A Fast Sorting Algorithm and Its Generalization on Broadcast Communications[⋆]

Shyue-Horng Shiau and Chang-Biau Yang [⋆⋆]

Department of Computer Science and Engineering,
National Sun Yat-Sen University,
Kaohsiung, Taiwan 804, R.O.C.,
{shiaush, cbyang}@cse.nsysu.edu.tw,
http://par.cse.nsysu.edu.tw/~cbyang

Abstract. In this paper, we shall propose a fast algorithm to solve the sorting problem under the *broadcast communication model*(BCM). The key point of our sorting algorithm is to use successful broadcasts to build broadcasting layers logically and then to distribute the data elements into those logic layers properly. Thus, the number of *broadcast conflicts* is reduced. Suppose that there are n input data elements and n processors under BCM are available. We show that the average time complexity of our sorting algorithm is $\Theta(n)$. In addition, we expand this result to the generalized sorting, that is, finding the first k largest elements with a sorted sequence among n elements. The analysis of the generalization builds a connection between the two special cases which are maximum finding and sorting. We prove that the average time complexity for finding the first k largest numbers is $\Theta\left(k + \log\left(n-k\right)\right)$.

1 Introduction

One of the simplest parallel computation models is the *broadcast communication model* (BCM)[1, 2, 5, 6, 14, 13, 4, 9, 7, 11, 12, 15]. This model consists of some processors sharing one common channel for communications. Each processor in this model can communicate with others only through this shared channel. Whenever a processor broadcast messages, any other processor can hear the broadcast message via the shared channel. If more than one processor wants to broadcast messages simultaneously, a *broadcast conflict* occurs. When a conflict occurs, a conflict resolution scheme should be invoked to resolve the conflict. This resolution scheme will enable one of the broadcasting processors to broadcast successfully. The Ethernet, one of the famous local area networks, is an implementation of such a model.

The time required for an algorithm to solve a problem under BCM includes three parts: (1) resolution time: spent to resolve conflicts, (2) transmission time:

[⋆] This research work was partially supported by the National Science Council of the Republic of China under NSC89-2213-E110-005.

[⋆⋆] To whom all correspondence should be sent.

spent to transmit data, (3) computation time: spent to solve the problem. For the sorting problem, it does not seem that transmission time and computation time can be reduced. Therefore, to minimize resolution time is the key point to improve the time complexity.

The probability concept, proposed by Martel [7], is to estimate a proper broadcasting probability, which can reduce conflict resolution time. Another idea to reduce conflict resolution time is the layer concept proposed by Yang [13]. To apply the layer concept for finding the maximum among a set of n numbers [11], we can improve the time complexity from $\Theta(\log^2 n)$ to $\Theta(\log n)$ [12]. As some research has also been done on the BCM when processors have a known order, our paper is the case that processors have a unkown order.

Some researchers [8, 2, 4, 5, 15, 10] have solved the sorting algorithm under BCM. The selection sort can be simulated as a straightforward method for sorting under BCM. The method is to have all processors broadcast their elements and to find the maximum element repeatedly. In other words, the maximum elements which are found sequentially form the sorted sequence. Levitan and Foster [5, 6] proposed a maximum finding algorithm, which is nondeterministic and requires $O(\log n)$ successful broadcasts in average. Martel [7], and Shiau and Yang [11] also proposed maximum finding algorithms, which require $O(\log n)$ and $\Theta(\log n)$ time slots (including resolution time slots) in average respectively. Thus, the time required for the straightforward sorting method based on the repeated maximum finding is $O(n \log n)$.

Martel *et al.* [8] showed the sorting problem can be solved in $O(n)$. They also proved that the expected time for finding the first k largest numbers is $O(k + \log n)$. They used a modified version of selection sort, i.e., sort by repeatedly finding maximum based on the results from previous iterations. It can be viewed as a modified Quicksort. Until now, Martel's algorithm is the fastest one for solving the sorting problem under BCM. However, the constants associated with $O(n)$ and $O(k + \log n)$ bounds proved by Martel *et al.* [8] are not very tight.

