Sattler College
Computer Science
CS 420 – Systems and Parallel Processing

Lab 4: Reduction and Scan

1. Objective

This lab is divided into two parts. The objective of the first part is to get you familiar with the parallel reduction algorithm. The objective of the second part is to get you familiar with the parallel prefix scan algorithm. Please read Mark Harris's report "Parallel Prefix Sum (Scan) with CUDA" to learn the algorithmic background for the second part. https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-39-parallel-prefix-sum-scan-cuda

2. Procedure

Step 1: Download the program files in Lab_4.zip

Step 2: Edit `<lab-directory-4.1>/kernel.cu` where indicated to implement the device kernel code for the parallel reduction algorithm, assuming an input array of any size that fits in one kernel. You only have to produce the partial sums of each thread block for this part. We will sum up the partial results on the host.

Step 3: Compile and test your code.

```
cd <lab-directory-4.1>

make


./reduction             # Uses the default input size

./reduction <m>         # Uses an input with size m
```

Step 4: Answer the following questions in a new file named `<lab-directory-4.1>/answers.txt`:

- How many times does a single thread block synchronize to reduce its portion of the array to a single value?
- 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.

Step 5: Edit `<lab-directory-4.2>/kernel.cu` to implement host and device kernel code for the parallel prefix scan algorithm. The algorithm should be work-efficient. You are expected to support an input array of any size that fits in one kernel and are not allowed to add up partial sums on the CPU. In other words, you must use the hierarchal scan approach.

Step 6: Compile and test your code.

```
cd <lab-directory-4.2>

make


./prefix-scan                 # Uses the default input size
```


```
./prefix-scan <m>            # Uses an input with size m
```

Step 7: Answer the following question in `<lab-directory-4.2>/answers.txt`:

- Describe how you handled arrays not a power of two in size and all performance-enhancing optimizations you added.

Step 8: Submit your assignment. You should submit all files, including the following files:

- `<lab-directory-4.1>/kernel.cu`
- `<lab-directory-4.1>/answers.txt`
- `<lab-directory-4.2>/kernel.cu`
- `<lab-directory-4.2>/answers.txt`

…

3. Grading:

Your submission will be graded based on the following criteria.

- Functionality/knowledge: 60%
  - Correct code and output results
  - Algorithm efficiency, use of shared memory to hid global memory access latencies, and other optimizations
  - Correct handling of boundary cases
- Report: 40%
  - Correct answers to questions in steps 4 and 7
  - Sufficient work shown
  - Neatness and clarity