EECS 368/468 – Lab 4
Programming Massively Parallel Processors with CUDA

Parallel Prefix Scans in CUDA

In this assignment, you will work with your teammates to write and optimize a parallel prefix scan code in CUDA. The lab is due on Thursday March 10 at 11:59 pm, but I decided to offer an automatic 9-day extension that you can voluntarily take if you wish. Thus, for a chance to receive full grade, you must submit the lab by Saturday March 19 at 11:59pm. There will be no late submissions permitted for this lab, as grades are due immediately after it. Please follow the instructions below to do the lab.

Introduction

Scan (i.e., parallel prefix sum) is a useful building block for many parallel algorithms, such as radix sort, quicksort, tree operations, and histograms. Exclusive scan applied to an Array A will produce an Array A', where:

A'[i] = A'[i-1] + A[i-1] , A[0] = 0

or:

A'[i] = sum(j = 0 to i-1) { A[j] } , A[0] = 0

Install Lab 4 at your Wilkinson Lab account

1. Download the lab4 tarball from canvas into the `EECS468-CUDA-Labs` directory in your Wilkinson Lab account.
2. **Remember**: every time you login to the Wilkinson lab to program in CUDA you need to setup the CUDA environment. If you use csh or tcsh:
   ```
   source /usr/local/cuda-5.0/cuda-env.csh
   ```
   or if you use the bash shell:
   ```
   . /usr/local/cuda-5.0/cuda-env.sh
   ```
3. Install the lab4 tarball and compile the lab sources:
   ```
   cd EECS468-CUDA-Labs
   tar xvf EECS468-CUDA-Lab4.tgz
   make
   ```

Application description

The source code you will be working on is at `EECS468-CUDA-Labs/labs/src/lab4`. Compiling your code will produce the executable `lab4` in the directory `EECS468-CUDA-Labs/labs/bin/linux/release`.

There are several modes of operation for the application:

- **No arguments**: randomly generate input data and compare against the host's result.
- **One argument**: randomly generate input data, and write the result to the file specified by the argument.
- **Two arguments**: the first argument specifies a file, the contents of which are read for the array size. Randomly generate input data, and write it to the file specified by the second argument (good for generating test arrays).

- **Three arguments**: the first argument specifies a file, the contents of which are read for the array size. The second and third arguments specify files to read the input array and write the resulting array respectively.
- **Note:** if you wish to use the output of one run of the application as an input, you must delete the first line in the output file, which displays the accuracy of the values within the file. The value is not relevant for this application.

When run, the application will report the timing information for the sequential code followed by the timing information of the parallel implementation. It will also compare the two outputs and print "Test PASSED" if they are identical, and "Test FAILED" otherwise. The base code provided to you should compile and run without errors or warnings, but will fail the comparison.

Complete the Assignment

Your task is to edit the source files to complete the functionality of the exclusive scan on the device. Use a tiled implementation that can process a very large array.

While scan is an appropriate algorithm for any associative operator, we will be using addition. Please read Mark Harris's report "Parallel Prefix Sum (Scan) with CUDA" and the class notes that cover the same subject, to learn the algorithmic background for this assignment. Note that you do not need to replicate any of their algorithms exactly, but this assignment does require a "work-efficient" algorithm.

You are free to make any modifications you need in the provided *.cu source files.

You may not use anyone else's solution; however, you are allowed to use third-party implementations of primitive operations in your solution. If you choose to do so, it is your responsibility to include these libraries in the tarball you submit, and modify the `Makefile` so that your code compiles and runs. You must also mention any use of third-party libraries in your report. Failure to do so will be considered plagiarism. If you are uncertain whether a function is considered a primitive operation or not, please inquire with the instructor.

Submitting your solution

1. Create a tarball of your solution, which includes the contents of the `lab4` source folder, with all the changes and additions you made to the source files. Please make sure you do a "`make clobber`" before submitting; do not include object code or executables, as these are typically large files:

   ```
   cd EECS468-CUDA-Labs/labs/src/lab4
   make clobber
   tar cvfz EECS468-lab4-YourNames.tgz *
   ```

   where you should replace "`YourNames`" with the concatenated last names of the team members.

2. In addition to the code, add a report file your `lab4` source directory with your answers to the following questions:

   a. Near the top of "scan_largearray.cu", set #define DEFAULT_NUM_ELEMENTS to 16777216. Set #define MAX_RAND to 3. Record the performance results when run without arguments, including the host CPU and GPU processing times and the speedup.

b. Describe how you handled arrays not a power of two in size, and how you minimized shared memory bank conflicts. Also describe any other performance-enhancing optimizations you added.
c. How do the measured FLOPS rate for the CPU and GPU kernels compare with each other, and with the theoretical performance limits of each architecture? For your GPU implementation, discuss what bottlenecks your code is likely bound by, limiting higher performance.

Please be brief in the descriptions above; there is no need for lengthy descriptions. Even bulleted lists are enough if they convey everything you want to say.

3. Submit your solution tarball via canvas. Only one student in a team needs to submit. If you submit multiple times, only the latest submission will be graded. Please do all submissions from the same account.

Grading

Your submission will be graded based on the following parameters:

a. **Correctness** (20%): the kernel produces correct results at the output.
b. **Optimizations** (50%): this is a qualitative portion of your grade. For this portion, we will grade the thoughtfulness you put into speeding up the application. It is important that your code handles boundary conditions correctly.
c. **Report** (30%)

Good luck!