Shawn Hinnebusch

Parallel Computing for Engineers

Oct 19, 2020

Professor: Dr. Senocak

Homework #5: CUDA timing exercise

For all of these calculations, the compiling flag was set to -O3 optimization for all the runs. This had a significant impact for the CPU code compared to none or even -O2. Part 1 was run multiple times and studied to see how to setup the rest of the homework assignment.

Part 2) Vector addition

Table 1 shows the results of Part 2 vector addition. Using this simple calculation, the CPU speed is actually faster than the GPU for all the calculations. This is due to the overhead time it takes to copy all the data to the GPU memory and copy the memory back. With such a small amount of calculation, the overhead is actually more than the time saved doing the addition in parallel. This shows that it does not always make sense to use the GPU to solve everything.

Table 1: Vector time CPU vs. GPU of varying array size

N size	CPU (ms)	GPU (ms)
1,000,000	1.1110	5.4743
10,000,000	11.8450	38.9816
100,000,000	121.8430	223.1081

Part 3) Stencil with radius 3

Part 3 is using a stencil of adding values surrounding the block. For example, the stencil with a radius of 3 will add the previous 3 blocks along with the next 3 blocks to the current block number. In this case, it was a total of 7 blocks being added together. The results show that for a smaller array size, the calculation is faster on the CPU than the GPU, but as the size increases, the GPU quickly becomes faster as the calculation time becomes greater than the overhead of the memory copy. The results are shown in Table 2.

Table 2: Vector time CPU vs. GPU of varying array size of the stencil

N size	CPU (ms)	GPU (ms)
1,000,000	2.2420	3.8451
10,000,000	27.8940	32.0123
100,000,000	433.5350	173.7370

Part 4) Repeat task 3 but for Radius = 1,5,7,9

Part 4 results are shown in Table 3. While running these results, it was noticed that the GPU time can fluctuate 10-20 seconds if the program is run multiple times. Because of this, the GPU time for the larger radius does not change much or stays about the same time. Some of the larger runs actually executed faster than the smaller radius. This is probably due to GPU time fluctuation.

Table 3: Vector time CPU vs. GPU of varying array size and radius size

N Size	CPU (ms)	GPU (ms)
Radius = 1		
1,000,000	0.9170	4.6254
10,000,000	11.9810	31.9672
100,000,000	180.5180	169.0757
Radius = 5		
1,000,000	3.1800	4.4698
10,000,000	41.0270	27.9521
100,000,000	555.0370	164.8988
Radius = 7		
1,000,000	4.1590	4.3674
10,000,000	50.5690	32.1413
100,000,000	654.4440	187.9389
Radius = 9		
1,000,000	11.6970	3.6454
10,000,000	118.3050	30.9381
100,000,000	1326.7550	171.5497

All the data was plotted in Figure 1 in a Log-Log plot to show a linear comparison of the speed it takes to solve the 1D stencil. It is interesting to notice that there is not much of a time increase on the GPU as the radius increases. The time stays fairly consistent, whereas the time for the CPU increases substantially for each of the added calculations. This is most likely due to saving the local variables into the local memory instead of the global memory. It can pull the values much faster and has less latency than getting the data from the global memory along with running the calculations in parallel.

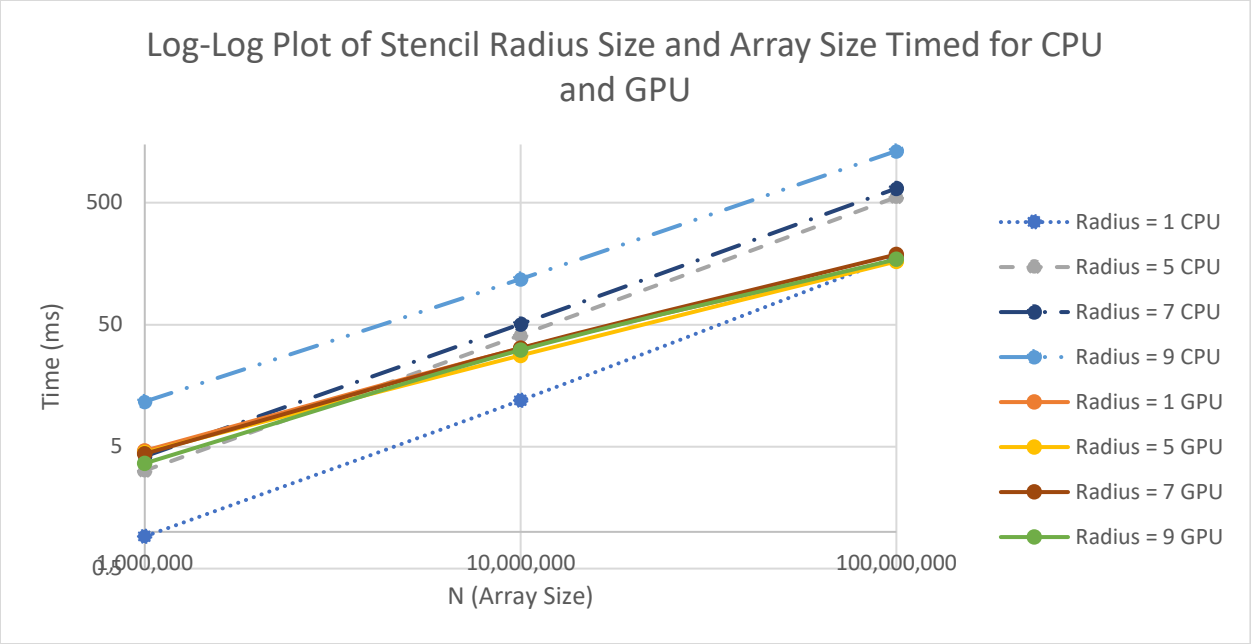

Figure 1: Speed comparison of the stencil radius and array size for the GPU and CPU

Part 5) For vector size of 100,000,000 test the following number of threads per block, 8,16,32,64,128,256,512,1024

For this part, the threads per block were changed with the results shown in Table 4. While changing the threads per block, the only timing that should change is the GPU time. The CPU time was left to show that the time can vary over 20 ms or approximately 5% with repeatedly running the executable. Having a small threads per block does not run as efficiently as larger values. From these results, it seems the best size is either 256 or 512 which is expected as Nvidia recommends a size of 256.

Table 4: GPU time for Threads per Block

Threads per block	CPU (ms)	GPU (ms)
8	429.8510	226.6030
16	435.1910	205.9642
32	432.8160	206.6801
64	424.6640	187.7258
128	420.3780	183.2376
256	424.5750	166.7082
512	441.9090	158.1240
1024	422.8680	162.2938