School of Computing: Assessment Brief

Module title	Parallel Computation
Module code	XJCO3221
Assignment title	Coursework 2
Assignment type and description	MPI Programming Assignment and Analysis
Rationale	To implement a distributed parallel program in MPI, use collective communication to distribute the problem and accumulate the answer, and to implement a binary tree using point-to-point communication. Finally, to evaluate and discuss the parallel scaling of your solution.
Guidance	See overpage (identical to COMP3221)
Weighting	20%
Submission deadline	10am (China), Monday 17$^{\text{th}}$ April.
Submission method	Gradescope
Feedback provision	Marks and comments returned *via* Gradescope
Learning outcomes assessed	Apply parallel design paradigms to serial algorithms. Evaluate and select appropriate parallel solutions for real world problems.
Module lead	Peter Jimack

The Assignment

1. **Assignment guidance**

   Implement an MPI program that reads in an image file and outputs a histogram of the number of pixels with each grey scale value. Constructing the histogram should use multiple processes. The image files are in uncompressed `.pgm` *(portable greymap)* format, in which each pixel takes a single integer value corresponding to its greyscale value, with 0 for black and some value `maxValue` for white. The image dimensions and `maxValue` are provided in the first few lines of the image file[1].

2. **Assessment tasks**

   You are provided with code that reads in the image file and constructs the histogram in serial. You need to adapt the code so that it constructs the histogram in parallel using MPI. For the first part of this assignment, you should use collective communication routines wherever possible, including distributing the necessary global parameters (*i.e.* the number of image pixels per process, and `maxValue`) to all processes.

   Once you have this working, you should then implement an alternative implementation for the distribution of global parameters that uses point-to-point communication in a binary tree pattern. This method should be automatically called if the number of processes `numProcs` is a power of 2; if it is not a power of 2, your code should continue to use your original version. Your submitted code should therefore include two versions of the distribution of global parameters, each being called depending on the number of processes specified when launching with `mpiexec`.

   Once you have the final version of your code, perform timing runs using the batch queue on `cloud-hpc1.leeds.ac.uk` to determine the parallel execution time as the number of processes is varied (the provided code already outputs this time). The combination of numbers of processes and nodes is given in the file `readme.txt`, and you should insert your results and discussion into this file, and include it when submitting. You should also interpret your results in terms of your understanding of MPI and the architecture you used for the batch nodes on `cloud-hpc1.leeds.ac.uk`.

   You have been provided with code that reads in the image file to a data array on rank 0, with additional padding values of -1 so that the total array size is a multiple of the number of processes (this is a common way of handling data sets of arbitrary size). An example of how to construct the histogram in serial is also included.

[1] Note that `.pgm` is an ASCII format which you can read in a standard text editor. You can also view as an image using, for example, "`gimp image.pgm`" (gimp is installed on `cloud-hpc1.leeds.ac.uk` but remember to use "`ssh -Y`" to ensure X forwarding). You can also use gimp (and other tools) to convert images to `.pgm` format (use "`export as`" from the `file` menu in gimp).

The provided files are:

`cwk2.c`	:	Starting point for your solution.
`readme.txt`	:	Use to provide results of your timing runs, and interpretation.
`cwk2_extra.h`	:	Routines to load the image and save the histogram. **Do not modify this file**; it will be replaced with a different version for assessment.
`image.pgm`	:	An image to test your code on.
`makefile`	:	A simple makefile (usage optional).
`plotHistogram.py`	:	Plots the histogram file (usage optional)[2]

3. **General guidance and study support**

   If you have any queries about this coursework please raise these during your timetabled lab sessions in the first instance.

   It is recommended that you proceed incrementally, testing your code after each stage of your calculations. For the binary tree, you may find it useful to insert print statements showing which rank each process is sending to or receiving from, to ensure that all sends have corresponding receives.

   Collective communication was covered in Lecture 10 and reduction in Lecture 11. For the binary tree, you may like to consider an upside-down version of the second binary tree considered in Lecture 11, as it is easier to code. Think carefully about which ranks send data, and which receive, at each level of the tree. You may find the following useful:

`1<<n`	:	Evaluates as 2^n.
`if( n && ((n&(n-1))==0) )`	:	Evaluates as `true` if `n` is a power of 2.
`int lev=1; while(1<<lev<=p)lev++;`	:	Finds the number of levels `lev` in a binary tree with `p` leaf nodes.

4. **Assessment criteria and marking process**

   Your code will be checked using an autograder on Gradescope to test for functionality. Staff will then inspect your code in order to allocate the marks as per the mark scheme (see below).

[2] To use on `cloud-hpc1.leeds.ac.uk` you will need to use the anaconda version of python (since the default system version does not include the necessary packages). This requires you to activate it first, with "`conda activate base`", and then you can execute it with "`python plotHistogram.py`" (assuming the output file `hist.out` is in the same directory).

5. **Submission requirements**

   Submission is *via* Gradscope.

   Since you should only have edited `cwk1.c` and `readme.txt`, submit only these two files.

   Do not modify `cwk2_extras.h`, or copy any of the content to another file and then modify. This file will be replaced with a different version as part of the assessment, so it is imperative that your code still calls these routines.

   You should first submit your solution to **Coursework 2: Check submission**. This will run some checks to make sure your solution runs on the test system, and immediately report back any problems. Only after you have passed all tests should you submit your solution to **Coursework 2: Final Submission** for assessment. Your timing runs should be performed on `cloud-hpc1.leeds.ac.uk` prior to submitting to Gradescope.

6. **Academic misconduct and plagiarism**

   Academic integrity means engaging in good academic practice. This involves essential academic skills, such as keeping track of where you find ideas and information and referencing these accurately in your work.

   By submitting this assignment you are confirming that the work is a true expression of your own work and ideas and that you have given credit to others where their work has contributed to yours.

   Code similarity tools will be used to check for collusion, and online source code sites will be checked.

7. **Assessment/marking criteria**

   There are 20 marks in total:

4 marks	:	Histogram calculated and output correctly.
7 marks	:	Parallel calculation of histogram without the binary tree.
6 marks	:	Binary tree distribution of global parameters.
3 marks	:	Speed-up calculations and interpretation.

   Note that you can achieve a first class mark without implementing the binary tree, in which case your code should use collective communication for the distribution of the global parameters for any number of processes.