Parallel Implementation of Bitonic Sort using CUDA

Presented For CSE702

Instructor: Dr. Russ Miller

Presented By:

Anushree Parmar

University at Buffalo The State University of New York

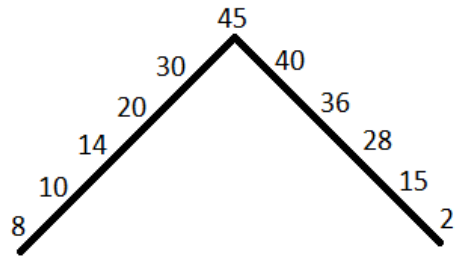
Why Bitonic Sort?

- No of comparisons in Bitonic sort are O(n Log ^{2}n)
- No of comparisons done by most of the algorithms like Merge Sort or Quick Sort take O(n Logn)
- Bitonic sort is better for parallel implementation

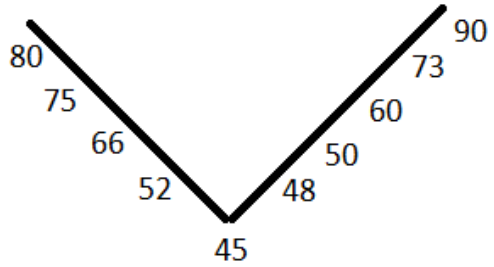
Bitonic Sequence

A sequence numbers is said to be *bitonic* if and only if

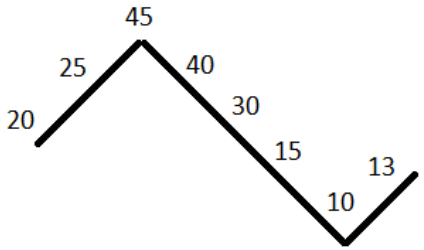
1. Monotonically increases and then monotonically decreases



