This is my readme file for my code repository on GitHub: `https://github.com/smsoltan/MD`

Project of simulating molecular dynamics

Overview

One way to obtain the macroscopic properties of matter is simulating its molecular dynamics [MD]. The underlying assumption here is that values of the specific parameters describing the system could be expressed as functions of microscopic parameters averaged over sufficiently long times (e.g. temperature is related to average kinetic energy of particles involved). According to the ergodic hypothesis this approach is equivalent to statistical physics approach based on statistical ensembles. However, simulating MD avoids some cumbersome analytical calculations that may be involved in statistical physics approach.
Due to the sheer number of particles in a physically meaningful sample, simulating MD requires long calculation times - unless the code is parallelised. The purpose of this project was to compare the times of simulating about 500 particles on CPU and on GPU, using NVidia graphic cards and CUDA programming tools.
This repository contains source codes of programs written for the project. More details are provided further below. Full overview together with the obtained results are provided in the original report (.pdf file, currently only in Polish).

Licensing

This project uses some open source code attached to the book *CUDA by examples* by Jason Sanders and Edward Kandrot. The appropriate license is included.

Synopsis of the simulation

At the beginning of the simulation, all particles are located at nodes of a cubic grid. All components of their velocity vectors are chosen at random with a bound that their center of mass is at rest. At each simulation step new positions and velocities are calculated. Periodic boundary conditions are used [2].
Any two particle's interaction is described by a known potential with a cut-off distance (a distance between them above which the potential is zero). Given that, two approaches may be taken:

- (naive) Mutual distance is checked for all pairs of particles and then calculations of mutual forces are performed. Therefore the calculation time grows as N^2 where N is the number of particles.
- (optimized) A simulation involves a so-called Verlet lists [2,3]: a list for each particle containing other particles that reasonably may interact with the original particle in a close future. This list is updated every certain number of steps. This is in principle less accurate than the naive approach, but the calculation time should scale approximately as N.

Implementation of second option differs from the original Verlet proposition (see *More details about the simulation* below). The Verlet algorithm could be further parallelized, shortening

the calculation time even further [4]. In this version however each Verlet list was processed sequentially on its thread.
The grid's constant b, the time of a single step h and the number of steps L are all changeable parameters of the simulation. The values referenced in the report are:

- b is taken from the range between 1.0 and 4.0
- $h = 0.01$ was used in the report. Note that the longer the time step, the less precise the calculations become. If the total energy is not conserved over time, then the time step should be shortened.
- The report indicates that, given the values above, about 100-200 steps are required to pass from the highly ordered initial state to a physically expected equilibrium state. The value of L should be appropriately adopted.

The number of particles N could also be varied, as indicated by programs' description below.

Installation (on Linux systems)

NVidia card with computing capabilities and nvcc compiler are required. Download all the files to a chosen directory. In a terminal, go to the directory and execute a command
`make`
to trigger Makefile. This will run commands to compile 5 programs, as well as create a directory `init_sets` with one file that may be used to initialize calculations. To remove the created files and the directory, type
`make clean`

Programs and scripts

`cpu_velocityverlet <initial_velocities> <output_file> <b> <h> <L>`
`cpuopt_velocityverlet`
`gpu_velocityverlet`
`gpuopt_velocityverlet`

As names imply, two first programs perform calculations on CPU, others - on GPU. Programs with `opt` in the name use the optimized approach. Both GPU programs use GPU lock-free synchronization [5].
All programs require the same arguments:

- `<initial_velocities>`: the name of the input file. Its' first line should contain the length of the edge of the simulated space (given in the number of particles alongside the edge). Subsequent lines should contain velocity coordinates of subsequent particles. Note that for physical reasons the average velocity should be zero, i.e. the center of mass should be at rest. Files in this format are created by `randv` and `random` described below.
- `<output_file>`: the name of the output file. First line have the information about parameters and calculation time. For GPU programs, this value does not take into account the time needed to initialize kernel and to copy information back to CPU. Next lines of the file contain the values of kinetic energy, potential energy and total energy of the whole system.
- `<b>`, `<h>` and `<L>` - free parameters of the simulation, described in the synopsis above.

`randv <name> <n>`

Creates a set of random initial velocities for particles obeying the rule that the center of mass is at rest. The set is named `<name>.dat` and is formatted as required by simulation programs. The particles are assumed to be initially located on a 3D grid $n \times n \times n$, and so each set contains data for $N = n^3$ particles.

`randoms <name> <n>`

This is a bash script that invokes randv program five times. The sets are saved in `init_sets` folder under names `<name>i.dat` where `i` runs from 0 to 4.

`visualisation.py`

Simple *ad hoc* python script to visualize the data. When run, it initializes a user dialog in order to specify the name of the input and the output. It plots kinetic, potential and total energy versus time.

More details about the simulation

The interaction of any two particles in this simulation separated by r is described by the Lennard-Jones potential:

$$V_{LJ} = 4\epsilon \left(\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right) + const$$

where ϵ is the depth of the density well and σ is the value of r for which the potential starts to be very small. Due to that, a cut-off is introduced: the above equation is used only for $r < R_C$ and the potential is taken to be 0 otherwise. In order for the potential to be continuous at $r = R_C$, the constant is taken to be equal

$$const = -4\epsilon \left(\left(\frac{\sigma}{R_C} \right)^{12} - \left(\frac{\sigma}{R_C} \right)^{6} \right)$$

In the simulation, $R_C = 2.5\sigma$. Both σ and the mass of the particle are used as units in the simulation and hence are set to 1 by definition.
In principle to simulate MD one has to check the mutual interaction of every pair of particles. But, given the cut-off, some optimization may be introduced. For a given particle, one could assign a list - so called Verlet list - of particles that have the chance to interact in the near future. This list is actualized every specified number of steps, by finding all the particles that are closer than certain threshold R_M . This number of steps is usually given as a constant of simulation (dependent on other parameters). Here, a more "paranoid" version was implemented: this number is calculated based on the largest velocity v_{max} registered since the last actualization: $R_M - R_C \simeq m \cdot 2v_{max} \cdot h$, where m is the number of steps until next actualization and h is the timestep of the simulation. R_M was taken to be equal 3.3σ.

Initially, particles are located on a cubic grid with random velocities. In each step of a simulation, new positions and velocities are calculated by the Verlet algorithm (that approximates the proper Newton dynamics):

$$\vec{r}(t+h) = \vec{r}(t) + \vec{v}(t)h + \frac{1}{2}\vec{a}(t)h^2$$

$$\vec{v}(t+h) = \vec{v}(t) + \frac{1}{2}[\vec{a}(t) + \vec{a}(t+h)]h$$

Literature

[1] J. Sanders, E. Kandrot, *CUDA by Example. An introduction to general-purpose GPU programming*, Addison-Wesley 2011

[2] M. P. Allen, *Introduction to molecular dynamics simulation*, Computational soft matter: from synthetic polymers to proteins **23**(1), 1-28, 2004.

[3] L. Verlet, *Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules*, Physical Review **159**(1), 98-103, 1967

[4] T. Lipscomb, A. Zou, S. S. Cho, *Parallel Verlet Neighbor List Algorithm for GPU-Optimized MD Simulations*, Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine, 321-328, 2012

[5] S. Xiao, W. Feng, *Inter-Block GPU Communication via Fast Barrier Synchronization*, 2010 IEEE International Symposium on Parallel and Distributed Processing (IPDPS), IEEE, 2010.