Broadcast-Efficient Algorithms on the Coarse-Grain Broadcast Communication Model with Few Channels. *

Koji Nakano[†] Stephan Olariu[‡] James L. Schwing[‡]

Abstract

The main contribution of this work is to present elegant broadcast-efficient algorithms for permutation routing, ranking, and sorting n items on the Broadcast Communication Model (BCM, for short) endowed with p processors and k communication channels. We begin by presenting an optimal off-line routing algorithm using $\frac{n}{k}$ broadcast rounds for any k, p, and n. We then go on to develop an on-line routing algorithm that takes $2\frac{n}{k} + k - 1$ broadcast rounds on a p-processor, k-channel BCM, whenever $k \leq \sqrt{\frac{p}{2}}$. Using this routing algorithm, we develop a ranking algorithm that takes only $3\frac{n}{k} + o(\frac{n}{k})$ broadcast rounds, as well as a sorting algorithm that takes $4\frac{n}{k} + o(\frac{n}{k})$ broadcast rounds on a p-processor, k-channel BCM, provided that $k \leq \sqrt{\frac{p}{2}}$ and $p \in o(n)$. Our algorithms offer a significant improvement over the state of the art.

Keywords: *communication networks, wireless networks, mobile computing, permutation routing, ranking, sorting.*

1 Introduction

Broadcast communication plays a crucial role in disseminating information in computer networks. The Broadcasting Communication Model (BCM) proposed in the literature [1, 3] captures, in an elegant way, the essence of the issues encountered in both wired and wireless networks. The BCM(p, k) involves p processors (or stations) PE(1), PE(2), $\ldots$, PE(p) and k disjoint communication channels (radio frequencies, for example) C(1), C(2), $\ldots$, C(k). As an illustration, Figure 1 depicts a 7-processor BCM in a wireless environment. In unit time

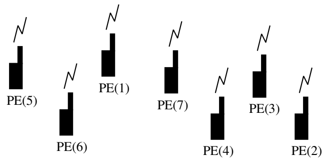

Figure 1. *Illustrating a 7-processor BCM*

every processor of the BCM can tune in to one of the channels and/or broadcast data on a channel. These two channels may or may not be distinct. As in [2, 3, 5, 6, 8], we assume that any number of processors may tune in (i.e. read) concurrently to the same channel but no two processors may broadcast on the same channel at the same time.

If the input to the problem is of size n then each processor of a BCM(p, k) is assumed to have a local memory of size at least $O(\frac{n}{p})$. We assume a slotted system [2, 3] where the duration of a slot corresponds to the time it takes to broadcast one packet. For this reason we assume that each packet consists of a data item along with $O(\log n)$ bits of additional data, such as the index of a processor, the number of items, etc. The set of all broadcasts that take place in the same time unit is referred to as a *broadcast round*. In addition to broadcast operations, the processors of a BCM can also perform local computation involving data residing in their local memory.

In many military applications it is important to minimize the number of broadcast rounds involved in solving a given problem. This is motivated by the need to conserve battery power in a mobile environment and to avoid detection as well. With this in mind, we set out to design protocols that lead to broadcast-efficient solutions to the problem of interest. In this work we are interested in the problems for n items pretiled on the BCM(p, k) as follows:

Permutation routing: Each item has a *unique* destination. The problem is to route all items to their destination. Since the routing must be a permutation, at the end of the routing each processor will store exactly $\frac{n}{p}$ items.

Ranking: The items are assumed to come from from a to-

*Work supported by NASA grant NASI-19858, by NSF grant CCR-9522093, and by ONR grant N00014-97-1-0526, by Grant-in-Aid for Encouragement of Young Scientists (09780262) from Ministry of Education, Science, Sports, and Culture of Japan, and by a grant from the Hori Information Science Promotion Foundation.

