An Optimal Randomized Ranking Algorithm on the k-channel Broadcast Communication Model *

Koji Nakano
School of Information Science
Japan Advanced Insitute of Science and Technology
Tatsunokuchi, Ishikawa 923-1292, JAPAN
knakano@jaist.ac.jp

Abstract

A Broadcast Communication Model (BCM, for short) is a distributed system with no central arbiter populated by n processing units referred to as stations. The stations can communicate by broadcasting/receiving data packets in one of k communication channels. We assume that the stations run on batteries and expands power while broadcasting/receiving a data packet. Thus, the most important measure to evaluate algorithms on the BCM is the number of awake time slots, in which a station is broadcasting/receiving a data packet. We also assume that the stations are identical and have no unique ID number, and no station knows the number n of the stations. For given n keys one for each station, the ranking problem asks each station to determine the number of keys in the BCM smaller than its own key. The main contribution of this paper is to present an optimal randomized ranking algorithm on the k-channel BCM. Our algorithm solves the ranking problem, with high probability, in $O(\frac{n}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$ time slots. We also prove that any randomized ranking algorithm is required to run in expected $\Omega(\frac{n}{k} + \log n)$ time slots with at least one station being awake for expected $\Omega(\log n)$ time slots. Therefore, our ranking algorithm is optimal.

1 Introduction

A *Broadcast Communication Model* (BCM, for short) is a distributed system with no central arbiter populated by p processing units referred to as *stations*. The stations are identical and cannot be distinguished by serial or manufacturing numbers. Further, no station knows the number n of stations in the BCM. For definiteness, the stations are assumed to have the computing power of the random access machine (RAM), and to be synchronous – in particular, they all run the same algorithm and can generate random bits that provide "local data" on which the stations may perform computations. The stations can communicate using k distinct communication channels. We assume that in a unit time slot, each station can broadcast a *data packet* to one of the channels and can receive a data packet from one (possibly the same) channel.

The fundamental characteristic of the model is the broadcast nature of communications. A data packet broadcast in a channel can be received by every station tuned to the channel. The nature of the stations is immaterial: they could be processors in a parallel computing environment or radio transceivers in a wireless network. Likewise, the nature of the broadcast is immaterial: it could be a global bus in a multiprocessor system or a radio frequency channel in a radio network. It is important to note that the BCM model provides a common generalization of bus-based parallel architectures, cluster computing environment, local area networks, and single-hop radio networks. Although the BCM is assumed to be operate in synchronous mode, we do not prescribe a particular synchronization mechanism. We feel that this is best left to the particular application.

We assume that the BCM has the following collision detection capability. In the BCM, the status of a channel is: *NULL:* if no station broadcasts on the channel in the current time slot; *SINGLE:* if exactly one station broadcasts on the channel in the current time slot; and *COLLISION:* if two or more stations broadcast on the channel in the current time slot. The status of a channel can be detected by stations that tune to it. We also assume that broadcasting/receiving a data packet in the BCM is very costly. If the stations run on batteries, saving battery power is exceedingly important, as recharging batteries may not be possible while on mission. It is well known that a station expends power while

*Work supported in part by the Ministry of Education, Science, Sports, and Culture, Government of Japan, Grant-in-Aid for Young Scientist Research (14780220)

its transceiver is active, that is, while broadcasting or receiving a packet. Consequently, we are interested in developing algorithms that allow the stations to power their transceiver off (i.e. go to sleep) to the largest extent possible. Accordingly, we judge the goodness of a algorithm by the following two yardsticks: *running time slots*: the overall number of time slots required by the algorithm to terminate, *awake time slots*: for each individual station the total number of time slots when it has to be *awake* in order to broadcast/receive a data packet.

We formalize the *ranking problem* on the BCM as follows: Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n distinct keys with total order. The *rank* $r(x_i)$ of key x_i $(1 \le i \le n)$ is the number of keys in X smaller than x_i. We assume that the n keys are distributed to the n stations so that each station is storing one key. The ranking problem asks each station storing key x_i $(1 \le i \le n)$ to compute the rank $r(x_i)$. We also assume that a data packet transferred through a channel can contain at most one key. It is clear that $\Omega(\frac{n}{k})$ time slots are required to solve the ranking problem under this assumption, because every key must be broadcast at least once.

