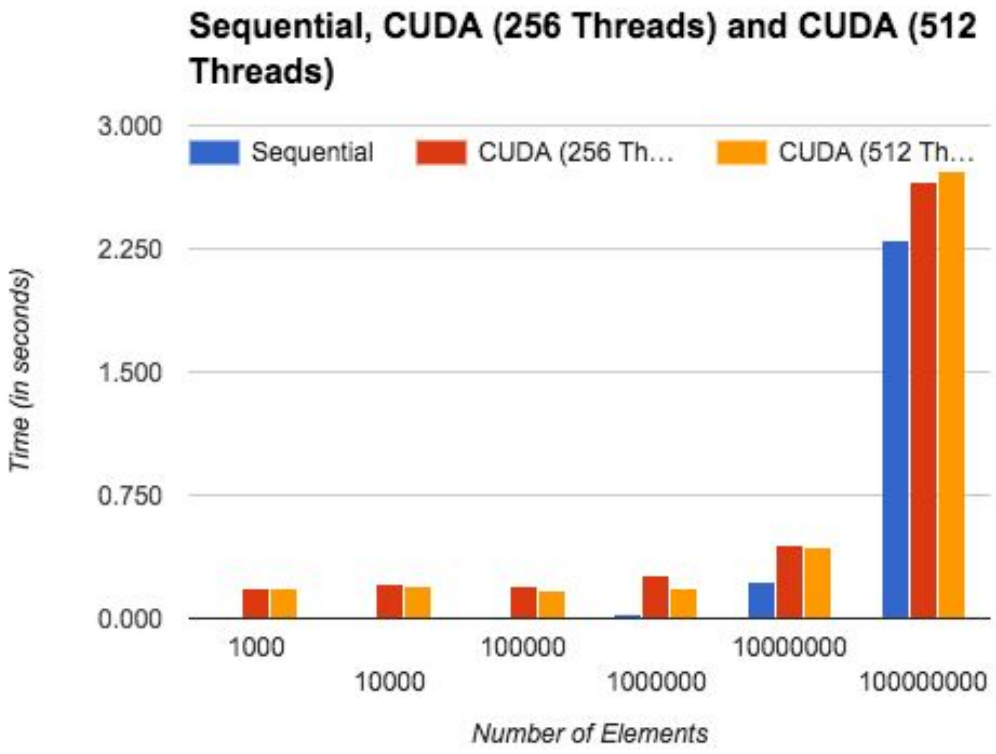
Adithep Narula
an1375

Machine: cuda5

command line to compile my code: nvcc -arch=compute_30 -code=sm_30 maxgpu_v2.cu -o maxgpu_v2

To run program: ./maxgpu_v2 1000000



Number of Elements	Sequential	CUDA (256 Threads)	CUDA (512 Threads)
1000	0.003	0.183	0.186
10000	0.002	0.208	0.192
100000	0.004	0.199	0.170
1000000	0.026	0.258	0.190
10000000	0.225	0.447	0.434
100000000	2.298	2.655	2.723

1. Choices of why I picked the block and grid sizes/dimension

The block size (threads per block) I selected was 256. There were a few considerations that I had to make to get 256 threads per block. First was to select a block size that is a multiple of warp size (32 threads) because threads are dispatched in warps. If block size is not a multiple of warp size, the “last warp” will have threads that are fewer than 32 so SPs will be idle when the last warp is dispatched. Second was to pick a block size that is at least big enough to take advantage of multiple block schedulers. If I am not mistaken, the GeForce GTX TITAN Z has at least 4 warp schedulers, so at minimum my block size should be 4*32 = 128. This ensures that even if I have only one block assigned to the SM, all my threads can be divided into four warps and scheduled simultaneously. On the other hand, if I picked block size that’s less than 128, say 64, only two warps will be executed simultaneously even though my SM has the capability to dispatch four at the same time. Thus, I experimented with 128, 256, 512 and 1024 threads per block. The results were very similar for the first three and slightly slower for 1024. 1024 threads per block may have been slower because the bigger the block size the more registers the block requires, so this may have affected the occupancy rate. Because the first three were the relatively the same, I decided to stick with 256 because it’s mid way because small and large. Third, in terms of grid size, I need to ensure that I am picking a size that will be big enough so at least a few blocks will be assigned to an SM. This will ensure that the SM will be kept as busy as possible because when warps from one block is inactive a warp from another block can be scheduled. For large data sizes, a simple calculation of ceil(number of elements/threads per block) would suffice. I picked 1 dimension for both grid and block because the data that we are working with is 1d.

4. Explain the behavior of the graph

CUDA version of the max program is slower than the sequential version for 1000 to 100 million elements. However, we can clearly see that as the number of elements increases, the CUDA program performs better relative to the sequential version. This trend shows that as M gets even larger, eventually the CUDA program will be faster than the sequential. There reason why my CUDA program did not perform nearly as fast as sequential version, but gets better as M increases is because overhead of running CUDA program outweighs the cost of parallelizing the program because the data set is too small, thus as data set gets larger, the overhead heads becomes less of a fraction of the overall run time. This overhead includes the need to allocate (malloc and cudamalloc) memory in both CPU and GPU, memory transfer such as copying data from CPU and GPU (cudamemcpy) and back from GPU to CPU, and launching the kernel, which is a relatively expensive operation. We can essentially hide the latency of kernel call if CPU has enough work while the kernel executes. We are also transferring our data through a PCI Express system, which includes a certain amount of overhead. These overheads will matter less as the number of elements increases.