Grenze International Journal of Engineering and Technology, June Issue

Parallelization of Cycle Sort Algorithm

Anjali Mathew[1], Prerana C Rao[2], N Gopalakrishna Kini[3] and Ashwath Rao B[4]
[1-4]Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal-576 104, Karnataka, India
Email: ng.kini@manipal.edu, {anjalimathew1999, craoprerana, ashwath.rao.b}@gmail.com

***Abstract*— Cycle Sort is a straightforward in-place sorting algorithm that is appropriate for certain use cases where limitations on memory or write costs are the main concern. It does this by minimizing the amount of writes to memory. Cycle Sort has an O(n^2) time complexity where *n* is the number of elements in the array to be sorted in both the worst and average cases. Inefficiency may result from this quadratic complexity, particularly for large datasets. This issue can be overcome by applying parallelization techniques to execute this cycle sort algorithm. In this paper, the parallelized solutions for cycle sort using the Message Passing Interface (MPI) programming and Compute Unified Device Architecture (CUDA) programming models are discussed. The elements are separated into smaller arrays, which are then subjected to the sorting cycles in the proposed methods. Until all of the elements in an array are sorted, cycles are applied to them. The findings emphasize the potential advantages of parallelization in reducing the overall computation time for cycle sorting. The outcome of this shows how scalable the parallelized cycle sort method for sorting huge datasets.**

***Index Terms*— Parallelization, CUDA, Cycle sort**

I. INTRODUCTION

Sorting is a method for putting the components in either descending or ascending order. Sorting algorithm's primary benefit is that they can aid in the methodical and effective organization and analysis of the data [1]. It will be simpler to interpret the data when it is provided in a sorted and filtered manner. Data searching is made easier and faster by sorting. Cycle sort is less effective with large arrays than other sorting algorithms like merge sort and quick sort. It is usually used with a constrained value range.

Cycle sort is a comparison-based algorithm that is best in regard to the quantity of arrays since it lessens the number of writes. The disadvantages of this algorithm is with its time complexity. Cycle sort is slower than other sorting algorithms, like merge sort and quick sort, for large arrays because it always has a quadratic time complexity. Cycle sort is an unstable sorting algorithm [3] which means that the relative order of equal elements in the array is not maintained. This is so because the algorithm can create a sorted array, which depends on the elements' ability to be rotated cyclically. The primary benefit of cycle sorting is that it uses less memory, which facilitates algorithm parallelization. This algorithm's core principle is to split the input data into cycles.

Every cycle comprises an element, and the elements are contained in an array as shown in Figure 1. A sequence of swaps is then carried out to arrange the items in the correct order. This procedure keeps going until every component in the array are sorted. This algorithm's primary benefit is that it doesn't need additional memory for variables [1]. A graph can be used to represent cycles. The array's items that are diametrically opposed to one another and that are present on the same circle are compared; if one element is discovered to be in the incorrect

Grenze ID: 01.GIJET.10.2.733

order, the elements are switched. The time complexity of the algorithm is O(n^2) for best, worst, and average cases.

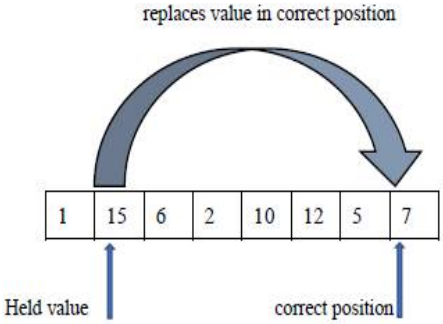

Figure 1. Working of cycle sort algorithm

Among the cycle sort algorithm's real-time applications are large datasets may be sorted and analyzed which will be helpful for data exploration [2]. This approach is frequently used to reduce database speed, such as when creating index and sorting the data for storing or retrieving the data. By classifying the objects according to their distance from the camera, this technique can be used to reduce the amount of time that 3D images are rendered. By organizing the data according to characteristics, the sorting algorithms can also be used to preprocess data for machine learning algorithms.

The main contribution in this paper is:

- To present how a parallel approach is investigated with cycle sort algorithm.
- To present how a novel MPI and CUDA based parallel approach is investigated in cycle sort ing algorithm to improve overall computation performance.
- To compute the speedup performance of parallel algorithm using MPI and CUDA implementation with classical implementation on single CPU core.

II. Related Work

Cycle sort that rely on a permutation's ability to be broken down into a product of cyclic permutations that are explained in [1]. Sorting can be completed in linear time when these algorithms are used under the right circumstances. These algorithms, which allow sorting to be completed in the linear time, are based on the decomposability of a permutation into a product of cyclic permutations and are applicable under certain conditions. The primary feature of the sorting algorithms described in the Cycle-Sort is that they sort in place and in linear time. This indicates that the algorithms are effective and appropriate in the scenarios where memory is constrained or making a copy of the data is not desired because they don't require extra memory space to carry out the sorting process.

