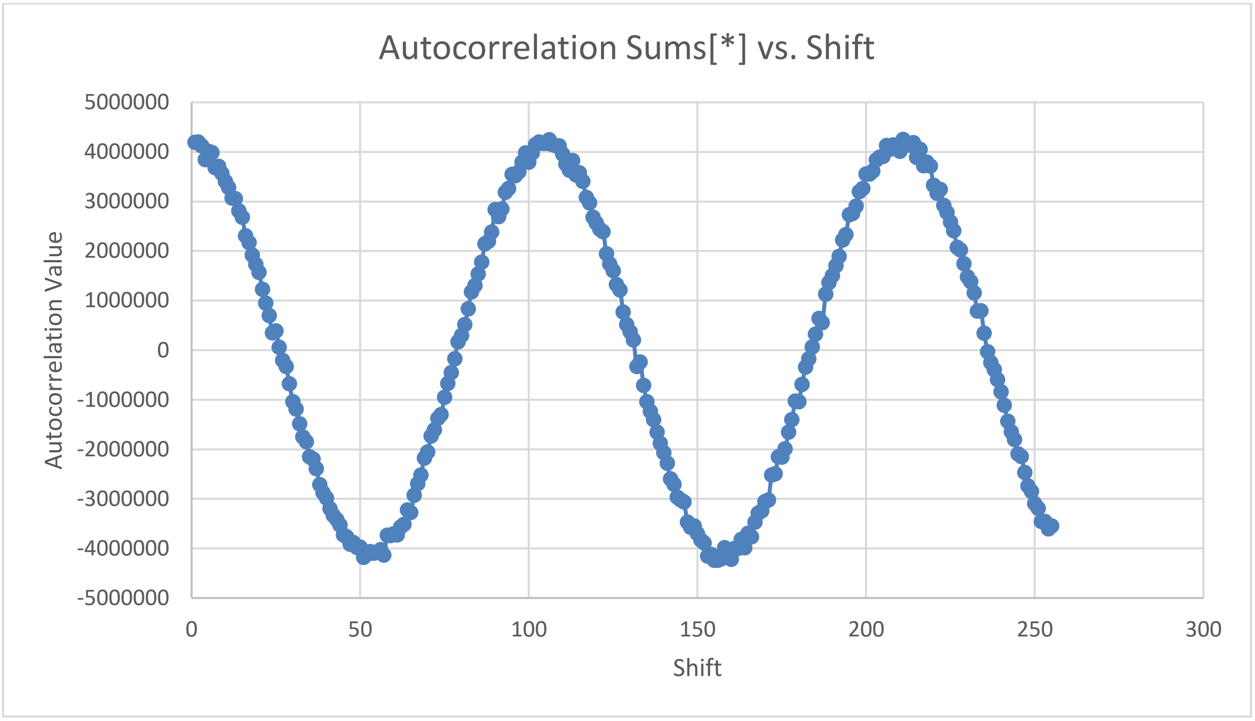
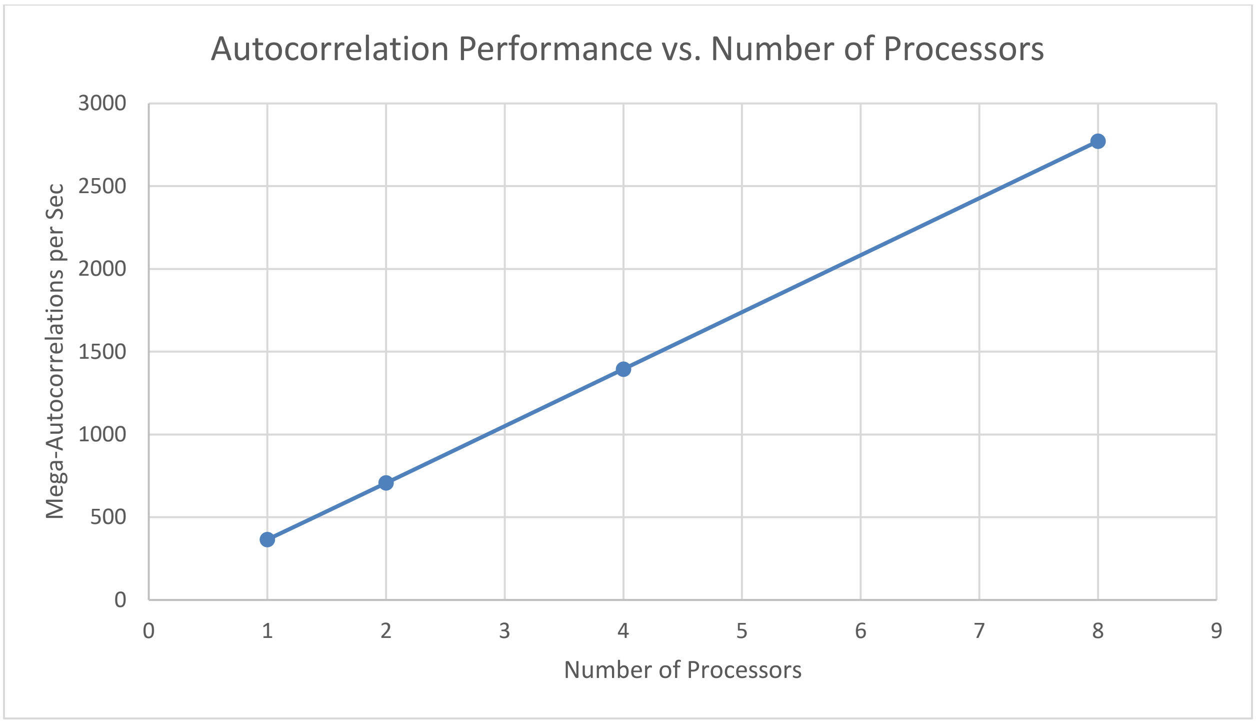
CS 475 – Parallel Programming
Project: Project #7b
Title: Autocorrelation using MPI
Name: Christopher Felt
Email: feltc@oregonstate.edu