This paper can be divided into two main parts. In the first part, we shall propose a sorting algorithm under BCM. In the algorithm, we make use of the maximum finding method [11] to build layers and reduce conflicts. After the process of maximum finding terminates, we can use these successful broadcasts to build broadcasting layers logically and then to distribute the data elements into those logic layers properly. Thus, conflict resolution time can be reduced. And we shall give the time complexity analysis of our algorithm. The average number of time slots (including conflict slots, empty slots, and slots for successful broadcast) required for sorting n elements is T_n, where $\frac{7}{2}n - \frac{1}{2} \leq T_n \leq \frac{23}{6}n - \frac{7}{6}$. Here, it is assumed that each time slot requires constant time. As we can see that the bound of our time complexity is very tight.

In the second part, we shall generalize the result of the first part. We prove that the average number of time slots required for finding the first k largest numbers with a sorted sequence is T_k^n, where $\frac{7}{2}k + 4\ln(n - (k-2)) - \left(\frac{1}{2} + 4\ln 2\right) \leq T_k^n \leq \frac{23}{6}k + 5\ln(n - (k-1)) - \frac{7}{6}$. The time complexity is also very tight. By the analysis of the generalization, we figure out the connection between maximum finding and sorting, which are two special cases of our generalization.

On the high level view, our sorting algorithm and generalization algorithm have a similar approach as that proposed by Martel *et al.* [8]. On the low level view, our algorithms are based on the layer concept, thus they can be analyzed easily and a tighter bound can be obtained. And Lemma 1 of this paper can be regarded as a generalization for the work which has the same high level approach. If there is another maximum finding algorithm under BCM whose bound can be analyzed , not only it can be applied into our sorting and generalized sorting approaches, but also the approaches based on it can be analyzed immediately by our lemmas in this paper.

This paper is organized as follows. In Section 2, we review the previous results for our maximum finding[11]. Based on the maximum finding, we present the sorting algorithm and its analysis in Section 3 and 4. In Section 5, we present the generalization of the sorting. In Section 6, we provide the analysis of the generalization. Finally, Section 7 concludes the paper.

2 The previous results for maximum finding

We shall first briefly review the maximum finding algorithm under BCM, which is proposed and analyzed by Shiau and Yang [11]. In the maximum finding algorithm, each processor holds one data element initially. The key point of the maximum finding algorithm is to use broadcast conflicts to build broadcasting layers and then to distribute the data elements into those layers. For shortening the description in our algorithm, we use the term "data element" to represent "the processor storing the data element" and to represent the data element itself at different times if there is no ambiguity. When a broadcast conflict occurs, a new upper layer is built up. Each data element which joins the conflict flips a coin with equal probabilities. That is, the probability of getting a head is $\frac{1}{2}$. All which get heads continue to broadcast in the next time slot, and bring themselves up to the new upper layer. This new layer becomes the active layer. The others which get tails abandon broadcasting, and still stay on the current layer. At any time, only the processors which are alive and on the active layer may broadcast.

In this paper, we shall use the notation $\langle x_0, x_1, x_2, \cdots, x_t \rangle$ to denote a linked list, where x_t is the head of the list. Suppose $L_1 = \langle x_0, x_1, x_2, \cdots, x_t \rangle$ and $L_2 = \langle y_0, y_1, y_2, \cdots, y_{t'} \rangle$, then $\langle L_1, L_2 \rangle$ represents the concatenation of L_1 and L_2, which is $\langle x_0, x_1, x_2, \cdots, x_t, y_0, y_1, y_2, \cdots, y_{t'} \rangle$.

The maximum finding algorithm can be represented as a recursive function: $Maxfind(C, L_m)$, where C is a set of data elements (processors). Initially, each processor holds one alive data element and sets an initial linked list $L_m = \langle x_0 \rangle$, where $x_0 = -\infty$.

