Parallel programming in OpenMP Lab

J. Daniel García Sánchez (coordinator)
David Expósito Singh
Javier García Blas

Computer Architecture
ARCOS Group
Computer Science and Engineering Department
University Carlos III of Madrid

1. Objective

The objective of this lab is guide the student to parallel programming in shared memory architectures. The lab is focused on making use of the OpenMP programming model.

2. Lab description

Two implementations of the N-body problem are provided. Both implementations are written in C++11. In both implementations, the strategy can be selected to be either AOS (*Array of structures*) or SOA (*Structure of Arrays*).

2.1. The N-body problem

The problem to be solved consists in the simulation over time of N bodies moving in a 2D space and considering gravitational attraction. This computation is performed over time in time increments Δt.

The program performs a simulation in a bi-dimensional space. The space is considered to be a closed box. Consequently, when an object impacts a box border, a rebound happens, and the movement direction changes.

2.2. Support code

The provided code consists of a set of class and library functions and two main programs.

Main programs are in directory *app*. The rest of source files are in directory *include* and *src*.

Program **nbodyfile** reads from a file the simulation setup and the initial state of objects to be simulated. The file contains a header line with simulation parameters followed by a set of lines (one per object), containing initial state for each object. It takes the following command line parameters:

- **mode**: Execution mode. It may be either *aos* (*array of structures*) or *soa* (*structure of arrays*).
- Input file name. If not specified, file `in.txt` will be used.
- Output file name. If not specified, file `out.txt` will be used.

Program **nbodyrnd** generates a set of objects in a pseudo-random way using two parameters: the number of objects and the number of simulation iterations. This program generates a given number of objects uniformly distributed in space and with a mass that follows a normal distribution. Program takes the following parameters:

- **mode** : Execution mode. It may be either *aos* (*array of structures*) or *soa* (*structure of arrays*).
- Number of bodies to be simulated. If not specified, default value is 50.
- Number of iterations for simulation. If not specified, default value is 100.
- Output file name. If not specified, file `out.txt` will be used.

3. Tasks

3.1. Performance initial evaluation

This task consists in an initial performance evaluation of the provided programs. You may use program *nbodyfile* to experiment with code and understand how it works. After that, run program *nbodyrnd* with different input parameters.

Important Note: Please, make sure that you compile all the examples with optimizations enabled (compiler option `-O3`, or *CMake* option `CMAKE_BUILD_TYPE=Release`).

To evaluate performance, you must measure application execution time. You must represent graphically results. Keep in mind the following guidelines:

- Perform each experiment a number of times and take the average value. A minimum of 10 executions is recommended.
- Study results for object population sizes from 250 objects to 1,000 objects in 250 increments (i.e.: 250, 500, 750, 1,000).
- Study results for different number of iterations from 50 to 200 in increments of 50.
- Remind that the goal is to compare *aos* and soa strategies.

Plot the obtained total execution time for each case. Represent in a different figure the average iteration time.

Include in the lab report the conclusions that you may infer from the results. Please, do not limit yourself to simply describing data. You must also look for a convincing explanation of the results.

3.2. Parallelization

This task includes the development of the corresponding parallel versions for the provided programs using OpenMP. You must provide in your lab report a detailed explanation of the modifications performed.

The lab report must also include the design decisions you have taken to achieve parallelization. For each decision, you must include which other alternatives could be considered and justify reasons for the selected choice.

3.3. Parallel version evaluation

Repeat evaluations from first section. Consider different number of threads, from 1 to 16 threads (1, 2, 4, 8, and 16).

Important Note: Please, make sure that you compile all the examples wit optimizations enabled with (compiler option `-O3`, or *CMake* option `CMAKE_BUILD_TYPE=Release`).

Besides from figures in the first section, plot the obtained *speedup*.

Include in the lab report the conclusions that you may infer from the results. Please, do not limit yourself to simply describing data. You must also look for a convincing explanation of results.

3.4. Impact of scheduling

Perform a study, for the cases of 4 and 16 threads, on the impact that different scheduling models have on performance (*static*, *dynamic*, and *guided*) provided by OpenMP.