# Hypercube 컴퓨터에서의 Sorting 알고리즘에 대한 비교연구

○이 흥 규 차 혜 경 김 병 국

한 국 과 학 기 술 대 학 전 자 전 산 학 부

# COMPARATIVE STUDY OF SORTING ALGORITHMS ON HYPERCUBE MACHINE

Heung-Kyu Lee, Hye-Kyung Cha, and Byung-Kook Kim

School of Electrical Engineering and Computer Science, Korea Institute of Technology

## ABSTRACT

We discuss several issues related to the sorting algorithms on hypercube machine. The efficiencies of the algorithms are compared under the simpler assumption and the load balancing problem arising from the various algorithms and the data distribution is also discussed.

## I. INTRODUCTION

With the vast development in the computer hardware technology, there are many architectures for parallel computers which can be classified according to their connection topologies[1]. Among them the hypercube structure is now being studied widely and constructed as one of the most versatile parallel computers. Along with the topological properties of the hypercube structure[2] which allow simple and deadlock-free routing, there has been effort to accommodate the already existing algorithms for numerical and nonnumerical problems or to devise more efficient algorithms on the hypercube.

For the numerical problems with regularity in such fields as matrix algebra and partial differential equations, there are results on the time complexity in connection with the communication overhead which affects the overall performance of the algorithms[3]. However, for the problems like sorting there have not been many algorithms examined on the hypercube in this context even though there are many outcomes on some other concurrent computation models like shared memory and network computation models.

The sorting algorithms proposed on the concurrent computer in common have the assumption that there are enough processors ;each of which has one data item[4]. It is not always feasible that the number of processors available outnumbers that of sorting data. Thus we consider the sorting problem with N data items, where $N = n*p$, $n$ is the number of data items assigned to each processor, $n \geq 1$ and $p$ is the number of nodes in hypercube machine.

The paper is organized as follows. In section 2, we give the measure of time complexity for parallel algorithms and the computation model on which the algorithms are executed. In section 3 we present several sorting algorithms which are readily implementable on the hypercube with their time complexities. In section 4, we compare the efficiencies of these algorithms concluding with discussion in section 5.

## II. SORTING PROBLEM

Before investigating parallel sorting algorithm on hypercube machine, let us consider the optimal bound of the sorting algorithm on the parallel architecture. Since the best sequential algorithm has $\Theta(N\log N)$ timing bound at the average case, $O(\frac{N\log N}{p}) = O(n\log n + nD)$ is optimal for the parallel sorting algorithm, where nD is the term for the communication between the nodes[5]. We investigate various sorting algorithms such as bitonic sort, quick sort, and radix sort for small order of time complexity at the average case. Since sorting is a communication-intensive computation at parallel models, it is naturally desirable to minimize the communication needs to reduce the order.

In the model we used, the host computer has independent broadcasting channels to the nodes[6]. We use this one for the cube model under consideration since it has the advantage of accessing data in any pattern. In this model there is no shared or common memory for node-to-node communication. Thus the communication uses the message passing method. We assume that the message is passed without deadlock. The message path length is $[\log p = \log 2^D = D]$ where $p = 2^D$, at worst case regardless of the topology reconfiguration. If the corresponding node does not exist, the reconfiguration mapping algorithm[7], will provide proper structure.

We define two parameters $t_{flop}$ and $t_{comm}$, where $t_{comm}$ = minimal time necessary to pass a single packet through a given node, and $t_{flop}$ = time required to execute a simple expression(including floating point operation) within a node processor. Besides them, $t_{search}$ ($t_{compare}$ and $t_{sort}$) is used for a time unit to search(compare and sort respectively) the data. The value of each time unit can be measured in various ways[3], but we take $t_{comm} \approx 60$ μsec and $t_{flop} \approx 40$ μsec.

