Parallel and Distributed Computing
HW6 - Shear and Bitonic Sort

Group No: 18

A. Aswin Sampath - 53844684 - saswin@uci.edu
B. Shreya Chetan Pawaskar - 12041645 - pawaskas@uci.edu

Consider an n-element integer number sequence S we wish to sort in non-decreasing order such that si ≤ si+1. For the purposes of this exercise, n is assumed to be a multiple of k. Show how the sequence S can be sorted in this cluster based on:

1. Shear Sort Algorithm.
2. Bitonic Sort Algorithm

Shear Sort Algorithm and Working :

Let's consider the following NxN matrix (N=4) below:

1	4	9	13
2	6	10	14
3	7	11	15
5	8	12	16

We first sort the matrix row wise with each alternate row in same direction

1	4	9	13
2	6	10	14
3	7	11	15
5	8	12	16

1	4	9	13
14	10	6	2
3	7	11	15
16	12	8	5

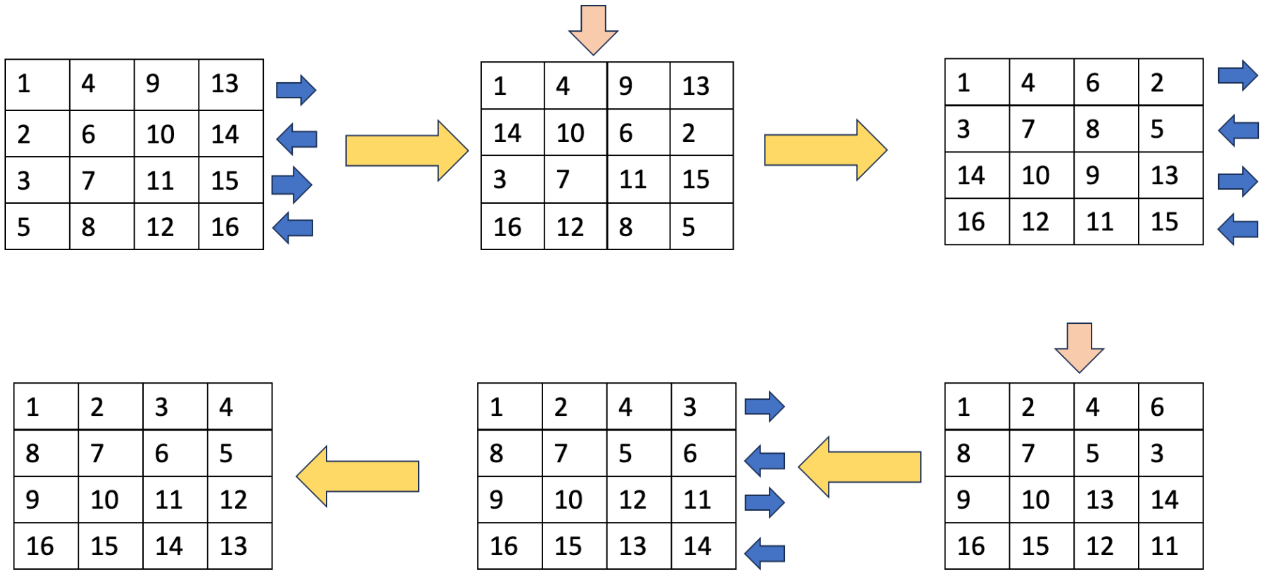
We then sort the matrix column wise , so the initial matrix now becomes :

1	4	9	13
2	6	10	14
3	7	11	15
5	8	12	16

1	4	9	13
14	10	6	2
3	7	11	15
16	12	8	5

1	4	6	2
3	7	8	5
14	10	9	13
16	12	11	15

We repeat the process to finally get the sorted matrix



In the Shear Sort algorithm, each processor in a system with k processors is assigned specific rows to sort based on a predetermined formula. For instance, with n integers to sort and k processors (forming a grid for n=16 integers and k=4 processors), each processor, denoted as Pi, is responsible for sorting the ith row.

Size of memory required by each processor is n//k

The number of compare-exchange operations during each phase of the sort is determined based on whether k is odd or even.

If k is odd, during the odd phase of sorting, k/2 operations occur, and the same number of operations occur during the even phase.

However, if k is even, k/2 operations occur during the odd phase, while $k/2-1$ operations occur during the even phase.

