Jason Gallagher

Steven Kravitsky

**CUDA Vector Dot Product**

This assignment was parallelizing the calculation of the dot product of two n-element vectors using CUDA acceleration. CUDA cores are Nvidia’s naming convention for the cores in their GPU, therefore CUDA acceleration is parallelizing something using an Nvidia GPU To use CUDA parallelization you have to do a kernel level call, as seen in Figure 1. For this assignment, we used one dimensional block(256, 1 , 1) and grid sizes(128, 1). A mutex had also been created, Figure 2, and allocated on the device, it is used for thread blocks to write its final dot product. The kernel implementation uses a shared memory approach and uses a stride variable to allow the thread to do multiple dot products by setting stride to be blockDim.x \* gridDim.x. The code used to do the dot product can be seen in Figure 3, it is broken into a while loop, a for loop, and an if statement. In the while loop each thread is performing the local dot product and moving based on its stride. The local sums are then all added to the sum vector. The stride is then used in the for loop to make sure that each threadblock has a single partial sum value in position 0 of stored memory. The if statement is used to add its thread block’s partial sum serially using locks.

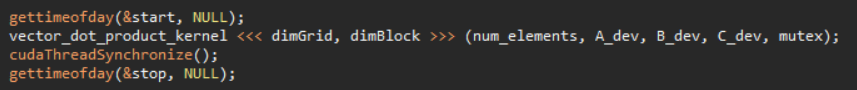


Figure : Calling the CUDA kernel

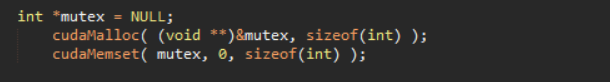


Figure : The mutex creation

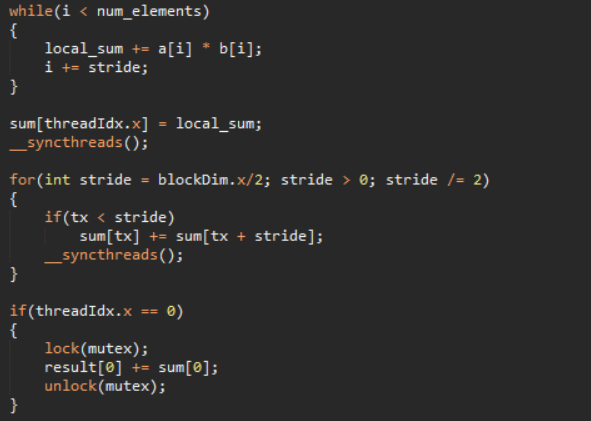


Figure : The shared memory implementation

The time taken to run the program can be seen in Table 1 while the speedup can be seen on Figure 4. The largest speedup attained was for the 10,000,000 element vector, with a speedup of 43x. This makes sense as a ten million element vector is a perfect use case for parallelization, since it can be chunked into much smaller pieces. The GPU implementation is actually slower than the CPU implementation once you start going below 100,000 elements as the overhead for the GPU becomes longer than just doing the code on the CPU.

Table : The time taken for each run in seconds and speedup

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **CPU (us)** | **GPU (us)** | **Speedup** |
| 100000 | 238 | 99 | 2.40404 |
| 1000000 | 1952 | 118 | 16.54237 |
| 10000000 | 17432 | 405 | 43.04198 |

Figure 4: The speedup achieved