Algorithm Maximum-Finding: $Maxfind(C, L_m)$

Step 1: Each alive data element in C broadcasts its value. Note that a data element is alive if its value is greater than the head of L_m. Each element smaller than or equal to the head drops out (becomes dead).

Step 2: There are three possible cases:

Case 2a If a broadcast conflict occurs, then do the following.

Step 2a.1: Each alive data element in C flips a coin. All which get heads form C' and bring themselves to the upper layer, and the others form C'' and stay on the current layer.

Step 2a.2: Perform $L_m = Maxfind(C', L_m)$.

Step 2a.3: Perform $L_m = Maxfind(C'', L_m)$.

Step 2a.4: Return L_m.

Case 2b: If exactly one data element successfully broadcasts its value y, then return $\langle L_m, y\rangle$.

Case 2c: If no data element broadcasts, then return L_m. (C is empty or all data elements of C are dead.)

In the above algorithm, we can get a sequence of successful broadcasts, which is represented by the linked list $L_m = \langle x_0, x_1, x_2, \cdots, x_{t-1}, x_t\rangle$, where $x_{i-1} < x_i$, $1 \le i \le t$ and $x_0 = -\infty$. Note that L_m is held by each processor and the head x_t is the maximum. A new upper layer is built up when the recursive function $Maxfind$ is called and the current layer is revisited after we return from $Maxfind$. In Case 2c, there are two possible situations. One is that no element goes up to the upper layer. The other is that no alive element remains on the current layer after the current layer is revisited because all elements go up to the upper layer, or some elements broadcast successfully on upper layers and they kill all elements on the current layer.

Shiau and Yang [11] proved that the total number of time slots, including conflict slots, empty slots and slots for successful broadcasts, is M_n in average, where $4\ln n - 4 < M_n < 5\ln n + \frac{5}{2}$. Grabner *et al.* [3] gave a precise asymptotic formula that $M_n \sim (\frac{\pi^2}{3\ln 2})\ln n \sim (4.74627644\ldots)\ln n$.

Here we use the lemma

$$\tfrac{4}{n} \le M_n - M_{n-1} \le \tfrac{5}{n},$$

which was proved by Yang *et al.* [11], rather than the precise asymptotic formula $M_n \sim (\frac{\pi^2}{3\ln 2})\ln n \sim (4.74627644\ldots)\ln n$. This is done because it is hard to apply the precise asymptotic formula and its fluctuation into the proof directly.

3 The sorting algorithm

In our sorting algorithm, each processor holds one data element initially. The key point of our algorithm is to use successful broadcasts which occur within the progress of maximum finding to build broadcasting layers and then to distribute the data elements into those layers properly.

When a series of successful broadcasts occur within the progress of maximum finding, some correspondent layers are built up. All processors which participated in the maximum finding are divided into those layers. Each layer performs the sorting algorithm until the layer is empty or contains exactly one data element.

Our algorithm can be represented as a recursive function: $Sorting(M)$, where M is a set of data elements (processors). In the algorithm, each processor holds one data element and maintains a linked list L_s. Our sorting algorithm is as follows:

```
Algorithm Sorting: $Sorting(M)$
Step 1: Each data element in $M$ broadcasts its value.
Step 2: There are three possible cases:
    Case 2a: If exactly one data element successfully broadcasts its value $y$,
        then return $\langle y \rangle$.
    Case 2b: If no data element broadcasts, then return $NULL$.
    Case 2c: If a broadcast conflict occurs, then perform
        $L_m = Maxfind(M, \langle x_0 \rangle)$, where $L_m = \langle x_0, x_1, x_2, \cdots, x_{t-1}, x_t \rangle$ is a
        series of successful broadcasts and $x_{i-1} < x_i$, $1 \leq i \leq t$ and $x_0 = -\infty$.
        Step 2c.1: Set $L_s = NULL$.
        Step 2c.2: For $i = t$ down to 1
                       Set $L_s = \langle x_i, L_s \rangle$.
                       do $L_s = \langle Sorting(M_i), L_s \rangle$,
                              where $M_i = \{x | x_{i-1} < x < x_i\}$.
        Step 2c.3: Return $L_s$.
```

In the above algorithm, x_1 is the element of the first successful broadcast, and x_t is the element of the last successful broadcast and also the largest element in M.

