Project 2 – N body simulation

Overview

The purpose of this project is to implement the n-body simulation using CUDA and OpenMP, and compare the performance between the two.

The project is due Monday, December 8 through Sakai. You can work in groups of 2. We have provided a cluster for you to use, instructions on how to access it are on Sakai together with some example code that you can use to get familiar with OpenMP and CUDA.

Note: Most Nvidia cards support CUDA, so if your computer has one, you can download the SDK and develop in your computer. Most MacBook pros have Nvidias, as well as a lot of mid-high end laptops, it may be worth checking.

Problem description

When computing the interaction between two particles, Newton laws provide an explanation on how the forces change over time. This can be computed easily for two bodies, but things start to get complicated after that, and in order to see the evolution of the system, computer simulations must be performed. The n body problem tries to do exactly this, given n bodies, the idea is to use Newton's equations on discrete time steps to approximate the evolution of the system over time.

Algorithm

Given n bodies, we have that he force vector acting on body i can be calculated as

$$\vec{F_i} = -\sum_{i \neq j} \frac{G m_i m_j \left(\vec{p_i} - \vec{p_j}\right)}{\left(\left|\vec{p_i} - \vec{p_j}\right|^2 + \epsilon\right)^{3/2}},$$

Where

$\vec{F_i}$: Force on particle i
m_i: mass of particle i
m_j: Mass of particle j
p_i: Position of particle i
p_j: Position of particle j
G: Gravitational constant
ϵ: Softening factor

Notice that each F_i can be computed independently, as it does not depend on the otherF_j, as such this exhibits data parallelism that can be exploited by using CUDA.

Also notice that this is a vector equation and $\left|\vec{p_i} - \vec{p_j}\right|^2$ means the square of the Euclidian distance of the particles.

The softening factor has to be chosen manually, and it's purpose is to avoid division by very small numbers, which lead to high numerical errors, try starting with small ones, like 0.00001 and increase it or decrease it by factors of ten.

After computing all of the $F_i$the new positions must be calculated. For that recall the Newton laws, which states

$$F^{\rightarrow} = ma^{\rightarrow} \leftrightarrow a^{\rightarrow} = \frac{F^{\rightarrow}}{m}.$$

So you can calculate the new acceleration for every particle, and then using the equation

$$v^{\rightarrow} = v_{old}^{\rightarrow} + a^{\rightarrow}\Delta t,$$

You can calculate the new velocity from the old velocity. Here Δt is the simulation time step, also specified by you (the smaller it is, the more accurate, but also more demanding).

Having the velocities for each particle, then the final step is to compute the new positions, which can be done with the following

$$p^{\rightarrow} = p_{old}^{\rightarrow} + v^{\rightarrow}\Delta t.$$

Finally after getting the position for each new particle, you do it all over again.

Notice that computing the acceleration velocities and positions, is also independent of the rest, so you could parallelize those computations.

What you should do

You should implement the n body simulation using both CUDA and OpenMP, and compare the performance between both solutions. Try making plots for this.

Your algorithm is expected to work for thousands of bodies.

How to test

In order to test if your solution is correct, you have to visualize it while it is running, for this purpose you can use OpenGL in order to draw the particles at their positions after every simulation step (or maybe after k simulation steps if your simulation is running too fast, so you can skip some of them).

For this purpose you will be provided with a simple OpenGL code, which will show how to draw simple circles on screens at some given positions.

Input

The input to the problem will be a file with the following format, the first line will contain only one number n, the number of bodies to simulate. n Lines will follow, one for each particle, each containing 7 numbers, which are in order *mass, x position, y position, x velocity, y velocity, x acceleration, y acceleration*, which completely describe the state of the particle at time 0. A sample is

```
2
5.0 0.0 0.0 1.0 0.0 0.0 0.5
2.2 1.5 1.0 0.0 0.0 0.0 0.0
```

This specifies two bodies, the first of mass 5.0 at position (0.0, 0.0), with a starting velocity of (1.0, 0.0) and acceleration of (0.0, 0.0). The second particle is at (1.5, 1.0), and starts with 0 velocity and acceleration.

Output

Your program should display using OpenGL or something similar, the evolution of the system over time.