REPORT: Assignment 4

Submitted by:

Sumaiya Dabeer

20CSZ8509

Introduction:

The objective of this experiment is to sort an integer array by using Cuda programming. Cuda is the first successful attempt to provide GPU programming environment by Nvidia. As we all know that GPUs due to their architecture provides massive parallelism due to blocks as well as GPU threads. Cuda gives an interface to program that. While translating the CPU code to GPU there are some important things to take into consideration.

Nature of Algorithm: performance of algorithm on GPU is highly dependent on nature of algorithm. Some Algorithms are sequential in nature (like merging, prefixScan, Quicksort etc) gives vary bad performance when run directly on GPU(Although the modified parallel versions of them are efficient). On the other hand some algorithms are parallel in nature like radix sort (has first rank of all GPU sorting algorithms) and bitonic sort (we have used this in our code).

Structure of Program: performance is also highly dependent of program structure as organization of that is more into hand of programmer. There are some steps that are very costly in programming (moving data between GPU and CPU, synchronization of threads, number of blocks and threads, recursion or function call from kernel to kernel, nonefficient use of resources like global and shared memory).

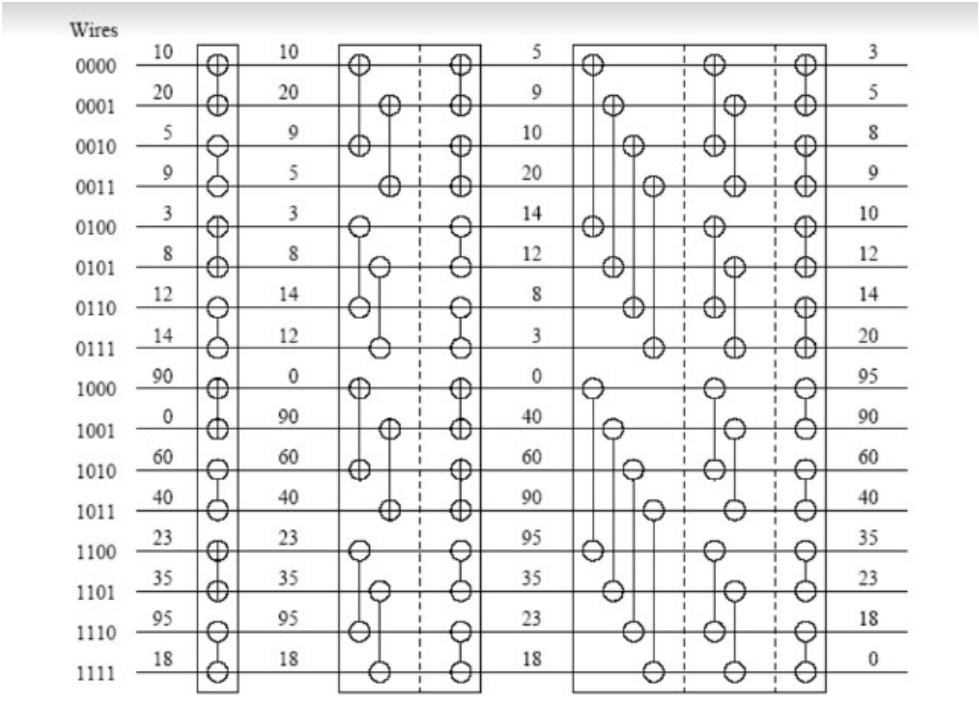

Figure 1: Pictorial representation bitonic sort's working

Implementation:

We have implemented bitonic sort using Cuda by taking the nature of this algorithm in consideration. From Figure 1 we can see that for *n* nnumber of dataitems we need *Log n phases* (shown as bounding box in figure) and for each phase we need to have *steps from 1 to phases_number.* Nature of algorithm is proper parallel as we can see that at each step all operations can be done simultaneously. Also this algorithm is very much like Odd-Even sort despite direction of swap is depending upon the phase number (inversion of swap operation is done after *2^phase_number*). A kernel function (*gpu_sort*) is called from main and then for each phase other kernel function (*gpu_sort_inner*) is called.

Results:

According to my program's output I can say that my program is completing all the basic requirement, very efficient in speed and also scaled very well. I have summarized the results below.

Scalability: Able to run it with 2^25 size of data without any issue. I hope it will scale more than that.

Sorting: Used bitonic sort and working fine.

Experiment:

Commands when compiled using main function in sortcu.cu file

```
nvcc -arch=sm_35 --device-c -o sortcu.o sortcu.cu #to craete object file
nvcc -arch=sm_35 sortcu.o # this will create output file
```

Commands to generate compiled library

```
nvcc -c -arch=sm_35 --device-c -o sortcu.o sortcu.cu
ar -rsc libsortcu.a sortcu.o
```

Run the file using:

```
./a.out
```

I have checked upto test the data size of 2^25 and it is working fine without taking much time.

Scalability:

Scaling very well and taking very little time for execution and compilation.