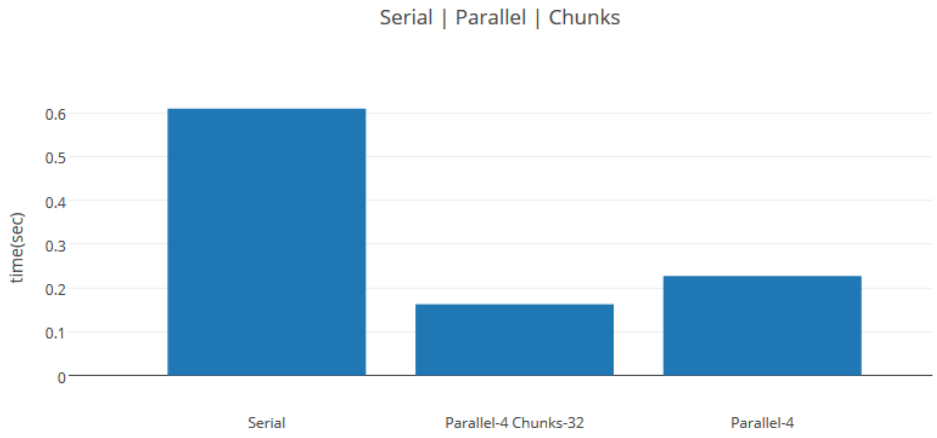
Christopher Almajose
calmajos@ucsd.edu

James Keophavone
jkeophav@ucsd.edu

PA1

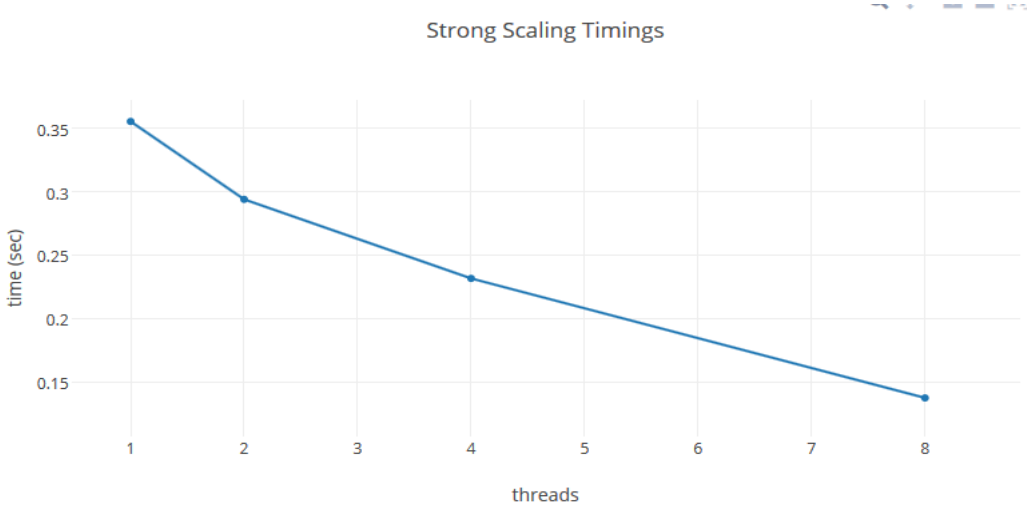
3.2



time(sec)	x
0.609753	Serial
0.162677	Parallel-4 Chunks-32
0.227224	Parallel-4

Link: https://plot.ly/~calmajos/9/serial-parallel-chunks/

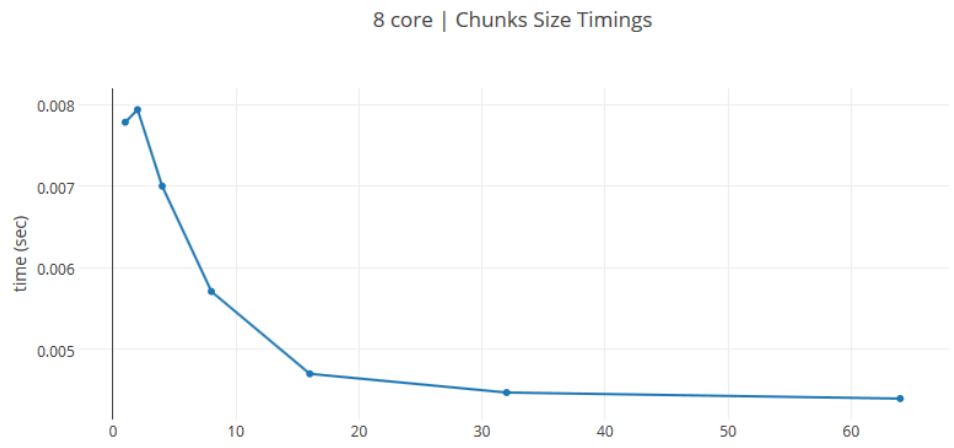
3.3.3



time (sec)	threads
0.355326	1
0.293894	2
0.231577	4
0.137356	8

Link: https://plot.ly/~calmajos/27/strong-scaling-timings/

3.3.5

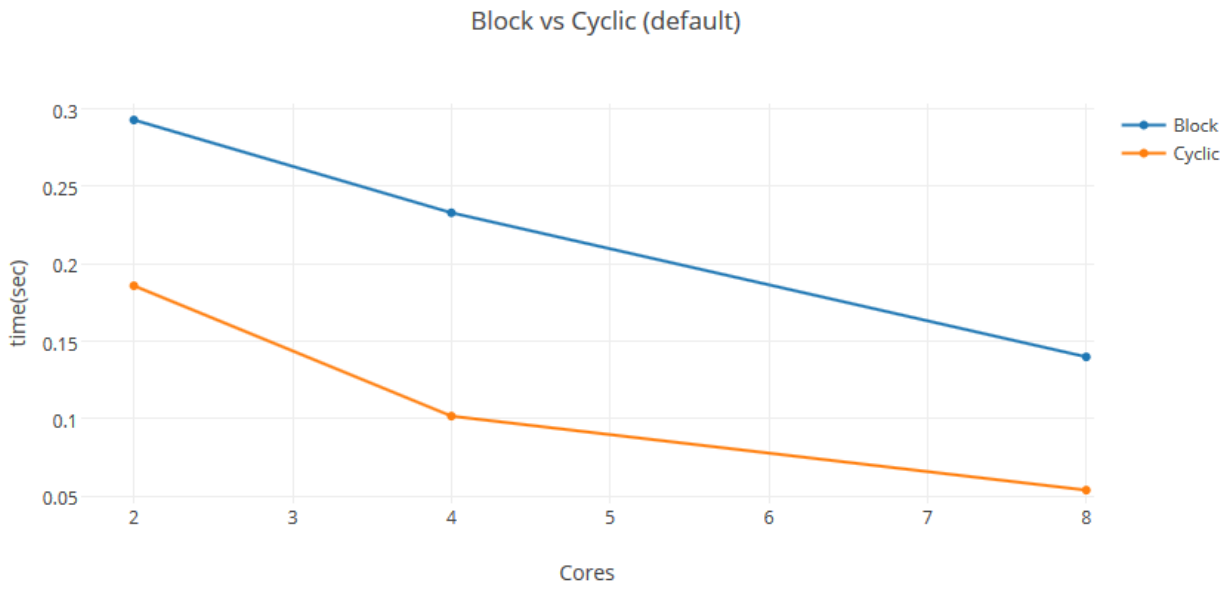


time (sec)	chunks
0.007786	1
0.007939	2
0.007001	4
0.005709	8
0.004699	16
0.004469	32
0.004395	64

Link: https://plot.ly/~calmajos/22/_8-core-chunks-size-timings/

Explaination:
Calculating the Mandelbrot point for specific region can be more intensive than others. By increasing the chunk sizes to a reasonable amount also increases the likelihood that all the cores can almost do equal amounts of work, thus decreasing the program's runtime due to other the absence of waiting for all other cores to finish.

