Clinton Hawkes

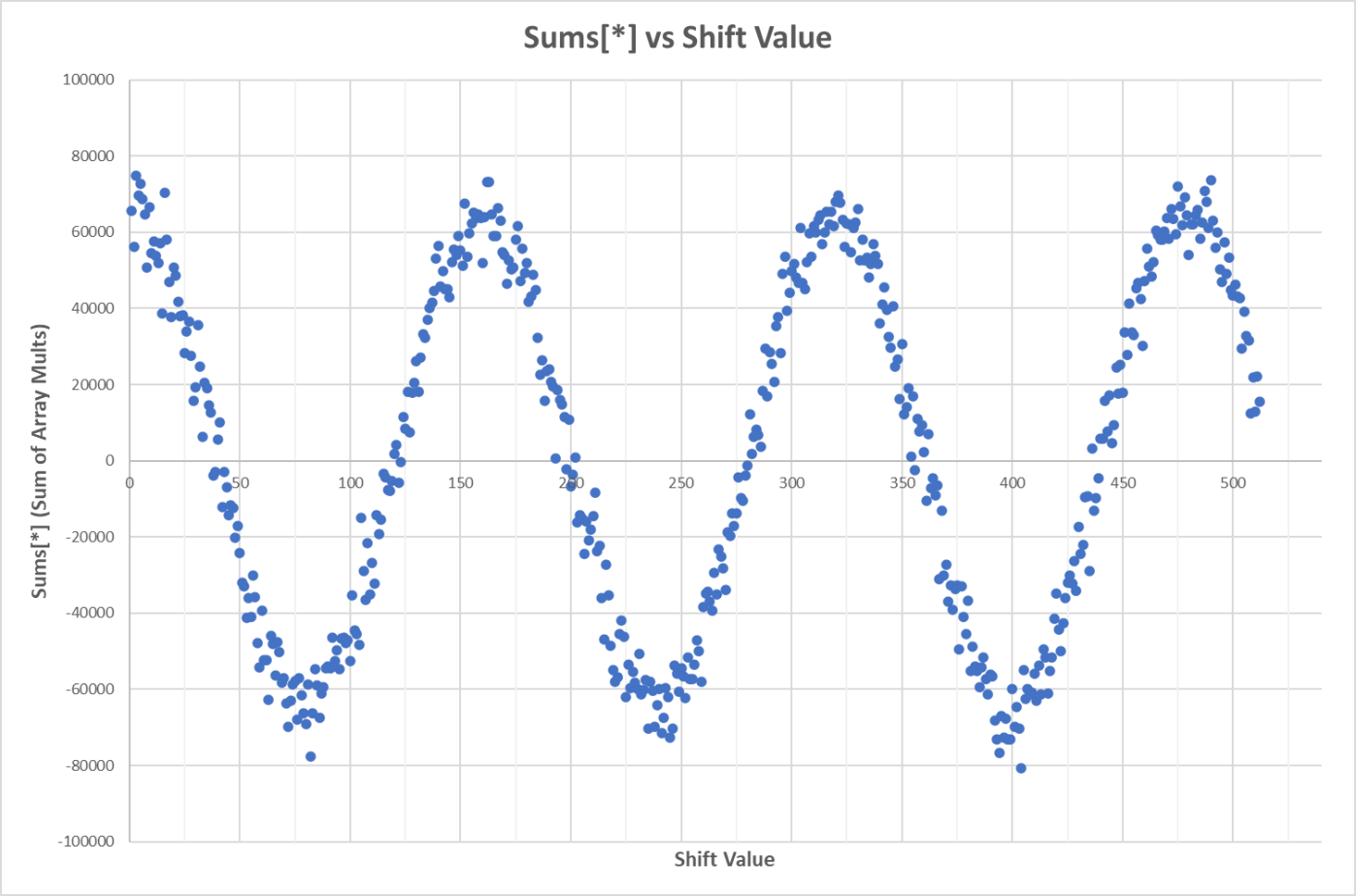
hawkesc@oregonstate.edu

CS475-400

Project #7b

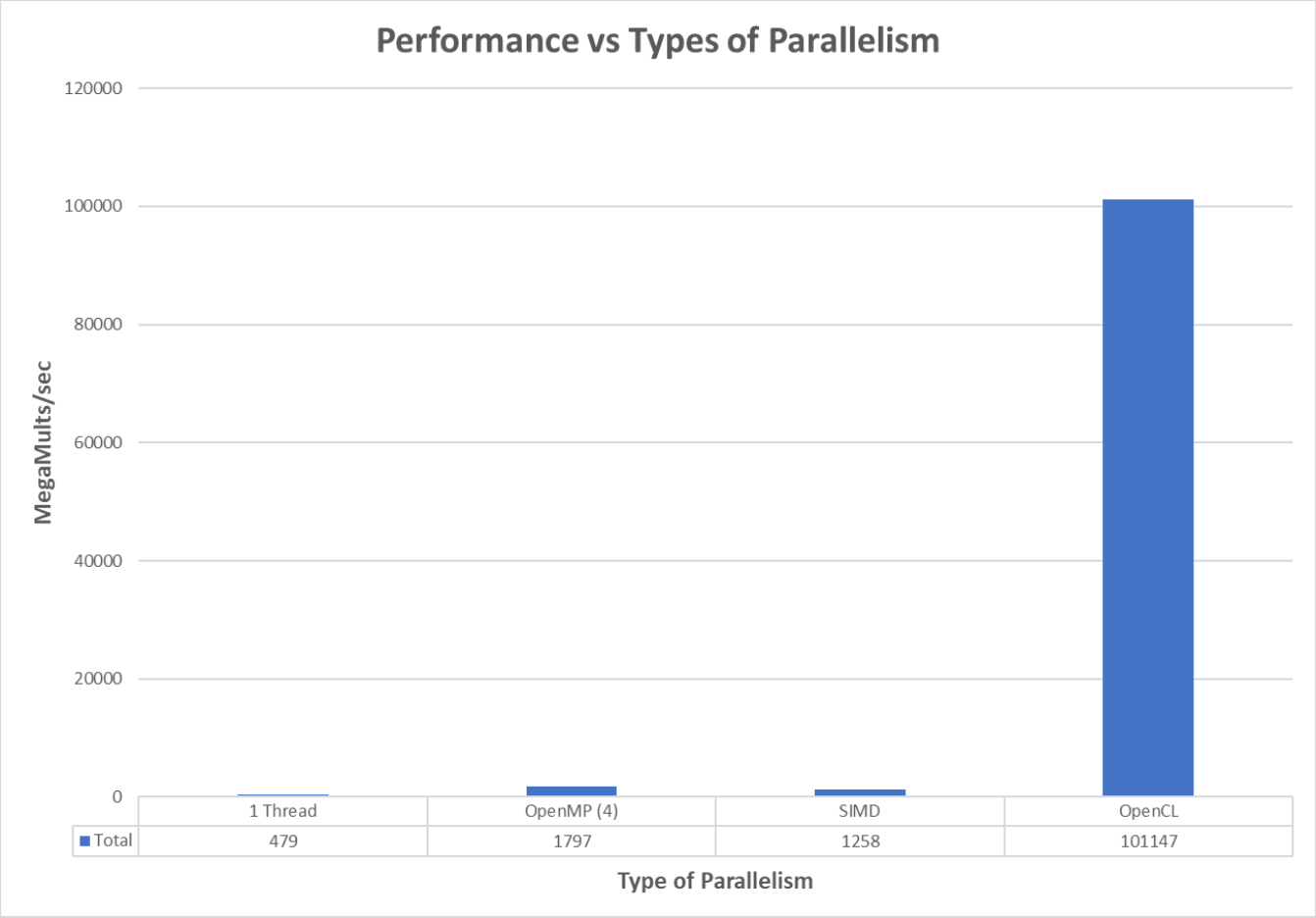
1. I ran this program on my personal laptop:
   Lenovo X1 Extreme
   Intel Core i7 (6 cores/12 threads)
   32GB ram
   Nvidia GTX 1650 Max Q
   Running Linux kernel 5.5.9

2. Here is the scatter plot of Sums[*] vs the shift value:



3. The period of the hidden sine wave appears to be 160. It is a little difficult to tell exactly what the period value is, but 160 is a good even number that fits the pattern in the plot.

4. Here is a graph comparing the performance values for the different types of parallelism:



The highest performing group in the bunch is the OpenCL. This is what I expected. The performance of OpenCL is 56x the performance seen by the next best option. This level of performance is what we saw in the other projects, and I don't think any non-GPU method can match it.

The remaining groups' performance values are dwarfed by those of the OpenCL, and it is somewhat difficult to see which one performs better when comparing them, so I just look at the table below the graph. The second highest performing group was the openMP program running 4 threads. SIMD came in third place among the parallelism types, and the group that did not utilize any form of parallelism came in last.

If you take a look at the graph above, you will see that the performance values of the three groups on the left do not appear to vary much. This is just because the huge gain in performance seen by using OpenCL throws off the scale. SIMD is almost 3x the performance of the no parallelism group and openMP using 4 threads is almost 4 times the performance of the no parallelism group. The bar graph above just doesn't represent this very well. (thanks OpenCL!)

5. When performing the types of calculations seen in this project, OpenCL (or CUDA) will most likely always outperform the standard CPU parallelism options. This is assuming you aren't comparing a small number of graphics cards to a super computer. GPU parallelism options are so quick, because they have many more threads at their disposal than a CPU does. My graphics card is nothing to brag about, and it has 1024 cores(threads) that it uses to perform calculations. You can see why the performance is so much greater for GPU parallelism than it is when using the CPU. This is why I was able to get a 56x performance increase over using only one thread on the CPU.

