F20DP Distributed and Parallel Technology

Assessed Coursework 1

Evaluating Low-level Parallel Programming models

Purpose

In this coursework you will develop parallel programming, measurement, and technology evaluation skills. Additionally, there needs to be some reflections on the advantages and dis-advantages of the chosen parallel programming technologies. The parallel architectures are a multicore and a Beowulf cluster. You will develop and measure parallel versions of a program using one of the following low-level parallel programming technologies: C+MPI or OpenCL. Finally you will compare the performance and programming models of these technologies.

Sequential Program

The program that should be parallelised is the computation of the sum of Euler totient computations over a range of integer values. The Euler totient function computes, for a given integer \( n \), the number of integers that are relatively prime to \( n \), i.e. \( \Phi(n) \equiv \{ m \mid m \in \{1, \ldots, n\} \land m \perp n \} \). Two numbers \( m \) and \( n \) are relatively prime, if the only integer number that divides both is 1. To test this, it is sufficient to establish that their greatest common divisor is 1, i.e. \( m \perp n \equiv \text{gcd} \ mn = 1 \). Thus, the task for this program is: for a given integer \( n \), compute \( \sum_{i=1}^{n} \Phi(n) \).

The following C code is a direct implementation of the above specification, as starting point for the parallel algorithm: [http://www.macs.hw.ac.uk/~hwloidl/Courses/F21DP/srcs/TotientRange.c](http://www.macs.hw.ac.uk/~hwloidl/Courses/F21DP/srcs/TotientRange.c)

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 both 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.

Tools that can help you, and have been discussed in the lectures, are gprof and cachegrind for sequential C. For plotting parallel performance results gnuplot is recommended.

Each member of the team chooses one of the following technologies to implement this function:

- C with MPI for explicit message passing;
- C with OpenCL for off-loading computations to a GPGPU;

Deadline: 3:30PM on Friday 6th of March, 2015
Deliverable

The deliverable is a blog and a report (including sources) with the structure below. The deliverable is due on Friday 6th of March, 2015. Reports not following this structure, or not containing all of the specified results, will be penalised. Submission should be through Vision (sub-menu Assessment).

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. (4 marks).

Structure of the report:

Section 1: Introduction (4 marks).

This should give a short summary of the task to implement and parallelise, describe the environment it is performed in, the parallel technologies used, and the learning objective of this coursework.

Section 2: Sequential Performance Measurements (4 marks).

Run the sequential version of the programs and analyse the performance, possibly using some of the tools mentioned above. Discuss the sequential performance of and possible improvements to these programs. Of interest are in particular hotspots in the program and good cache usage. max 1 A4 page

Section 3: 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 these 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 different 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 3a: Runtimes. Measure the runtime of the sequential C program for DS1 and DS2. Plot three runtime graphs

- DS1: execution times for the sequential C program, and both parallel programs on 1,2,3, . . . , 8 cores.
- DS2: execution times for the sequential C program, and both parallel programs on 1,2,3, . . . , 8 cores.
- DS3: execution times for the sequential C program, and both parallel programs using 8, 16 . . . , up to the maximal number of cores available (256 cores for the C+MPI implementation).

Deadline: 3:30PM on Friday 6th of March, 2015
Section 3b: Speedups. Plot three relative speedup graphs corresponding to the runtime results for DS1, DS2 and DS3 showing the ideal speedup and the speedups for both parallel programs programs on 1,2,3, .. 8 cores. Recall that relative speedup is calculated using the runtime of the parallel program on a single core.

Section 3c A table summarising the sequential performance and the best parallel runtimes of both parallel programs.

Section 3d A discussion of the comparative performance of both parallel implementations max 1 A4 page.

Section 4: Programming Model Comparison (6 marks).

An evaluation of the parallel programming models, specifically 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 max 1 A4 page.

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.

Section 5: Reflection on Programming Models (0 marks).

Not applicable.

Appendix A and B (10 marks each).

For each parallel implementation, an appendix should give the listing of your parallel 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 C with MPI and OpenCL Lab exercises before starting the coursework.

2. Graphs and tables must have appropriate captions, and the axes must have appropriate labels.

3. There is no batch-job system to give you exclusive access to all bwlf nodes. Thus, 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.

Deadline: 3:30PM on Friday 6th of March, 2015