UNIVERSITY OF MALTA
FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

CPS3236 Assignment : `N-Body Simulator`

Keith Bugeja, Alessio Magro, Kevin Vella

Instructions:

1. This is an **individual** assignment and carries **100**% of the final CPS3236 grade.
2. The report and all related files (including code) must be uploaded to the VLE by the indicated deadline. Before uploading, archive all files into a compressed format such as ZIP. It is your responsibility to ensure that the uploaded file and all contents are valid.
3. Everything you submit must be your original work. All source code will be run through plagiarism detection software. Please read carefully the plagiarism guidelines on the ICT website.
4. Reports (and code) that are difficult to follow due to poor writing-style, organisation or presentation will be penalised.
5. Always test function return values for errors; report errors to the standard error stream.
6. Each individual may be asked to discuss their implementation with the examiners. The outcome may affect the final marking.

Introduction

The n-body problem is the problem of predicting the motions for n particles interacting gravitationally. The problem is motivated by the desire to understand the motions of celestial bodies. An n-body simulation uses numerical methods to approximate the motion of particles that interact with one another through some type of physical force. The goal of this assignment is to implement a scalable multiprocessor implementation of the n-body simulation. This assignment will be evaluated on a number of criteria, including correctness, structure, style and documentation. Therefore, your code should do what it purports to do, be organised into functions and modules, be well-indented, well-commented and descriptive (no cryptic variable and function names), and include a design document describing your solution.

Deliverables

Upload your source code (C files together with any accompanying headers), including any unit tests and additional utilities, through VLE by the specified deadline. Include makefiles with your submission which can compile your system. Make sure that your code is properly commented and easily readable. Be careful with naming conventions and visual formatting. Every system call output should be validated for errors and appropriate error messages should be reported. Include a report describing:

- the design of your system, including any structure, behaviour and interaction diagrams which might help;
- any implemented mechanisms which you think deserve special mention;
- your approach to testing the system, listing test cases and results;
- a list of bugs and issues of your implementation.

Do **not** include full code listings in your report; only snippets are allowed.

Part I

The n-body Problem

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 $\vec{F_{12}}$ exterted 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 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:

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

The Barnes-Hut Algorithm

The $O(n^2)$ complexity of the naïve algorithm described above limits its use in interactive simulations to a few hundred or thousand bodies. The algorithm proposed by Barnes and Hut dispenses with some accuracy to achieve lower complexity $O(n \log(n))$ by simplifying the force contribution of relatively far point mass clusters, by treating each cluster as a single point mass located at the associated centroid. The clusters are constructed by recursively partitioning the space occupied by the bodies up to some termination criteria such as maximum tree depth and minimum number of bodies within a partition. In two dimensions, the typical partitioning structure adopted is the Quad-tree. The following is an outline of the main simulation algorithm accommodating Barnes-Hut force calculation:

1. Allocate N bodies with random masses, positions and velocities
2. Build quad-tree for current mass configuration
3. For each body in the simulation
   (a) Calculate force by invoking force sub-algorithm with root node
   (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
4. Go to step (2)

Each node within the quad-tree represents a cluster at a given level which may exclusively consist of either individual masses (a leaf node) or clusters (an intermediary node or sub-tree). Each node maintains an aggregate mass and centroid, representing the cluster as a single point mass. The aggregate mass M of a leaf node is simply the sum of all masses in the node, given by:

$$M = \sum_{i \in S} m_i$$

where m_i is the i^{th} point mass and is some index subset of the set of all mass indices $\{1 \ldots N\}$. The corresponding aggregate centroid is the mass-weighted average of the individual mass positions, given by

$$\vec{c} = \frac{1}{M} = \sum_{i \in S} m_i \vec{p_i}$$

For an intermediary node, M and $\vec{c}$ may be computed in the same manner, using the aggregate mass values and centroids of the node's children in place of the individual mass values and positions.

The BH force algorithm is a recursive function that accepts a point mass and a tree node as parameters, returning the net force to be applied on the given mass by the masses within the given sub-tree. The following is an outline of the algorithm:

1. If given node is a leaf
   (a) Compute and return sum force exerted by node masses on given mass
2. Else if given mass is "far enough" from given node
   (a) Compute and return force exerted by node as an aggregate mass (cluster)
3. Else (if given mass is not far enough)
   (a) Initialise aggregate force value
   (b) For each child node of given node
      i. Compute force by recursively invoking this algorithm with child node
      ii. Add computed force to aggregate value
   (c) Return aggregate force value

