

Dokumentation Mandelbrotmenge
Programmierung verteilter Systeme
*Valine Richter s85875, Anastasia Suglobow s84401,
Marlene Pannoscha s83814*

Inhaltsverzeichnis

1. Einleitung	1
2. Architekturübersicht	2
2.1. Architekturkomponenten nach MVP & verteilten RMI-Design	2
2.2. Architekturdiagramm	2
3. Ermittlung des Verteilungsgewinns	4
3.1. Messmethodik	4
3.2. Zeitmessung & Auswertung	4
3.2.1. Diagramm zur Laufzeit und Verteilung	4
3.2.2. Darstellung von Skalierung und Effizienz	4
3.2.3. Hinweis zur Reproduzierbarkeit	5
3.3. Interpretation der Ergebnisse	5
4. Teamarbeit und Rollenverteilung	7
5. Fazit und Ausblick	8
6. Quellen	9

1. Einleitung

Die Mandelbrot-Menge stellt ein prototypisches Beispiel für ein mathematisches Fraktal dar, das durch einfache iterative Berechnungen erzeugt lässt. Durch die geforderte Rechenleistung und Ressourcennutzung wird dieses Projekt mit verteilten Systemen realisiert.

Zur effizienten Lösung dieses Problems bietet sich der Einsatz von verteilten Systemen an, dass mehrere miteinander verbundene Rechner nutzt, um Aufgaben parallel auszuführen und somit Rechenzeiten deutlich zu verkürzen. In diesem Projekt wird ein verteiltes System zur Berechnung und Darstellung der Mandelbrot-Menge entworfen und implementiert. Ziel ist es, die Berechnungen auf mehrere Knoten zu verteilen, die parallel einzelne Bildausschnitte generieren. Dadurch soll sowohl die Skalierbarkeit als auch die Effizienz der Mandelbrot-Berechnung demonstriert werden.

Diese Arbeit beschreibt zunächst die konzipierte und implementierte Systemarchitektur, analysiert anschließend die Skalierbarkeit anhand empirisch erhobener Messwerte und reflektiert die im Rahmen der Teamarbeit gewonnenen Erfahrungen. Die Ergebnisse zeigen, dass durch die Parallelisierung der Berechnung eine signifikante Leistungssteigerung erzielt werden kann, was die Effizienz und Skalierbarkeit des gewählten Ansatzes unterstreicht.

2. Architekturübersicht

In diesem Abschnitt wird die Systemarchitektur beschrieben und mittels geeigneter graphischer Darsteckung erlŠutert.

2.1. Architekturkomponenten nach MVP & verteilten RMI-Design

Das System basiert auf einer strukturierten Architektur, die aus zwei Hauptkomponenten besteht: einem Client, der dem Model-View-Presenter (MVP)-Muster folgt, und einem verteilten Server-System nach dem Master-Worker-Prinzip. Die Kommunikation erfolgt Ÿber Remote Method Invocation (RMI).

Die Architketur verfolgt das Ziel, BenutzeroberflŠche (View), die Anwendungslogik (Presenter) und die Datenhaltung bzw. Verarbeitung (Model) strikt voneinander zu trennen. Diese Trennung gewŠhrleistet sowohl eine klare Strukturierung der ZustŠndigkeiten im Client als auch eine effiziente, parallele Verarbeitung rechenintensiver Aufgaben auf der Serverseite wie der Darstellung der Mandelbrotmenge.

2.2. Architekturdiagramm

Die Systemarchitektur der Anwendung nutzt zwei Software-Architekturmuster: Auf der Client-Seite kommt das Model-View-Presenter-(MVP)-Modell zum Einsatz, wŠhrend die Serverseite das Master-Worker Prinzip realisiert. Das zugehŠrige Architekturdiagramm verdeutlicht diese Aufteilung.

Auf der linken Seite befinden sich die Client-Komponenten des MVP-Modells. Die View ist zustŠndig fŠr die grafische Darstellung sowie die Interaktion mit dem Benutzer. Sie leitet Eingaben direkt an den Presenter weiter. Der Presenter vermittelt zwischen View und Model. Er verarbeitet Benutzereingaben, sendet Anfragen per Remote Method Invocation (RMI) an den Master (Server) und Ÿbernimmt zusŠtzlich die Aufbereitung der RŠckmeldungen. Nach der Antwort vom Master aktualisiert der Presenter die View entsprechend. Das Model dient der Datenhaltung auf Client-Seite und Ÿbernimmt ggf. einfache Datenverarbeitungsschritte, bevor es strukturierte Informationen an den Presenter zurŠckgibt.

