Übungen zur Vorlesung

**Paralleles Höchstleistungsrechnen**

Dr. S. Lang, D. Popović

Abgabe: 18. Januar 2012 in der Übung

---

Beginnend mit dieser Übung untersuchen wir eine Variante des Gravitations-$N$-Körper-Problems (engl. Gravitational-$N$-Body-Problem, *GNBP*), das im Detail noch in der Vorlesung besprochen wird. Wir werden das Problem mit den bisher erlernten Techniken Tiling, *OpenMP*, *PThreads*, MPI und *CUDA* parallelisieren und die jeweils erreichte MFLOP-Rate messen.

Das Problem erscheint zunächst sehr physikalisch, jedoch reichen Schul-Physik-Kenntnisse aus. Lesen Sie zuallererst den Abschnitt am Ende des Übungsblattes, in dem die von uns verwendete Variante des N-Körper-Problems und der auf der Vorlesungsseite bereitgestellte Code erklärt werden. Auf der Homepage finden Sie zusätzliche Hinweise zu den Dateien. Machen Sie sich mit dem Code vertraut, der die Bewegung mehrerer Körper im leeren Raum, die sich gegenseitig durch Gravitation anziehen, berechnet.

Für jede Variante der Parallelisierung existiert ein eigener Kernel. Diese Kernel können mit dem vorhandenen Makefile durch `make` kompiliert werden. Ein Kernel besteht im Wesentlichen aus den Funktionen `accelerate()`, `leapfrog()` und `main()`. Die Funktion `leapfrog()` ist fast immer gleich und implementiert das Zeitschritt-Verfahren. Die wichtigste Funktion ist `accelarate()`, da sie die Hauptarbeit verrichtet. Es handelt sich bei der Funktion um einen Algorithmus der Komplexität $N^2$, dies ist die Funktion, die wir in den verschiedenen parallelen Varianten implementieren wollen. Manche Varianten sind schon ausimplementiert (Sequentiell, Tiling, *OpenMP*), für andere muss diese Funktion im Kernel ausgearbeitet werden. Es gibt auch die Möglichkeit, die Komplexität der Funktion auf $O(N \log N)$ oder gar $O(N)$ zu reduzieren. Dies würde hier aber zu weit führen und wird in der Vorlesung besprochen. Beachten Sie bitte auf jeden Fall die Hinweise am Ende des Übungsblattes und auf der Homepage!

### Übung 23 N-Körper-Problem: Sequentiell, Tiling, *OpenMP* (5 Punkte)

In der seriellen Variante des N-Körper-Problems berechnet die Funktion `acceleration()` die Beschleunigung aller Körper. Für einen Körper $i$ muss sie über alle anderen Körper $j = i+1, \ldots, N-1$ iterieren und ihre Positionen `x[j]` und Massen `m[j]` laden, die Beschleunigungen $a_{ij} = -a_{ji}$ berechnen und in `a[i]` und `a[j]` akkumulieren.

Sie finden im Archiv die sequentielle, Tiling- und *OpenMP*-Variante ausimplementierte. Der jeweilige Kernel erhält als Parameter $N$, die Anzahl der auszuführenden Zeitschritte sowie die Wahl der Ausgabe-Schritte. Der Zeitschritt $dt$ ist gesetzt auf $dt = 0.001s$, was für unsere Fälle gut reichen sollte. Voreingestellt ist die Gleichverteilungs-Anfangsbedingunen. Lassen Sie hier einmal folgende Simulation laufen: `./nbody_vanilla 40 100000 100`, und betrachten Sie das Ergebnis mit Paraview!

Wir betrachten zunächst die Kachelung. In der folgenden Abbildung wird die serielle Durchlaufreihenfolge zur Berechnung der Beschleunigungen links gezeigt, rechts ist die optimierte Reihenfolge für das Kacheln zu sehen, mit einer Blockgröße von $B = 2$.

Sie finden diese Variante im Kernel `nbody_tiled.c`. Beachten Sie, wie die ursprüngliche Loop über `i` und `j` in Loops über die Blöcke und innere Loops aufgeteilt wurde:

```
for(I = 0; I < N; I += B)
  for(J = I; J < N; J += B)
    for(i = I; i < MIN(N, I+B); ++i)
      for(j = MAX(i+1,J); j < MIN(N, J+B); ++j)
      {
        /* code goes here */
      }
```

Die *OpenMP*-Parallelisierung baut auf der gekachelten Variante auf und befindet sich im Kernel `nbody_openmp.c`.

