Performance oriented programming

J. Daniel García Sánchez (coordinator)

Computer Architecture
Computer Science and Engineering Department
Universidad Carlos III de Madrid

1 Objective

This project has as a fundamental goal to help students to acquire **concerns on performance of sequential programs** and to familiarize with **their optimization**. In addition, they will get familiarize with **performance evaluation techniques**.

More specifically, the project focuses in the development of sequential software in the C++ programming language (including the latest versions).

2 Project description

To carry out this project you must implement in C++ a gravitation simulation application for a set of objects. The program must only use sequential programming techniques without making use of any kind of concurrency or parallelism.

Tow versions of the program will be implemented using the arrays of structures (**aos**) and structures of arrays (**soa**) techniques.

For a successful execution of the project, the following sections of this statement provides the requirements that must be satisfied by the developed programs. Besides, a binary file with a version of the executable program to make easier results comparison.

2.1 The gravitational interaction problem

The problem you must solve consists of the simulation over time of the positions of a set of objects in an enclosure. This computation is performed during the program execution at regular time intervals with length Δt.

The program must perform the simulation in a 3D space. We consider the space a closed enclosure. Consequently, when an object impacts a plane in the border of the enclosure, a rebound happens, changing the direction of the object motion.

Finally, in every time increment Δt possible collisions between different objects are checked.

In this way, it is expected that students simulate the behavior of N different objects moving in a 3D space.

2.2 Requirements

2.2.1 Program execution and input parameters

- Teams will develop two programs named **sim-soa** (structure of arrays version) and **sim-aos** (array of structures version).
- The program will take the following parameters for its correct execution:
  - **num_objects**: integer number, greater or equal than 0, indicating the number of objects that will be simulated.
  - **num_iterations**: integer number, greater or equal than 0, indicating the number of iterations (time steps) that will be simulated.
  - **random_seed**: positive integer number to be used as a seed for random distribution generator functions.

    Note: Two simulations with the same parameters but different seed will lead to different scenarios.
  - **size_enclosure**: real positive number indicating the size of the simulation enclosure. The enclosure is considered to be a perfect cube with a vertex in the coordinates origin and with side equal to **size_enclosure**.

    Consequently, if the value for **size_enclosure** is n, the opposite vertices fo the cube will have coordinates $(0, 0, 0)$ and (n, n, n).
  - **time_step**: real positive number indicating the time increment for each iteration of the simulation.
- When the number of parameters is wrong, the program will terminate with code error -1 and an error message to the standard error output.

```
user@machine:~$
user@machine:~$./sim-soa 100 2000 1
sim-soa invoked with 3 parameters.
Arguments:
  num_objects: 100
  num_iterations: 2000
  random_seed: 1
  size_enclosure: ?
  time_step: ?
```

- When any parameter is wrong, the program will terminate with error code -2 and an error message to the standard error output.

```
user@machine:~$
user@machine:~$./sim-soa 100 2000 1 -1000000 0.1
Error: Invalid box size
sim-soa invoked with 5 parameters.
Arguments:
  num_objects: 100
  num_iterations: 2000
  random_seed: 1
  size_enclosure: -1000000
  time_step: 0.1
```

- Once all input parameters have been processed and initial positions and velocities have been processed for all simulation elements, the program will write this information to a file named **init_config.txt**, with the following format:

- All real values will be printed with tree decimal places in fractional part.
- A header line including:
  - Size of the enclosure.
  - Time step.
  - Number of initial objects.
- One line per object where the position (x, y, z coordinates), velocity (x, y, z components) and mass must be written. This seven values must be separated by blanks.
- Example for an enclosure of size 1000000.0, with time step 0.100 and 2000 initial objects.

```
1000000.000 0.100 2000
903604.026 850236.140 783820.465 0.000 0.000 0.000 999999999999999868928.000
135885.825 224540.656 99650.335 0.000 0.000 0.000 1000000000000000262144.000
22087.739 685842.870 654084.972 0.000 0.000 0.000 999999999999999737856.000
...
```

Note: This is just an example of file format. Content data does not need to correspond to those obtained with the example program and input parameters given in the example.

