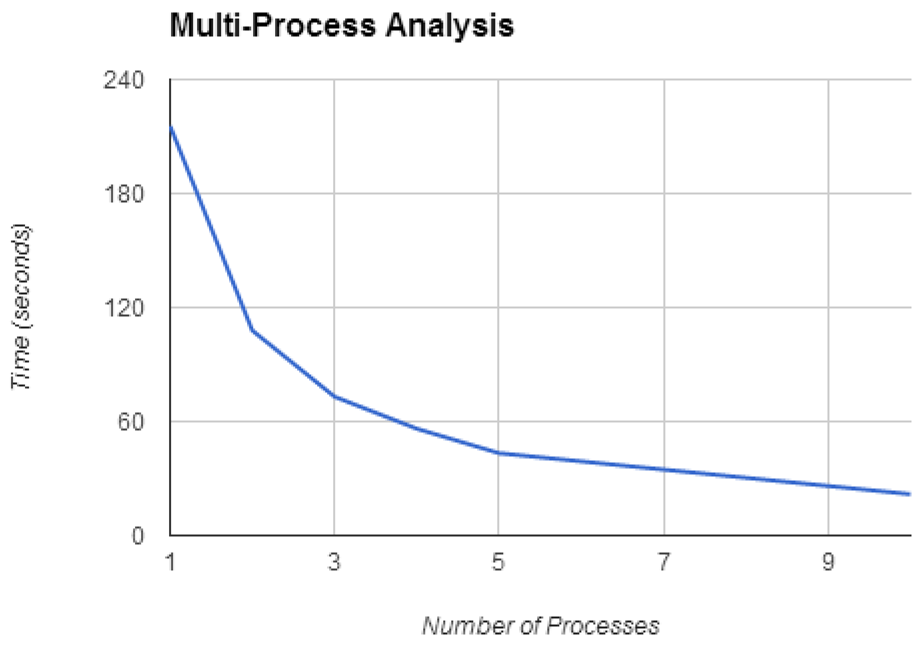
Nicholas LaRosa
CSE 30341 Project 3
Evaluation Report

All tests were run on student01.cse.nd.edu. My mandel modification program was unable to run more than 25 threads without experiencing segmentation faults; as mentioned by TA Lu, this is likely caused by limits on thread processes while SSH-ing. However, the general trend of thread number usage is still clear from the available data gathered. All measurements were made with UNIX time.

mandelmovie:

Results:

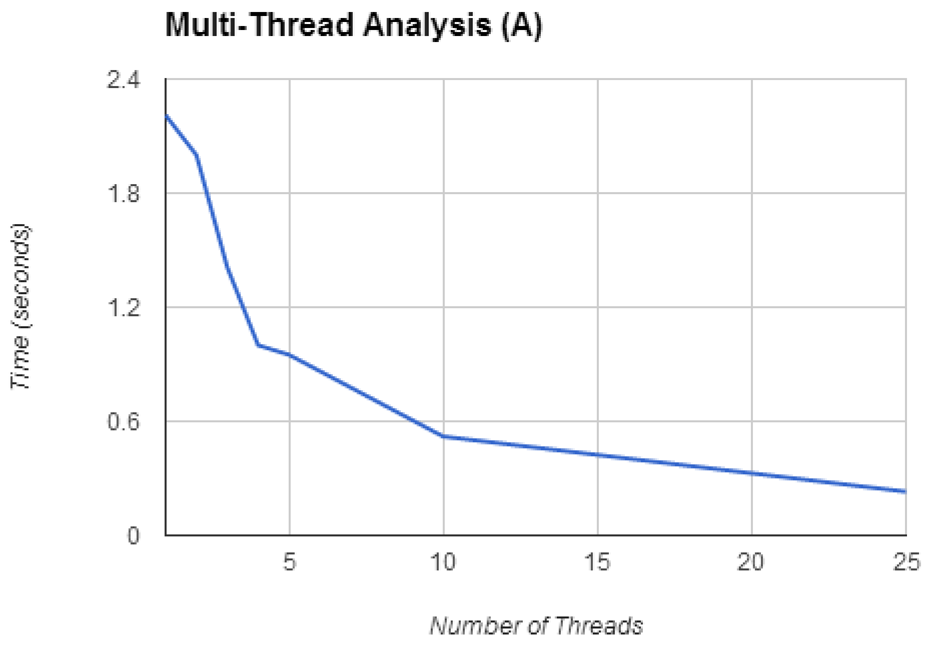
```
time mandelmovie 1 - 215.173u 0.164s 3:35.51 99.9%   0+0k 0+165200io 0pf+0w, 215.51
time mandelmovie 2 - 215.269u 0.278s 1:47.87 199.8%  0+0k 0+165600io 0pf+0w, 107.87
time mandelmovie 3 - 214.673u 0.263s 1:13.03 294.3%  0+0k 8+165600io 0pf+0w, 73.03
time mandelmovie 4 - 215.259u 0.273s 0:56.18 383.6%  0+0k 0+165600io 0pf+0w, 56.18
time mandelmovie 5 - 215.476u 0.291s 0:43.30 498.2%  0+0k 0+165600io 0pf+0w, 43.30
time mandelmovie 10 - 215.008u 0.511s 0:21.74 991.3%  0+0k 8+165600io 0pf+0w, 21.74
```



As seen from the multi-process analysis graph, the execution time of mandelmovie decreases as the number of processes increases. However, the rate of execution time decrease actually decreases, meaning that as the process count increases, the CPU spends more and more time scheduling the separate processes rather than creating an image. It is thus definitely possible to have too many processes, in which case the CPU has to switch between too many ongoing processes and thus makes little actual progress. As the rate of performance improvement is nearly zero at ten processes, it is likely that the optimal number of processes in actually ten.

