COM2039: Parallel Computing

Lab Class 4

Reduce and Scan with Multiple Blocks

Contents

1 Introduction

I provided a solution to last week's lab class on Scan that only works when the input array can be processed within a single block. The reason for that, as discussed in Lecture 11, is that although we have a CUDA primitive to synchronise all the threads within one block, we are not able to synchronise thread execution across blocks. Lecture 11 provided a strategy for handling the case where we need many blocks to process an input array. Essentially, we make sure that each block produces a coherent partial result. We then execute the Kernel again on the collection of partial results. With a very large array, we may need to iterate through repeat executions of the Kernel multiple times in order to obtain the final result.

In this week's lab class you will develop implementations of both Reduce and Scan that work when the input array requires multiple blocks. I gave you all the Kernel code for Reduce in Lecture 11, so your programming task is just to develop the host code.

In the case of Scan, I will show you the basic principle of how to do it, but you will need to have a stab at writing the Kernel code yourselves. As always, I will provide solutions after Friday's lab class.

Not surprisingly, perhaps, Part III of the coursework will require you to obtain timings for these algorithms, including a comparison of the Hillis and Steels and Blelloch versions of Scan.

2 Reduce

As I mentioned, you have the Kernel code for Reduce in the lecture slides. So, the key additional thought you need is in developing the for loop that will execute the Kernel 2 or more times in order to produce an implementation of this. You should find this straightforward, but as always in the case of lab classes, do feel free to exchange ideas with others on the course if you find it hard to get started.

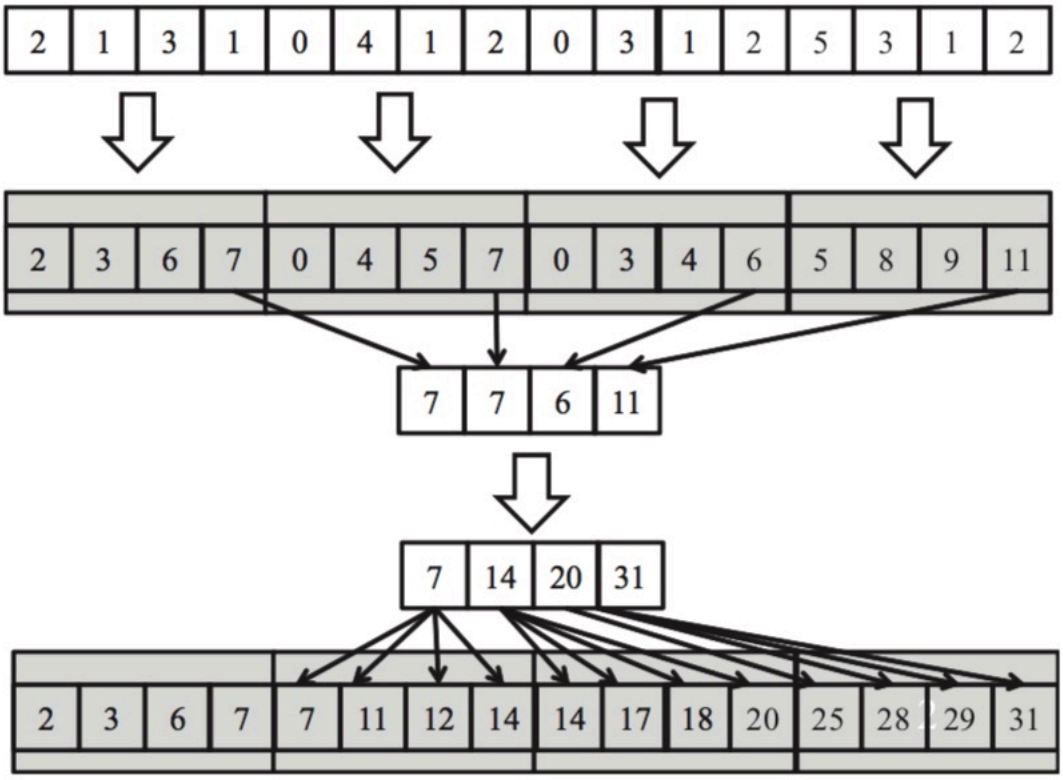
In order to test your implementation, you will need to generate input arrays of certain sizes. I suggest you work in powers of 2, to make it easy to work out what the expected result should be and how many executions of the Kernel you will need. Refer back to Lab Class 1 to see how to use the "left shift" operator to generate an array sizes that is a specified power of 2.

When testing that your program is correct, it is easiest to start with small block sizes (16) at first so you do not need particularly large array sizes to induce multiple Kernel invocations: 16x16 will need two executions of the kernel; 16x16x16 will need three; and so forth.

Once you are comfortable that your code is working, increase the block size and get some timings for input arrays of the order of 1M elements.

3 Scan

Now we need to take a look at modifying the solution to last week's lab class to produce a version of Scan that works across multiple blocks. We will take a similar strategy to Reduce but it is a little more complex. In the first execution of the Kernel, each block will scan a part of the input array and produce a "local scan". See the first grey line on the figure on the next page.



[From Kirk and Hwu, Programming Massively Parallel Processors 2nd Edition]

Note, this is using BLOCK_SIZE = 4 for ease of presentation.

The last element in each block provides the Reduce for each local block. As the reduction of Block 0 is the sum of all the elements to the left of Block 1, we will need to add that value (7) to all the elements in Block 1. If we also add the reduction of Block 0 to the reduction of Block 1 (7+7=14) we get the sum of all the elements to the left of Block 2. So we need to add that value (14) to all the elements in Block 2 to get their final values. And so forth.

You see what is happening. We perform scans on each block with the first invocation of last week's Kernel. Then we need to pull out the last result from the resulting scan output for each block, and then perform a separate scan on that. Finally, we do a map on each element in all but the output from Block 0 to update their respective elements by adding in the appropriate values from our scan of the Reductions (the array on the fourth line of the diagram above.

So, now please try and create the Host code that will implement the above using the Kernel from last week. Again, start by testing it with a BLOCK_SIZE of 16 and then when it is working, try it out with much bigger block sizes, and dimension of the input array.