## Aufgabe

Entscheiden Sie sich für eine der beiden Anfangsbedingungen `plummer` (voreingestellt) oder `cube` und verwenden Sie diese für alle folgenden Übungen. Führen Sie nun eine Perfomance-Analyse der sequentiellen Variante und mit Kachelung und *OpenMP* durch, in dem Sie die *MFLOP*-Rate messen. Verwenden Sie dazu Werte von $N >= 100$ und verschiedene Kachelgrößen $B$. Passen Sie die Anzahl der Zeitschritte so an, daß einige Zeitschritte ausgeführt werden und mitteln Sie die *MFLOP*-Rate über die Zeitschritte. Bei den Testmessungen ergab sich im Pool mit Kachelung keine Verbesserung der *MFLOP*-Rate, wenn es Ihnen also auch nicht gelingt, wundern Sie sich nicht. Auf anderen Rechnern konnte mit Kachelung hingegen ein Performance-Gewinn wie erwartet festgestellt werden. Nehmen Sie die gemessenen Raten über der Problemgröße in einer Tabelle auf. Diese Tabelle werden wir noch mit den anderen Parallelisierungs-Arten auffüllen und dann vergleichen.

## Freiwillige Zusatz-Aufgabe

Um das Problem des nicht vorhandenen Perfomance-Gewinns genauer zu untersuchen, wollen wir das Datenlayout ändern, um die Daten in anderer Reihenfolge in den Cache zu laden. Definieren Sie dazu eine Datenstruktur `Body`, die `m, x, v` und `a` für jeden Körper speichert, und legen Sie dann ein Feld `Body[n]` für die `n` Körper an. Die temporären Beschleunigungen können Sie in einem Feld wie gehabt lassen. Ändern Sie nun die Funktionen `accelerate` und `leapfrog` so, dass sie jeweils volle Körper laden. Im Gegensatz zur Speicherung einzelner Felder für die Komponenten liegen die Körper nun konsekutiv im Speicher. Ist eine Verbesserung der durchgeführten Messungen erkennbar? Alternativ dürfen Sie gerne eigene Ideen zur Ausnutzung des Caches einbauen.

## Hinweis

Um parallele Varianten auf Richtigkeit zu prüfen, können Sie die generierten *VTK*-Dateien vergleichen. Die Simulations-Ergebnisse können Sich dabei aber in einigen Nachkommastellen unterscheiden. Das bereitgestellte Skript `fuzzy_diff` kann zwei *VTK*-Dateien vergleichen bezüglich einer vorgegebenen Toleranz, z.B.:

```
./fuzzy_diff sequential.vtk parallel.vtk 1e-14
```

Verwenden Sie das Skript, um Ihre parallelen Codes zu validieren.

### Übung 24 GNBP with MPI (10 Punkte)

In dieser Übung parallelisieren wir das N-Körper-Problem mit *MPI*. In der MPI-Version des Codes hält jeder Prozess nur einen Teil der Koordinaten und Massen. Der Einfachheit halber können Sie für diese Übung $N \, mod \, p = 0$ annehmen.

## Aufgabe

Implementieren Sie ein *MPI*-paralleles Lösungsschema für das $N$-Körper-Problem. Eine Idee wäre, die beteiligten Prozesse im Ring blockierend kommunizieren zu lassen unter Verwendung von Färben der Kanten. Es steht Ihnen das Gerüst `nbody_mpi.c` zur Verfügung, in dem Hilfsroutinen wie das Generieren der Anfangsbedingungen und das Schreiben oder Lesen der `VTK`-Dateien schon parallelisiert sind. Der bereitgestellte Code funktioniert für einen einzelnen Prozess.

