

# A New Parallel Quicksort with Efficient Processor Allocation and Minimal Communication

Sangman Moh, Shinil Kim, and Minkyu Lee

August 20, 1999

## Abstract

In this term project, we design a new parallel quicksort algorithm and implement it on the Cray T3E parallel computer. Then, we measure the performance of the proposed algorithm and compare it with that of the conventional parallel quicksort algorithm. According to our comparative evaluation, the proposed algorithm is more efficient than the conventional one in viewpoint of performance. This effect is mainly due to the efficient processor allocation (partition) and minimized communication, which are key ideas of our algorithm.

## 1. Introduction

In sequential sorting, the best performance is bounded to  $O(n \log n)$  which is achieved for two well-known algorithms: mergesort and quicksort. In general, the performance of sorting algorithms is indicated as average performance. The worst-case performance of mergesort is  $O(n \log n)$  and it is approximately the same as the average performance. On the other hand, the best-case performance of quick sort is  $O(n \log n)$  and it is proven that it is the same as the average performance. The worst-case performance of quicksort is  $O(n^2)$  [1].

For parallel computer systems, some parallel sorting algorithms have been newly developed. On the other hand, the parallelized version of the sequential sorting algorithms has been also researched and used more actively rather than newly-developed parallel sorting algorithms [2]. Hence, we focus on the parallelized algorithm of sequential sorting. In particular, our work is been concentrated on the quicksort which is popular and effectively used in many computing areas.

In this term project, we have designed and evaluated a new parallel quicksort algorithm, which is suitable for distributed-memory multiprocessor systems. We have implemented the proposed algorithm in C language using MPI APIs, and run it on the Cray T3E parallel computer at ETRI Supercomputer Center. Also, we have measured the performance of the proposed algorithm and compared it with that of the

conventional parallel quicksort algorithm.

According to our comparative evaluation, the proposed algorithm is more efficient than the conventional scheme. This is due to the more balanced processor allocation and less amount of message communication compared to the conventional one.

## 2. New Parallel Quicksort Algorithm

Basically, the quicksort is based on divide-and-conquer method, which consists of partition and merge. In particular, the partition is the major time-consuming part of the quicksort, whereas the merge phase is very simple. The conventional parallel quicksort algorithm is well described in [2], and an example of the algorithm is shown in Figure 1.

The key idea of our proposed algorithm is the efficient partition of numbers, which enables better load balancing and less communication during the quicksort. The input numbers are initially taken by the master processor. By default, the master processor has the lowest processor identifier (*i.e.* 0) among the participating processors.

Let the number of inputs which are less than the pivot and the number of inputs which are greater than the pivot be  $NL$  and  $NR$ , respectively. Let the partition composed of inputs which are less than the pivot and the partition composed of inputs which are greater than the pivot be  $LP$  and  $RP$ , respectively. Then, at each level of recursive tree operations, the proposed parallel quicksort algorithm operates as follows:

- (1) Partition the processors into two partitions (groups) by the ratio of  $NL$  to  $NR$  in increasing order of processor identifier;
- (2) Send the smaller of  $LP$  and  $RP$  to the first processor in the other partition excluding the host processor;

And the other parts of the proposed algorithm are the same as the conventional quicksort algorithm.

We have implemented this algorithm using MPI APIs, as given in the Appendix of this report. This parallel algorithm minimizes the communication cost in the quicksort. In worst case, in particular, this scheme remarkably reduces the amount of messages to be transferred between processors as shown in Figure 2. Moreover, due to the efficient processor allocation (partition), the parallelized work is more balanced among participating processors. This also shortens the run time of the proposed parallel quicksort.

Intuitively, we may easily infer that (i) for best-case input patterns, the proposed algorithm has the same performance as the conventional one, (ii) for worst-case input patterns, the proposed algorithm outperforms the conventional one, and (iii) for average-case input patterns, the proposed algorithm also outperforms the conventional one. In general, since the performance of sorting algorithms is indicated as

average performance, we can argue that our algorithm is better than the conventional parallel quicksort algorithm.

