Christopher Lewis

PA2: Mandelbrot

3/14/2017

REDO: 3/27/17

Overview:

For Project Two, We did Mandelbrot. Mandelbrot is a fractal that has unique properties to it. One of which is that it is embarrassingly parallel which means that it has little to no message passing for the entire runtime except right at the start of runtime. This project takes two pieces of code and compares the two together. One being sequential and the other being parallel. The findings may be fascinating.

Diagrams/Code:

Sequential mandelbrot proved to be quite trivial as it did what I did expect it to do.. And that was to be very fast. Sequentially Mandelbrot didn't take time at all as it was very linear in time when the load got harder. Refer to figure 1 to describe the results.

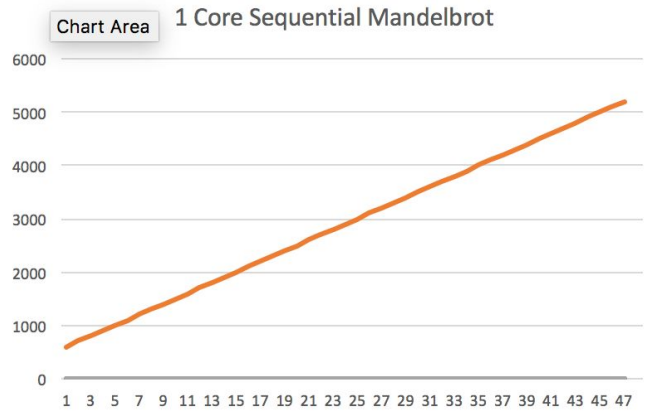

Figure 1: X-Axis is the time taken starting at 0.0 and ending at 10.54 seconds. The Time was very linear as the Y-Axis describes the resolution of the picture Which started at 600x600 and worked its way up.

Running sequential mandelbrot produced a result that was very usual and showed that the image that needed to be ran was a success. Refer to Figure 2.

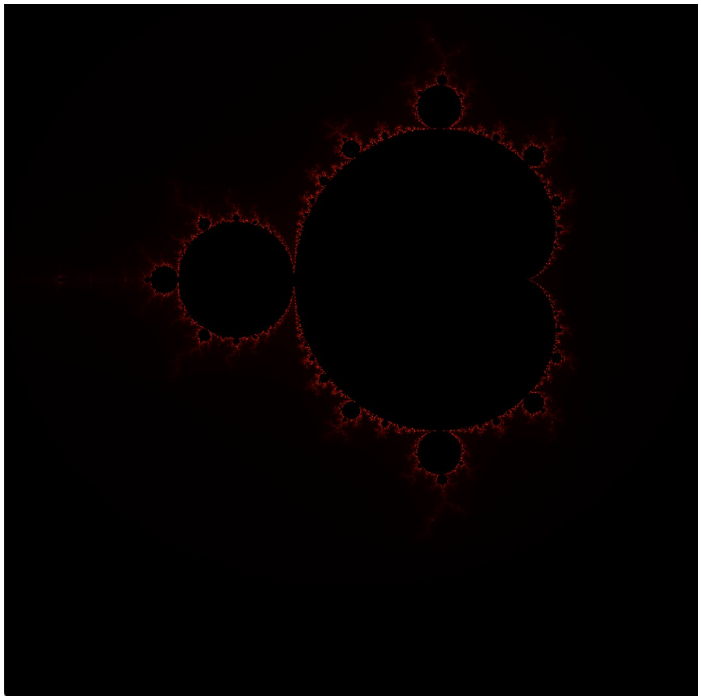

Figure 2: Above is the result of the mandelbrot set. This picture is a little hard to see but the line is in dark red and shows the production of the mandelbrot set.

When you change the color array of the pixels in mandelbrot you can produce a similar result but in a different color see Figure 3 for an example.

