|epcc|

MPP Coursework Assessment

1 Introduction

The purpose of this assessment is for you to demonstrate that you can write a message-passing parallel program for a simple two-dimensional grid-based calculation that employs a two-dimensional domain decomposition and uses non-blocking communications. This will involve extending and enhancing the MPI program you developed for the Case Study exercise.

You will be provided with simple solutions to the Case Study. However, I recommend that you still attempt to write your own solution so that you fully understand this more straightforward problem, which uses a one-dimensional parallel decomposition, before attempting the two-dimensional decomposition.

The key aims of the entire assessment are to:

- write a working MPI program (in C, C++ or Fortran) for the percolation problem that can use a two-dimensional domain decomposition and uses non-blocking communication for halo-swapping;
- use periodic boundary conditions in the vertical direction (second index "j") – we will provide an example of how to implement this in the serial version;
- calculate the average value of the map array in the iterative loop, and print at appropriate intervals;
- be able to terminate the calculation when all clusters have been identified, i.e. when there are no changes in the map array between steps;
- demonstrate that the parallel program works correctly and that the performance improves as you increase the number of processes.

2 Structure

The assessment comprises two parts:

Part 1: a short report which contains a description of how you plan to design, implement and test your parallel program.

Part 2: your final program plus a short report describing any differences from your initial plan and the results of your tests.

Your tests should cover both correctness and performance.

Although you do not submit any code for part 1, you should run your correctness and performance tests on the Case Study exercise (i.e. the one-dimensional decomposition with non-periodic boundary conditions). You can either use your own solution or the simple example parallel solution from the course.

The reason for having two parts is to ensure that, after part 1, you have a clear plan for your work when you start to develop your own parallel program for part 2. For part 1, you will also have developed and implemented the infrastructure for all the required tests. This means that the testing processes for part 2 will be straightforward as you can just re-run all the same tests. The performance results from part 1 will also give baseline values to compare to your own program.

3 Part 1

The overall weighting for Part 1 is 30% of the total course marks.

3.1 Report

Submit a short report, **a maximum of 6 pages** (excluding any cover page, table of contents or appendices), including a description of the proposed design and implementation of your MPI program followed by brief results from tests applied to the Case Study exercise.

Your description should cover how you plan **to improve the Case Study exercise** to meet the **requirements set out in Section 1**.

Your report should cover:

- Which parts of the existing program need to be changed and why.
- What additional MPI functions you plan to use and why.
- How you plan to structure your program including use of any of your own functions / subroutines and how it will be split across files.
- What correctness and performance tests you plan for the full program and brief results of some of these tests applied to the Case Study exercise.

You should present results from a number of tests designed to show that the parallel program works correctly and that its performance improves as the number of processes is increased. You should quantify the parallel performance using appropriate metrics; you may also wish to consider how performance is affected by the problem size. If you use the supplied solution to the Case Study, you may still have to add some additional timing calls to help in measuring the performance.

3.2 Marking Scheme

Marks will be allocated for the report as follows: proposed changes, use of MPI, program structure, testing, presentation; 20% for each category (the presentation component includes quality of English, layout, figures and diagrams).

The deadline for this assessment is **16:00 on Friday 6th November 2020**.

4 Part 2

The overall weighting for Part 2 is 50% of the total course marks.

4.1 Report

Submit a short report, **a maximum of 6 pages** (excluding any cover page, table of contents or appendices), including a description of any changes to the design and implementation of your MPI program from your original proposals in Part 1, followed by results such as the tests you have done to investigate correctness and performance. You should study performance for a range of process counts and problem sizes and analyse and comment on the results, including the parallel scaling. You should also give overall conclusions for your work as a whole.

4.2 Source Code

Ensure that your program is clearly written and easy to understand with appropriate use of comments, multiple source files, functions or subroutines, meaningful variable names etc. Preference will be given to simple, elegant and robust programs rather than code that is unnecessarily complicated or difficult to understand. Use MPI features described in the lectures (derived datatypes, virtual topologies, etc.) where appropriate.

5 Marking Scheme

Marks will be allocated as follows, **independently** for the report and source code:

- Report (weighting of **30%**)

  Presentation, code description, testing, performance analysis, conclusions; 20% for each category.

- Source code (weighting of **70%**)

  Presentation & structure, use of MPI, correctness, functionality, flexibility; 20% for each category.

The deadline for this assessment is **16:00 on Wednesday 2nd December 2020**.

6 Submission Guidelines

6.1 Reports

Preference will be given to reports that present carefully chosen, clean and well explained results as opposed to large amounts of raw data. As well as written text you should use graphs, figures and tables as appropriate. If you do want to include the raw data as well, this can be included in an appendix but should only be a few pages long. The report **must be in PDF format** and must have a file name of the form "MPP2021-B1234567.pdf" where you replace B1234567 with your own **exam number** and **not your UUN or matric number**. The report **must be written anonymously** but **must contain your exam number** on the title page. You **must not** include your name or UUN anywhere on the report. When uploading to Turnitin, your "Submission title" **must match the file name**, e.g. "MPP2021-B1234567".

6.2 Code

You should also submit your complete MPI program including a README file briefly describing it – see the separate document `MPP-Code-Submission.pdf` for full details.

6.3 Late penalties

Standard penalties apply to late submissions: 5% reduction for each elapsed calendar day (or part day) after the deadline; submissions more than seven days late will be awarded 0%. See the programme handbook for full details.

7 Lab Sessions

The lab session from 14:10 - 15:00 on Mondays will continue for the rest of the semester. These labs are held so you can ask questions and discuss problems relating to your MPP coursework (or any part of the MPP course as a whole). This includes the content of your report as well any programming issues.

8 Notes

Here are a few points that you should take into account.

- The supplied solution to the Case Study is very basic. As well as the mandatory aims listed in Section 1, there are other parts of the program you may wish to improve. For example, the problem size and number of processes are fixed at compile time, and it does not run correctly when L is not an exact multiple of the number of processes (this is not even checked).
- It is essential that your report contains tests that demonstrate your parallel program works correctly.
- Think carefully about what sections of your program you time to measure the parallel performance. There are a number of choices (e.g. time the entire program from start to finish), but the average time per step is usually a good measure as we are not particularly interested in the performance of the (serial) initialisation and IO sections. You should run on the backend compute nodes of Cirrus and check that the performance is stable and reproducible.
- Be careful to do sensible performance tests and not to burn huge amounts of CPU time unnecessarily: consider how long you need to run to give a reasonable assessment of performance. When benchmarking large HPC programs (as opposed to doing actual computations), it is normally not necessary to run to completion; perhaps only a limited number of steps is required. Conversely, for small problem sizes, you may need to run for additional steps to obtain a reasonable run time.
- You will **not** easily be able to use `MPI_Scatter` and `MPI_Gather` for distributing and collecting the map array in a 2D decomposition. Implement these operations in a simple manner so you can start developing a working MPI program – it does not matter if these phases are inefficient.
- This is not an exercise in serial performance optimisation. However, to get realistic parallel performance numbers it is necessary to use a reasonable level of serial compiler optimisation. For example, you should compile your programs with the `-O3` option on Cirrus.
- You should concentrate on writing an elegant and efficient MPI program, performing a solid investigation of its performance and writing a good report. However, if time is available, you are welcome to investigate enhancements to your parallel program (some suggestions are given in the Case Study).
- If you cannot produce a working solution you should still submit any code you have written. The report should describe its design and implementation, a description of how far you got and what the problems were. Correctness, performance tests and scaling analysis should be done with your Case Study solution.