PA 2
CS 615
By Samson Haile
March 14, 2017

Assignment Description

The objective of this assignment is to determine differences in computation time when executing a naturally parallel computation sequentially and in parallel. The computation involved calculates the mandelbrot set, which is used to produce an image related to the set. The larger the size of the image is, the more computation time is required to calculate the image. In the case of a parallelized algorithm, the computation time is expected to decrease with an increase in the number of cores utilized. With respect to parallel computation, subtasks of the computation can be distributed to slave processors in a static manner, as well as a dynamic manner. Both types of parallel algorithms will be implemented for this assignment and data will collected with respect to each parallel algorithm. The sequential algorithm will be plotted in a two dimensional pixel size versus time graph. The parallel algorithms will be plotted in a three dimensional pixel size versus cores used versus time graph. An assessment will then be made to explain trends and differences between the graphs.

Assignment Methodology

The assignment was implemented in the form of three c files and three batch files, as well as a makefile. Each c file and batch file match to designate one of the three implementation types of the mandelbrot set computation. This includes sequential, static parallel, and dynamic parallel. The sequential code simply executes the computation code on a single core, iterating across the different image sizes and computing the mandelbrot set for each image. The parallel programs uses the general heuristic the sequential does, but involves extra components for properly sending and receiving work between the master and slave nodes. The number of cores the parallel programs run on is modified through the -n and -N parameters of the batch scripts used to run the programs. This is to ensure that the programs request only as much resources as it uses.

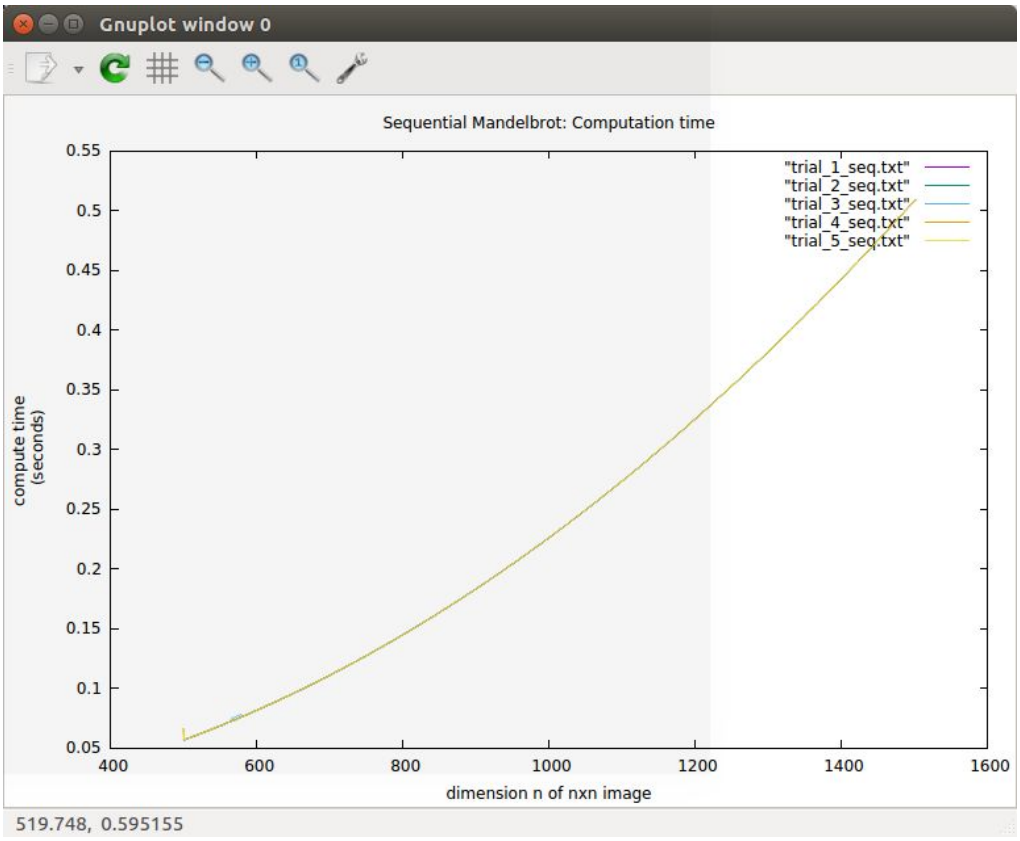
The implementation of the parallel programs share a common send and receive mechanism in order to relay work orders or the end of a work set. A slave knows work is done on an image if the master relays a pixel range of -1 to -1 for a dynamic allocation, or assumes it is done after it performs a singular set of computation. If the master sends otherwise, then the slave performs computation on all pixels within the pixel range. In a static allocation of tasks, the total number of pixels to compute is split evenly amongst all slave processes as best as possible. The slaves are then to assume that after work is done, they can return their results to the master and expect no more work. For the dynamic allocation, each slave is given a pixel range representative of a row of the image. When the slave finishes its work, it will receive another range of pixels to do work on and perform work on the next row. The slaves will then loop

