|epcc|

Coursework Assignment
MPP: Message-Passing Programming

David Henty

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 lattice-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 or Fortran) for the image processing problem that can use a two-dimensional domain decomposition and uses non-blocking communications for halo-swapping;
- during the iterative loop, calculate the average value of the pixels in the reconstructed image and print this out at appropriate intervals;
- terminate the calculation when the image is sufficiently accurate, based on the value of the Δ parameter (see the Case Study notes for a definition);
- test that the parallel code works correctly;
- demonstrate that the performance of the code improves as you increase the number of processors.

You are required to submit your MPI program and a short report describing it. Marks will be awarded independently for the quality of your code and the quality of your report (see Section 5).

2 Source Code

You should submit a complete MPI program including sufficient information so that the marker could, if desired, easily compile and run the code to reproduce the results presented in your report. For example, you might want to include instructions on how to build the code and how to submit parallel jobs. Ensure your program is clear and easy to understand with appropriate use of comments, multiple source files, functions and 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.

3 Report

Submit a short report, no longer than about 10 pages, including a brief 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. The report must be in PDF format. 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 processors 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.

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.
- You should think carefully about what sections of your program you need to time in order to assess the parallel performance. There are a number of choices (e.g. you could time the entire program from start to finish), but the average time per iteration is usually a good quantity to measure. You should do some tests to check that your program's performance is stable and reproducible.
- When doing performance studies you should be careful to do sensible runs and not burn huge amounts of CPU time unnecessarily. You should consider how long the code needs to run in order to give a reasonable assessment of its performance and/or correctness. When benchmarking HPC codes (as opposed to doing actual computations), it is normally not necessary to run to completion; for example, perhaps only a limited number of iterations is required.
- You will not easily be able to use `MPI_Scatter` and `MPI_Gather` for IO when using a 2D process decomposition. You should implement a very simple form of IO so that you can quickly and easily start to develop a working MPI program – it does not matter if the IO is not very efficient.
- 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. If you use the default compilers on ARCHER then the compiler optimsation level will already be quite high; just make sure that you **do not** compile with the debugging flag `-g`. On other systems you should specify a high level of optimisation, e.g. `-O3`.
- 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 MPI program you should still submit whatever code you have written. The report should describe the design and implementation, plus a description of how far you got and what the problems were. Even if you cannot present any results, you should describe what your testing strategy would have been.

5 Marking Scheme

Marks will be allocated as follows:

- Report
  - Content (code description, testing, results, comments and conclusions) 40%
  - Presentation (quality of graphs, figures and text) 20%
  - **Total 60%**
- Source code
  - Submitted MPI code (quality, readability, correctness and performance) **40**%