†Department of Electrical and Computer Engineering, Nagoya Institute of Technology, Showa-ku, Nagoya 466, JAPAN

‡Department of Computer Science, Old Dominion University, Norfolk, Virginia 23529, USA

tally ordered universe. The problem is to compute the rank of each item, that is, the number of items smaller than it.

Sorting: Sorting asks to perform a permutation routing of the items according to their ranks.

Dechter and Kleinrock [3] have addressed the problem of sorting n items on a single-channel, p-processor BCM, for $p \leq n$. They have shown that if broadcast conflicts can be resolved in constant time, then the n items can be sorted in $O(\frac{n}{p} \log \frac{n}{p} + n)$ time. More recently, Marberg and Gafni [4] have shown that on a k-channel, p-processor, conflict-free BCM a sequence of n items stored $\frac{n}{p}$ per processor can be sorted in $O(\frac{n}{k} + n_{max})$ time, where $k \leq p \leq n$, $k^2(k-1) \leq n$ and n_{max} is the largest number of items ever stored by a processor during the course of the algorithm. Yang *et al.* [8] have improved on the results of Marberg and Gafni [4] and Dechter and Kleinrock [3] by showing that the running time can be reduced to $O(\frac{n}{p} \log \frac{n}{p} + \frac{n}{k} \log^2 k)$. Nakano [5] presented a Columnsort-based sorting algorithm that runs in $O(\frac{n}{k})$ broadcast rounds on an n-processor, k-channel BCM, with $k \leq n^{1/4}$. Although Nakano's algorithm is asymptotically optimal, the coefficient of $\frac{n}{k}$ is rather large. Quite recently, Nakano *et al.* [6] presented a permutation routing algorithm using $2\frac{n}{k} + o(\frac{n}{k})$ broadcasting rounds and a sorting algorithm using $3\frac{n}{k} + o(\frac{n}{k})$ broadcasting rounds on a BCM(p, k) such that $p = n$ and $k \leq \sqrt{\frac{n}{\log n}}$.

The main contribution of this work is to present elegant broadcast-efficient algorithms for the problems of permutation routing, ranking, and sorting on the BCM model. Since each of n items must be broadcast at least once, a BCM endowed with p processors and k channels needs at least $\frac{n}{k}$ broadcast rounds to solve instances of size n of these problems. We begin by discussing a broadcast-optimal off-line permutation routing algorithm that uses $\frac{n}{k}$ broadcast rounds. This is an idealized algorithm in which the processors are controlled by an omniscient oracle. Inspired by this algorithm, we go on to design broadcast-efficient on-line permutation routing algorithm that takes $2\frac{n}{k} + k - 1$ broadcast rounds on a BCM(p, k) provided that $k \leq \sqrt{\frac{p}{2}}$. In particular, our algorithm works well whenever $k \ll p \ll n$, which is the case in the vast majority of practical situations, where the number n of items to be routed is much larger than the number p of processors, which in turn is much larger than the number k of communication channels available.

2 Off-Line Permutation Routing

This section shows an off-line permutation algorithm using the broadcasting rounds matching this lower bound. In the off-line algorithm, all the processors are controlled by an oracle, external to the BCM, that knows the distribution of all the input items. The off-line algorithm proceeds as follows. First, the oracle builds a bipartite graph from the input. The nodes of this graph correspond to the source and the destination processors. For each item to be routed from processor PE(i) to processor PE(j), an edge is drawn in the graph between node PE(i) and node PE(j). Since each processor has exactly $\frac{n}{p}$ items, the resulting bipartite graph is $\frac{n}{p}$-regular, i.e. every node is adjacent to $\frac{n}{p}$ other nodes. We make use of the following classic result [7].