through all assigned work until the master sends the range -1 to -1 to indicate all work has been assigned.

The c code is compiled through the usage of makefile which is capable of producing an executable to run the code, as well as the ability to remove the produced executable from the directory. Once the executable is produced, one of three batch files can be run in the form of the command sbatch <file_name>. The batch file SeqMandel.sh measures the time it takes to calculate the mandelbrot set across multiple images with sequential execution of tasks. The batch file SParMandel.sh measures the time it takes to calculate the mandelbrot set across multiple images with parallel execution of tasks under a static allocation of tasks to slave processes. The batch file DParMandel.sh measures the time it takes to calculate the mandelbrot set across multiple images with parallel execution of tasks under a parallel allocation of tasks to slave processes.

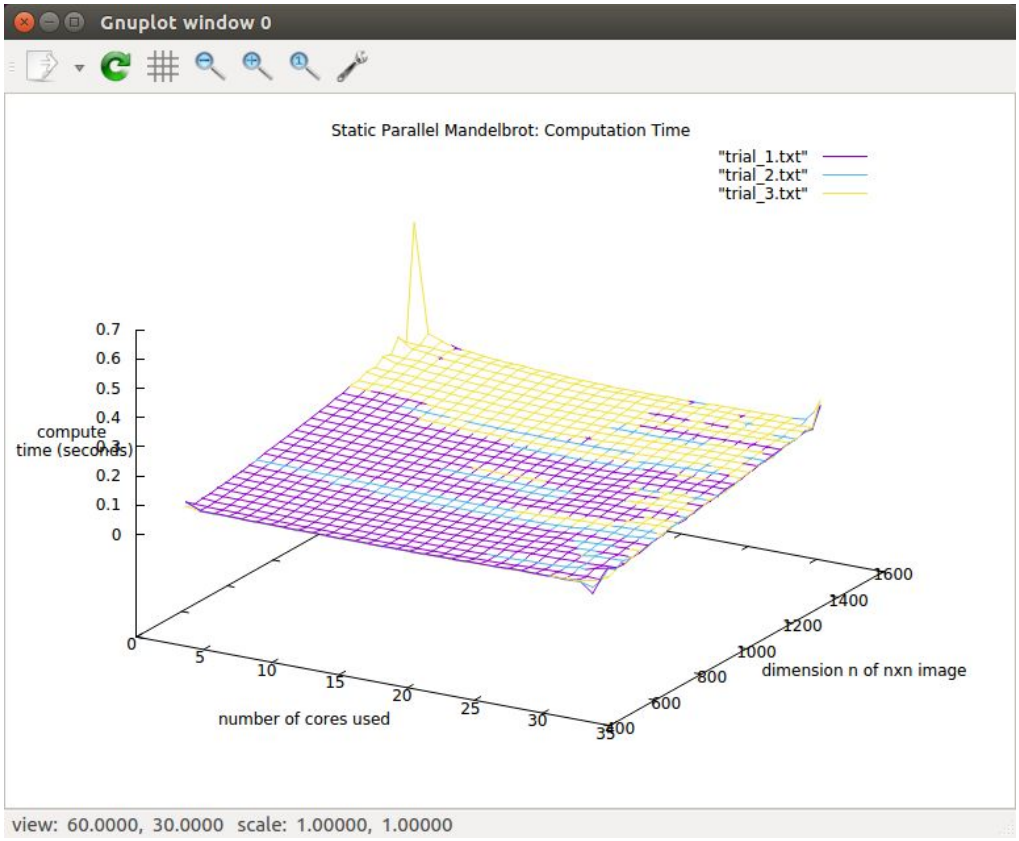
Data and Analysis

*note: for the parallel implementation, the master process does no computational work.



