CSCD 439/539 GPU Computing HW4

Modify Reduction

No Late Submissions are accepted. **Rules:** Your code must CUDA C Language. If your program shows a compilation or run-time error, you get a zero for this assignment.

**Submission:** Wrap up all your **source files** into a single zip file. Name your zip file as *FirstInitialYourLastName*Hw4.zip. For example, if your legal name is Will Smith, you should name your zip file as wsmithHw4.zip. A simple makefile has been provided in the zip file.

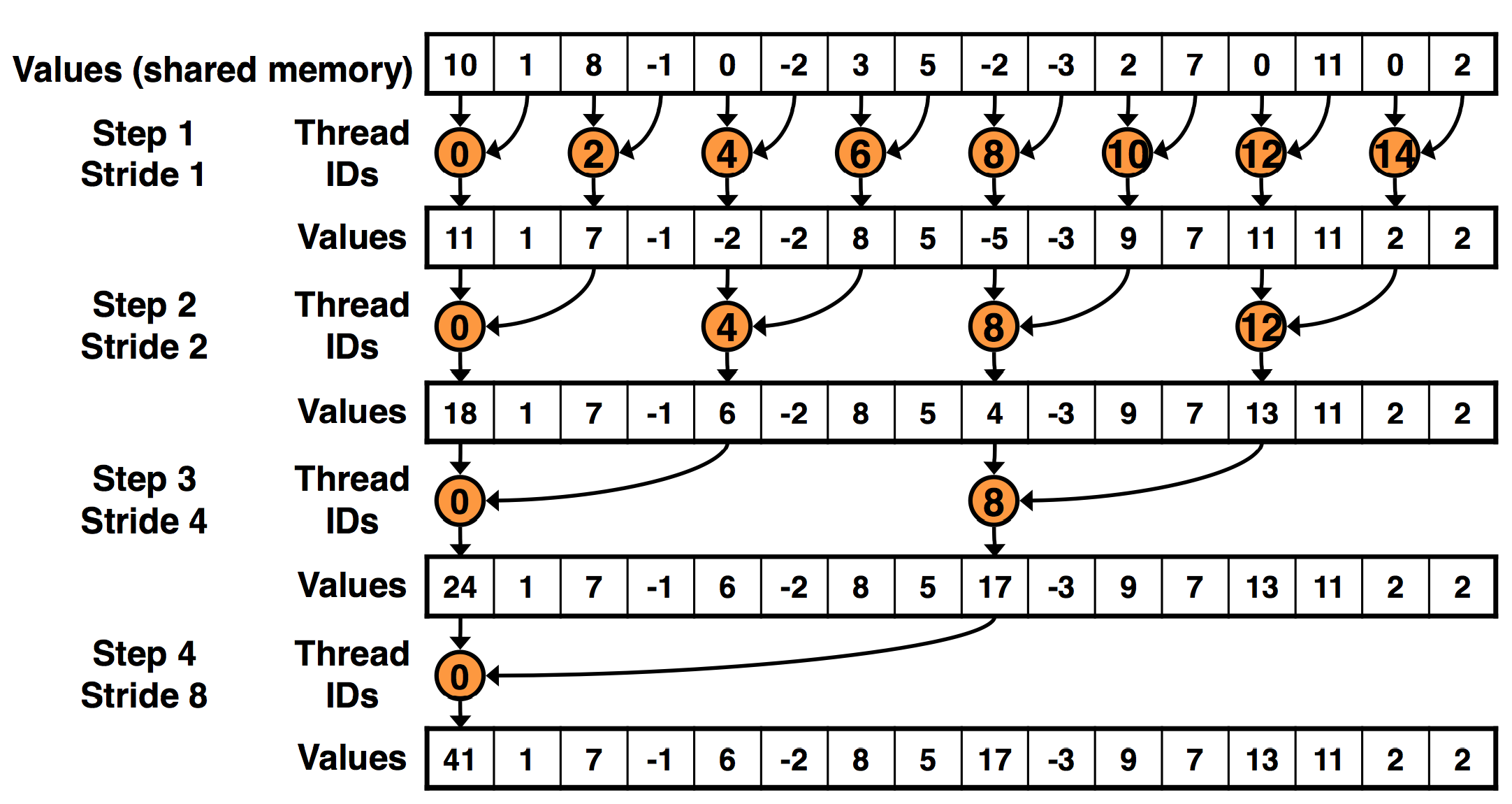
**For archive purpose, please also submit your single zip file on EWU Canvas by following CSCD439-01 Course 🡪Assignments🡪Hw4🡪 Submit Assignment to upload your single zip file.**

**Problem Description:**

Based on the lecture about reduction algorithm on CUDA device, you are required to implement the following features and answer the questions.