Lemma 2.1 *For any $\rho \geq 1$, a ρ-regular bipartite graph is always ρ-edge-colorable.*

By Lemma 2.1, the bipartite graph constructed above can be painted by $\frac{n}{p}$ colors. Let $c_1, c_2, \ldots c_{\frac{n}{p}}$ be the $\frac{n}{p}$ color classes in the above coloring. Clearly, for every i, $(1 \leq i \leq \frac{n}{p})$, each set c_i involves p edges none of which share a node. The oracle mandates each processor to route its items according to the schedule implicit in the ordering $c_1, c_2, \ldots, c_{\frac{n}{p}}$ as follows. Each set c_i, $(1 \leq i \leq \frac{n}{p})$, uses $\frac{p}{k}$ broadcast rounds. Since there are only k channels, the oracle selects k edges from c_i and instructs the source processors incident with these edges to route the corresponding items to their destinations. Since no two edges in c_i share a node, no two items among the corresponding k items share a source processor or a destination processor. Thus, the k items can be routed, without any conflicts, in one broadcast round, using the k channels available. By repeating this operation, the p items in c_i can be routed in $\frac{p}{k}$ broadcast rounds. Altogether, the number of broadcast rounds is exactly $\frac{p}{k} \cdot \frac{n}{p} = \frac{n}{k}$ and we have the following result.

Lemma 2.2 *The off-line permutation routing of n items can be performed optimally in $\frac{n}{k}$ broadcast rounds on a BCM(p, k).*

3 On-Line Permutation Routing

In this section we show that the permutation routing of n items can be performed in $2\frac{n}{k} + k - 1$ broadcast rounds on a $O(\frac{n}{p})$-memory BCM(p, k), provided that $k \leq \sqrt{\frac{p}{2}}$.

Begin by partitioning the processors into k groups $G(1), G(2), \ldots, G(k)$ of $\frac{p}{k}$ processors each; similarly, partition the items into k groups $g(1), g(2), \ldots, g(k)$ of $\frac{n}{k}$ items each. The algorithm consists of two stages. In the first stage, having assigned channel C(i) to group $G(i)$, $(1 \leq i \leq k)$, we proceed to route the items in group $G(i)$ such that (1) each processor receives items from one $g(j)$, and (2) no processor contains more than $\frac{2n}{p}$ items.

Suppose that at the end of the first stage the processors are partitioned into k groups $H(1), H(2), \ldots, H(k)$ such that $H(i)$, $(1 \leq i \leq k)$, is the set of processors containing items in $g(i)$. In the second stage, in order to move the items in $g(i)$ to their final destination, channel $C(i)$ is assigned to the processors in $H(i)$.

We now provide a detailed implementation of the first stage. We only focus on the routing that takes place in group $G(1)$ using channel $C(1)$, the broadcasts on all other channels following a similar pattern. Initially, the first k processors PE(1), PE$(2), \ldots,$ PE(k) are used as *main processors*: processor PE(i), $(1 \leq i \leq k)$, has the mandate to store items from $g(i)$. The remaining processors PE$(k+1)$, PE$(k+2), \ldots,$ PE$(\frac{p}{k})$ act as *overflow processors* and will be used to store items when one of the main processors becomes full.

Each processor maintains k counters $c(1), c(2), \ldots, c(k)$, initialized to zero. Each counter $c(j)$ is used to record the number of items in group $g(j)$ broadcast, thus far, on channel C(1). Hence, the values of the $c(j)'s$ are the same in all processors in $G(1)$. Note that, if k is much larger than $\frac{n}{p}$, the processors may not have enough local memory to store the k counters. Hence, we assume that $k \leq \frac{n}{p}$. Later, we show how to modify the algorithm to handle the case $k > \frac{n}{p}$.

In the first stage each processor in $G(1)$ broadcasts its items, one by one, on channel C(1). All processors in $G(1)$ read the channel and increment $c(j)$, whenever the item broadcast is in $g(j)$. The corresponding main processor stores the item in its own local memory. The first unused overflow processor checks whether $c(j)$ mod $\frac{2n}{p} = 0$, that is, if the current main processor for $g(j)$ is already full. If so, it will take over as the current main processor. Since each item is broadcast only once, the first stage takes $\frac{n}{k}$ broadcast rounds. The readers should have no difficulty to confirm that each group $G(1)$ has enough processors to satisfy conditions (1) and (2) above.