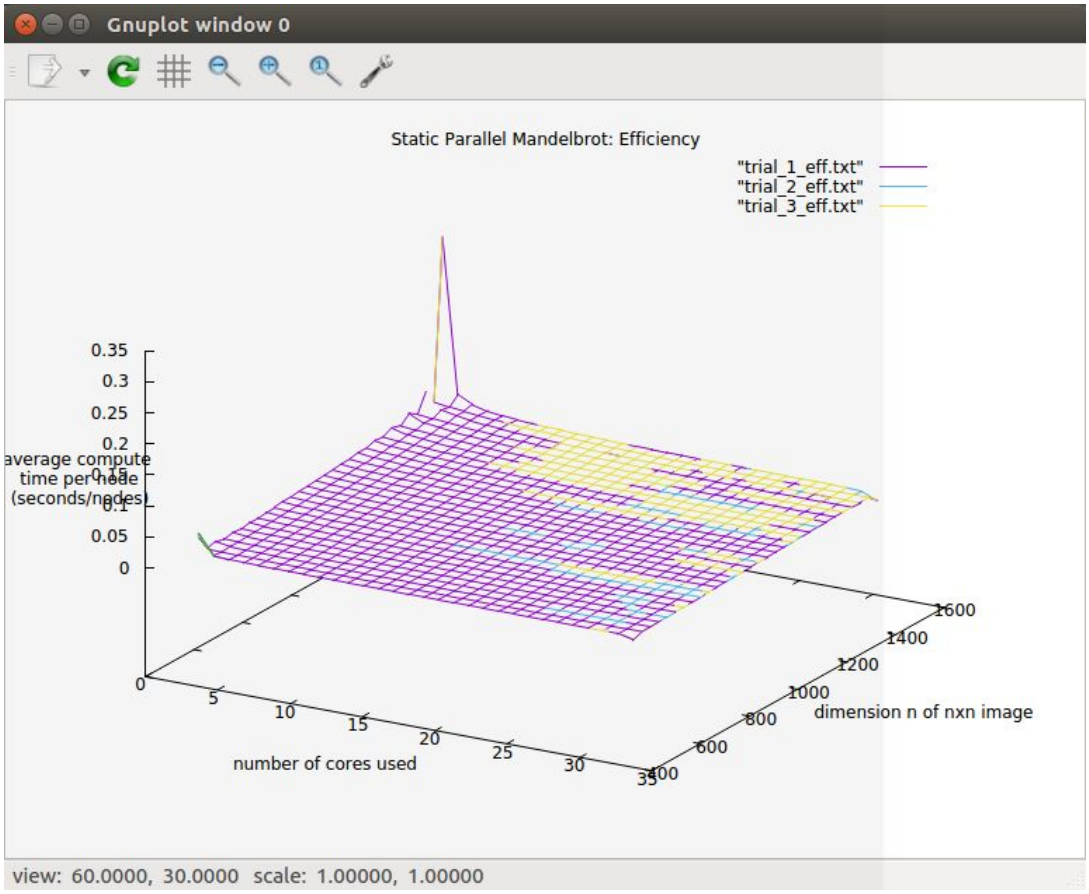
Graph 1:

The graph displays the execution time of the sequential mandelbrot program across different sized images. As can be observed by the graph, the computation increases quadratically with the quadratic increase in pixels.



