Holly Straley
straleyh@oregonstate.edu
CS475 – Spring 2018
Project 7B

Autocorrelation using CPU OpenMP, CPU SIMD, and GPU OpenCL

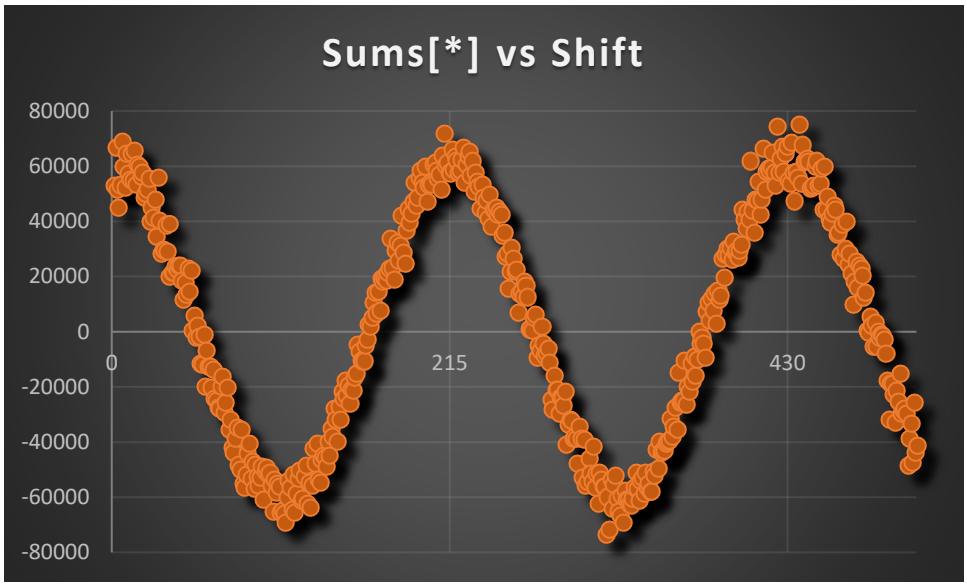
1. Platform

I ran this program on the flip and rabbit (as noted) servers.

2. Performance Data

2.1 Sums[*] vs Shift

2.1.1 Graph

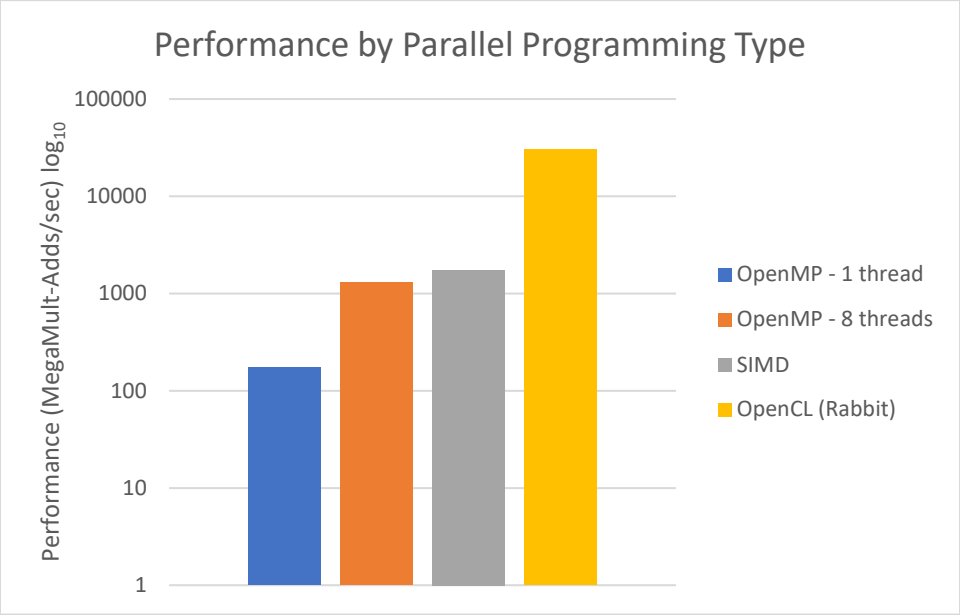


2.1.2 Graph Analysis

The graph for Sums[*] vs Shift has the x-axis label altered as to clearly show the sine-wave period which is approximately 215.

2.2 Comparison of different techniques

2.2.1 Graph



2.2.2 Graph Analysis

The above bar graph is comparing average performance values for each parallel programming technique used for this assignment. The blue bar is OpenMP using one thread and is used as the baseline since using one thread means that no parallelism took place. The blue bar is the slowest. The orange bar, which is the next fastest, shows average performance using OpenMP with eight threads which, as expected, has better performance than the baseline. The gray bar is the average performance using SIMD and, as expected, has better performance than the baseline and slightly better than OpenMP with eight threads. Finally, the yellow bar is the average performance using OpenCL on Rabbit which, also as expected, has MUCH better performance than all other techniques.

2.3 Performance Analysis

One reason that the OpenCL technique proved to be so much faster than the others is that is was ran using Rabbit. It was expected that it would be the fastest technique since it uses the GPU instead of CPU like all other techniques, but the extreme difference is explained by the server difference.

As I mentioned above, the use of GPU vs CPU another cause for the obvious difference in performance in the techniques used here. The CPU techniques and GPU techniques each have different conditions that will cause them to have optimal performance. The CPU parallel processes work well with irregular data structures, like linked lists, and irregular flow control. The GPU parallel processes, on the other hand, work will with programs like this one that have regular data structures and regular flow control.

As said in a lecture early this quarter, just about anyone can do parallel programming, the difficult part is doing it well. This experiment shows that it is important to understand the processes within the program to choose the most optimal parallel programming technique.