We now turn to the implementation of the second stage, whose mandate is to route the items of $g(i)$, $(1 \leq i \leq k)$, from the processors in $H(i)$ to those in $G(i)$. For this purpose, we dedicate channel C(i) to the items in $g(i)$. The second stage begins with a preamble in which information about the number of items in $g(i)$ stored by the processors in each group $G(j)$, $(1 \leq i \leq k-1)$, is broadcast on channel C(i). The idea is to allow every processor in $H(i)$ to know the exact moment at which it should start broadcasting the items in $g(i)$ it stores as a result of the data movement in the first stage. To understand how this is done, consider the set of processors in $H(i) \cap G(j)$. Each of these processors, with the exception of exactly one of them, contains exactly $\frac{2n}{p}$ items from $g(i)$. Moreover, with one exception, all of these processors were, initially, overflow processors. Clearly, each of them remembers the number of items in the corresponding counter at the moment when it took over as the current main processor. In other words, it remembers its relative position among the processors in $H(i) \cap G(j)$. Consequently, the last such processor (which may not be full) knows exactly the number of items of $g(i)$ stored by the processors in $H(i) \cap G(j)$. It is this number that will be broadcast, in the preamble, on channel C(i).

The reader should have no difficulty to confirm that in $k-1$ broadcast rounds all the processors storing items in $g(i)$ learn the exact moment in time at which they should start broadcasting. Once the preamble is out of the way, the processors begin broadcasting items. We only focus on the routing of the items in $g(1)$ using channel C(1), the broadcasts on all the other channels being similar. Each processor in $H(1)$, $(1 \leq i \leq k)$, broadcasts its items on channel C(1) and the processors in $G(1)$ monitor channel C(1) on which items in $g(1)$ will be sent. Evidently, a processor picks up an item only if its identity matches the destination of the item being broadcast. First, the processors in $G(1) \cap H(1)$ broadcast their items, one by one, on channel C(1). The process is continued, as described, until all the items in $g(1)$ have been broadcast to their destination. Since each item is broadcast once and since $g(1)$ has $\frac{n}{k}$ items the second stage requires exactly $\frac{n}{k}$ broadcast rounds. Furthermore, as discussed, the preamble to the second stage takes exactly $k-1$ broadcast rounds. Thus, the second stage involves at most $\frac{n}{k} + k - 1$ broadcast rounds, and we have the following result.

We now show how to modify the first stage of the algorithm to handle the case $k > \frac{n}{p}$. To save local storage, each processor maintains a counter for $c(j)$ only if it has an item in $g(j)$. Specifically, if processor PE(1) has items in $g(j_1), g(j_2), \ldots$, $(1 \leq j_1 < j_2 < \cdots)$, it maintains counters $c(j_1), c(j_2), \ldots$. In the previous algorithm, the first unused overflow processor could check the value of $c(j)$ mod $\frac{2n}{p}$. This is no longer possible, as the processors no longer have counters for all the $g(i)$'s. To solve this problem, the processor broadcasting an item also checks whether the current broadcast will completely fill the corresponding main processor. If such is the case, that is, if $c(j)$ mod $\frac{2n}{p} = 0$ after incrementing $c(j)$, the processor attaches a *control tag* indicating overflow. Now, the first unused overflow processor learns that it must take over from the main processor. Clearly, this modification does not increases the number of broadcast rounds. Thus, we have

Theorem 3.1 *The permutation routing of n items on a BCM(p,k) can be performed in $2\frac{n}{k} + k - 1$ broadcast rounds, provided that $k \leq \sqrt{\frac{p}{2}}$.*

4 Ranking and Sorting on a BCM

In this section we are interested in developing broadcast-efficient ranking and sorting algorithms for a coarse-grain BCM endowed with *few* channels. In addition to relying on the permutation routing algorithms developed in the previous section, our ranking and sorting algorithms benefit from an elegant sampling scheme that we discuss next.