2. Monotonically decreases and then monotonically increases

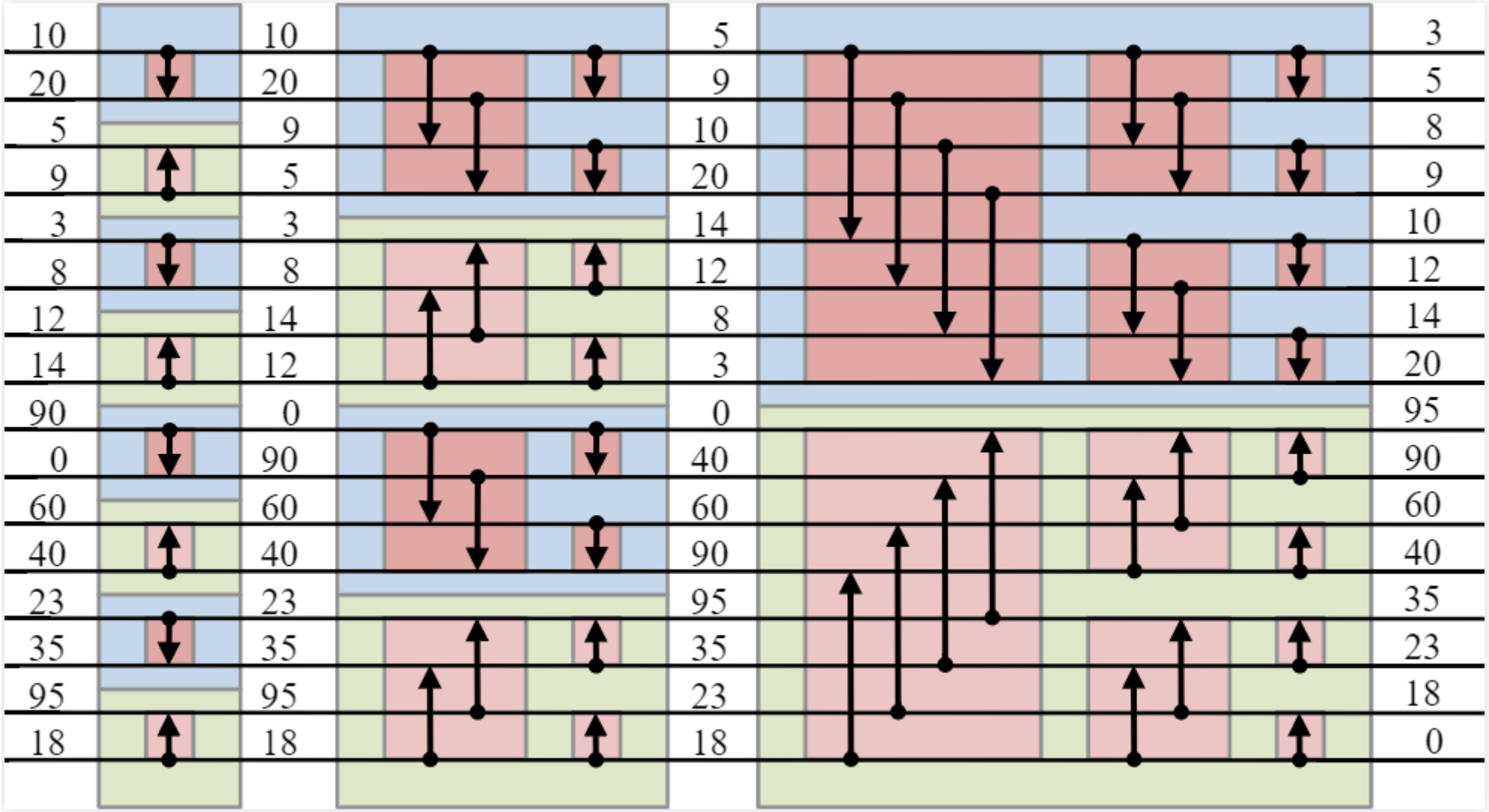


3. Can be split into two parts that can be interchanged to give either of the first two cases.



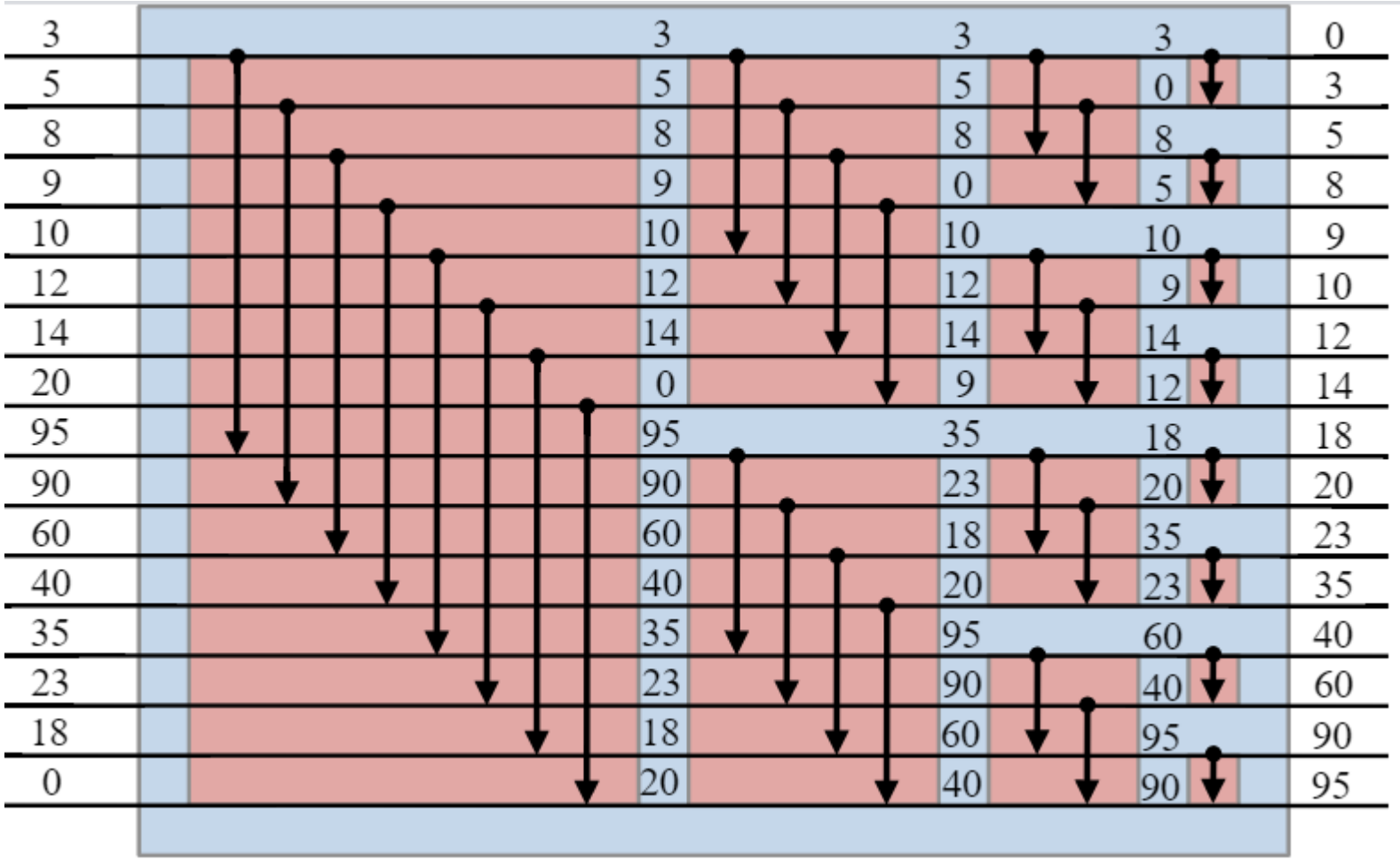
Rearrange to a bitonic sequence

<table>
<tr><td>⊕ BM[2]</td><td rowspan="2">⊕ BM[4]</td><td rowspan="4">⊕ BM[8]</td><td rowspan="8">⊕ BM[16]</td></tr>
<tr><td>⊖ BM[2]</td></tr>
<tr><td>⊕ BM[2]</td><td rowspan="2">⊖ BM[4]</td></tr>
<tr><td>⊖ BM[2]</td></tr>
<tr><td>⊕ BM[2]</td><td rowspan="2">⊕ BM[4]</td><td rowspan="4">⊖ BM[8]</td></tr>
<tr><td>⊖ BM[2]</td></tr>
<tr><td>⊕ BM[2]</td><td rowspan="2">⊖ BM[4]</td></tr>