## III. SORTING ALGORITHMS ON HYPERCUBE MACHINE

### 3.1. Parallel Bitonic Sort Algorithm

Since the bitonic merging algorithm[4] can be readily implemented on the hypercube structure, we derive a sorting algorithm based on the bitonic merging algorithm. Initially the data items are distributed evenly onto each node of the cube. Let S be a sequence of N data items to be sorted where $N = n*p$ as before. For convenience, a sequence of n data items is called a n-sequence. First each node of the cube sorts its data using the best sequential sorting algorithm. Then the algorithm merges the p sorted n-sequences iteratively to generate the sorted sequence according to the predetermined ordering of the nodes.

The bitonic merge operation can be done in $D = \log p$

phases. At the $k$*th* phase, k = 1, ..., D, each pair of $2^{k-1}$n -sequences which is bitonic is merged concurrently. Since the cube of dimension D consists of $2^{d-k}$ subcubes of dimension k, the subcube of dimension k can be considered to have a sorted $2^k$n-sequence after the $k$*th* phase. Note that the data on each node can be arranged such that the data exchange be done only between neighboring nodes.

The steps of $k$*th* phase can be described as follows:

**step 1 :**

Do the first step of bitonic merging for each pair of subcubes of dimension k-1. Then each subcube of dimension k-1 will contain a bitonic $2^{k-1}$n-sequence.

**step 2 :**

If each subcube contains a bitonic n-sequence then merge the sequence in O(n). Otherwise iteratively apply bitonic merging to the bitonic sequence on each subcube of dimension k-1.

For the time complexity, the recurrence relation $T(k) = T(k-1) + \frac{n}{2}t_{compare} + 2nt_{comm}$, k = 2, ...,D and $T(1) = \frac{3n}{2}t_{compare} + 2n\,t_{comm}$ hold. It can be easily shown that $T(k) = \frac{n}{2}(k+2)t_{compare} + 2knt_{comm}$, k = 1, ..., D. The total time taken is then $n(\log n)t_{sort} + O(nD^2)t_{compare} + O(nD^2)t_{comm}$. Thus the order of the algorithm is $O(n\log n + nD^2)$, if $t_{sort}$, $t_{compare}$, and $t_{comm}$ have the same time unit. Then the algorithm is optimal with respect to the cost when $D < \sqrt{\log n}$.

### 3.2. Parallel Quicksort Algorithm

Distribute the N data items onto the p nodes evenly. The initial condition is that each node has n data items. The parallel quicksort algorithm is as follows:

**step 1 :**

Break the hypercube into two subcubes, U and L. The lowest indexing node in the lower subcube selects the median value $m$. And broadcast the value. (Time $t = i*t_{comm}$ for i = 1, ..., D).

**step 2 :**

The data items with the greater value than the $m$ are moved to the upper subcube and smaller to the lower. (Time $t = n*t_{compare} + n*t_{comm}$).

**step 3 :**

Repeat the steps 1 and 2 until the subcube consists of a single node.

**step 4 :**

Each node sorts the given data using the best sequential sorting algorithm(Time $t = n(\log n)*t_{sort}$).

Now consider the time to be taken for the broadcasting and data transfer. The hypercube with D dimension requires the iteration of D times. The time to be taken for the step 3 is $\sum_{i=1}^{D} i*t_{comm} + nDt_{compare} + nDt_{comm} = \frac{D(D+1)}{2}*t_{comm} + nD(t_{compare} + t_{comm})$. Therefore, the total computing time of the above algorithm is $n(\log n)*t_{sort} + (\frac{D(D+1)}{2} + nD)*t_{comm} + nDt_{compare}$.

### 3.3. Shell Sort Algorithm

This sorting algorithm is a modified bitonic sorting algorithm. Examing the bitonic sort algorithm, most of data are in the right place after a few bitonic merge steps. Thus, the shell sort algorithm tries to reduce the merge step as follows:

**step 1 :**

Sort data items within each node.

**step 2 :**

Do naive merge between the node[8]. One step for each dimension of the cube.

**step 3 :**

Do modified odd-even transposition merge[5]. Repeat step 3 until global order is found.

The time taken is: $n(\log n)t_{sort} + 2nD(t_{compare} + t_{comm}) + T(n, D)$. The last term is for step 3. Step 3 continues until no more data items are exchanged. In practice T(n, D) is becomes small constant since global order is found in only a few steops of odd-even transposition. Thus it is neglected in obtaining computing time for the estimate convenience.