For example, suppose that M has 2 data elements $\{1, 2\}$. We can get a series of successful broadcasts L_m after $Maxfind(M, \langle x_0 \rangle)$ is performed. The first successful broadcast may be one of the two elements randomly, each case having probability $\frac{1}{2}$. If the first successful broadcast is 2, then $L_m = \langle x_0, 2 \rangle$, and the final sorted linked list is $\langle x_0, Sorting(M_1), 2 \rangle$, where $M_1 = \{x | x_0 < x < x_1\} = \{x | -\infty < x < 2\} = \{1\}$. If the first successful broadcast is 1, then $L_m = \langle x_0, 1, 2 \rangle$, and the final sorted linked list is $\langle x_0, Sorting(M_1), 1, Sorting(M_2), 2 \rangle$, where $M_1 = \{x | x_0 < x < x_1\} = \{x | -\infty < x < 1\} = NULL$, $M_2 = \{x | x_1 < x < x_2\} = \{x | 1 < x < 2\} = NULL$.

Martel *et al.* [8] showed the sorting problem can be solved in $O(n)$. They also proved that the expected time for finding first k largest numbers is $O(k + \log n)$. However, the constants associated with $O(n)$ and $O(k + \log n)$ bounds proved by Martel *et al.* [8] are not very tight.

On the high level view, our sorting algorithm has a similar approach as that proposed by Martel *et al.* [8]. On the low level view, our algorithm is based on the layer concept, thus it can be analyzed easily and a tighter bound can be obtained. And the analysis can be applied to other sorting algorithms which have the same high level approach under BCM.

4 Analysis of the sorting algorithm

In this section, we shall prove that the average time complexity of our sorting algorithm is $\Theta(n)$, where n is the number of input data elements. Suppose that there are n data elements held by at least n processors in which each processor holds at most one data element. Let T_n denote the average number of time slots, including conflict slots, empty slots and slots for successful broadcasts, required when the algorithm is executed. When there is zero or one input data element for the algorithm, one empty slot or one slot for successful broadcast is needed. Thus $T_0 = 1$ and $T_1 = 1$.

If $n = 2$, we have the following recursive formula:

$$T_2 = M_2 + R_2, \text{ where } R_2 = \left\{ \begin{array}{rll} \frac{1}{2}[& & T_1] \\ + \frac{1}{2}[T_0 & & +T_0] \end{array} \right\}. \tag{1}$$

We shall explain the above two equations. In the first equation, the first term M_2 is the average number of time slots required for the maximum finding algorithm while $n = 2$. The second term R_2 is the average number of time slots required for finishing the sorting after the maximum finding ends.

In the equation for R_2, the first successful broadcast may be either one of the 2 elements randomly, each case having probability $\frac{1}{2}$. The subterm of first term, T_1, arises when the first successful broadcast is the largest element. Thus, the layer between $x_0 = -\infty$ and the largest element contains one element, which is the second largest element, and needs T_1 time slots.

In second term, the subterm, $[T_0 + T_0]$, arises when the first successful broadcast is the second largest element and the second successful broadcast is the largest element. Thus, the layers between $x_0 = -\infty$, the second and the largest element are both empty and needs T_0 time slots in both layers.

Shiau and Yang [11] have proved that $M_2 = 5$, then we have

$$T_2 = 5 + \{\frac{1}{2}[1] + \frac{1}{2}[1 + 1]\} = \frac{13}{2}.$$

If $n = 3$, we also have the following recursive formula:

$$T_3 = M_3 + R_3, \text{ where } R_3 = \left\{ \begin{array}{rll} \frac{1}{3}[& & T_2] \\ + \frac{1}{3}[T_0 & & +T_1] \\ + \frac{1}{3}[R_2 & & +T_0] \end{array} \right\}. \tag{2}$$

In the equation for R_3, the first successful broadcast may be any one of the 3 elements randomly, each case having probability $\frac{1}{3}$. The subterm of first term, T_2, arises when the first successful broadcast is the largest element. Thus, the layer between $x_0 = -\infty$ and the largest element contains two elements, which is the second and third largest element, and needs T_2 time slots for sorting.

