Informatik und Angewandte Kognitionswissenschaft
Lehrstuhl für Hochleistungsrechnen

UNIVERSITÄT DUISBURG ESSEN
Open-Minded

Rainer Schlönvoigt
Thomas Fogal
Prof. Dr. Jens Krüger

High-Performance Computing

http://hpc.uni-duisburg-essen.de/teaching/wt2013/pp-nbody.html

Exercise 1 - Serial N-Body (80 Points)

All assignments are to be uploaded to Moodle. Assignments are due at midnight on the due date. No late assignments will be accepted.

All assignments must include a `Makefile` for compiling your assignments. The assignment specification should include what the default target of your makefile should be. Assignments which do not compile will receive 0 points. Sometimes, we provide sample inputs and outputs; assignments which do not satisfy these test inputs will receive very few points.

Please do not include additional output other than what was requested by the assignment details. *Hint: if you want more debugging output, use a 'debug' flag in your program's arguments and have it only print when that flag is active.*

Your assignment will be graded on the `duecray.uni-due.de` supercomputer. It does not matter if your program runs correctly on another machine; it must run correctly on `duecray` to receive credit.

1 N-Body Simulation

In this assignment your task will be to write a serial n-body-simulation to be executed on a single processing unit.

1.1 Quick Reminder

In an n-body-simulation we try to simulate the gravitational effects that multiple objects have on each other. The necessary computational workload grows very fast with the amount of objects. This is due to the fact that we have to compute how every single object is being affected by every other object.

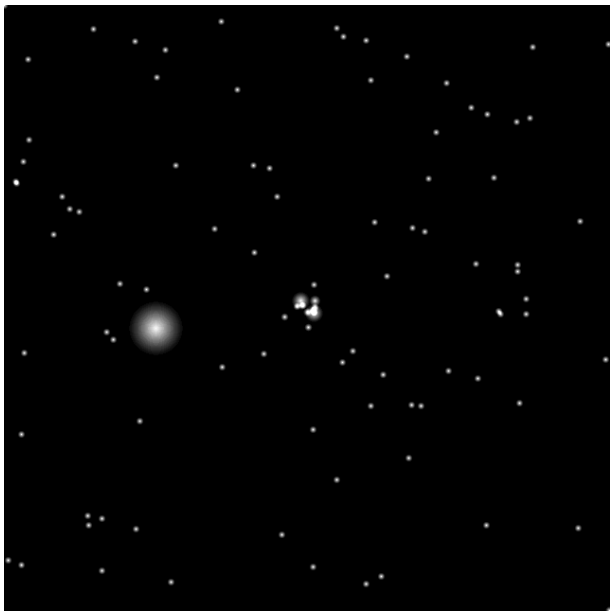

Figure 1: Example

1.2 Relevance

In reality we should have to calculate the gravitational influence for every possible point in time—a continuous variable. Since that isn't possible on computers, we have to settle for discrete timesteps. This assignment will be the foundation for upcoming assignments. The calculations that have to be done within every timestep offer themselves for parallelization and future assignments will therefore be about adapting the code to be executed in parallel. Having a proper architecture to begin with will save you time in the future. In later assignments there will be less information provided.

1.3 Overview

Figure 2 might be helpful to give you a better understanding of what needs to be done.