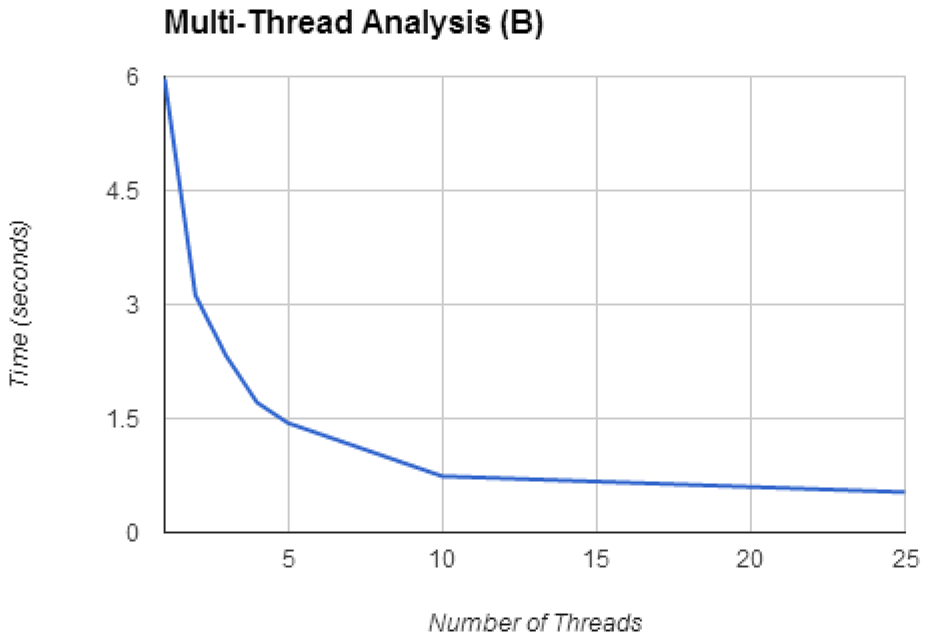
mandel (A):

```
mandel -x -.5 -y .5 -s 1 -m 2000 -o /tmp/mandel.bmp

-n 1 - 2.207u 0.000s 0:02.21 99.5%     0+0k 0+1480io 0pf+0w
-n 2 - 2.213u 0.001s 0:02.00 110.5%    0+0k 0+1480io 0pf+0w
-n 3 - 2.211u 0.004s 0:01.41 156.7%    0+0k 0+1480io 0pf+0w
-n 4 - 2.217u 0.001s 0:01.00 221.0%    0+0k 0+1480io 0pf+0w
-n 5 - 2.215u 0.000s 0:00.95 232.6%    0+0k 0+1480io 0pf+0w
-n 10 - 2.214u 0.000s 0:00.52 425.0%    0+0k 0+1480io 0pf+0w
-n 25 - 2.167u 0.000s 0:00.23 939.1%    0+0k 0+1480io 0pf+0w
```



mandel (B):

```
mandel -x 0.2869325 -y 0.0142905 -s .000001 -W 1024 -H 1024 -m 1000 -o /tmp/mandel.bmp

-n 1 - 5.967u 0.005s 0:05.97 99.8%     0+0k 0+8216io 0pf+0w
-n 2 - 5.982u 0.005s 0:03.12 191.6%    0+0k 0+8216io 0pf+0w
-n 3 - 5.980u 0.006s 0:02.32 257.7%    0+0k 0+8216io 0pf+0w
-n 4 - 5.973u 0.006s 0:01.71 349.1%    0+0k 0+8216io 0pf+0w
-n 5 - 5.954u 0.008s 0:01.44 413.1%    0+0k 0+8216io 0pf+0w
-n 10 - 5.949u 0.006s 0:00.74 802.7%    0+0k 0+8216io 0pf+0w
-n 25 - 5.210u 0.007s 0:00.53 983.0%    0+0k 0+8216io 0pf+0w
```



As seen from the multi-thread analysis graphs, the execution time of mandel decreases as the number of threads increases. However, the rate of execution time decrease actually decreases, meaning that as the thread count increases, the CPU spends more and more time scheduling threads rather than computing the image. More apparent in the part B analysis is the fact that 25 threads may be near the optimal number of threads for this program; this can be seen because as the thread number increases to 25, there begins to be nearly zero improvement in performance. Between parts A and B, there is only a slight difference; with part A, the rate of time decrease is more constant than in part B, likely meaning that the thread scheduling is somewhat less computationally expensive for this particular Mandel image due to the nature of the argument setup.