Parallel programming assignment with OpenMP

J. Daniel García Sánchez (coordinador)
Estefanía Serrano López
Alberto Cascajo García

Arquitectura de Computadores
Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid

1. Objective

The main objective of this assignment is the familiarization of the students with the **optimization of sequential programs** and with the **parallel programming models** in shared memory architectures.

Specifically, the assignment will be focused on the development of sequential software in C++ (including the changes in C++11/14), as well as the use of parallelism techniques with *OpenMP*.

2. Assignment description

For the accomplishment of the assignment the student must implement in C++ two version of the n-asteroid problem. One must be based in a sequential model and the other in a parallel model using OpenMP.

In order to correctly accomplish the assignment the student is given, we give in the following sections of this statement the requirements to which these programs must comply as well as a binary with the program already implemented to facilitate the comparison of results.

2.1. The n-asteroid problem

The problem that must be solved consists in the simulation through time, of the movement of a set of asteroids, taking into account the gravitational attraction forces produced between them. This computation is done executing the simulation in discrete intervals of time of length Δt.

The program must execute the simulation in a bidimensional space. This space is considered to be a closed space. When an object reaches the border of the space, the object bounces back, changing the direction of movement of the object.

Additionally, at the border of the space there are other fixed bodies (planets). They exert a gravitational force on to the asteroids, but they are not affected by the forces (they are fixed in space).

Finally, in each time increment Δt a laser ray is shot through the space crossing it horizontally. This ray will be fixed in a position defined as input parameter, and will be able to destroy all those asteroids in its path.

This way, the students must simulate the behaviour of these N asteroids moving in this bidimensional space, and a laser ray that, in each instant of time tries to destroy these asteroids.

2.2. Requirements

2.2.1. Execution of the program and input parameters

- Two different versions must be implemented: a sequential version and a parallel version.
  - **nasteroids-seq**: sequential version.
  - **nasteroids-par**: parallel version.
- The parameters received by the program for its correct execution are the following:
  - **num_asteroids**: is an integer number, greater or equal than 0, which indicates the number of asteroids that will be simulated.
  - **num iterations**: is an integer number, greater or equal than 0, which indicates the number of iterations (time steps) that will be simulated.
  - **num_planets**: is an integer number, greater or equal than 0, which indicates the number of planets that will be included at the border of the space.
  - **pos_ray**: is a double precision floating point number, which indicates the exact position of the ray in the space (y coordinate).
  - **seed**: is a positive integer number that serves as seed for the generator functions for random numbers.
  - **Note**: Two simulations with the same parameters but a different seed will lead to two different scenarios.
- In case one or more of the parameters are not correct or not present the program must be terminated with error code -1 and a corresponding error message.
  - In the case of **nasteroids-seq**:

```
nasteroids-seq: Wrong arguments.
Correct use:
nasteroids-seq num_asteroids num_iterations num_planets pos_ray seed
```

  - In the case of **nasteroids-par**:

```
nasteroids-par: Wrong arguments.
Correct use:
nasteroids-par num_asteroids num_iterations num_planets pos_ray seed
```

- Once all input parameters are processed and the initial positions of the simulated elements computed, a file with the name **init_conf.txt** must be generated with the initial data of the simulation:
  - Input parameters separated by a space in the same order in which they were introduced (without including the name of the program).
  - For each asteroid first, planet and ray, their position must be stored (x and y), as well as their mass (only for asteroids). Each element must be written in a line with their parameters separated by spaces. When printing floating point numbers, 3 decimals must be printed.
  - Example:

```
5 2 1 50.5 2000
2.567 2.573 18.000
3.545 2.523 14.233
45.545 9.000 15.644
2.567 2.556 19.555
89.965 2.335 18.577
15.7 0.00 180.577
0.000 50.500
```

Example for 5 asteroids, 2 iterations, 1 planet, position of the ray 50.5, and 2000 as seed.

NOTE: This is an example of the file format. The data contained does not have to correspond to those obtained with the program and input parameters given in the example.

2.2.2. Generation of the simulation parameters

- The position of the asteroids in the bidimensional space (two coordinates in double precision floating point position), must be generated based on a random distribution with the seed that has been included in the command line.

  For this purpose the following instructions must be used:

```
// Random distributions
default_random_engine re{seed};
uniform_real_distribution<double> xdist{0.0, std::nextafter(width,
                                  std::numeric_limits<double>::max())};
uniform_real_distribution<double> ydist{0.0, std::nextafter(height,
                                  std::numeric_limits<double>::max())};
normal_distribution<double> mdist{mass, sdm};
```

Listing 1: Code for the generation of the random numbers.

- To assign a value to the coordinate "x" of an asteroid (for example), the position can be obtained using the following expression. **xdist(re)**.

