Conflict–Free Sorting Algorithms under Single–Channel and Multi–Channel Broadcast Communication Models***

Chang–Biau Yang*, **R. C. T. Lee**** and **Wen–Tsuen Chen****

* Department of Applied Mathematics, National Sun Yat–sen University
Kaohsiung, Taiwan 80424, Republic of China.

** Department of Computer Science, National Tsing Hua University
Hsinchu, Taiwan 30043, Republic of China.

*** This research work was partially supported by the National Science Council of the Republic of China under contract NSC77–0408–E007–04.

Abstract

In this paper, we shall first propose an optimal conflict–free parallel sorting algorithm under the single–channel broadcast communication model. The time complexity of this algorithm is $O((n/p)\log(n/p)+n)$ if p processors are used to sort n data elements. We then apply our single–channel sorting algorithm to improve the multi–channel sorting algorithm proposed by Marberg and Gafni. Since there is a constraint for the Marberg and Gafni's algorithm, we shall propose another conflict–free multi–channel sorting algorithm without this constraint. The time complexity of this algorithm is $O((n/p)\log(n/p)+(n/k)\log^2 k)$ if there are n data elements to be sorted by using p processors with k channels.

Section 1 Introduction

For designing parallel algorithms, the broadcast communication model [4, 5, 8, 9, 12, 14, 15, 17, 18] has attracted attentions from researchers recently. In this model, there is one or more than one shared communication channel. It is called a **single–channel broadcast communication system** if only one shared channel is used and called a **multi–channel broadcast communication system** if more than one shared channel is used. In a multi–channel broadcast communication system, each channel can be dynamically switched among processors. Each processor in a broadcast communication multiprocessor system can communicate with others only through the shared channels. Whenever a processor broadcasts messages in one channel, any other processor can hear the messages via this channel. If more than one processor attempts to use the same channel to broadcast messages

simultaneously, a **broadcast conflict** occurs in this channel. When a conflict occurs in a channel, we have to use a conflict resolution scheme to resolve the conflict. An algorithm is **conflict–free** if there is no conflict during the execution of the algorithm. Of course, a conflict–free algorithm has better performance because no resolution time is needed.

Parallel sorting algorithms have been studied and designed by many researchers. For complete reviews of this subject, see [3, 16]. Some researchers [5, 8, 9, 12] have used the broadcast communication model to solve the sorting problem.

Both Levitan's sorting algorithm [8] and Dechter and Kleinrock's [5] sorting algorithm have the problem of broadcast conflicts. Willard [19] proved that the lower bound of the average time for any algorithm to resolve a broadcast conflict in a p–processor broadcast communication multiprocessor system is $\Omega(\log\log p)$. In other word, it is impossible that there exists a mechanism to resolve a broadcast conflict in a constant time. Thus, it is preferred to design an algorithm without any broadcast conflicts, which is a conflict–free algorithm.

Marberg and Gafni [9] first proposed a conflict–free sorting algorithm under the multi–channel broadcast communication model. The time required for their algorithm to sort n numbers with p processors and k channels, $n \geq p \geq k$, is $O((n/k)\log(n/k))$ with the constraint $n \geq k^2(k-1)$.

In this paper, based upon the concept of enumeration sorting [11], we first propose an optimal conflict–free sorting algorithm under the single–channel model. The time required for this algorithm is $O((n/p)\log(n/p)+n)$ if there are n data elements to be sorted and p processors are incorporated in the broadcast communication multiprocessor system. Our algorithm is better than those proposed by previous researchers [5, 8, 12]. Based upon our single–channel sorting algorithm, we also propose an improvement of Marberg and Gafni's multi–channel sorting algorithm.

We then propose a conflict–free multi–channel sorting algorithm, which is based upon the concepts of bitonic sorting [1, 13] and block sorting [2, 10]. In this algorithm, we propose a simple channel schedule to avoid broadcast conflicts. The time required for this algorithm is $O((n/p)\log(n/p)+(n/k)\log^2 k)$. For this algorithm, we use the same model used by Marberg and Gafni [9]. Our algorithm does not have the constraint $n \geq k^2(k-1)$, which is needed in Marberg and Gafni's algorithm.