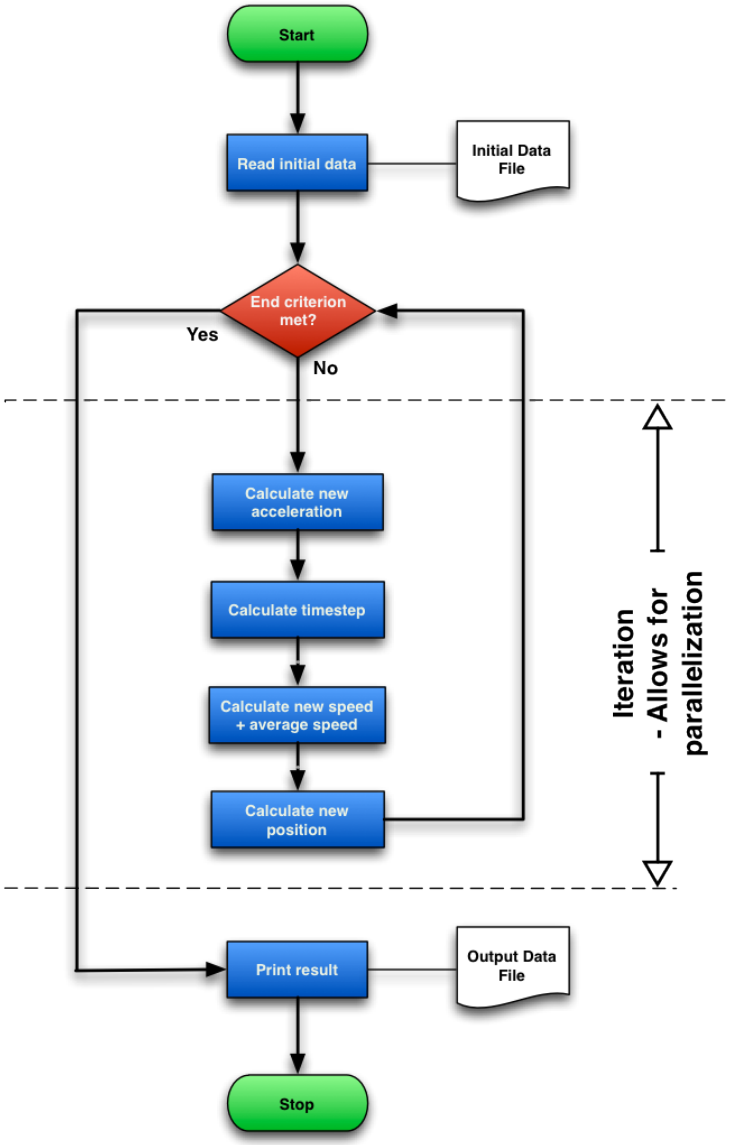

Figure 2: Overview

2 Input File Format

The input files that your simulation must read follow the format given in Listing 1:

```
5, 30000000, 0.5, 0.00001, 1,
 1,      1,      1,     20000000
 1000,   1000,   1000,  200000000
-1000,  -1000,  -1000,  200000000
-15000,  2000,   0,     900000000
 1500,  -1500,   0,     70000000
```

Listing 1: Simple input file.

The very first line consists out of configuration parameters. In order, these are:

1. the total amount of objects
2. an "end time criterion" at which to stop the simulation
3. a value δ by which to normalize Δt (more in 3.2)
4. an alternative Δt (more in 3.2)
5. a minimum distance to use in the simulation (more in 3.1)

Every line afterwards describes one object with its x-, y-, z-coordinates and its weight.

3 Iterator

Once reading and writing files works, we can take a look at the main part of your program; You will have to write code that starts and executes the iteration loop. What we want to calculate is where every object is going to be after every iteration step. For a hint at the necessary steps take another look at Figure 2. Some of those steps require additional precautions, which are described in this section.

3.1 Calculate new acceleration

The basics of this step are causing most of the necessary computational load. You will have to calculate the acceleration that every object will be affected by during the current timestep. To get this total acceleration of one object

you will have to accumulate the acceleration influence of all other objects on your current object.

$$\vec{p_i} = G \sum_{j=0|j\neq i}^{N} \frac{m_j(p_j - p_i)}{\|p_j - p_i\|^3} \tag{1}$$

Note that $\|var\|$ is the vector norm[1] of the vector var.

Unfortunately the formula gets very instable for distances near zero. To counter unwanted side effects, the configuration data of the input file contains a minimum distance. Should your objects be closer than this distance, use the minimum distance instead.

Keep in mind that the acceleration is a vector and therefore has a magnitude and a direction.

3.2 Update the current time

As mentioned before, we can only work in discrete timesteps. Choosing the right size is very important. Fast objects or objects receiving a strong acceleration need very small timesteps or the results will become inaccurate. Slow objects on the other hand are less time-critical.