If all of the n stations have unique IDs in $[1, n]$, we can use known efficient parallel sorting algorithms to design energy-efficient ranking algorithms on the BCM. For example, the AKS sorting network [1], which is one of the most efficient parallel sorting algorithms can be implemented on the the k-channel n-station BCM $(k \le n)$ as follows: The AKS sorting network sorts n keys by repeating $\frac{n}{2}$ parallel compare-exchanges for $O(\log n)$ times. Each compare-exchange operation compares two keys and exchanges them if necessary. Since k channels are available, k keys can be transferred in a time slot. Hence, $\frac{n}{2}$ parallel compare-exchanges can be implemented in $\frac{n}{k}$ time slots with each station being awake for two time slots. Since this operation is repeated for $O(\log n)$ times, the AKS sorting network can be implemented on the k-channel BCM to run in $O(\frac{n \log n}{k})$ time slots with no station being awake for more than $O(\log n)$ time slots. Note that, after sorting, each key x_i is stored in station $r(x_i)$. By routing $r(x_i)$ to station i in obvious way, the ranking problem can be solved in $O(\frac{n \log n}{k})$ time slots with no station being awake for more than $O(\log n)$ time slots. However, this algorithm is not time-optimal for the $\Omega(\frac{n}{k})$ lower bound.

Several results of sorting have been obtained in the context on limited bandwidth. Nakano [11] showed that the n-processor network with k global buses, which is essentially the same as the the k-channel n-station BCM, can sort n keys in $O(\frac{n}{k})$ if $k \le n^{1-\epsilon}$ for any small fixed $\epsilon > 0$. Later, it has shown in [13] that the k-channel n-station BCM can sort n keys in $3\frac{n}{k} + o(\frac{n}{k})$ time slots provided that $k \le \sqrt{\frac{n}{2}}$. The running time slots of these algorithms are optimal for the $\Omega(\frac{n}{k})$ lower bound. However, these algorithms are not energy-efficient, because the stations must be awake for almost all the $O(\frac{n}{k})$ running time slots. Further, these algorithms cannot be adapted to the BCM with many channels. The challenge is to find a ranking algorithm which runs on the k-channel BCM, for every k, in $O(\frac{n}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$. The main contribution of this paper is to present a randomized ranking algorithm that attains this performance with high probability.

We first present a ranking algorithm that runs on the k-channel n-station BCM for every k, with high probability, in $O(\frac{n}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$ time. We then go on to show the lower bounds of the running and the awake time slots for the ranking on the k-channel n-station BCM. We first prove that, if each station can store a constant number of keys in its local memory, then every randomized algorithm needs to run in expected $\Omega(\frac{n}{k} + \log n)$ time slots with at least one station being awake for expected $\Omega(\log n)$ time slots. Thus, our algorithm is optimal, because it uses a limited size of storage. We also show that, even if every station has unlimited size of local memory and is allowed to perform unlimited number of operations on the local memory in each time slot, the ranking problem requires to run in expected $\Omega(\frac{n}{k} + \frac{\log n}{\log \log n})$ time slots with with at least one station being awake for expected $\Omega(\frac{\log n}{\log \log n})$ time slots.

2 Outline of our ranking algorithm

In this section, we show outline of our ranking algorithm. The detailed implementation will be discussed in Section 5. Throughout this paper, $\log$ denotes the logarithm to the base 2 unless otherwise mentioned.

