

LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Praktikum Rechnerarchitektur

Parallele Berechnung der Mandelbrotmenge

Wintersemester 2018/19

Maximilian Frühauf

Tobias Klasuen

Florian Lercher

Niels Mündler

1 Einleitung

1.1 Didaktische Ziele

Die Leistung und Geschwindigkeit individueller Rechenkerns stagniert seit einigen Jahren. Moderne Computer erlangen einen Großteil ihrer erhöhten Rechenleistung seit einiger Zeit nur noch durch Parallelisierung. Da dies stets einen geringen Overhead verursacht werden gut durchdachte Konzepte zur Ausnutzung mehrerer Kerne immer wichtiger.

Dieses Projekt soll Nutzern ermöglichen intuitiv ein Gefühl dafür zu gewinnen, welche wichtige Rolle hierbei die Aufteilung der Last zwischen den Rechenknoten spielt.

1.2 Verwendung der Mandelbrotmenge

Die Mandelbrotmenge ist eine Teilmenge der komplexen Zahlen. Um sie zu berechnen wendet man folgende Formel wiederholt auf jede komplexe Zahl c an:

$$z_{n+1} = z_n^2 + c, \quad z_0 = 0 \quad (1)$$

In der Mandelbrotmenge befinden sich alle c , für die der Betrag von z_n für beliebig große n endlich bleibt. Wenn der Betrag von z nach einer Iteration größer als 2 ist, so strebt z gegen unendlich, das zugehörige c liegt also nicht in der Menge. Sobald $|z_n| > 2$ kann die Berechnung daher abgebrochen werden.

Um nun für eine beliebige Zahl zu bestimmen, ob diese in der Mandelbrotmenge liegt, müssen theoretisch unendlich viele Rechenschritte durchgeführt werden. Zur computergestützten Bestimmung werden die Rechenschritte nach einer bestimmten Iteration abgebrochen die Zahl als in der Menge liegend betrachtet.

Es handelt sich also um eine Berechnung, die sehr zeitaufwändig ist, wobei die benötigte Zeit durch Erhöhen der Iterationszahl beliebig erhöht werden kann. Zusätzlich ist die Berechnung für jede einzelne komplexe Zahl unabhängig von jeder anderen Zahl.

Diese Eigenschaften ermöglichen es, zweierlei Dinge zu kontrollieren:

- Die Dauer der Berechnung
- Die Aufteilung der Berechnung auf unterschiedliche Rechenkerne

Somit kann gesichert werden, dass eine wahrnehmbare Zeit (100-200 ms) zur Berechnung benötigt wird. Zudem können die Berechnungen frei auf verfügbare Rechenknoten verteilt werden, sodass für verschiedenste Aufteilungen eine Rechenzeit visualisiert werden kann.

1.3 Darstellung der Mandelbrotmenge

Komplexe Zahlen lassen sich auch grafisch darstellen, indem man sie in ein Koordinatensystem einträgt. Dabei entspricht die x-Koordinate dem Realteil und die y-Koordinate dem Imaginärteil der Zahl. Für das Projekt wird ein Ausschnitt des Bildschirms als

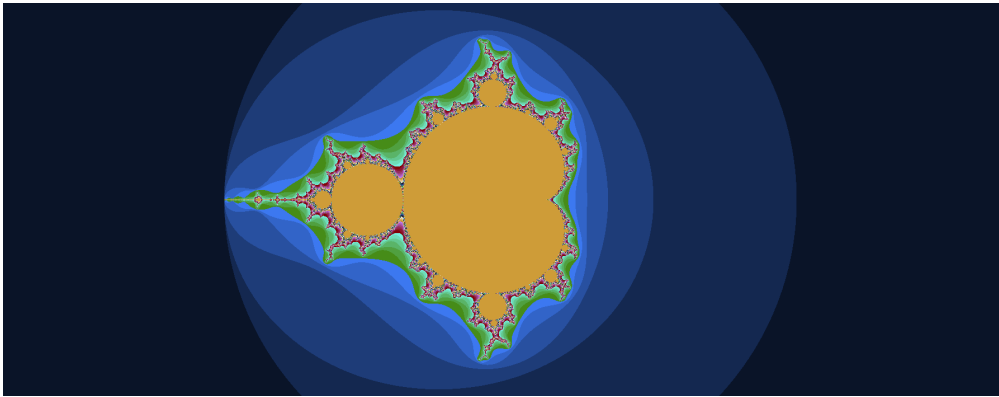


Abbildung 1: Die Mandelbrotmenge, visualisiert in einem Ausschnitt des komplexen Zahlenraumes.

zweidimensionale Darstellung des komplexen Raumes betrachtet und für jeden darin liegenden Punkt die Zugehörigkeit zur Mandelbrotmenge bestimmt. Dabei wird der Raum jedoch diskretisiert, indem jedem Pixel des Bildschirms die komplexen Koordinaten c der linken oberen Ecke zugeordnet werden.

