Com4521/Com6521: Parallel Computing with GPUs

Assignment: Part 1

Deadline: 23rd March 2020 15:00 (week 7)

Last Edited: 20/02/2020

Marking

Assignment 1 (of 2) is worth 30% of the total assignment mark. The total assignment is worth 80% of the total module mark.

Assignment 1 marks will be weighted as 50% for code functionality and performance and 50% for demonstrating understanding via a written report.

Document Changes

Any corrections or changes to this document will be noted here and an update will be sent out to the course google group mailing list.

Introduction

The aim of the assignment is to test your understanding and technical ability to implement efficient code on the CPU. You will be expected to benchmark and optimise the implementation of a simple NBody simulation. You will start by implementing a serial CPU version, you will then parallelise this version for multi-core processors using OpenMP. The emphasis of this assignment is on your ability to progressively improve your code to converge on an efficient implementation. In order to demonstrate this, you are expected to document (in a written report) any design consideration you have made in order to improve the performance (and ensure correctness). For example, you should use benchmarking to demonstrate how changes to your implementation have resulted in performance improvements. Handing in just a piece of code with excellent performance will not score highly in the assessment, unless you have also demonstrated an understanding in the written report, of how you have progressively refined your implementation to reach the final solution.

N-Body Simulation

An N-Body system represents a collection of (N) gravitational bodies each of which exert a gravitational force on each other. An N-Body system can be simulated using numerical integration by evaluating all pair wise interactions between bodies in the system. The force ($\overrightarrow{f_{ij}}$) on a body i exerted by a body j can be calculated through the following formula:

$$\overrightarrow{f_{ij}} = G\frac{m_i m_j(\overrightarrow{x_j} - \overrightarrow{x_i})}{\|\overrightarrow{x_j} - \overrightarrow{x_i}\|^3}$$

Where:

G is the gravitational constant
m_i is the mass of body i
m_j is the mass of body j
$\overrightarrow{x_i}$ is the position of body i
$\overrightarrow{x_j}$ is the position of body j

$\left\|\overrightarrow{x_i} - \overrightarrow{x_j}\right\|$ is the magnitude (i.e. length or normal) of the vector between position $\overrightarrow{x_i}$ of body i and position $\overrightarrow{x_j}$ of body j. *Note: The magnitude/length/normal is a scalar value and not a vector which can be calculated for a vector $\vec{v}$ (with an x and y component) as follows;*

$$\|\vec{v}\| = \sqrt{x^2 + y^2}$$

To avoid the force between two approaching bodies growing without bound, a softening factor (ε) can be added. The overall force ($\overrightarrow{F_i}$) exerted on body i can be calculated a summation ($\sum$) of all forces $\overrightarrow{f_{ij}}$ where j is 1 to N. E.g.

$$\overrightarrow{F_i} = Gm_i \sum_{1 \le j \le N} \frac{m_j\left(\overrightarrow{x_j} - \overrightarrow{x_i}\right)}{\left(\left\|\overrightarrow{x_j} - \overrightarrow{x_i}\right\|^2 + \varepsilon^2\right)^{3/2}}$$

The acceleration (a_i) of body i can then be calculated as

$$\overrightarrow{a_i} = \frac{\vec{F}_i}{m_i}$$

To simulate the movement using the equations of motion, simple Euler integration can be applied to give the velocity vector ($\vec{v}$) and position vector ($\vec{x}$) at time $t + 1$ with a fixed time-step of dt and an acceleration of $\vec{a}$.

$$\overrightarrow{v_{t+1}} = \overrightarrow{v_t} + dt * \vec{a}$$
$$\overrightarrow{x_{t+1}} = \overrightarrow{x_t} + dt * \overrightarrow{v_t}$$

The Assignment Requirements (Code)

You are expected to implement an N-Body simulation. You should start from the provided source code which is available from GitHub (https://github.com/mondus/com4521_assignment). You can either check it out using git or download the starting code as a zip file. The assignment has the following additional requirements.