In the second term, the subterm, $[T_0 + T_1]$, arises when the first successful broadcast is the second largest element and the second successful broadcast is the largest element. Thus, the layer between the second and the largest element is empty and needs T_0 time slots. And the layer between $x_0 = -\infty$ and the second largest element contains one element and needs T_1 time slots.

In third term, the subterm, $[R_2+T_0]$, arises when the first successful broadcast is the third largest element. Thus, after the first successful broadcast, there are two possible successful broadcast sequences, each having probability $\frac{1}{2}$. The first case is that the second successful broadcast is the largest data element. Thus, the layer between the first two successful broadcasts contains only one data element which is the second largest data element. The second case is that the second successful broadcast is the second largest data element, and the largest data element will be broadcast in the third successful broadcast. Thus, the layers between the first, second and third successful broadcasts, are both empty. Comparing with R_2 in Eq.(1), we can find that the above two cases are equivalent to those in R_2. Thus, R_2 represents the number of time slots required for the layers bounded by the sequence of successful broadcasts L_m in finding the maximum. Note that the time slots required for L_m is counted in M_3. Finally, the layer between $x_0 = -\infty$ and the third largest element is empty and needs T_0 time slots.

Generalizing Eq.(2), we have

$$T_n = M_n + R_n, \text{ where } R_n = \left\{ \begin{array}{lll} & \frac{1}{n}[& T_{n-1}] \\ + & \frac{1}{n}[T_0 & +T_{n-2}] \\ + & \frac{1}{n}[R_2 & +T_{n-3}] \\ + & \cdots & \\ + & \frac{1}{n}[R_{k-1} & +T_{n-k}] \\ + & \cdots & \\ + & \frac{1}{n}[\ R_{n-1} & +T_0] \end{array} \right\}. \tag{3}$$

In the above equations, M_n represents the average number of time slots required for finding the maximum among n data elements, and R_n represents the average number of time slots required for finishing sorting after the maximum is found.

In the equation for R_n, the first successful broadcast may be any one of the n elements randomly, each case having probability $\frac{1}{n}$. The subterm T_{n-1} of the second term arises when the first successful broadcast is the largest element. Thus, the layer between $x_0 = -\infty$ and the largest element contains $n-1$ elements and needs T_{n-1} time slots for sorting.

In the second term, the subterm, $T_0 + T_{n-2}$, arises when the first successful broadcast is the second largest element and the second successful broadcast is the largest element. Thus, the layer between the second and the largest element is empty and needs T_0 time slots. And the layer between $x_0 = -\infty$ and the second largest element contains $n-2$ elements and needs T_{n-2} time slots for sorting.

In the kth term, the subterm, $R_{k-1} + T_{n-k}$, arises when the first successful broadcast is the kth largest element. Thus, R_{k-1} is the number of time slots required for finishing the sorting on the $k-1$ elements after the maximum is found. And the layer between $x_0 = -\infty$ and the first successful broadcast element contains $n-k$ data elements and needs T_{n-k} time slots for sorting $n-k$ data elements.

