CS 415 Parallel Computing

Mandelbrot Sets

February 22, 2012

Erin Keith

Introduction:

This project focused on computing Mandelbrot sets and outputting an image which illustrates values contained in that set. A Mandelbrot set is a set of complex numbers, such that

$$M = \left\{ c \in \mathbb{C} | \lim_{n \to \infty} Z_n \neq \infty \right\}$$

where:

$$Z_0 = c$$
$$Z_{n+1} = Z_n^2 + c$$

Since the complex set can be represented in a two dimensional Cartesian coordinate system, and we can limit the number of iterations to 256, the form in Figure 1 will be displayed.

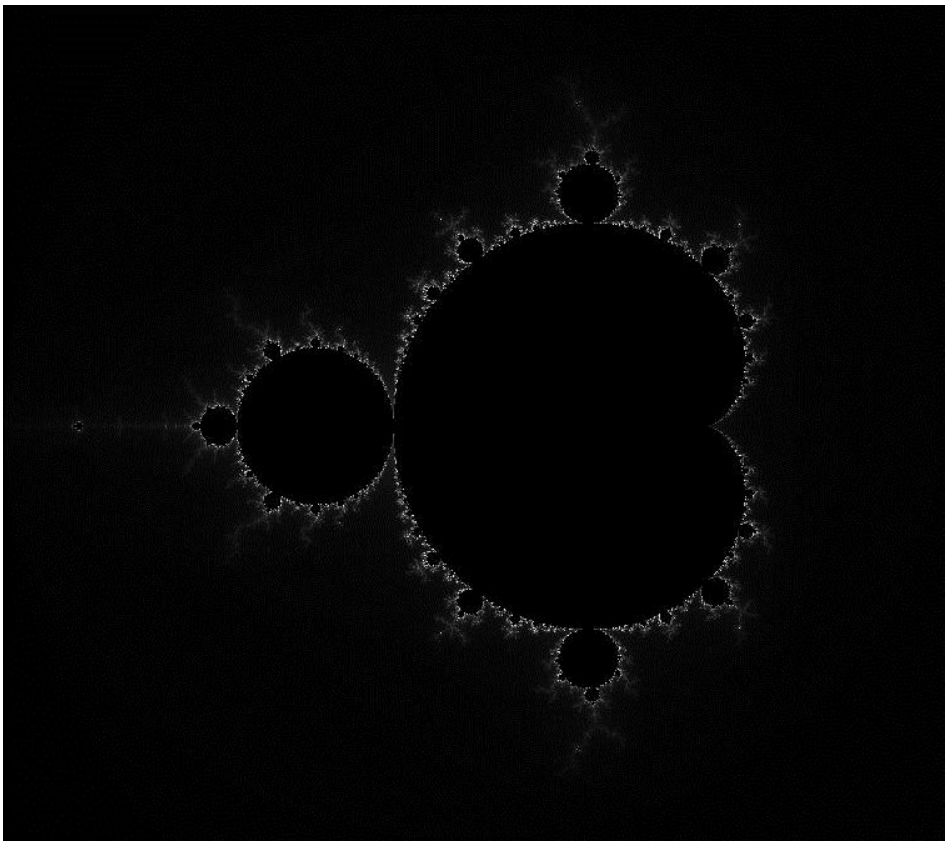

Figure 1: Mandelbrot Fractal

Procedure:

Sequential Program

To be able to determine speedup time, it was imperative to first create the algorithm, as provided in the book, sequentially. A struct, containing real and imaginary variables, was helpful for handling complex numbers. One complex struct handled the Cartesian coordinates as a representation of the set of complex numbers between -2-2i, and 2+2i, scaled by an image factor. Another complex struct started at zero and throughout the iterations the following formulas were computed:

$$z_{real} = z_{real}^2 - z_{imag}^2 + c_{real}$$
$$z_{imag} = 2z_{real}z_{imag} + c_{imag}$$

as long as the magnitude of z was less than 2 and the iterations were less than 256. The number of iterations was then stored as the pixel value for the image at that coordinate.

To determine the amount of time it took to calculate images of different sizes, each image size runtime was averaged over 100 executions on the grid, as shown in Figure 2.

