3460:677 LAB 6: LIST SCAN

Objective

Implement a kernel to perform an inclusive parallel scan on a one dimensional list. The scan operator will be the addition (plus) operator. You should implement the work efficient kernel from the lecture notes. Your kernel should be able to handle input lists of arbitrary length. To simplify the lab, the student can assume that the input list will be at most of length $2048 \times 65{,}535$ 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.

Prerequisites

Before starting this lab, make sure that you have completed all of the Module 11 lecture videos and materials.

Local Setup Instructions

The most recent version of source code for this lab can be found in a `.zip` file on the class web page. As before, download it to your Windows account on a lab machine, then copy it to your `tesla` account. Log into `tesla` and `unzip` the files there. (Note that this has been designed to work on and tested on `tesla`. You may do your work on other CUDA-capable equipment but you are on your own altering the seed files so that they execute.)

Overview

The **`Lab5`** directory contains four files:

- **`run.sh`** is a Linux batch file that compiles and executes a named CUDA program;
- **`listScan.cu`** is the code template for the work efficient list scan algorithm; and
- **`input.raw`** is the input data file to test your program, with **`output.raw`** the correct answer used to check your work.

The code templates are provided as a starting point. The code handles the import and export of data files as well as the checking of the solution. You are expected to insert your code in the sections demarcated with //@@. Leave the other code unchanged. To compile and execute use the command **`./run.sh listScan`**.

Instructions

Edit the `main()` code in **`listScan.cu`** to perform the following:

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

Additionally complete the list scan kernels as described above. Instructions about where to place each part of the code is demarcated by the `//@@` comment lines.

Lab Report and Submission

When finished write a one-to-two page lab report (single-spaced, 1-inch margins, 12- to 14-point font size) summarizing your work and findings. In your report answer these questions:

(1) Name three applications of parallel scan.
(2) How many floating operations are being performed in your reduction kernel? Explain.
(3) How many global memory reads are being performed by your kernel? Explain.
(4) How many global memory writes are being performed by your kernel? Explain.
(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*65535, 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? Explain.

Submit the edited `.cu` file and your lab report by the due date for grade.