Let ri represent the number of rows allocated to a processor Pi (i ranges from 0 to $k-1$), o denote the number of odd operations, and e denote the number of even operations needed to fully sort a row. Consequently, the total number of compare-exchange operations executed by a processor can be calculated as follows:

When k is odd: $ri \times ((o+e) \times [k/2])$

When k is even: $ri \times ((o+e) \times [k/2] - e)$

For the given example of n=16 integers and k=4 processors, the number of compare-exchange operations a processor performs during the row phase would be $2 \times o + e$

During the column phase of the Shear Sort algorithm, each of the k processors is allocated specific columns from 0 to $k-1$ in a sequential manner. For instance, in a scenario with n=16 integers and k=4 processors, Processor 0 handles Column 0, Processor 1 handles Column 1, and so on.

The number of compare-exchange operations in each phase of sorting depends on whether n/k results in an odd or even number.

If n/k is odd, both the odd and even phases of sorting require $n/(2 \times k)$ operations.

However, if n/k is even, the odd phase requires n/(2×k) operations, while the even phase requires [n/(2×k)]−1 operations.

Let o represent the count of odd swaps and e represent the count of even swaps necessary to sort a column. As a result, the total number of compare-exchange operations per processor is computed as follows:

When n/k is odd: (o+e)×[n2×k]
When n/k is even: (o+e)×[n2×k]−e

For the given example with n=16 and k=4 each processor executes 2×o+2×e2×o+2×e compare-exchange operations during the column phase.

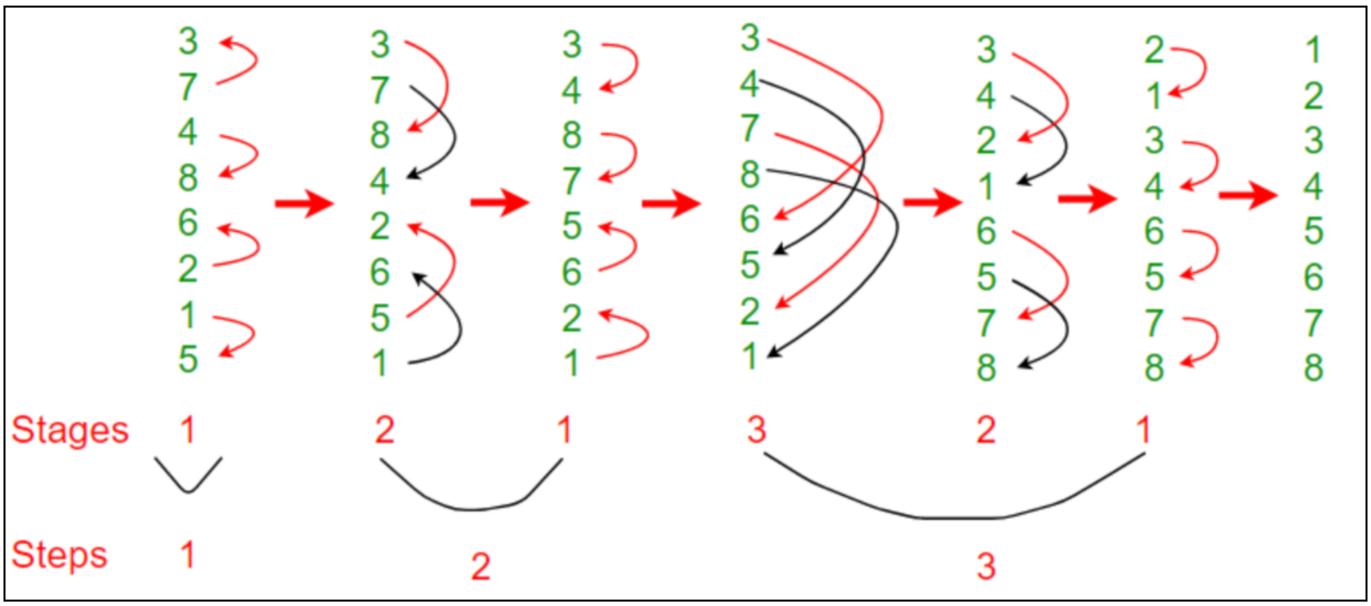
Time Complexity Analysis:

- Comparisons and Exchanges: Each processor performs a certain number of compare-exchange operations during the row and column phases. The number of these operations per processor is based on the size of the allocated rows and columns.
- Iterations: Shear Sort's convergence pattern involves multiple iterations to position elements correctly. For n elements distributed among k processors, Shear Sort typically takes O(log(n/k)) iterations to converge.
- Total Time Complexity: Combining the iterations and the operations within each iteration, the time complexity of Shear Sort can be expressed as **O(k×(log(n/k)+1))**, where: k is the number of processors., n is the number of elements to be sorted.

