UNIVERSITY OF EXETER
COLLEGE OF ENGINEERING, MATHEMATICS
AND PHYSICAL SCIENCES

ECM3426/ECMM448
High-performance Computing and Distributed Systems

Continuous Assessment (2)

Date Set: 17 March 2020
Date Due: 30 April 2020
Return Date: 21 May 2020

This CA comprises 15% of the overall module assessment.

This is an **individual** exercise and your attention is drawn to the College and University guidelines on collaboration and plagiarism, which are available from the College website.

This is the second coursework (Distributed Systems) for this module and tests your understanding of distributed system architectures and Inter-Process Communication.

Calculating the Mandelbrot set with distributed computing

In the High Performance Computing part of the module we studied method for solving problems with parallel computing using OpenMP and MPI. In cases where a workload can be decomposed with little or no communication between the different parts it may instead be appropriate to use distributed computing. Examples of using distributed computing to solve scientific problems include climateprediction.net (https://www.climateprediction.net), and SETI@home (https://setiathome.berkeley.edu).

Workshop 2 used a C program parallelized with OpenMP to calculate the Mandelbrot set, and workshop 4 used a C program parallelized with MPI. In this assignment you will **design** a distributed computing system to calculate the Mandelbrot set (https://mathworld.wolfram.com/MandelbrotSet.html), and **implement** your design using the Java programming language.

Background

The Mandelbrot set is the set of complex numbers which contains the points for which the iteration

$$z_{i+1} = z_i^2 + C$$

remains finite, where C is a complex number and the initial value $z_0 = C$. This apparently simple relation leads the complicated fractal structure shown in Figure 1.

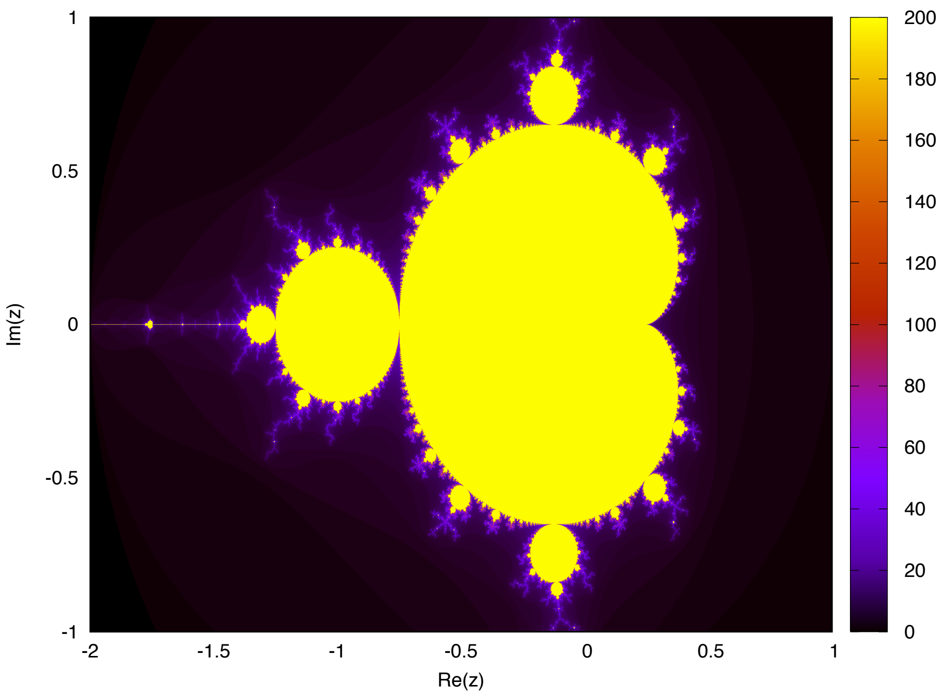

Figure 1: the Mandelbrot set over the range -2<Re(z)<1, -1<Im(z)<1 with a maximum of 200 iterations. .

To calculate the iterations it is not necessary to use a complex number variable type. The following code snippet shows how to calculate the Mandelbrot iterations using only floats and ints:

```
float z_Re = z0_Re;
float z_Im = z0_Im;
int niter = 0;
while (niter < 100){
    float z_sq_re = z_Re*z_Re - z_Im*z_Im; // Re(z^2)
    float z_sq_im = (float) 2.0*z_Re*z_Im; // Im (z^2)
    float mod_z_sq_sq = z_sq_re*z_sq_re + z_sq_im*z_sq_im; // |z^2|^2
    if ( mod_z_sq_sq > 4 ) break; // Equivalent to |z^2|>2
    z_Re = z_sq_re + z0_Re;     // Update z to value at next iteration
    z_Im = z_sq_im + z0_Im;
    niter++;                    // Update iteration counter
}
```

This is not the only way to implement the iterations but you may use this code snippet verbatim in your program if you wish.

Requirements

For this assignment you should design a client-server system which calculates the Mandelbrot set by implementing manager-worker parallelism. The server is the manager which hands out work to worker processes running on clients. The clients calculate the number of iterations for which $|z| \leq 2$ at points in the complex plane and report the results to the server.

Your system should be able have the following characteristics:

- The server (manager) should be able to handle requests from an unspecified number of clients (workers) i.e. the server should not rely on a particular number of clients being available
- The server must write results out to a file with ordered entries i.e. written in order of increasing real part, then increasing imaginary part and not simply the order in which clients report the results
- Clients should exit cleanly when the calculation is complete

To demonstrate the correct operation your system you should calculate the Mandelbrot set in the region $-2 < Re(z) < 1, -1 < Im(z) < 1$ i.e. the same region as shown in figure 1. The data set used to plot figure 1 has 3000x2000 values but you may choose to use fewer values if the calculation takes too long to complete at this resolution.

Deliverables

The coursework requires electronic submission of a report. The report should describe the **design** of your program including the choice of transport layer protocol (TCP or UDP), and the sequence and content of messages passed between server and clients.

The report should also present the code used to **implement** the client and server programs. You should also include a screen shot showing your programs carrying out the specified task, and a plot of your results similar to that in figure 1. Figure 1 was generated using gnuplot but you are free to choose a different plotting package and colour scheme if desired.

To ensure that your work can be marked anonymously you should remove your user name from the command line prompt when taking the screen shots. To do this run the command:

```
export PS1='> '
```

Submission must be made by 12pm (noon) on the date indicated on the front page of this document.

Submission

The coursework requires electronic submissions:

- You should submit a PDF version of your report to eBART
- You should also submit the source code for your client and server programs using the Turnitin submission link on the ELE page: https://vle.exeter.ac.uk/course/view.php?id=4176

Marking Scheme

The following aspects will be evaluated:

- Correctness of the design and code: 40%
- Explanation of the design and code: 40%
- Screenshot showing system in operation: 10%
- Plot of results: 10%