<tr><td>⊖ BM[2]</td></tr>
</table>



No of comparison levels

1 2 3



No of comparison levels

Algorithm

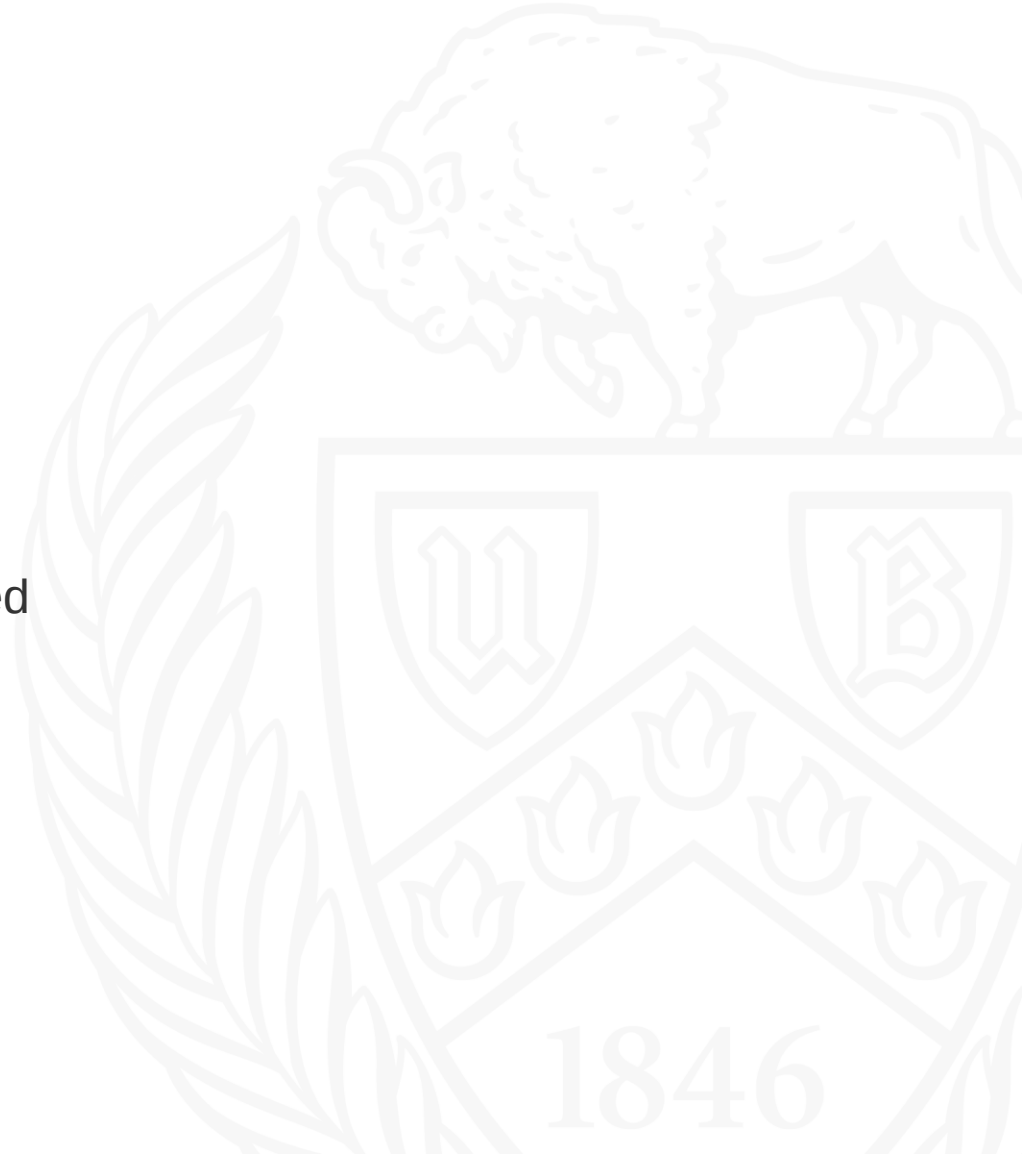
```
BitonicSort(a, low,high,direction):
    if high> 1:
        k = high/2
        BitonicSort(a, low, k, 1)
        BitonicSort(a, low+k, k, 0)
        BitonicMerge(a, low, high, direction)
```

```
BitonicMerge(a, low,high, direction):
    if high > 1:
        k = high/2
        for i in range(low , low+k):
 // Based on direction swap the data
            a[i],a[i+k] = a[i+k],a[i]
            BitonicMerge(a, low, k, direction)
            BitonicMerge(a, low+k, k, direction)
```

