CPS3227

Concurrency, HPC and Distributed Computing
Assignment: n-body Simulator

Adrian De Barro
Department of Computer Science
University of Malta

Academic Year 2017/2018

Instructions

- This is **an individual** assignment. This assignment carries **30**% of the final **CPS3227** grade.
- While it is strongly recommended that you start working on the tasks as soon as the related material is covered in class, the firm submission deadline is **25**th **May 2018 at Midnight**. Hard copies **are not** required to be handed in.
- You are to allocate approximately **32** to **40** hours for this assignment.
- A soft-copy of the report and all related files (including code) must be uploaded to the VLE by the indicated deadline. All files must be archived into a single .zip file. It is the student's responsibility to ensure that the uploaded zip file and all contents are valid.
- Reports (and code) that are difficult to follow due to low quality in the writing-style, organisation or presentation will be penalised.

Preliminaries

For this assignment you are required to:

- develop a parallel implementation of a simulator for the n-body problem (described in Appendix A) using C/C++, MPI and OpenMP;
- characterise its performance by plotting the speed-up against the number of processors used.

This assignment represents 35% of the total mark for this unit.

Provided Material

To help you start out, the following material is provided:

- source code for a sequential implementation of the simulator (`nbody.cpp` and `vector2.h`);
- test input files `input_64.txt`, `input_1024.txt`, `input_4096.txt` and `input_16384.txt` containing masses and initial positions for 64, 1024, 4096 and 16384 bodies respectively.

Input

The simulator is expected to read input from a file containing a list of body masses and initial positions separated by commas. For example:

Mass	X	Y
	...	
10,	-40,	7
4,	32,	23
6,	5,	-9
7,	16,	49
	...	

Computation

The simulator should compute the successive positions of the particles in the input set for a number of iterations, taking into consideration the influence of the Newtonian gravitational force. The simulation time increases monotonically by a constant delta with every iteration.

Your solution should possess the ability to scale up across shared-memory cores on a single node using OpenMP and scale out across distributed-memory nodes using MPI.

Output

The simulator should output the mass and position of every particle at each iteration of the simulation to a separate file called `nbody_`x`.txt`, where x is the iteration count. Naturally, this functionality should be disabled while measuring the performance of the simulator.

Performance Measurement

For each provided test input file, performance measurements are to be carried out for the following configurations:

- naïve shared memory (12 cores),
- distributed memory (4 nodes),
- hybrid (shared and distributed memory, 12 cores $\times$ 4 nodes) and

executing up to one thousand iterations in each case. Some configurations will take far too long to execute, so choose your test runs wisely. You should record the execution time against the number of processors, taking the average over multiple test runs for each result.

Deliverables

The following items should be submitted on VLE by 9^{th} January 2017:

- documentation in PDF, which (i) outlines the techniques used for parallel decomposition of the problem, (ii) describes the design of the performance measurement experiments, and (iii) plots and analyses the obtained speed-up results for each configuration;
- source code and executables for your parallel implementation;
- output files (`nbody_`x`.txt`) from your test runs;
- a signed plagiarism declaration.

Please do not submit printed copies of the project documentation.

A The n-body Problem[1]

A naïve implementation of the n-body problem is characterised by the following algorithm:

1. Allocate n bodies with random masses, positions and velocities
2. For each body in the simulation
   (a) Calculate the sum force exerted by all other bodies
   (b) Derive acceleration by dividing mass into computed force
   (c) Integrate acceleration over a small time delta to update velocity
   (d) Integrate velocity over time delta to update position
3. Go to step (2)

Within the n-body problem, each point mass is affected by the collective gravitational forces resulting from the presence of the other masses. For two given point masses, m_1 and m_2, with respective position vectors $\vec{p_1}$ and $\vec{p_2}$ given as 2D positional vectors, the Newtonian gravity force $\vec{F_{12}}$ exerted by m_2 over m_1 is given by:

$$\vec{F_{12}} = \frac{Gm_1m_2\vec{d_{12}}}{{d_{12}}^3},$$

where G is a gravitational constant, $\vec{d_{12}} = \vec{p_2} - \vec{p_1}$ is the offset vector of $\vec{p_2}$ with respect to $\vec{p_1}$ and $d_{12} = |\vec{d_{12}}|$ is the magnitude of vector d_{12}. The ${d_{12}}^3$ term is comprised of the scalar product of $\vec{d_{12}}$ with itself, that is, ${\vec{d_{12}}}^2 = \vec{d_{12}} \cdot \vec{d_{12}}$, to account for squared distance between the two masses, multiplied by d_{12} which normalises $\vec{d_{12}}$ in the numerator such that $\vec{F_{12}}$ points towards $\vec{p_2}$.

