A4

Alexander Mueller and Jonas Bonnaudet

November 2022

1 Implementation, Optimizations

Our implementation gives one pixel per thread, with warp sizes of 64. We don't use any threads to compute the sides because they will stay at 0 anyways.

In order to save time, we used only one if statement to check if we are computing the middle four cells which should be set to 1000. It is faster because then we have only one divergent branch compared to four which all would happen serially.

We can omit setting the mid values to 1000 if we never try to compute their values. This made our code slightly slower.

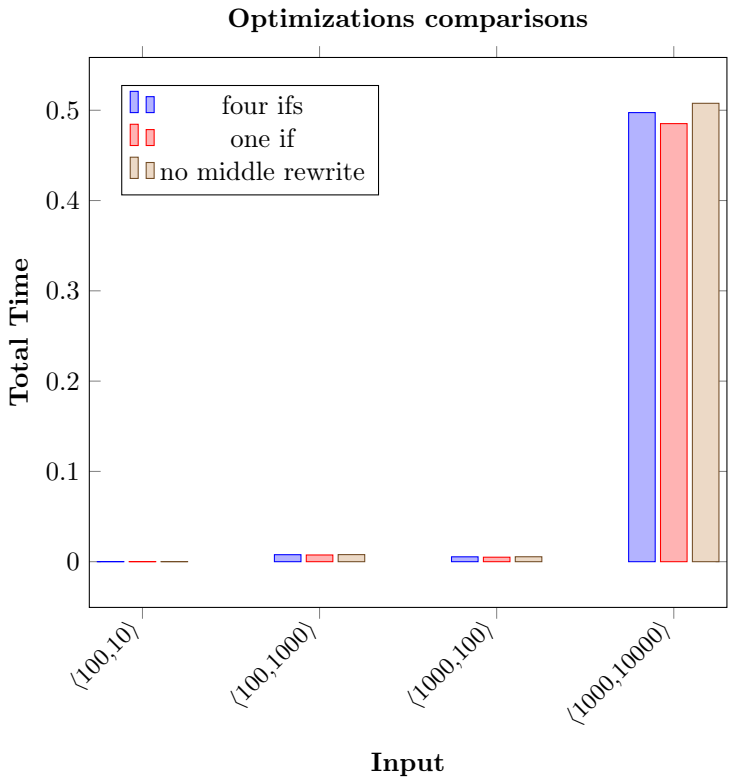
Another optimization that is possible is using shared memory instead of global memory. Unfortunately, we were not able to make this work for length bigger than 76.

We also moved the for loop out of the kernel function, and instead invoke the kernel function n times instead of having the for loop inside the kernel function.

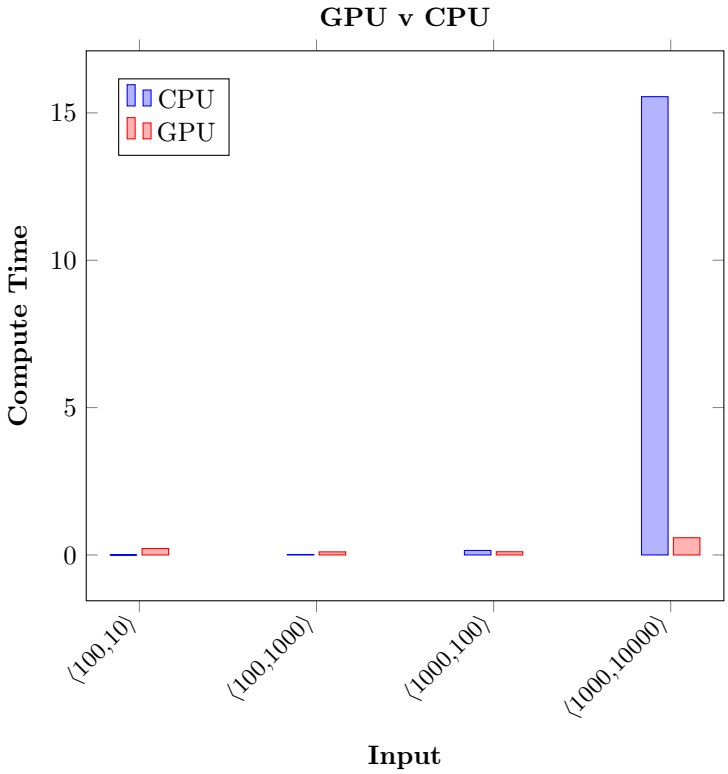
2 Tests on individual optimizations

To test our optimization with the if statements, we look at the other possible implementation, having each thread set the middle 4 cells to 1000, and removing the if statement.

We can also test our implementation with the for loop inside the kernel function:


Optimizations comparisons
four ifs
one if
no middle rewrite
Total Time
0
0.1
0.2
0.3
0.4
0.5
⟨100,10⟩
⟨100,1000⟩
⟨1000,100⟩
⟨1000,10000⟩
Input

3 CPU vs GPU Performance



4 Analysis

The GPU is much faster than the CPU on higher input sizes and iterations, even including the memory copy, due to the DMA. This is because the GPU instead of having a complexity of $O(N^3)$ has a linear complexity because we don't have to loop over each pixel, just over all iterations.