Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bericht im Zuge des
Praktikum "Parallele Programmierung"

Direct Gravitational N-body Simulation

vorgelegt von

Nicholas Hickson-Brown, Michael Eidus

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
Arbeitsbereich Wissenschaftliches Rechnen

Abstract

Contents

1. Einführung

Unter dem Begriff N-body Simulation versteht man die Lösung numerischer Gleichungen, welche die Bewegung von N-Partikeln unter Einfluss der Gravitation oder anderer physischer Kräfte beschreiben. Zweck einer solchen Simulation ist es, die Interaktion der Partikel untereinander auf eine Weise zu beobachten, die in der Realität nicht möglich ist. N-body Simulationen können unser Verständnis des Universums erweitern, indem sie uns die Möglichkeit bieten, komplexe aber wohldefinierte Vorgänge mit Hilfe von Software darzustellen und zu analysieren, oftmals in einer vielfach höheren Geschwindigkeit als diese Vorgänge in der Wirklichkeit stattfinden.

N-body Simulationen finden Anwendungen in verschiedenen Bereichen der Astronomie und Astrophysik. Mit der Hilfe von solchen Computersimulationen lassen sich, relativ gesehen, einfache Vorgänge wie die Umlaufbahn unsere Mondes um die Erde simulieren, aber auch deutliche komplexere Abläufe, wie die Entstehung von Sternhaufen oder der Einfluss dunkler Materie. Des Weiteren finden sie nicht nur bei der Simulation großer Himmelskörper wie Planeten und Sternen Anwendung, sondern beispielsweise auch bei der Simulation einzelner Atome innerhalb einer Gaswolke.

Verschiedene Methoden existieren, um solche Vorgänge zu simulieren, die allerdings alle eine Sache gemeinsam haben: Sie sind enorm rechenintensiv. Einige Methoden versuchen die Komplexität solcher Simulationen zu verringern, indem sie lediglich Schätzungen in bestimmten Fällen vornehmen und so weniger Ressourcen in Anspruch nehmen. Beispiel für solche Methoden sind beispielsweise der Barnes-Hut-Algorithmus, bei dem ein Sternhaufen in Quadranten unterteilt wird und diese Quadranten sich untereinander nur als Summer ihrer Masse betrachten, oder Particle-Mesh-Methoden.

Der Nachteil von Methoden, die Annahmen oder Annäherungen treffen, um Rechenzeit einzusparen, ist, dass sie zwar an Effizienz gewinnen aber an Genauigkeit verlieren. Dem Gegenüber stehen sogenannte "Direct Methods", welche die numerischen Newtonschen Bewegungsgleichungen direkt implementieren und damit zur Berechnung der Interaktionen innerhalb eines Systems jeden Partikel mit jedem anderen Partikel vergleichen müssen. Infolgedessen sind diese Methoden zwar sehr genau, besitzen aber eine wesentlich höhere Komplexität und erfordern demnach auch mehr Rechenzeit. Nennenswerte Vertreter der direkten Methoden sind beispielsweise die Runge-Kutta-Methode oder das Hermite-Schema.

2. Problemstellung

In dieser Arbeit wollen wir ergründen, ob es durch ausreichende Parallelisierung und Optimierung möglich ist, die Laufzeit von direkten Methoden dementsprechend zu verringern, dass diese eine effiziente und realistische Option gegenüber Tree-Codes (Barnes-Hut-Algorithmus) und Particle-Mesh-Methoden bieten. Dabei ist insbesondere der Faktor der Genauigkeit von direkten Methoden eine treibende Kraft hinter unserer Fragestellung. Sollte es möglich sein, mit entsprechender Parallelisierung einer sequentiellen N-body Simulation deren Laufzeit zu verringern und Effizienz zu steigern, so wären diese gegenüber anderen Methoden zu präferieren.

Zur Beantwortung dieser Frage konzentrieren wir uns auf sogenannte "Direct Gravitational N-body Simulations". Diese Art von N-body Simulationen verwendet direkte Methoden zur Lösung der numerischen Bewegungsgleichungen und beachtet dabei nur den Einfluss der Gravitation um die Interaktionen und Bewegungen der Partikel zu berechnen. Infolgedessen betrachten wir hier keine äußeren Einflüsse, wie etwa dunkle Materie oder schwarze Löcher, und auch keine Kollisionen, entweder der Partikel unter einander oder mit eventuellen Komponenten des Sternhaufens, die keine Sterne sind. Diese Arten von Simulationen sind unserer Meinung nach den Direct Gravitational N-body Simulations untergeordnet und wir wollen versuchen eine Aussage für den allgemeinen Fall treffen, nicht den speziellen.