In der Mitte des Diagramms ist der Master-Server dargestellt. Dieser empfŠngt Anfragen vom Presenter des Clients und teilt komplexere Aufgaben in kleinere, parallel verarbeitbare Teilaufgaben. Diese Aufgaben werden gezielt an vier Worker verteilt, die auf der rechten Seite des Diagramms abgebildet sind.

Jeder Worker bearbeitet die ihm zugewiesene Teilaufgabe und sendet das Ergebnis zusammen und sendet sie in strukturierter Form an den Client zurŠck.

Die KommunikationsflŠsse im Diagramm sind farblich differenziert:

- ¥ Orange Pfeile zeigen die Aufgabenverteilung vom Master an die Worker,
- ¥ Lila Pfeile reprŠsentieren den RŠckweg der Ergebnisse vom Server an den Client,
- ¥ GrŠne Pfeile markieren interne GUI-Aktualisierungen durch den Presenter.

Der gesamte Ablauf lässt sich wie folgt zusammenfassen. Eine Benutzeraktion löst im Client einen Ereignisfluss aus, den der Presenter entgegennimmt. Er übermittelt die Aufgabe per RMI an den Master-Server. Der Master verteilt die Aufgabe auf mehrere Worker, die ihre Ergebnisse zurückmelden. Der Master aggregiert die Resultate und sendet die Antwort an den Client. Dort verarbeitet der Presenter das Ergebnis und aktualisiert die grafische Oberfläche über die View.

Durch diese Architektur wird eine klare Trennung von Präsentationslogik, Datenverarbeitung und Benutzerinteraktion erreicht. Gleichzeitig erlaubt das Master-Worker-Modell eine horizontale Skalierung der Rechenleistung über mehrere Worker-Prozesse hinweg, was die Effizienz bei komplexen Berechnungen deutlich erhöht.