CUDA Steps

To execute any CUDA program, there are three main steps:

- Copy the input data from host memory to device memory, also known as host-to-device transfer (cudaMemcpyHostToDevice)
- Load the GPU program and execute
- Copy the results from device memory to host memory, also called device-to-host transfer (cudaMemcpyDeviceToHost)

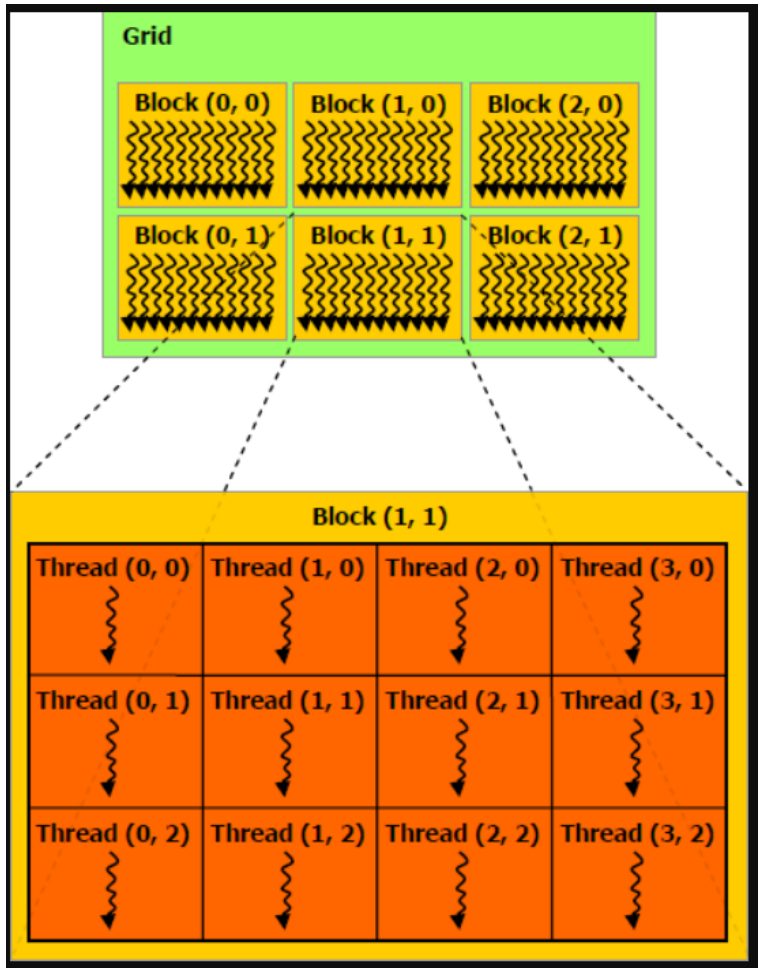


Parallel Algorithm Implementation

- N – No. of threads
- n – No. of blocks
- Generate the (n*N) data randomly
- Allocate the GPU Memory
- Transfer the array to GPU
- Launch the kernel
- Compare the element using block id and thread id parallelly
- Repeat the same process for each level
- Copy back array to CPU