Substituting $R_k = T_k - M_k$, $2 \leq k \leq n-1$ into Eq.(3), we have

$$T_n = M_n + \left\{ \begin{array}{ll} \frac{1}{n}[& T_{n-1}] \\ + \frac{1}{n}[T_0 & +T_{n-2}] \\ + \frac{1}{n}[(T_2 - M_2) & +T_{n-3}] \\ + \cdots & \\ + \frac{1}{n}[(T_{k-1} - M_{k-1}) & +T_{n-k}] \\ + \cdots & \\ + \frac{1}{n}[(T_{n-1} - M_{n-1}) & +T_0] \end{array} \right\}. \tag{4}$$

Lemma 1.

$$\frac{1}{n+1}T_n - \frac{1}{3}T_2 = \sum_{3 \leq k \leq n} \frac{1}{k+1}(M_k - M_{k-1}).$$

Lemma 2.

$$4(\frac{1}{3} - \frac{1}{n+1}) \leq \sum_{3 \leq k \leq n} \frac{1}{k+1}(M_k - M_{k-1}) \leq 5(\frac{1}{3} - \frac{1}{n+1}).$$

Theorem 1.

$$\frac{7}{2}n - \frac{1}{2} \leq T_n \leq \frac{23}{6}n - \frac{7}{6}.$$

5 Analysis of the generalization

T_n in Eq.(4) can be regarded as T_n^n. Therefore generalizing Eq.(4), we obtain

$$T_k^n = M_n + \frac{1}{n} \left\{ \begin{array}{ll} [& T_{k-1}^{n-1}] \\ + [T_0^0 & +T_{k-2}^{n-2}] \\ + [(T_2^2 - M_2) & +T_{k-3}^{n-3}] \\ + \cdots & \\ + [(T_{k-3}^{k-3} - M_{k-3}) & +T_2^{n-(k-2)}] \\ + [(T_{k-2}^{k-2} - M_{k-2}) & +T_1^{n-(k-1)}] \\ + [(T_{k-1}^{k-1} - M_{k-1}) & +0] \\ + [(T_k^k - M_k) & +0] \\ + \cdots & \\ + [(T_k^{n-1} - M_{n-1}) & +0] \end{array} \right\}.$$

In the above equations, the first term, M_n, is the average number of time slots required for the maximum finding algorithm on n data elements.

The second term, $\frac{1}{n}$, is one of the cases that each data element in M_n successfully broadcasts its value. Then, the first successful broadcast may be one of the n elements randomly, each case having probability $\frac{1}{n}$.

The third term $\left[T_{k-1}^{n-1}\right]$ is that the first successful broadcast is the largest data element in the progress of the maximum finding algorithm. Since we have the largest data element, we need to find the $k-1$ largest data elements among the $n-1$ data elements, which is presented by T_{k-1}^{n-1}.

The fourth term $\left[T_0^0 + T_{k-2}^{n-2}\right]$ is that the first successful broadcast is the second largest data element. Obviously the second successful broadcast is the largest data element. T_0^0 represents a $NULL$ layer because it is empty between the largest and second largest data elements. And the subterm T_{k-2}^{n-2} represents that we need to find the $k-2$ largest data elements among $n-2$ data elements after the largest and second largest data elements broadcasting.

The $(k+3)$th term $\left[(T_{k-1}^{k-1} - M_{k-1}) + 0\right]$ is that the first successful broadcast is the kth largest data element. Thus, $(T_{k-1}^{k-1} - M_{k-1})$ is the number of time slots required for finishing the sorting on the $k-1$ elements after the maximum is found. And since the first k, $k \geq 1$, largest elements was found in the layers upper than the first successful broadcast element, thus it dose not need any time slots for sorting data elements in the lower layer.

We omit the explanation of the rest of the equation, since it is quite similar to the above cases.

Lemma 3. *Let $L_n = M_n - M_{n-1}$, then*

$$T_k^n - T_k^{n-1} = L_n + \frac{1}{n}L_{n-1} + \frac{1}{n-1}L_{n-2} + \cdots + \frac{1}{n-(k-2)}L_{n-(k-1)}.$$

Lemma 4.

$$T_k^k + 4\left[\ln(n-(k-2)) - \ln 2\right] \leq T_k^n \leq T_k^k + 5\ln(n-(k-1)).$$

Theorem 2.

$$\frac{7}{2}k + 4\ln(n-(k-2)) - \left(\frac{1}{2} + 4\ln 2\right) \leq T_k^n \leq \frac{23}{6}k + 5\ln(n-(k-1)) - \frac{7}{6}.$$

Martel *et al.* [8] showed that the expected time for finding the first k largest numbers is $O(k + \log n)$. However, the constants associated with $O(k + \log n)$ bounds proved by Martel *et al.* [8] are not very tight. And they did not point out the relationship between the two special cases maximum finding and sorting.

