N-Body-Simulation

Semesterprojekt

im Rahmen der Veranstaltung

Parallel Systems

an der

Hochschule für Technik und Wirtschaft Berlin
Studiengang Angewandte Informatik (Master)

Jonathan Nowca

30. Juli 2018

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Semesterprojekt 1
1.2 Aufgabe 1

2 Definition **2**
2.1 N-Body-Problem 2
2.2 Problemkategorie 2
2.3 Parallelisierung 2

3 Grundlagen **3**
3.1 Weg, Geschwindigkeit, Beschleunigung 3
3.2 Newtons Graviationsgesetz 4
3.3 N-Body-Formulierung 5

4 Implementierung **6**
4.1 Erläuterung 6
4.2 Entwicklungsumgebung 6
4.3 Softwarearchitektur 6
4.4 Algorithmus 7
4.5 Parallelisierung 7
4.6 Prototyp 8

5 Tests **9**
5.1 Parallelisierung 9
5.2 Netzwerk 10

6 Fazit **10**

1 Einleitung

1.1 Semesterprojekt

Das Semesterprojekt soll als Belegaufgabe im Rahmen der Veranstaltung *Parallel Systems* im Sommersemester 2017 dazu dienen, praxisbezogene Inhalte als Softwareprojekt umzusetzen. Das Ziel ist die Entwicklung einer plattform-unabhängigen Anwendung zur Lösung eines Problems durch parallele Berechnung mittels eines Algorithmus.

1.2 Aufgabe

Mit der gewählten Aufgabe ist ein Programm zur *N-Body-Simulation* mit einem parallelisierten Simulationsalgorithmus zu entwickeln. Neben der schriftlichen Dokumentation der Aufgabe wird der Lösungsumfang durch folgende Kriterien festgelegt:

- Implementierung von mindestens einer, besser zwei parallelen Schnittstellen
- fehlerfreier Programmablauf
- Initialisierung über Kommmandozeileneingabe oder Konfigurationsdatei
- Visualisierung der Berechnungen
- Auswertung der Ergebnisse in Abhängigkeit steigender Problemgröße
- Dokumentation des Quelltextes
- Optional: Grafische Benutzerschnittstelle

2 Definition

2.1 N-Body-Problem

Bei dem N-Body-Problem (zu deutsch: N-Körper Problem) handelt es sich um ein primär aus der Astrophysik bedeutsames Problem zur Beschreibung des Bahnverlaufs von N Massekörpern (auch Teilchen) unter dem Einfluss der gegenseitigen gravitativen Wechselwirkung der Körper untereinander. Die Berechnung des Problems mit Hilfe von N-Body-Simulationen unterstützt das Verständnis und die Voraussage für die Entwicklung und den Verlauf von Planeten, Sternen und Galaxien und weiteren Objekten im Weltraum. Während das Problem für $N = 2$, das Zwei-Körper-Problem, bereits gelöst wurde, verbleibt es für $N > 2$ ungelöst und muss durch numerische Integration berechnet werden.

2.2 Problemkategorie

In der Informatik wird einem Algorithmus zur direkten Berechnung des N-Body-Problems mittels einfacher Integration, für alle Körper untereinander, der Laufzeitkomplexität $O(n^2)$ zugeordnet. Damit fällt das Problem in die Komplexitätsklasse P und ist auf deterministischen Turingmaschinen in polynomialer Zeit berechenbar. Da für jeden $Nten$ Körper $O(n)$ Operationen auf alle weiteren Körper ausgeführt werden, steigt die Laufzeit zur Berechnung mit zunehmender Anzahl der beteiligten Körper quadratisch an. Es gibt unterdessen verschiedene Verfahren und vereinfachte Algorithmen, die den Rechenaufwand auf $O(n\ log\ n)$ oder sogar $O(n)$ reduzieren. Jedoch unterscheiden sich die verschiedenen Verfahren stark untereinander und liefern teils unterschiedliche und auch weniger-genauere Ergebnisse.

2.3 Parallelisierung

Aufgrund der Beschaffenheit des Problems, kann die numerische Berechnung effizient parallelisiert werden. In jedem Zeitschritt können mehrere Kalkulationen gleichzeitig durchgeführt werden, da die gegenseitige gravitative Wirkung der Körper untereinander separat berechnet werden kann. Aus diesem Grund eignet sich das N-Body-Problem sehr stark zur Verwendung in Performance Benchmarks für parallel-arbeitende Prozessoren, vor allem grafische Prozessoren (GPUs), aufgrund ihrer hohen Anzahl von Kernen.