Zudem betrachten wir in unserer Arbeit einen Kugelsternhaufen von variabler Anzahl an Sternen, sprich Partikeln, ab einem Zeitpunkt t_0, an dem die Sterne des Systems bereits geboren sind und sich im Zentrum der Masse des Sternhaufens konzentrieren. Ab diesem Zeitpunkt starten wir die Simulation und betrachten die Evolution des Sternhaufens. Der Endzeitpunkt ist dabei undefiniert, beziehungsweise die Partikel besitzen keine biologische Uhr und sterben demnach nicht, sondern die Simulation endet bei Erreichen eines Zeitpunkts t_x, der vor Beginn des Ablaufs vom Benutzer definiert wird.

Außerdem wollen wir den Ablauf der Simulation visualisieren. Die Visualisierung fließt weder in die Betrachtung der Laufzeiten, noch in die Parallelisierung unseres Programms mit ein. Sie soll lediglich als Veranschaulichung der Ergebnisse dienen und die Funktionalität unserer Implementierung für den Verwender belegen.

Nach erfolgreicher Implementation einer sequentiellen N-body Simulation wollen wir eine effiziente Methode zur Parallelisierung des Programms finden und diese umsetzen. Die parallele Umsetzung soll dabei Antwort auf unsere eingehend genannte Frage geben.

3. Design

Im Folgenden wollen wir den Aufbau und die theoretischen Grundlagen unseres sequentiellen Programms darlegen. Dabei unterscheiden wir zwischen der Simulation an sich, sprich dem sequentiellen Programm, welches die numerischen Gleichungen löst, und der Visualisierung, welche die entstehenden Datensätze ansprechend darstellt und interaktiv gestaltet. Wir wollen mit diesem Kapitel eine Übersicht über die der Implementation zugrundeliegende Theorie und unsere Überlegungen geben.

3.1. Simulation

Bei der Implementation der Simulation hatten wir drei grundlegende Entscheidungen zu treffen: Die Methode der Generierung der initialen Konditionen unseres Sternhaufens, die Art der direkten Berechnung der Bewegungen der Partikel, sowie die Form der Datenausgabe. Im folgenden Abschnitt wollen wir die Theorie hinter unseren Entscheidungen näher betrachten und damit die Grundlage für das Verständnis der Implementation schaffen.

Um unsere Simulation zu starten benötigen wir initiale Koordinaten, Geschwindigkeiten und Massen für unsere Partikel, die zudem auch noch relativ realistisch sein sollten. Außerdem war uns klar, dass wir einen Kugelsternhaufen simulieren wollten, da in einem solchen durch die Dichte an Sternen und den zentralen Punkt der Masse eine höhere Anzahl an Interaktionen von Partikeln wahrscheinlich ist und diese mehr Rechenaufwand bedeutet, den es durch Parallelisierung zu verringern gilt. Die Prämisse unserer Entscheidung für ein bestimmtes Modell zur Generierung der initialen Konditionen war demnach vor allem der zu erwartende Rechenaufwand.

Unser erster Ansatz war ein einfacher Linear-Congruential-Generator (LCG), da dieser eine relativ gute statistische Verteilung besitzt und Aufwandsarm zu implementieren ist. Nach einigen ersten Tests fiel uns allerdings auf, dass der LCG zwar die erwartete Verteilung besaß, dieser aber weder Geschwindigkeiten noch Massen für unsere Sterne generieren konnte, da das Intervall des LCG durch die Größe des Datentyps *long* begrenzt ist und somit eine hohe Anzahl an wiederkehrenden Werten zu erwarten war. Außerdem wurde die Verteilung der Partikel keinen astrophysischen Kennzahlen gerecht. Nach einer eingehenden Suche einigten wir uns dann auf das sogenannte Plummer Model.

3.1.1. Plummer Model

Das Plummer Model oder "Plummer density profile" besitzt einen hohen Bekanntheitsgrad, da es einerseits relativ bekannt ist und anderseits mit relativer Leichtigkeit zu implementieren ist. Es wird häufig als sogenanntes "Toy Model" zur Generierung initialer Konditionen in N-body Simulationen eingesetzt. Das Plummer Model beschreibt die observierte Dichte von Sternenhaufen besser als andere Modelle seiner Art, hat aber auch einige Schwachstellen.

