ParaQuest

Alisa Dammer
Sven-Hendrik Haase

7. Oktober 2013

Inhaltsverzeichnis

1 Einführung

ParaQuest ist eine hochparallele Simulation von Kreaturen in einer 2D-Welt mit einem simplen Kampf- und Kollisionssystem. Die technische Umsetzung erfolgte mit C++11, cereal (für die Serialisierung) und MPI (für die Prozesskommunikation) sowie Qt (für die Visualisierung).

2 Problemstellung

Es soll eine Simulation erstellt werden, deren Berechnung per MPI auf mehrere Prozessen verteilt stattfinden kann. Simulationsgegenstand ist hierbei eine 2D-Kachelwelt voller Kreaturen, die sich bewegen können und kämpfen, wenn mehrere von ihnen auf dem gleichen Feld stehen. Dabei soll ein einfaches Kampfsystem mit Trefferpunkten, Angriffsstärke und Ausweichchance zum Einsatz kommen.
Zudem soll es mehrere verschiedene Spezies von Kreaturen geben: Goblin, Hobbit, Orc, Elf, Dwarf, Human. Jede Spezies hat andere Werte der Attribute Strength und Agility. Diese Attribute sind entscheiden bei der Kampfsimulation. Jede Kreature kann sich pro Welttick um ein Feld bewegen. Einige Felder sind durch Steine gesperrt. Auf diesen Feldern kann sich keine Kreatur befinden. Es soll zudem eine schicke Visualisierung der Welt erstellt.

3 Lösungsansatz

3.1 Simulation

Die Welt wird als sparse Matrix umgesetzt, da sich nicht in jedem Feld immer eine Kreatur befindet. Jede Kreatur bekommt also einfach Ganzzahlkoordinaten als Attribut sowie eine global eindeutige Id.
Am Anfang der Simulation wird die Welt im Hauptprozess populiert und dann an die anderen Prozesse verteilt. Nach jedem Tick wird von jedem Prozess der jeweils eigene Teil der Welt gedumped. Dieser Dump dient als Resultat des Simulationsschrittes und wird später für die Visualisierung benötigt.

Eine Kreatur hat folgende simulationsrelevante Attribute:

```
spezies
position
strength
```

`agility`
`hitpoints`

Die Welt hat folgende simulationsrelevante Attribute:

`size`
`creatures-list`

Das gesamte Programm wurde programmatisch mit C++11 umgestzt. Für die Serialisierung und die Erstellung der Dumps kommt cereal zum Einsatz. Für die Interprozesskommunikation wird MPI eingesetzt. Qt wird für die Visualisierung verwendet.

Die Simulation hat folgenden Ablauf:

1. Welt im Hauptprozess bevölkern
2. bevölkerte Welt an andere Prozesse aufteilen und senden
3. in jedem Prozess die Welt für n Ticks simulieren

Im Detail geschieht während der Simulation der Welt folgendes:

1. alle Kreaturen bewegen
2. Kreaturen an den Rändern an Nachbarn senden bzw. von Nachbarn empfangen
3. Kollisionen finden und Kreaturen kämpfen lassen
4. Verlierer und gesendete Kreaturen aus sich selber löschen
5. eigene Welt dumpen

Ein Dump der Welt hat die Aufgabe, als inkrementelles Simulationsresultat des jeweiligen Simulationsschrittes zu dienen. Es wird nach folgendem Schema benannt:

`<knotenname>-<knotenrank>_tick-<simulationsschritt>.log`

Der Inhalt besteht aus einer binär serialisierten Liste aller Kreaturen dieses Prozesses und ihrer Attribute. Alle diese Logs werden später in der Visualisierung wieder zu einem Gesamtresultat zusammengesetzt.

Das Simulationsprogramm nimmt folgende Parameter:

- -h zeigt die Hilfe an
- -r <seed> setzt einen gewählten Randomseed
- -s <size> setzt die Weltdimensionen nach size x size
- -c <count> setzt die maximale Anzahl der Startkreaturen
- -t <ticks> setzt die Anzahl der Simulationsschritte
- -o <count> setzt die Anzahl der gesperrten Felder

3.2 Visualisierung

Die Visualisierung erlaubt es dem Benutzer per Schiebeelemente, sich verschiedene Ausschnitte der Welt anzuschauen. Auf der rechten Seite befindet sich der Schieberegler für den Zoomlevel, auf der unteren Seite befindet sich ein Schieberegler für den Simulationsschritt. Man kann den Ausschnitt bewegen, indem man klickt und zieht.

