Benjamin Horn

Parallel Computing

Homework #5 Due 10/16/17

Q1: Scan Algorithms

1.a Show the trace the parallel executions of the Hillis-Steele scan algorithm on vector [1,2,3,4,5,6,7,8].

On attached PDF.

1.b Show the trace the parallel executions of the Blelloch scan algorithm on vector [1,2,3,4,5,6,7,8].

On attached PDF.

1.c. Which is likely the faster scan to use if you have n elements and n processor device, and why?

For the same number of elements and processors you should prioritize step efficiency. Hillis Steel will prioritize steps over work, therefore it should be used. You would not use Blelloch because you already have more than enough processors to the work needed so you are not concerned with work efficiency.

Hillis Steele - More Step Efficient - Use for more processors than work.

* Work = nlog(n)
* Steps = log(n)

Blelloch - More work efficient - More work than processors

* Work = 2n
* Steps = 2log(n)

1.d Which is likely the faster scan to use if you have n^3 elements and n processor device, and why?

You should use Blelloch because it is the most work efficient. You do not have a ton of processors but you have a lot of operations/ work to do. This means you should prioritize work efficiency, which Blelloch does.

Q2: Transpose Coding Exercise:In this coding exercise you will write two tiled versions of the transpose operation, and gather timing results.Starter code is available here: http://pastebin.com/TSMzEHZd You are encouraged to try different values for K in a KxK tiling. What value of K maximizes occupancy/performance on your device?

//////////////////////////////////////////// k = 1

-bash-4.1$ ./a.out

transpose\_serial: 77.1228 ms.

Verifying ...Success

transpose\_parallel\_per\_row: 3.24138 ms.

Verifying ...Success

NOT SHARED: transpose\_parallel\_per\_element\_tiled 1x1: 7.63168 ms.

Verifying ...Success

SHARED: transpose\_parallel\_per\_element\_tiled\_shared 1x1: 8.97046 ms.

Verifying ...Success

-bash-4.1$

//////////////////////////////////////////// k = 2

-bash-4.1$ ./a.out

transpose\_serial: 77.1193 ms.

Verifying ...Success

transpose\_parallel\_per\_row: 3.20704 ms.

Verifying ...Success

NOT SHARED: transpose\_parallel\_per\_element\_tiled 2x2: 2.0639 ms.

Verifying ...Success

SHARED: transpose\_parallel\_per\_element\_tiled\_shared 2x2: 2.46195 ms.

Verifying ...Success

-bash-4.1$

//////////////////////////////////////////// k = 8

-bash-4.1$

-bash-4.1$ ./a.out

transpose\_serial: 77.1198 ms.

Verifying ...Success

transpose\_parallel\_per\_row: 3.20157 ms.

Verifying ...Success

NOT SHARED: transpose\_parallel\_per\_element\_tiled 8x8: 0.231456 ms.

Verifying ...Success

SHARED: transpose\_parallel\_per\_element\_tiled\_shared 8x8: 0.236544 ms.

Verifying ...Success

-bash-4.1$

//////////////////////////////////////////// k = 16

-bash-4.1$ ./a.out

transpose\_serial: 77.1245 ms.

Verifying ...Success

transpose\_parallel\_per\_row: 3.2096 ms.

Verifying ...Success

NOT SHARED: transpose\_parallel\_per\_element\_tiled 16x16: 0.174336 ms.

Verifying ...Success

SHARED: transpose\_parallel\_per\_element\_tiled\_shared 16x16: 0.134816 ms.

Verifying ...Success

-bash-4.1$

//////////////////////////////////////////// k = 32

-bash-4.1$ ./a.out

transpose\_serial: 77.141 ms.

Verifying ...Success

transpose\_parallel\_per\_row: 3.2112 ms.

Verifying ...Success

NOT SHARED: transpose\_parallel\_per\_element\_tiled 32x32: 0.349728 ms.

Verifying ...Success

SHARED: transpose\_parallel\_per\_element\_tiled\_shared 32x32: 0.345664 ms.

Verifying ...Success

-bash-4.1$

When k is 32 it maximizes performance on my device. It has the best run times consistantly.

Q3: Complete Homework Problem Set #2 from the Udacity CS344 course on Gaussian Blurring. Upload your modified student\_func.cu.

Student\_func.cu is attached.