3 Grundlagen

3.1 Weg, Geschwindigkeit, Beschleunigung

Die Position eines Massenpunktes durch seine Bewegung (auch Kinematik) kann über die zeitliche Integration in Zusammenhang des Weges mit der Geschwindigkeit und der Beschleunigung errechnet werden. Zur Erläuterung gelten folgende Definitionen:

- s ist der Weg (oder auch die Strecke) in Meter $[m]$
- v ist die Geschwindigkeit in Meter pro Sekunde $[m/s]$
- a ist die Beschleunigung in Meter pro Sekunde-Quadrat $[m/s^2]$
- t ist die Zeit in Sekunden $[s]$

Wenn der Weg s eine Funktion der Zeit ist $s(t)$, dann ist die Geschwindigkeit deren Ableitung nach der Zeit und die Beschleunigung die Ableitung der Geschwindigkeit nach der Zeit. Folglich ist die Beschleunigung die zweite Ableitung des Weges nach der Zeit:

$$a(t) = \dot{v}(t) = \ddot{s}(t) = \frac{dv}{dt} = \frac{d^2s}{dt^2}$$

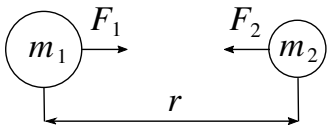
Folglich kann der Weg durch Integration der Geschwindigkeit nach der Zeit und die Geschwindigkeit durch Integration der Beschleunigung nach der Zeit ermittelt werden:

$$s(t) = \int v(t)\ dt = v \cdot t \qquad\qquad v(t) = \int a(t)\ dt = a \cdot t$$

3.2 Newtons Graviationsgesetz

Die Voraussage der Bewegungen von N Objekten kann durch die Verwendung von Newtons Gravitationsgesetz berechnet werden. Um die gemeinsame (äquivalente) Anziehungskraft zweier Körper (F_1 und F_2) zu berechnen, wird sowohl die Gravitationskonstante (G), die Massen der Objekte (m_1 und m_2), wie auch die Distanz der Objekte (r) verwendet:

$$F_1 = F_2 = G\frac{m_1\ m_2}{r^2}$$



Zum Verständnis nun einige wichtige Definitionen zu den einzelnen Bestandteilen von Newtons Gravitationsgesetz:

- F ist die Wirkende Kraft in Newton $[N = kg \cdot m/s^2]$
- G ist die Gravitationskonstante als $G = (6,674\ 08\ \pm\ 0,000\ 31)\ \cdot\ 10^{-11}\frac{m^3}{kg \cdot s^2}$
- m_1 und m_2 sind die Massen der einwirkenden Körper in Kilogramm $[kg]$
- r ist die Distanz $(r_2 - r_1)$ der interagierenden Körper in Meter $[m]$

In vektorieller Form samt Richtung ändert sich die mathematische Formulierung wie folgt:

$$\vec{F_1} = \vec{F_2} = G\frac{m_1\ m_2}{|\vec{R}_{12}|^3}\vec{R}_{12}$$

Die Distanz im Divisor ändert sich hier in einen vektoriellen Betrag in euklidischer Form:

$$|R_{12}| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Die resultierende Gravitationsbeschleunigung auf ein Objekt lässt sich wie folgt berechnen:

$$\vec{a}_1 = G\frac{m_2}{|\vec{R}_{12}|^3}\vec{R}_{12}$$

3.3 N-Body-Formulierung

Um die im vorangegangenen Kapitel erläuterte Formulierung zur Errechnung der gravitativen Beschleunigung eines Objekts für eine N-Body Simulation nutzbar zu machen, muss nun die (2-Body) Formel so geändert werden, dass nun mehrere (also N) Körper auf diesen einwirken (N-Body):

$$\vec{a}_i = \sum_{j \neq i}^{N} G \frac{m_j}{|\vec{R}_{ij}|^3} \vec{R}_{ij}$$

Die N-Body Formel besagt, dass für N Körper (iteriert über i) die Massen und Distanzen aller anderen Körper (iteriert über j) einberechnet werden. Eine wichtige Restriktion ist der Ausschluss des eigenen Körpers innerhalb des Durchlaufs ($j \neq i$).