4.1 Our Sampling Scheme

Consider a collection $A = \{a(1), a(2), \ldots, a(n)\}$ of n items. We assume, without loss of generality, that all the items in A are distinct. Our sampling scheme partitions the n items into k, $(1 \le k < \sqrt{n})$, buckets $B(1), B(2), \ldots, B(k)$ such that

- $B(1) < B(2) < \ldots < B(k)$, that is, for $1 \le i \le k-1$, every item in $B(i)$ is smaller than every item in $B(i+1)$, and
- each bucket $B(i)$ contains at most $\frac{n}{k} + O(\sqrt{n})$ items.

We refer the reader to Figure 2 for an illustration of our sampling scheme.

Suppose that the n items are partitioned into k subsets $A(1), A(2), \ldots, A(k)$ of size $\frac{n}{k}$ each, and enumerate each $A(i)$ in sorted order as $a(i,1) < a(i,2) < \cdots < a(i, \frac{n}{k})$. Further, partition each $A(i)$ into $\sqrt{n}$ groups $A(i,1), A(i,2), \ldots, A(i, \sqrt{n})$ of size $\frac{\sqrt{n}}{k}$ each such that for each j, $(1 \le j \le \sqrt{n})$, $A(i,j)$: $a(i, (j-1)\frac{\sqrt{n}}{k} + 1), a(i, (j-1)\frac{\sqrt{n}}{k} + 2), \ldots, a(i, j\frac{\sqrt{n}}{k})$. Therefore, the n items are partitioned into $k\sqrt{n}$ groups. Let Sample(A) be the set of $k\sqrt{n}$ items consisting of the smallest item in each such group. Enumerate the items in Sample(A) in sorted order as $s(1) < s(2) < \cdots < s(k\sqrt{n})$. Next, Sample($A$) is partitioned into k subsets $S(1), S(2), \ldots, S(k)$ each of size $\sqrt{n}$, such that $S(i)$: $s((i-1)\sqrt{n}+1), s((i-1)\sqrt{n}+2), \ldots, s(i\sqrt{n})$. Let Pivot($A$) : $v(1) < v(2) < \cdots < v(k)$ consist of the smallest item in each of the subsets $S(1), S(2), \ldots, S(k)$. In other words, $v(i) = s((i-1)\sqrt{n}+1)$. Finally, partition A into k buckets $B(1), B(2), \ldots B(k)$, such that

- $B(j) = \{a \in A \mid v(j) \le a < v(j+1)\}$ for $1 \le j < k$, and
- $B(k) = \{a \in A \mid v(k) \le a\}$.

The readers should have no difficulty to confirm the following lemma.

Lemma 4.1 *The following hold:*

1. $B(1) < B(2) < \ldots < B(k)$, *and*
2. *for every* i, $(1 \le i \le k)$, *bucket* $B(i)$ *contains at most* $\frac{n}{k} + O(\sqrt{n})$ *items.*

4.2 Our Ranking and Sorting Algorithms

Next, we demonstrate a ranking algorithm for the coarse-grain BCM(p,k) with $k \le \sqrt{\frac{p}{2}}$. Partition the processors into k groups $G(1), G(2), \ldots, G(k)$ of $\frac{p}{k}$ processors each, and let $A(i) = \{a((i-1)\cdot\frac{n}{k}+1), a((i-1)\cdot\frac{n}{k}+2), \ldots, a(i\cdot\frac{n}{k})\}$ denote the items stored by the processors in group $G(i)$, $(1 \le i \le k)$. We find it convenient to import the notation and terminology of the previous subsection. The ranking algorithm involves the following seven phases.