Program Inputs:

Your program should accept the arguments described by the existing `print_help()` function. The input file format for N-Body initial states should be plain text CSV. The input file should also allow comment lines which are any line starting with '#' character. Each non comment line should define the initial conditions of a single body in the and should be terminated with a carriage return ('\n') in the following format. E.g.

```
float x, float y, float vx, float vy, float mass
```

An example line might be;

```
0.5f, 0.5f, 0.0f, 0.0f, 0.1f
```

Where each of the 5 variables represents the x position, y position, x velocity, y velocity and mass of the body respectively. Incomplete lines (where only some initial values are specified) should use default values for the unspecified variables. An example of this is given in the sample file with the starting code. The default values are;

`x` and `y` = random position within the range of `0` and `1`

`vx` and `vy` = `0`

`m` = `1/N`

Where `N` is the number of bodies in the system. When executing the program, an error should be thrown if the value of `N` supplied at runtime does not match the number of bodies in the input file.

Visualisation and Timing:

You N-Body simulation should be able to run in either console mode (for a fixed number of iterations, `I`) or in visualisation mode (if the value of `I` is omitted as a program argument). When running in console mode you should use an appropriate timing method to measure the runtime of your simulate function's execution for the `I` number of iterations. This should be output in the form;

```
"Execution time 123 seconds 456 milliseconds"
```

Writing the visualisation code is not required. A source module and header file have been provided which implements visualisation (which can be very helpful in checking that your simulation behaves as expected). You will need to use the functions defined in the header file to set pointers to your data and functions.

Activity Heat-map:

In addition to simulating the N-Body system you are required to calculate an activity map. The activity field map is a discretised 2-dimensional grid with `D*D` locations. The grid overlays the N-Body simulation space and each location can therefore be used to calculate a normalised count (number of body's within the grid location divided by N) of the population. The normalised count will always be a value in the range of `0` to `1` which will be translated by the visualizer to the red colour channel. For large values of `N`, very few locations within the map will register visible activity so the normalised count can be multiplied by D to give a more even spread of activity. Values which exceed `1` will be clamped by the visualiser.

You must calculate the values of the `D*D` locations and store these in a one dimensional array (or piece of allocated memory). Use the `setHistogramData` function from the `NBodyVisualiser.h` header file. You can visualise your calculated values by using the '`d`' key when in visualisation mode to toggle the display of the activity map. The display of the bodies can also be toggled using the '`b`' key.

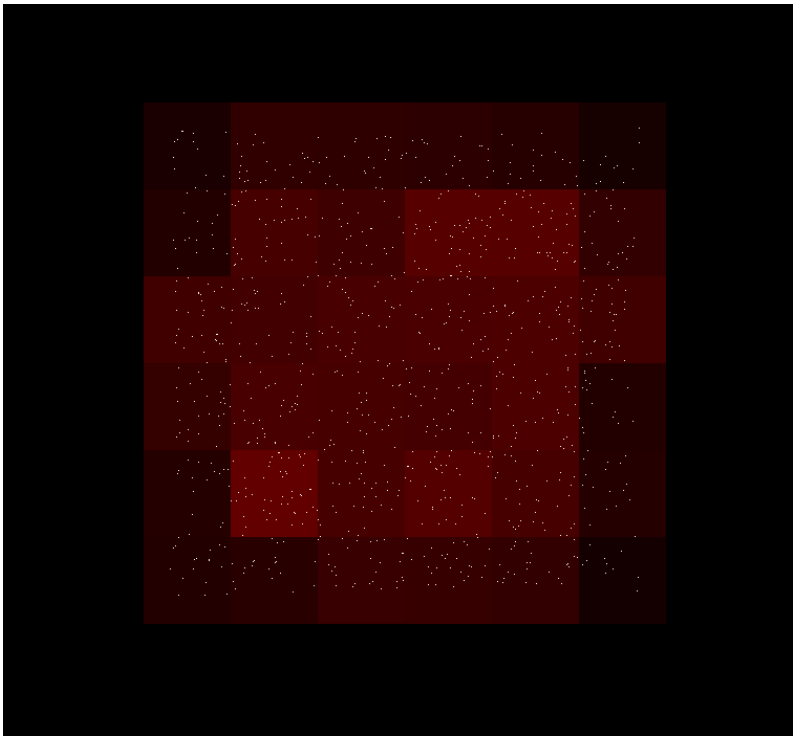

Figure 1 Example of the visualiser showing body positions and activity map.