Section 2 A Conflict–Free Single–Channel Sorting Algorithm

In this section, we shall propose an optimal conflict–free sorting algorithm under the single–channel broadcast communication model. First, we define the rank of an element in a set of numbers. Let S be a set of numbers and i belong to S. The **rank** of i, denoted as rank(i), is defined as the number of elements in S which are smaller than i. For example, consider $S=\{10, 16, 2, 20, 5\}$. The ranks of all elements in S are as follows: rank(10)=2, rank(16)=3, rank(2)=0, rank(20)=4 and rank(5)=1.

In the following, we shall apply the idea of enumeration sorting [11] to sort n data elements by using p processors, $p \leq n$, under the single–channel broadcast communication model. We assume

that p divides n. If it is not this case, we only add some dummy data elements with values being positive infinite to fulfill our assumption. These processors are numbered by 1, 2, ... and p. Each processor stores n/p data elements and reserves the storage for storing the ranks of these data. We assume that all data are mutually distinct; otherwise, each data element would be replaced by a 3–tuple consisting of itself, the identifier of the processor storing it and the index of the storage storing it. Our algorithm first computes the ranks of all data elements and then reports the sorted sequence by the increasing order of the ranks.

Algorithm Single–Channel–Sorting

Step 1: Each processor sorts its local data into an ascending list and computes the rank of each data elements with respect to the set of data elements stored in this processor.

Step 2: i=1. Repeat the following substeps p times.

Step 2.1: Processor i broadcasts all data elements stored in it in the ascending sequence. All other processors temporarily store the broadcast data in their local memory.

Step 2.2: All processors except processor i apply the **two–way ranking** scheme to update the ranks of the data elements stored in them.

Step 2.3: i <— i+1.

Step 3: Report the sorted sequence according to the ranks of these n data elements.

In the above algorithm, only the two–way ranking scheme used in step 2.2 has to be designed. Suppose that there are two sorted lists $a_1<a_2<...<a_m$ and $b_1<b_2<...<b_m$ and one ranking list r_1, r_2, ..., r_m associated with the first list such that r_i is the rank of a_i with respect to the first list. That is, $r_i=i-1$, $1\le i\le m$. We assume that the data of these two sorted lists are all distinct. We apply the concept of the two–way merging algorithm [6] to find the position of the second list where a_i is bounded. If $b_j<a_i<b_{j+1}$, $1\le j\le m-1$ and $1\le i\le m$, then r_i is replaced by r_i+j. For the boundary condition that $a_i>b_m$, we substitute r_i with r_i+m. After the two–way ranking, r_i will be the rank of a_i with respect to these two lists. For example, consider two sorted lists A=(2, 5, 10, 16, 20) and B=(6, 7, 12, 13, 14), and a ranking list R=(0, 1, 2, 3, 4) associated with list A. After performing the two–way ranking, the ranking list R will become R'=(0, 1, 4, 8, 9). The time required for this two–way ranking scheme is O(m).

The total time required for our algorithm is $O((n/p)\log(n/p)+n)$, which can be easily calculated from the time complexity for each step as follows:

Step 1: $O((n/p)\log(n/p))$.

Step 2: $O((n/p)+(n/p))\cdot p=O(n)$.

Step 3: O(n).

In the following, we shall prove that the above sorting algorithm is optimal under the single–channel broadcast communication model. To prove the following lemmas, we need a definition. For two data elements x and y in a set, y is the **predecessor** of x if rank(y)=rank(x)–1.

[**Lemma 2.1**] Let a_1, a_2, ..., a_n be a sorted sequence where $a_1 \le a_2 \le ... \le a_n$. Suppose that these n data elements are stored in p processors arbitrarily. The number of data elements stored in processor i is n_i, $1 \le i \le p$. Let n_j be the maximum among all n_i's, $1 \le i \le p$. If $n_j < n/2$, there exists one case such that for each data element a_k, $2 \le k \le n$, its predecessor, a_{k-1}, is not stored in the same processor with a_k.

