Uniwersytet Jagielloński
Wydział Matematyki i Informatyki
Zespół Katedr i Zakładów Informatyki Matematycznej

Wielowątkowa symulacja N ciał z implementacją w architekturze CUDA

Autor
Damian Stachura

Opiekun
dr Piotr Danilewski

June 10, 2018

Contents

1 Przedstawienie problemu symulacji N ciał

Symulacja N ciał jest zagadnieniem z mechaniki klasycznej, które polega na wyznaczeniu toru ruchów wszystkich ciał danego układu o danych masach, prędkościach i położeniach początkowych w oparciu o prawa ruchu i założenie, że ciała oddziałują ze sobą zgodnie z prawem grawitacji Newtona.

1.1 Szczególne przypadki

Problem wyznaczenia ruchu dowolnej liczby ciał jest trudny, wielu naukowców próbowało rozstrzygnąć go dla małej, ustalonej liczby ciał.

1.1.1 Problem dwóch ciał

Problem dla dwóch ciał podlegających prawom klasycznej dynamiki Newtona i przyciągających się zgodnie z newtonowskim prawem powszechnego ciążenia został rozstrzygnięty przez J. Bernoulliego przy założeniu, że masa obiektu koncentruje się w jego środku. [Rog71] Ruch dwóch ciał wygląda wtedy tak, że obiekty poruszają się po krzywych stożkowych oraz rodzaj krzywej zależy od całkowitej energii układu. Przykładowo, w przypadku małej energii, gdy ciała nie mogą się od siebie uwolnić, to krążą wokół siebie po elipsach. W innych przypadkach obiekty mogą poruszać się chociażby po hiperboli.

1.1.2 Problem trzech ciał

Problem dla N = 3 wciąż nie jest rozwiązany w ogólności. Istnieją rozwiązania dla szczególnych przypadków[Ala00; GLo03]. Innym ważnym przypadkiem jest problem, w którym masa jednego z ciał jest zaniedbywalnie mała, jest to tak zwany ograniczony problem trzech ciał - przedstawiony przez J. L. Lagrange'a w XVIII wieku. Badał on układ Słońce-Ziemia-Księżyc.

1.2 Zastosowania

Symulacje N ciał są szeroko wykorzystywanymi narzędziami w fizyce oraz astronomii. Problemami, w których symulacje są użyteczne są na przykład dynamika systemu z kilkoma ciałami jak układ Słońce-Ziemia-Księżyc. W fizycznej kosmologii from investigating the dynamics of few-body systems like the Earth-Moon-Sun system t understanding the evolution of the large-scale structure of the universe.[1] In physical cosmology, N-body simulations are used to study processes of non-linear structure formation such as galaxy filaments and galaxy halos from the influence of dark matter. Direct N-body simulations are used to study the dynamical evolution of star clusters. Common examples include a satellite orbiting a planet, a planet orbiting a star, two stars orbiting each other (a binary star), and a classical electron orbiting an atomic nucleus (although to solve the electron/nucleus 2-body system correctly a quantum mechanical approach must be used).

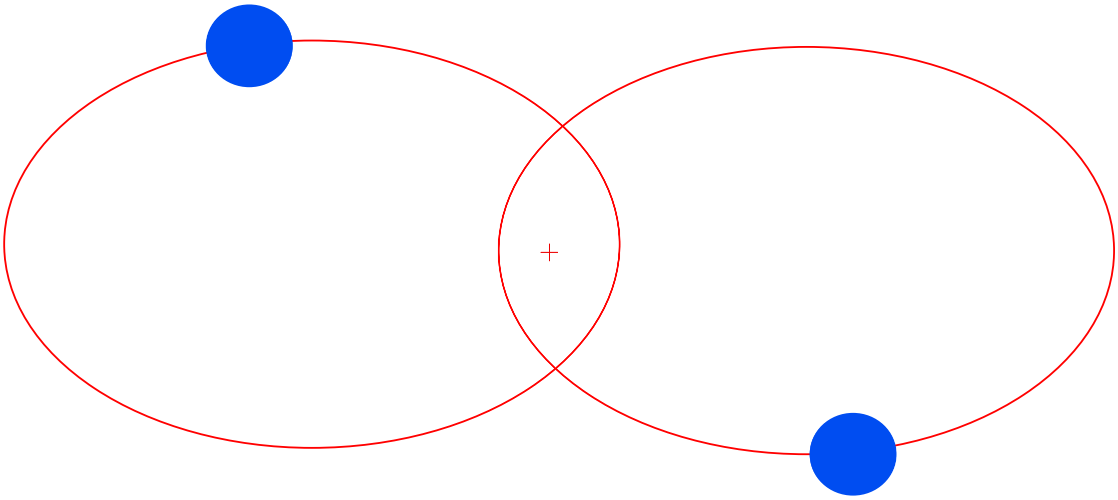

Figure 1: Symulacja dwóch ciał poruszających się po elipsach

1.3 Implementacja i wykorzystane technologie

W pierwszej części mojej pracy przedstawię implementację naiwnego algorytmu symulacji N ciał, który w każdym ruchu dla każdego ciała wyznacza jego ruch w oparciu o interakcję z pozostałymi obiektami, więc każda tura działa w czasie $\mathcal{O}(N^2)$. Kolejnym etapem mojej pracy będzie paralelizacja tego algorytmu. W drugiej części przybliżę moją implementację algorytmu Barnesa-Huta, który w wersji jednowątkowej ma złożoność obliczeniową $\mathcal{O}(N \log N)$, a następnie jego zrównolegloną wersję.