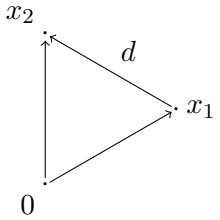
Parallelisieren Sie nun die Routine `accelerate_mpi`, so daß sie das MPI-Kommunikationsschema implementiert. Führen Sie dann Rechnungen mit denselben Parametern und Anfangsbedingungen wie in der vorigen Aufgabe durch. Messen Sie die Rechenzeiten und berechnen Sie den Speed-Up des parallelen Programms. Überprüfen Sie auch die Rechenergebnisse, in dem Sie die Resultate der parallelen Rechnung mit den Ergebnissen der sequentiellen Variante mit dem `fuzzy_diff`-Skript vergleichen. Tragen Sie die gemessenen *MFLOP*-Raten in die Tabelle ein und erstellen Sie nun einen Plot *MFLOP*-Rate über $N$.

# Das Gravitations-$N$-Körper-Problem (GNBP)

Beim $N$-Body-Problem untersucht man, wie sich $N$ Körper in dem durch Ihre Massen hervorgerufenen Gravitations-Kraft-Feld bewegen. Dieser Abschnitt erklärt in Kürze die verwendete Modellierung und Numerik des N-Körper-Problems sowie die Implementierung und verwendeten Tools.

## Die Bewegungsgleichungen

Mehrere sich bewegende Körper ziehen sich durch die Gravitationskraft gegenseitig an und beeinflussen so ihre gegenseitige Bewegung. Wir stellen ein System gewöhnlicher Differentialgleichungen auf, die den Ort und die Geschwindigkeit aller $N$ Körper beschreiben. Wir beginnen dazu mit zwei Körpern im leeren Raum an den Orten $x_1$ uns $x_2$:




$$x_1 + d = x_2$$
$$\Rightarrow d = x_2 - x_1$$

Der Ort $x_i$ eines Körpers hängt, ebenso wie seine Geschwindigkeit $v_i$ und seine Beschleunigung $a_i$, von der Zeit ab, wir notieren dies durch $x_i(t)$. Nach dem Gesetz von Newton lautet die Kraft, die Körper 1 mit Masse $m_1$ auf den Körper 2 mit der Masse $m_2$ ausübt ($x_i$ hängt von $t$ ab!):

$$F_{12}(t) = \gamma \frac{m_1 m_2}{\|x_2 - x_1\|^3}(x_2 - x_1),$$

wobei $\gamma$ die Gravitationskonstante ist. Für den Fall mit $N$ Körpern ist die Gesamt-Kraft auf Körper $i$ durch die Superposition der Teilkräfte, die von den anderen Körpern hervorgerufen werden, gegeben:

$$F_i(t) = \sum_{\substack{j=0 \\ j\neq i}}^{N-1} \gamma \frac{m_i m_j}{\|x_j - x_i\|^3}(x_j - x_i) \qquad \forall i = 0, \ldots, N-1.$$

Das zweite Newtonsche Gesetz verknüpft die Beschleunigung eines Körpers mit der Kraft, die auf ihn ausgeübt wird: $F_i(t) = m_i a_i(t) \quad \forall i = 0, \ldots, N-1$. Man kann man die Geschwindigkeit eines Körpers durch die Beschleunigung ausdrücken: $a_i(t) = dv_i(t)/dt$, und die Geschwindigkeit durch den Ort: $v_i(t) = dx_i(t)/dt$. Unbekannt sind hier also der Ort $x_i(t)$ sowie die Geschwindigkeit $v_i(t)$ aller Körper $i = 0, \ldots, N-1$. Diese Unbekannten können mit folgendem linearem, gekoppeltem System gewöhnlicher Differentialgleichungen, das numerisch zu Lösen ist, bestimmt werden:

$$\left.\begin{aligned} \frac{dx_i(t)}{dt} &= v_i(t) \\ \frac{dv_i(t)}{dt} &= \frac{1}{m_i} F_i(t) = \sum_{\substack{j=0 \\ j\neq i}}^{N-1} \gamma \frac{m_j}{\|x_j - x_i\|^3}(x_j - x_i) \end{aligned}\right\} \quad \forall i = 0, \ldots, N-1. \tag{0.1}$$

Für die Orte und Geschwindigkeiten der Massen müssen noch passende Anfangswerte vorgegeben werden.

