Calculating the Mandelbrot Set with Distributed Computing

Design of the program

For this program, it is essential that no packets are lost, otherwise some iterations may be skipped, resulting in some black lines across the output image. For this reason, I will be using the TCP protocol for the transport layer, to ensure no data is lost.

The system implements manager-worker parallelism, with a server being the manager that handles requests from an unknown number of clients. For each index in the real axis, the server will wait for a "nextIter" request from a client, where it will open a new connection in a new thread and send across the next value on the axis. When the client gets the value, it will loop over the imaginary axis for that real value and store the results at each of these coordinates. When the server has handed out all the real values, there will be an unspecified number of clients that each have different parts of the results.

Now when the server receives a "nextIter" request, it will return an integer being -1, this notifies the client that all values have been calculated, and the client will know to begin sending its results. Each client will now store the results of the 2D array as a 1D array of strings, these strings will be in the format "A-B-C" where A is the real value, B is the imaginary value, and C is the result. We only need to store values where the result is not 0 in this array, as those are the only values we need to change in the server's results array. Once this array has been created, the client sends a "startColl" request, this signals the server to start a new connection, where it waits for an integer, this integer will be the length of the list that the client is about to send. The connection now starts a loop that iterates for however long the array is, and each iteration it waits for the next string from the client. When this string is received, we can split the string into its individual values, and store the values in the results array in the server.

When all values that the client has calculated have been sent, it can close. Once all created clients have closed, a new client is created who's only purpose is to send an "end" request. When a connection receives an end request, it notifies the server to start writing the results of the array, and when this has finished, the connection will send back a -1 integer so that the client can be notified that all the results have been written, and the results file can now be viewed.

The following pages show screenshots of the task being carried out, with four clients used, and the output of the results array, with 3000x2000 values. Sadly, the plot isn't exactly what it should be, with results only appearing to be correctly recorded around a radius somewhere in the middle. I believe this to be a problem with the algorithm that solves the equation and not to do with the parallelism or client-server architecture, as the same algorithm was tested in a serial context and the same results were output.

Screenshot Program Running

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

[redacted]\client>javac Main.java

[redacted]\client>java Main
Client Calculating Results 1. Clients Calculate Results
Client Calculating Results
Client Calculating Results
Client Calculating Results
Client packing results 4. Clients recieve end flag, and so pack results
Client packing results    into 1D array
Client packing results
Client packing results
Client Sending Results 5. Clients make request to send results
Client Sending Results
Client Sending Results
Client Sending Results
Server notified about finishing 9. Each client recieves the end flag
Client finished
Server notified about finishing
Client finished
Server notified about finishing
Client finished
Server notified about finishing
Client finished
All clients finished 10. All clients have finished, a new client sends an "end" request

[redacted]\client>
```

```
C:\Windows\System32\cmd.exe - java  Main
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

[redacted]\server>javac Main.java

[redacted]\server>java Main
starting mandelbrot
Gathering Results 2. Server Finishes handing out iterations
initial request: nextIter 3. Clients request new iterations, but server is finished,
initial request: nextIter    so it reponds with -1 (end flag for clients)
initial request: nextIter
initial request: nextIter
initial request: startColl 6. Server recieves request to recieve results
initial request: startColl
Collecting list of size: 1157330
initial request: startColl
initial request: startColl
Collecting list of size: 1132434
Collecting list of size: 1137259
Collecting list of size: 1146783
all added
all added
all added
all added
initial request: end 11. End request is recieved
Writing results 12. Server writes results
Finished Writing 13. Server indicates it has finished
```

7. Server recieves the array size for each client

8. Server finishes recieving results, and so responds back with -1

Screenshot of Final Plot