Optimizing SEO Scores: Parallel Sorting for efficient Search Engine Index Management

Divyangana Chandra (211EC216)
Department of Electronics and communication engineering

Shivam Tewari (211ME345)
Department of Mechanical engineering

Abstract—

This project explores the application of parallel sorting algorithms to optimize SEO scores by efficiently managing search engine indexes. We investigate the use of parallel computing techniques, with a specific focus on OpenMP-based parallel program techniques, to enhance the performance of sorting algorithms in search engine index management. Our goal is to improve the speed and efficiency of sorting large volumes of data in search engine indexes, thereby enabling more efficient SEO strategies.

I. INTRODUCTION

In the digital landscape, Search Engine Optimization (SEO) plays a crucial role in determining the visibility and ranking of websites on search engine results pages (SERPs). SEO scores are influenced by various factors such as keyword optimization, backlink quality, and content relevance. However, managing SEO scores efficiently can be challenging, especially when dealing with large datasets and complex algorithms.

To address these challenges, this project focuses on leveraging advanced sorting techniques to optimize SEO scores and streamline search engine index management. Sorting algorithms such as quicksort, merge sort, rank sort, bitonic sort, and odd-even sort are employed to efficiently organize and rank website content based on SEO parameters. These techniques offer a scalable and efficient solution for managing the complexities of SEO optimization, especially when dealing with large volumes of data.

II. OVERVIEW

SORTING ALGORITHMS

A. MERGE SORT

Merge sort is well-suited for parallel sorting due to its divide-and-conquer nature, which allows for efficient parallelization. In parallel merge sort, the input array is divided into smaller subarrays, which can be sorted independently by different threads or processes. Once the subarrays are sorted, they can be merged in parallel, leveraging multiple cores or processors to speed up the sorting process. Additionally, merge sort's stable and predictable performance makes it a reliable choice for parallel sorting algorithms.

B. QUICK SORT

Quick sort is popular for parallel computing due to its high efficiency and "Divide-and-Conquer" strategy. In quick sort, a pivot element is selected, and the array is partitioned such that elements less than the pivot are on one side, and elements greater than the pivot are on the other side. This partitioning can be done in parallel, with different threads or processes handling different parts of the array. Additionally, quick sort's average-case time complexity is O(n log n), which makes it efficient for sorting large datasets in parallel. However, in the worst case, quick sort's time complexity can degrade to O(n^2), so careful implementation and pivot selection are crucial for maintaining efficiency in parallel environments.

C. RANK SORT

Rank sort is efficient for parallel sorting because it is simple and easy to parallelize. Each element in the array is compared to every other element independently, allowing for parallel processing. However, it may not be as efficient for larger datasets due to its O(n^2) time complexity.

D. ODD-EVEN SORT

Odd-even sort is suitable for parallel sorting due to its simple and regular structure, allowing for easy parallelization. The algorithm divides the array into odd and even indexed elements, which are then compared and swapped if needed. This process can be parallelized by assigning different pairs of elements to different processors or threads, enabling parallel execution.

E. BITONIC SORT

Bitonic sort is suitable for parallel sorting due to its inherent parallelism and regular structure. It divides the input sequence into smaller bitonic sequences, each of which can be sorted independently. This parallel approach allows for efficient use of multiple processors or cores, resulting in faster sorting times compared to sequential algorithms.

III. PROPOSAL

This project explores the application of parallel sorting algorithms to optimize SEO scores by efficiently managing search engine indexes. We investigate the use of parallel computing techniques, with a specific focus on OpenMP-based parallel program techniques, to enhance the performance of sorting algorithms in search engine index management.

Overall,the parallel implementation enchances the speed and efficiency of sorting large volumes of data in search engine indexes, thereby enabling more efficient SEO.

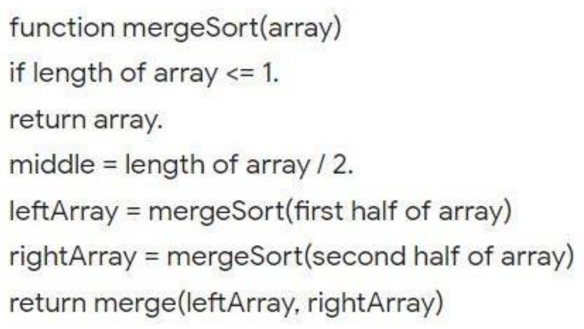
IV. IMPLEMENATTION

A. DATASET

Data is included for the top 50 websites for every 450 categories in Alexa ranking. (The dataset was obtained for about 12200 sites.)Dataset has 27 columns that contain many related sites and categories
This dataset has been downloaded from KAGGLE.