Im Folgenden wird folgende Bezeichnung für die Unbekannten $x_i(t)$ verwendet:

$$\vec{x}(t) = \begin{pmatrix} \vec{x}_0(t) \\ \vdots \\ \vec{x}_{N-1}(t) \end{pmatrix} = \begin{pmatrix} (x_0(t))_0 \\ (x_0(t))_1 \\ (x_0(t))_2 \\ (x_1(t))_0 \\ \vdots \end{pmatrix}.$$

Im Vektor $\vec{x}(t)$ sind also die ersten drei Einträge die Koordinaten $(x_0(t))_0$, $(x_0(t))_1$ und $(x_0(t))_2$ des Körpers mit dem Index 0, in den folgenden drei Einträgen die des zweiten Körpers mit dem Index 1, usw., gespeichert. In der gleichen Weise werden $\vec{a}(t)$, $\vec{v}(t)$ and $\vec{m}$ verwendet.

### Massenschwerpunkts-Korrektur

Der Massenschwerpunkt und seine Geschwindigkeit sind

$$R(t) = \frac{1}{M}\sum_{i=0}^{N-1} m_i x_i \qquad V(t) = \frac{1}{M}\sum_{i=0}^{N-1} m_i v_i \qquad \text{mit} \quad M = \sum_{i=0}^{N-1} m_i.$$

Da keine externen Kräfte vorhanden sind, ist $V(t)$ konstant und der Schwerpunkt bewegt sich gleichförmig (geradlinig). Daher kann man Massenschwerpunkt in den Usrprung eines Koordinatensystems legen:

$$\left.\begin{aligned} \tilde{x}_i(0) &= x_i(0) - R(0) \\ \tilde{v}_i(0) &= v_i(0) - V(0) \end{aligned}\right\} \qquad \forall i = 0, \ldots, N-1.$$

Diese Korrektur wird einmal am Beginn vorgenommen. Die Tilde $\tilde{}$ wird von jetzt an nicht mehr mitgeschrieben und es wird angenommen, daß $R(t) = V(t) = 0$.

### Stabilisierung des Modells

Wenn zwei Körper kollidieren, $x_j \to x_i$, wird die Kraft, die ein Körper auf den Anderen ausübt, unendlich groß, $F_{ij} \to \infty$. Um diesen Fall numerisch handzuhaben, wird der Nenner in der Beschleunigung so geändert, daß er niemals 0 wird:

$$a_{ij} = \frac{\gamma m_i}{(\|x_j - x_i\|^2 + \epsilon^2)^{3/2}}(r_j - r_i)$$

Die korrespondierende Kraft kann vom sogenannten *Plummer*-Potential

$$\Phi(x_i, x_j, \epsilon) = -\frac{\gamma m_i m_j}{(\|x_j - x_i\|^2 + \epsilon^2)^{1/2}}$$

abgeleitet werden. Die physikalische Interpretation ist, daß die ursprünglich hergeleitete Kraft für Punktmassen gilt, während die korrigierte für Massen mit ausgedehnter, variabler Dichteverteilung

$$\rho(r) \propto \frac{1}{(r^2 + \epsilon^2)^{5/2}}$$

gilt. Das *Plummer*-Potential hat außerdem den Vorteil, an der Singularitätsstelle des zugehörigen Gravitationspotentials stetig und differenzierbar zu sein.

### Numerik

Um das zeitabhängige System (0.1) zu lösen, unterteilen wir das Simulationsintervall $(0, T]$ in $q$ gleich lange Intervalle der Länge $\Delta t$, und bezeichnen die Ränder dieser Zeitintervalle mit $t_k$, $k = 0, \ldots, q$. Angenommen, die Lösung $(\vec{x}^k, \vec{v}^k)$ zu einem Zeitpunkt $t_k = k \cdot \Delta t$ ist bekannt, so berechnen wir die unbekannte Lösung $(\vec{x}^{k+1}, \vec{v}^{k+1})$ zum nachfolgenden Zeitpunkt $t_{k+1} = t_k + \Delta t$ über folgendes hier *Leapfrog*-Verfahren genanntes Einschritt-Verfahren mit der Iterations-Vorschrift:

$$\begin{aligned} \vec{x}^{k+1} &= \vec{x}^k + \vec{v}^k \cdot \Delta t + \vec{a}^k \tfrac{1}{2}(\Delta t)^2, \\ \vec{v}^{k+1} &= \vec{v}^k + (\vec{a}^k + \vec{a}^{k+1}) \cdot \tfrac{1}{2}\Delta t. \end{aligned}$$

