Enhanced Merge Sort on Multi-core Architecture

Boby Gupta¹, Om Prakash Pal².

¹ Computer Science and Engineering Department, Bharat Institute of Technology Meerut
Partapur By-Pass Meerut India.
Boby.gupta89@gmail.com.

² Computer Science and Engineering Department, Bharat Institute of Technology Meerut
Partapur By-Pass Meerut India.
oppal.info@gmail.com

Abstract: In this paper, a comparative analysis of performance of three different types of sorting algorithms viz. merge sort, 3-way merge sort and parallel merge sort is presented. The merge sort algorithm to sort a sequence of n elements is based on divide and conquers approach of solving problems. The 3-way merge sort is also a sorting algorithm which follows divide and conquer approach and is an extension of simple merge sort. The comparative analysis is based on comparing average sorting time in parallel sorting over merge sort and 3-way merge sort. The time complexity for each sorting algorithm will also be mentioned and analyzed.

Keywords: Sorting, Algorithm, Merge Sort, 3-Way merge Sort, Multi-core architecture, Parallel merge sort, Average running time, Performance.

1. Introduction: The word “algorithm” comes from the name of a Persian author, Abu Ja far Mohammed ibn Musa al Khwarizmi, who wrote a textbook on mathematics. An algorithm is a way to accomplish any task in step by step process. Basically, algorithm is an approach to do any work efficiently and logically and at the same time it gives the sense to work without confusion and without being puzzled. Algorithms is a set of rules for carrying out calculation either by hand or by machine. Every Algorithm have these five properties:

1. Input
2. Output
3. Definiteness
4. Effectiveness
5. Termination

Therefore an Algorithm can be define as a sequence of definite and effective instructions , while terminates with production of correct output from the given input.

Algorithm is step-by-step procedure for solving a problem[2], which is guaranteed to terminate after a finite number of steps. In other words algorithm is logical representation of the instructions which should be executed to perform meaningful task. For a given problem, there are generally many different algorithms for solving it. Some algorithms are more efficient than others, in that less time or memory is required to execute them. The analysis of algorithms studies time and memory requirements of algorithms and the way those requirements depend on the number of items being processed. Sorting is generally understood to be the process of rearranging a given set of objects in a specific order and therefore, the analysis and design of useful sorting algorithms has remained one of the most important research areas in the field. Despite the fact that, several new sorting algorithms being introduced, the large number of programmers in the field depends on one of the comparison-based sorting algorithms: Bubble, Insertion, Selection sort etc[1]. Hence sorting is an almost universally performed and hence, considered as a fundamental activity. The usefulness and significance of sorting is depicted from the day to day application of sorting in real-life objects. For instance, objects are sorted in: Telephone directories, income tax files, tables of contents, libraries, dictionaries. The methods of sorting can be divided into two categories:

INTERNAL SORTING: If all the data that is to be sorted can be adjusted at a time in main memory, then internal sorting methods are used.
EXTERNAL SORTING: When the data to be sorted cannot be accommodated in the memory at the time and some has to be kept in auxiliary memory (hard disk, floppy, tape etc), then external sorting method are used. The complexity of a sorting algorithm measures the running time of function in which „n“ numbers of items are sorted. The choice of which sorting method is suitable for a problem depends on various efficiency considerations for different problem.

Sorting is the one of the core computational algorithms used in many scientific and engineering applications. Some sorting algorithms are applied for small number of elements and some are used for floating point number, and some are used for large number of data. We sort data in either numerical order or in lexicographical order, sorting numerical values either in decreasing order or in increasing order. Sorting has two different meaning ordering and categories, ordering means order the of same items and categorizing means grouping and labeling the same types of items. Sequential sorting algorithm are classified into two categories. The first categories is “comparison sort”, is based on comparing the data items to find the correct relative order. The second one is “distribution sort” is based on distributing the unsorted data items to multiple intermediate structures, which are collected and sorted into a single into a single sorted list.

2. Merge Sort Algorithm: This algorithm was invented by John von Neumann in 1945. It closely follow divide and conquer paradigm. Conceptually it works as follows:[1]

- **Divide**: Divide the unsorted list into two sub lists of about half of the size.
- **Conquer**: Sort each of the sub lists recursively until we have list size of length 1, in which case the list itself is returned.
- **Combine**: Merge the two–sorted sub lists back into one sorted list.

Merge sort takes advantage of the ease of merging already sorted lists into a new sorted list. It starts by comparing every two elements (i.e., 1 with 2, then 3 with 4...) and swapping them if the first should come after the second. It then merges each of the resulting lists of two into lists of four, then merges those lists of four, and so on; until at last two lists are merged into the final sorted list. Of the algorithms described here, this is the first that scales well to very large lists, because its worst-case running time is O(n log n). Merge sort has seen a relatively recent surge in popularity for practical implementations,

- **Pros:**
  - Time complexity is O(n log n).
  - It can be used for internal and external sorting.