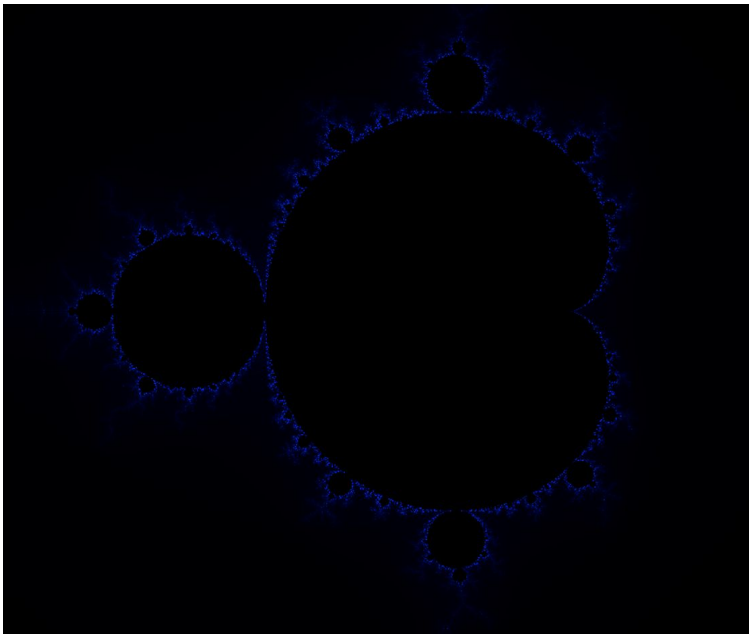

Figure 3: Mandelbrot but in blue.

Dynamic mandelbrot proved to be very challenging in its nature since it took the most time to code. Dynamic mandelbrot took node balancing into play because at any time there was always a node working on a task so there wasn't a stray node off doing nothing while the others were working. This is important in parallel computing because a node sitting is time wasted. To implement this you need to do message passing to make sure there is always work being distributed. Below [Figure 4] is a snippet of code that helped produce results and aided in message passing.

```
MPI_Send( &row, 1, MPI_INT, pingPongSend, DATA_TAG, MPI_COMM_WORLD );
row++;
```

Figure 4: Above is a line of code (note the pingPongSend) that takes
Any nodes that are needing work and give it to them by giving them a row

Dynamic allocation produced a graph that was not ordinary by any means. The results looked the exact same as the sequential except that the time was faster when doing by parallel. Figure 5 below shows the results.

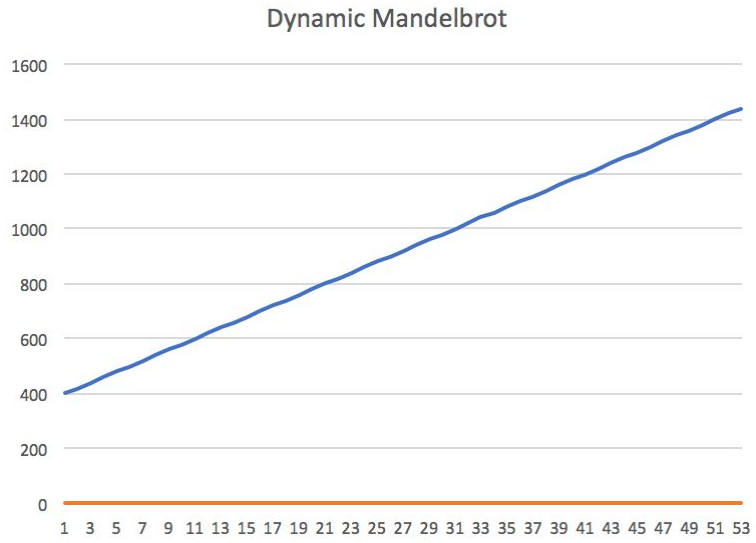

Figure 5: X-Axis is the time taken starting at 0.0 and ending at 0.192580 seconds. The
Time was very linear as the Y-Axis describes the resolution of the picture
Which started at 400x400 and worked its way up.

When the results were compiled the data taken shows that the best amount of processors used is 2 or 4 on **one box** after the amount of boxes increased the time spikes increased. Figure 6 below shows an example.

