Kurt Andersen
CS 615
PA2 Mandelbrot
March 14, 2017

Introduction:

For this project we were assigned to copy a program out of the book that generated the Mandelbrot set. Our main goal for this assignment was to learn how to do dynamic job assigning, as well as figuring out the best way to split up jobs for a specific task. With this project we had to create a sequential program, and a static job assignment or a dynamic job assignment. We were then to analyze the data in order to determine when the process ran the best for each given sized image.

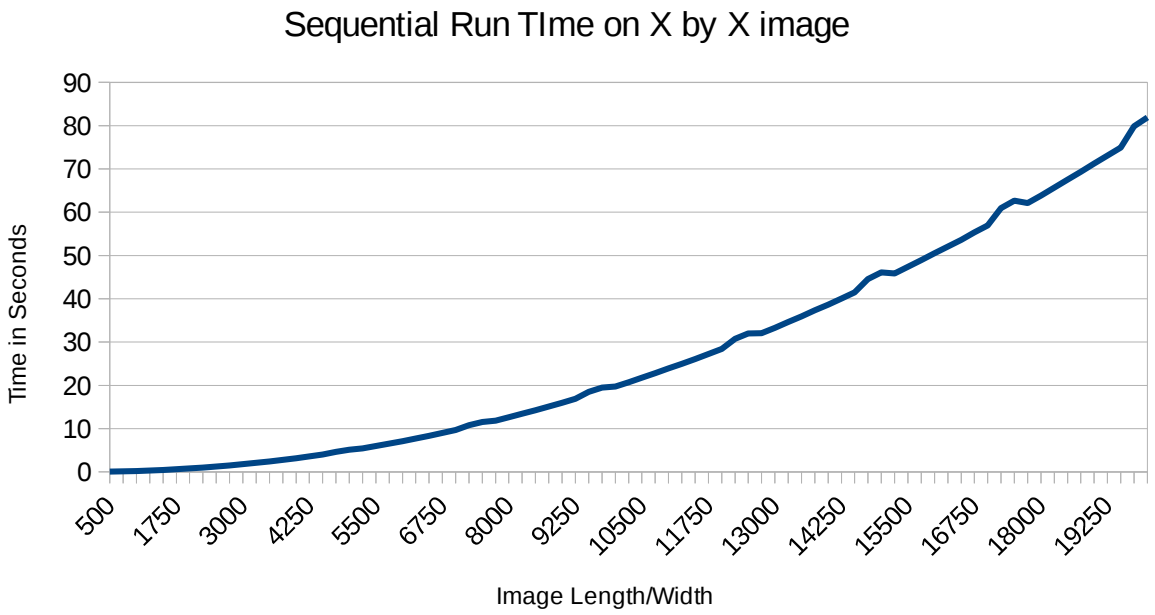
Process:

When I sat down to do the code for this project, the coding aspect of it was actually quite simple. The book had verbatim what needed to be written as code. The only difference really was just the naming convention of the variables being used. For this project I used a single dimension array because I found it easier to organize the data this way rather than a multi-dimensional array. The best reason for using a single dimension array is if I wanted to send multiple lines to be calculated at once, I only had to send a single data address. The sequential code was just a double nested for loop that would always take O(N^2) for the time complexity of the algorithm. When using parallel processing we were able to reduce it down to nearly O(nlogn) for the time complexity. The parallel portion was also written verbatim for us in the book. This included both dynamic and static job assigning. I however only did dynamic job assigning. With my batch files, I only wrote it to run it on a set number of processors. I would have to go in to the program to physically change the size of the images being generated, or go into the batch file to change how many processors it would be run on.

There were numerous issues that I came across while conducting this project. The depressing thing is that most of these issues were other students in the class, rather than technical issues with the coding portion. When trying to collect my own data, I would place a single job in the queue in order to run 1 test. While there were other selfish students who didn't care about anyone else in the class trying to gather their own data. Students would queue upwards of 100 jobs at one given time. These students