PROOF: To prove this lemma, we need only to generate one such case. We rearrange the sorted sequence into a new sequence as $a_1a_3a_5...a_{n-1}a_2a_4...a_n$ (Without losing generality, we assume that n is even.). We store the n data elements into the p processors according to the rearranged sequence. At step i, $1 \le i \le p$, we store the first n_i data elements of the rearranged sequence into processor i and then delete these n_i data elements from the sequence. After doing this, we claim that a_k and a_{k-1}, $2 \le k \le n$, would not be stored in the same processor. Two cases should be considered as follows.

Case 1: k is odd. If both a_k and a_{k-1} are stored in processor i, $1 \le i \le p$, then the subsequence $a_ka_{k+2}...a_{n-1}a_2a_4...a_{k-1}$ would be stored in processor i. The length of this subsequence is n/2. This is impossible since $n_i < n/2$. Thus, processor i can not store both a_k and a_{k-1}.

Case 2: k is even. For this case, it is similar to case 1.

By both these case, our claim is correct. Therefore, the proof is complete.

Q. E. D.

Dechter and Kleinrock [5] have proved that the lower bound of number of broadcasts is $\Omega(n)$ when p processors are used to sort n data elements under the single–channel broadcast communication model and these n data elements are divided and stored in the p processors evenly. We shall give a more general lower bound for this problem and prove it as follows.

[**Lemma 2.2**] In the single–channel broadcast communication model, the time required for an algorithm using p processors, $p \ge 1$, to sort n data elements is $\Omega(n)$ under the assumption that the data are preloaded into the processors and each data element is loaded into only one processor.

PROOF: If the sorted data need to be reported, the time required for reporting is $\Omega(n)$ since only one channel is available to report the sorted data. In the following, we assume that data do not need to be reported.

To determine the final position of one data element in the entire sorted sequence, we must compare it at least with its predecessor (except the smallest data element). Suppose that n_i data elements are preloaded into processor i. Here, $\sum_{i=1}^{p} n_i = n$. Let n_j be the maximum among all n_i's, $1 \le i \le p$. We have to consider two cases as follows.

Case 1: $n_j \ge n/2$. For each data element in processor j, either processor j compares this data element with its predecessor or broadcasts this data element to other processors to perform the comparison. Thus, processor j must read each data element stored in it at least once. The time required for processor j to read the data elements stored in it is at least $\Omega(n/2) = \Omega(n)$. Therefore,

the time required for sorting is $\Omega(n)$.

Case 2: $n_j<n/2$. Let a_1, a_2, ..., a_n be the sorted sequence of these n data element where $a_1\leq a_2\leq...\leq a_n$. Without losing generality, it is assumed that n is even. Consider the following n/2 data elements: a_2, a_4, a_6, ..., a_n. For each data element a_k where $2\leq k\leq n$ and k is even, by Lemma 2.1, there exists one such case that a_{k-1} (the predecessor of a_k) would not be stored in the same processor with a_k. In order to compare each pair of a_k and a_{k-1}, either a_k or a_{k-1} must be broadcast. At least n/2 successful broadcasts are required if we want to compare a_2, a_4, ..., a_n with their predecessors respectively. Thus, the time required for sorting at least $\Omega(n)$ since only one channel is available.

By both these two cases, this lemma is true. Q. E. D.

Under any parallel computation model, the time required for an algorithm using p processors, $p\geq 1$, to sort n data elements is at least $\Omega((n\log n)/p)$ since the lower bound for a sequential sorting algorithm to sort n data elements is $\Omega(n\log n)$ [6]. Combining with Lemma 2.2, we obtain that under the single–channel broadcast communication model, the lower bound for an algorithm using p processors, $p\geq 1$, to sort n data elements is $\Omega(n+(n\log n)/p)$.

[Theorem 2.3] Under the single–channel broadcast communication model, Algorithm Single–Channel–Sorting is optimal under the assumption that the data are preloaded into the processors and each data element is loaded into only one processor.