This equation can be generalised for the n-body case to yield an overall force $\vec{F_i}$ affecting mass m_i by summing the individual attraction forces $\vec{F_{ij}}$ due to all the other masses m_j, giving:

$$\begin{aligned}
\vec{F_i} &= \sum_{\substack{1 \le j \le N \\ j \neq i}} \vec{F_{ij}} \\
&= G \sum_{\substack{1 \le j \le N \\ j \neq i}} \frac{m_i m_j}{{|\vec{d_{ij}}|}^2} \cdot \frac{\vec{d_{ij}}}{|\vec{d_{ij}}|} \\
&= Gm_i \sum_{\substack{1 \le j \le N \\ j \neq i}} \frac{m_j \vec{d_{ij}}}{{d_{ij}}^3}.
\end{aligned}$$

Relating Newton's first law of motion $\vec{F_i} = m\vec{a} = m\ddot{\vec{p}}$ with the gravity equation for $\vec{F_i}$ yields a set of n simultaneous differential equations relating the mass positions $\vec{p_i}$ with their second order time derivatives $\ddot{\vec{p_i}}$.

The classical analytical solution to the n-body problem requires solving for n equations expressing the mass positions $\vec{p_i}(t)$ as a function of time t, where $\vec{p_i}(0)$ and $\dot{\vec{p_i}}(t)$ are, respectively, the initial position and velocity of each point mass m_i. The analytical solution to the n-body problem is generally intractable and hence numerical approaches are employed. These approaches introduce

[1] by Keith Bugeja and Colin Vella

errors due to the approximate nature of numeric integration, but are relatively simpler to implement in real-time simulations and scale well with the problem size.

Given a function f in variable t, it is possible to find a function $\triangle f(f, \dot{f}, t, \triangle t)$, such that for some small $\triangle t$, $f(t + \triangle t) \approx f(t) + \triangle f$. Euler integration defines $\triangle f = \dot{f}(t)\triangle t$, giving the approximation $f(t + \triangle t) \approx f(t) + \dot{f}(t)\triangle t$. This approximation leads to a method for numerically computing integrals. Assuming $\dot{f}(t)$ is known over the range $a \leq t \leq b$, its definite integral $\int_a^b \dot{f}(t)\mathrm{d}t$ over this range can be numerically approximated using this algorithm:

$f \leftarrow \dot{f}(a)$
$t \leftarrow a$
while $t < b$ **do**
  $f \leftarrow f + \dot{f}(t)\triangle t$
  $t \leftarrow t + \triangle t$
end

To numerically integrate the motion of a point mass m_i forward in time within the simulation loop, the acceleration $\vec{a_i}(t)$ is first derived by solving the force equation for the acceleration to yield

$$\vec{a_i} = \frac{1}{m_i}\vec{F_i}.$$

Given the current acceleration $\vec{a_i}$, a new velocity $\vec{v_i}'$ is computed from current velocity $\vec{v_i}$ using the integration step

$$\vec{v_i}' = \vec{v_i} + \vec{a_i}\triangle t.$$

Similarly, a new position $\vec{p_i}'$ is computed from current position $\vec{p_i}$ using

$$\vec{p_i}' = \vec{p_i} + \vec{v_i}\triangle t.$$

Numerical integration methods may introduce stability problems during simulation. For instance, singularities can occur in cases where $|\vec{d_{ij}}| = 0$ or very small, leading to arithmetic overflows during the force computations. Such problems can be mitigated by clamping $|\vec{d_{ij}}|$ to some minimum value:

$$d_{ij} = \max\left(|\vec{d_{ij}}|, \frac{m_i + m_j}{2}\right).$$