Samantha Rack
CSE 30341
Project 3 Report

Overview

The experiments with the Mendelbrot Set computation are done to determine the affect of using multiple processes or threads on the execution time of a computationally intense program. Since the programs can be run parallelly to produce the output, the use of more than one process or thread can improve the running time of the program.

For the experiments, the programs were run on a Linux machine in the Fitzpatrick cluster. All output files were written to the /tmp directory.

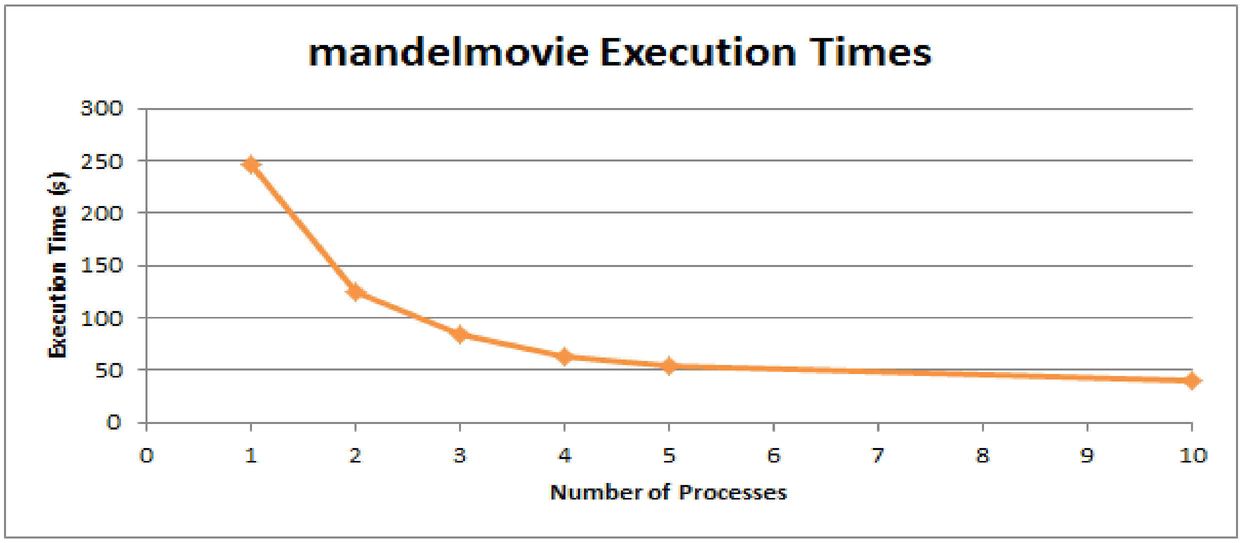
Multiple Processes - mandelmovie

For these experiments, ‘time ./mandelmovie <numProcesses>’ was run at the command line. The configuration of parameters for the Mandelbrot image was as follows:

x = .3855	y = .15
min s = 0.0000000001	m = 1500
W = 1360	H = 1360

Results:

Processes	1	2	3	4	5	10
Time (s)	247.16	124.90	84.05	63.59	54.78	39.90



The curve of the graph of number of processes used vs. execution time of mandelmovie is decreasing and concave up. The execution time decreases as the number of processes increases because of the concurrency. There are more processes working on producing the 50 mandel images, so if one of the processes requires I/O or has a different interruption, another of the processes can use the CPU. Additionally, the machine likely has multiple cores on which each of the processes can run simultaneously. However, as the number of processes increases, there are diminishing returns. This is due to the increased number of context switches and the cost of creating new processes which offset the benefits of the increasing number of processes.

The optimal number of processes is 3 or 4. Though the execution time reduces for each of the experimental trials with increasing number of processes, there are diminishing returns.