2.2.2 Simulation parameters generation

- Object position in the 3D space (with coordinates in floating point double precision), and mass must be generated following random distributions as follows:
  - A single pseudo-random number generator engine instance must be used for all distributions.
  - It must be a 64 bits Mersenne-Twister generator (**std::mt19937_64**. This engine will be initiated with a random seed received as an argument by the program. (Information is available at `https://en.cppreference.com/w/cpp/numeric/random/mersenne_twister_engine`).
  - All position values will be generated using a uniform distribution with lower bound equal to **0** and upper bound equal to **size_enclosure**, using an object of type **uniform_real_distribution**. (Information is available at `https://en.cppreference.com/w/cpp/numeric/random/uniform_real_distribution`).
  - All mass values will be generated using a normal distribution with mean equal to 10^{21} y standard deviation equal to 10^{15} using an object of type **normal_distribution**. (Information is available at `https://en.cppreference.com/w/cpp/numeric/random/normal_distribution`).
- Generation order of random values is important. Values will be generated object by object and following within an object the order $< x, y, z, m >$
  - x_0, y_0, z_0, m_0.
  - x_1, y_1, z_1, m_1
  - x_2, y_2, z_2, m_2
  - ...

2.2.3 Attraction forces computation

Note: students are recommended to follow the described order in operations to make sure they get correct results.

- **Gravitational force computation**: Gravitational constant (G), object masses of both objects (m_i y m_j), and position vectors ($\vec{p}_i$ y $\vec{p}_j$) will be used.

  $$\vec{F}_{ij} = \frac{G \cdot m_i \cdot m_j}{||\vec{p}_j - \vec{p}_i||^2} \cdot \frac{(\vec{p}_j - \vec{p}_i)}{||\vec{p}_j - \vec{p}_i||} = \frac{G \cdot m_i \cdot m_j \cdot (\vec{p}_j - \vec{p}_i)}{||\vec{p}_j - \vec{p}_i||^3}$$

  Where the norm on a vector $\vec{v}$, is the **euclidean norm**.

  The universal gravitational constant will be the value $6.674 \cdot 10^{-11}$.

  For an object **i**, this force is applied with a positive value and for object **j** with negative value. An object does not apply any force on itself.

  $$F_{ij} = -F_{ji}$$

  $$F_{ii} = 0$$

- **Acceleration vector computation**: Acceleration ($veca$) is determined using its mass (m) and all the force contributions.

  $$\vec{a}_i = \frac{1}{m_i} \sum_{j=1}^{N} \vec{F}_{ij}$$

- **Velocity vector computation**: Velocity ($\vec{v}$) is determined using its acceleration ($\vec{a}$) and the time step (Δt):

  $$\vec{v}_i = \vec{v}_i + \vec{a}_i \cdot \Delta t$$

- **Position vector computation**: Position is determined using the velocity ($\vec{v}$) and the time step (Δt):

  $$\vec{p}_i = \vec{p}_i + \vec{v}_i \cdot \Delta t$$

- **Rebound effect**: If the position passes one of the enclosure bounds:

  1. The object is placed in the corresponding bound plane:

     $$p_x \leq 0 \Rightarrow p_x \leftarrow 0$$

     $$p_y \leq 0 \Rightarrow p_y \leftarrow 0$$

     $$p_z \leq 0 \Rightarrow p_z \leftarrow 0$$

     $$p_x \geq tam \Rightarrow p_x \leftarrow tam$$

     $$p_y \geq tam \Rightarrow p_y \leftarrow tam$$

     $$p_z \geq tam \Rightarrow p_z \leftarrow tam$$

  2. Besides the corresponding velocity component changes its sign.

2.2.4 Object collisions

When two objects collide they collapse into an object with larger mass. Two objects are considered to collide when their distance is less than the unit.

In this case both objects (a and b) lead to a single object (c) with the following properties:

- New mass results from mass addition ($m_c = m_a + m_b$).
- New velocity results from velocity addition ($\vec{v}_c = \vec{v}_a + \vec{v}_b$).
- New position is the position of the first object ($\vec{p}_c = \vec{p}_a$).
- Object a is replaced by object c.
- Object b no longer exists.

Note: Keep in mind that in those cases there is on object less in the system.