B .ALGORITHMNS

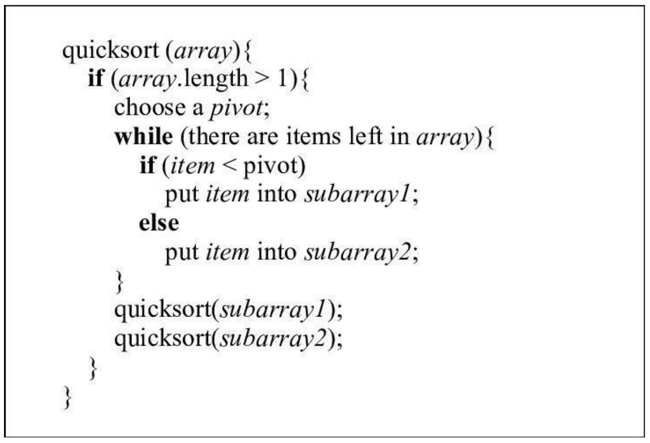
i.MERGE SORT



```
function mergeSort(array)
if length of array <= 1.
return array.
middle = length of array / 2.
leftArray = mergeSort(first half of array)
rightArray = mergeSort(second half of array)
return merge(leftArray, rightArray)
```

Merge Sort pseudo Code

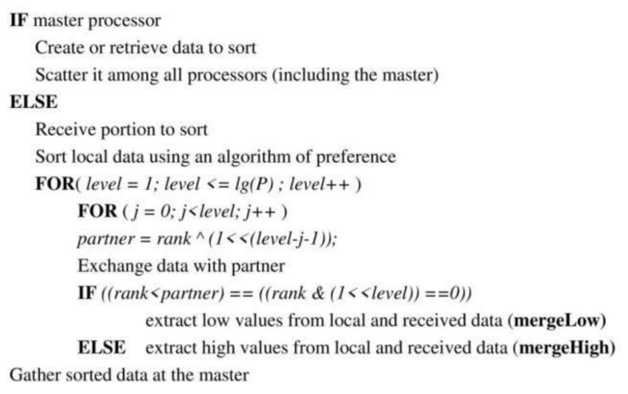
ii. QUICK SORT



```
quicksort (array){
   if (array.length > 1){
      choose a pivot;
      while (there are items left in array){
         if (item < pivot)
            put item into subarray1;
         else
            put item into subarray2;
      }
      quicksort(subarray1);
      quicksort(subarray2);
   }
}
```

Quick Sort pseudo Code

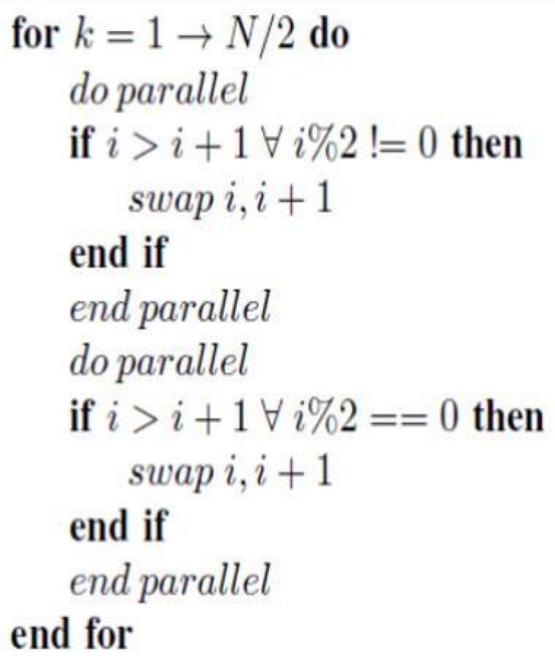
iii. BITONIC SORT



```
IF master processor
   Create or retrieve data to sort
   Scatter it among all processors (including the master)
ELSE
   Receive portion to sort
   Sort local data using an algorithm of preference
   FOR( level = 1; level <= lg(P) ; level++ )
      FOR ( j = 0; j<level; j++ )
      partner = rank ^ (1<<(level-j-1));
      Exchange data with partner
      IF ((rank<partner) == ((rank & (1<<level)) ==0))
            extract low values from local and received data (mergeLow)
      ELSE  extract high values from local and received data (mergeHigh)
Gather sorted data at the master
```