By Theorem 2, we prove that the average time complexity of finding the first k numbers is $\Theta(k + \log(n-k))$. And the connection between the two special cases, maximum finding and sorting, is built by the analysis of our generalization.

6 Conclusion

The layer concept [13] can help us to reduce conflict resolution when an algorithm is not conflict-free under BCM. In this paper, we apply the layer concept to solve the sorting problem and get a tighter bound. The total average number of time slots, including conflict slots, empty slots and slots for successful broadcasts, is

$\Theta(n)$. And the generalization of our sorting algorithm can be analyzed that the average time complexity for selecting k largest numbers is $\Theta(k + \log(n-k))$.

On the high level view, our sorting algorithm has a similar approach as that proposed by Martel *et al.* [8]. On the low level view, our algorithm is based on the layer concept, thus it can be analyzed easily and a tighter bound can be obtained.

We are interested in finding other maximum finding algorithms which can be applied into our sorting approach and can get better performance. That is one of our future works. And the other future work is to find the native bound of the sorting approach under BCM.

References

1. Capetanakis, J.I.: Tree algorithms for packet broadcast channels. IEEE Transactions on Information Theory, **25(5)** (May 1997) 505–515
2. Dechter, R., Kleinrock, L.: Broadcast communications and distributed algorithms. IEEE Transactions on Computers, **35(3)** (Mar. 1986) 210–219
3. Grabner, P.J., Prodinger, H.: An asymptotic study of a recursion occurring in the analysis of an algorithm on broadcast communication. Information Processing Letters, **65** (1998) 89–93
4. Huang, J.H., Kleinrock, L.: Distributed selectsort sorting algorithm on broadcast communication. Parallel Computing, **16** (1990) 183–190
5. Levitan, S.: Algorithms for broadcast protocol multiprocessor. Proc. of 3rd International Conference on Distributed Computing Systems, (1982) 666–671
6. Levitan, S.P., Foster C.C.: Finding an extremum in a network. Proc. of 1982 International Symposium on Computer Architechure, (1982) 321–325
7. Martel, C.U.: Maximum finding on a multi access broadcast network. Information Processing Letters, **52** (1994) 7–13
8. Martel, C.U., Moh, M.: Optimal prioritized conflict resolution on a multiple access channel. IEEE Transactions on Computers, **40(10)** (Oct. 1991) 1102–1108
9. Martel, C.U., Moh W.M., Moh T.S.: Dynamic prioritized conflict resolution on multiple access broadcast networks. IEEE Transactions on Computers, **45(9)** (1996) 1074–1079
10. Ramarao, K.V.S.: Distributed sorting on local area network. IEEE Transactions on Computers, **C-37(2)** (Feb. 1988) 239–243
11. Shiau, S.H., Yang, C.B.: A fast maximum finding algorithm on broadcast communication. Information Processing Letters, **60** (1996) 81–96
12. Shiau, S.H., Yang, C.B.: The layer concept and conflicts on broadcast communication. Journal of Chang Jung Christian University, **2(1)** (June 1998) 37–46
13. Yang, C.B.: Reducing conflict resolution time for solving graph problems in broadcast communications. Information Processing Letters, **40** (1991) 295–302
14. Yang, C.B.,Lee, R.C.T., Chen, W.T.: Parallel graph algorithms based upon broadcast communications. IEEE Transactions on Computers, **39(12)** (Dec. 1990) 1468–1472
15. Yang, C.B., Lee, R.C.T., Chen, W.T.: Conflict-free sorting algorithm broadcast under single-channel and multi-channel broadcast communication models. Proc. of International Conference on Computing and Information, (1991) 350–359