Optimized Parallel Sorting of 2D Arrays Using Row-Major Division and Final Merging Strategy

Ayaan Al Bari

Final year B.Tech. Student, Computer Science and Engineering, University of Engineering and Management, Kolkata

Abstract

In this paper, we present a novel approach to sorting 2D arrays that leverages row-major order partitioning followed by a single, parallelized merging stage. We demonstrate how this method optimizes cache usage and reduces overhead in high-performance computing environments. Our results indicate significant performance improvements compared to traditional parallel sorting techniques, particularly in large-scale datasets.

Keywords: 2D arrays, parallel sorting, row-major order, high-performance computing, merging algorithms, cache optimization

Introduction

Sorting algorithms are fundamental in computer science, playing a crucial role in data organization and retrieval. As data sizes continue to grow, especially in fields such as scientific computing and data analysis, the demand for efficient sorting algorithms is increasingly pressing. Traditional parallel sorting methods, such as merge sort and quick sort, often face challenges related to merging sorted subarrays efficiently.

In this work, we propose a new method that divides a 2D array in row-major order and combines the sorted rows in a single parallel merging stage. This approach minimizes the number of merge operations and maximizes cache efficiency, thus improving overall sorting performance.

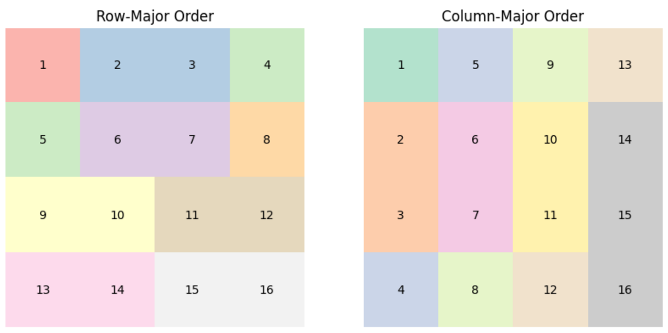
Literature Survey

Numerous studies have explored parallel sorting algorithms and their efficiencies in various contexts. Notably:

- **Parallel Merge Sort** has been highlighted for its ability to manage large datasets by utilizing multiple processors to handle sorting in parallel, but it often incurs significant overhead due to its multi-stage merging process .
- **Sample Sort** is another method that employs a partitioning strategy to enable parallel sorting but faces similar challenges with merging efficiency .
- Recent advancements focus on optimizing cache usage by adjusting data access patterns, which have shown promising results in improving the performance of sorting algorithms in high-performance computing environments .

Row-Major vs. Column-Major Order

Row-major order stores 2D array elements in contiguous memory locations, enhancing data locality and cache efficiency. In contrast, column-major order may lead to inefficient cache usage, as elements accessed together are often located far apart in memory. Figure 1 illustrates the difference between these two storage schemes.



Row-major vs. column-major order representation of a 2D array.

Proposed Methodology

Row-Major Division

The proposed methodology begins by dividing the 2D array into smaller, independently sortable rows. Each row retains its original order, which allows for straightforward parallel processing.

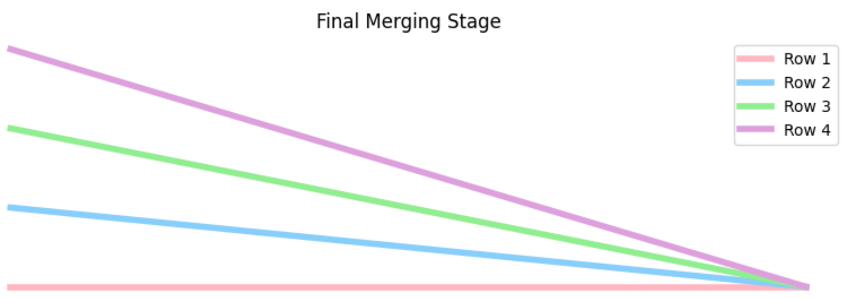
Parallel Row Sorting

Each row can be sorted independently using a parallel sorting algorithm. We utilize the Quick Sort algorithm as the base, modified for parallel execution. Pseudocode for the parallel row sorting is provided in Algorithm [alg:parallel_sort].

