Achse) aufgetragen und anstatt eines Pixels pro Iteration acht Pixel pro Iteration berechnet, um die Performanz der Implementierung zu verbessern. Das Programm endet entweder, wenn die Anzahl der maximalen Iteration erreicht oder den Betrag der komplexen Zahl größer als 2 wird.

Weitere Untersuchungen können zum einen für die Farbdarstellung der Mandelbrot-Menge förderlich sein, weil die derzeitige Implementierung Schwarzweiß ist und die Zahlen, die nicht zur Mandelbrot-Menge gehören, nicht von den anderen unterscheiden lassen. Eine bessere Farbdarstellungsform kann sein: Die Zahlen, die nicht zur Mandelbrot-Menge gehören, werden abhängig von der Anzahl der Schritte, bis die Abbruchbedingung erfüllt ist, gefärbt.

Zusätzlich werden die Zahlen, die zur Mandelbrot-Menge gehören, abhängig von der Anzahl der Schritte gefärbt, die jede Zahl braucht, bis sie eine bestimmte Nähe zu ihren Häufungspunkten hat, sodass sich die Position kaum noch ändert.[2, 3]

In weiterer Untersuchung kann die Performanz der Implementierung weiter verbessert werden, indem man überprüft, ob ein beim Iterieren eines Pixels erreichter Punkt zuvor schon erreicht wurde. Allerdings kostet diese Implementierung mehr Speicherplatz, weil mindestens eine vorherige Iteration gespeichert werden soll.

Literatur

[1] David Eck. Mandelbrot viewer. `https://web.mit.edu/jorloff/www/chaosTalk/mandelbrot/davideck-js/MB.html`, visited 2022-07-22.

[2] García, Francisco; Ángel Fernández; Javier Barrallo; Luis Martín. Coloring dynamical systems in the complex plane. `https://en.wikipedia.org/wiki/Plotting_algorithms_for_the_Mandelbrot_set#:~:text=%22Coloring%20Dynamical%20Systems%20in%20the%20Complex%20Plane%22`, visited 21-01-2008.

[3] Jens Zumbrägel. Farbdarstellungen komplexer fraktale. `https://staff.fim.uni-passau.de/~zumbraegel/fractals/fracfarb.html`, visited 22-07-22.