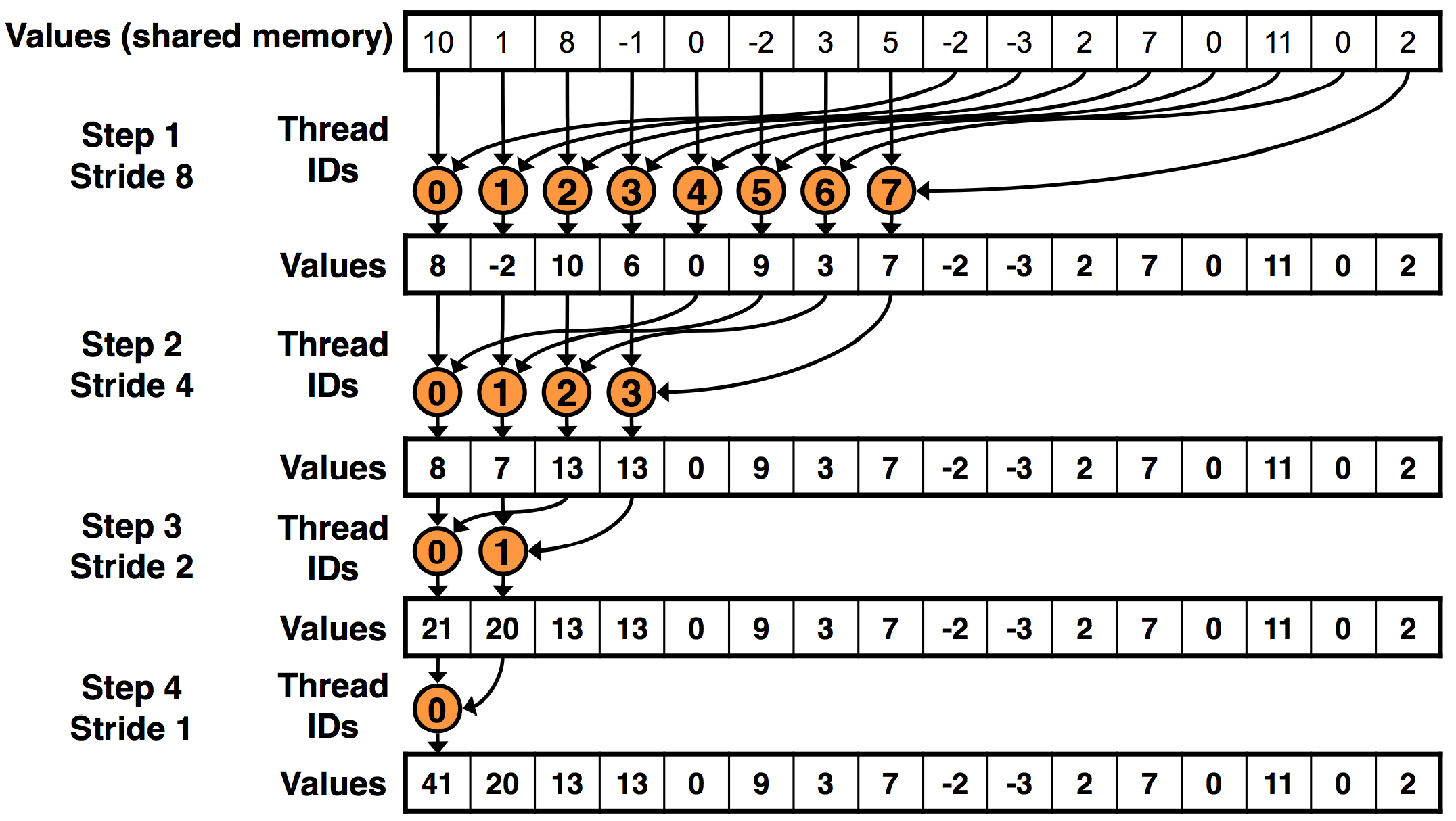
1, Download the demo code d10\_reduce4\_add.tar.gz on canvas under Files🡪DemoCode. Read and understand the provided CUDA C program. We have already gone through this code in class. The demo code is also included in the start up package.

2, The provided kernel in the demo code, \_\_global\_\_ void reduce2(float \*in, float \*out, int n), is not optimized. The basic idea about kernel reduce2() is described in the lecture notes regarding reduction algorithm. The following diagram (figure 1) illustrates reduce2() kernel.



**Figure 1**

3, You have to write another kernel reduce3(), that also performs reduction on CUDA device. Kernel reduce3() accepts the same set of parameters list as reduce2() kernel have, with each parameter maintaining the same meaning. But reduce3() kernel uses the following threads-data mapping and data access patterns, as described below in figure 2.



**Figure 2**

In the design of figure 2, **the size of thread block (in a shape of 1 \* N) is required** to be half of the shared memory size, (i.e. also half of the thread block size in the kernel reduce2()), while each thread block is able to perform the same amount of work in both reduce2() and reduce3(). In this way, we improve the performance of reduce3(), compared with reduce2(). **You have to** **change the execution configuration parameters in the main()** in order to set up the new block size variable for reduce3(). Another major difference between reduce2() and reduce3() kernel pertains to which pair of data elements are processed by a thread.

3, Change the main() function to invoke the kernel **reduce3()** you implemented, and measure the kernel execution time. Then you have to compare the kernel reduce3() with the reduce2() by filling out the following table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input size | 1048576 | 16777216 | 67108864 | 134217728 |
| Block Dimensions | 1 X 1024 | 1 X 1024 | 1 X 1024 | 1 X 1024 |
| T1:time cost for reduce2 (ms) | .000376 | .005392 | .021387 | .042681 |
| T2:time cost for reduce3 (ms) | .000120 | .001251 | .004809 | .008667 |
| Speedup = T1 / T2 | 3.133 | 4.310 | 4.447 | 4.9245 |

4, Do you identify any advantage of reduce3() over reduce2() kernel? Or vice versus? And why ? ( hint: in terms of performance, such as bank conflicts, condition divergence or idle threads. )

This program is much more efficient because it doesn’t have condition divergence or idle threads. With the current program all of the threads are being used at least one, other than with the initial read. Each threads reads in itself and another element from the input array, and then those threads all use those elements in resulting arrays. The warps are also beinig used effectively, because threads that are right next to each other are being used together. When every other thread is being used, then you have a lot of wasted if statements and if every warp takes twice as long to run, then it will at least double the total time.