Repozytorium jest dostępne pod tym linkiem. Całość zaimplementowana jest w języku C++ z wykorzystaniem poniżej wymienionych technologii
Instalacja do poniższych technologii jest zamieszczona w repozytorium (dla systemów Linux oraz Windows).

1.3.1 Architektura CUDA

CUDA to uniwersalna architektura procesorów wielordzeniowych (głównie kart graficznych) umożliwiająca zaimplementowanie ich mocy obliczeniowej w wielu problemach, które mogą się wykonywać zarówno sekwencyjne i wielowątkowo. Wykorzystałem CUDĘ w wersji v9.1.85 do paralelizacji dwóch powyżej wspomnianych algorytmów.

1.3.2 Thrust

Thrust jest szablonową biblioteką dla CUDA bazująca na bibliotece STL z C++. Thrust umożliwia implementację aplikacji wielowątkowych przy minimalnym wysiłku programistycznym za pośrednictwem interfejsu wysokiego poziomu, który jest w pełni zgodny z CUDA C. Wykorzystałem ją do łatwiejszego przenoszenia danych między CPU oraz GPU. Korzystałem z wersji v9.2.88.

1.3.3 OpenGL

OpenGL jest API do tworzenia grafiki. Skorzystałem z OpenGL3, w celu zaprezentowania symulacji w 2D.

2 Pierwsze podejście

2.1 Sformułowanie problemu

W celu przedstawienia ogólnego sformułowania problemu potrzebujemy przytoczyć następujące trzy prawa dynamiki sformułowane przez Isaaca Newtona [Rog71]

Prawo 1. *Każde ciało pozostaje w stanie spoczynku lub ruchu jednostajnego w linii prostej, chyba że jest zmuszony zmienić ten stan zewnętrzne oddziaływanie z innymi ciałami, czyli każde ciało jest w układzie inercjalnym.*

Prawo 2. *Szybkość zmiany pędu jest proporcjonalna do siły wywieranej i znajduje się w tym samym kierunku co siła.*

Co oznacza, że w inercjalnym układzie odniesienia zachodzi równość $F = ma$, gdzie F jest wektorem sum sił działających na obiekt, m to masa obiektu, a to jego przyśpieszenie.

Prawo 3. *Każdej akcji towarzyszy reakcja równa co do wartości i kierunku, lecz przeciwnie zwrócona.*

Co oznacza, że jeśli ciało A działa na ciało B siłą F (akcja), to ciało B działa na ciało A siłą (reakcja) o takiej samej wartości i kierunku, lecz o przeciwnym zwrocie.

Niezbędne jest również przytoczenie prawa powszechnego ciążenia Newtona

Prawo 4. *Każdy obiekt przyciąga każdy inny obiekt z siłą, która jest wprost proporcjonalna do iloczynu ich mas i odwrotnie proporcjonalna do kwadratu odległości między ich środkami.*

Czyli między dowolną parą ciał posiadających masy pojawia się siła przyciągająca, która działa na linii łączącej ich środki, a jej wartość rośnie z iloczynem ich mas i maleje z kwadratem odległości.

Aplikując to prawo do symulacji N ciał, uzyskujemy że na każde i^{th} ciało działa siła F_i zdefiniowana następująco:

$$F_i = -G \cdot m_i \sum_{j=1, j \neq i}^{N} \frac{m_j(r_i - r_j)}{|r_i - r_j|^3},$$

gdzie G to stała grawitacji, m_i masa ciało na które oddziałują inne ciała, m_j masy ciało oddziałujących na i^{th} ciało, $r_i - r_j$ to różnica wektorów pozycji dwóch ciał, $|r_i - r_j|$ to dystans między ciałami.

Z wykorzystaniem powyższych praw możemy podać następującą definicję

Symulacja N ciał - Dla N ciał mających ustalone masy oraz początkowe położenie oraz prędkość, ruch każdego obiektu jest symulowany z wykorzystaniem prawa powszechnego ciążenia oraz poprzez wyznaczenie przyspieszenia obiektu korzystając z drugiego prawa dynamiki Newtona.