- **Cons:**
  - At least twice the memory requirements of the other sorts because it is recursive.
  - Space complexity is high.

There are some examples of merge sort is given below:
Fig 1: Graphical representation of the Merge sort algorithm

Fig 2: Working procedure of merge sort
Fig 3: Working procedure of merge sort

**Analysis:** In order to analyze the Merge Sort function, we need to consider the two distinct processes that make up its implementation. First, the list is split into halves. We divide a list in half \( \log n \) times where \( n \) is the length of the list. The second process is the merge. Each item in the list will eventually be processed and placed on the sorted list. So the merge operation which results in a list of size \( n \) requires \( n \) operations. The result of this analysis is that \( \log n \) splits, each of which costs \( n \) for a total of \( n \log n \) operations.

- **Best Case** = \( O(n \log n) \)
- **Average Case** = \( O(n \log n) \)
- **Worst Case** = \( O(n \log n) \)


**Output:** Arr[low…….high] in ascending order.
3. **3-Way Merge sort:** 3 way merge sort is an efficient divide and conquer sorting algorithm. It is an extension of simple merge sort algorithm. The sequence is divided into three parts and then generated sub lists are further divided until each number is obtained individually. The numbers are then merged together as a pair to form sorted subsequently until the whole sequence is constructed.

![Graphical representation of 3-Way Merge Sort](image)

**Pseudo code:**

**Input:** Arr[low] show beginning element and Arr[high] show the ending element.

**Output:** Array[Low ....high] in ascending order.
```java
private void ThreeWayMergeSort(T[] to, T[] temp, int low, int high) {
    if (low >= high) return;
    var mid1 = (high-low) / 3;
    var mid2 = 2*mid1;
    ThreeWayMergeSort(temp, to, low, mid1);
    ThreeWayMergeSort(temp, to, mid1 + 1, mid2);
    ThreeWayMergeSort(temp, to, mid2 + 1, high);
    ThreeWayMerge(to, temp, low, mid1, mid2, high);
}
```

Fig 6: pseudo code of 3-Way Merge Sort

Fig 7: Recursion Tree of 3-way Merge Sort
4: **Parallel Merge sort:** Multi core architecture have 2, 4 or 8 cores on a single die. Chip multiprocessors (CMP) is a multithreaded architecture, which integrates more than one processor on single chip. In this architecture, each processor has its own L1 cache, the L2 cache and bus interfaces are shared among processors. **Intel core 2 Duo** [12] is an example of such architecture; it has a two processor on a single chip, each of them has L1 cache, both of them are sharing L2 cache.

Multi-core architecture are designed to provide a high performance feature on a single chip. In parallel merge sort a large array [3] is partitioned into equal parts and efficient sorting function is applied on sub array in parallel, then parallel execution results in faster processing. It takes less time to merge all sorted array. Most basic construct for task parallelism is:

Parallel .Invoke (Do Left, DO Right); it execute the method Do Left and Do Right in Parallel and waits for both to finish. Invoke is a synchronous method it will only return when it has executed all tasks. This method provide a simple way in which a number of tasks may be created and executed in parallel. As with other methods in parallel task library, Parallel .Invoke provide potential parallelism.

**Pseudo code:**

**Input:**
Arr[low] is the beginning element and Arr[high]is the ending element.

**Output:**
Arr[low...high] in ascending order.
private void ParallelMergeSort(T[] to, T[] temp, int low, int high, int depth)
{
    if (high - low + 1 <= SEQUENTIAL_THRESHOLD || depth <= 0)
    {
        MergeSort(to, temp, low, high);
        return;
    }
    var mid = (low + high) / 2;
    depth--;
    Parallel.Invoke(
        () => ParallelMergeSort(temp, to, low, mid, depth),
        () => ParallelMergeSort(temp, to, mid + 1, high, depth)
    );
    ParallelMerge(to, temp, low, mid, mid + 1, high, low, depth);
}

parallel .Invoke construct execute the parallel merge sort into two sub arrays, partitioned around mid element in parallel manner. Merging of two sub array can also be performed by invoking Parallel.Invoke for parallel execution Which is given below:

private void ParallelMerge(T[] to, T[] temp, int lowX, int highX, int lowY, int highY, int lowTo, int depth)
{
    ......................

    if (lengthX < lengthY)
    {
        ParallelMerge(to, temp, lowY, highY, lowX, highX, lowTo, depth);
        return;
    }
    ......................
    ......................
    Parallel.Invoke(
        () => ParallelMerge(to, temp, lowX, midX - 1, lowY, midY - 1, lowTo, depth),
        () => ParallelMerge(to, temp, midX + 1, highX, midY, highY, midTo + 1, depth)
    );
}
6. **Comparison of Results:** The performance of merge sort, 3-way merge sort and parallel merge sort can be analyzed by considering the running time. The table shows the average running time of merge sort, 3-way merge sort and parallel merge sort with respect to increasing number of inputs. It represents the parallel merge sort using Parallel.Invoke perform better over merge sort and 3-way merge sort due to the use of parallelism of both CPUs.