So ist bei großen Radei die Dichte nicht mehr so akkurat, wie in der Realität beobachtet wurde und andererseits ist es keine gute Beschreibung von elliptischen Galaxien. Diese Schwachstellen stellen aber kein Hindernis für uns da, da wir weder elliptische Galaxien noch große Radei betrachten. Außerdem hat bereits Sverre Aarseth et. al. in "A comparison of numerical methods for the study of star cluster dynamics" [AHW74](Appendix) das Plummer Model als Grundlage für ähnliche Betrachtungen vorgeschlagen.

Das dreidimensionale Dichte-Profil von Plummer lässt sich durch folgende Formel beschreiben:

$$\rho P(r) = (\frac{3M}{4\pi a^3}) \cdot (1 + \frac{r^2}{a^2})^{-\frac{5}{2}}$$

wobei M die totale Masse des Sternhaufens darstellt, und a den Radius.

Das Plummer Model soll uns drei 7 Werte für jeden Partikel erzeugen: Die Koordinaten (x-, y- und z-Achse), die Geschwindigkeiten (x-, y- und z-Achse) und die jeweilige Masse. Die Erzeugung der Masse ist simpel, da in unserem System alle Sterne die selbe Masse besitzen sollen (sog. "mass equilibrium").

Dies ist natürlich nicht sonderlich realistisch, aber zum Vergleich von Laufzeiten und Implementation fundamental, da es keine Varianz in den Ergebnissen und Energiewerten erzeugt. Mit einem Gleichgewicht der Masse können wir sicher sein, dass zwei unterschiedliche Systeme mit den selben Rahmenbedingungen, sprich Anzahl Partikel, Zeitschritt und Endzeit, zumindest sehr ähnliche Laufzeiten erzeugen, womit ein Vergleich von sequentieller und paralleler Implementierung erst möglich wird. Somit erzeugen wir die Masse der einzelnen Partikel, indem wir lediglich die totale Masse des Sternhaufens durch die Anzahl von Partikeln teilen.

Zusätzlich benötigen wir natürlich auch noch die Koordinaten und Geschwindigkeiten für unsere Partikel, die durch das Plummer Modell gegeben sind. Dazu greifen wir auf das eingehend genannte Paper von Aarseth et. al. [AHW74] zurück, in dessen Appendix eine ausführliche Anleitung zur Bestimmung initialer Konditionen mit dem Plummer Modell steht. Die dort genannten Formeln und Techniken werden wir als Grundlage für die Implementation eines Generators für initiale Konditionen eines Kugelsternhaufens verwenden.

3.1.2. Mersenne Twister

Für die Berechnung der initialen Konditionen benötigen wir zufällig gewählte Werte innerhalb eines bestimmter Intervalle, die zudem gute statistische Eigenschaften besitzen müssen. Aus diesem Grund haben wir uns gegen die Implementation eines eigenen Random Number Generators (RNG) entschieden und stattdessen für einen bereits implementierten. Der Mersenne Twister bot sich dafür an, da dieser alle von uns gewünschten Eigenschaften besitzt und dessen Entwickler Makoto Matsumoto und Takuji Nishimura bieten auf ihrer Website eine Implementationen des Mersenne Twister in verschiedenen Programmiersprachen mit freigiebiger Lizenz an [MN07].

Außerdem sind wir nicht der Ansicht, dass die Verwendung des Mersenne Twister erheblichen Overhead produzieren wird, da wir in nur an einigen wenigen Stellen benötigen werden. Deshalb sehen wir auch nicht die Notwendigkeit zur Parallelisierung unseres RNG. Des Weiteren ist es zur Beantwortung unserer Leitfrage nicht relevant, ob wir einen RNG verwenden oder nicht. Da wir aber die Erstellung initialer Konditionen in unsere Laufzeitberechnungen einbeziehen wollen, ist die Verwendung des Mersenne Twister zwar eine Notwendigkeit, aber keinesfalls relevant zur Fragestellung oder N-body Simulationen an sich. Aus diesen Gründen haben wir uns für die Verwendung einer bereits vorhandenen Implementation an dieser Stelle entschieden.

3.1.3. Hermite Integrator

Abschließend benötigen wir noch eine Methode, um die Newtonschen Bewegungsgesetze umzusetzen und damit die Interaktion und Bewegung der durch das Plummer Model erstellten Partikel unseres Kugelsternhaufens zu simulieren. Zur Auswahl stehen viele Varianten, da es sich hierbei um die Lösung von Integralen handelt.

Begonnen haben wir damit, eine sehr simple Variante der direkten Methoden zu implementieren, indem wir die Newtonschen Bewegungsgleichungen direkt in stark vereinfachter Form implementiert haben. Diese Implementation war zwar einfach zu verstehen, jedoch nicht sonderlich elegant und die erzeugten Ergebnisse waren außerdem nicht sehr akkurat. Wir wollten eine Methode implementieren, die auch in der anderen N-body Simulationen eingesetzt wird, um so nah wie möglich an den echten Use-Case der N-body Simulationen zu gelangen.