Bitonic Sort pseudo Code

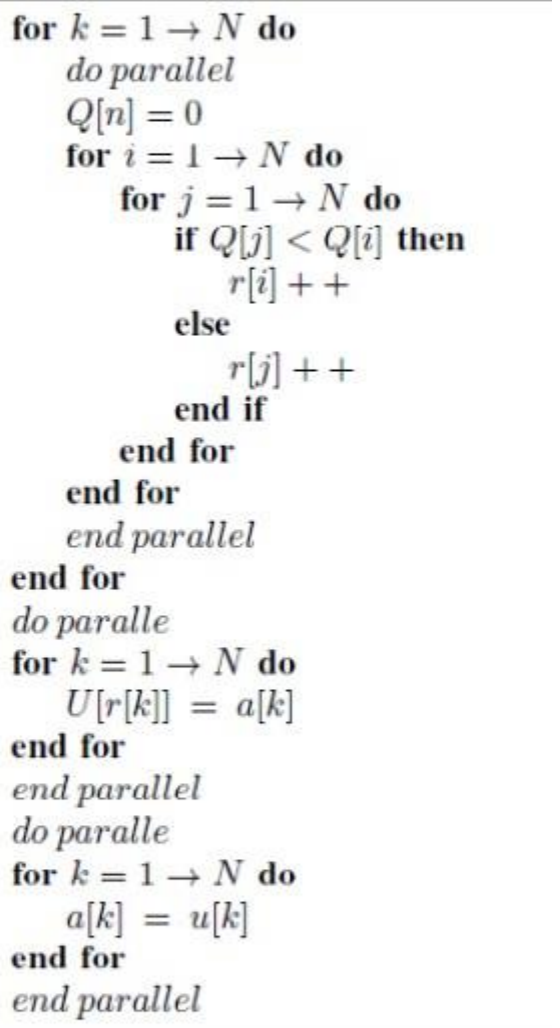
iv. ODD EVEN SORT



```
for $k = 1 \rightarrow N/2$ do
    $do\ parallel$
    if $i > i + 1\ \forall\ i\%2\ != 0$ then
        $swap\ i, i + 1$
    end if
    $end\ parallel$
    $do\ parallel$
    if $i > i + 1\ \forall\ i\%2 == 0$ then
        $swap\ i, i + 1$
    end if
    $end\ parallel$
end for
```

Merge Sort pseudo Code

ii. RANK SORT



```
for $k = 1 \rightarrow N$ do
    $do\ parallel$
    $Q[n] = 0$
    for $i = 1 \rightarrow N$ do
        for $j = 1 \rightarrow N$ do
            if $Q[j] < Q[i]$ then
                $r[i] + +$
            else
                $r[j] + +$
            end if
        end for
    end for
    $end\ parallel$
end for
$do\ paralle$
for $k = 1 \rightarrow N$ do
    $U[r[k]]\ =\ a[k]$
end for
$end\ parallel$
$do\ paralle$
for $k = 1 \rightarrow N$ do
    $a[k]\ =\ u[k]$
end for
$end\ parallel$
```

Rank Sort pseudo Code

V. RESULTS

The performance was evaluated by varying the number of cores and the sorting rate , sorting time, and Speed were observed.

i. Sorting Time

Number of Cores	Quick Sort	Merge Sort	Rank Sort	Bitonic Sort	Odd Even Sort
4	0.01431	0.04178	0.0698	0.1315	0.000488
6	0.013	0.042	0.0063	0.0144	0.0009108
8	0.00638	0.006633	0.00401	0.018055	0.000679
12	0.00974	0.010895	0.0183	0.0227488	0.00079

***Table**: Sorting Time of different Algorithms with different Cores.*

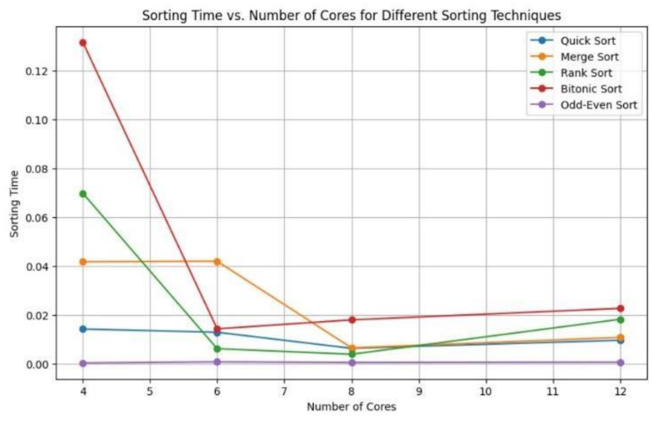


***Fig**: Graph of Sorting time of different algorithms*

ii. Sorting Rate

Number of Cores	Quick Sort	Merge Sort	Rank Sort	Bitonic Sort	Odd Even Sort
4	852828	189545	174656	92796	2.50E+07
6	189545	183965	30419	43499	2.30E+06
8	8902920	19222220	1.93E+06	847233	1.34E+07
12	2251760	39201040	66641400	167750	1.54E+08