Our ranking algorithm first picks $\frac{n}{\log n}$ keys uniformly at random as *samples* from $X = \{x_1, x_2, \ldots, x_n\}$ and sorts them. Let p_i $(0 \le i \le \frac{n}{\log n} - 1)$ denote the i-th smallest sample keys. We take every $\log n$ keys in sorted order as *pivots* from the $\frac{n}{\log n}$ keys. Let $q_j = p_{j \log n}$ $(0 \le j \le \frac{n}{(\log n)^2} - 1)$ denote pivots thus obtained. We partition the n keys into $\frac{n}{(\log n)^2} + 1$ groups $Q(0), Q(1), \ldots, Q(\frac{n}{(\log n)^2})$ such that each $Q(j)$ contains the keys larger than or equal to q_{j-1} and smaller than q_j, where $q_{-1} = -\infty$ and $q_{\frac{n}{(\log n)^2}} = +\infty$ for convenience. Let a_i be the index of Q such that x_i is in group $Q(a_i)$. In other words, each key x_i is in the range $[q_{a_i - 1}, q_{a_i})$. We compute the *local rank* $l(i)$ of each key x_i $(1 \le i \le n)$ within the group $Q(a_i)$, which is the number of smaller keys in $Q(a_i)$. We then compute the prefix-sum $|Q(0)| + |Q(1)| + \ldots + |Q(j)|$ for every j $(1 \le j \le \frac{n}{(\log n)^2})$. Let $g(i)$ $(1 \le i \le n)$ be the prefix-sum $|Q(0)| + |Q(1)| + \ldots + |Q(a_i - 1)|$ if $a_i \ge 1$ and 0 if $a_i = 0$. Finally, each station storing x_i $(1 \le i \le n)$ computes $l(i) + g(i)$, which is equal to the desired rank

$r(x_i)$ of x_i.

Figure 1 illustrates the values of local variables computed in our ranking algorithm for $n = 27$ stations. In the figure, for convenience, the log to the base 3 is used, so $\log n = \log_3 n = 3$. The leftmost 9 keys are chosen as samples, and they are sorted. Three pivots 11, 31, and 49 are selected from the samples, and they are used to partition into four groups.

Surprisingly, as we are going to show in Section 5, groups $Q(i)$s are well balanced so that none of the $\frac{n}{(\log n)^2}$ groups has more than $16(\log n)^2$ keys with high probability. Using this fact, stations in $Q(i)$ can concentrate on the task for $Q(i)$ and can sleep during that for the other groups. This idea enables us to obtain an energy-efficient ranking algorithms.

3 A brief refresher of probability theory

We first review elementary probability theory results that are useful for analyzing the performance of our ranking algorithm. For a more detailed discussion of background material we refer the reader to [10]. We then go on to show the analysis of several probabilistic events, which is used to evaluate our ranking algorithm.

Throughout, $\Pr[A]$ denotes the probability of event A. Let X be a random variable denoting the number of successes in n independent Bernoulli trials with success probability p. It is well known that X has a *binomial distribution* and that for every r, $(0 \leq r \leq n)$,

$$\Pr[X = r] \;=\; \binom{n}{r} p^r (1-p)^{n-r}.$$

Further, the expected value of X is given by

$$E[X] \;=\; \sum_{r=0}^{n} r \cdot \Pr[X = r] = np.$$

To analyze the tail of the binomial distribution, we shall make use of the following estimates, commonly referred to as *Chernoff bounds* [10]:

$$\Pr[X < (1-\epsilon)E[X]] \;<\; e^{-\frac{\epsilon^2}{2}E[X]} \;\; (0 \leq \epsilon \leq 1). (1)$$
$$\Pr[X > (1+\epsilon)E[X]] \;<\; e^{-\frac{\epsilon^2}{3}E[X]} \;\; (0 \leq \epsilon \leq 1). (2)$$

For later reference, we state two lemmas which can be obtained using the Chernoff bounds. Due to the stringent page limitation we omit their proofs. The proofs will be shown in the journal version of this paper.

Lemma 3.1 *For every n and m, in order to have n head coins, a fair coin is flipped at most $6n + 10m$ times with probability at least $1 - e^{-m}$.*

Lemma 3.2 *Suppose that, for enough large n, we have $\frac{n}{\log n}$ red and $n - \frac{n}{\log n}$ black bins each of which can store one ball, and have n balls partitioned into $\frac{n}{8(\log n)^2}$ groups with $8(\log n)^2$ balls each. If the n balls are thrown into the n bins at random, every group occupies at least $\log n$ red bins with probability at least $1 - n^{-2}$.*

4 Subalgorithms used in our ranking algorithm

The main purpose of this section is to show fundamental algorithms necessary to implement our ranking algorithm on the BCM.