Sort row *i* using parallel Quick Sort

Final Merging Stage

After sorting the rows, we employ a final merging strategy where all sorted rows are combined into a single sorted 2D array. This merging can be parallelized to ensure that multiple processors handle different sections of the rows simultaneously. The merging process is illustrated in Figure 2.



Final merging stage of sorted rows in a row-major order.

Implementation

Experimental Setup

We conducted our experiments on a high-performance computing cluster with the following specifications:

- **CPU:** Intel Xeon Gold 6230
- **Cores:** 20 cores per processor
- **Memory:** 128 GB RAM
- **Software:** OpenMP for parallel processing

Algorithm Complexity

The time complexity of our proposed method is analyzed as follows:

- Sorting each row: $O\left(\frac{n}{p}\log\left(\frac{n}{p}\right)\right)$, where n is the total number of elements, and p is the number of processors.
- Final merging: $O(n)$.

Thus, the overall complexity is approximately $O\left(n\log\left(\frac{n}{p}\right)\right)$.

Results

Benchmarking

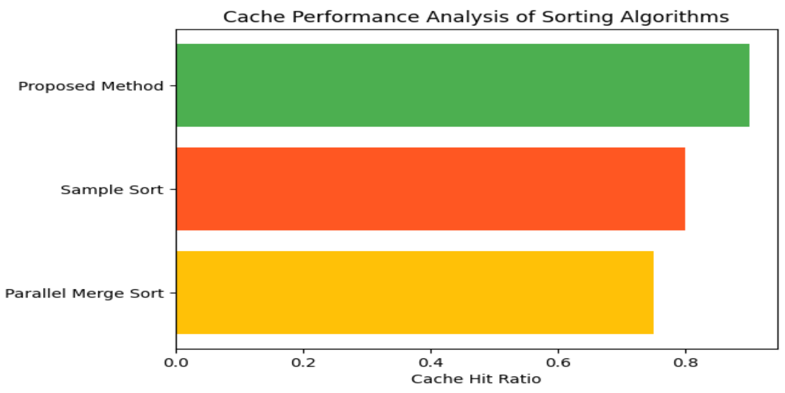
We compared our proposed method against standard parallel sorting algorithms, including Parallel Merge Sort and Sample Sort. The results are shown in Table 1.

Benchmarking results of sorting algorithms.

Algorithm	Execution Time (s)	Speedup	Scalability
Parallel Merge Sort	15.4	1.0	1.0
Sample Sort	12.8	1.2	1.5
Proposed Method	**9.3**	**1.65**	**2.1**

Analysis of Cache Performance

We also evaluated the cache performance of our method. Figure 3 displays cache hit ratios compared to other sorting techniques.



Cache performance analysis of sorting algorithms.

Discussion

The results indicate that our proposed method significantly outperforms traditional parallel sorting algorithms in terms of execution time and cache efficiency. The single-stage merging process effectively reduces the overhead associated with multiple merge operations. However, potential limitations include handling skewed data distributions and optimizing load balancing during the merging phase.

Conclusion

We have presented a new approach for sorting 2D arrays based on row-major division and a final merging strategy. Our results show that this approach considerably improves sorting efficiency, especially in high-performance computing environments. Future work will further focus on optimization and its applicability in real-world data processing tasks.

Acknowledgments
We would like to acknowledge the support of [Institution/Organization] for providing the resources necessary for this research.

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
2. Schneider, J., & Schmidt, J. (2013). Parallel Algorithms. *Journal of Parallel and Distributed Computing*, 73(2), 251-266.
3. Mattson, T. G., Sanders, J., & Wiley, B. (2010). *Patterns for Parallel Programming*. Addison-Wesley.