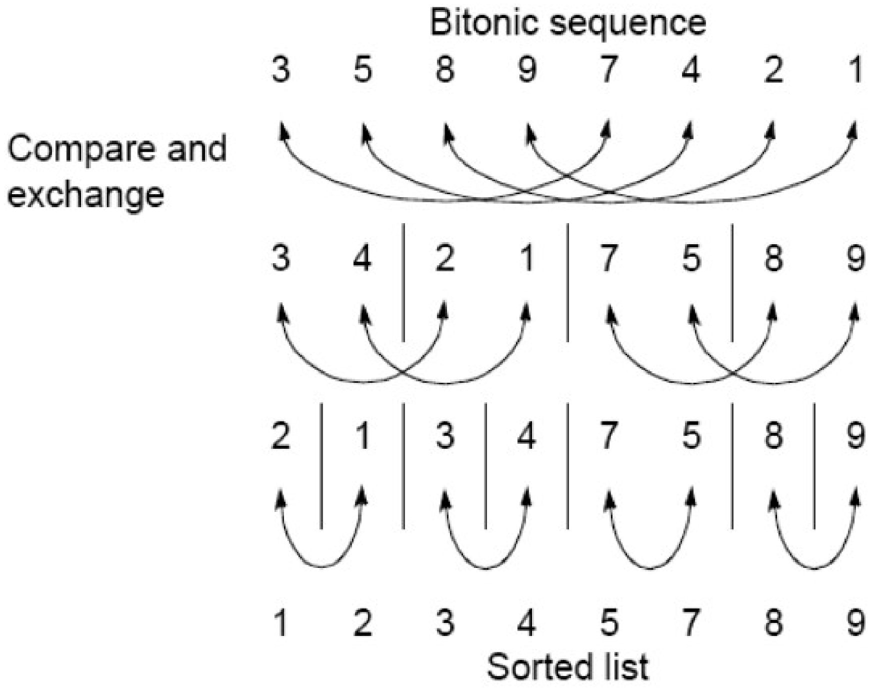
Objective

- To implement bitonic sorting on GPU using CUDA in C to sort 1 block using shared memory
- To merge the sorted blocks to form the large sorted array

Algorithm

- Here we consider a bitonic sequence (list of numbers) having one local maxima or one local minima; Eg; 1 3 5 9 8 5 4 2 , here 9 is the local maxima
- A bitonic sequence can easily be converted to a sorted sequence in logn steps , where n is the total number of elements as shown below



courtesy : https://people.scs.carleton.ca/~dehne/teaching/4009/lectures/pdf/05c-Parallel_bitonic_sort.pdf

- And it can be easily seen that two sorted sequence form a bitonic sequence by arranging them in ascending and descending order or vice versa
- Start with sequence of two numbers . As they are already bitonic sequence , they can be used to form sorted sequences of length 2
- Taking consecutive sorted sequence of length 2 a bitonic sequence of length 4 is obtained and then they are sorted
- This process is continued and finally sorted array is obtained

GPU implementation using cuda

- A block is transferred to the device memory which again copied to the shared memory
- Each thread runs (logn)^2 loops to swap the corresponding indices obtained by xoring the thread id and the variable of inner loop

```
for( i=2;i<=n;i=i*2)
 {
  for( j=i/2;j>0;j=j/2)
  {
       index =thread_id ^ j;
       */
        do parallel swap
       */
  }
 }
```

- The file bitonic_sort_single_block.cu sorts one block with a given block size

Merging

- The k sorted arrays are merged using binary search
- Each thread tries to find the correct position of a single element
- for an element i in block j , upperbound for 0 to j-1 blocks are summed up with lower bound for j+1 block onwards with with the index of element i to obtain the position in the final array
- The file bitonic_sort_multiple_blocks.cu sorts individual blocks and merge them to form the final sorted

Correctness check

- To check the correctness the non decreasing order of adjacent elements are compared