   After the OpenCL group, the next two runners up were pretty close in performance. Personally, I thought these two would be a little closer. I used four threads for the openMP program so it would closely match SIMD, but SIMD was not quite up to par. My thinking was that each SIMD instruction multiplies four floats, so using four threads with openMP should be very similar in performance. I forgot that I ran into this discrepancy in performance with the SIMD project earlier in the course. The lecture notes showed a performance gain of almost 4.0x by using SIMD, but I have only been able to get 2.6x performance gain.

   After brainstorming, my thought is that the openMP utilizes four independent cores to perform calculations while SIMD utilizes one single core for all calculations. For openMP, this means that once the cores are assigned a section of the dataset, they can perform the calculations without having concern about the other cores. If one of the cores is blocked for scheduling when using openMP, the other cores will still run and perform calculations. This is not the case for SIMD. If the one core running the SIMD calculations is blocked for scheduling, all calculations stop until the scheduler gives the thread control again. This could be part of the reason why SIMD performance is lower than openMP. I don't remember Prof. Bailey providing an explanation as to why I (and many other students) weren't getting the performance gain shown in the lecture notes, so I can't provide a definitive explanation.

   It is no surprise that the group using one thread and no SIMD to perform calculations was the lowest performer. All the other groups are performing multiple calculations concurrently, so it is easy to see why this group is our control group. The performance of this group gives us a good base value to compare performance increases among the other groups.