|epcc|

MPP Coursework Assignment

1 Introduction

The purpose of this assignment is for you to demonstrate that you can write a message-passing parallel code 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. The basic aims of the coursework are to:

- write a working MPI code (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 horizontal direction (first index "i") – we have provided an example of how to implement this in the serial version;
- calculate the average of the map array during 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 code works correctly and that the performance improves as you increase the number of processes.

2 Report

Submit a short report, **a maximum of 10 pages** (excluding any cover page, table of contents or appendices), including a description of the design and implementation of your MPI program followed by results such as the tests you have done to investigate correctness and performance. You should present results from a number of tests designed to show that the parallel code 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.

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 "MPP1920-B1234567.pdf" where you replace B1234567 with your own exam 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. "MPP1920-B1234567".

3 Source 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. 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.

4 Notes

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

- 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 codes (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 codes with the `-O3` option on Cirrus.
- You should concentrate on writing an elegant and efficient MPI code, 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 code (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 and performance tests should be done with your Case Study solution.

5 Lab Sessions

The lab session at 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.

6 Marking Scheme

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

- Report (weighting of **60%**)
  - Content 70%: code description, testing, performance data, performance quantification, analysis of results, completeness and originality, conclusions; 10% for each category.
  - Presentation 30%: written communication, layout, quality of figures; 10% for each category.
- Source code (weighting of **40%**)
  - Presentation and structure, use of MPI, correctness, functionality; 10% for each category.

The deadline for this assignment is **16:00 on Wednesday 27th November 2019**. 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.