4.1 PRAM simulation, sorting, and computing prefix-sums

The *CRCW-PRAM* is a parallel machine with synchronous processors and a shared memory [6, 7]. We assume that, if two or more processors write on the same memory cell simultaneously, a special collision symbol is written on it. We are going to show that CRCW-PRAM algorithm can be implemented on the k-channel BCM efficiently. Without loss of generality, we assume that one CRCW-PRAM step is either a *reading step* or a *writing step*. In a reading step, processors can read from one of the shared memory cells, and no processor writes on the shared memory. Similarly, no processor reads from the shared memory in a *writing step*. We will show that reading and writing steps on the PRAM can be simulated by stations on the BCM.

Suppose that an algorithm on the CRCW-PRAM runs in $T(n)$ time using n processors and $M(n)$ shared memory cells. To simulate this algorithm, $M(n)$ shared memory cells are equally assigned to n stations at most $\lceil \frac{M(n)}{n} \rceil$ cells each. Each processor maintains the assigned $\lceil \frac{M(n)}{n} \rceil$ cells stored in its local memory. To simulate a reading step, a station broadcast the values of $\lceil \frac{M(n)}{n} \rceil$ memory cells in turn into k channels. More specifically, the first k stations broadcast the value of the assigned memory cells in $\lceil \frac{M(n)}{n} \rceil$ time slots. The remaining stations repeat the same process. The broadcast by k stations is repeated for $\lceil \frac{n}{k} \rceil$ times at all. Since every memory cell is broadcast, the reading operation by the PRAM processor can be simulated by receiving a data packet on the corresponding channel in an appropriate time slot. Clearly, this task can be done in $\lceil \frac{n}{k} \rceil \lceil \frac{M(n)}{n} \rceil = O(\frac{M(n)}{k} + 1)$ time slots. Further, every station broadcasts in $\lceil \frac{M(n)}{n} \rceil$ time slots and receives a packet in at most one time slot, and thus no station is awake for more than $O(\frac{M(n)}{n} + 1)$ time slots. The writing operation by PRAM processors can be simulated by stations similarly, Thus, we have the following important lemma:

$\boxed{11}$	17	29	$\boxed{31}$	33	41	$\boxed{49}$	57	62
q_0			q_1			q_2		

	62	17	31	29	57	41	33	11	49	68	23	4	69	32	47	43	53	12	5	24	8	51	71	21	37	36	15
a	3	1	2	1	3	2	2	1	3	3	1	0	3	2	2	2	3	1	0	1	0	3	3	1	2	2	1
l	4	3	0	7	3	5	2	0	0	5	5	0	6	1	7	6	2	1	1	6	2	1	7	4	4	3	2
g	19	3	11	3	19	11	11	3	19	19	3	0	19	11	11	11	19	3	0	3	0	19	19	3	11	11	3
r	23	6	11	10	22	16	13	3	19	24	8	0	25	12	18	17	21	4	1	9	2	20	26	7	15	14	5

Figure 1. The values of local variables computed in our ranking algorithm

Lemma 4.1 *Any parallel algorithm running on the CRCW-PRAM in $T(n)$ time using n processors and $M(n)$ shared memory cells can be simulated on the k-channel BCM with n stations in $(\frac{T(n)M(n)}{k} + T(n))$ time slots with no station being awake for more than $O(\frac{T(n)M(n)}{n} + T(n))$ time slots.*

It is well-known that the EREW-PRAM with n processors can sort n keys in $O(\log n)$ time by the AKS sorting network [1]. Since $T(n) = O(\log n)$ and $M(n) = O(n)$, the k-channel BCM with n stations can sort n keys in $O(\frac{n \log n}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$ time slots. However, the AKS sorting network has an extremely large constant coefficient in $T(n)$. If our ranking algorithm uses the AKS sorting network as a subroutine, it may have unrealistic large constant factor in time slots. On the other hand, Cole's optimal merge sort, which terminates in $O(\log n)$ time using n processors on the EREW-PRAM [5] has a small constant factor. However, it needs $M(n) = O(n \log n)$ shared memory cells. Alternatively, we use the $\sqrt{n}$ *sample sort* [7, pp.470] which runs, with probability at least $1 - O(n^{-6})$, in $O(\log n)$ time using n-processor and $O(n)$ shared memory cells on the CRCW-PRAM. Using the $\sqrt{n}$ *sample sort* and the simulation result in Lemma 4.1, we have,