PROOF: The time complexity of Algorithm Single–Channel–Sorting is $O((n/p)\log(n/p)+n)$. We have to consider two cases as follows.

Case 1: $n\geq(n\log n)/p$. In this case, $n\geq(n/p)\log(n/p)$. Then $n+(n\log n)/p=\Omega(n)$ and $n+(n/p)\log(n/p)=O(n)$.

Case 2: $n\leq(n\log n)/p$. In this case, $p\leq\log n$. Then $n+(n\log n)/p=\Omega((n\log n)/p)$ and $n+(n/p)\log(n/p)=O((n\log n)/p)$.

For both these cases, the time complexity of Algorithm Single–Channel–Sorting achieves the lower bound of any sorting algorithm under this model. The proof is complete.

Q. E. D.

Section 3 An Improvement of Marberg and Gafni's Sorting Algorithm

In this section, we shall propose an improvement of Marberg and Gafni's sorting algorithm [9]. Marberg and Gafni applied the column sort [7] to the multi–channel broadcast communication model and designed the first conflict–free sorting algorithm in this model. Their sorting algorithm uses p processors with k channels to sort n data elements where $n\geq p\geq k$. In their algorithm, they needed to sort the data stored in a group of processors which share one broadcast channel. In a group of processors, they only chose one of these processors to perform the sorting. In other words, all the data in one group of processors are sent to one of these processors to be sorted. The time required for their algorithm to sort n data elements by using p processors with k channels, $n\geq p\geq k$, is $O((n/k)\log(n/k))$ with the constraint $n\geq k^2(k-1)$. The constraint $n\geq k^2(k-1)$ is induced because of the

use of Leighton's column sort.

There are two problems for Marberg and Gafni's sorting algorithm as follows. (1). Using more processors can not reduce the required time. The time required for the algorithm depends only on n and k, not on p. (2). There is a serious constraint: $n \geq k^2(k-1)$.

For the first problem, we can easily improve Marberg and Gafni's algorithm by applying Algorithm Single–Channel–Sorting. In our improvement, we use Algorithm Single–Channel–Sorting to perform the sorting of a groups of processors which share one channel. Since no processor is idle when Algorithm Single–Channel–Sorting is executed, the time complexity can be reduced. The time required for our improvement algorithm is $O((n/p)\log(n/p)+n/k)$ since there are n/p data elements in each processor and n/k data elements in each group.

For the second problem, we shall propose a new algorithm which does not have the serious constraint $n \geq k^2(k-1)$ in the next section.

Section 4 A Conflict–Free Multi–Channel Sorting Algorithm

In Marberg and Gafni's sorting algorithm [9], there is a constraint $n \geq k^2(k-1)$. In the following, we shall present a conflict–free multi–channel sorting algorithm which will work without this constraint.

In our multi–channel sorting algorithm, we assume that p processors with k channels are used in the multi–channel broadcast system and there are n data elements to be sorted, where $n \geq p \geq k$. These p processors are divided into 2k groups and each group consists of p/2k processors. We first apply our single–channel sorting algorithm to sort the data in every two groups by using one channel in parallel. Then, we use the famous bitonic sorting [1, 13] as our basic parallel sorting algorithm. When we perform the sorting, we view each group as a data element and replace each comparison–exchange operation by one merging–splitting of two groups. This is the spirit of blocking sorting [2, 10]. We give a simple channel schedule such that the merging–splitting of every two groups can be performed in parallel without any conflicts. Our multi–channel sorting algorithm is as follows.

Algorithm Multi–Channel–Sorting

Assumption: p processors with k communication channels are used to sort n data elements. p divides n, 2k divides p and $k=2^{m-1}$ where m is a positive integer. The p processors are divided into 2k groups with equal sizes. These 2k groups are numbered by 0,1,2,...,2k–1. Each processor is numbered by a pair of indices (i,j) where $0 \leq i \leq 2k-1$ and $0 \leq j \leq \frac{p}{2k} - 1$. The first index is the group index and the second index is the processor index within that group. These k channels are also numbered by 0,1,2,...,k–1.

