Dokumentation Mandelbrotmenge
Programmierung verteilter Systeme

Valine Richter s85875, Anastasia Suglobow s84401,
Marlene Pannoscha s83814

Inhaltsverzeichnis

1. Einleitung

Die Mandelbrot-Menge ist ein klassisches Beispiel für ein mathematisches Fraktal, das durch einfache iterative Berechnungen erzeugt wird, jedoch eine enorme Rechenleistung erfordern kann, insbesondere bei hoher Auflösung und Detailtiefe. Mit zunehmender Komplexität stoßen Einzelrechner schnell an ihre Leistungsgrenzen. Um dieses Problem effizient zu lösen, bietet sich der Einsatz verteilter Systeme an.

Ein verteiltes System nutzt mehrere miteinander verbundene Rechner, um Aufgaben parallel auszuführen und somit Rechenzeiten deutlich zu verkürzen. In dieser Arbeit (oder 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.

Neben der technischen Umsetzung wird auch die Kommunikation zwischen den Komponenten, das Lastverteilungskonzept sowie die Robustheit des Systems betrachtet. Die Ergebnisse zeigen, dass durch Parallelisierung eine signifikante Beschleunigung der Fraktalberechnung möglich ist.

2. Architekturübersicht

In diesem Abschnitt wird die Systemarchitektur beschrieben.

2.1. Architekturkomponenten nach MVP & verteilten RMI-Design

Das System besteht aus einem Client, der dem MVP-Architekturmuster (Model-View-Presenter) folgt und einem verteilten Server-System nach dem Master-Worker-Prinzip. Die Kommunikation erfolgt typischerweise über RMI (Remote Method Invocation).

Ziel der Architektur: Trennung von Benutzeroberfläche, Anwendungslogik und Hintergrundverarbeitung + Auslagerung rechenintensiver Aufgaben an mehrere Worker zur besseren Skalierung und Parallelisierung

2.2. Architekturdiagramm

Client (MVP-Modell)

- View
  - zuständig für grafische Darstellung und Benutzerinteraktion
  - gibt Eingaben an Presenter weiter
- Presenter
  - vermittelt zwischen View und Model
  - schickt Anfragen an den Master via RMI
  - verarbeitet Antworten und aktualisiert die View
- Model
  - hält interne Daten bzw. verarbeitet Ergebnisse vom Server
  - gibt strukturierte Daten an den Presenter zurück

Server: Master-Worker-Modell

- Master
  - empfängt Aufgaben vom Client
  - teilt komplexe Aufgaben in kleinere Teilaufgaben auf
  - weist Aufgaben gezielt an verfügbare Worker zu
  - sammelt erledigte Aufgaben ein und sendet das Ergebnis zurück an den Client
- Worker (1-4)
  - bearbeitet Teilaufgaben, die vom Master delegiert wurden
  - melden Ergebnisse an den Master zurück