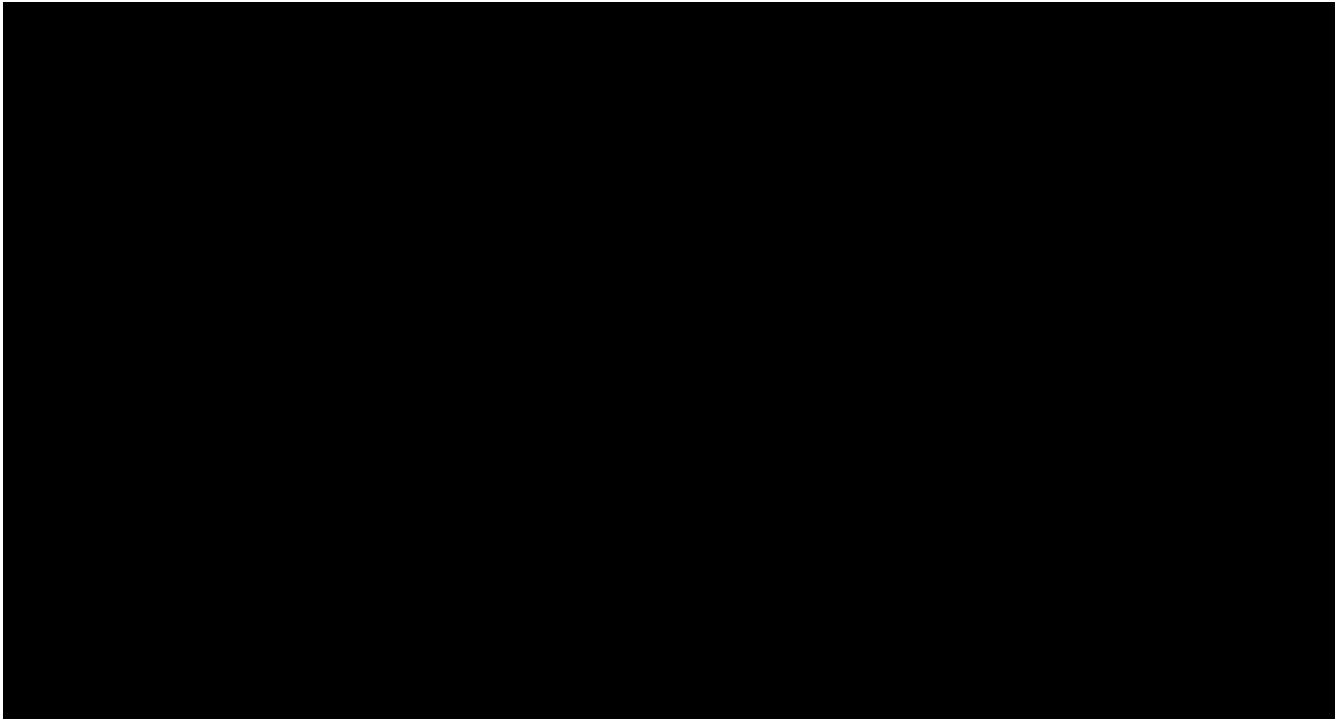


Abbildung 1. Architektur

3. Ermittlung des Verteilungsgewinns

3.1. Messmethodik

Zur Bewertung der Leistungssteigerung durch Parallelisierung wurde die Rechenzeit für einen festen Bildausschnitt der Mandelbrot-Menge unter verschiedenen Verteilungsszenarien analysiert. Die zu berechnende Fläche wurde in 1200 Chunks unterteilt, die unterschiedlichen Worker-Knoten zugewiesen wurden. Ziel war es, den Einfluss der Lastverteilung auf die Gesamtlaufzeit zu untersuchen und den Effekt zunehmender Parallelisierung zu dokumentieren.

3.2. Zeitmessung & Auswertung

3.2.1. Diagramm zur Laufzeit und Verteilung

Abbildung 2. Verarbeitungszeit und Chunkverteilung

! Beschreibung:

- ! Kombinierte Darstellung der Bearbeitungsdauer in Millisekunden und der pro Worker verarbeiteten Chunks je Szenario (104 Worker).
- ! Gestapeltes Balkendiagramm, das die Aufteilung der Chunks auf die einzelnen Worker für jedes Szenario veranschaulicht.

3.2.2. Darstellung von Skalierung und Effizienz

Im Folgenden wird die Auswertung der Laufzeit und der Verteilung der Rechenlast auf verschiedene Worker dargestellt. Die Berechnungsaufgabe bestand in jedem Fall aus 1200 Chunks.

1. Einzelner Worker

! Verteilung:

" worker_1: 1200 Chunks (100%)

! Gesamtlaufzeit: 30.920 ms \$ Diese Konfiguration dient als Referenzwert. Die gesamte Berechnung wurde von einem einzigen Worker ausgeführt und stellte die längste Rechenzeit dar.

2. Zwei Worker

! Verteilung:

" worker_1: 604 Chunks (50,33%)

" worker_4: 596 Chunks (49,67%)

! Gesamtlaufzeit: 15.613 ms \$ Durch den Einsatz eines zweiten Workers konnte die Rechenzeit nahezu halbiert werden.

3. Drei Worker

! Verteilung:

" worker_1: 390 Chunks (32,50%)

" worker_2: 409 Chunks (34,08%)

" worker_4: 401 Chunks (33,42%)

! Gesamtlaufzeit: 10.572 ms \$ Mit drei Workern wurde eine weitere Reduktion der Laufzeit erreicht. Die Lastverteilung war nahezu gleichmäßig.

4. Vier Worker

! Verteilung:

" worker_1: 297 Chunks (24,75%)

" worker_2: 296 Chunks (24,67%)

" worker_3: 296 Chunks (24,67%)

" worker_4: 311 Chunks (25,92%)

! Gesamtlaufzeit: 8.212 ms \$ Die beste Performance wurde mit vier Workern erzielt. Die Arbeitslast war nahezu gleichmäßig verteilt.

3.2.3. Hinweis zur Reproduzierbarkeit

Alle Diagramme wurden mit `matplotlib` in Python erstellt. Die zugrunde liegenden Messwerte wurden aus dem Projekt zur verteilten Berechnung der Mandelbrot-Menge entnommen.

3.3. Interpretation der Ergebnisse

¥ Von 30.920ms (1 Worker)

¥ auf 15.613ms (2 Worker)

¥ auf 10.572ms (3 Worker)

¥ bis auf 8.212ms (4 Worker)

Die Messergebnisse verdeutlichen, dass sich die Verarbeitung der Mandelbrot-Menge durch

verteiltes Rechnen signifikant beschleunigen lässt. Bereits mit zwei Workern lässt sich die Rechenzeit nahezu halbieren. Eine gleichmäßige Lastverteilung führt zu hoher Effizienz, insbesondere wenn die beteiligten Worker eine ähnliche Leistung aufweisen.

Interessant ist, dass die geringste Verarbeitungszeit nicht durch fokussierte Lastverteilung auf den leistungsstärksten Worker, sondern durch eine nahezu symmetrische Aufgabenzuteilung erreicht wurde. Dies weist darauf hin, dass bei überschaubarer Problemgröße und ähnlicher Netzwerklatenz eine gleichmäßige Verteilung eine effektive Strategie darstellt.