- The planets must be placed at the borders of the generated space, starting with the left axis ($x = 0$),the second at the top border ($y = 0$), the third at the right border, and the fourth at the bottom border. The second coordinate will follow a random distribution with the same seed. In terms of mass, it is computed in the same way as the asteroid, but multiplying the random value obtained by a constant value of 10.

The order in which the random numbers must be obtained is the following: first the position of all the asteroids and then the position for each planet.

2.2.3. Computation of the attraction forces

- **Distance**: For each asteroid, the program must compute the contribution of all the other elements (asteroid and planets) to its movement:

  1. Compute the distance between the asteroid and the other element:

     $$dist(a, b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

  2. **Rebound effect**: if the asteroid reaches a border of the space:

a) The asteroid must be positioned 2 points inside the space from the limit of the space:

$x \leq 0 \Rightarrow x \leftarrow 2$
$y \leq 0 \Rightarrow y \leftarrow 2$
$x \geq width \Rightarrow x \leftarrow width - 2$
$y \geq height \Rightarrow y \leftarrow height - 2$

b) Moreover, its velocity component at the conflict axis must be multiplied by -1 ($v_x = -v_x$ o $v_y = -v_y$).

3. **Normal movement**: If the distance between the asteroid and the other element is greater than 2 (minimum distance constant):

a) Compute the angle of influence:

1) Compute of the slope:

$$slope = \frac{y_a - y_b}{x_a - x_b}$$

2) Correction of the slope:

- If the slope is greater than 1 or less than -1:

$$slope = slope - Trunc(slope)$$

3) Computation of the **angle**: It is determined by the arctangent of the slope.

$$\alpha = atan(slope)$$

b) Compute the **attraction force**: the gravitational constant will be used (G), also, the mass of both asteroids (or asteroid and planet) (m_a y m_b), the distance between them ($dist(a, b)$) and the angle obtained previously (α). The maximum value of the force will be 200. Any computation that leads to greater values than this will be truncated to this maximum value:

$$f_x = \frac{G \cdot m_a \cdot m_b}{dist(a,b)^2} \cdot cos(\alpha)$$
$$f_y = \frac{G * m_a \cdot m_b}{dist(a,b)^2} \cdot sin(\alpha)$$

For the asteroid **a**, this force will be applied positively and for the second asteroid **b** negatively. An element will not exert force over itself.

c) For each force and angle computed, the following computation must be applied to the asteroids:

1) Calculation of the acceleration (a), using the mass(m) and the force (f) for each axis:

$$a = \sum \frac{f}{m}$$

2) Computation of the velocity (v) for each axis (x and y):

$$v = v + a \cdot \Delta t$$

3) Modification of the position (p) in x and y, due to the velocity in each axis:

$$p = p + v \cdot \Delta t$$

2.2.4. Laser ray execution

- After computing the movement due to the forces, the laser ray must be activated, locating all the asteroids that are inside its range of influence.
- All asteroids whose vertical coordinate is at a distance less or equal than 2 from the position of the ray (in the y axis) will be destroyed.
- The destruction of these asteroids imply the removal of these asteroids from the program.

2.2.5. Constants to be used

- Universal gravitational constant in a parallel universe: $gravity = 6{,}674e-5$.
- Time interval: $\Delta t = 0{,}1$.
- Minimum distance: $dmin = 2{,}0$.
- Space width: $width = 200$.
- Space height: $height = 200$.
- Width of the ray on each side of the position 2, in total the ray width is $ray\_width = 4$.
- Average for the computation of the normal distribution for the masses: $m = 1000$.
- Standard deviation for the computation of the normal distribution of the masses. $sdm = 50$.

2.2.6. Data storage

- Once all iterations are completed the final data must be stored in a text file named **out.txt**, including the following data:

  - Final position of the asteroids (x, y).
  - Their velocity (x, y).
  - Their masses.

  The data for each asteroid will be written in different lines with the parameters separated by spaces. When printing double precision floating point numbers, 3 decimals should be printed.

- Example:

```
2.567 2.573 45.567 28.573 18.000
8.567 102.573 4.567 67.573 19.760
```

  Example for a simulation in which 2 asteroids are left. **NOTE:** This is an example of the file format, The data contained does not have to correspond to those obtained with the program and input parameters given in the example.

3. Parallelism with OpenMP

For the implementation of the parallel version the functional requirements for the program are identical. This means that the **numerical results must be the same as the ones obtained in the sequential version**. The main difference is the usage of all the cores available inside a multi-core system employing the parallel programming model based on OpenMP.

Note: to check the results, the files generated by the students must be identical to those generated by the binary provided by the teachers with the same input values. When verifying this point it is recommended to use the command "diff" in Linux with both files.

