Digital Systems

EEE4084F

Practical 4 – MPI

[30 Marks]

Introduction

The focus of this practical is message-passing in a distributed memory programming model. The processing algorithm can be copied from your Practical 2 code, as the same median filter will be used. (Mark allocations are shown in square brackets []).

You can find a tutorial at https://computing.llnl.gov/tutorials/mpi/, but this is much more detailed that what you need for this practical.

The Programming Model

One thing to keep in mind is that each process runs in a completely separate memory-space. There are no shared memory structures, so any global memory you define is local to each instance. The only means of sharing data is through sending and receiving messages.

Typically MPI is only useful on distributed systems (systems where each node has its own memory, and the nodes are connected through some form of network – typically Ethernet). Setting up such a system is not trivial, and therefore not included in this practical. For this practical, all the processes will be running on the same CPU, with memory being separated by means of memory virtualisation (through use of the memory management unit).

MPI Commands

MPI programs are compiled with **mpic**, not gcc, and run with **mpirun**, not by calling the program binary directly. The makefile associated with this practical runs the program with 5 instances (one master and 4 slaves). Feel free to change this if you so wish.

Study the source code provided. It has been adapted from https://en.wikipedia.org/wiki/Message_Passing_Interface#Example_program and uses simple blocking communication. You can implement this practical by means of blocking communication, so you don't need any more commands than those in the example code.

If you feel like being fancy, have a look at the MPI Iprobe command.

You can also have a look at http://stackoverflow.com/questions/12637550/using-mpi-byte-to-receive-any-data-type to see how to tag your messages. Tags are typically used to determine the message type before actually receiving it, but it could be used for many other things as well.

There are many MPI datatypes, but for the purpose of this practical you can get away with using bytes (MPI BYTE) exclusively.

Part 1: Problem Partitioning

Choose a memory partitioning scheme. In your report, explain your partitioning scheme and why you chose it. Do not send the entire dataset to each process, as this is a waste of data. [5 marks for describing the partitioning]

You also need to have some means to indicate to the slave what size the data is. You can implement this any way you wish, but I would suggest that you send one message, with a known size and fixed format, that indicates data size and any other parameters you wish to send to the slave. The next message can then be the actual data to work on. [5 marks for getting data to slaves]

When the slave is finished, it must send the result back to the master. The master must then assemble the results and save the resulting filtered image to disk. [5 marks sending back the data and reassembling]

Provide snippets of your code in the report and attached your code in a zip repository (it is important for your code snippets to be suitable commented, the use of comments in your main code repository is not as important for marking but is good practice). [5 marks code]

Part 2: Experiments

Feel free to experiment with different number of processes, although this is not required. It is sufficient to simply get it working. [5 marks for experiments]

Your report should comment on the time performance though, so make sure you take time measurements. [5 marks graphs and display of results]

Part 3: Doxygen

If you have Doxygen installed, you should be able to do a *make doxy* to generate the doxygen files (or run doxygen Prac4.doxy to make the documentation). You should then get a file called doxy and inside that you should find a html subfolder and an html subfolder. Go into the html subfolder and see what the automated documentation looks like (note that it also generates a latex version but this needs to be manually compiled to generate the pdf providing the same documentation).

Part 4: Report

Compile your experiments and findings into an IEEE-style conference paper. Compare results with those from the previous median filter prac.

The page limit is 3 pages. Submit your paper and zipped code repository to the Vula Assignment for this practical.

END OF ASSIGNMENT