Die Parallelisierung zeigte in allen Fällen klare Skalierungsvorteile. Zwischen Einzelbetrieb und Vierfach-Parallelisierung wurde eine Reduktion der Laufzeit um rund 73,4% erreicht. Damit zeigt das System eine gute horizontale Skalierbarkeit im gegebenen Setup.

4. Teamarbeit und Rollenverteilung

Die Zusammenarbeit im Team verlief durchweg effektiv. Wir haben den Beleg gemeinsam in regelmäßigen Treffen erarbeitet, bei denen wir unsere Zwischenergebnisse besprochen, Aufgaben verteilt und technische Fragen geklärt haben. Uns war es leider nur selten möglich, den Beleg vor Ort an der HTWD zu bearbeiten. Da zwei Teammitglieder momentan ihr Vollzeit Praktikum absolvieren.

Die Aufgaben wurden sinnvoll aufgeteilt: Marlene Pannoscha war hauptsächlich für die Gestaltung und Implementierung der grafischen Benutzeroberfläche (Client) zuständig, Valine Richter übernahm die Entwicklung des Masters, und Anastasia Suglobow konzentrierte sich auf die Worker-Komponente. Trotz der klaren Verteilung unterstützten wir uns gegenseitig regelmäßig, gaben uns Hilfestellung und sorgten dafür, dass alle auf dem gleichen Stand bleiben. So konnte jedes Teammitglied jederzeit den Gesamtzusammenhang nachvollziehen. Die Architektur und andere Feinarbeiten wurden zusammen erarbeitet.

Der regelmäßige Austausch, das gegenseitige Verständnis und die strukturierte Aufgabenteilung trugen maßgeblich zu einem reibungslosen Ablauf und guten Fortschritten bei.

5. Fazit und Ausblick

Das Projekt zeigt, dass verteilte Verarbeitung eine effektive Lösung zur Bewältigung rechenintensiver Aufgaben wie der Berechnung der Mandelbrot-Menge darstellt. Die durchgeführten Messungen belegen, dass die Verarbeitungsgeschwindigkeit mit zunehmender Anzahl von Worker-Knoten deutlich gesteigert werden kann und bei gleichzeitiger gleichmäßiger Auslastung der Ressourcen. Die Systemarchitektur ermöglicht eine saubere Trennung von Steuerlogik, Benutzeroberfläche und Berechnungskomponenten und lässt sich flexibel an unterschiedliche Hardwarekonfigurationen anpassen.

Besonders hervorzuheben ist die effiziente Aufgabenverteilung, die es ermöglicht, die Rechenlast auf Basis verfügbarer Ressourcen gleichmäßig zu verteilen. Die Skalierung ist nachvollziehbar und stabil und bereits durch den Einsatz eines zweiten Workers konnte die Rechenzeit nahezu halbiert werden. Mit vier parallel arbeitenden Workern wurde eine Reduktion um über 70% gegenüber der Einzelverarbeitung erreicht. Für die Zukunft sind folgende Erweiterungen möglich:

Für weiterführende Arbeiten ergeben sich folgende Perspektiven:

- **Dynamische Lastverteilung:** Bisher wurde die Chunk-Verteilung statisch vorgenommen. Zukünftig könnte ein dynamisches Scheduling auf Basis der tatsächlich gemessenen Ausführungszeiten eine noch bessere Ausnutzung heterogener Systeme ermöglichen.
- **Visualisierung & Benutzerfreundlichkeit:** Eine interaktive Darstellung der berechneten Fraktale sowie eine visuelle Rückmeldung über den Fortschritt könnten die Anwendung deutlich intuitiver gestalten.

Insgesamt liefert das Projekt eine solide Grundlage für weitere Experimente und Optimierungen im Bereich verteilter Verfahren und demonstriert anschaulich das Potenzial moderner vernetzter Rechenarchitekturen.

6. Quellen

- ¥ <https://de.wikipedia.org/wiki/Mandelbrot-Menge> Ð Allgemeine Einführung und mathematische Grundlagen zur Mandelbrot-Menge.
- ¥ <https://www.youtube.com/watch?v=LqbZpur38nw&t=260s> Ð Visualisierung der Komplexität und Zoomtiefe der Mandelbrot-Menge (YouTube, 3Blue1Brown).
- ¥ https://www.youtube.com/watch?v=_78kGnsYXLc Ð Erläuterung der zugrunde liegenden Mathematik und Struktur (YouTube, Numberphile).