Mark Randles
Lab 2 - Report
2004-03-17

As of the time of writing the dynamic allocation portion of the assignment is still incomplete. Problems have occurred that take inordinate amounts of time to diagnose and due to the limited time available I was unable to complete this part of the assignment. However I believe that a cursory analysis of the algorithm i used to do the dynamic assignment is in order.

The actual code that generates the Mandelbrot set is rather simple, and runs at O(n) = n^3 time. Also the quality of plot is dependent on the number of iterations, and for most of the “classical” views of a Mandelbrot set, they only require a max of 1000 iterations. However to keep the iterations constant across all tests and as an effort to get the best possible plot of the set at a variety of zoom factors, 100000 iterations was used as the max threshold.

For a single processor (using the OSC login machine as the test system) it took approx. 405 seconds to execute the single processor version of the Mandelbrot code. This was close to what was expected for the test criteria.

The statically allocated version of the Mandelbrot code was incrementally faster. The adaptation of the algorithm to a master/slave MPI code was quite easy. The actual speedup factor of the parallel code on the test set of nodes was quite surprising. For the most part the speedup factor appeared to be constant, and I believe this is due to the fact that both sides of the speedup curve appear in the data. More data would be needed to actually validate my hypothesis, but it would be the logical following from what data is available.

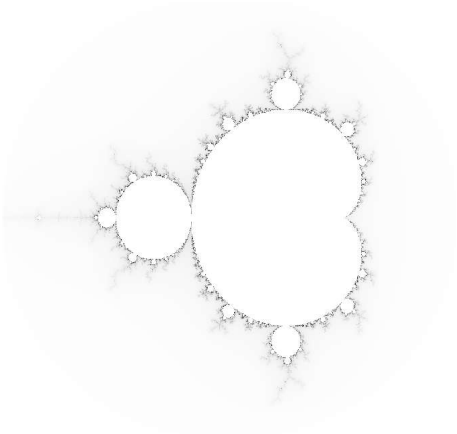
The dynamic algorithm used was a direct adaptation of what was present in both the single processor code and the static allocation code. The each child was essentially a endless while loop that could be broken when the parent sent a sentinel value as the line to process. This made the most sense as the child is essentially dumb as to how many lines it will need to process. The parent used a for loop to answer each child's request for another line. The request was a array of the processed data with the line number of that line as the final element in the array. The parent would immediately dispatch the next line to start the child processing, and then the parent would process what data that the child sent back.

I'm not sure what the hangup is with the dynamic algorithm. It's taking

a while to run, which can either mean that the sentinel value isn't being sent or there is way to much communication overhead for the program to complete in a reasonable manner. Either option has about the same probability of happening.

Since the lab is incomplete no final conclusion can be gathered from this experiment. A preliminary conclusion is that dynamic allocation is slower then the static allocation of the dataset. This would be assumed as the communication overhead for passing the messages in the dynamic algorithm would kill any small speedup that the dynamic algorithm would provide in computation time. However the dynamic approach might show a greater speedup on a slower cluster where the time to compute a iteration is much higher then on the cluster this was tested on.

And now for some pretty pictures...



Full Mandelbrot Set

Zoom in on bottom half

(Pictures were generated with my single processor program)