Bitonic Sort Algorithm and Working :

Bitonic sort is better for parallel implementation because we always compare elements in a predefined sequence and the sequence of comparison doesn't depend on data. Therefore it is suitable for implementation in hardware and parallel processor array.
A sequence is called Bitonic if it is first increasing, then decreasing. Bitonic Sort can only be done if the number of elements to sort is 2^n. A group is a contiguous part of the array that is monotonic. Subgroups are sections within a group with equal length. A 2n-sized array can be partitioned into 2m groups, each with 2n-m elements. For instance, an 8-element array can be subdivided into:

- 1 group of size 8
- 2 groups of size 4 each
- 4 groups of size 2 each
- 8 groups of size 1 each



Algorithm:

1. Bitonic sort sorts small subsequences in parallel.
2. These are called the bitonic sequences.
3. These are then recursively merged for a sorted array.
4. We divide S into n/k subsequences of length k, one per computer.
5. Each computer sorts locally using any O(nlogn) sorting algorithm like merge sort or heapsort.
6. This results in n/k sorted subsequences of length k.
7. We then merge the sorted sequences in log(k) merge steps.
8. In each step, we merge subsequences i and i+n/2k for i from 0 to n/2k-1 in parallel across computers.
9. This results in n/2k merged sorted sequences of length 2k.
10. After log(k) merge steps, we get a sorted array

Let's consider the following array S with **N=8** and **k=4** below:

4	2	70	1	88	3	6	5

S is initially divided evenly:

Processor	Block	
P0	4	2
P1	70	1
P2	88	3
P3	6	5

Each computer sorts its elements locally:

Processor	Block	
P0	2	4
P1	1	70
P2	3	88
P3	5	6

We then merge P0 with P2 and P1 with P3:

Processor	Block			
P0	2	3	4	88
P1	1	5	6	70

Finally, we merge P0 and P1 to get full sorted sequence on P1:

Processor	Result							
P0	1	2	3	4	5	6	70	88

Complexity :

We conclude the iterations when 2^i equals n, resulting in log^2n iterations. During each iteration i, we conduct i subgroupings, ranging from 2^(i-1) to 1.

This leads to a maximum of k * i potential compare-and-swap operations for each iteration, considering that each subgrouping involves, at most, 4 compare-and-swap operations for an array of length 8.

By applying the same formula, the final iteration executes at most k * log2n compare-and-swaps.

- Best Case: $O(log^2n)$
- Average Case: $O(log^2n)$
- Worst Case: $O(log^2n)$

Space Complexity: $O(n.log^2n)$

Analysis:

- **Comparisons:**
  - $O((log_2n)_2)$
- **Memory:**
  - 2 * sizeof(int)
- **Communications:**
  - Within a given subgroup, processors operate independently as each one handles a distinct task, corresponding to a different index pair.

- Communication is not necessary among processors in this scenario.
- However, between subgroups, processors may exchange index values.

In summary, both algorithms achieve O(nlogk) complexity for computation, memory, and communication when implemented on a crossbar interconnect cluster of k computers. Shear sort is a simpler approach while bitonic sort can have better constant factors.

References

1. Sent, Isaac D. Schersont Sandeep, and Adi Shamirf. "t Department of Electrical and Computer Engineering University of California." (https://www.cse.iitd.ac.in/~ssen/conf/ICPP86.pdf)
2. Mu, Qi & Cui, Liqing & Song, Yufei. (2015). The implementation and optimization of Bitonic sort algorithm based on CUDA.
3. M. F. Ionescu and K. E. Schauser, "Optimizing parallel bitonic sort," Proceedings 11th International Parallel Processing Symposium, Genva, Switzerland, 1997, pp. 303-309, doi: 10.1109/IPPS.1997.580914.
4. https://www.geeksforgeeks.org/bitonic-sort/
5. http://www.inf.ufes.br/~raulh/ufes/teaching/courses/pp/texts/bitonic.pdf
6. https://www.geeksforgeeks.org/bitonic-sorting-network-using-parallel-computing/
7. https://disco.ethz.ch/courses/podc_allstars/lecture/chapter4.pdf
8. K. Batcher and J. Lee, "Minimizing Communication in the Bitonic Sort" in IEEE Transactions on Parallel & Distributed Systems, vol. 11, no. 05, pp. 459-474, 2000. doi: 10.1109/71.852399
9. https://people.cs.rutgers.edu/~venugopa/parallel_summer2012/bitonic_overview.html
10. Sen, Sandeep & Scherson, Isaac & Shamir, Adi. (1986). Shear Sort: A True Two-Dimensional Sorting Techniques for VLSI Networks.. 903-908.
11. https://en.wikipedia.org/wiki/Massively_parallel_processor_array