saubere Trennung von Präsentation, Logik und Verarbeitung + Skalierbarkeit

durch parallele Worker

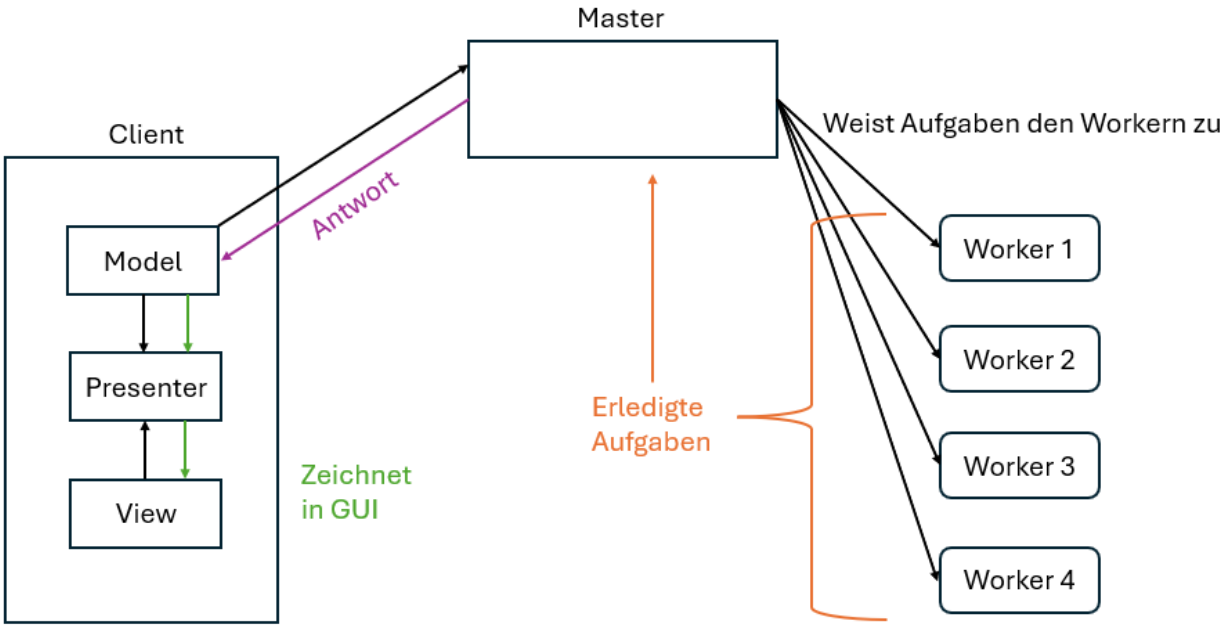

Abbildung 1. Architekture

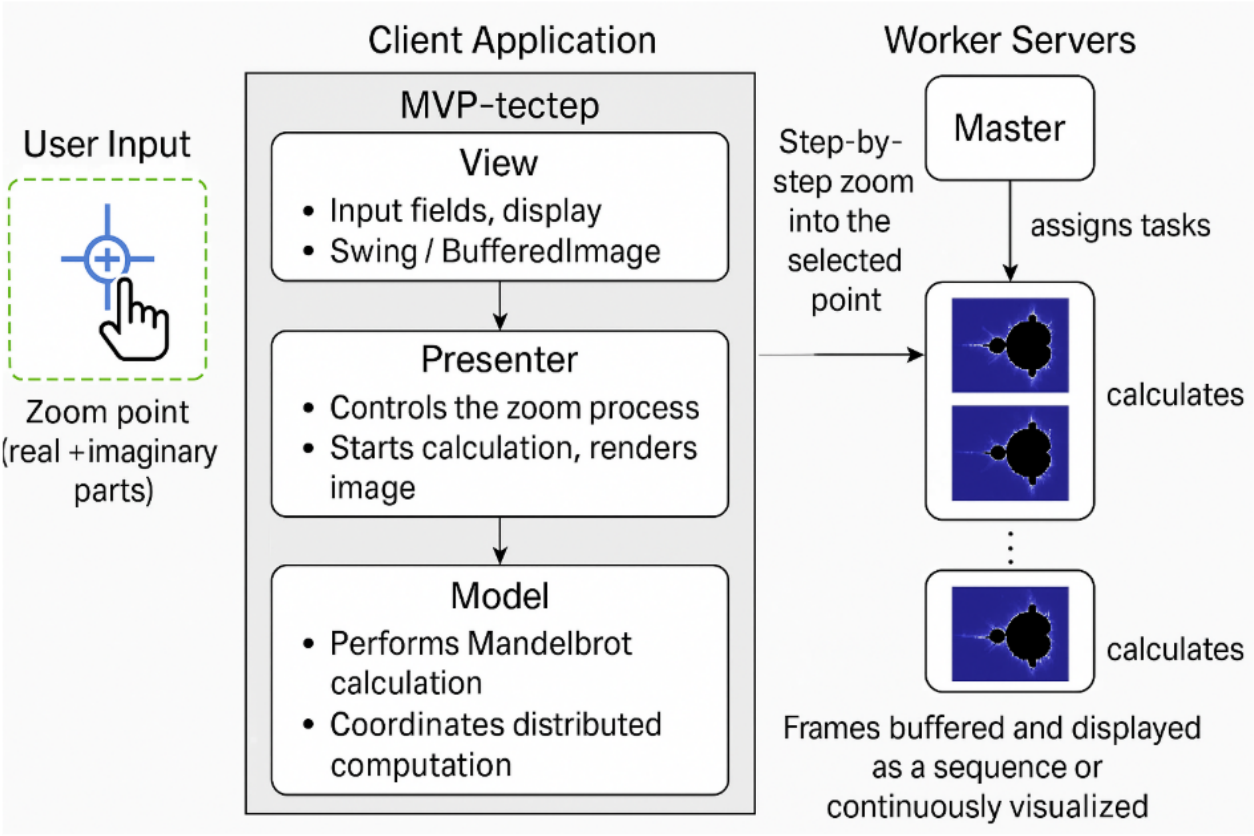

Abbildung 2. Architekture 2

3. Ermittlung des Verteilungsgewinns

3.1. Messmethodik

Zur Bewertung der Leistungssteigerung durch Parallelisierung wurde die Laufzeit zur Berechnung eines fixen Bildausschnitts der Mandelbrot-Menge unter verschiedenen Verteilungsszenarien gemessen. Die zu berechnende Fläche wurde dabei in **1200 Chunks** unterteilt, die einzelnen Worker-Knoten zur Berechnung zugewiesen wurden.