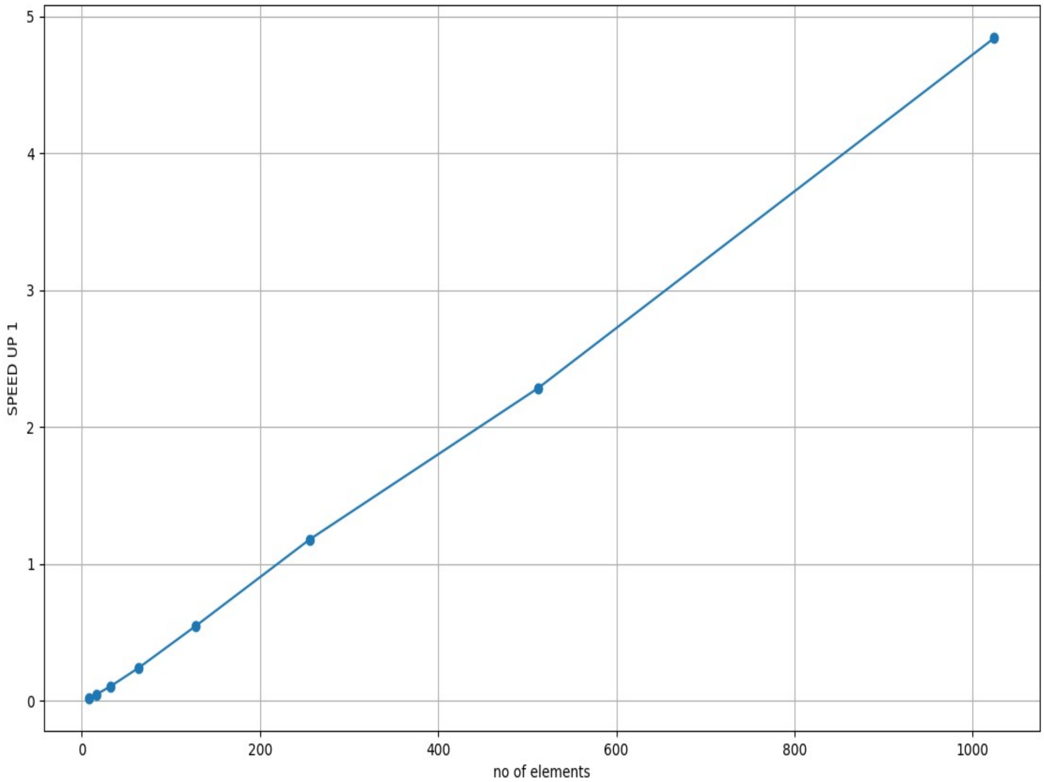
Results

For a single block:

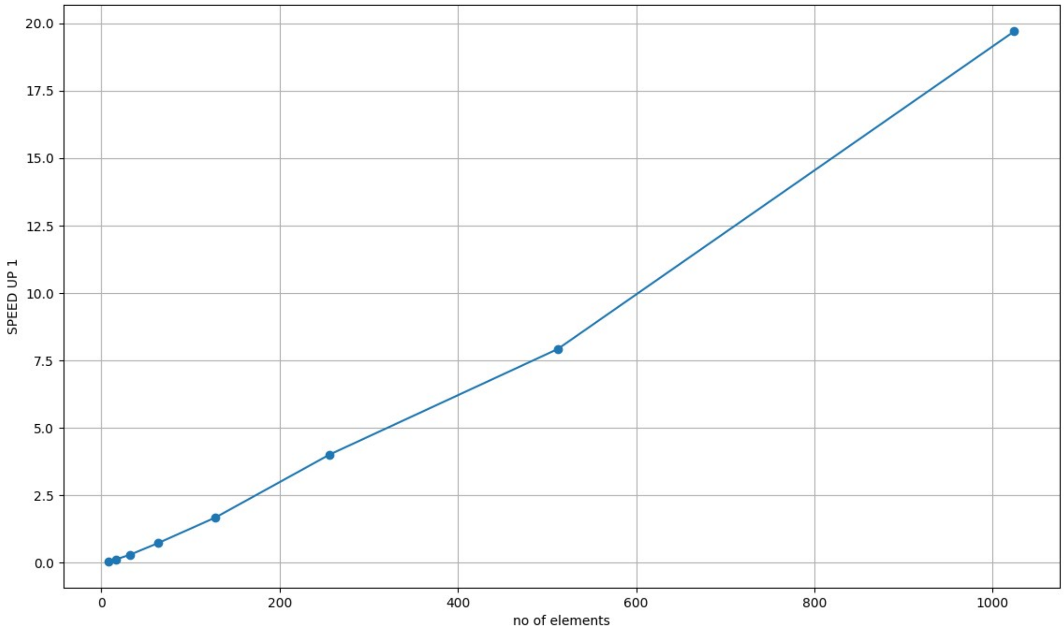
Size of array	Time for sequential program (in nano seconds)	Time for GPU program with memcopy (TIME 1) (in nanao seconds)	Speed up with respect to TIME 1	Time for GPU program without memcopy (TIME 1) (in nanao seconds)	Speed up with respect to TIME 2
8	1134.900000	50085.50000	0.022659	19500.20000	0.058199
16	2439.200000	52716.30000	0.046270	19863.90000	0.122796
32	6287.800000	59408.40000	0.105840	21166.50000	0.297064

64	14598.10000	60158.70000	0.242660	19841.60000	0.735732
128	33639.70000	61343.50000	0.548382	20038.50000	1.678753
256	78699.20000	66633.50000	1.181076	19618.50000	4.011479
512	161799.500	70805.40000	2.285129	20415.30000	7.925404
1024	394536.7000	81463.30000	4.843122	20025.30000	19.701912

