An Investigation of Chaos Theory Using Supercomputing Techniques

Bryan Ward

June 12, 2007

Abstract

Chaos theory is the study of dynamic systems in which small differences in the environment, can create large, unpredictable results. The classic example of chaos theory is the Butterfly effect, or the theory that a Butterfly flapping can effect large scale weather patterns such as tornadoes and hurricanes even from hundreds of miles away. Chaos Theory is also applicable in systems other than weather, such as the stock market and physics. While these are very complex systems, there are chaotic mathematical systems represented by fractal images which this project aims to investigate. In this project distributed computing algrithms will be used to investigate these fractals.

1 Introduction

1. Problem Statement
   Generate and analyze several different fractals to gain more insight into the complex topic of chaos theory. Distributed computing algorithms will be learned and applied to this problem to increase performance.

2. Purpose and Goal
   The purpose of this research project is to implement distributed computing algorithms while investigating the complex topic of Chaos Theory. I chose this field for my project because Chaos Theory as fascinated me for years. This subject lends itself to the multi-processor supercomputer or a computing cluster. Another goal of this project is for me to learn about high performance computing.

2 Previous Work

Chaos Theory has many applications to numerous different real world. Lorenze coined the phrase Chaos Theory while studying meteorology because there are so many different variables that can have significant effects on weather patterns, even if they are only minor initial changes. Chaos is also evident in physics. One example could be letting a full balloon loose, the result is an unpredictable flight. This is because the countless air molecules can have differing effects, and the trajectory of the balloon is therefor unpredictable. Some more recent research has been found showing that the structure of arteries in the human body is chaotic and may be coded as such in the DNA. Brain and heart functions may be chaotic as well.

Fractals have applications outside of the physical sciences as well. In computer science, there is work being done to use fractals to generate more realistic computer graphics. This not only creates more life like graphics, but also speeds up the rendering process. Work has also been done to use fractals to compress data, and especially images, using fractals.

3 Procedure

In this project the Mandelbrot and Julia set fractals were investigated. To begin with code was written to generate each of these fractals in a gray scale image on a single processor. This formed a basis for the rest of the project and each iteration enhances or adds functionality to these original programs.

The next iteration for this project was to generate a series of Julia set fractals to be strung together into a video. The program mencoder was used to create the video from the individual frames. This iteration was still run on a single processor.

The next logical step was begin using the Message Passing Interface (MPI) to distribute the computational load of all of these frames to different processors. This was tested on both the School Cray SV1 and the Sun cluster. This was an excellent starting point for getting to know MPI because there was very little data that had to be sent between processors.

To extend my knowledge of MPI, I then looked at distributing the workload of an individual fractal, such as a single frame of the Julia Set or the Mandelbrot Set. This was done by breaking the image into blocks which were then distributed to different processors. Each processor then iterates through each point recording the result in RAM. Each processor then sends the resulting array back to the master processor which compiles the fractal images and write it to a single file.

Until this point every fractal generated was done in gray scale but no study of fractals is complete without color. Different color schemes were experimented with, each with differing weights given to the red, green, and blue color components based on the escape iterations computed. This results in banded images without as high a level of detail. The distance estimator coloring algorithm was used to add more detail to the colors. In order to produce valuable fractal images the colors had to be generated in the Hue - Saturation - Value color space (HSV). These values had to be converted back to the traditional Red - Green - Blue color space (RGB) to be output to the file. The HSV color space allowed for the use of both the escape iterations algorithm and distance estimator algorithm to both be used to increase the detail of the colors.

The "Buddhabrot" rendering algorithm was also written both for a single processor and for MPI. This algorithm plots the iterates of points which are in the Mandelbrot set. The resulting image is said to look like buddha, hence the name. In this rendering algorithm increasing the number of iterations and the number of points plotted drastically increases the clarity and precision of the fractal image. With multiple processors the precision is achieved much more quickly and easily.

Original file output was done in ASCII for easy reading and writing (by the user). Binary file output was implemented to decrease the file size and I/O time. This was especially useful when viewing very large fractal images because the file had to be read into RAM to be viewed on the screen.

4 Testing and Analysis

In this project there were two types of testing, testing for proper results, and testing for performance. The first of these two was rather simple in that there are many fractal images on the internet of which results can be compared to. Performance tests were done periodically to analyze the efficiency of different algorithms and the performance increase with the addition of more processors. In the case of MPI testing the results can be compared to the results of its single processor counterpart. This would show not only if the actual results are different, but also any differing performance in runtime.

Tests on the Cray were conducted using anywhere from 2-16 processors. I have noticed that especially on the Cray, the more processors the program is run on, the more time it takes for messages to be passed between the processors. When a 12800 x 10240 pixel Mandelbrot fractal was generated on the Cray using 16 processors, it took approximately 20 seconds on average for each processor to iterate through it's respective segment, but the entire program took 83 seconds to run. This means that it took over a minute to pass the data between the processors. This was still significantly faster than running it on a single Cray processor which took 365 seconds.

After continuous hardware trouble with the Cray Supercomputer, final performance tests were run on the Sun cluster. These results show that the total time spent is equal to $(iterations * n)/p + (p-1)(t_m) + k$ where n is the number of pixels, p is the number of processors, and t_m is the time spent in message passing. This explains why the performance of n processors in not n times faster than a single processor. It also validates the above hypothesis that time spent in message passing increases with more processors.

5 Conclusion

In conclusion, using MPI and multiple processors the runtime is decreased. The runtime is not reduced by the factor of the number of processors used, because the processors must send messages to one another which can be time consuming. Passing large amounts of data between processors takes a substantial amount of time. Algorithms which utilize more processors but reduce the amount of information needing to be passed between the processors are the most efficient using MPI.

References

[1] Bourke, P., *An Introduction to Fractals.* Retrieved February 13, 2003, from http://astronomy.swin.edu.au/ pbourke/fractals/fracintro

[2] *Chaos Theory: A Brief Introduction.* (n.d.). Retrieved January 24, 2007, from http://www.imho.com/grae/chaos/chaos.html

[3] Devaney, R. L. (n.d.). *The Fractal Geometry of the Mandelbrot Set*. Retrieved February 7, 2003, from http://math.bu.edu/DYSYS/FRACGEOM/FRACGEOM.html

[4] Donahue, M. J., III (n.d.). *An Introduction to Mathematical Chaos Theory and Fractal Geometry*. Retrieved February 7, 2003, from http://www.duke.edu/ mjd/chaos/chaos.html

[5] Munafo, Robert. *The Encyclopedia of the Mandelbrot Set*. Retrieved June 11, 2007, from http://www.mrob.com/pub/muency.html