Initialization: Each processor stores n/p data elements.

Final: Each processor stores a sorted sequence with n/p data elements. All data in processor (i,j) are smaller than those in processor (u,v) if $i<u$, or $i=u$ and $j<v$.

Step 1: For each group i, $0 \le i \le k-1$, the processors of group i share channel i to sort the data stored in the processors of group i by using Algorithm Single–Channel–Sorting. After sorting, each processor stores a sorted sequence with n/p data elements and all data stored in processor (i,j) are smaller than those stored in processor (i,j+1) for $0 \le i \le k-1$, $0 \le j \le \frac{p}{2k} - 2$.

Step 2: The processors of group i+k, $0 \le i \le k-1$, share channel i to sort the data stored in the processors of group i+k as do in step 1.

Step 3: Viewing each group as a data element, use bitonic sorting as the basic parallel sorting algorithm. Each comparison–exchange of two data elements is replaced by a merging–splitting of two groups which is performed by using one channel.

In the above algorithm, only step 3 has to be designed more detailedly. Our first problem is: How do we perform the merging–splitting of two groups under the broadcast communication model? Note that this must be done by using one broadcast channel and in a conflict–free fashion. We can perform the two–way merging algorithm [6] to merge the data in two groups by using one channel in a conflict–free fashion. Then, we can split the sorted data into the processors of these two groups. The time for this merging–splitting procedure is O(m) if there are totally m data elements to be merged and split.

Our multi–channel sorting algorithm is based upon bitonic sorting [1, 13]. Thus, we can use the same decision to overcome the second problem: Which two groups should be merged and split? Our third problem is how to select one channel for performing the merging–splitting of two groups. We shall also use the shuffle scheme proposed by Stone [13].

Assume that there are n data elements to be sorted in bitonic sorting. Stone used a shuffle scheme to choose two data elements to be compared for a comparison–exchange step. A **stage** consists of some parallel comparison–exchange steps. The jth stage consists of j parallel comparison–exchange steps. After the jth stage is completed, $n/(2^j)$ sorted sequences, each of length 2^j, are found. Naturally, the n data elements will be sorted after the (logn)th stage is completed. Thus, the total parallel comparison–exchange steps required for bitonic sorting is $O(\log^2 n)$. Assume that $n=2^m$, where m is a positive number. An index i with the binary representation $i_{m-1}i_{m-2}\cdots i_0$ will become $i_{m-2}\cdots i_0 i_{m-1}$ after one left rotation, or say one **shuffle**. At the beginning of the jth stage, all indices are shuffled m–j times. For the subsequent j steps, each consists of one comparison–exchange followed by one shuffle. When a comparison–exchange is performed, the two data elements with their current indices (after several shuffles) differing only in the leftmost bit are compared. For example, consider that n=8. At the beginning of the second stage, 4=100 and 6=110. They become 001 and 101 respectively after one shuffle. At the first comparison–exchange step of the second stage, the data in positions 4 and 6 are compared since their current indices 001 and 101 differ only in the leftmost bit.

Here, we use the group indices instead of position indices used by Stone. When two groups are merged, the current group indices (after several shuffles) of these two groups must differ only in the leftmost bit. We remove the leftmost bit from these two indices to obtain a channel index, which is the index of the channel that we use to merge these two groups. Fig. 1 shows an example with

group number	channel usage									
	stage 1	stage 2		stage 3			stage 4			
0	0	0	0	0	0	0	0	0	0	0
1	0	4	0	2	4	0	1	2	4	0
2	1	0	1	4	0	1	2	4	0	1
3	1	4	1	6	4	1	3	6	4	1
4	2	1	2	0	1	2	4	0	1	2
5	2	5	2	2	5	2	5	2	5	2
6	3	1	3	4	1	3	6	4	1	3
7	3	5	3	6	5	3	7	6	5	3
8	4	2	4	1	2	4	0	1	2	4
9	4	6	4	3	6	4	1	3	6	4
10	5	2	5	5	2	5	2	5	2	5
11	5	6	5	7	6	5	3	7	6	5
12	6	3	6	1	3	6	4	1	3	6
13	6	7	6	3	7	6	5	3	7	6
14	7	3	7	5	3	7	6	5	3	7
15	7	7	7	7	7	7	7	7	7	7

Fig. 1 The channel schedule for Algorithm Multi–Channel–Sorting with 16 groups. For each step, the two groups which use the same channel will be merged and split.

sixteen groups and eight channels for the channel schedule. Our algorithm under this simple channel schedule is conflict–free since only two groups share one channel and these two groups are merged and split without any conflicts.