The effectiveness of the modified cycle sort algorithm is covered in [2]. The modified cycle sort algorithm is compared to other sorting algorithms that are currently in use. It investigates the effectiveness and usefulness of the modified cycle sort algorithm in practical settings. Completing these tasks is crucial to comprehending new algorithm and how it might affect sorting procedures. The sorting algorithm is the central component of the issue that needs to be addressed.

In [3], author attempt to establish a new sorting algorithm and compare it with some other standard sorting algorithms by their execution times. Here the author addresses a methodology that can be executed on ordering data, random data, and on some specific data which can provide the best result for sorting.

In [4]-[7], the speeding up of few sorting algorithms are achieved using MPI and CUDA. These implementation techniques are referred in this work.

III. Methodology

The fundamental principle behind cycle sorting algorithm is input array is divided into cycles. Every cycle addresses the elements that belong to the same position in the output array. Within its cycle a series of swaps is done to place each element in its correct position. This is repeated until all cycles are complete and the array gets sorted.

A. Cycle sort as sequential

The sequential algorithmic steps of the cycle sort algorithm [8] are as in Figure 1.

Algorithmic steps of the Cycle sort:

Step 1: Consider a given array containing *n* elements.

Step 2: A variable, *cycleStart*, is initialized to 0.

Step 3: Each element is compared with every other element on its right in the array. The variable *cycleStart* is incremented by one if there are any elements that are smaller than the current element.

Step 4: After comparing the first element with all other elements, if the variable *cycleStart* = 0 then, move to the next element and continue with *step* 3.

Step 5: Swap the current element with the first element in its cycle if a smaller element is found. Continue the cycle until the current element returns to its original position.

Step 6: Repeat *steps* 3 through 5 until all cycles have been completed.

Figure 1. Cycle sort algorithm

B. Cycle sort as parallel using MPI

The main benefit of Cycle sorting is that it is inherently parallelizable, making it an ideal candidate for high performance computing environments. In the proposed parallel sorting algorithm, the array is divided into smaller sub arrays and each sub array is given a different processor. The cycle sort algorithm would then be used by each processor to sort the sub array that it was assigned. Intermittent results obtained from each processor are then iteratively merged until the entire dataset is sorted. Statistics distribution and merging techniques are optimized to limit verbal exchange overhead. Our set of rules additionally addresses load balancing demanding situations to make sure that no technique turns into a bottleneck for the duration of sorting.

Using the parallelization methodology, the cycle sort algorithm is divided into smaller steps, each of which is carried out on a separate processor. By decreasing the amount of time that each processor must wait for the other processors to complete their steps, this can enhance the algorithm's performance. The cycle sort algorithm is first broken down into number of steps to implement.

The steps involved with MPI environment are listed:

- Initialize the MPI environment.
- Scatter the input arrays in smaller chunks to other processors.
- Let each processor find the minimum element and place the element into its correct position.
- Gather the sorted local elements from all the processors to obtain the consolidated sorted array.

C. Cycle sort as parallel using CUDA

In CUDA, parallelism is achieved by dividing data into threads, each running on a separate CUDA core. Divide the input list into segments of equal size and assign each segment to a CUDA thread. Each thread independently sorts the data in its assigned segment using cycle sorting technique. When each segment of sub array is finished sorting, connect the sorted segments in parallel. This means comparing and merging adjacent elements between segments to ensure that the overall list is always more ordered. Repeat the process for the specified number of cycles of iterations until the entire array is sorted. The number of blocks per grid and number of threads per block are determined by the size of the input data and the kernel is executed accordingly. Once the data has been sorted and combined in parallel, it is moved back to the CPU from the GPU. Finally, allocated GPU memory is freed to prevent memory leaks. The challenge is to optimize the parallel strategy and memory usage to take full advantage of the GPU's parallel processing power, resulting in a more efficient implementation of Cycle Sort compared to its sequential counterpart.

The steps involved in CUDA:

- Transfer the input array from CPU to GPU memory.
- Launch the CUDA kernel that executes the cycle sorting.
- Every thread in kernel is assigned to handle some part of input array segments allocated to sort with cycle sorting.
- Sorted array segments by all the threads are transferred back to the CPU memory from GPU memory to obtain the consolidated sorted array.

IV. RESULTS

Cycle sorting algorithm is highly parallelizable, enabling the comparison and swapping of the multiple elements at once. When compared to conventional sequential CPU-based sorting algorithms, this leads to noticeably faster sorting. This can be utilized by sorting algorithms to process multiple elements in parallel, to achieve the high throughput, and shorten the total sorting time.

