

F21DP2 Distributed and Parallel Technology

Assessed Coursework

Evaluating Parallel Programming Models

Purpose

In this coursework you will develop parallel programming, experimentation, and technology evaluation skills. The parallel architectures are a multicore and a Beowulf cluster. You will tune sequential versions of a program, and then develop and measure parallel versions using 3 technologies: C with MPI, C with OpenMP, and Glasgow parallel Haskell (GpH). Finally you will compare the performance and programming models of the three technologies.

Organisation

The assessed coursework work is to be carried out in pairs, and you should choose your own partner. Together you must develop and tune the parallel performance of the three programs and prepare a comparative report. It is relatively easy to produce a simple parallelisation of both programs, however *additional marks are available for thoughtful sequential and parallel performance tuning.*

- The first member of the pair develops a parallel C with MPI version of the totient program available from <http://www.macs.hw.ac.uk/~hwloidl/Courses/F21DP/srcs/TotientRange.c>
- The second member of the pair develops *both*:
 - A C with OpenMP version of `TotientRange.c`, as above.
 - A GpH version of the `TotientRange.hs` program available from <http://www.macs.hw.ac.uk/~trinder/ParDistr/Examples/TotientRange.hs>

Deliverables

The deliverables are a **blog** and a **report** (including sources) with the structure below. The deliverables are due by 3:30pm on **Friday of week 8**. Reports not following this structure, or not containing all of the specified results, will be penalised.

Blog: The development of the code should be documented by a blog, as set up on Vision, documenting the main steps in getting to the final parallel code. Use this opportunity to discuss challenges you encountered and possibly wrong directions of parallelisations you tried and that didn't work. (*5 marks*).

Report with the following structure:

- **Section 0 Sequential Performance Measurements** (5 marks).

Profile and analyse the sequential C and Haskell programs. Tools that can help you, and have been discussed in the lectures, are `gprof` and `cachegrind` for sequential C, GHC's time and heap profiling for sequential Haskell (`ghc -prof`), and Threadscope for parallel Haskell. You are welcome to use other tools, if you find them useful. In each case you should motivate your choice of tool and reflect how useful it was for tuning performance. For plotting parallel performance results `gnuplot` is recommended.

This section of the report should discuss the sequential performance of, and possible improvements to, these programs. Of particular interest are hotspots in the program and good cache usage - max 1 A4 page, plus any graphs.

- **Section 1 Comparative Parallel Performance Measurements** (12 marks).

You should measure and record the following results in numbered sections of your report. The measurements are based on three inputs:

- DS1: calculating the sum of totients between 1 and 15000.
- DS2: calculating the sum of totients between 1 and 30000.
- DS3: calculating the sum of totients between 1 and 100000.

Runtime measurements should be the middle (median) value of three executions. N.B. For comparison purposes the performance of the GpH, C+OpenMP and C+MPI systems must be reported on the *same* graph. You may also plot other graphs to show interesting features, or use larger numbers of cores.

- **Section 1.1 Runtimes.** Measure the runtime of the sequential Haskell and C programs for DS1 and DS2. Plot **three** runtime graphs
 - * DS1: execution times for the sequential C and Haskell programs, and the parallel C+MPI, C+OpenMP and GpH programs on 1,2,3, .. 8 cores of an `lxpara` machine.
 - * DS2: execution times for the sequential C and Haskell programs, and the parallel C+MPI, C+OpenMP and GpH programs on 1,2,3, .. 8 cores of an `lxpara` machine.
 - * DS3: execution times for the C+MPI program on 8, 16, 32, 64, 128, 192, 256 cores of the beowulf cluster.
- **Section 1.2 Speedups.** Plot **three** absolute speedup graphs corresponding to the runtime results for DS1, DS2 and DS3 showing the ideal speedup and the speedups for the C+MPI and GpH programs on 1,2,3, .. 8 cores. Recall that absolute speedup is calculated using the runtime of the sequential program on a single core.
- **Section 1.3** A table summarising the sequential performance and the best parallel runtimes of your C+MPI, C+OpenMP and GpH programs.
- **Section 1.4** A discussion of the comparative performance of the GpH and C+MPI programs - max 1 A4 page.

- **Section 2 Programming Model Comparison** (8 marks).

An evaluation of the three parallel programming models for the totient application. You should indicate any challenges you encountered in constructing your programs and the situations where each technology may usefully be applied. Refer to blog entries when discussing the development, but make sure that the discussion in the report is self-contained.

The comparison should be based on the TotientRange application, and focus on the technology in general, and be supported by technical arguments - max 1 A4 page

- **Appendix A C+MPI TotientRange Program** (*10 marks*).

A listing of your C+MPI TotientRange program, clearly labelled with the single author's name. Also include a paragraph, and possibly diagram(s), identifying the parallel paradigm used, and performance tuning approaches used.

- **Appendix B C+OpenMP TotientRange Program** (*5 marks*).

A listing of your C+OpenMP TotientRange program, clearly labelled with the single author's name. Also include a paragraph, and possibly diagram(s), identifying the parallel paradigm used, and performance tuning approaches used.

- **Appendix C GpH TotientRange Program** (*5 marks*).

A listing of your GpH TotientRange program, clearly labelled with the single author's name. Also include a paragraph, and possibly diagram(s), identifying the parallel paradigm used and performance tuning approaches used.

Notes

1. Complete the GpH and OpenMP Lab exercises before starting the coursework.
2. Graphs and tables must have appropriate captions, and the axes must have appropriate labels.
3. To ensure a fair comparison all measurements should be made on a very lightly loaded machine. Check the load on nodes using something like the unix `top` command.