CS 629/729 GPU Programming
Lab 6

Spring 2024

Learning Outcomes

- Understanding how to design parallel histogram, scan, and merge algorithms using CUDA.
- Learn how to apply various optimization techniques to better use GPU computing resources.

Task 1

In task 1 we are going to implement a kernel that does parallel histogram operation on a given array. The code structure is provided in task01.cu. Complete the following changes:

1.1 Implement the GPU_histogram kernel by having each thread work on one input element and update the histogram using atomic operation. For your kernel, you should use privatization and shared memory to optimized the performance. You may use the pseudo code provided in the lecture slides.

1.2 Implement a CPU version function for verifying the correctness of the GPU kernel.

1.3 Implement the proper kernel launch code. Adjust the block size to achieve the best performance.

1.4 Add code to call you CPU function to verify the GPU output.

Task 2

In task 2 we are going to implement a kernel that does parallel scan operation on a given array. The code structure is provided in task02.cu. Complete the following changes:

2.1 Implement the GPU_scan kernel using either Kogge-Stone or Brent-Kung algorithm (your choice). For your kernel, your implementation should follow the algorithm you choose and use shared memory to optimized the performance. You may use the pseudo code provided in the lecture slides.

2.2 Implement a CPU version function for verifying the correctness of the GPU kernel.

2.3 Implement the proper kernel launch code. Adjust the block size to achieve the best performance.

2.4 Add code to call you CPU function to verify the GPU output.

Task 3 (CS 729 only)

In task 3 we are going to implement a kernel that does parallel merge operation on two given arrays. The code structure is provided in task03.cu. Complete the following changes:

3.1 Implement the GPU_merge kernel using the tiled algorithm discussed in class. For your kernel, your implementation should use co-ranks to determine the workload per block and per thread and use shared memory to optimized the performance. You may use the pseudo code provided in the lecture slides.

3.2 Implement a CPU version function for verifying the correctness of the GPU kernel.

3.3 Implement the proper kernel launch code. Adjust the block size to achieve the best performance.

3.4 Add code to call you CPU function to verify the GPU output.