To make things a little simpler we are going to use the same Δt for all objects. Each individual Δt is being calculated using the following formula:

$$\Delta t_i = \delta \times \min\left(\frac{1}{\|\vec{p_i}\|}, \frac{1}{\sqrt{\|\vec{p_i}\|}}\right)$$

Do not perform a division by zero when the acceleration or the velocity is zero! If that is only the case for one of the two, simply use the other one. In the extremely unlikely case, that both speed and acceleration are zero, use the alternative Δt supplied in the input file.

Due to difficulties in this part of the assignment, we have decided to allow an alternative time implementation. The alternative is simply to use a constant Δt for the entire simulation run. If you would like to use this simplified implementation, then the third

[1] https://en.wikipedia.org/wiki/Norm_(mathematics)#Euclidean_norm
https://de.wikipedia.org/wiki/Vektor#L.C3.A4nge.2FBetrag_eines_Vektors

value of your input file (in Listing 1, this would be 0.5) can be used as the amount to step the time. In this case, your 'time' loop is simply as given in Listing 2. In that example, '`increment`' is the third value of the input file.

```
for(float time=0.0f; time < end_time_criterion; time += increment) {
  ...
}
```

Listing 2: Outer simulation loop.

3.3 Calculate new speed, average speed and position

The integration scheme we will use for this course is similar to leapfrog integration, with some small modifications to make it easier to calculate.

$$a = F(...) \tag{2}$$
$$v_{i+1} = v_i + a\Delta t \tag{3}$$
$$x_{i+1} = x_i + v_{i+1}\Delta t \tag{4}$$

F is simply a function which computes Equation 3.1 from earlier.

4 File Handler

We will supply you with some example input and output files. Your first step should be to write

- a simple data structure in which to keep the data stored in the input files,
- code that helps you with reading the input files and
- code that helps you with writing your results back out into a file.

5 Output Format

You will need some way to verify your code is working as expected. For that, your program should output a CSV (comma-separated values) file for a timestep. You do not necessarily need to output every timestep—this will

create more output than you can reasonably analyze. More details on CSV files can be found by searching the web, but they are simple; your CSV file should have the following format:

```
x, y, z, scalar-name, scalar-name, scalar-name, ...
X0, Y0, Z0, scalar1, scalar2, scalar3, ...
X1, Y1, Z1, scalar1, scalar2, scalar3, ...
X2, Y2, Z2, scalar1, scalar2, scalar3, ...
X3, Y3, Z3, scalar1, scalar2, scalar3, ...
```

Listing 3: CSV output file format

where $X0, Y0, Z0$ gives the location of a particle, and *scalar*1, *scalar*2, usw. are associated scalar values. Note that it is important to output a space after the comma! Scalar values can be anything, but should be consistent: every line should have the same number of values, and a column should have the same meaning within each line. Examples of variables you might want to include are the particle's velocity and/or acceleration. Note these are both vectors, but just as the 3-component position was written out as `X0, Y0, Z0`, other 3-component variables can be output as 3 separate scalars.

The first line of this file is a 'header' line, which simply gives a name to each column. You must have a name for every field, otherwise common tools will not be able to read the file; likewise, you cannot have more fields in the header than you have in the actual file.

In most cases, these values should be floating point numbers. You can output them using (e.g.) "`%7.4f`" in your C program.

As a concrete sample of the output format, see Listing 4. This file describes 4 particles, each with 3 associated scalars (presumably the 3 components of the particle's velocity).

```
x, y, z, vx, vy, vz
07.3713,   0.3218,   3.1043,  1.1833, 2.0134,   2.2291
10.2746,  26.4729,  70.5028,  8.0630, 8.4932,   6.0171
02.2911,  09.1276,  10.8019,  6.3234, 1.1237,  15.5061
24.1717,  10.4758,  15.2072,  2.0334, 4.7838, 101.1451
```

Listing 4: Sample CSV output file

If the file describes (z.B.) timestep 164, and thus it would make sense to store in a file named "`164.csv`".

6 *sine qua non*

Pack your file into a single compressed tarball for submission. The tarball should extract all files into a single subdirectory. Upload your tarball to

Moodle.