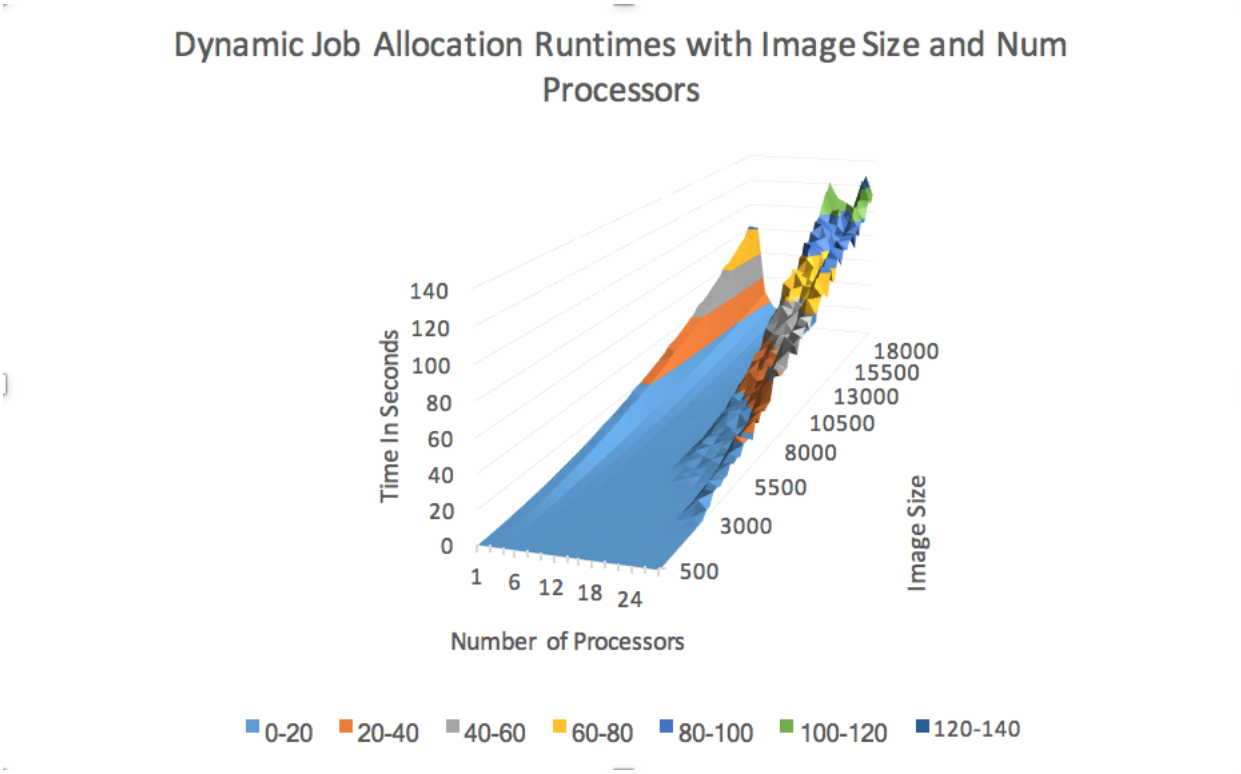
would not set a time limit on their jobs, so the jobs would run well over 5 minutes. The other students would use more than 4 total nodes, reducing the amount of available nodes for everyone else. This was especially frustrating because it was clearly stated in class not to use more than 32 processors (or 4 nodes). This made it extremely difficult to gather my own personal data. I believe we need to implement some sort of administrators on the cluster in order to stop this kind of abuse on the cluster. I don't think students should be able to queue more than maybe 5 jobs, and have a forced time limit on the job. There is no reason these students should be stealing all the resources for themselves when there are 60+ other students needing the same resources.

Results:

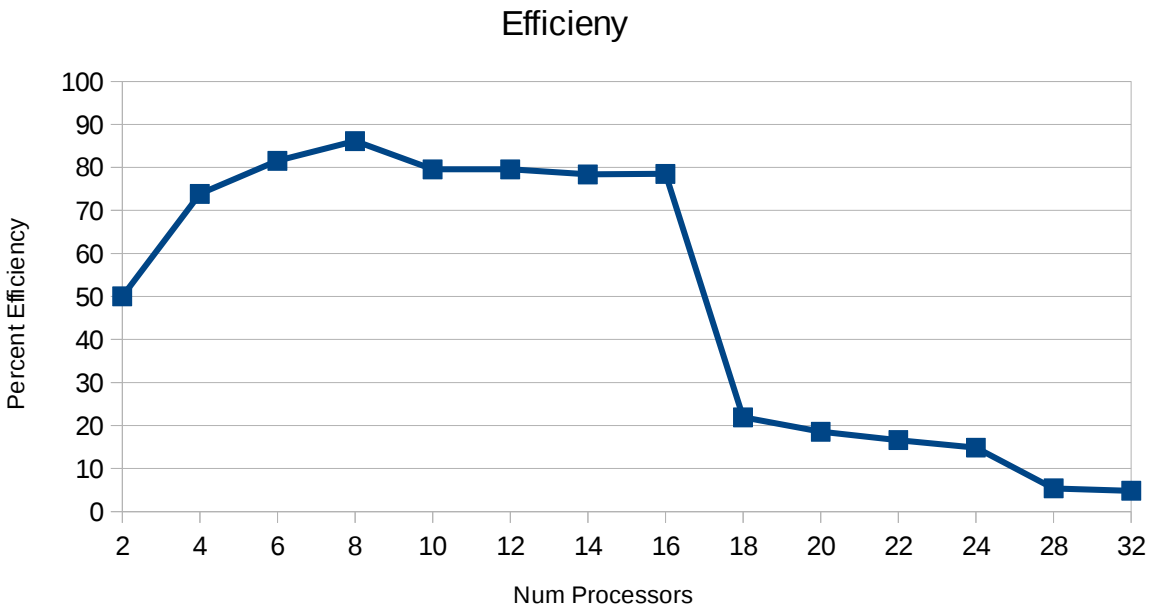


This graph the total run time it took to calculate the Mandelbrot set on a single processor. The growth in this graph appears to be an exponential increase. This is due to the time complexity of a double nested for loop where each loop runs N amount of times. N being the length of the image. The width of the image is also the same value as the length. The time complexity reflects on this graph for the