Threads and Blocks

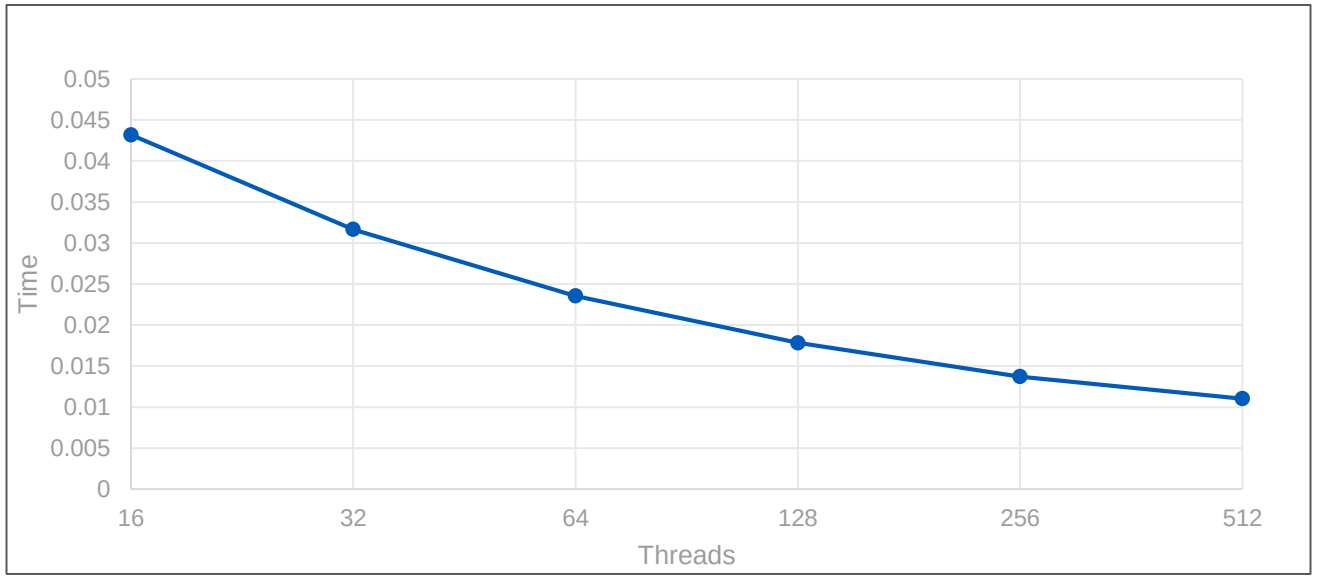
- A group of threads is called a CUDA block. CUDA blocks are grouped into a grid.
- Threads are indexed using the built-in 3D variable threadIdx
- Blocks are also indexed using the in-built 3D variable called blockIdx
- Multiple threads in one block are more optimal than multiple blocks with one thread



