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 beruecktsichtig 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 Material, Temperatur oder genaure Form werden nicht berücksichtigt. Partikel werden immer in einer zur sonne ortogonalen laufbahn generiert um die Anzahl von Objekten die eine stabile Umlaufbahn erreichen koennen zu steigern. Die Dichte von Objekten wird nicht als Datensatz geschpeichert 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 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 datentransfaer 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 uebernimmt Prozess 0 immer die verwaltung der daten und somit das Verteilen und Sammeln. Hinzu kommt das Speichern der Simulationsdaten und die ausgabe des Fortschritts. Alle anderen Prozesse empfangen die Daten bearbeiten sie und senden dann die durch die Kollisionsabfrage geaenderte laenge des Datensatzes an Prozess 0 und daruf hin die geaenderten Daten.

Funktions weise: ALLE: - variablen initialisierung am Anfang - erhoen des privaten zaehlers 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 - erhaelt dann in richtiger Reinfolge die neu berechneten Daten und fuegt sie wieder zusammen - ueberscheibt seine alten Daten mit den neuen

- speichert sie PRO 1 - n - empfaengt die Broadcasts von 0 - erechnet seine start und end positionen in den vektoren - berechnet collisionen - aendert die laenge der zu bearbeitenden Daten - berechnet die neunen geschwindigkeiten - berechnet die neuen positionen - sendet neue positionen - sendet neue geschwindigkeiten

```
#include <Heidmann.h>
std::cout << "lol" << std::endl;
```

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 fuer die einfache Benutzung

Kommandos:

long name shot_name argument_type arguments

draw_ids dis none draw_it_number din none show_particle_number spn none show_simulated_time sst none show_delta_time sdt none display_data dd (unsigned long) [onject_id] show grav_range sgr (unsigned long) (double) [object_id][min_force] focus_on_object fon (unsigned long) [object_id] clear_displayed_data cdd none jump_to_time jmp (unsigned long) [interation] show_trajectory st (unsigned long) (unsigned long) [object_id] [iteration] clear_trajectory st (unsigned long) [object_id] clear_all_trajectories cat none

What they do:

draw_ids: draws object ids of all objects clears object ids

draw_it_number: draws the current interation number in the top right corner clears iteration number

draw_particle_number: draws the current number of particles clears particle number

show_simulated time: draws the elapsed time in y* dd hh clears simulated time

show_delta_time: shows how much time is calculated in one iteration (in seconds) clears delta time

display_data: shows the velo, pos, radius, distance to sun, mass of given object clear displayed date for given object

show_grav_range: draws a rircle around given object in which radius the gravitational force is larger than min_force clear the circle for given object

focus_on_object: sets the camera to given obeject

clear_displayed_data: clears all object data from screen (counter to display data)

jump_to_time: jumps to given iteration

show_trajectory: draws the trajectory of given object. the second arguments is for limiting the length of the line if there is no second argument the complete trajectory is drawn which can cause lag if sufficient iterations are loaded

clear_all_trajectories: clears all trajectories
Hotkeys:
p: pause/unpause s: step forward b: step backward r: reset to iteration 0 x/y: change perspective enter: start input for commands/ confirm esc: quit up: move camera up down: move camera down left: move camera left right: move camera right mousewheel: zoom

4 Performance

ayy lmao

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.

6 Fazit und Ausblick

6.1 Fazit

Das haben wir gelernt

6.2 Ausblick

Fuer die Zukunft des Projektes sind folgende Features geplant: Simulation: - verbesserte Kollision die mehrere neue Objekte generiert - verbesserte Performance - neu strukturiertes senden und empfangen - erweiterung des sendes fuer datensaetze die die maximale buffer laenge von mpi ueberschreiten - performantere Kollisionsabfrage - physikalisch korrekte errechnung von Kollisionen und den Richtungsvektoren der daraus resultierenden neuen Objekte - zusaetzliche daten zb durchschnitts Temperatur - genauere angeaben welche daten gespeichert werden und welche nicht - Errechnung und Speicherung fuer Kollsionsdaten z.B. impuls, Ekin, geschwindigkeit bei einschlag etc. - Berechnung der Roche-Grenze und die zerstoerung von Objekten durch diese Grenze. Visualizer: - laufbahnen als lienen darstellen - lienen der Laufbahnen auf wunsch der Geschwindikeit entsprechend einfaerben - temperatur Berechnung - Objekte bei einem gewissen Zoomfaktor als kreise Darstellen - Visualizer in 3D oder wenigstens rotierbar in 2 achsen

7 Zahlen

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