The condition "far enough" in the above algorithm is usually expressed as a comparison of a clustering parameter θ against the ratio of cluster's radius with respect to the mass's distance from the cluster:

$$\frac{r}{d} > \theta$$

where $d = |\vec{p_i} - \vec{c}|$. The cluster radius r is an indicative measure as a function of the node's rectangular dimensions w and h or mass $\vec{p_j}$. A simple formula for r is $r = \frac{1}{2}(w + h)$. More sophisticated metrics, such as $r = \frac{1}{2}\sqrt{(w^2 + h^2)}$ and $r = \max_{j \in S} |\vec{p_j} - \vec{c}|$ are possible but do not warrant the increased computation cost. The parameter θ effectively controls the quality of the force approximation. As $\theta \to \infty$, the BH algorithm regresses to the naïve implementation, whereas $\theta \to 0$ leads to faster computation at the expense of lower accuracy. A suitable value is usually $\theta = 1$.

Part II

Objectives

Your aim is to design and implement multicore and multiprocessor implementations of the n-body simulation. In particular, you will develop shared memory, message passing and hybrid implementations and measure their performance and scalability under prescribed configurations. To help you start out, source code for a C++ reference implementation of the simulator will be provided (`nbody.cpp` and `vector2.h`); you are under no compulsion to use the provided reference implementation. In addition to the source code, you will also be provided with four input files containing the particle descriptions to be used in the simulations. The descriptions come in the form of mass and initial positions, and range in quantity from 64 to 16K bodies (`input_64.txt`, `input_1024.txt`, `input_4096.txt` and `input_16384.txt`).

Simulator 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 bodies 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. The implemented 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. You are also encouraged to attempt optimisation of computations via The Barnes-Hut Algorithm.

Simulator 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.

Task 1

As your first task, extend the sequential n-body simulator source code to support command line arguments that specify:

(a) an input file containing body positions and masses, to initialise the simulator. When no input file is provided, the simulator should generate the body positions and masses randomly. In the latter case, the number of bodies generated may also be specified as a command line argument

(b) the simulation runtime parameters such as the maximum number of iterations, the simulation time delta and the gravitational constant

(c) an option to enable or disable output. If output is enabled, each iteration of the simulation should write its results to a separate file, as specified in §Simulator Output.

A summary of the command line arguments is given in the table below:

Switch	Description	Default
-f	input file	
-o	enable output	true
-b	number of particles	1024
-g	gravitational constant	1
-i	number of iterations	1000
-d	simulation time step	0.005

[5 marks]

Task 2

Write a program to visualise output from the n-body simulator. The program takes as input a sequence of ordered output files in the format `nbody_`x`.txt`, each representing an iteration (where x is the iteration number), and displays a visual representation of the bodies in each file. The user should be able to specify:

(a) a folder/directory location that contains the sequence of output files to visualse

(b) the dimensions of the visualisation window in terms of its width and height

(c) the duration of each visualised frame

(d) the option to loop the animation

For this task and this task only, you are free to use a language/framework of your choice.

[10 marks]

Task 3

Extend the sequential n-body simulator in §Task 1 to distribute computation across multiple CPU cores. Specifically, your system will:

(a) use OpenMP (a shared-memory implementation) to scale across multiple CPU cores on a single computational node

(b) use MPI (a message-passing implementation) to scale out across multiple distributed-memory nodes

(c) use a hybrid OpenMP and MPI implementation to take advantage of both shared-memory (for same-node scaling) and message passing (to scale across different nodes)

[40 marks]

Task 4

For each provided test input file, carry out performance measurements for the following configurations:

(a) naïve shared memory (up to 12 cores),

(b) distributed memory (up to 4 nodes), and

(c) hybrid (shared and distributed memory, up to 12 cores $\times$ 4 nodes),

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. In your results, clearly show the scalability of your system for each of the three configurations.

[15 marks]

Task 5

Optimise the shared-memory n-body simulator (OpenMP version) from §Task 3 by implementing The Barnes-Hut Algorithm. Measure the performance of your implementation for each of the provided input files. Use up to 12 cores on a single node, executing up to one thousand iterations in each case. You should record the execution time against the number of processors, taking the average over multiple test runs for each result. In your results, clearly show the scalability of your system for each of the three configurations.

[15 marks]

Task 6

As your final task, draw a report that includes amongst others:

(a) the design of your system, including any structure, behaviour and interaction diagrams which might help;

(b) performance and scalability results using appropriate plots and tables;

(c) any implemented mechanisms which you think deserve special mention;

(d) your approach to testing the system, listing test cases and results;

(e) a list of bugs and issues of your implementation.

Do **not** include full code listings in your report; only snippets are allowed.

[15 marks]

---

end of paper