Phase 1 Rank each set $A(i)$ in $\frac{n}{k}$ broadcast rounds;

Phase 2 Construct and sort Sample(A) in $\frac{p}{k} + O(\sqrt{n})$ broadcast rounds;

Phase 3 Retains Pivot(A) from Sample(A) and define the buckets $B(1), B(2), \ldots, B(k)$ in k broadcast rounds;

Phase 4 Route the items within each $G(i)$ in $\frac{n}{k}$ broadcast rounds such that

(4.1) each processor has items in exactly one bucket $B(j)$, and

(4.2) no processor contains more than $\frac{2n}{p}$ items.

Phase 5 Compute the rank of each item within each bucket in $\frac{n}{k} + O(\sqrt{n}) + k - 1$ broadcast rounds.

Phase 6 In k broadcast rounds compute the rank of the smallest items in each bucket with respect to $A = A(1) \cup A(2) \cup \cdots \cup A(k)$;

Phase 7 Determine the rank of all input items by adding this rank to the rank of each item within $B(i)$ and by subtracting one.

The correctness of this algorithm being obvious, we now turn to implementation details. Phase 1 assigns a channel to each $G(i)$ and broadcast every item in $G(i)$ in turn. Since the number of items smaller than each item can be computed, the rank of every item $A(i)$ in $\frac{n}{k}$ broadcast rounds.

In Phase 2, using the rank information from Phase 1, each processor identifies which of its own items should be in Sample(A). More precisely, an item is in Sample(A) if and only if its rank modulo $\frac{\sqrt{n}}{k}$ is 1. Note that each group has $\sqrt{n}$ items in Sample(A). If $\sqrt{n} \le \frac{p}{k}$, then route the $\sqrt{n}$ items in Sample(A) to the first $\sqrt{n}$ processors in each group. To do this, each processor broadcasts, one by one, the items in Sample(A) on the channel. Note that if a processor has no item in Sample(A) this will be indicated by a special signal. Since each subset has $\sqrt{n}$ items in Sample(A), and since each group has $\frac{p}{k}$ processors, this task can be performed in $\sqrt{n} + \frac{p}{k} - 1$ broadcasts. If $\sqrt{n} > \frac{p}{k}$, then route the items in Sample(A) such that each processor has $\frac{k\sqrt{n}}{p}$ items. This can be performed in $\sqrt{n} + \frac{p}{k} - 1$ broadcast rounds in the obvious way. Next, sort Sample(A) by using a sorting algorithm on the BCM [5]. Since Sample(A) has $k\sqrt{n}$ items and since k channels are available, this task can be performed in $O(\sqrt{n})$ broadcast rounds. Altogether, therefore, Phase 2 involves $\frac{p}{k} + O(\sqrt{n})$ broadcast rounds.

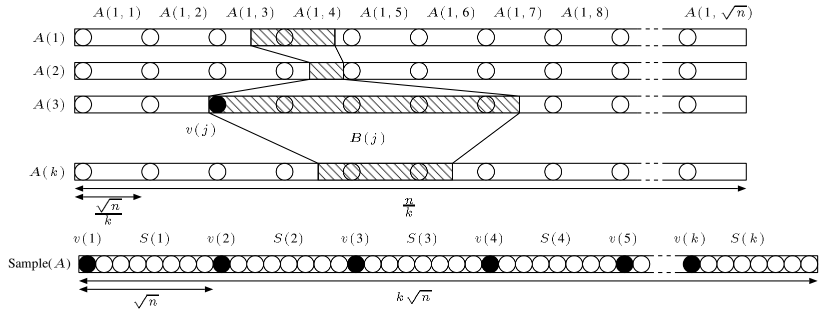

Figure 2. *Illustrating our sampling scheme.*

In Phase 3 each processor which has items in Sample(A) identifies its own items in Pivot(A). Since Sample(A) is sorted, each processor has at most one item in Pivot(A). Thus, each processor which has an item $v(i) \in$ Pivot(A) broadcasts the items $v(i)$ on channel C(1), one by one, using k broadcast rounds. Attach a tag j to each item that belongs to bucket $B(j)$.

Phase 4 routes items using the first phase of the algorithm of Theorem 3.1. This phase uses $\frac{n}{k}$ broadcast rounds.

