A4 – Reduction and Scan

Reference: Wen Mei Hwu, PMPP, 2e..
Deadline: 9AM, September 05, 2017. Upload assignment here: https://cloud.nitk.ac.in/index.php/s/0DkF8AdpRbbsWsW

These assignment questions are courtesy the GPU Accelerated Computing kit by UIUC and NVIDIA. Dataset generators and the template CUDA code may have errors. The image processing programs in this assignment use image read/write code from libwb (https://github.com/abduld/libwb). Do understand what's happening under the hood to extract the max out of this assignment.

Q1. Reduction. Implement a kernel the performs reduction of a 1D list. The reduction should give the sum of the list. You should implement the improved kernel discussed in week 4. Your kernel should be able to handle input lists of arbitrary length. However, for simplicity, you can assume that the input list will be at most 2048 x 65535 elements so that it can be handled by only one kernel launch. The boundary condition can be handled by filling "identity value (0 for sum)" into the shared memory of the last block when the length is not a multiple of the thread block size. Further assume that the reduction sums of each section generated by individual blocks will be summed up by the CPU.

Q2. Reduction using the Thrust Libray. Use the Thrust template library to perform a reduction of a 1D list of 32-bit floats. The reduction should give the sum of the list. Your code should be able to handle lists of arbitrary length, but for brevity assume that all data will fit within the GPU global memory.
Questions

1. Name 3 applications of reduction.
2. Are there places in your solution where there is an implicit memory copy between the host and device (a copy that is not from thrust::copy)?
3. If the Thrust version of reduce were not performing as well as you expected, how might you go about investigating and solving the problem?

Q3. Scan. Implement a kernel to perform an inclusive parallel scan on a 1D list. The scan operator will be the addition (plus) operator. Implement a work efficient kernel. Your kernel should be able to handle input lists of arbitrary length. Assume that the input list will be at most of length 2048 x 65,534 elements. This means that the computation can be performed using only one kernel launch.
The boundary condition can be handled by filling "identity value (0 for sum)" into the shared memory of the last block when the length is not a multiple of the thread block size.

Instructions
Edit the code in the template to perform the following:

1. allocate device memory
2. copy host memory to device
3. initialize thread block and kernel grid dimensions
4. invoke CUDA kernel
5. copy results from device to host
6. deallocate device memory
7. implement the work efficient scan routine
8. use shared memory to reduce the number of global memory accesses, handle the boundary conditions when loading input list elements into the shared memory

The executable generated as a result of compiling the lab can be run using the following command:
./ListScan_Template -e <expected.raw> -i <input.raw> -o <output.raw> -t vector
where <expected.raw> is the expected output, <input.raw> is the input dataset, and <output.raw> is an optional path to store the results. The datasets can be generated using the dataset generator built as part of the compilation process.

Questions

1. Name 3 applications of parallel scan.
2. How many floating operations are being performed in your reduction kernel?
3. How many global memory reads are being performed by your kernel?
4. How many global memory writes are being performed by your kernel?
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.
6. How many times does a single thread block synchronize to reduce its portion of the array to a single value?
7. Describe what optimizations were performed to your kernel to achieve a performance speedup.
8. Describe what further optimizations can be implemented to your kernel and what would be the expected performance behavior?

9. Suppose the input is greater than 2048*6535, what modifications are needed to your kernel?
10. Suppose a you want to scan using a a binary operator that's not commutative, can you use a parallel scan for that?
11. Is it possible to get different results from running the serial version and parallel version of scan?

Q4. Thrust List Scan. Implement a kernel to perform an inclusive prefix scan on a 1D list using Thrust.
Given an input
x = [x0, x1, x2, ...]
Produce an output
y = [y0, y1, y2, ...]
where
y[0] = 0
y[1] = 0 + x[0]
y[2] = 0 + x[0] + x[1]
y[i] = y[i-1] + x[i-1]
The prefix scan should produce y given x and use the thrust::inclusive_scan function. The input and output will both be float arrays of equal length.

Instructions
Edit the code in the template to perform the following:
1. Generate a thrust::dev_ptr<float> for host input arrays
2. Copy host memory to device
3. Invoke thrust::inclusive_scan
4. Copy results from device to host

Run command:
./ThrustListScan_Template -e <expected.raw> -i <input.raw> -o <output.raw> -t vector
where <expected.raw> is the expected output, <input.raw> is the input dataset, and <output.raw> is an optional path to store the results. The datasets can be generated using the dataset generator built as part of the compilation process.

Questions
1. Name 3 applications of scan.
2. Suppose a you want to perform the algorithm using a binary operator that's not commutative, can you use still use parallel scan?
3. Is it possible to get different results from running the serial version and parallel version of reduction?