Corollary 4.2 *The task of sorting n keys in the k-channel BCM with n stations can be done in $O(\frac{n \log n}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$ time slots with probability at least $1 - O(n^{-6})$.*

It is known that, by recursive execution of Columnsort [9], sorting can be done efficiently on limited communication bandwidth models [2, 11]. For example, suppose that, for arbitrary fixed $\epsilon > 0$, $n^{1-\epsilon}$ channels are available to sort n keys. It has been shown in [11] that the sorting can be done in $O(n^\epsilon)$ time slots by repeating the Columnsort. Since the communication of each channel can be simulated by a shared memory cell in an obvious way, the sorting of n keys on the n-processor PRAM can be done in $T(n) = O(n^\epsilon)$ time using $M(n) = n^{1-\epsilon}$ shared memory cells. Thus, from the simulation result in Lemma 4.1, we have

Corollary 4.3 *For every fixed $\epsilon > 0$, the task of sorting n keys in the k-channel BCM with n stations can be done in $O(\frac{n}{k} + n^\epsilon)$ time slots with no station being awake for more than $O(n^\epsilon)$ time slots.*

It is well known that the n-processor EREW-PRAM can compute the prefix-sums of n numbers in $O(\log n)$ time using $O(n)$ memory cells [6, pp.13]. Thus, by according to the simulation result in Lemma 4.1, we have

Corollary 4.4 *The task of computing the prefix-sums of n numbers in the k-channel BCM with n stations can be done in $O(\frac{n \log n}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$ time slots.*

By a complicated implementation of the prefix-sums algorithm, the awake time slots can be reduced to $O(1)$ [15]. We do not have to use this complicated implementation for the ranking.

4.2 Initialization

This subsection is devoted to show initialization algorithms that gives unique IDs in the range to $[1, n]$ to the n stations in the BCM.

We briefly explains a non-energy efficient initialization algorithm, which has presented in [14]. The algorithm repeats partitioning the stations into two subsets by flipping a fair coin as follows. First, every station flip a fair coin and they are partitioned into two subsets. Let P_1 and P_2 be the two subsets with heads and tails, respectively. The stations in P_1 broadcast on the channel. Clearly, the status of the channel is NULL if P_1 is empty, SINGLE if P_1 has one station, and COLLISION if P_1 has two or more stations. If P_1 has one station, then an ID 1 is assigned to it. If P_1 has two or more stations, then, by recursion, IDs are given to all the stations in P_1. After that, the same procedure is executed for P_2.

In order to evaluate the running time slots, we will trace the behavior of stations using a binary tree as follows: Initially, all the stations are in the root. Every station moves to one of the left or right children at random by flipping a coin. The station repeats moving to one of the two children until no other stations are in the same node. If a station is alone in some node it receives an ID and terminates moving. It is easy to see that the running time slots corresponds to the number of nodes that have visited by a station. We say that, after termination of the algorithm, a node is

- *single* if one station has visited it,
- *null* if no station has visited it although two or more stations have visited its parent,
- *good* if two or more station have visited it and both of the children are non-empty,
- *bad* if two or more station have visited it and one of the children is empty.