Die grafische Darstellung der Mandelbrotmenge wird durch Einfärbung des zu c gehörigen Pixels erhalten. Die Zahl der benötigten Iterationen bis zum Abbruch der Berechnung bestimmt dabei die Farbe, sodass alle Pixel innerhalb der Menge und alle Pixel außerhalb jeweils gleichfarbig sind.

Das entstehende Fraktal ist aufgrund seiner Form auch als “Apfelmännchen” bekannt (siehe 1.3). Die Menge ist zusammenhängend, jedoch bilden sich an ihren Rändern viele kleine und sehr komplexe Formen, die visuell ansprechend sind. Es eignet sich daher gut, um optisch Interesse am Projekt zu wecken.

1.4 MPI

2 Problemstellung und Motivation

- Fachliche Spezifikation in Anhang
- NFRs in Einleitung schreiben

3 Dokumentation der Implementierung

- Sollte größter Teil werden
- 1. High level overview?
 2. Wie läuft das auf meinem System?
 3. Code Dokumentation / Entwicklerdokumentation

3.1 Lastbalancierung

Um die Mandelbrotmenge effizient parallel zu berechnen, muss die Last gleichmäßig auf die Worker verteilt werden. Die Aufgabe der Lastbalancierung besteht darin zu einer gegebenen Region und einer Anzahl von Workern eine solche Unterteilung in sogenannte Teilregionen zu finden. Wichtig dabei ist, dass der garantierte Teiler von Höhe und Breite der Teilregionen dem der angeforderten Region entspricht, da es sonst im Frontend zu Schwierigkeiten bei der Darstellung kommt. Die Klassenstruktur der Lastbalancierer entspricht dem Strategy-Pattern. So kann der Balancierer zur Laufzeit leicht gewechselt werden und auch die Erweiterung des Projekts um eine weitere Strategie gestaltet sich einfach. Was dabei genau beachtet werden muss findet sich im Teil Erweiterung.

Damit die Unterschiede zwischen guter und schlechter Lastverteilung deutlich werden, wurden hier verschiedene Strategien zur Lastbalancierung implementiert.

Naive Strategie

Bei der naiven Strategie (zu finden in den Klassen `NaiveBalancer` und `RecursiveNaiveBalancer`) wird versucht den einzelnen Workern etwa gleich große Teilregionen zuzuweisen. Dies geschieht allerdings ohne Beachtung der eventuell unterschiedlichen Rechenzeiten innerhalb der Teilregionen. Die naive Strategie wurde hier in einer nicht-rekursiven und einer rekursiven Variante implementiert.

Zur nicht-rekursiven Aufteilung wird zuerst die Anzahl der zu erstellenden Spalten und Zeilen berechnet. Dazu wird der größte Teiler der Anzahl der Worker bestimmt. Dieser gibt die Anzahl der Spalten an, das Ergebnis der Division ist die Anzahl der Zeilen. Damit ist sichergestellt, dass die Region in die richtige Menge von Teilregionen unterteilt wird.

Als nächstes wird die Breite und Höhe der Teilregionen berechnet. Hierbei ist wichtig, dass der garantierte Teiler erhalten wird. Die Breite berechnet sich also durch:

$$\frac{region.width}{region.guaranteedDivisor * nodeCount} * region.guaranteedDivisor$$

Dabei ist `nodeCount` die Anzahl der Worker und `region` die zu unterteilende Region. Es ist zu beachten, dass es sich hier um eine Ganzzahldivision handelt, deren Rest angibt,

wie viele Teilregionen um *region.guaranteedDivisor* breiter sind. Die Höhe berechnet sich analog.

Bevor die eigentliche Aufteilung beginnt werden noch die Deltas für Real- und Imaginärteil bestimmt. Diese geben an, wie groß der Bereich der komplexen Ebene ist, den ein Pixel der Region überdeckt. Die Deltas können über Methoden der Klasse *Fractal* berechnet werden.

Zur Aufteilung wird nun mittels zweier verschachtelter Schleifen über die Zeilen und Spalten iteriert. Die benötigten Start- und Endpunkt der Teilregionen (also die linke obere Ecke und die rechte untere Ecke auf der komplexen Ebene) können nun mithilfe der Schleifenzähler und der Deltas bestimmt werden. Zusätzlich wird noch der vertikale und horizontale Offset der Teilregion vom Startpunkt der Eingaberegion abgespeichert. Diese Information ermöglicht es die Region im Frontend einfach anzuzeigen.

Falls die aktuell betrachtete Teilregion zu den Breiteren und/oder Höheren gehört, müssen alle Werte entsprechend angepasst werden. Die letzte Teilregion einer Spalte bzw. Zeile wird immer so gewählt, dass sie auf jeden Fall mit dem Rand der Eingaberegion abschließt.

Die Teilregionen werden in einem Ergebnisarray gespeichert, welches dann zurückgegeben wird.

Strategie mit Vorhersage

Erweiterung

4 Ergebnisse / Evaluation

- Skalierbarkeitsgraph
- Wie gut ist SIMD / OpenMP / MPI / Mischformen?
- Frontend Overhead messen

5 Zusammenfassung

- Zusammenfassung
- Ausblick