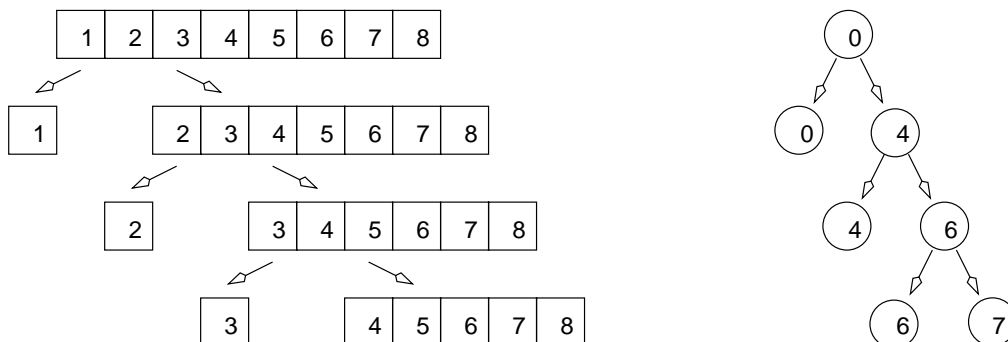


Figure 1: An example of conventional parallel quicksort

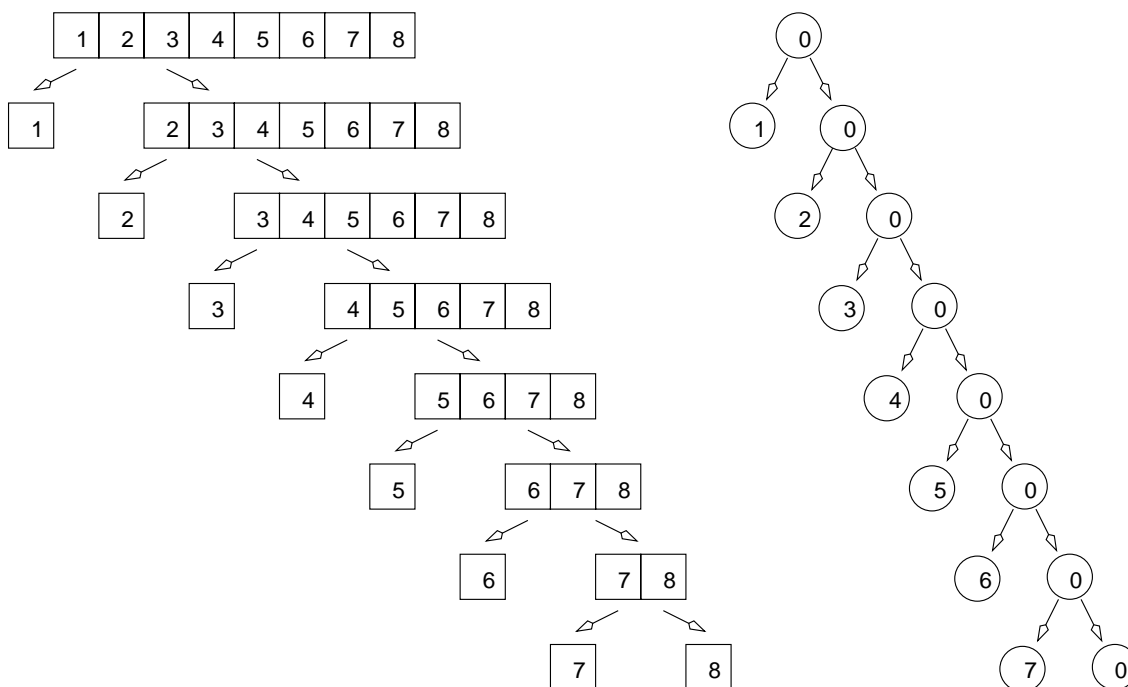


Figure 2: An example of proposed parallel quicksort

### 3. Comparative Performance Evaluation

In order to evaluate the performance of the proposed parallel quicksort algorithm and compare it with that of the conventional algorithm, we have implemented and run both the conventional parallel quicksort algorithm and our proposed algorithm on the Cray T3E parallel computer system. We have measured the run time of the two parallel quicksort algorithms, and compared them each other.