### 3.4. Index Sort Algorithm

For this algorithm we assume that the values of data items are bounded by the lowest and highest values, L and H and the data tiems are uniformly distributed.

**step 1 :**

Local partition, each node splits the interval (L, H) into p segment $\Delta_i$, such that $\Delta_0 = (L, \frac{H-L}{p})$, $\Delta_1 = (\frac{H-L}{p}, \frac{2(H-L)}{p})$, ..., $\Delta_i = (\frac{i}{p}, \frac{i+1}{p})$, ...
and the local data items to be sorted are partitioned over the segments.

**step 2 :**

The data items in nodes are transferred like Fig. 1.

**step 3 :**

Local sorting.

The total computing time is $n(\log p)t_{search} + \frac{1}{2}(n\log p)t_{comm} + n(\log n)t_{sort}$.

## IV. COMPUTING TIME AND EFFICIENCY

Now compare the algorithms on hypercube machine. The efficiency $\varepsilon$ of the concurrent algorithm is given by:

$$\varepsilon = \frac{1}{E} = p\frac{T_C}{T_S}, \quad T_S = N(\log N)t_{sort}$$

where $\frac{T_C}{T_S}$ is the total concurrent/sequential time. The inefficiency parameter $\tau$ is $\frac{t_{comm}}{t_{flop}}$. To get rough estimate of efficiency, we assume typical numerical values for $N = p^2$ and $\tau \approx 1.5$. For estimate's sake we take $t_{search} \approx t_{compare} \approx t_{sort} \approx 2t_{flop}$(one $t_{flop}$ for "compare" action and one $t_{flop}$ for decision, address evaluation, etc.).

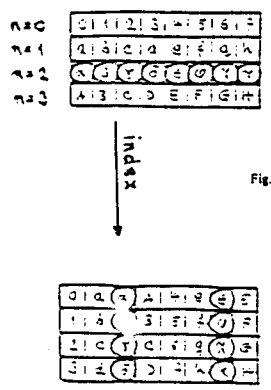

Fig. 1 Index Operation

Parallel Bitonic Sort:

$$T_C = \{2n(\log n) + 2nD(D+2)\}t_{flop}$$

$$E = \frac{p}{2N\log N} * \{2n(\log n) + \frac{1}{2}nD(D+5) + \tau nD(D+1)\}$$

$$= \frac{1}{4D} * \{2D + \frac{1}{2}D(D+5) + \tau D(D+1)\}$$

$$= \frac{1}{2} + \frac{1}{8}(D+5) + \frac{\tau}{4}(D+1)$$

$$= \frac{3}{2} + \frac{1}{2}D$$

Parallel Quicksort:

$$T_C = \{2n(\log n) + \frac{3D(D+1)}{4} + \frac{7}{2}nD\}t_{flop}$$

$$E = \frac{p}{2N\log N} * [2n(\log n) + 2nD + \tau*\{\frac{D(D+1)}{2} + nD\}]$$

$$= \frac{1}{2\log N} * [2(\log n) + 2(\log p) + \tau*\{\frac{D(D+1)}{2n} + D\}]$$

$$= 1 + \frac{\tau}{4D}*\{\frac{D(D+1)}{2n} + D\}$$

$$= 1 + \tau\{\frac{1}{4} + \frac{(D+1)}{2^{D+3}}\}$$

$$= \frac{11}{8} + \frac{3}{2^{D+4}}(D+1)$$

Shell Sort :

$$T_C = 2n(\log n) + 7nD + T(n, D)$$

$$E = \frac{1}{2\log N}\{2(\log n) + 4D(1 + \frac{\tau}{2})\}$$

$$= \frac{3}{2} + \frac{\tau}{2}$$

$$= \frac{9}{4}$$

Index Sort :

$$T_C = 2n(\log n) + \frac{11}{4}nD$$

$$E = 1 + \frac{\tau}{4} * \frac{\log p}{\log N}$$

$$= 1 + \frac{3}{8}*\frac{1}{2}$$

$$= \frac{19}{16}$$

## V. DISCUSSION AND FUTURE WORKS

Fig. 2 shows the efficiency of the sorting algorithms according to dimensions. Obviously the performance is overestimated in the algorithms since, even in the uniform distribution case, one should include the load imbalance effects due to statistical fluctuations. It is expected that the real efficiency will be lowered.