Aus der ermittelten Beschleunigung für den i-ten Körper, kann nun über zeitliche Integration die neue Position des aktuellen Körpers berechnet werden:

Änderung der Geschwindigkeit: $v_i(t + \Delta t) = v_1(i) + a_i \ \Delta t$

Änderung der Position: $r_i(t + \Delta t) = r_1(i) + v_i \ \Delta t$

4 Implementierung

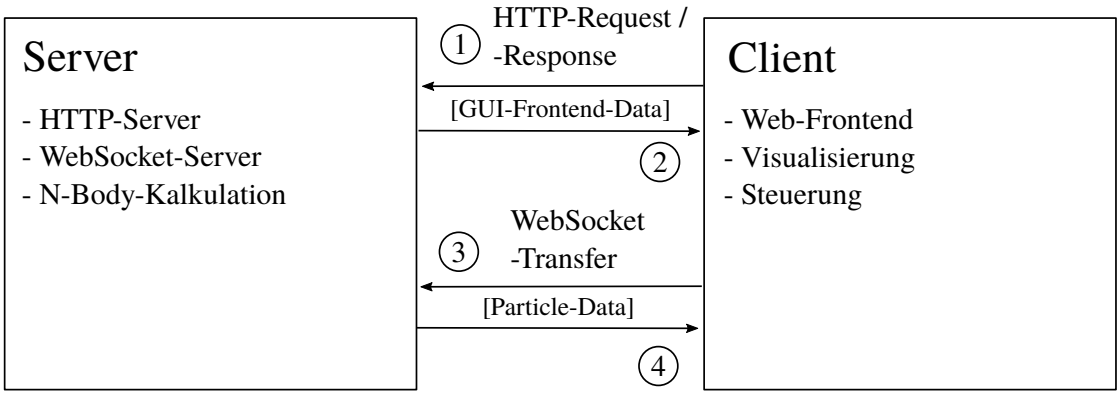
4.1 Erläuterung

Die Implementierung des Projekts wird gemäß der gestellten Aufgabe (siehe 1.2) als Client-Server Architektur umgesetzt. Zum Datentransfer und zur Steuerung soll das WebSocket-Protokoll dienen. Zur Realisierung der Server-Anwendung dient die systemnahe Programmiersprache C. Das clientseitige Frontend wird in HTML/JavaScript umgesetzt.

4.2 Entwicklungsumgebung

Software	**Einsatzzweck**
Ubuntu 16.04.10 LTS	Betriebssystem zum Programmablauf und Tests
Sublime Text 3125	Texteditor als Programmierwerkzeug
GCC 5.4.0	Softwarepaket zum Kompilieren
OpenMP 5	Bibliothek zur Parallelisierung
Libwebsockets 2.1.0	Bibliothek zum Einsatz des WebSocket Protokolls
Firefox Quantum 61.01	Webbrowser für das Frontend

4.3 Softwarearchitektur



Der mit GCC kompilierte Server läuft mit Hilfe der Libwebsockets-Bibliothek und öffnet einen HTTP-Dienst zum Aufrufen des Web-Frontends (z.B. über http://127.0.0.1:8080). Nach Aufruf im Webbrowser verbindet sich die Client-Webanwendung über den selben Port auf den Server und tauscht die berechneten Daten aus.

4.4 Algorithmus

Der Algorithmus zur Berechnung durchläuft iterativ für jeden Körper p_i aus P alle anderen Körper p_j, unter Ausschluss des selben Körpers und Berechnet die aus der N-Body-Formulierung (siehe 3.3) notwendigen Berechnungen für die Beschleunigungs- $\vec{a}_i$, Geschwindigkeits- $\vec{v}_i$ und Positionsvektoren $\vec{r}_i$

1: **procedure** NBodyBerechnen(dt) ▷ Zeitschritt dt
2: **for** $p_i < n$ **do** ▷ Körper p_i, Anzahl der Körper n
3: $\vec{a}_i = 0.0$ ▷ Beschleunigungswerte auf Null setzen
4: **for** $p_j < n$ **do** ▷ Partikel P_j, Anzahl der Körper n
5: **if** $p_i \neq p_j$ **then** ▷ für alle anderen Körper
6: $\vec{d} = \vec{r}_j - \vec{r}_i$ ▷ Distanzen d aus Differenzen der Positionen $\vec{r}_i$ und $\vec{r}_j$
7: $|R_{ij}| = \sqrt{(d_x)^2 + (d_y)^2 + (d_z)^2}$ ▷ Euklidische Distanzberechnung
8: $|R_{ij3}| = |R_{ij}|^3$ ▷ Potenzieren auf 3D
9: $f = \frac{G * m_j}{|R_{ij3}|}$ ▷ Kraft f aus Gravitation G, Masse m_j und Distanz $|R_{ij3}|$
10: $\vec{a}_i = \vec{a}_i + f * \vec{d}$ ▷ Beschleunigung $\vec{a}_i$ für Partikel p_i
11: **end if**
12: **end for**
13: $\vec{v}_i = \vec{a}_i * dt$ ▷ Geschwindigkeit $\vec{v}_i$ aus Beschleunigung $\vec{a}_i$ und Zeitschritt dt
14: $\vec{r}_i = \vec{v}_i * dt$ ▷ Position $\vec{r}_i$ aus Geschwindigkeit $\vec{v}_i$ und Zeitschritt dt
15: **end for**
16: **end procedure**