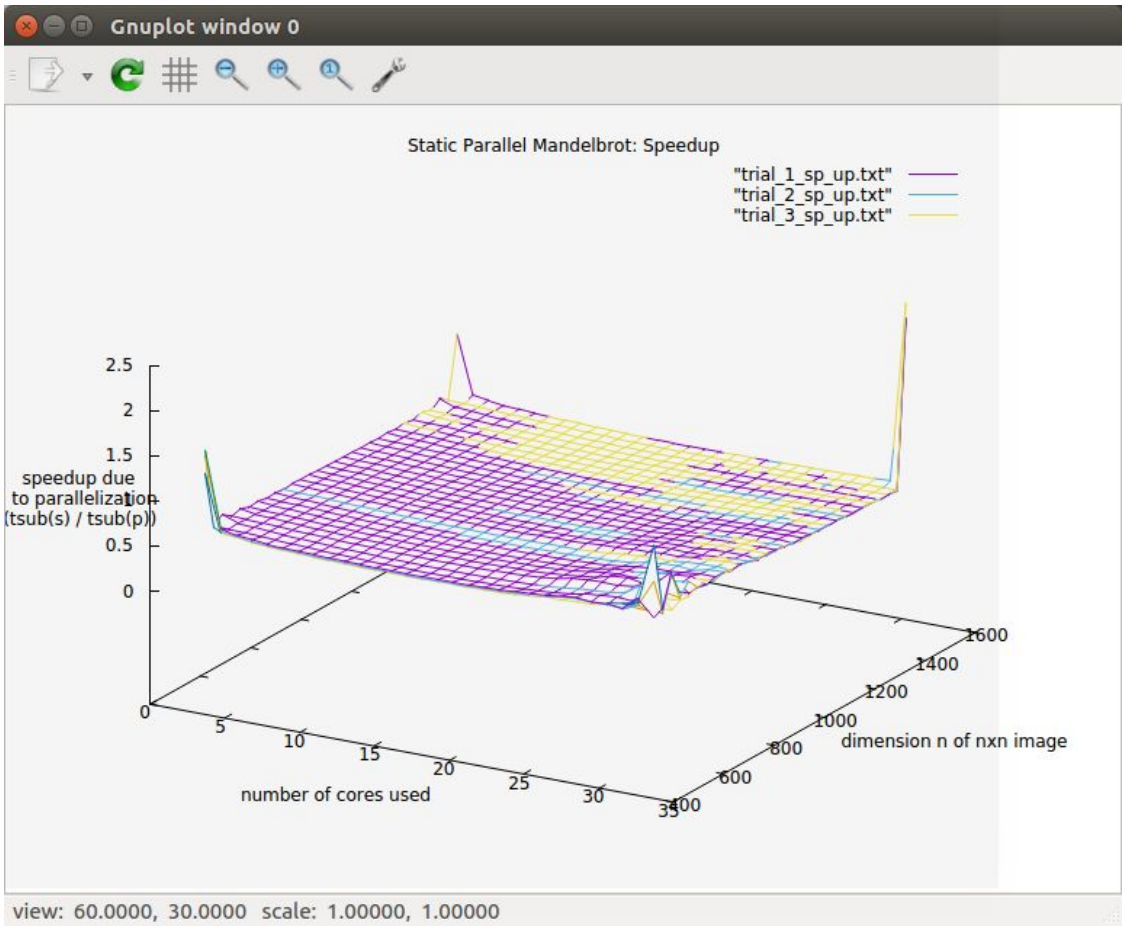
Graph 2:

The graph displays the computation time of the static parallel mandelbrot program against the number of cores used and resolution of the image. The graph displays an increase in computation time as the resolution size of the image increases, as well as a decrease in computation time with an increase in cores used.



Graph 3:

The graph displays the efficiency of the static parallel computation of the mandelbrot image. Ideally, the lower the compute time is per node (core), the more efficient cores are at making computations



Graph 4:

The graph displays the speedup of the static parallel mandelbrot program with respect to the number of cores used and image resolution. Speedup is calculated by dividing the sequential time obtained from the average of the times in Graph 1 by the parallel time obtained from Graph 2.

Examining the computation graph, the increase in computation time occurs along the expected directions of each axis. Computation time of the static parallel program increases rapidly when working only on a single core. However, the computation time increases slowly when working on 32 cores. The division of work between different cores can be used to explain the reduction in computation time. A noticeable feature of the computation time graph is the sudden jump where cores used is at minimum and the image resolution is maximized. The best explanation for this would be the increase in work for the single core to handle, as well as the communication time between the slave and master which adds to the total computation time.

The remaining graphs, efficiency and speedup, show concise results describing the static parallel program. The efficiency graph expresses a decrease in computation time as the number of nodes increase. This is to be expected as the amount of work each node performs is reduced as the number of nodes is increased over a constant image resolution. As for the speedup, the program has demonstrated sublinear speedup consistently when the increase in cores should severely decrease the computation time. A likely explanation for this is that the computation time is dominated by the communication time, potentially causing the program to run slower than the sequential. The speedup is closer to linear around two cores, which is expected as communication time is at a minimum. The speedup then lowers drastically with each computing box added to the computation, indicating an increase in communication time. Then, near the maximum amount of cores (32 cores), speedup sporadically increases. A good explanation for this phenomenon is a switch in which computation time possibly synchronizes with communication time, the master node receiving back work at the same rate it is being completed.