**Christopher Lang**

**STA9794 Spring 2017, Assignment/Project A**

The scrub program has been through many iterations and revisions in order to better optimize the performance of the program

There were two types of optimizations made; switching to Python 2.7 (it was originally tested on Python 2.6) and code optimizations that greatly enhanced the speed of the program

Within the context of MPI, there were serious issues of the first implementation. Originally, the root process would actually read chunks of the data and send them as messages to the other process. This caused odd hanging of the program. To get around this, without using MPI IO, each process would read the file on their own. The root process is the one who decides which line number each process would start and stop at. To facilitate this, a Rowread object was created, designed to iterate through the file for each process, starting at a particular line number. Hence, to avoid re-seeking line numbers as it changes, the root process ensures that for any given process, the chunks are in sequence within the file e.g. for process 1, read rows 1 – 1000, then 1001-2000, etc.

One of the biggest noise detection implementation that saw big speedups was the use of regular expression to detect properly formatted rows. This provided a 2x speedup from the old method, which actually took the row as a string and split it into a tuple, turning prices and units traded into number, etc. That proved all unnecessary, a well-designed regular expression would do the same in much less time

From this point on, it was simple code cleanup. I felt towards the end that I optimized the scrubbing process as much as I could. The remaining pain point is, is there a more effective way (aka fast) to read the file? For example, would reading by bytes be faster in order to have Python seek to the right line number to start from? These are questions the interesting questions

Performance seems scalable. Using Synth.py, I created a roughly 9.4GB data file to test on Penzias. It had roughly 150+ million rows (I think 187 million), which the program completed in about 4.5 minutes