IMPORTANT: Collision checks must be performed at the end of each simulation iteration. Additionally, this check must also be performed before starting the simulation.

2.2.5 Data storage

Once all simulation iterations have finished the program shall store final data into a file named **final_config.txt**, with the same format and contents than the file **init_config.txt**.

3 Tasks

3.1 Sequential version development

This task consists in the development of the sequential version of the described application in C++17 (or C++20).

3.1.1 Compiler configuration

All your source files must compile without problems and shall not emit any compiler warning. In particular, the following compiler warning specific flags must be enabled:

- `-Wall -Wextra -Wno-deprecated -Werror -pedantic -pedantic-errors`,

Keep also in mind that you will have to perform all evaluations with compiler optimizations enabled (compiler options `-O3` y `-DNDEBUG`). You can easily get this with `-DCMAKE_BUILD_TYPE=Release`.

You are allowed to use additional compiler flags as long as you document them in the project report and justify its use. Such flags must be properly included in the CMake configuration file.

3.1.2 Libraries

You are allowed to use any element from the C++ standard library that is included in your compiler distribution. Note that this includes many header files like **<vector>, <list>, <fstream>, <cmath>, <random>, . . .** .

In general, additional libraries are not allowed. However, permisions can be granted by the subject coordinator if a justification is given.

3.2 Sequential performance evaluation

This task consists of the performance evaluation of the sequential application.

To carry out the performance evaluation you must measure the application execution time. You are expected to represent graphically your results. Keep in mind the following considerations:

- All evaluations must be performed in an university computer room. You must include in your report all the relevant parameters of the machine where you have run the experiments (processor model, number of cores, main memory size, cache memory hierarchy, ...) as well as system software (operating system version, compiler version, ...).
  - Alternatively, students are allowed to use for their evaluations their own computer provided that this has similar characteristics. In case of doubt, students are encouraged to contact the instructor in charge of the reduced group where they are enrolled.
- Run each experiment a given number of times and take the average value. You are recommended to perform each experiment at least 10 or more executions so that you can give a confidence interval.
- Study results for different object populations. Consider cases with 1000, 2000 and 4000.
- Study results for different number of iterations: Consider cases with 50, 100 and 200.

Represent graphically total execution times. Represent graphically average time per iteration.

Include in your report the conclusions you may infer from results. Do not limit to describing data. You must search for a convincing explanation of results.

4 Evaluation

The final grad for this project is computed as follows:

- **Achieved performance** (50%).
- **Functional tests** (20%).
- **Quality of the project report** (30%).

Warnings:

- If the delivered code does not compile, the final grade will be 0.
- In case of copy or any other kind of fraud, all implied groups will get a grade of 0.

5 Delivery procedure

The delivery of source code and the corresponding project report will be performed through Aula Global.

- **Source code delivery**
  - A compressed file in **zip** format will be delivered. This file will contain all source code for compiling the application.

- All source files (both **.hpp** and **.cpp**) used must be included.
- Additionally all CMake files used for the compilation process must be included.
- No resulting file (as binary object files or executable) shall be included.

- **Project report**
  - A project report in **PDF** format must be delivered.
  - Avoid including source code in the report. If needed make reference to delivered source code.
  - The report must include the following contents.
    - **Title page**. It will contain:
      - The project name.
      - The reduced group where the team is enrolled.
      - The assigned team number.
      - Name and NIA of all participants.
  - **A table of contents**
  - **Original design**. An explanation of the general design of the application specifying the general structure, the design choices taken, the considered alternatives and the main data types that have been used.
  - **Optimization**. It shall contain a discussion of applied optimizations and their impact. Optionally, it may include (if applicable) the use of additional optimization flags explaining their drawbacks and advantages.
  - **Performance evaluation**. It shall present the different performance evaluations that have been carried out, comparing the original developed version and the optimized version.
  - **Tests**. A description of the testing plan that has been executed to ensure the correct execution of both versions. It must contain functional tests and, if needed, unit tests.
  - **Conclusions**. Special attention will be devoted to those conclusions derived from results of performance evaluation as well as those that relate the carried out work with the contents of the course.

The project report shall have a maximum length of 15 pages including title page and all sections. In case it is longer, page 16 and beyond will not be taken into consideration for grading.