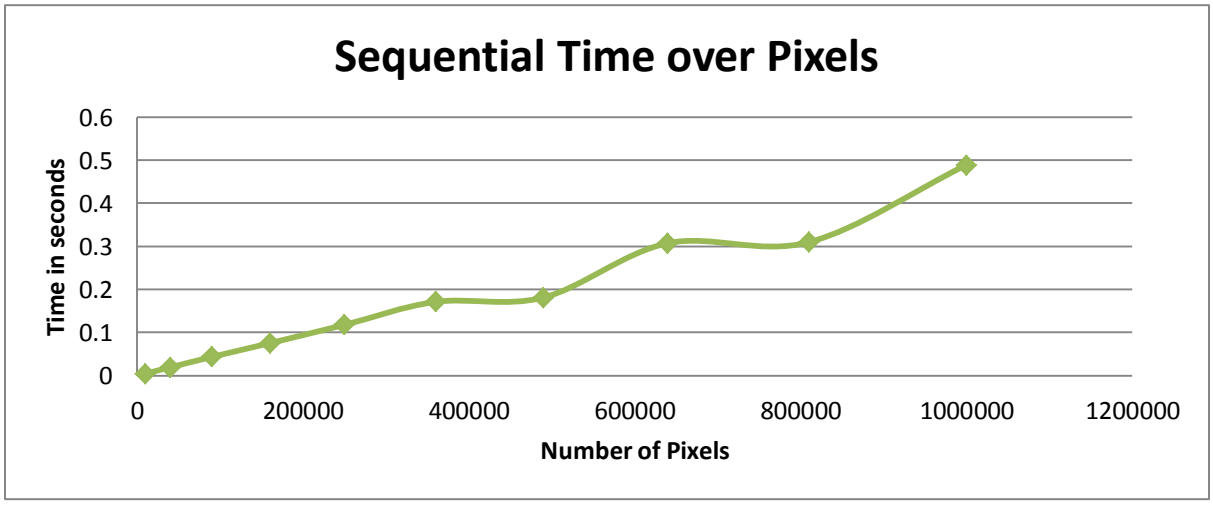

Figure 2: Sequential Runtime for Different Image Sizes

Part II: Static Parallel

The next part of the assignment was to design a parallel algorithm, assigning the slave processes a fixed range of rows for which to compute the values. Since the dimensions of my image were declared as global constants, I determined that it would be redundant to send the slaves their ranges. I made the slaves "intelligent" and had them compute their range to save communication time. The master then determined the appropriate range based on the ID of the process whose message it received. Figure 3 displays the runtime for the static parallel program, for the same range of pixels as the Sequential and a range of processors.

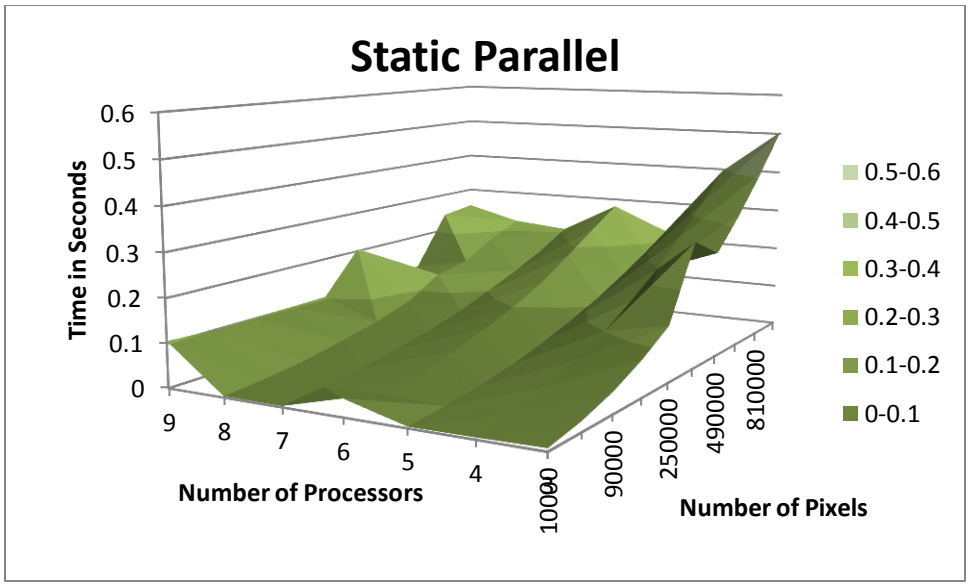

Figure 3: Static Parallel Runtimes

Part III: Dynamic Parallel

Finally, I attacked the problem of dynamically assigning rows from a workpool to processors as they became available. Instead of using the books suggestion to send the array of pixel values and the row number, I used a lookup table to keep track of the rows being processed based on processor ID, which helped cut down on communication time. Figure 4 illustrates the runtime of the dynamic parallel program over the same ranges as the static parallel program.

