Simulation eines Sonnensystems
Praktikumsbericht

Oliver Heidmann, Tronje Krabbe

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik, DKRZ
Praktikum Paralleles Programmieren 2015

Zusammenfassung.
Aufgabenstellung:
Programmierung einer parallelisierten Applikation mittels MPI.

Idee:
Simulation von Partikeln in einem Sonnensystem.

Inhaltsverzeichnis

1 Idee

Die Idee, die unserem Projekt zugrunde liegt, ist relativ simpel zu formulieren: Wir wollen ein Sonnensystem simulieren. In diesem System sollten die enthaltenen Objekte realistisch miteinander interagieren. Alle Objekte beinflussen sich durch die Gravitation und durch physischen Kontakt, also Kollisionen.

2 Modellierung

In unserem System wird die Bewegung des ganzen Systems nicht berücktsichtigt und somit ist die Geschwindigkeit und die Position des sich in der Mitte befindlichen massereichesten Objektes fixiert. Kollisionen sind nur teilweise realistisch umgesetzt. Bei einer Kollision werden zwei Objekte miteinder verschmolzen und es werden keine weiteren kleineren Objekte erzeugt. Alle einheiten sind in SI-Einheiten.

2.1 Die Partikel

Partikel haben folgende Eigenschaften:

- Geschwindigkeitsvektor
- Position
- Masse
- Radius

Wir gehen von sphärischen Objekten aus. Spezifische Details wie z.B. Material, Temperatur oder genauere Form werden nicht berücksichtigt. Partikel werden immer in einer zur Sonne orthogonalen Laufbahn generiert um die Anzahl von Objekten, die eine stabile Umlaufbahn erreichen können, zu steigern. Die Dichte von Objekten wird nicht als Datensatz gespeichert, sondern nur bei Bedarf errechnet, um die Datenmenge zu reduzieren.

2.2 Das Zentrum

Im Zentrum unseres Systems befindet sich ein Partikel mit einer besonderen Eigenschaft. Es ist um ein Vielfaches schwerer als die zufällig generierten Objekte. Die Existenz dieses Zentralpartikels bewirkt einerseits eine übersichtliche Simulation, da sich die meisten Partikel irgendwann auf einem Orbit um die 'Sonne' finden, und bringt die Simulation näher an die tatsächlichen Zustände in unserem Universum.

2.3 Ablauf

Der Ablauf der Simulation ansich ist relativ simpel; die Partikel werden sortiert, auf alle die Gravitation appliziert; es wird auf Kollisionen geprüft und dann werden die Objekte bewegt. Dieser loop wird für die spezifizierte Anzahl Iterationen

fortgeführt.
Die parallelisierte Version hat eine andere Reinfolge. Hier haben wir die Kollisionsabfrage vor den Bewegungsbefehl und die Errechnung der neuen Geschwindigkeiten verlegt. Dies geschah um den Datentransfer zwischen den einzelnen Prozessen zu veringern.

2.4 Visualisierung

Zum Zweck der Visualizierung werden für jede Iteration und für jedes Partikel seine Eigenschaften gespeichert. Diese Daten können dann von einem separaten Programm, unserem Visualizer, eingelesen und die Simulation damit dargestellt werden.

3 Implementation

3.1 Allgemein

Den Kern der Simulation bildet die Klasse 'Particle':

```
class Particle
{
    private:
        std::vector<Vec3<double> > m_velocity_vectors;
        std::vector<Vec3<double> > m_positions;
        std::vector<double> m_masses;
        std::vector<double> m_radiuses;
        std::vector<unsigned long> m_ids;
        unsigned long m_number_of_particles;

    /* [...] */
}
```

3.2 Parallelisierungsschema

In unserer Simulation übernimmt Prozess 0 immer die verwaltung der daten und somit das Verteilen und Sammeln. Hinzu kommen das Speichern der Simulationsdaten und die Ausgabe des Fortschritts. Alle anderen Prozesse empfangen die Daten, bearbeiten sie, und senden dann die durch die Kollisionsabfrage geänderte Länge des Datensatzes an Prozess 0.

