**COM4521: Assignment 1**
**Vladyslav Bondarenko**

| | | |
|---|---|---|
| **FINAL Mark (100%)** | 84 | 100 |
| Lateness penalty (days) | 0 | |
| Final mark before lateness penalty | 84 | 100 |

## Part 1: Code (50%)

Is it functionally correct?; Has iterative improvement produced a sufficiently optimised final program?; Does the code make good use of OpenMP?; Is memory handled appropriately and efficienty?; Are there compiler warnings?; Is the structure clear and easy to parse?

A low mark suggests that you need to think more carefully about what is required to write and optimise C and OpenMP code; A mark in the middle of the range suggests that you have some understanding of how to use C and OpenMP. A high mark suggests you are well on your way to understanding how to use C and OpenMP to produce efficient optimised code. 41 50

**Functionally Correct** 8 10

• Incorrect force calculation CPU. Your force calculation is incorrect on the CPU. Check the mathematics to ensure you have implemented the requirements in the handout.Force calculation is clearly wrong visually; Incorrect magnitude calculation;
• Correct activity map CPU.
• Incorrect force calculation OpenMP. Your force calculation is incorrect in the OpenMP version . This may be caused by an incorrect CPU calculation. You should check the mathematics to ensure you have implemented the requirements in the handout.
• Correct activity map OpenMP.
• Simulation runs correctly with random data.
• Correct output format.
• Program takes the expected arguments.
• Code compiles.
• Program does not crash.
• Correct file reading.

**Iterative Improvement** 6 10

• Clear iterative improvement.
• Parallelised outer loop.
• Slow performance.

**Memory Bandwith** 6 10

• No coalesced memory access. The memory access could be improved by using a different data structure. Use coalesced memory access by using a structure of arrays (SoA) rather than array of structure (AoS).
• Unnecessary data movement. The code can be optmised by reducing the movement of data. Consider using double buffering and pointer swaps to reduce memory copies and memory movement.
• Correct memory allocation.

**OpenMP Race Conditions** 10 10

• No race condition in force calculation.
• No race condition in activity map.

**OpenMP Variable Scoping** 10 10

• Correct use of explicit scoping.
• No unnecessary shared variables.

**Memory Access & Warnings** 10 10

• Code has no memory leaks.
• The allocated memory is freed appropriately.
• Compiler does not throw warnings.
• No use of uninitialised variables.

**Robust Input Handling** 8 10

• Code deals with badly formatted files.
• Code deals with different number of bodies.
• Code reads comment lines.
• Code deals with empty values.
• Program does not inform input errors or crashes. Program does not always inform/warn when the input file is badly formatted.

**Well Written Code** 8 10

• Comments could be improved. Comments could be added to not obvious code.
• Good variable naming.
• Correct use of data types.
• Code should make greater use of methods. Use more methods to break up long methods into smaller pieces.
• Good code structure.

## Part 2: Documentation (50%)

Is there a description of the implemntation? Is there a justifcation of the final implementation, including any steps taken to reach that conclusion? Description of the general testing process? Has performance throughout development been analysed – is there evidence of this?
A low mark suggests that you need to think more carefully about what is requiredto document development and optimisation. A mark in the middle of the range suggests that you have some understanding of the process and some consideration of attention to detail. A high mark suggests you are well on your way to understanding how to document the optimisation of code you develop. 43 50

**Description of Implementation** 9 10

Follows iterative steps of improvements; Good use of tables with timings, graphs and code snippets; Some justification as to the optimizations chosen;

**Description of OpenMP Implementation (Force calculation)** 9 10

Detailed examination of inner/outer loop parallelising with measurements to back up; Some scheduling examination was done but could be improved; Little comparison of atomic or critical sections;

**Description of OpenMP Implementation (Activity map)** 7 10

Detailed examination of inner/outer loop parallelising with measurements to back up; Could have included scheduling timings; Not much consideration to critical sections;

**Analysis of Optimisation** 9 10

Good structure of iterative improvements; Optimisations done and results were clear; Decisions were clearly influenced by well found data; Explanation of why timings were better or worse was acceptable; Could have done more to find hotspots in code and show process;

## Overall

What is good about this work?

Correct activity map CPU. Correct memory allocation. No race condition in force calculation. No race condition in activity map. Correct use of explicit scoping. Follows iterative steps of improvements; Optimisations done and results were clear; Efficient use of the profiler to find places to optimise;

What needs to be done to make it better?

Force calculation is clearly wrong visually; Incorrect magnitude calculation; No coalesced memory access. The memory access could be improved by using a different data structure. Slow performance.