No of elements versus speed up 1



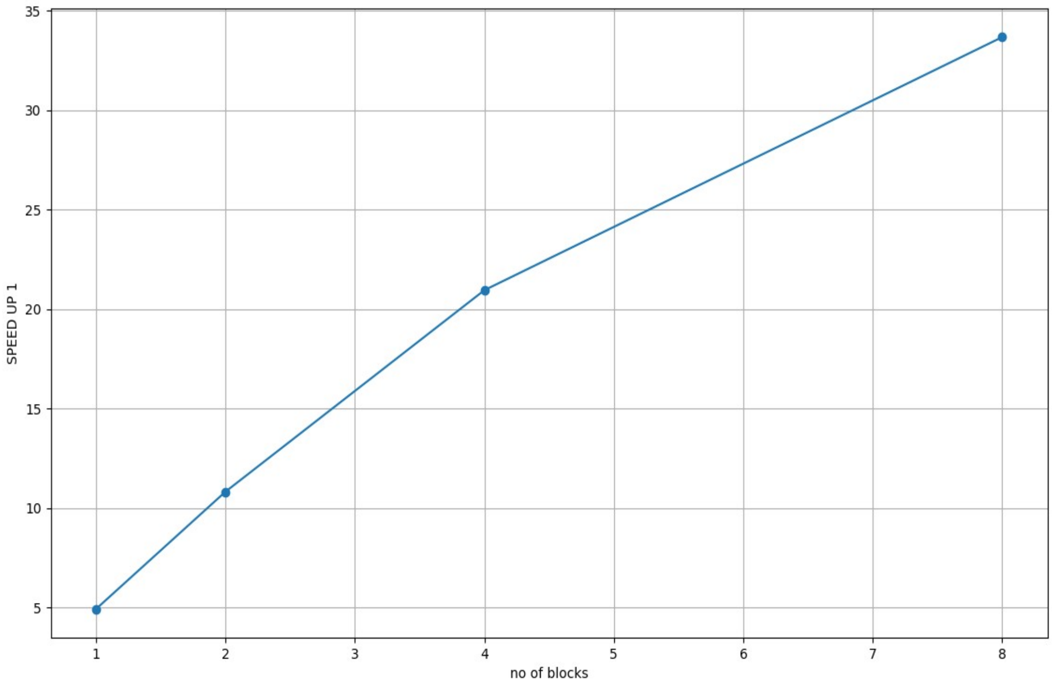
No of elements versus speed up 2



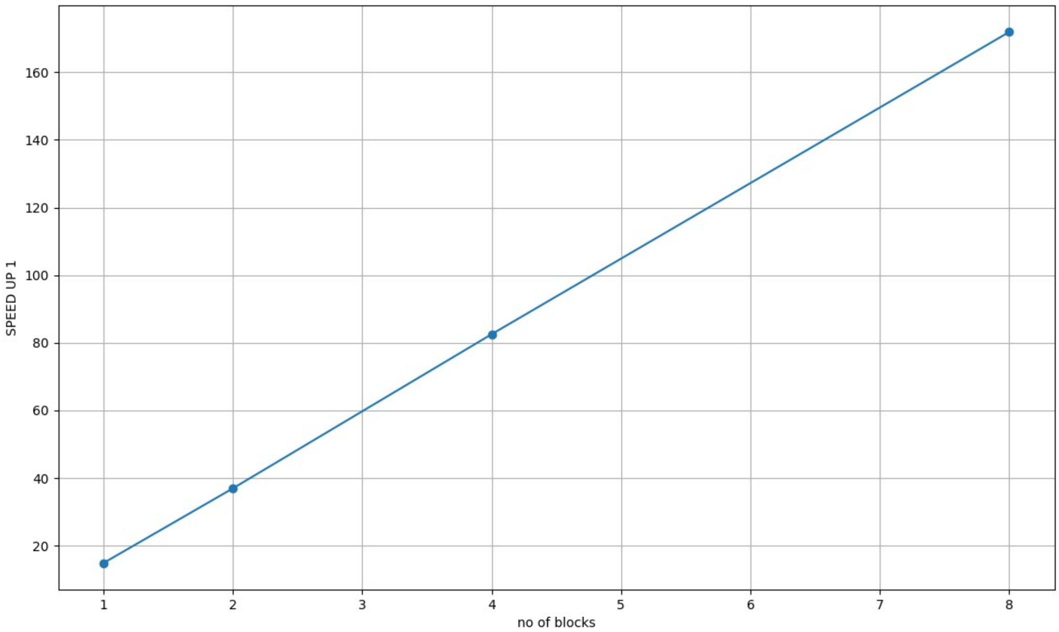
With Merging

No of blocks	Time for sequential program (in nano seconds)	Time for GPU program with memcopy (TIME 1) (in nanao seconds)	Speed up with respect to TIME 1	Time for GPU program without memcopy (TIME 1) (in nanao seconds)	Speed up with respect to TIME 2
1	383568.6000	77664.90000	4.938764	25743.20000	14.899803
2	956147.8000	88239.30000	10.835850	25865.10000	36.966716
4	2186792.900	104309.0000	20.964566	26495.20000	82.535437
8	4552330.600	135144.8000	33.684837	26488.50000	171.860641

No of blocks versus speed up 1



No of blocks versus speed up 2



Inference

- For a single block , with increase in number of elements , we get good speed up while for lesser number of elements the speed up is poor . But we can fit upto 1024 elements in a single block while give maximum speed up upto 4.8431 including memcopy
- In the next case with merging also with increase in number of blocks the speed up increases . For number of blocks 8, we get 33.68 speed up as compared to sequential
- It is also clear that we spend most of the time in copying data from host to device and from device to host. If we don't consider this copying time, we get really nice speed up which is visible from the table

Reference

- https://people.scs.carleton.ca/~dehne/teaching/4009/lectures/pdf/05c-Parallel_bitonic_sort.pdf