Results

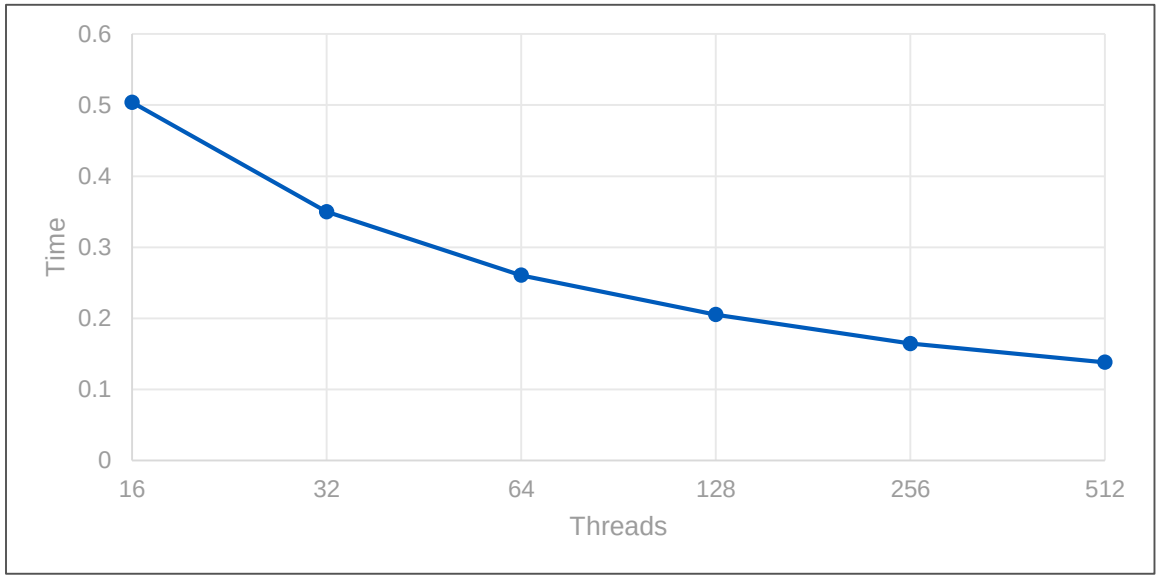
1 Million Data

No. of Threads	Time(s)
16	0.043177
32	0.031664
64	0.023544
128	0.01783
256	0.013715
512	0.011017



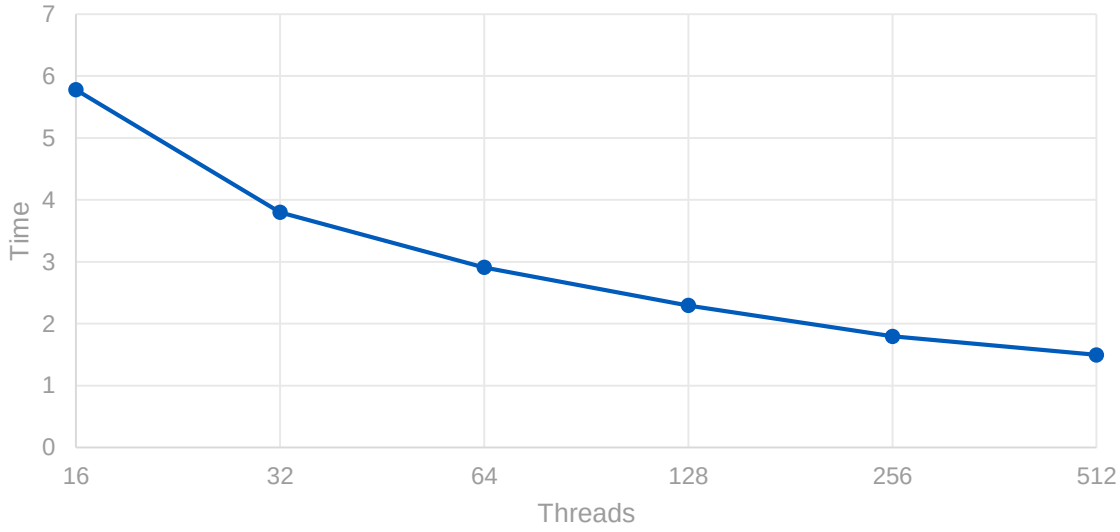
16 Million Data

No. of Threads	Time(s)
16	0.503972
32	0.350034
64	0.260489
128	0.205095
256	0.164531
512	0.13801



134 Million Data

No. of Threads	Time(s)
16	5.777644
32	3.796656
64	2.908625
128	2.294748
256	1.794665
512	1.494665



Conclusion

- Since in CUDA kernels issue instruction in wraps(32 threads). If we select 50 threads per block, CUDA would still assign it in batch of 64, which will waste the usage of 14 threads.
- Blocks with multiple thread will run faster as threads have a shared memory for each block

References

- Algorithms Sequential and Parallel: A Unified Approach by Russ Miller and Laurence Boxer
- https://www.nvidia.com/content/GTC-2010/pdfs/2131_GTC2010.pdf
- https://en.wikipedia.org/wiki/CUDA
- https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/
- https://www.youtube.com/watch?v=F620ommtjqk&list=PLGvfHSgImk4aweyWlhBXNF6XISY3um82_&index=1&ab_channel=Udacity

Thank You.

University at Buffalo The State University of New York