Dieses Verfahren

- berechnet aus gegebenen $(\vec{x}^k, \vec{v}^k)$ zunächst die neuen Beschleunigungen $a^{k+1}$, und daraus die neue Lösung $(\vec{x}^{k+1}, \vec{v}^{k+1})$,
- braucht nur eine Auswertung von $\vec{a}$ pro Schritt (falls $\vec{a}^k$ und $\vec{a}^{k+1}$ gespeichert werden),
- ist 2. Ordnung akkurat in der Zeitdiskretisierung.

### Für Interessierte: Herleitung des Zeitschritt-Verfahrens

Im Skript von Prof. Bastian, Kapitel 10, wird nur das explizite Euler-Verfahren zum Lösen von (0.1) gezeigt, welches ein Verfahren erster Ordnung ist. Die Herleitung unseres Verfahrens und die Konvergenz zweiter Ordnung sieht man durch Anwenden einer Taylor-Entwicklung für $\vec{x}(t)$ und $\vec{v}(t)$. Für $\vec{x}(t)$ lautet diese zum Beispiel:

$$\begin{aligned}\vec{x}(t+\Delta t) &= \vec{x}(t) + \Delta t\frac{d\vec{x}(t)}{dt} + \tfrac{1}{2}\Delta t^2\frac{d^2\vec{v}(t)}{dt^2} + O(\Delta t^3)\\ &= \vec{x}(t) + \Delta t\vec{v}(t) + \tfrac{1}{2}\Delta t^2\vec{a}(t) + O(\Delta t^3)\\ \rightsquigarrow \vec{x}^{k+1} &= \vec{x}^k + \vec{v}^k\cdot\Delta t + \vec{a}^k\tfrac{1}{2}(\Delta t)^2.\end{aligned}$$

In ähnlicher Weise gilt für $\vec{v}(t)$:

$$\begin{aligned}\vec{v}(t+\Delta t) &= \vec{v}(t) + \Delta t\frac{d\vec{v}(t)}{dt} + \tfrac{1}{2}\Delta t^2\frac{d^2\vec{v}(t)}{dt^2} + O(\Delta t^3)\\ &= \vec{v}(t) + \Delta t\vec{a}(t) + \tfrac{1}{2}\Delta t^2\frac{d\vec{a}(t)}{dt} + O(\Delta t^3).\end{aligned}$$

Die verbleibende Zeitableitung $d\vec{a}(t)/dt$ kann mit der Taylorentwicklung für $a(t)$ behandelt werden:

$$\frac{d\vec{a}(t)}{dt} = \frac{\vec{a}(t+\Delta t) - \vec{a}(t)}{\Delta t} + O(\Delta t),$$

was zu

$$\begin{aligned}\vec{v}(t+\Delta t) &= \vec{v}(t) + \tfrac{1}{2}\Delta t\vec{a}(t) + \tfrac{1}{2}\Delta t\vec{a}(t+\Delta t) + O(\Delta t^3)\\ \rightsquigarrow \vec{v}^{k+1} &= \vec{v}^k\tfrac{1}{2}\Delta t\vec{a}^k + \tfrac{1}{2}\Delta t\vec{a}^{k+1}\end{aligned}$$

führt. Daher hat das Verfahren die Ordnung 2.

## Implementierung

### Datenstrukturen

Die Daten $\vec{x}^k$, $\vec{v}^k$, $\vec{a}^k$, $\vec{a}^{k+1}$ und $m$ werden in der Form *Name – Körper-Id – Komponente* gespeichert, also `double x[N][3], v[N][3], a[N][3], anew[N][3], m[N]`.

### Zeitmessung

Es wird für alle Kernel die Wall-Time in $s$ gemessen über Differenzen der Rückgabewerte der Funktion `omp_get_wtime()`. Der bereitgestellte Code kann die gemessene *MFLOP*-Rate ausgeben.

### Anfangsbedingungen

Es gibt zwei Funktionen, um die Anfangswerte für die Massen und Geschwindigkeiten zu erzeugen:

`plummer()` Alle Körper haben die gleiche Masse $\frac{1}{N}$:

$$\Phi(r) = -\gamma M\frac{1}{(r^2+\vec{a}^2)^{1/2}}.$$

Das System ist im Gleichgewicht, und die Massenschwerpunkt-Korrektur wird angewendet.

