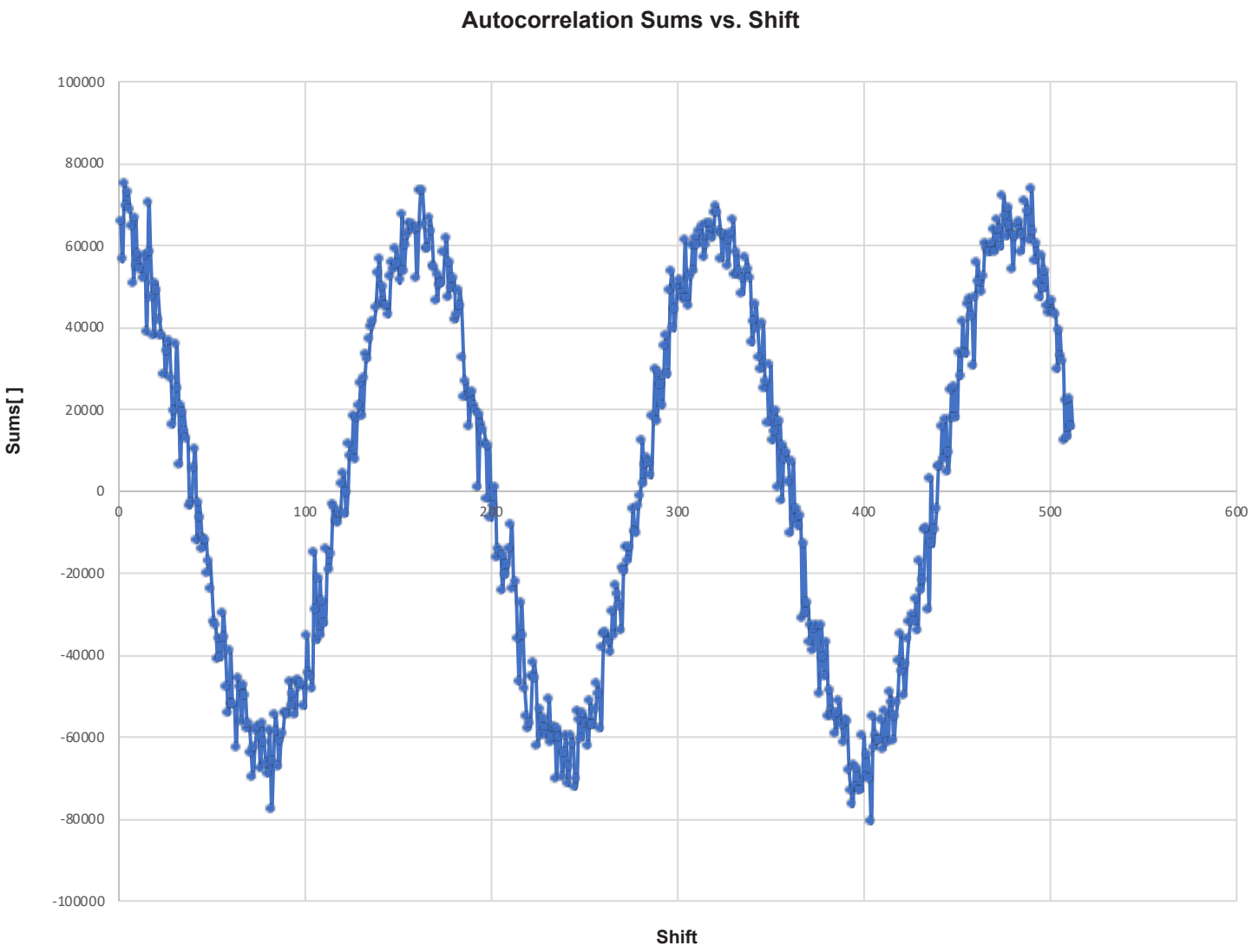
Kristin Schaefer

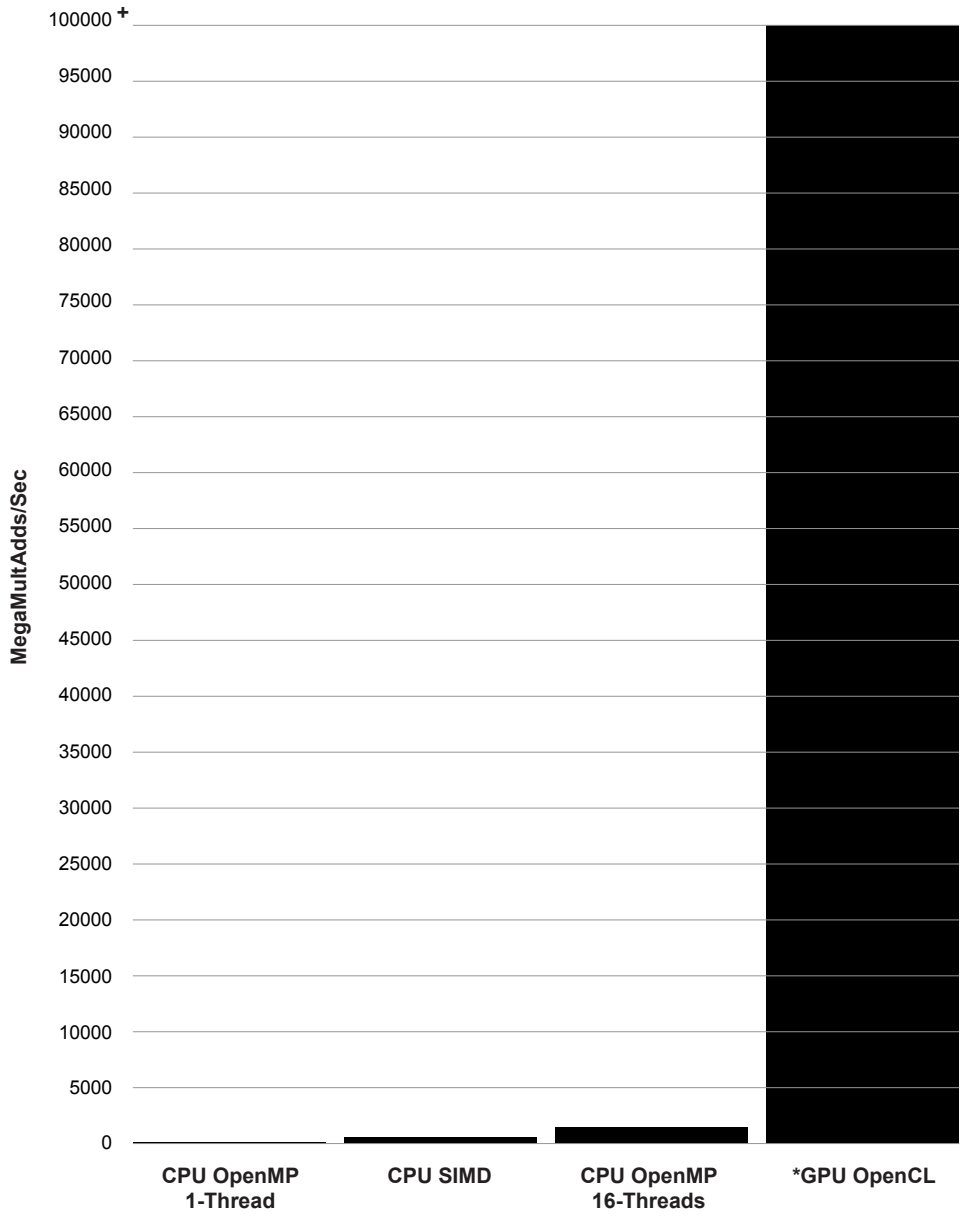
Project #7B : Autocorrelation Using CPU OpenMP, CPU SIMD, and GPU OpenCL



Kristin Schaefer

Project #7B : Autocorrelation Using CPU OpenMP, CPU SIMD, and GPU OpenCL

Autocorrelation Performance: CPU OpenMP 1-Thread & 16-Thread vs. CPU SIMD vs. GPU OpenCL



	OpenMP 1-Thread	SIMD	OpenMP 16-Threads	OpenCL
Performance (MegaMultAdds/Sec)	169.27	620.64	1498.16	798825.38

***GPU OpenCL: Note that the performance is reduced to 100,000+ MegaMultAdds/Second in order to preserve graph legibility.**

Kristin Schaefer

Project #7B : Autocorrelation Using CPU OpenMP, CPU SIMD, and GPU OpenCL

Machine:

FLIP & DGX

The hidden sine-wave period:

The hidden sine-wave period is approximately a shift of 160-165. The maximum (or peak) sum is approximately 72,000 and the minimum sum is approximately -72,000.

What patterns are you seeing in the performance bar chart? Which of the four tests runs fastest, next fastest, etc.?

By a little, or by a lot? Why do you think the performances work this way?

As expected, the lowest performance was CPU OpenMP with 1 thread, which had a performance of 169.27 MegaMultAdds/Second. It had the lowest performance because only 1 thread was used, so there was really no parallelization taking place. The second lowest performance was CPU SIMD with 620.64 MegaMultAdds/Second. It had the second lowest performance because compilers have not caught up (as discussed in the lectures in week #4), so the performance increase was less than four times CPU OpenMP with 1-Thread. The next fastest performance was CPU OpenMP with 16 threads, which had a performance of 1498.16 MegaMultAdds/Second. The fastest performance by a very high margin was GPU OpenCL with a local work size of 64 and a performance of 798,825.38 MegaMultAdds/Second. I was shocked to see such a high performance, and I confirmed with other classmates that they had similar results. I think the performance was very high because GPU data parallelism is at its best when there is a large data set and an operation such as array multiplication is being parallelized. Both cases were met in the code for this project, so consequently excellent performance results were observed.