Now consider the algorithms with this kinds of effect. The bitonic sorting algorithm is hardly affected from load imbalance due to its inherent characteristics. The Fig.2 shows that it is fairly efficient until dimension gets extremly large. The shell sort is a modified version of parallel bitonic sorting algorithm. Thus different workload hardly occurs. The figure shows the efficiency of shell sort is constant since we neglect T(n, D) factor. With T(n, D), it is expected that the efficiency will be slowly dropped as the dimension D is increasing.

Quicksort shows that the efficiency is enhanced with the increase of dimension. Unfortunately, though it shows fairly good efficiency, the efficiency is much prone to be dropped if the workload is different. Index sort shows extraordinarily high efficiency. The efficiency is constant over all dimension. However, one of disadvantage is that the algorithm should know the highest and lowest value of the data items. Another disadvantage is the load imbalance. If many data items locate in small range while remaining small data items are in wide range, the efficiency may drop to the bottom.

There are some remaining works in sorting algorithm on hypercube machine. The exact order of T(n, D) in shell sort should be investigated. In case of quicksort, appropriate selection of splitting key is important in the efficiency. On the contrary to the high efficiency at average case, the computing time requires great time at worst case is $2N(\log N) + \frac{7}{2}N$, which is much worse than sequential algorithm at average case.

We are now making a prototype for the hypercube machine based on the 68020 32-bit processors. Our testbed will have eight node processors and an intermediate host computer based on Unix system. We will try to develop the parallel sorting algorithms on the real testbed. At that time the algorithms will be investigated for various kinds of data , size of data, and load imbalance with experimental results.

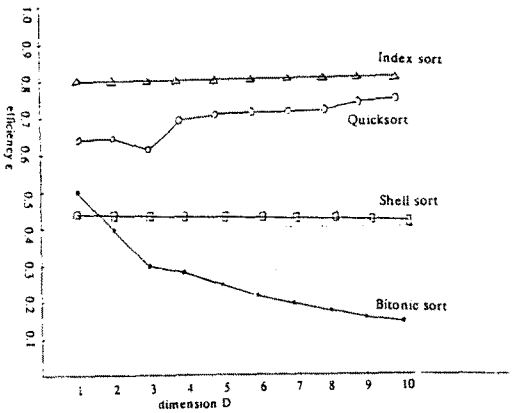


Fig. 2 Efficiency Comparison Table

## REFERENCES

[1] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, 1984, pp. 361-367.

[2] Y. Saad and M. Schultz, "Topological Properties of Hypercubes," Research Report, YALEU/DCS/RR-389, Dept. of Computer Science, Yale University, June 1985.

[3] G. C. Fox and W. Furmanski, "Optimal Communication Algorithms on Hypercube," $C^3$P-314, California Institute of Technology, July 1986.

[4] K. E. Batcher, "Sorting Networks and Their Applications," Spring Joint Computer Conference, AFIPS Proceedings, vol. 32, 1968, pp. 307-314.

[5] G. Baudet and D. Stevenson, "Optimal Sorting Algorithms for Parallel Computers," *IEEE Trans. on Computers*, vol. 27, no. 1, January 1978, pp. 84-87.

[6] *iPSC™ System Product Summary*, Intel Corp., 3065 Bowers Avenue, Santa Clara, CA 95051, April 1986.

[7] H. P. Katseff, "Incomplete Hypercubes," Final Program and Abstracts of the 2nd Conf. on Hypercube Multiprocessors, 1986, p. 50.

[8] E. Felton, S. Karlin, and S. Otto, "Concurrent Sorting," $C^3$P-297.9, California Institute of Technology, 1986.