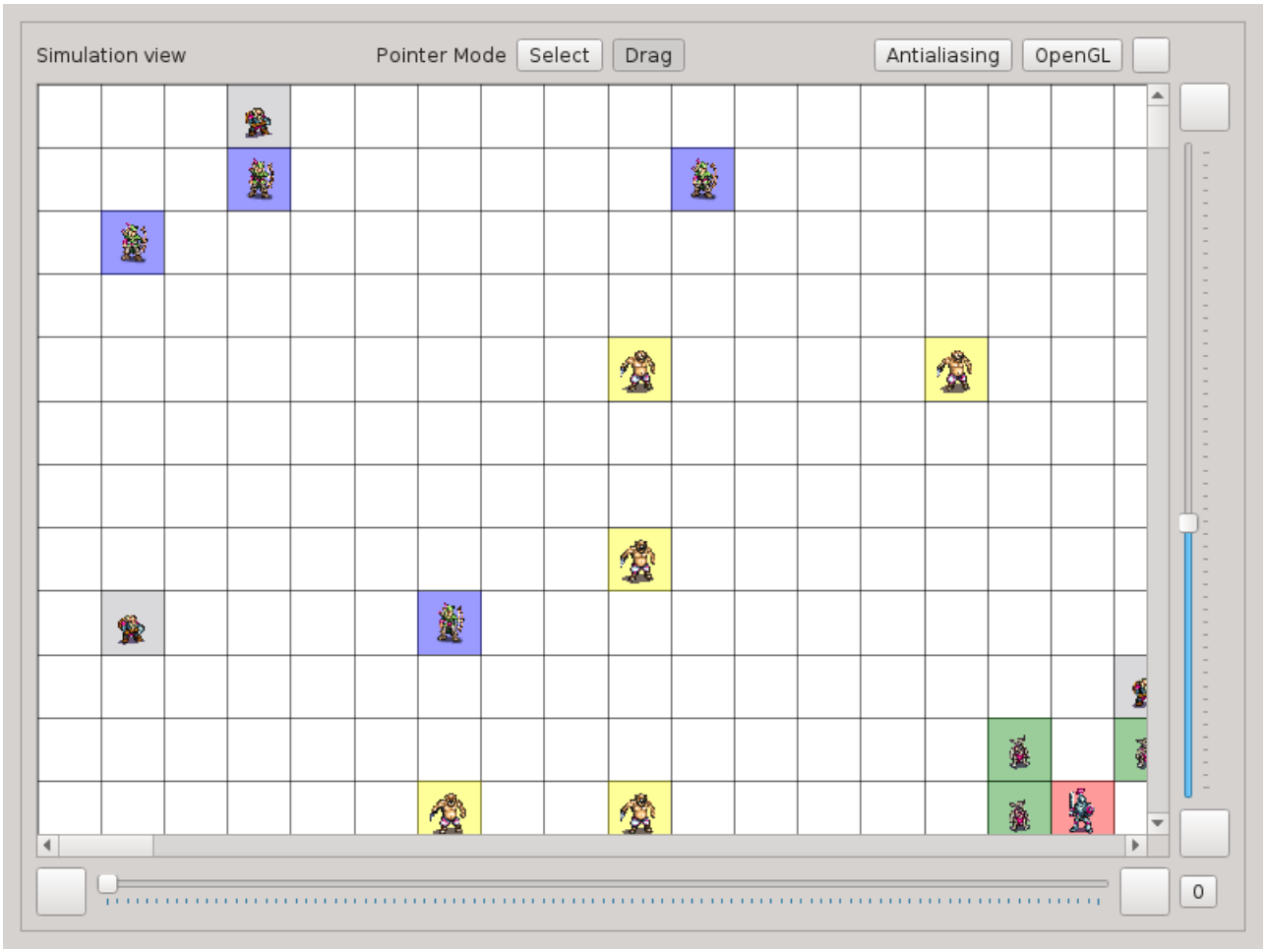

Abbildung 1: Visualisierung

4 Parallelisierungsschema

Die Welt wird zeilenweise aufgeteilt. D.h. eine Welt der Größe 100x100 wird auf 5 Prozesse so aufgeteilt, dass jeder Prozess einen Bereich von 20x100 Felder zugewiesen bekommt. Die Prozesse kommunizieren nach links und rechts mit ihren direkten Nachbarn.

Zusätzlich zum eigenen Bereich bekommt jeder Prozess noch eine Extraspalte des jeweils direkten Nachbarn links und rechts, um bereichsübergreifende Kollisionsabfragen durchführen zu können. Kreaturen, die aus dem eigenen Bereich herauslaufen, werden an den angrenzenden Nachbar gesendet und aus dem eigenen Speicher gelöscht.

5 Laufzeitmessungen

Mit folgenden Parametern wurden die Messungen durchgeführt:

```
./paraquest_simulation -r 10 -s 10000 -t 1000 -c 1000 -o 1000
```

- 1 Prozess: 285,8s
- 2 Prozesse: 74,4s
- 4 Prozesse: 23,34s
- 8 Prozesse: 11,9s

6 Leistungsanalyse

Bei der Leistungsanalyse ist besonders aufgefallen, dass am meisten Zeit in der Kollisionsabfrage benötigt wird. Dies hängt damit zusammen, dass die Kollisionsabfrage on $O(n^2)$ erfolgt, während die Kampfschleife $O(n)$ Zeit benötigt.

Das Senden und Empfangen verwendet dank der dünnen Matrix einen sehr geringen Teil der Laufzeit.

7 Skalierbarkeit

Bei der Skalierbarkeit ist uns aufgefallen, dass das Programm scheinbar superlinear skaliert. Hierführ haben wir keine Erklärung. Bei mehr Prozessen verhält sich das Programm annähernd linear.