Wissenschaftliches Rechnen – Paralleles Höchstleistungsrechnen

Christian Engwer
http://wwwmath.uni-muenster.de/num/Vorlesungen/WissenschaftlichesRechnen_SS11/

Abgabe 20. 4. 2011. Abgabe der Programmieraufgaben bis per Email an christian.engwer@uni-muenster.de, schriftliche Abgabe Donnerstags in der Vorlesung.

- Alle Programmierübungen müssen per Email und in ausgedruckter Form abgegeben werden.
- Achten sie darauf, ihr Programm ordentlich zu formatieren und gut zu kommentieren.

In der Vorlesung wurde das N-Körper-Problem eingeführt, dieses führt auf eine System gewöhnlicher Differentialgleichungen.

$$\frac{dr_i(t)}{dt} = v_i(t)$$
$$\frac{dv_i(t)}{dt} = a_i(t) = \sum_{\substack{j=0 \\ j\neq i}}^{N-1} G\frac{m_j}{(\|r_j - r_i\|^2 + \epsilon^2)^{3/2}}(r_j - r_i)$$

Weitere Details zu diesem Thema können sie auf der Webseite "The Art of Computational Science – How to build a computational lab" (www.artcompsci.org) finden.

Der Kugelsternhaufen M80 im Sternbild Skorpion – ca. hunderttausende Sterne.

Laden sie sich von der Vorlesungshomepage die Header nbgenerator.hh nbio.hh nbtypes.hh und das Programmskelett nbody_skel.cc herunter. Wir hatten uns bereits in der Vorlesung mit dem Hauptprogramm etwas vertraut gemacht.

Übung 1 *N*-Körper Problem – Erste Schritte

Im Programmskelett nbody_skel.cc sind zwei Funktionen noch nicht implementiert.

Kopiere die Programmskelettdatei und speichere sie unter einem neuen Namen

- acceleration: hier soll für jeden Körper i die Beschleunigung a_i^k zum aktuellen Zeitschritt k berechnet werden.
$$a_i^k = \sum_{\substack{j=0 \\ j\neq i}}^{N-1} G\frac{m_j}{(\|r_j^k - r_i^k\|^2 + \epsilon^2)^{3/2}}(r_j^k - r_i^k) \ .$$
- euler: mit Hilfe des expliziten euler Verfahrens sollen aus der aus der aktuellen Position r_i^k, Geschwindigkeit v_i^k und Beschleunigung Beschleunigung a_i^k die neue Position und Geschwindigkeit berechnet werden.
$$r_i^{k+1} = r_i^k + v_i^k \cdot \Delta t \ , \qquad v_i^{k+1} = v_i^k + a_i^k \cdot \Delta t$$

Ergänzen sie das Programm um die fehlenden Funktionen acceleration und euler.

Wenn sie das Programm aufrufen, werden 100 VTK Dateien geschrieben, die die Position der Körper zu verschiedenen Zeitpunkten beschreiben. Sie können diese mit dem Programm paraview visualisieren. Hierzu starten sie Paraview und öffnen die ganze Sequenz über File → Open. Die Sequenz wird im Dateidialog als nbody_..vtk angezeigt. Mit dem Abspielbutton können sie nun die Bewegung der Körper sehen.

Übung 2 N-Körper Problem – Fliesskomma Operationen

1. Überlegen und begründen sie wie viele Fließkomma Berechnungen jeweils pro Aufruf in `acceleration` und `euler` durchgeführt werden.
2. Laden sie von der Homepage zusätzlich die Datei `npstopwatch.hh` herunter und ergänzen sie das Hauptprogramm (`main`) um Aufrufe zur Zeitmessung. Hieraus können sie die Fließkommaoperationen pro Sekunde (FLOPS) berechnen.
   Hinweise:
   - Binden sie den Header mit `#include "npstopwatch.hh"` ein.
   - Mit dem Kommando `getTime()` können sie sich die aktuelle Zeit in Einheiten Sekunden von Sekunden geben lassen. Wenn sie die Zeit zu zwei Zeitpunkte subtrahieren erhalten sie die Dauer des Intervalls.
   - Bestimmen sie die Zeit die für `mod` Zeitschritte benötigt wird und berechnen sie hieraus die FLOPS. Geben sie das Ergebnis aus.
   - Achten sie darauf, dass in der Zeitmessung die Zeit für das Schreiben der VTK Dateien nicht nicht berücksichtigt wird!
3. Informieren sie sich im Internet, mit welchem Prozessor sie ihre Berechnungen druchgeführt haben und wie viele Fließkommaoperationen pro Sekunde dieser Prozessor nach Spezifikation hat. Vergleichen sie die gemessenen FLOPS mit den spezifizierten FLOPS.

Übung 3 N-Körper Problem – Zeitschrittverfahren

Es gibt unterschiedliche Zeitintegrationsverfahren. Ein verbessertes Verfahren, gegenüber dem Expliziten Euler Verfahren ist das Leapfrog Verfahren. Hierbei werden die neuen Zeitschritte durchfolgende Formel berechnet:

$$r_i^{k+1} = r_i^k + v^k \cdot \Delta t + a^k \frac{(\Delta t)^2}{2} \tag{1}$$

$$v_i^{k+1} = v^k + (a^k + a^{k+1}) \cdot \frac{\Delta t}{2} \tag{2}$$

Zur Berechnung der neuen Position r_i^{k+1} wird die aktuelle Geschwindigkeit, sowie eine Extrapolation mittels der Beschleunigung verwendet.

Es fällt auf, dass zur Berechnung der neuen Geschwindigkeit v_i^{k+1} bereits die Beschleunigung zum neuen Zeitschritt $k+1$ benötigt wird. Das heisst insbesondere, dass nach der Berechnung der neuen Positionen bereits die neuen Beschleunigungen berechnet werden müssen.

1. Implementieren sie eine Funktion `leapfrog` mit dem gleichen Interface, wie die Funktion `euler`:
   `void leapfrog (const int N, double dt, vector3 r[], vector3 v[], double m[], vector3 a[])`
   Hinweis: Denken sie daran, dass zwischen der Berechnung der Positionen und der Berechnung der Geschwindigkeiten die neuen Beschleunigungen berechnet werden müssen. Hierzu benötigen sie einen weiteren Vektor, den sie wie im Hauptprogramm mit `vector2 anew[N];` *anlegen können.*
2. Passen sie das Hauptprogramm so an, dass der Leapfrog anstelle des expliziten Eulers verwendet wird.
3. Vergleichen sie die Ergebnisse der beiden Zeitschrittverfahren. Was passiert, wenn das Δt variieren?
   Hinweis: Spielen sie ruhig mal mit sehr verschiedenen Werten für Δt.

Anhang C++

Einige `C++` Befehle, die für sie hilfreich sein könnten:

- `std::abs(x)` berechnet den Betrag $|x|$
- `std::sqrt(x)` berechnet die Wurzel $\sqrt{x}$
- `std::pow(x,y)` berechnet die Potenz x^y. *Hinweis:* `std::pow(x,2)` ist *deutlich* langsamer als `x*x`.
- Zur Textausgabe können sie den Header `iostream` verwenden (`#include <iostream>`) und dann mit dem bekannten Ausgabeoperator `<<` nach `std::cout` schreiben. Bsp:
  `std::cout << "Ergebnis:  " << 27 << std::endl;`