Before addressing the differences in performance between the 3 versions we have to address the limitations in floating point accuracy. Given the test cases 2^20,2^22,2^24 are a high amount of elements to calculate as vectors as float they might lead to inaccuracies in the final dotproduct sum. We implemented the 2 data types of each version in the specs that make use of the float and double data type. The float data type yields us with a lower accuracy however the execution time of the program would be faster. In Implementations where accuracy is prioritized the runtime would take longer.

Performance of the grid stride version with no shared memory was found to be faster than the base C++ version of the dot product program. There were no differences in performance when it came to changing the threads per block on the float version however when it came to the version that utilizes the double data type it was a lot slower due to how we implemented the atomic add function. The double data type implementation performance also varies as the threads per block goes up due to how atomicAdd was used however despite these findings the program was still found to be faster than the base C++ program when calculating a huge amount of vectors (2^20,2^22,2^24).

The 3rd version that utilizes a shared memory is significantly faster than the grid stride w/o shared memory and c++ version. This is due to the fact that shared memory was utilized and how shared memory is located on the chip causing the threads to cooperate better. The threads per block would lead to the program running a tiny bit slower however it's not that significant compared to how the double data type was implemented on the grid stride version. The runtimes were also found to be consistent on all block sizes and # of vectors.

References:

Using Shared Memory in CUDA C/C++ | NVIDIA Technical Blog. (2022, August 21). NVIDIA Technical Blog. https://developer.nvidia.com/blog/using-shared-memory-cuda-cc/#:~:text=Summary,mechanism%20for%20threads%20to%20cooperate.

What Every Computer Scientist Should Know About Floating-Point Arithmetic. (n.d.). https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

CUDA Floating Point. (n.d.). https://docs.nvidia.com/cuda/floating-point/index.html