Die Testumgebung bestand aus folgenden Hardware-Komponenten:

- **worker_1**: Intel Pentium Silver (geringste Leistung)
- **worker_2**: ARM Cortex-A76 (mittlere Leistung)
- **worker_3**: Intel Core i7, 8. Generation (höchste Leistung)

Gemessen wurde jeweils die **Gesamtlaufzeit** vom Start der Berechnung bis zur vollständigen Fertigstellung aller Chunks.

3.2. Zeitmessung & Auswertung

3.2.1. Tabellen/Diagramme zur Laufzeit

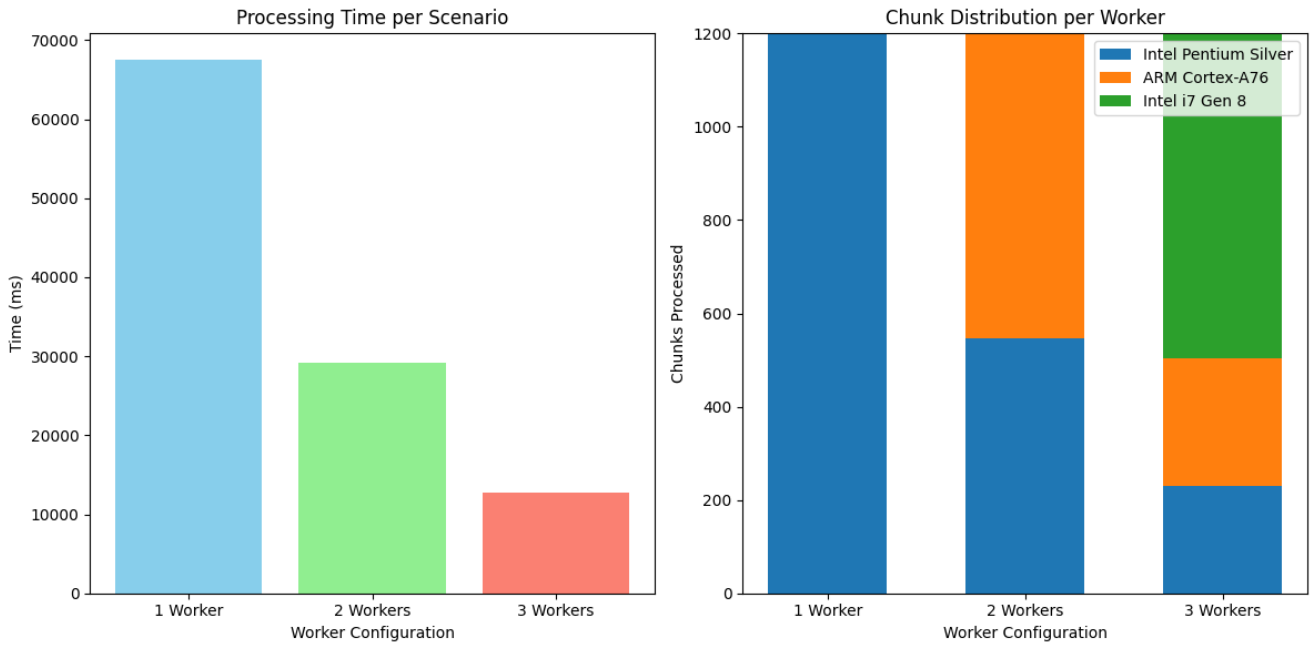

Abbildung 3. Diagramm 1

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 Aufgabe bestand aus der Bearbeitung von insgesamt 1200 Chunks.