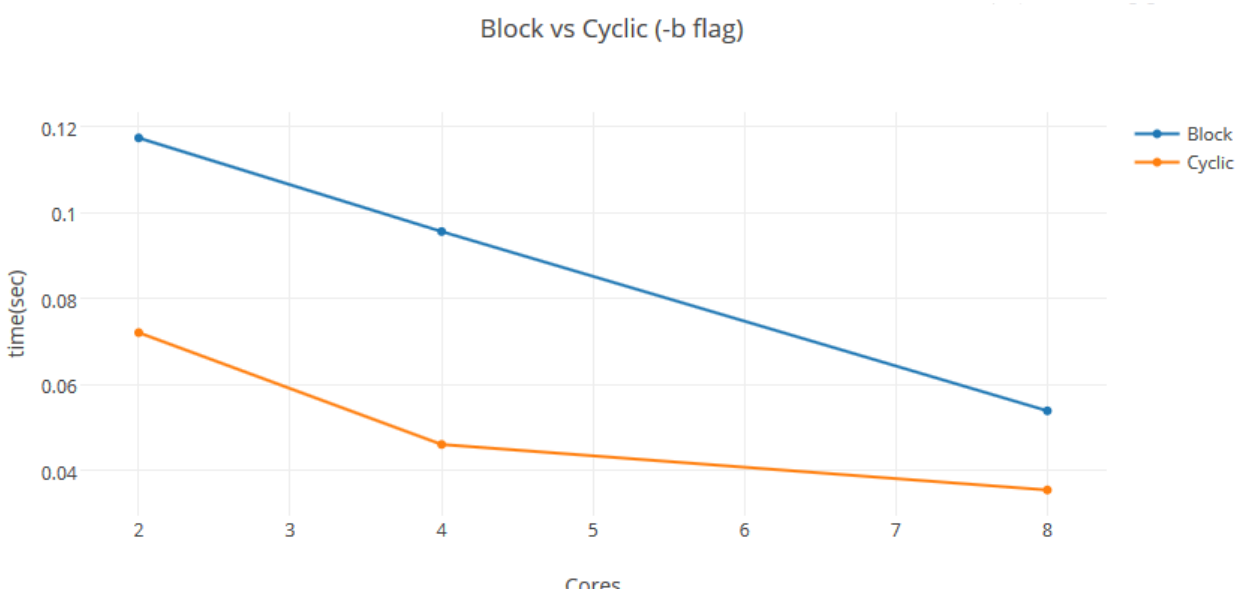
3.3.6



Block	Cores	Cyclic
0.293029	2	0.185869
0.233053	4	0.10165
0.139918	8	0.053797

Link: https://plot.ly/~calmajos/24/block-vs-cyclic-default/

3.3.7



Block	Cores	Cyclic
0.117465	2	0.072129
0.095676	4	0.046093
0.05393	8	0.03549

Link: https://plot.ly/~calmajos/25/block-vs-cyclic-b-flag/

Results:
Though the tests show that that the times are faster due to less computations of the Mandelbrot points, the graphs and tables are almost identical in the context of timing, based on the number of cores used, regardless of the decreased size of the bounding box.

Citation:
Python was used to plot the data recieved from the timings of the program in conjunction with plot.ly API for python to plot the datapoints of our data. Graphs are available for view from in courtesy of plot.ly from the given links. The python scripts to parse the data was written by the students of the report.