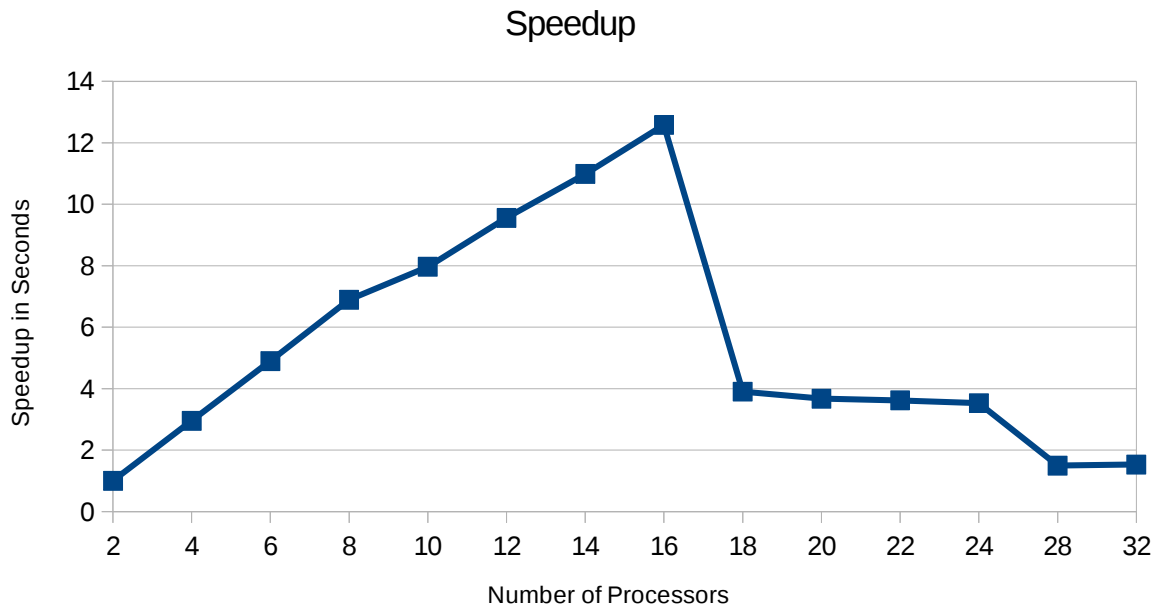
overall run time. The bigger the image became, the longer it took to calculate the overall data for the entire image.



In this graph, it shows the overall distribution of the time complexity of the number of processors, compared with the image size and the number of processors being used. The higher up the graph goes, the amount of time it took for the process to complete. The depth of the graph shows the size of the image. The color slices are just to show an easier view of where the time differences are on the surface. In this graph though, it shows that about 16 cores was the sweet spot for the parallel computing time. When there were less cores, there was more time spent computing all of the data on each slave process. Where as we gained more processors, although less computing was done per processor, the amount of time taken to send the messages between the master and the slave process increased. We started to spend more time just sending messages than actually doing useful computations.



This graph shows the percent efficiency for the number of processors used. The data in the line is the average runtime per number of processors and how much more efficient the program is than the sequential. The peak point of the efficiency is at 8 processors, but as the image size increased, as seen in the last graph, the time remained lower around 16 processors.



This graph shows the overall average speedup time in seconds compared to the number of processors being used. As seen in the 3d plane above, 16 processors is our sweet point. Where when we go beyond, we really aren't being that much more efficient. The most improved times remain between 8 processors and 16 processors, where 16 is being the best possible average outcome. This is the point where the right amount of processing was balanced with the proper number of computational time.

Conclusion/Future Work:

Overall I feel as though this project was a great way to learn how to distribute jobs statically and dynamically. It gave me a good idea of how much faster dynamic job allocation can make processes run in comparison to static job allocations. As I just stated this was a good way to learn dynamic and static job allocation because the calculations on the slaves side were so simple, then it was just as easy to place the computed data back into the original data array.

In the future I will be able to apply what I learned here to any other parallel project I do. This project taught how to allocate jobs to the slaves and it will allow me to organize my future jobs in the best way possible.

Special Mentions:

I would like to note that I spoke with Vinh Le and Connor Scully-Allison. We discussed different ways to obtain multiple sets of data in order to construct our data sets for our graphs. We also compared data to see how similar each others data was.