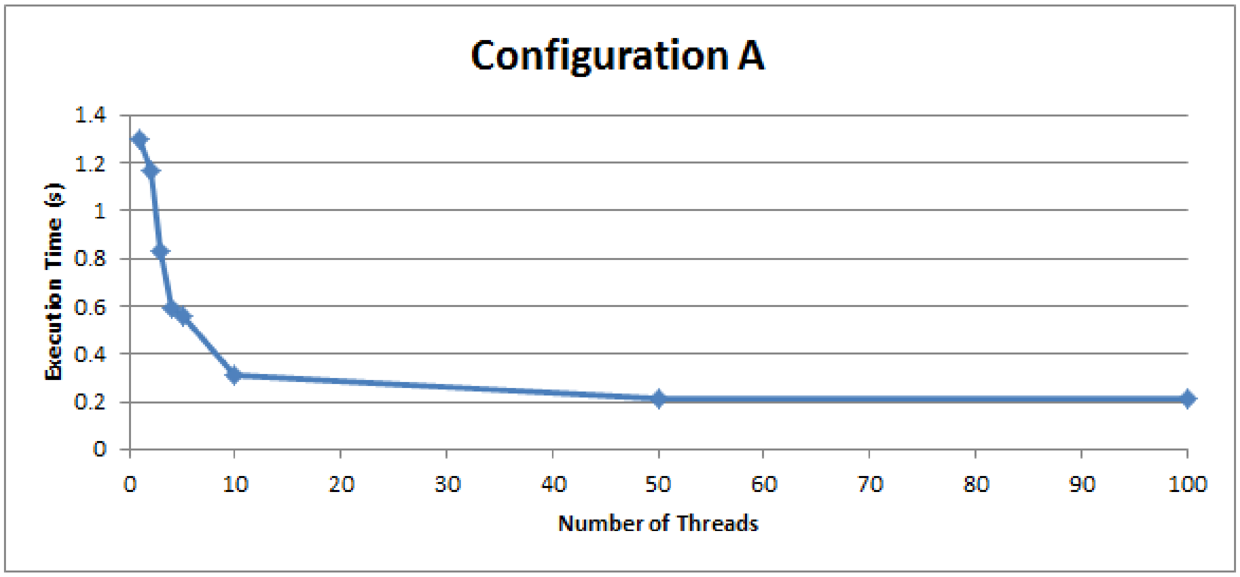
It is possible to have too many processes. As mentioned above, the execution time for the program decreases as the number of processes increases, but this is at the cost of context switches and overhead from process creation. With a larger number of processes, the execution time for this program may decrease (at least until 10 processes), but the CPU is spending more cycles completing context switches and other processes on the machine may be hindered. By choosing 3 or 4 processes for this program, there is a large reduction in CPU time, but there is still a balance with how much work the CPU will have with context switches and process creation.

Multiple Threads - mandel

For these experiments, ‘time ./mandel <given configuration> -n <numThreads>’ was run at the command line for configuration A and B.

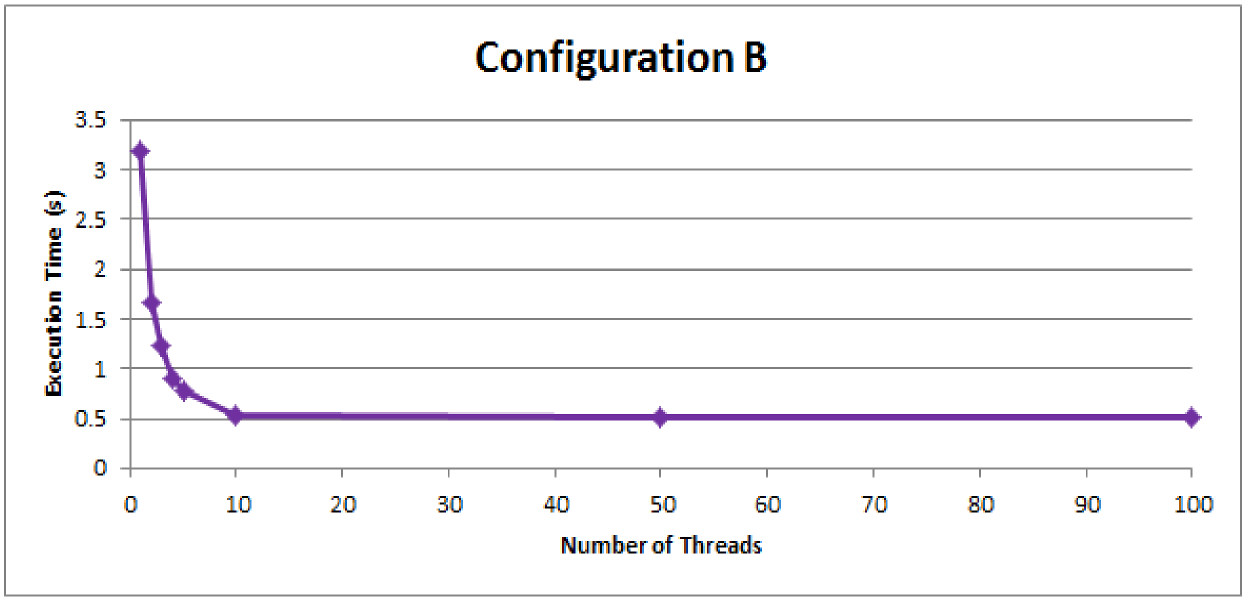
Configuration A:

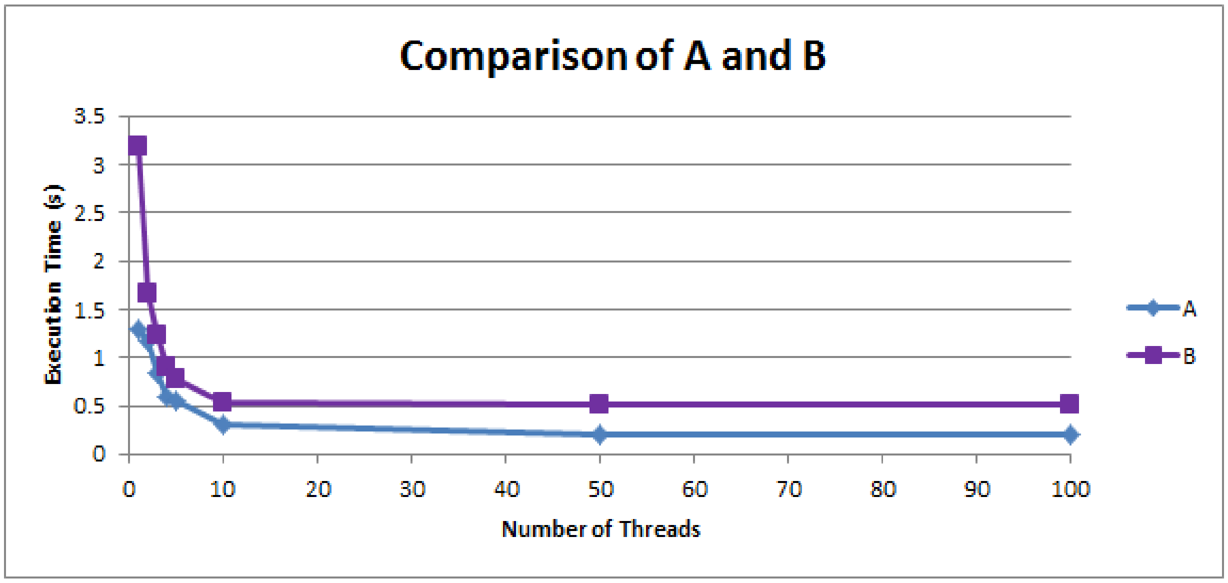
Threads	1	2	3	4	5	10	50	100
Time (s)	1.30	1.17	0.83	0.59	0.56	0.31	0.21	0.21



Configuration B:

Threads	1	2	3	4	5	10	50	100
Time (s)	3.18	1.66	1.23	0.91	0.77	0.54	0.51	0.51





The two curves of the number of threads used vs. the execution time of the mandel program demonstrate the same diminishing returns found with increasing the number of processes. However, as will be discussed below, the curve of Configuration B shows a more rapid decrease in execution time initially than Configuration A.

The optimal number of threads for Configuration A is approximately 10. Though the amount by which the execution time decreases is smaller between 5 and 10, it is still significant (45% decrease in time, so an average of 9% per thread added if it were a linear change). Looking beyond 10 threads, though, it is evident that there is little benefit from creating more threads to execute the program.

The optimal number of threads for Configuration B is between 5 and 10. At this point in the curve, diminishing returns begins to take effect, and the amount by which execution time decreases is not enough to increase the thread count.

Configuration B of mandel has a much more drastic decrease in execution time as the number of threads increases from 1 than Configuration A has. The curves both eventually approach a limit and demonstrate diminishing returns with 50 and 100 threads. However, the greater benefit displayed initially by Configuration B (10% decrease in A from 1 to 2 threads vs. 92% decrease in B) is likely due to the increased amount of computation in Configuration B. When one additional thread is added, then the splitting of work between the two threads greatly improves the execution time. With Configuration A, there is still an improvement because the computation is split, but since it is not as complex, the context switches between threads and cost of thread creation decreases the effect of two threads working on the image.