Phase 5 computes the rank of each item within each bucket $B(j)$, $(1 \leq j \leq k)$, in the same way as Phase 1. For this purpose, broadcast items in bucket $B(j)$ on $C(j)$ one by one. Since all items are routed to satisfy the two conditions in Phase 4, this can be performed by the second phase of the algorithm of Theorem 3.1. Since each bucket $B(j)$ has at most $\frac{n}{k} + O(\sqrt{n})$ items, Phase 5 can be performed in $\frac{n}{k} + O(\sqrt{n}) + k - 1$ broadcast rounds. During this broadcast, the number $m(j)$ of items in bucket $B(j)$ can be computed in the obvious way.

To compute the rank of the smallest item in $B(j)$, Phase 6 computes the prefix-sums of $m(1), m(2), \ldots, m(k)$. This can be achieved in k broadcast rounds using a single channel.

Phase 7 broadcasts the rank of the smallest item in each $B(j)$, $(1 \leq j \leq k)$, on channel $C(j)$. Processors having items in $B(j)$ can now compute the correct rank of each item in the obvious way. This can be done in one broadcast round.

By summing up the numbers of broadcast rounds performed in all steps, this algorithm uses $3\frac{n}{k} + \frac{p}{k} + 3k + O(\sqrt{n})$ broadcast rounds. Since $k \leq \sqrt{\frac{p}{2}}$ and $p \in O(n)$, we have, $k = O(\sqrt{n})$. Consequently, we have the following result.

Theorem 4.2 *The task of ranking a set of n items can be performed in $3\frac{n}{k} + \frac{p}{k} + O(\sqrt{n})$ broadcast rounds on a BCM(p, k), provided that $k \leq \sqrt{\frac{p}{2}}$.*

To complete the sorting after ranking, we need to permute the items by their ranks. For this purpose, each channel C(i), $(1 \leq i \leq k)$, is used to broadcast items in bucket $B(i)$ in the same way as the second stage of the algorithm of Theorem 3.1. Since each bucket has at most $\frac{n}{k} + O(\sqrt{n})$ items, this routing task can be performed in $\frac{n}{k} + O(\sqrt{n}) + k - 1$ broadcast rounds. Thus, we have

Theorem 4.3 *The task of sorting n items can be performed in $4\frac{n}{k} + \frac{p}{k} + O(\sqrt{n})$ broadcast rounds on a BCM(p, k), provided that $k \leq \sqrt{\frac{p}{2}}$.*

References

[1] J. I. Capetanakis, Tree algorithms for packet broadcast channels, *IEEE Transactions on Information Theory*, IT-25, (1979), 505–515.

[2] I. Chlamtac and S. Kutten, Tree-based broadcasting in multihop radio networks, *IEEE Transaction on Computers*, C-36, (1987), 1209–1223.

[3] R. Dechter and L. Kleinrock, Broadcast communication and distributed algorithms, *IEEE Transactions on Computers*, C-35, (1986), 210–219.

[4] J. M. Marberg and E. Gafni, Sorting and selection in multi-channel broadcast networks, *Proc. International Conference on Parallel Processing*, St-Charles, Illinois, August 1985, 846–850.

[5] K. Nakano, Optimal sorting algorithms on Bus-Connected Processor Arrays, *IEICE Transactions Fundamentals*, E-76A, 11, (1994), 2008–2015.

[6] K. Nakano, S. Olariu, and J.L. Schwing, Broadcast-efficient sorting in the presence of few channels, *Proc. International Conference on Parallel Processing*, August 1997, 12–15.

[7] R. J. Wilson, *Introduction to Graph Theory*, Longman, 3rd edition, 1985.

[8] C. B. Yang, R. C. T. Lee, and W.-T. Chen, Conflict-free sorting algorithms under single and multi-channel broadcast communication models, *Proc. ICCI'91*, LNCS 497, Springer-Verlag, 1991, 350–359.