Parallel Implementation:

You should first implement a serial C version of the code and make this as efficient as you can. Once implemented, your serial version should act as a baseline to measure potential performance speedup. You should implement a multi core parallel version of the same simulation which uses OpenMP. By passing different runtime program arguments (see `print_help` functions `M` argument in the starting code), it should be possible to run either the serial CPU version or the OpenMP version.

Documentation Requirements

You are expected to document the implementation of your code. More specifically you are expected to compare and contrast various implementation techniques to show how you have converged on a particular implementation. In particular you should benchmark your code in console mode to compare alternative techniques and give explanation as to why one implementation technique (or optimisation you have made) is better than the other. Some examples of benchmarks that are expected include;

- A comparison of OpenMP parallelisation strategies for your main N-Body simulation. Is parallelising over inner or outer loops more effective?
- A comparison of OpenMP techniques which could be used for calculating the activity map. How have you avoided race conditions?
- Any optimisations you have made to improve the main N-Body simulation loop. How has this effected performance?
- Any other interesting aspects of the implementation or optimisation techniques you have applied to either the serial or OpenMP version of your code.

Good benchmarking should be done over various values of `N` (or `D`). For each significant improvement to your code try to show the performance of your code before and after changes. You should highlight (with short code samples) any novel aspects or optimisation you have made.

Project Hand In

You should hand in your program code via MOLE with the documentation as a single pdf within a single zip file. You should also include the visual studio solution and any project files. Your code should build in the release mode configuration without errors (or warnings) on Diamond computer room 4 (lab) machines. You should submit whatever you have done if you have not completed the entire assignment. Your code should not rely on any third party libraries or tools other than libraries provided in the labs (i.e. OpenGL).

Marking

The marks for part 1 of assignment will be distributed as follows:

- *50%* of the assignment is for the coding aspect. Half of this percentage is for the quality of the programming and optimisation (including performance of your code) and the other half is for satisfying the requirements.
- *50%* of the assignment is for the production of a document describing the processes you have undertaken to implement and optimise your code. This should include benchmarking and iterative refinement of approaches as described in the documentation requirements.

In assessing your work, the following requirements will be considered for the code aspect.

1. Is the code functionally correct? I.e. Has the simulation been implemented correctly and does it produce the correct behaviour?
2. Has iterative improvement of the code yielded a sufficiently optimised final program for each of the CPU and OpenMP implementations?
3. Does the code make good use of memory bandwidth?
4. Does the OpenMP implementation avoid race conditions?
5. Are the OpenMP variables correctly scoped? Is their excessive/unnecessary use of shared variables?
6. Are there any compiler warnings or dangerous memory accesses (beyond the bounds of the memory allocated)? Does your program free any memory which is allocated?
7. Is your handling of input files correct and robust? Does the program deal with incorrect input file formatting elegantly (e.g. raising an error and exiting rather than crashing)?
8. Is your code structured clearly and well commented?

In assessing your documentation, the following will be considered and should act as a guideline for discussing incremental improvements to your code.

1. Description of the technique and how it is implemented.
2. Have a range of OpenMP approaches been considered for parallelising the simulation? Have appropriate investigations been made into scheduling? Are good explanations given to benchmarking results?
3. Have a range of OpenMP approaches been explored to implement the calculation of the activity map? Are good explanations given to why a particular approach has been chosen?
4. Does your document describe optimisations to your code and show the impact of these?

Tips for Developing Your Code and Documentation

You should start with a simple serial implementation and describe how you iteratively improve the performance in your documentation. If you are unable to complete some specific part (e.g. the activity map with OpenMP) then you should default back to using the simple CPU version for that part so that your code correctly builds and executes producing the correct result. Similarly, if you apply a technique that does not improve the performance, you should include this in your documentation and explain your belief/understanding as to why it did not work as expected.

You should comment your code to make it clear what you have done. You should test your code to make sure that it works for **all** values of `N` and `D`. For values and input files which are incorrect (for example a grid width of negative 10 or an input file with the incorrect number of initial values) your code should exit elegantly with an error message. Your code should never read or write beyond allocated.

Assignment Help

Help for your assignment will be available in general lab classes. For specific questions outside of the labs you should use the course google discussion group.