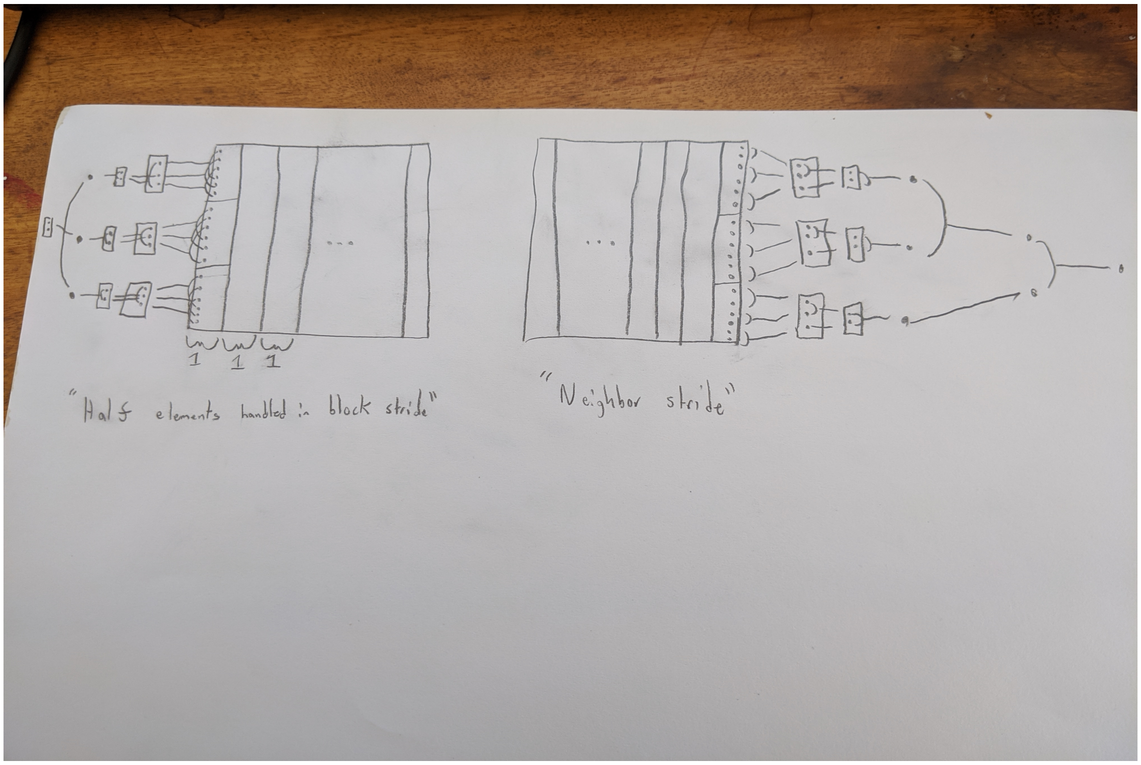
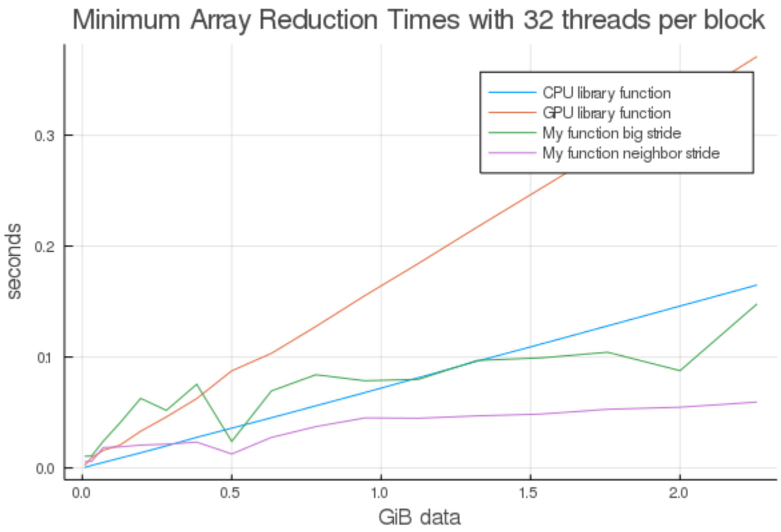
HPC hw2

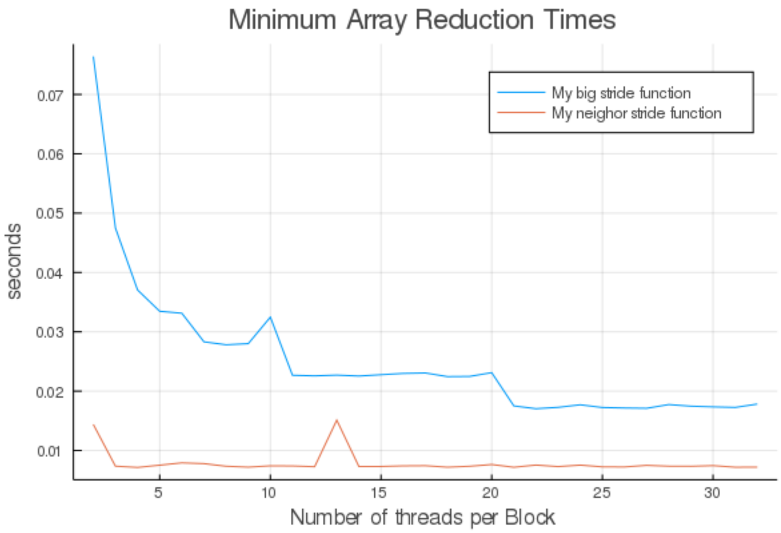
Toby Harvey

March 19, 2020

While I implemented all the GPU functions for find edges, I became particularly interested in the reductions for min and max. I decided to take two different strategys for the reduction, both share the same basic scheme in which each column of an array is split into a constant number of blocks. I partitioned the array like this because I assumed the GPU will access memory contigiously by column. If we are comparing two elements then the GPU has a chance of reading both at the same time. Once a reduction is done on every single block of the Array we have effectively 1-dimensionalized our Array to vector of block-mins or block-maxs. From there we can again reduce the vector on the GPU with a almost identical algorthim (our indices are just no longer two dimensional). I choose to experiement with how to compare elements/what the stride between elements in a block should be for each comparison by a thread. One way (that jeremy implemented) is shown in my drawing, in which we stride by the number of threads or half the number of elements in a block. Alternatively, I also tried striding just to a neighbor, as shown in the right side of my drawing. My thinking was that the GPU would definitely read both an element and its neighbor at the same time which would improve bandwidth, where as there still my be a chance they arent read together in the larger strided version. The the second two figures compare the speed of each implementation.


1 1 1
"Half elements handled in block stride"
"Neighbor stride"


Minimum Array Reduction Times with 32 threads per block
CPU library function
GPU library function
My function big stride
My function neighbor stride
seconds
0.3
0.2
0.1
0.0
0.0
0.5
1.0
1.5
2.0
GiB data


Minimum Array Reduction Times
My big stride function
My neighor stride function
seconds
0.07
0.06
0.05
0.04
0.03
0.02
0.01
5
10
15
20
25
30
Number of threads per Block