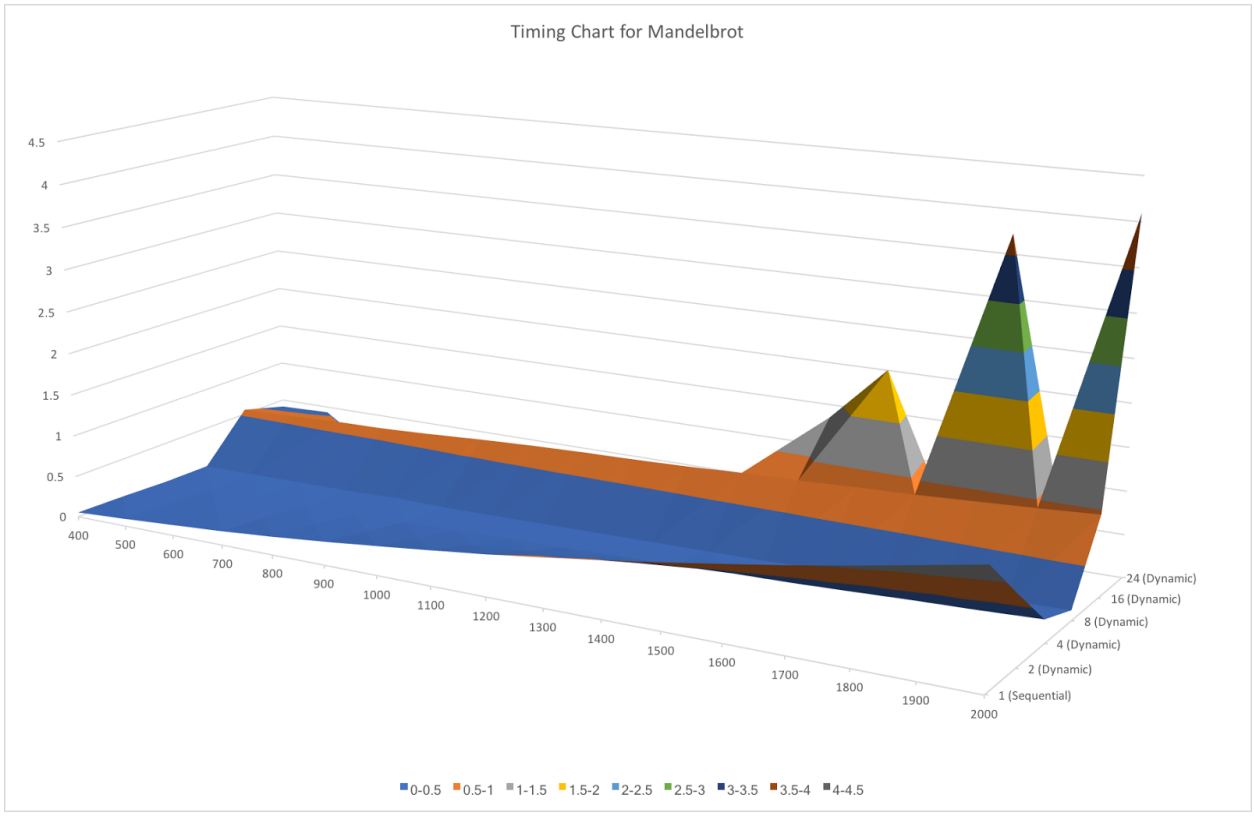

Figure 6: Above is a 3-D plot of the timing. The X-Axis (Labeled sequential/dynamic) is the number of cores. The Y-Axis is the image size and the Z-Axis is the time taken.

Speed-up

The purpose of mandelbrot is the importance of speedup which is defined as:

Speedup= Ts/Tp

The speedup I had received is based in the table below.

	Speedup
1 (Sequential)	NA
8 (Dynamic)	11.63

Dynamic and sequential mandelbrot looked the exact same as they both produced a picture like in figures 2 and 3 above. Looking at speed-up produced positive results because I found that I did get speed up. Overall, the project was a success and the findings presented showed success.