`cube()` Alle Körper sind zufällig in einem Würfel verteilt. Die Körper haben keine Geschwindigkeit und ihre Masse ist zufällig in $m \pm \Delta m$ verteilt. Die Massenschwerpunkts-Korrektur wird verwendet.

Die Daten sind in Dateien im *VTK*-Format (Visualisation Toolkit, siehe den nächsten Abschnitt) gespeichert. Die Funktionen, um diese Dateien zu Lesen und zu Schreiben, heißen `read_vtk_file_double()` und `write_vtk_file_double()` und liegen in `io_vanilla.h` bzw. `io_mpi.hh`.

### Ein- und Ausgabe von Daten

Der Code erzeugt zum Visualisieren der Simulationsergebnisse *Paraview*-Dateien im *VTK*-Format (siehe dazu auch die Hinweise auf der Homepage). Diese Dateien haben je nach Art der dargestellten Daten verschiedene Endungen, polygonale Daten werden in `.vtp`, Daten auf unstrukturierten Gittern in `.vtu` Dateien gespeichert. Die Dateien können mit dem im Pool installierten Programm *Paraview* geöffnet und angesehen werden. Bei Zeitreihen werden mehrere Dateien (für welche Schritte, gibt die Variable `mod` an), herausgeschrieben. Öffnet man die erste Datei, lädt *Paraview* automatisch die konsekutiv folgenden und im Animations-Dialog kann man einen Film starten. Die Software startet im Pool mit `paraview`. Wenn Sie sich über `ssh` einloggen, vergessen Sie bitte nicht, die Grafikausgabe mit dem Flag `-X` auf Ihren lokalen Rechner umzuleiten: `ssh -X phlr025@pool.iwr.uni-heidelberg.de`.

### Das Haupt-Programm

`main()` Die `main()`-Funktion liest Programm-Parameter, allokiert Speicher für die Variablen und initialisiert sie. Ausserdem startet sie die Zeitschritt-Intregration durch den Aufruf von `leapfrog()` und schreibt Output in Dateien. Die Zeitschrittweite ist auf $dt = 0.001s$ gesetzt, womit das Verfahren für unsere Problemgrößen gut konvergieren sollte. Die Anzahl der Körper und der auszuführenden Schritte wird als Parameter übergeben, ebenso das im vorigen Abschnitt angesprochene `mod`.

`leapfrog()` Diese Funktion führt einen Zeitschritt aus.

**Input-Parameter:** Anzahl Körper $n$, Zeitschrittweite `dt`, Vektoren der alten Positionen `x` und alten Geschwindigkeiten `v`, der Massen `m`, und der alten Beschleunigungen `a`.

**Output-Parameter:** Vektoren der neuen Positionen `x`, neuen Geschwindigkeiten `v` und neuen Beschleunigungen `a`.

Der zusätzliche Parameter `aold` wird als temporärer Speicher für die alten Beschleunigungen verwendet. Seine Werte bei Input and Output sind aber nicht relevant.

`acceleration()` Diese Funktion wird von `main()` verwendet, um die initialen Werte der Beschleunigungen aus den initialen Werten der Positionen zu berechnen. In jedem Zeitschritt wird sie von `leapfrog()` verwendet, um die neuen Beschleunigungs-Vektoren aus den neu berechneten Positionen zu berechnen.

**Input-Parameter:** Vektoren der Positionen `x` und Massen `m`.

**Output-Parameter:** Vektor der Beschleunigungen `a`. Die Funktion akkumuliert die Beschleunigungen in dem Vektor, so daß der Vektor mit bei Eintritt mit 0 belegt werden muss.

Die Funktion nutzt die Symmetrie in der Wirkung zweier Körper $i$ und $j$ auf die Beschleunigung des jeweils Anderen ($a_{ij} = -a_{ji}$). Diese Funktion muss für jeden Körper über alle anderen Körper iterieren und besitzt daher die Komplexität $N^2$, sie trägt die Hauptlast der Berechnung. Wir werden ausschließlich diese Funktion parallelisieren!