4.5 Parallelisierung

Die parallelisierte Berechnung wird mit der OpenMP Bibliothek implementiert. Hierfür erhält der Funktionsrumpf noch einen Parameter für die maximale Anzahl von Threads. Die innere FOR-Schleife für die Iteration durch alle anderen Körper p_j wird hierfür mit eingeteilten Start- und Endwerten für jeden Thread versehen. Diese werden berechnet aus Steps per Thread $spt = \frac{n}{t_n}$ mit der Anzahl der Körper n, Anzahl der Threads t_n, dem Startwert für p_j mit Thread-Nummer t_i aus $j_{start} = spt * t_i$ und $j_{end} = j_{start} + spt$.

4.6 Prototyp

Der beiliegende Prototyp kann mit dem Kommando `make build` kompiliert werden oder mit `make` kompiliert und testweise mit Parametern ausgeführt werden. Die Parameter zum Ausführen der Software setzen sich zusammen aus:

```
nbody_server [Anzahl der Körper] [Anzahl der Threads] [Port]
```

Der Prototyp ist simpel umgesetzt und soll zur Überprüfung der Funktionsfähigkeit des Konzepts dienen. Die Daten werden über das WebSocket Protokoll mittels JSON-Format übermittelt und clientseitig über die JavaScript-API eingelesen. Nach dem Starten der Server-Anwendung kann im Web-Browser die Visualisierung als WebGL-Webanwendung aufgerufen werden. Über die grafische Benutzeroberfläche lässt sich der Server steuern (Pausieren, Geschwindigkeit, Anzahl der Threads).

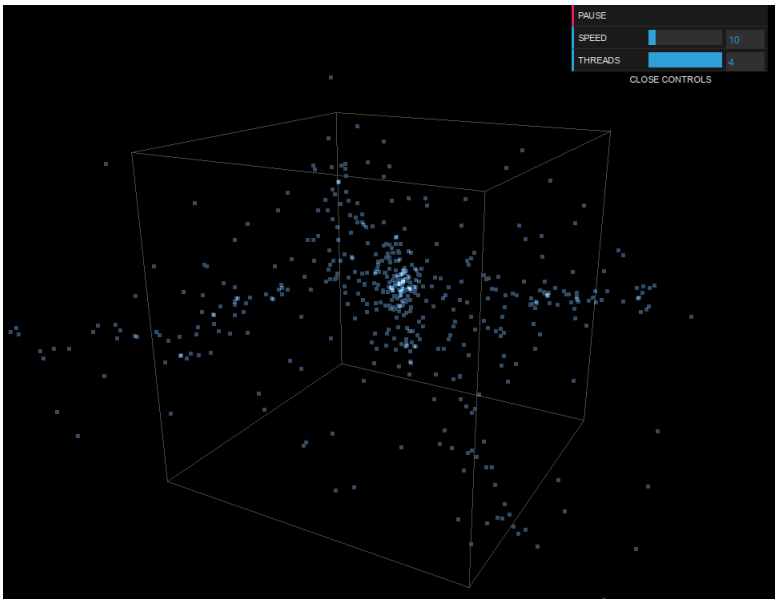


Abbildung: Screenshot der Webanwendung

5 Tests

5.1 Parallelisierung

Die Anwendung wird mit `timeout -s 2 30 ./nbody_server {n_p} {n_t} 8080` getestet, wobei mit n_p die Anzahl der Körper und mit n_t die Anzahl der Threads gesteuert wird. Es wird geprüft, wie viele Zeitschritte nach einem Durchlauf von 30 Sekunden erreicht werden. Die Testdurchläufe wurden auf einer Maschine mit Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz unter Ubuntu 16.04.10 LTS durchgeführt.