Table I and Table II shows the results of parallelization of cycle sort using MPI and CUDA respectively. The sequential execution time required for cycle sort is compared with the time required for parallel cycle sort using MPI and CUDA. The Speedup achieved on parallelization for the same is computed.
In MPI environment the data is shared between multiple processes, which enables simultaneous cycle sorting and efficient use of available processing resources. Each process sorts its assigned data elements independently.

TABLE I. EXECUTION TIME AND SPEEDUP FOR CYCLE SORT IN SEQUENTIAL AND MPI

Tests	Sequential Execution time (ms)	Number of processes	MPI Execution time (ms)	Speedup
1	17.07	2	4.57	3.73
2	17.42	4	1.31	13.29
3	17.28	6	0.99	17.45
4	17.29	8	0.44	39.29
5	17.59	10	0.36	48.86

TABLE II. EXECUTION TIME AND SPEEDUP FOR CYCLE SORT IN SEQUENTIAL AND CUDA

Tests	Sequential Execution time (ms)	CUDA Execution time (ms)	Speedup
1	17.07	0.012	1417
2	17.42	0.014	1244
3	17.28	0.013	1329
4	17.29	0.01	1729
5	17.59	0.019	925

By utilizing the parallel nature of the algorithm and numerous cores of the GPU, CUDA can greatly increase the algorithm's performance. The benefits of parallelism become more apparent as dataset sizes rise. Compared to sequential sorting algorithms, CUDA enables sorting algorithms to scale effectively with larger datasets, offering superior performance on large data sizes.
The parallel implementation of Cycle sort using MPI and CUDA has given an average speedup of 24.53 and 1329 respectively with a significant reduction in execution time.

V. CONCLUSIONS

Parallel computing is increasingly used to meet the growing computing needs of data- driven applications and large data sets. Parallel cycle sort demonstrates the effectiveness of parallelization in improving the overall performance of sorting operations.
In this paper, Cycle sorting in parallel using MPI and CUDA are achieved by breaking the problem down into smaller tasks and then distributing these small tasks across multiple processes or processing units to take advantage of the parallelism. A speedup over traditional sequential methods is achieved with increased execution speed and shorter execution time.
In this work, the dataset is distributed among different processors. By using MPI programming, it offers an effective way to sort the datasets. Datasets are also sorted very effectively with CUDA since it can scale too many GPU cores. A speedup over traditional sequential methods is achieved with increased execution speed and shorter execution time.

REFERENCES

[1] B. K Haddon, “Cycle sort: A Linear sorting Method,” University of Colorado at Boulder, vol. 3(4), 1999.
[2] Nakib Aman Turzo, Pritom Sarker, Biplob Kumar, Jyotirmoy Ghose, Amit Chakraborty, “Defining a Modified Cycle sort algorithm and Parallel Critique with other cycle sorting algorithms,” GRD Journals-Global Research and Development Journal for Engineering, vol. 5(5), pp. 1-8, 2020.
[3] Md. Hasan Imam Bijoy, Md. Rakibul Hasan, Masad Rabbani, “A New Comparative study and better solution of Sorting algorithm for array,” 11th International Conference on Computing, Communication and Networking Technologies, IEEE, Kharagpur, 2020.

[4] Anaghashree, Sushmita Delcy Pereira, Rao B. Ashwath, Shwetha Rai, and N. Gopalakrishna Kini, “Super sort algorithm using MPI and CUDA,” Intelligent Data Engineering and Analytics: Frontiers in Intelligent Computing: Theory and Applications (FICTA 2020), vol. 2, pp. 165-170, Springer Singapore, 2020.
[5] Yadav, Harshit, Shraddha Naik, B. Ashwath Rao, Shwetha Rai, and Gopalakrishna Kini, “Comparison of CutShort: a hybrid sorting technique using MPI and CUDA,” Evolution in Computational Intelligence Frontiers in Intelligent Computing: Theory and Applications (FICTA 2020), vol.1, pp. 421-428, Springer Singapore, 2020.
[6] Karthik, C. R., Ashwin G. Shanbhag, B. Ashwath Rao, Prakash K. Aithal, and Gopalakrishna N. Kini, “Parallelization of Cocktail Sort with MPI and CUDA,” Proceedings of the International Conference on Paradigms of Communication, Computing and Data Sciences: PCCDS 2021, pp. 231-242, Springer Singapore, 2022.
[7] Raghunandan, B. Aishwarya, B. Ashwath Rao, Prakash K. Aithal, and Gopalakrishna N. Kini, “A Parallel Implementation of FastBit Radix Sort Using MPI and CUDA,” Electronic Systems and Intelligent Computing: Proceedings of ESIC 2021, pp. 1-13, Springer Nature Singapore, 2022.
[8] https://www.geeksforgeeks.org/cycle-sort/ , last accessed 2024/03/07.