In the following, we shall calculate the complexity for Algorithm Multi–Channel–Sorting. The time required for step 1 or step 2 is $O(n/p)\log(n/p)+n/(2k)=O((n/p)\log(n/p)+n/k)$ since there are $n/2k$ data elements in each group to be sorted and there are n/p data elements in each processor to be sorted. The time required for step 3 is $O((n/2k)\log^2(2k))=O((n/k)\log^2 k)$ since each merging–splitting step requires $O(n/2k)$ time and the bitonic sorting requires $O(\log^2(2k))$ parallel

merging–splitting steps. Thus, the total time required for Algorithm Multi–Channel–Sorting is $O((n/p)\log(n/p)+(n/k)\log^2 k)$.

In Algorithm Multi–Channel–Sorting, we assume that $n \geq p \geq k$, p divides n, 2k divides p and $k=2^{m-1}$ where m is a positive integer. We shall modify the algorithm to work correctly even if all constraints except $n \geq p \geq k$ are removed.

(1). If $2^{m-1}<k<2^m$ where m is a positive integer, we shall use k' to replace k where $k'=2^{m-1}$. If $k=2^{m-1}$ where m is a positive integer, let k'=k.

(2). If p can not divide n, we add (n'–n) dummy data elements with the infinite positive value such that n' is the smallest number satisfying that p can divide n'.

(3). If $p \geq 2k'$ and 2k' can not divides p, we still partition these p processors into 2k' groups evenly such that the difference of numbers of processors in different groups is at most one.

(4). If $k' \leq p < 2k'$, let $k''=k'/2$ and k' be replaced by k''.

With the above modification, the time complexity for our algorithm remains unchanged. Our algorithm uses the same broadcast communication model as used by Marberg and Gafni [9]. Their sorting algorithm requires $O((n/k)\log(n/k))$ time with a serious constraint $n \geq k^2(k-1)$. Our algorithm does not have the constraint $n \geq k^2(k-1)$. We only assume that $n \geq p \geq k$.

Section 5 Concluding Remarks

In this paper, we proposed some conflict–free sorting algorithms under the broadcast communication model. We first proposed a conflict–free single–channel sorting algorithm based upon enumeration sorting [11]. The time required for this algorithm is $O((n/p)\log(n/p)+n)$ if p processors are used to sort n data elements. We also proved that this algorithm is optimal under the single–channel broadcast communication model. Then, we applied this algorithm to improve the multi–channel sorting algorithm proposed by Marberg and Gafni [9]. The time complexity of their algorithm is $O((n/k)\log(n/k))$ if the algorithm sorts n data elements by using p processors with k channels, where $n \geq p \geq k$ and $n \geq k^2(k-1)$. The improved algorithm proposed by us requires $O((n/p)\log(n/p)+n/k)$ time. We also designed a new conflict–free multi–channel sorting algorithm without the constraint $n \geq k^2(k-1)$ which is required in Marberg and Gafni's algorithm. This algorithm mixes the concepts of our single–channel sorting algorithm, Batcher's bitonic sorting algorithm [1, 13] and merging and splitting for block sorting algorithm [2, 10]. The time complexity of this algorithm is $O((n/p)\log(n/p)+(n/k)\log^2 k)$ if there are n data elements to be sorted with p processors and k channels, $n \geq p \geq k$.

For sorting algorithms under the multi–channel broadcast communication model, we have not found a lower bound of the time complexity yet. Intuitively, we believe that our new algorithm under the multi–channel broadcast communication model is not optimal. We are presently working on a better sorting algorithm under the multi–channel broadcast communication model.