We are going to evaluate the number of these nodes. Let N_s, N_n, N_g, and N_b denote the number of single, null, good, and bad nodes in the binary tree. It is easy to confirm that $N_s = n$, $N_g = n - 1$ and $N_n = N_b$. Further, one broadcast is performed for each of these four types of nodes. Hence, the running time T satisfies

$$\begin{aligned} T &= N_s + N_n + N_g + N_b \\ &= 2N_b + 2n - 1. \end{aligned}$$

Thus, we are going to evaluate N_b to get a good estimation of T. If m stations ($m \geq 2$) are visiting a node, then it will be good with probability at least $1 - 2(\frac{1}{2})^m \geq \frac{1}{2}$. Hence, from Lemma 3.1, with probability at least $1 - e^{-m}$, we have no more than $6n + 10m$ internal nodes (i.e either good or bad nodes). It follows that $N_b + N_g \leq 6n + 10m$ with probability at least $1 - e^{-m}$. Hence, we have $N_b \leq 5n + 10m + 1$. Thus, the initialization algorithm runs in $12n + 20m + 1$ time slots with probability at least $1 - e^{-m}$. Note that every station have to be awake for all the time slots in this algorithm.

Lemma 4.5 *The BCM with n stations can be initialized in $12n + 20m + 1$ time slots with probability at least $1 - e^{-m}$ for every $m \geq 1$.*

Using this algorithm, we show an energy-efficient initialization algorithm on the k-channel BCM ($k \leq \log n$) which will be used in our ranking algorithm to initialize each $Q(i)$. We assume that $Q(i)$ has $16(\log n)^2$ stations. Our goal is to initialize stations in $Q(i)$ in $O(\frac{\log n^2}{k})$ time slots with no station being awake for more than $O(\log n)$ time slots with high probability. The idea is to partition the $16(\log n)^2$ stations into $\log n$ subgroups equally at random. For this purpose, each station in the BCM randomly choose an integer in the range $[1, \log n]$. Let $G(i)$ $(1 \leq i \leq \log n)$ denote the set of stations that have selected i. Clearly, $E[|G(i)|] = 16 \log n$. Using the Chernoff bound (2), the probability that a particular $G(i)$ has more than $32 \log n$ stations is at least

$$\begin{aligned} \Pr[|G(i)| > 32 \log n] &= \Pr[|G(i)| > (1+1)E[|G(i)|]] \\ &< e^{-\frac{1}{3}16 \log n} < n^{-7.6}. \end{aligned}$$

Thus, all the subgroups have at most $32 \log n$ stations with probability at least $1 - n^{-7.6} \log n > 1 - n^{-7}$. Using Lemma 4.5, a subgroup with less than $32 \log n$ stations can be initialized in $12(32 \log n) + 20(2 \log n) + 1 = 424 \log n + 1$ time slots with probabilityat least $1 - e^{-2 \log n} > 1 - n^{-2.8}$. Hence, we use $424 \log n + 1$ time slots to initialize a subgroup. Because k ($\leq \log n$) channels are available, k subgroups can be initialized concurrently. Thus, all of the subgroups can be initialized in $\lceil \frac{\log n}{k} \rceil \cdot (424 \log n + 1) = O(\frac{(\log n)^2}{k})$ time slots. The stations in subgroup $G(i)$ are awake for $424 \log n + 1$ time slots to initialize $G(i)$, and they are sleep during the initialization of the other subgroups. Hence, the stations are awake for $424 \log n + 1 = O(\log n)$ time slots. It follows that, no subgroup fails to initialize with probability at least $1 - n^{-2.8} \log n > 1 - n^{-2}$. Thus, we have,

Lemma 4.6 *With probability at least $1 - n^{-2}$, the task of initializing a set of $16(\log n)^2$ stations on the k-channel ($k \leq \log n$) BCM can be done in $O(\frac{(\log n)^2}{k})$ time slots with no stations being awake for more than $O(\log n)$ time slots.*

5 Implementation of our ranking algorithm

We are now in position to show the implementation of our ranking algorithm on the k-channel BCM. The details are spelled out as follows.

Step 1 Initialize the stations in the BCM. Let $S(i)$ $(1 \leq i \leq n)$ denote the station with ID i, and x_i denote the key stored in $S(i)$.

Step 2 Let $x_1, x_2, \ldots x_{\frac{n}{\log n}}$ be the sample keys and sort them such that $S(i)$ $(1 \leq i \leq \frac{n}{\log n})$ is storing p_i, the i-th smallest sample keys. After that, each $S(j \log n)$ $(0 \leq j \leq \log n - 1)$ is storing a pivot q_j.