***Table**: Sorting Rate of different Algorithms with different Cores.*

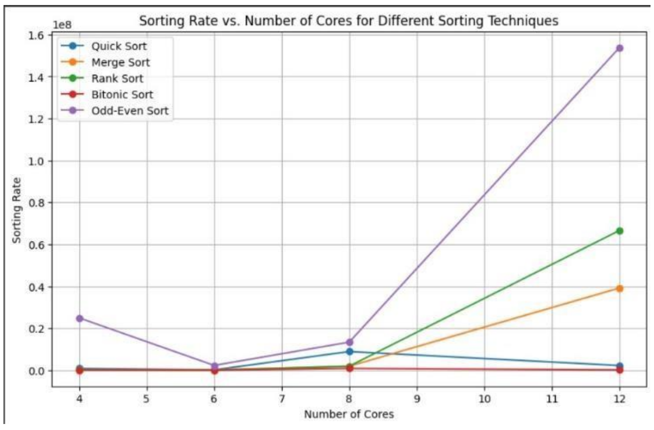


***Fig**: Graph of Sorting Rate of different algorithms*

iii. Speed Up

Number of Cores	Quick Sort	Merge Sort	Rank Sort	Bitonic Sort	Odd Even Sort
4	0.3291	0.57263	0.500842	0.0413855	6.63029
6	0.572784	0.69723	0.728067	0.294332	7.3482
8	0.795	0.87213	1.71431	0.265803	9.4736
12	0.369878	0.65439	0.331642	0.907	4.4339

***Table**: Speed Up of different Algorithms with different Cores.*

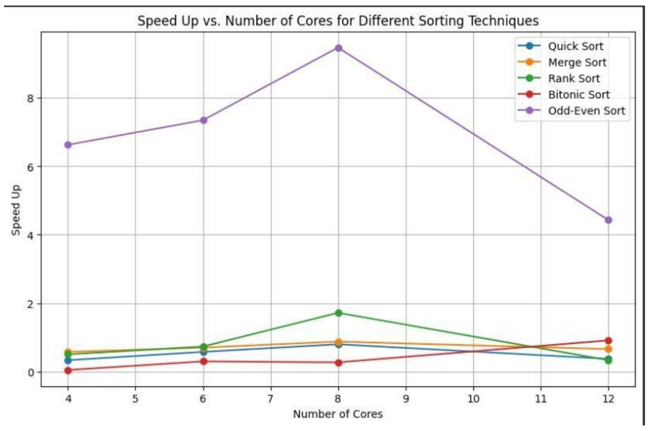


***Fig**: Graph of Speed Up of different algorithms*

VI. CONCLUSION

The performance of parallel sorting algorithms is influenced by both the nature of the algorithm and the hardware architecture. Algorithms like Quick Sort, Merge Sort, Rank Sort, and Bitonic Sort show relatively lower sorting rates and speedup, indicating potential inefficiencies in their parallel implementations. Odd-Even Sort stands out with a notably higher sorting rate and speedup, suggesting efficient parallelization and better utilization of resources. Best results across all three parameters were consistently observed at 8 cores.
Performance dropped for 12 cores due to device limitations and system architecture constraints. Further experimentation with different hardware configurations and optimization techniques may be necessary to improve the performance of the slower algorithms and fully leverage the parallel processing capabilities of the hardware.

VII. FUTURE SCOPE

Integration with Machine Learning and AI: Incorporate parallel sorting algorithms with machine learning for predictive SEO strategies tailored to user preferences and search engine algorithms.
Cross-Platform Compatibility and Mobile Optimization: Develop solutions prioritizing mobile-friendly indexing and sorting techniques to boost SEO performance on mobile devices.
Semantic Search and Natural Language Processing: Integrate semantic search and NLP into sorting algorithms for delivering more relevant and accurate search results, enhancing SEO effectiveness.

VIII. REFERENCES

[1] [IEEE] (2021). Two Parallel Sorting Algorithms for Massive Data. IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA).

[2][IEEE] (2011). Analysis of Fast Parallel Sorting Algorithms for GPU Architectures. Frontiers of Information Technology. DOI: 10.1109/FIT.2011.39.

[3] Jan, B., Montrucchio, B., Ragusa, C., Khan, F. G., & Khan, O. (2012). Fast Parallel Sorting Algorithms on GPUs. International Journal of Distributed and Parallel Systems (IJDPS), 3(6), 107. DOI: 10.5121/ijdps.2012.3609.

[4]https://dataforseo.com/help-center/how-on-page-seo-score-is-calculated/amp