REFERENCES

[1] K. E. Batcher, "Sorting Networks and Their Applications," Proc. AFIPS 1968 Spring Joint Computer Conf., Atlantic City, New Jersey. Vol. 32, 1968, pp. 307–314.

[2] G. Baudet and D. Stevenson, "Optimal Sorting Algorithms for Parallel Computers," IEEE Trans. Comput., Vol. C–27, No. 1, Jan. 1978, pp. 84–87.

[3] D. Bitton, D. J. DeWitt, D. K. Hsiao and J. Menon, "A Taxonomy of Parallel Sorting," Computing Surveys, Vol. 16, No. 3, Sept., pp. 287–318.

[4] S. H. Bokhari, "Max: An Algorithm for Finding Maximum in an Array Processor with a Global Bus," Proc. 1981 Int. Conf. Parallel Processing, 1981, pp. 302–303.

[5] R. Dechter and L. Kleinrock, "Broadcast Communications and Distributed Algorithms," IEEE Trans. Comput., Vol. C–35, No. 3, Mar. 1986, pp. 210–219.

[6] D. E. Knuth, The Art of Computer Programming: Sorting and Searching, Vol. 3, Addison, Wesley Publishing Company Inc., Reading, Massachusetts, 1973.

[7] T. Leighton "Tight Bounds on the Complexity of Parallel Sorting," IEEE Trans. Comput., Vol. C–34, No. 4, April 1985, pp. 344–354.

[8] S. Levitan, "Algorithms for Broadcast Protocol Multiprocessor," Proc. 3rd Int. Conf. Distributed Computing Systems, 1982, pp. 666–671.

[9] J. M. Marberg and E. Gafni, "Sorting and Selection in Multi–Channel Broadcast Networks," Proc. 1985 Int. Conf. Parallel Processing, 1985, pp. 846–850.

[10] J. Menon, "A Study of Sort Algorithms for Multiprocessor Database Machines," Proc. 12th Int. Conf. Very Large Data Bases, Kyoto, Japan, 1986, pp. 197–206.

[11] F. P. Preparata, "New Parallel Sorting Schemes", IEEE Trans. Comput., Vol. C–27, No. 7, July 1978, pp. 669–673.

[12] K. V. S. Ramarao, "Distributed Sorting on Local Area Networks," IEEE Trans. Comput., Vol C–37, No. 2, Feb. 1988, pp. 239–243.

[13] H. S. Stone, "Parallel Processing with the Perfect Shuffle," IEEE Trans. Comput., Vol. C–20, No. 2, Feb. 1971, pp. 153–161.

[14] C. Y. Tang and S. C. Wu, "Parallel Graph Algorithms under Broadcast Communication Model," Proc. Int. Computer Symp. 1988, Taipei, Taiwan, R. O. C., pp. 759–763.

[15] C. Y. Tang and M. J. Chiu, "Distributed Sorting on the Serially Connected Local Area Networks," Proc. 1989 Singapore Int. Conf. Networks, Singapore, pp. 458–462.

[16] S. S. Tseng, Parallel Sorting Algorithms, Ph. D. Dissertation, Institute of Computer Engineering, National Chiao Tung University, Hsinchu, Taiwan, Republic of China.

[17] C. B. Yang, R. C. T. Lee and W. T. Chen, "Finding Minimum Spanning Trees Based upon Single–Channel Broadcast Communications," Proc. Int. Computer Symp. 1988, Taipei, Taiwan, R. O. C., pp. 1451–1456.

[18] C. B. Yang, R. C. T. Lee and W. T. Chen, "Parallel Graph Algorithms Based upon Broadcast Communications," IEEE Trans. Comput., Vol C–39, No. 12, Dec. 1990, pp. 1468–1472.

[19] D. E. Willard, "Log–Logarithmic Protocols for Resolving Ethernet and Semaphore Conflicts," Proc. 16th Annual ACM Symp. on Theory of Computing, 1984, pp.512–521