In our practical measurement, the input patterns are randomly generated and then the performance metric of execution time is measured by in-line timing check function inserted into the quicksort program. The execution time of two parallel quicksort algorithms is given in Table 1, which has been measured for input size of 10,000,000. As shown in Figure 3, the proposed parallel quicksort algorithm sorts the same size of problem in less time than the conventional parallel quicksort algorithm. However, if the small number of processors (less than 4) is used, the performance gain is minimized or negligible.

Table 1: Execution time of parallel quicksort algorithms (input size = 10,000,000)

# of nodes	2	4	8	16	32	64
Conventional algorithm	8.046776	7.518168	6.093498	4.837848	4.416389	4.102324
Proposed algorithm	7.944448	7.648544	5.374566	3.709797	3.272716	2.753625

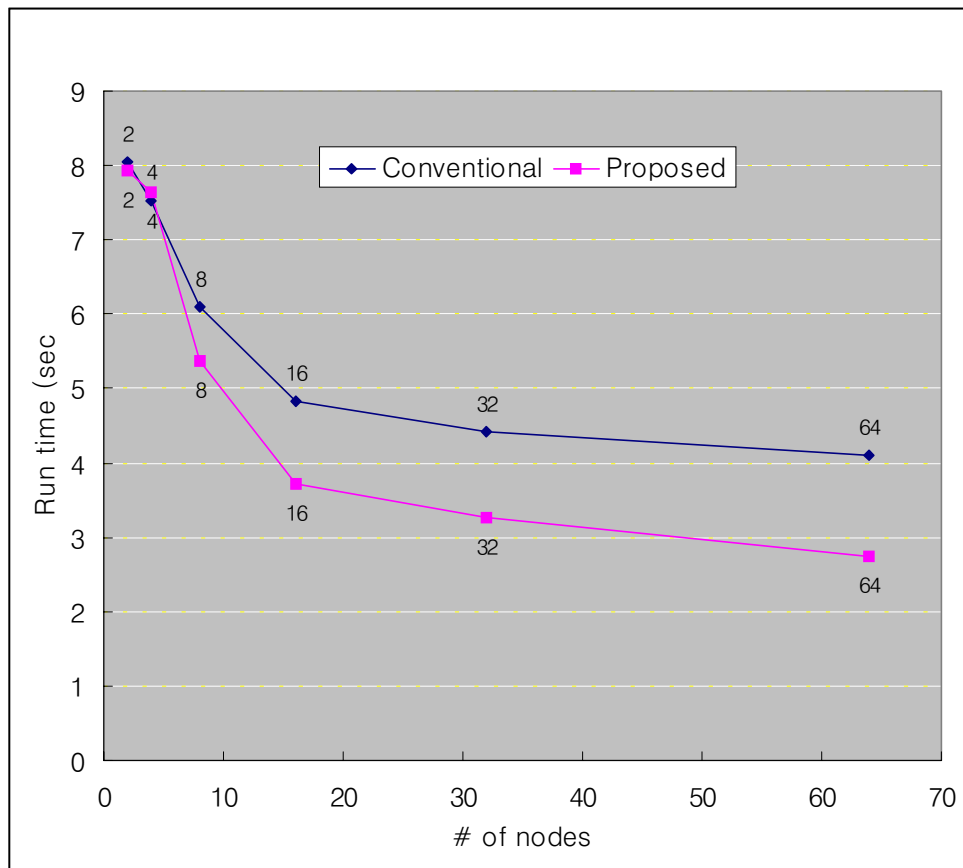


Figure 3: Performance comparison of two parallel quicksort algorithms (input size = 10,000,000)

## 4. Conclusion

In this paper, we have proposed a new parallel quicksort algorithm and implemented, and run it on the Cray T3E parallel computer. Then we have evaluated its performance and compared it with the conventional parallel algorithm with respect to run time. Conclusively, our algorithm outperforms the conventional one, which is primarily due to both the efficient problem partition/processor allocation and the minimized amount of communication.

## References

- [1] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts, 1994.
- [2] B. Wilkinson and M. Allen, *Parallel Programming*, Prentice-Hall, Inc., Upper Saddle River, New Hersey, 1999.

## Appendix

**A.1** Conventional parallel quicksort program

**A.2** Proposed parallel quicksort program