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Input Size (n)</th>
<th>Average Running Time (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Merge sort</td>
</tr>
<tr>
<td>1.</td>
<td>1 X 10³</td>
<td>16.28</td>
</tr>
<tr>
<td>2.</td>
<td>1 X 10⁴</td>
<td>16.30</td>
</tr>
<tr>
<td>3.</td>
<td>1 X 10⁵</td>
<td>16.55</td>
</tr>
<tr>
<td>4.</td>
<td>1 X 10⁶</td>
<td>16.56</td>
</tr>
<tr>
<td>5.</td>
<td>5 X 10⁶</td>
<td>17.58</td>
</tr>
<tr>
<td>6.</td>
<td>10 X 10⁶</td>
<td>18.99</td>
</tr>
<tr>
<td>7.</td>
<td>15 X 10⁶</td>
<td>20.47</td>
</tr>
<tr>
<td>8.</td>
<td>20 X 10⁶</td>
<td>21.94</td>
</tr>
<tr>
<td>9.</td>
<td>25 X 10⁶</td>
<td>23.36</td>
</tr>
</tbody>
</table>

Table 1. Average running time of merge sort, merge sort and parallel merge sort
Graph of Figure 11 shows the better performance of three way merge sort over two way merge sort. Three way merge sort has lower average running time than two way merge sort.

Fig11. Merge Sort vs Parallel Merge Sort
Graph of Figure 12 shows the better performance of Parallel merge sort over two way merge sort. Parallel merge sort has lower average running time than two way merge sort due to parallelism and use of multiple cores.

![Graph of Figure 12](image)

Fig 12. Merge Sort vs. Parallel Merge Sort.

Graph of figure 13 shows the better performance of parallel merge sort over Three way merge sort. Three way merge sort has higher average running time than parallel merge sort due to the absence of parallelism.
Graph of Figure 14 shows the comparison of all three sorting algorithms viz. merge sort, three way merge sort and parallel merge sort. From the three sorting algorithms parallel merge sort performs best which is obvious due to parallelism. Three way merge sort performs better in comparison to two way merge sort as stated above.
7. **Speed Up**: In the field of computer architecture, **speedup** is a metric for relative performance improvement when executing a task. The notion of speedup was established by Amdahl's law, also known as Amdahl’s argument [13], used to find the maximum improvement to an overall system, when only part of the system is improved. However, speedup can be used more generally to show the effect of any performance enhancement. Speedup is only useful when the experimental data is run on the same system.

\[
S = \frac{P_{\text{new}}}{P_{\text{old}}}
\]

Where

- \( S \) is the resultant speedup.
- \( P_{\text{old}} \) is the old performance, i.e., without the improvement.
- \( P_{\text{new}} \) is the new performance, i.e., with the improvement.

Graph of Figure 15 shows the speed up achieved by Three way merge sort with an average of 1.41. Also as the number of inputs increases it fluctuates along its average value.
Fig 15. Speed Up achieved in 3-way merge sort

Graph of Figure 16 shows the speed up achieved by Parallel merge sort with an average of 1.96. Also as the number of inputs increases it fluctuates along its average value.
Fig 16. Speed Up achieved in Parallel Merge Sort

Graph of Figure 17 shows the comparison of speedup achieved by parallel merge sort and Three way merge sort. Since parallel merge sort uses the multiple cores of CPU it achieves more speedup than three way merge sort.
8. **Conclusion:** In this paper, the sorting problem for large data sets is considered, and three sorting algorithms viz. Merge sort, Three way merge sort and Parallel merge sort are compared successfully. The effect of the number of cores on the performance of merge sort has been theoretically and experimentally studied. Based on theoretical modeling and experimental results, we found that

- Parallel merge sort has the best performance; as it sorts the subsequences in a parallel manner which reduces the average running time of sorting the data values. As the number of CPU cores increases, the average running time will reduced.
- Three way merge sort performs better than sequential merge sort due to the reduced number of calls to merge sort function on the subsequences. Also the running time is reduced due to base of log function to 3.
- Both parallel merge sort and Three way merge sort achieves speedup over sequential merge sort. The Speedup of Parallel Merge sort is greater in comparison to Three way Merge sort which is obvious due to the use of multiple cores.

Concerning future work, same analysis can be performed with parallel sorting algorithms (parallel quick sort and Hyper-quicksort) and parallel sorting by regular sampling algorithm (PSRS) for wide variety of MIMD architectures and the processors with more than two cores.
9. References:

[7] R. Kumar, V. Zyuban and D. Tullsen "Interconnections in multi-core architectures:

About Author: Boby Gupta D/O Mr. Mukesh Chandra Gupta is pursuing M-Tech (CSE) From Bharat Institute of Technology Partapur road By-Pass, Meerut, Uttar Pradesh, India. Her research area is Enhanced Merge Sort on Multi-core Architecture.