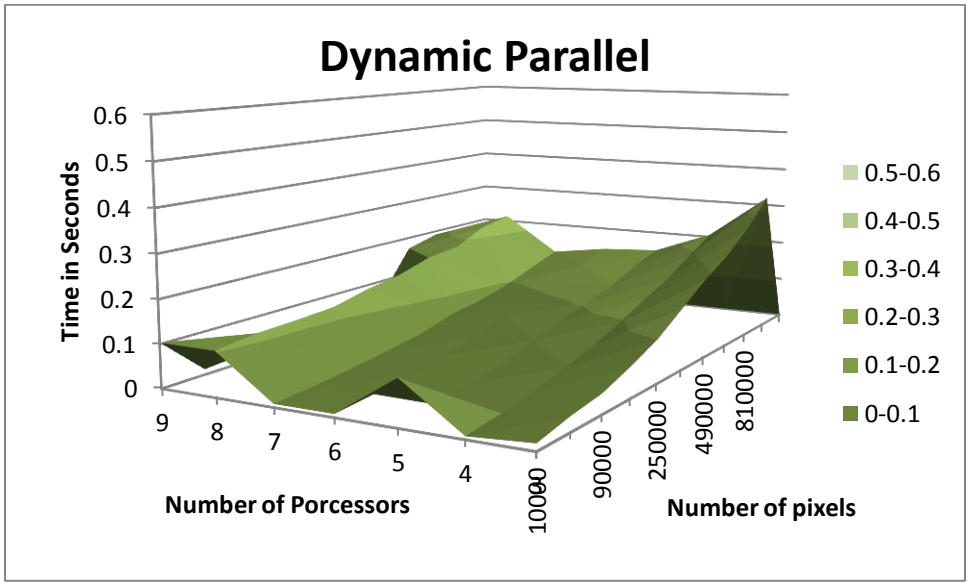

Figure 4: Dynamic Parallel Runtimes

Analysis:

The curve generated by the sequential program clearly demonstrates some sort of threshold between the 500 x 500 and 700 x 700 pixel images. Otherwise the curve shows the expected linear increase.

The curves produced by the parallel runtimes show a general trend of increasing with number of pixels and decreasing with number of processors. The overall runtimes of the dynamically implemented parallel program show a decreasing trend with number of processors, until we hit 8 processors. The probable explanation for this anomaly is my waiting until the day the project was due to finish running my tests. I believe the number of users on the grid significantly affected the data I collected. It is important to note that the dynamic program's overall runtimes were less than the static program, which demonstrates the increased efficiency of the dynamic over static parallel algorithms.

There is also significant speedup

$$\frac{t_{\text{sequential}}}{t_{\text{dynamic parallel}}}$$

of parallel over sequential algorithms, following a predictably increasing trend (except for that nasty 8 processor mark) as illustrated by Figure 5.

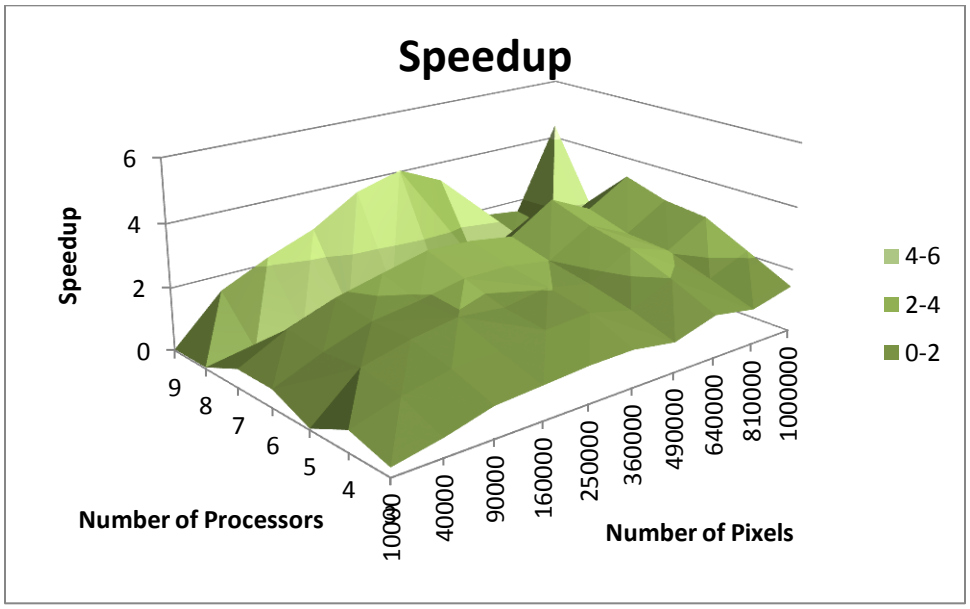

Figure 5: Speedup of Dynamic Parallel over Sequential

Conclusion:

My Mandelbrot programs were successful in demonstrating the speedup of a parallel implementation over a sequential one. My implementations also provide a good example of increased efficiency in dynamically parallel programs over statically parallel programs. I am especially pleased that my designs reduced communication time significantly; I noticed my runtimes were substantially less than the runtimes of some of my peers. This was an excellent illustration of key concepts in parallel computing.