Keeyon Ebrahimi
GPU
Lab 1

3. Optimizations.

- Each SM had a total of 2048 total threads. I made sure I chose grid and block dimensions to utilize this. Did this by using block dimensions of 32 x 32 and grid dimensions of 2 x 1.

- Each thread would also place it's current spot in the matrix into a shared array, making it quickly accessible to all other threads in the block. I made blocks as large as I could, 32 x 32, in order to have the most amount of memory in this shared array as possible.

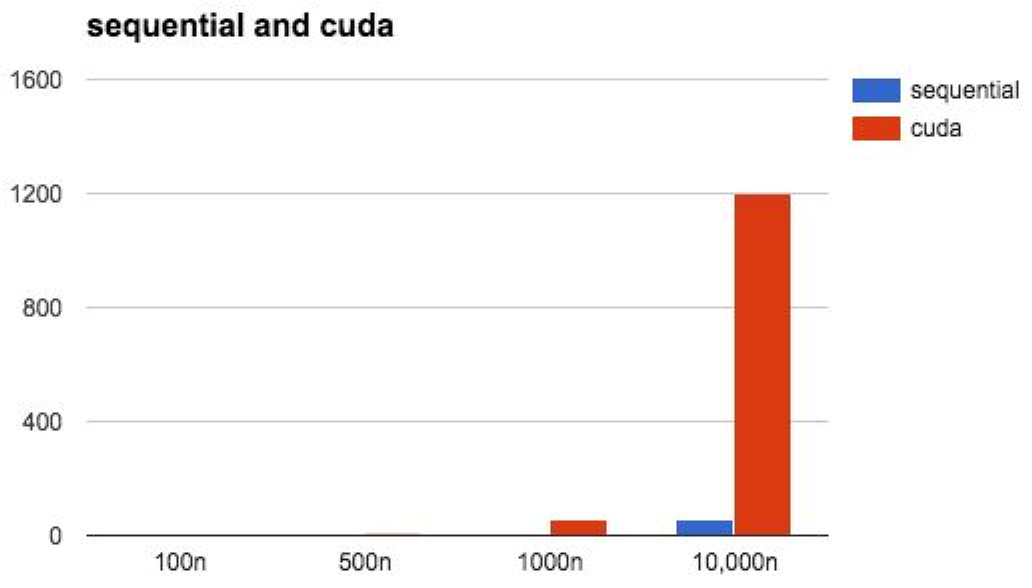
- Edge points received whatever value it placed into the shared memory

- Non Edge points would then be able to grab its left and right elements from shared memory whenever calculating its temperature after an iteration.
    - The very first thread of a block had to grab its left element from global memory and the very last thread of a block had to grab its right element from global memory as well. I maximized block size in order to decrease this number
    - This is where I need more optimization. It is apparent now to me that I cannot have each thread go to global to find the top and bottom position. I cannot think of a way to organize the code to rid the GPU of these hits to memory.

Rest on next page

50 Iterations

6. Graph of N's



7. When is GPU better
    - Note: This should go to 1,000,000n. This was causing segfaults. The segfault occurred when I would malloc a size of 100,000 * 100,000 floats. A way around my initial malloc segfault was to create an array that would then contain multiple arrays of size 10,000 * 10,000. This did not cause the segfault at allocation, yet I was not able to correctly organize the array of arrays of data and send it to the GPU without a segfault.

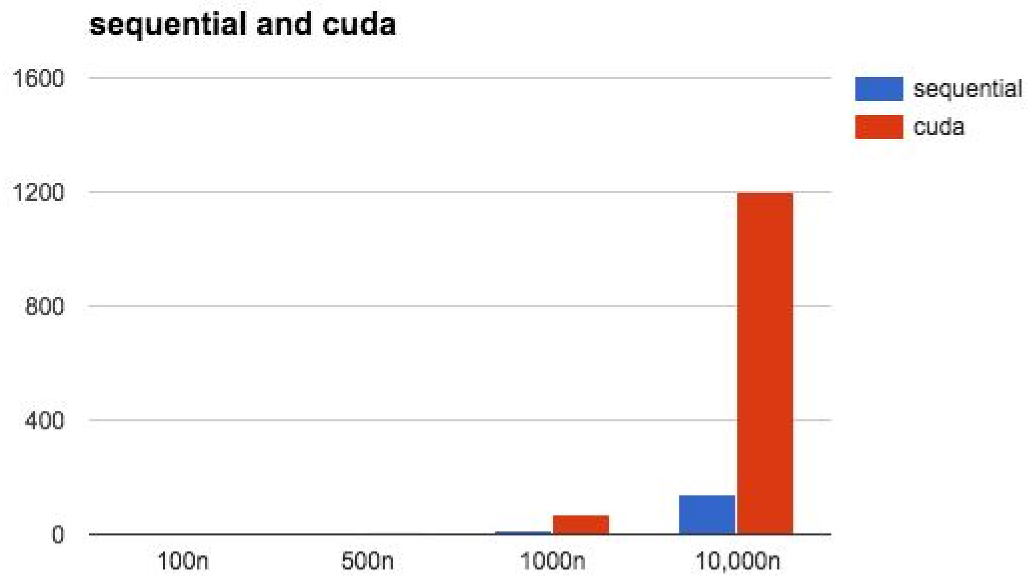
    - With my knowledge of GPU's I would say that theoretically the code should be running faster on the cuda machines around 10,000n. Maybe even more. That is not what I saw thought with my results. After this assignment, it shows me that the data transfer is quite slow, and so the size has to be quite large in order to have the be better than the CPU. I also don't think I am using the GPU efficiently as well.

    - Speedup should be at it's lowest when n is the lowest. This is because the memory transfer is the expensive part of the operation with GPU. When we are not using large amounts of data, we are not seeing the GPU utilized the best. My results did not come out this way though. This makes me come to believe that my GPU coding wasn't done the most efficient way.

    - Speedup should be highest when we have more data. This is because we have more computations, and with a large amount of computations, the GPU shines. I was not seeing this result though, which I once again believe is an issue with my GPU code that I cannot find.

500 Iterations

6. Graph of N's



Similar to the top, but we are getting much closer to the times of the sequential code. It is quite apparent that the more iterations that happen, the better it is to operate with the GPU.

7. In regards to the solutions to problem 7. I am seeing similar results. My GPU time never beats out my sequential code, yet I am getting much closer when we go to 500 iterations. As said before, the expensive part for the GPU is the data transfer. After the data transfer, the GPU starts to shine. With higher executions, we should be seeing much better results with the GPU.

9. The expensive part for the GPU is transferring data around. After we have done that, the GPU starts to shine. With more iterations, the GPU starts to perform better than previously.