Funktionsweise:
ALLE:

- Variableninitialisierung am Anfang
- Erhöhung des privaten Zählers am Ende

PRO 0:

- Prozess 0 errechnet die Anteile an den Datenvektoren des Systems
- sortiert die Vektoren nach der Entfernung zum Mittelpunkt
- broadcastet die Objekt-Anzahl
- broadcastet nacheinander die Datenvektoren
- erhält dann in richtiger Reihenfolge die neu berechneten Daten und fügt sie wieder zusammen
- überscheibt seine alten Daten mit den neuen
- speichert sie

PRO 1 - n

- empfängt die Broadcasts von 0
- errechnet seine Start- und End-Positionen in den Vektoren
- berechnet Kollisionen
- ändert die Länge der zu bearbeitenden Daten
- berechnet die neuen Geschwindigkeiten
- berechnet die neuen Positionen
- sendet neue Positionen
- sendet neue Geschwindigkeiten

3.3 Visualizer

Der Visualizer visualisiert eine fertige Simulation anhand der generierten Daten. Es handelt sich dabei um ein separates Programm, ebenfalls in C++ geschrieben, mit Hilfe von SDL2.

Integriert ist eine provisorische Konsole und mehrere Hotkeys für die einfache Benutzung.

4 Performance

Die Simulation läuft akzeptabel schnell und ein Speedup bei paralleler Ausführung ist erkennbar.

5 Probleme

Ein Problem bei der Parallelisierung war die Datenaufteilung. Da Gravitation einen effektiv unendlichen Einflussradius hat, muss jeder Prozess die gesamten, aktuellen Daten aller Partikel kennen.

Besonders die Kollisionsabfrage benötigt alle Daten, da bei einer Kollision der komplette Datensatz für eines der kollidierenden Objekte gelöscht werden muss.

6 Fazit und Ausblick

6.1 Fazit

Das Projekt war um einiges komplexer und umfangreicher, als wir am Anfang vermutet hatten; die sequentielle Implementation alleine hat uns äußerst viel Zeit gekostet. Dadurch sind Optimierung und Parallelisierung etwas zu kurz gekommen, was nicht hätte sein sollen. Dennoch haben wir wichtige Erkentnisse gewonnen und ein präsentables Programm entworfen und implementiert.

6.2 Ausblick

Für die Zukunft des Projektes sind folgende Features geplant:
Simulation:

- Redukion der zu verchickenden Daten durch Refactoring der Kollisionsabfrage, z.B nur die IDs der Kollisionspartner speichern und später Prozess 0 das löschen der Daten überlassen
- verbesserte Performance
- neu strukturiertes Senden und Empfangen
- Erweiterung des Sendens für Datensätze die die maximale Bufferlänge von MPI überschreiten
- performantere Kollisionsabfrage
- physikalisch korrekte Errechnung von Kollisionen, Richtungsvektoren, den daraus resultierenden neuen Objekte,i die errechnung der anzahl der generierten Objekte
- zusätzliche Daten z.B. Durchschnittstemperatur
- genauere Angaben, welche Daten gespeichert werden und welche nicht
- Errechnung und Speicherung für Kollsionsdaten z.B. Impuls, kinetische Energie, Geschwindigkeit bei Einschlag etc.
- Berechnung der Roche-Grenze und die Zerstörung von Objekten durch diese Grenze.
- Parallelisierte Erstellung der Objekte
- Parallelisierter Sort-Algorithmus
- Verbesserung des Sort-Algorithmuses
- Ausschreiben von Kollisonsdaten

Visualizer:

- Laufbahnen als lienen darstellen
- Lienen der Laufbahnen auf Wunsch der Geschwindikeit entsprechend
- einfärben
- Temperaturberechnung
- Objekte bei einem gewissen Zoomfaktor als Kreise Darstellen
- Visualizer in 3D, oder wenigstens rotierbar in 2 Achsen
- UI-Verbesserungen
- Konsolenfenster

- Objektdaten in Repositionierbaren Fenstern
- Repositionierbare fuer die Simulationsdaten wie die Anzahl der Objekte
- Klassen der Objekte anzeigen und errechnen z.B. M klasse Planet, Asterioid etc.
- Performance Verbesserungen. Inbesondere in Bezug auf die Flugbahn erstellung
- Markierung von Kollisionspunkten

7 Zahlen

- ca. 2700 Zeilen Code
- 163 Commits bis jetzt
- >20 Gb generierte Simulations-Daten