1. **Einzelner Worker (Intel Pentium Silver)** Bei Ausführung mit nur einem Worker (`worker_1 = Intel Pentium Silver`) betrug die Bearbeitungszeit **67481 ms**. Alle 1200 Chunks wurden von diesem Worker verarbeitet. → Diese Konfiguration zeigt die langsamste Laufzeit und dient als Referenzwert.
2. **Zwei Worker (Intel Pentium Silver & ARM Cortex-A76)** Bei der parallelen Ausführung mit zwei Workern (`worker_1` und `worker_2`) verringerte sich die Bearbeitungszeit auf **29229 ms**. Die Last wurde aufgeteilt:
    - `worker_1`: 547 Chunks (≈45,58%)
    - `worker_2`: 653 Chunks (≈54,42%) → Die Verarbeitungszeit sank signifikant um ca. **56,7%**, was die Effektivität der Parallelisierung unterstreicht.
3. **Drei Worker (Intel Pentium Silver, ARM Cortex-A76 & Intel i7 Gen 8)** Mit drei Workern sank die Zeit weiter auf **12771 ms**. Die Verteilung war:
    - `worker_1`: 229 Chunks (≈19,08%)
    - `worker_2`: 274 Chunks (≈22,83%)
    - `worker_3`: 697 Chunks (≈58,08%) → Die leistungsstärkste CPU (`worker_3`) übernahm den Großteil der Arbeit. Die Laufzeit wurde gegenüber der Einzel-Worker-Ausführung um ca. **81% reduziert**.

3.3. Interpretation der Ergebnisse

Die Ergebnisse zeigen, dass die Verteilung der Rechenlast auf mehrere Worker zu einer deutlichen Verbesserung der Gesamtlaufzeit führt. Besonders deutlich wird der Effekt beim Einsatz eines leistungsstarken Prozessors (Intel i7 Gen 8), der den größten Anteil der Arbeit übernimmt. Die Skalierung funktioniert effizient, solange eine intelligente Lastverteilung basierend auf der Leistungsfähigkeit der einzelnen Worker erfolgt.

4. Teamarbeit und Rollenverteilung

Die Entwicklung des Projekts erfolgte in enger Zusammenarbeit und wurde arbeitsteilig durchgeführt. Die Aufgabenverteilung gestaltete sich wie folgt:

5. Fazit und Ausblick

Das Projekt hat eindrucksvoll gezeigt, dass verteilte Verarbeitung ein wirkungsvoller Ansatz zur Beschleunigung komplexer Rechenprozesse ist. Die entwickelte Architektur erlaubt eine effiziente Aufteilung der Aufgaben und nutzt die vorhandenen Ressourcen optimal aus. Durch intelligente Lastverteilung werden alle verfügbaren Einheiten bestmöglich ausgelastet. Das System reagiert flexibel auf unterschiedliche Ausführungsumgebungen und liefert auch bei wachsender Datenmenge stabile und überzeugende Ergebnisse.

Besonders hervorzuheben ist die Fähigkeit der Anwendung, Aufgaben dynamisch und lastabhängig zu verteilen, sodass die Leistungsfähigkeit der beteiligten Komponenten maximal genutzt wird. Die Skalierbarkeit und Effizienz des Systems belegen die Praxistauglichkeit und das Potenzial für weiterführende Einsätze im Bereich verteilter Systeme.

Für die Zukunft sind folgende Erweiterungen geplant:

- **Erhöhte numerische Genauigkeit:** Der derzeitige Einsatz von `double`-Werten bei der Mandelbrot-Berechnung ist in seiner Präzision begrenzt. Um feinere Strukturen im Fraktal und tiefere Zoomstufen darzustellen, soll künftig auf höhere Genauigkeit umgestellt werden – etwa durch Verwendung von `BigDecimal` oder eigener Gleitkommaarithmetik.
- **Monitoring und Analyse:** Die Einführung eines Monitoring-Systems zur Laufzeitüberwachung soll die Analyse von Ressourcenverbrauch und Performance erleichtern.
- **GPU-Unterstützung:** Zur weiteren Beschleunigung der Berechnungen ist geplant, in zukünftigen Versionen die Unterstützung von Grafikprozessoren (GPUs) zu integrieren. Durch deren massive Parallelverarbeitungsfähigkeit kann die Rechenzeit insbesondere bei sehr hohen Zoomstufen erheblich reduziert werden.

6. Anhang