Über den Forward-Euler-Algorithmus, den Leapfrog-Algorithmus und die Runge-Kutta-Methode sind wir dann auf den Hermite Integrator oder das Hermite Schema gestoßen. Das Hermite Schema wird häufig in N-body Simulationen die direkte Methoden verwenden eingesetzt und war damit der perfekte Kandidat für unsere Leitfrage. Zusätzlich ist der Hermite Algorithmus der am einfachsten zu implementierende Algorithmus vierter Ordnung.

Der Hermite Algorithmus leitet die Newtonschen Gesetze der Bewegung nur einmal

ab und gelangt so direkt an die entsprechende Beschleunigung und den sogenannten "Jerk", eine Art ruckartiger Bewegung, für jeden einzelnen Partikel. Die resultierenden Gleichungen können dann durch eine Taylor Reihe erweitert werden. Der Ablauf ist dann wie folgt darstellbar:

- Zuerst berechnen wir eine Vorhersage für die nächsten Positionen und Geschwindigkeiten unser Partikel:

$$r_{p,i+1} = r_i + v_i\Delta t + \frac{1}{2}a_i(\Delta t)^2 + \frac{1}{6}j_i(\Delta t)^3$$

$$v_{p,i+1} = v_i + a_i\Delta t + \frac{1}{2}j_i(\Delta t)^2$$

- Danach berechnen wir eine Vorhersage für die jeweiligen Beschleunigungen und Jerks.

- Anschließend verwenden wir die berechneten Beschleunigungen und Jerks um die Vorhersagen aus dem ersten Schritt zu korrigieren:

$$v_{c,i+1} = v_i + \frac{1}{2}(a_i + a_{p,i+1})\Delta t + \frac{1}{12}(j_i - j_{p,i+1})(\Delta)^2$$

$$r_{c,i+1} = r_i + \frac{1}{2}(v_i + v_{c,i+1})\Delta t + \frac{1}{12}(a_i - a_{p,i+1})(\Delta)^2$$

Die entsprechenden Gleichungen und Beweise für diese haben wir dem Buch "The Art of Computational Science: The Kali Code" [HM07a](Kapitel 11) entnommen, welche als Grundlage für unsere Implementation des Hermite Algorithmus dienen sollen.

3.2. Visualisierung

4. Implementation

4.1. Simulation

4.2. Visualisierung

5. Parallelisierungsschema

6. Laufzeitmessungen

6.1. Sequentielle Laufzeitmessungen

6.2. Parallele Laufzeitmessungen

7. Leistungsanalyse

8. Skalierbarkeit

9. Ausblick: Verbesserungen

[AHW74] [HM07b] [HT08]

Bibliography

[AHW74] S. J. Aarseth, M. Henon, and R. Wielen. A comparison of numerical methods for the study of star cluster dynamics. *Astronomy and Astrophysics*, 37(12):183–187, Dezember 1974.

[HM07a] Piet Hut and Jun Makino. *The Art of Computational Science: The Kali Code*, volume 2 of *The Maya Open Lab for Dense Stellar Systems*. Hut, Piet and Makino, Jun, September 2007.

[HM07b] Piet Hut and Jun Makino. *The Art of Computational Science: The Kali Code*, volume 11 of *The Maya Open Lab for Dense Stellar Systems*. Hut, Piet and Makino, Jun, September 2007.

[HT08] Piet Hut and Michele Trenti. N-body simulations (gravitational). `http://www.scholarpedia.org/article/N-body_simulations_(gravitational)`, 2008. [Online, abgerufen 08.10.2017].

[MN07] Makoto Matsumoto and Takuji Nishimura. Mersenne twister in c, c++, c#. `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/C-LANG/c-lang.html`, 2007. [Online, abgerufen 09.10.2017].

Appendices

A. Verwendete Hard- und Software

Hardware

- Desktop PC (Windows 10)
  - AMD FX-8320 Octa-Core CPU (OC @ 4.0GHz)
  - 32GB DDR3-1600 RAM (Quadchannel)
  - 2 x Asus Strix - Radeon R9 380 4GB GPUs (2-Way Crossfire; OC)
- MacBook Pro (15-inch, Mid 2012)
  - Intel Core i7 Quad-Core CPU (Stock @ 2,3 GHz)
  - 8GB DDR3-1600 RAM (Dualchannel)
  - Intel HD Graphics 4000

B. Grafiken der Programmausgabe