Lab 6 Writeup – Shubh Patel

(1) Name three applications of parallel scan.

* In geometry: Finding out the convex hull of a set of points.
* Image processing: Filtering, and convolution operations. Like Gaussian blur or edge detection.
* Data compression.

(2) How many floating operations are being performed in your reduction kernel? Explain.

* 2 \* Log2(len) per thread. There are two floating point add instructions being done. All two are in loops. Each one executes Log2(len) times where len is the number of elements. I’m not sure if floating point assignments count, but if they do, then the total number of floating point operations would be 2 \* Log2(len) + 3 per thread. 2 of the assignments are in loops that execute Log2(len) times while the other three just execute once per thread outside of any loop.

(3) How many global memory reads are being performed by your kernel? Explain.

* There is 1 global memory read being performed per thread. Which is storing from global memory to shared memory.

(4) How many global memory writes are being performed by your kernel? Explain.

* There are 2 global memory writes per thread. 1 to the main output array, and 1 to the aux array.

(5) What is the minimum, maximum, and average number of real operations that a thread will perform? Real operations are those that directly contribute to the final reduction value.

* Min – 2. Max – 2log2(len). Avg – log2(len). The real operation done is defined as temp[] += temp[] I believe. And this operation is in two separate loops. So when length is 1, than the min is 2. When we have a len size that is as max length limited by hardware, than max is 2log2(len). And I’m assuming avg is the division of the two.

(6) How many times does a single thread block synchronize to reduce its portion of the array to a single value?

* 2 times. Once when building the array, once when reducing the array.

(7) Describe what optimizations were performed to your kernel to achieve a performance speedup.

* Shared memory is probably the main optimization. This is used to make the threads in a block cooperate together to perform reduction in shared memory space to prevent too many global accesses.
* Another big optimization is splitting the kernel into two phases. This was done to improve efficiency and have a work-efficient kernel.
* Utilized bit shift operations such as >> or << was also done as they are more efficient than multiplying or dividing by 2.

(8) Describe what further optimizations can be implemented to your kernel and what would be the expected performance behavior?

* Perhaps one can utilize loop unrolling. This would lead to higher performance as there would be less loop overhead and less wasted instructions.

(9) Suppose the input is greater than 2048 × 65,535, what modifications are needed to your kernel?

* I would need to utilize the aux array and have multiple kernel calls. I would also need to split the data in some way.

(10) Suppose you want to scan using a binary operator that's not commutative, can you use a parallel scan for that?

* I would assume so, but it seems like the algorithm would have to be heavily modified.

(11) Is it possible to get different results from running the serial version and parallel version of scan? Explain

* I would assume that this depends on the robustness of an algorithm. If your algorithm isn’t robust enough, I.E doesn’t contain enough fail safes or not accounting for edge and corner cases, then it probably would be possible to get different results. Especially on non-commutative binary operators because I assume there’s more room for errors in those.