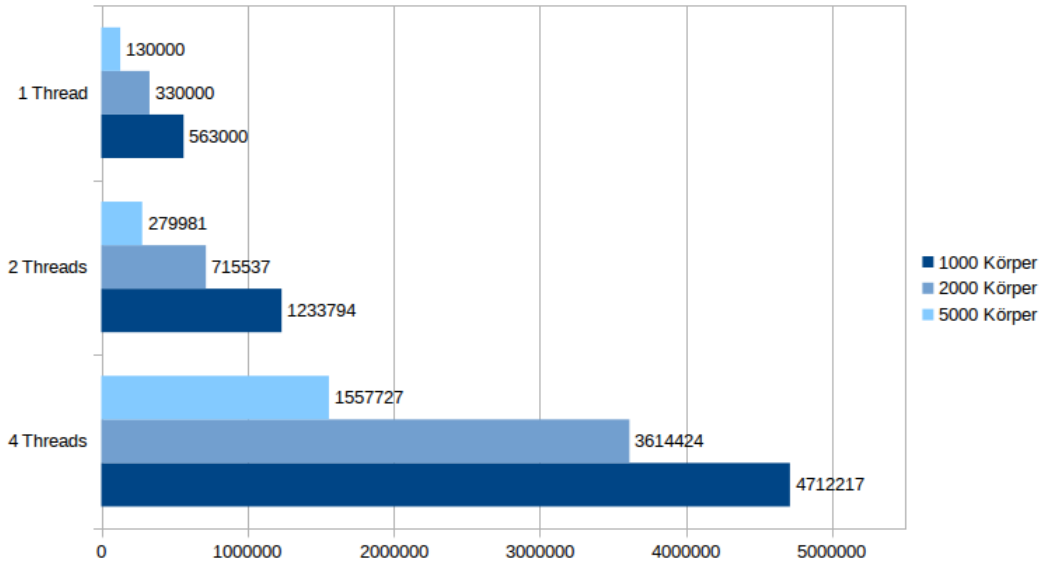


Abbildung: Testergebnisse der Zeitschritte aller Durchläufe zu je 30 Sekunden

Die Ergebnisse zeigen erheblich höhere Werte durch den Einsatz mehrerer Threads. Es hat sich jedoch auch gezeigt, dass die unterschiedliche Einstellung der Threads in noch höhere Werte ungewöhnlich hohe Unterschiede ausweist. Leider konnte dies im Rahmen der Umsetzung der Arbeit nicht genauer überprüft werden.

5.2 Netzwerkübertragung

Der Übertragung mit dem WebSocket-Protokoll über das JSON-Format erzeugt sehr viele Daten und hat zum Software Absturz geführt. Aus diesem Grund werden die Daten für die Übertragung in Segmente eingeteilt, die beim Empfang wieder zusammengesetzt werden. Dies führt leider zu einem erheblichen Performance-Verlust.

Die Analyse der Loopback-Schnittstelle über Wireshark mit 1000 Körpern über 30 Sekunden ergibt einen Netzwerklast von *2293 kilobytes* und benötigt damit eine hohe Bandbreite zu einer einigermaßen flüssigen Übertragung. Zur Optimierung sollten die Daten komprimiert übertragen werden. Im Rahmen des Belegs wird dies jedoch nicht umgesetzt und getestet, es sollte jedoch im Falle eines Einsatzes implementiert werden.

6 Fazit

Die Belegarbeit hat mir persönlich einen Einblick in physikalische Grundlagen, die Arbeit mit zugehörigen mathematischen Formeln und Algorithmen, wie auch das systemnahe Implementieren in C mit geeigneten Bibliotheken vermittelt. Zusätzlich konnte durch das Umsetzen des Frontends in WebGL die Grundkenntnisse in OpenGL ES gestärkt werden.

Die Idee mit der Übertragung über WebSockets hat sich als relativ geeignet, jedoch stark optimierungs-bedürftig erwiesen. Im Web-Bereich bietet sich das bidirektionale Protokoll als oft implementierter und verbindungs-orientierter Standard an. Die Entwicklung von parallelisierten Programmen mit koordinierter Netzwerkübertragung ist ein sehr interessantes Projekt. Es bieten sich für diese Art von Anwendungen viele Einsatzmöglichkeiten auf vielen Plattformen und auch unterschiedlichen Übertragungsmethoden und Netzwerktechnologien. Wenn das Projekt auch sehr aufwendig war, freue ich mich sehr über den Abschluss und das durchaus positive Ergebnis.