4. Tasks

4.1. Development of the sequential version

This task consists in the implementation of the sequential version of the described application in C++14.

All the source code must compile without problems and without emitting any warnings from the compiler. In particular the following flags must be activated:

- `-std=c++14 -Wall -Wextra -Wno-deprecated -Werror -pedantic -pedantic-errors`,

Also, take into account that you must make the evaluations with the compiler optimizations activated (compiler option `-O3`, CMake option *CMake* `CMAKE_BUILD_TYPE=Release`).

You can use the mathematical library available in C++ and add the corresponding flags for linking.

4.2. Performance evaluation of the sequential version

This task consists in the evaluation of the performance of the sequential version.

To evaluate the performance the student must measure the execution time of the application. The results must be represented in a graphic. The student must take into account the following considerations:

- All evaluations must be done in the university classrooms. The student must include in the report all the relevant parameters of the machine in which the evaluation was executed (processor model, number of cores (real and virtual), memory size, cache hierarchy, ...) and of the system software (version of the operating system, compiler version, ...).
- Execute the experiment a number of times and take the average value. It is recommended to execute the program at least 10 times.
- Study the results for different number of asteroids and planets. Consider the following cases 250, 500 and 1000.
- Study the results for a different number of iterations: 50, 100 and 200.

Represent in a graphic all the performance data obtained from the experiments. Represent in another graphic the average time per iteration. Include in the report of this assignment the conclusions that you can infer from the results. Do not limit yourself to simply describe the data. A valid and justified explanation to the results must be also given.

4.3. Parallelization

This task consists in the implementation of the corresponding parallel version of the program using OpenMP. The student must explain in detail in the report the modifications done to the code.

The students must take into account the following considerations:

- Any assignment that emits a (*warning*) will be considered as failed.
- Your programs must include between the compiler parameters, at least, the following:

  `-std=c++14 -Wall -Wextra -Wno-deprecated -Werror -pedantic -pedantic-errors`

  or equivalent.

In the report the student must include the decisions that were taken to reach parallelization. For each decision the student must include other alternatives that could be considered and justify the decision made.

You can use the mathematical library available in C++ and add the corresponding flags for linking.

4.4. Performance evaluation of the parallel version

The evaluations from the first section must be repeated. Consider a different number of threads, from 1 to 16 (1, 2, 4, 8 y 16).

Important note: Please ensure that all examples have the flags for optimization activated (compiler option `-O3`,or CMake option *CMake* `CMAKE_BUILD_TYPE=Release`).

In addition to the graphics from the first section, represent graphically the *speedup*.

Include in the report the conclusions that you may infer from the results. Do not limit yourself to simply describe the data. A valid and justified explanation to results must be also given. The student must also take into account the impact of the computer's architecture for their conclusions.

4.5. Impact of scheduling

Study, for the cases of using 4 an 8 threads, the impact of the different models of scheduling (*static*, *dynamic* and *guided*)included in OpenMP. Include in the report the conclusions that you may infer from the results.

5. Grading

Final mark for this assignment will be obtained according to the following:

- Mark for the implementation: 100 %
  - Sequential implementation 40 %
    - Functional tests 30 %
    - Performance Achieved 30 %
    - Report 40 %
  - Parallel implementation 60 %
    - Functional tests 30 %
    - Performance Achieved 30 %
    - Report 40 %

Warnings:

- If the source code submitted does not compile, the final mark for the assignment will be 0.
- In case of detecting a copy, all the groups involved will obtain a mark of 0.

6. Submission

The submission of the source code will be done through Aula Global:

- Final code and report must be submitted to their corresponding submission link in Aula Global.

- Final code must be compressed in a .zip with the name **popenmp2017.zip**. It will contain:
  - A text file named **authors.txt** which will contain:
    - A line with the number of the group in which the students are enrolled.
    - A line for each student in the group with the NIA, surname and name.
  - **Folder with sequential code**: it will be named **seq** and **contain all files needed for compiling and executing the code**.
  - **Folder with parallel code**: it will be named **par** and **contain all files needed for compiling and executing the code**.

A different submission link will be published for the submission of the report. The report must contain at least the following sections:

- Cover with title: it will contain the name of the assignment, the name, surname and NIA of the authors and the group.
- Table of contents.
- Introduction.
- Sequential version: Section in which the implementation and optimizations of the sequential code will be explained. Do not include code in it.
- Parallel version: Section in which the implementation and optimizations of the parallel code will be explained. Do not include code in it.
- Performance evaluation: Section in which the explanation of the different evaluations completed are included. It must compare the initial program, your sequential version, and your parallel version.
- Tests: Description of the test plan executed to guarantee the correct execution of both versions.
- Conclusions.

The report must not have more than 15 pages including all sections and must be submitted in PDF format.