UCT CS Honours 2019

High Performance Computing Course Assignment: **Parallel Sorting**

(version 1)

Lecturer: Michelle Kuttel

Due date: Thursday 23rd **May** 2019, **10 pm**

1 Aim

This project will give you experience in parallel programming using both OpenMP and MPI, as well as validation and profiling of parallel codes.

You will: write serial and parallel software to analyze simulation data; evaluate the parallel programs experimentally and write a clear, detailed report to present and discuss your methods and results.

2 Problem Description

For this assignment, you will implement two parallel algorithms for sorting arrays of integers and compare their performance: **quicksort** and, a variant, **regular sampling parallel sort**. Quicksort is naturally concurrent, as a consequence of the divide-and-conquer algorithm, but the performance is significantly impacted by the choice of pivot. Regular sampling parallel sort seeks to improve this, using a regular sampling of the data to ensure good pivot selection[1].

You will write a serial quicksort implementation (in C or C++), as well as parallel versions of **quicksort** and **regular sampling sort** in **both** OpenMP and MPI. You may adapt your solutions from other's code, but then you **must** give attribution to the original authors, in both your code and the report.

Note that we aim for parallel implementations that are both **correct** and **faster** than the serial versions. Therefore, you need to demonstrate both **correctness** and **speedup** for your assignment. You will evaluate the performance of your parallel sorting algorithms **experimentally**, using high-precision timing to evaluate the run times of your parallel methods, across a range of input sizes and numbers of cores and (for MPI) **numbers of** nodes and calculating the **speedup** of the algortihms with respect to each other and the serial quicksort. You will be testing on UCT's hpc cluster.

Finally, you will document the results in a report that clearly demonstrates and explains your findings.

3 Code requirements

You must submit a code archive including running/installation instructions in a README file. You will need to generate suitable test files of different sizes for testing.

4 Report

You must also hand in a report containing the following sections.

- Introduction
- Serial quicksort
  - Description
  - Implementation
  - validation
- Parallel algorithms: OpenMP
  - Description
  - Implementation
  - Validation
- Parallel algorithms: MPI
  - Description
  - Implementation
  - Validation
- Benchmarking
  - Methods – input files generation and testing methods detailed
  - System Architecture
  - Results and Discussion
- Conclusions

- The *Introduction* must explain the problem and highlighting its suitability for parallelization.
- For the three algorithms, the *Description* sections should include a theoretical estimate of the running times. The *Implementation* sections must explain your approach to solving the problem and any important implementation details. The *Validation* sections should explain how you ensured that your implementation was correct.
- Under *Benchmarking*, the *Methods* section must explain how you timed your algorithms (with different input, cores etc.) and how you measured speedup. Make sure that you do this properly, for different input sizes and consider the different components of the application. *System Architecture* should detail the machine architectures you tested the code on. The *Results and Discussion* section must document the **speedup** (not timing) of your parallel implementations. This section must have graphs and, if necessary, tables (do not include voluminous tables in the text – add them as appendices and graph the results). Graphs should be clear and labelled (title and axes). You should compare and contrast OpenMP and MPI and discuss any problems or

difficulties encountered.

- The *Conclusions* section must summarize the main results and findings.

Please do NOT ask the lecturer for the recommended numbers of pages for this report. Say what you need to say: no more, no less.

5 Assignment submission requirements and assessment rules

- You will need to create, **regularly update**, and submit a GIT archive of your code.
- Your submission archive must consist a technical report and your solution code archive (including a **Makefile** for compilation/running).
- Submit your report as a separate (from the source code) file **IN PDF FORMAT** named **HPC_*STDNUM001*.pdf,** replacing STDNUM001 with your student number. **Incorrect formats will incur a 5% penalty.**
- Upload the files and **then check that they are uploaded.** It is your responsibility to check that the uploaded file is correct, as mistakes cannot be corrected after the due date.
- The usual late penalties of 10% a day (or part thereof) apply to this assignment.
- The deadline for marking **queries** on your assignment is **one week after the return of your mark.** After this time, you may not query your mark.
- Note well: submitted code that does not run or does not pass standard test cases will result in a mark of zero. **Any plagiarism, academic dishonesty, or falsifying of results reported will get a mark of 0 and be submitted to the university court**.

6 References

[1] H. Shi and J. Schaeffer, *Parallel sorting by regular sampling,* Journal of Parallel and Distributed Computing, Volume 14, Issue 4, 1992, 361-372. https://doi.org/10.1016/0743-7315(92)90075-X.