2.2 Jednowątkowy naiwny algorytm z pseudokodem

Najprostszy algorytm dla problemu N ciał zadany pseudokodem może wyglądać tak:

Listing 1: pseudokod

```
ustaw mase oraz początkową pozycję i prędkość dla każdego ciała
while(true)
   for i in {1...N}:
       uaktualnijPozycje()
   narysujNowePozycje()
```

Prawo 5. *Uaktualnienie pozycji wszystkich ciał ma złożoność obliczeniową $\mathcal{O}(N^2)$.*

Dowód: Dla każdego ciała najpierw musimy wyznaczyć siłę działającą na nie poprzez interakcję z innymi ciałami, czyli dla każdego z N ciał musimy policzyć siłę oddziałującą nań z każdym innym obiektem, więc musimy policzyć wartość wzoru wynikającego z prawa 4 $N \cdot N - 1$ razy, czyli złożoność tej podoperacji $\mathcal{O}(N^2)$. Następnie dla każdego obiektu musimy wyznaczyć jego przyśpieszenie oraz nową pozycję i prędkość, co jesteśmy w stanie zrobić w czasie $\mathcal{O}(N)$. Poprzez zsumowanie złożoności obu podoperacji, widzimy że złożoność obliczeniowa jednego kroku symulacji naiwnego algorytmu wynosi $\mathcal{O}(N^2)$. □

Listing 2: compute next step

```
typedef thrust::host_vector<float> tf3;
void StepNaive::compute(tf3& positions, float dt) {
  std::fill(forces.begin(), forces.end(), 0);
  for(unsigned i=0; i<N; i++) {
    for(unsigned j=0; j<N; j++) {
      float distX = positions[j*3] - positions[i*3];
      float distY = positions[j*3+1] - positions[i*3+1];
      if(i!=j && fabs(distX) > 1e-10 && fabs(distY) > 1e-10) {
        float F = G*(weights[i]*weights[j]);
        forces[i*3] += F*distX/(distX*distX+distY*distY);
        forces[i*3+1] += F*distY/(distX*distX+distY*distY);
      }
    }
  }
  for(unsigned i=0; i<N; i++) {
    for(int j=0; j<2; j++) { // x, y
      float acceleration = forces[i*3+j]/weights[i];
      positions[i*3+j] += velocities[i*3+j]*dt + acceleration*dt*dt/2;
      velocities[i*3+j] += acceleration*dt;
    }
  }
}
```

2.3 Paralelizacja naiwnego algorytmu

2.4 Implementacja

3 Drugie podejście

3.1 Algorytm Barnesa-Huta z pseudokodem

3.2 Zrównoleglenie algorytmu Barnesa-Huta

sudo apt-get install texlive-full

3.3 Implementacja

$http : //www.deltami.edu.pl/temat/fizyka/mechanika/2015/11/26/Problem_dwoch_cial/$ apt-get install texlive-lang-polish
Random citation embeddeed in text.s
sudo apt-get install texlive-bibtex-extra
sudo apt-get install texlive-bibtex-extra biber
biber Praca
https://www.sharelatex.com/learn/Bibliography_management_with_biblatex
inkscape -D -z –file=drawing.svg –export-pdf=draw.pdf –export-latex

4 Podsumowanie

References

[Aar03] Sverre J. Aarseth. *Gravitional N-Body Simulations. Tools and Algorithms 1 edition.* Cambridge University Press, 2003.

[Ala00] Richard Montgomery Alain Chenciner. "A remarkable periodic solution of the three-body problem in the case of equal masses". In: *Annals of Mathematics* 152 (2000), pp. 881–901.

[Cora] NVIDIA Corporation. *CUDA C Programming Guide*. v9.1.85, 2018. URL: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (visited on 03/05/2018).

[Corb] NVIDIA Corporation. *NVIDIA CUDA Runtime API*. v9.1.85, 2018. URL: http://docs.nvidia.com/cuda/cuda-runtime-api/index.html (visited on 03/05/2018).

[GLo03] M.Kramer G.Lodge J. A. Walsh. "A Trilinear Three-Body Problem". In: *International Journal of Bifurcation and Chaos* 13 (2003), pp. 2141–2155.

[GOO] GOOGLE? *THRUST*. v9.2.88, 2018. URL: https://docs.nvidia.com/cuda/thrust/index.html (visited on 05/15/2018).

[Gro17] Khronos Group. *OpenGL API, OpenGL Shading Lanugage and GLX Specifications.* OpenGL 4.6. 2017. URL: https://www.khronos.org/registry/OpenGL/index_gl.php (visited on 07/30/2017).

[Lar07] Jan Prins Lars Nyland Mark Harris. "GPU Gems 3". In: 2007. Chap. Fast N-Body Simulation with CUDA. Chapter 31, pp. 677–694.

[Lin99] Tancred Lindholm. "Seminar presentation. N-body algorithms". In: *University of Helsinki* (1999).

[Mar11] Keshav Pingali Martin Burtscher. "GPU Computing Gems Emerald Edition". In: NVIDIA Corporation, Wen-mei W. Hwu, 2011. Chap. An Efficient CUDA Implementation of the Tree-Based Barnes HUT N-Body Algorithm. Chapter 6, pp. 75–92.

[Rog71] Jerry E. White Roger R. Bate Donald D. Mueller. "Fundamentals of astrodynamics". In: DOVER PUBLICATIONS, 1971. Chap. 1 TWO-BODY ORBITAL MECHANICS, pp. 1–49.