1. Table Data and Graphs:





Cores	Mega-Autocorrelations per Sec.
1	365.11
2	706.63
4	1394.71
8	2771.69

2. Summary:

In our final project, 7b, we explored parallelism across multiple CPUs using Message Passing Interface (MPI). This method of parallelism harnesses the power of physically distinct CPUs via communication across a network, and is often used together with the other forms of parallelism we learned in this course: OpenMP, SIMD, and the GPU parallelisms including CUDA and OpenCL/OpenGL. In this project, however, the only parallelism we employed was MPI. Specifically, we used MPI to facilitate the autocorrelation of a large, noisy transmission signal dataset in order to find a hidden sine wave.

We used MPI's MPI_SEND and MPI_RECEIVE functions to divvy up the data from the transmission signal across multiple CPUs. We designated one CPU to oversee the operation and send out equal portions of the transmission signal data to all of the CPUs (keeping a portion for itself as well) – in effect, a scatter operation. The data portioned out to each CPU included a buffer (MAXSHIFTS) of 1024 additional elements from the array in order to perform a local autocorrelation. We did this to avoid dealing with an if statement for wrapping around to other portions of the array, which would require more communication between the CPUs and slow performance down. Once the local autocorrelation was completed, we performed the equivalent of a gather operation by having the boss CPU collect all of the local autocorrelation results from the various CPUs. Finally, we had the boss CPU add all of these values into a final Sums array before writing the Sums array from element 1 to 255 into a CSV file.

For my implementation of project 7b, I used OSU's DGX cluster and tested performance for 1, 2, 4, and 8 core runs. Based on the Sums vs. Shift graph in section 1 above, I found a secret sine wave in the raw signal data with peaks at Sums[106] and Sums[211], for a resulting period of 105.

My performance results were very straightforward (see the performance table and graph in section 1 above). As cores increase, performance increases at a linear rate, approximately doubling each time cores were doubled. The performance I saw makes sense given that the work done by each core was halved each time the number of cores were doubled. Essentially, as each core was added, the work was divided equally among all cores, resulting in a proportionate increase in performance. Additionally, given that the array we processed in this project was decently large at ~8 million elements, it makes sense that we were able to coax some good performance numbers out with MPI and not run into any overhead issues slowing performance down (something more common with smaller datasets). I was somewhat surprised that we saw such a smooth rate of increase despite the CPUs being physically separate, however – the DGX cluster must have a very fast network connecting the CPUs together.