Step 3 Each $S(i)$ $(1 \leq i \leq n)$ finds an index a_i such that x_i is in $Q(a_i)$.

Step 4 Initialize the stations within each $Q(j)$. At the end of Step 4, the stations in each $Q(j)$ has IDs in the range $[1, |Q(j)|]$.

Step 5 Sort keys within each $Q(j)$ using the local ID obtained in Step 4 to compute the local rank of the keys. At the end of Step 5, every station $S(i)$ knows the local rank $l(i)$.

Step 6 Compute the prefix-sums of $|Q(0)|$, $|Q(1)|$, ..., $|Q(\frac{n}{(\log n)^2})|$. At the end of Step 5, every station $S(i)$ knows the global rank $g(i)$.

Step 7 Each $S(i)$ $(1 \leq i \leq n)$ computes $l(i) + g(i)$ which is equal to $r(i)$.

We are going to show that each step can be implemented to run for at most $O(\frac{n}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$ time slots with probability at least $1 - O(n^{-1})$.

Step 1 can be done in $O(\frac{n}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$ time slots with probability at least $1 - O(n^{-1.83})$ using the initialization algorithm in [3].

By Corollary 4.2, the $\frac{n}{\log n}$ sample keys can be sorted in $O(\frac{\frac{n}{\log n} \cdot \log \frac{n}{\log n}}{k} + \log \frac{n}{\log n})$ time slots with no station being awake for more than $O(\log \frac{n}{\log n})$ time slots with probability at least $1 - O((\frac{n}{\log n})^{-6})$. Thus, Step 2 can be done in $O(\frac{n}{k} + \log n)$ time slots with no station begin awake for more than $O(\log n)$ time slots with probability at least $1 - O(n^{-5})$. Note that the initializing algorithm performed in Step 1 gives unique IDs to the stations uniformly at random. Thus, the $\frac{n}{\log n}$ sample keys stored in $S(1), S(2), \ldots, S(\frac{n}{\log n})$ are randomly chosen from the n keys.

Step 3 can be done by traversing a binary search tree of pivots defined as follows. Imagine a binary search tree of nodes $q_0, q_1, \ldots, q_{\frac{n}{(\log n)^2}-1}$ such that the median $q(\frac{n}{2(\log n)^2} - 1)$ is the root. The left subtree is the binary search tree of $q_0, q_1, \ldots, q_{\frac{n}{2(\log n)^2}-2}$ and the right is that of $q_{\frac{n}{2(\log n)^2}}, \ldots, q_{\frac{n}{(\log n)^2}-1}$. Each station traverses the binary tree from the root to a leaf. Station $S(i)$ moves to the left child if x_i is smaller than the current pivot, and goes to the right child, otherwise. At the leaves each station $S(i)$ can determine the value of a_i. This traverse can be implemented as follows: Station $S(\frac{n}{2(\log n)^2})$ broadcast $q(\frac{n}{2(\log n)^2})$ to the fist channels and every station received it. Next, the left and the right children are broadcast on the first and second channels, respectively. Every station receives one of them. This broadcast is repeated until all of the $\frac{n}{(\log n)^2}$ pivots are broadcast on channels. Let h be the binary tree. Clearly, $h = \log \frac{n}{2(\log n)^2} - 1 = O(\log n)$. The level i $(1 \leq i \leq h)$ of the binary search tree has 2^{i-1} pivots. Using k channels, these 2^{i-1} pivots can be broadcast in $\lceil \frac{2^{i-1}}{k} \rceil$ time slots. Hence all of the pivots can be broadcast in $\sum_{i=1}^{h} \lceil \frac{2^{i-1}}{k} \rceil = O(\frac{n}{k(\log n)^2} + h) \leq O(\frac{n}{k} + \log n)$ time slots. Further, all stations broadcast a pivot at most once and receive pivots for h times. Thus, no station is awake for more than $O(\log n)$ time slots.

Before showing Step 4, we will evaluate the number of keys in each $Q(i)$ using Lemma 3.2. Recall that the initialization in Step 1 gives the n stations unique IDs in the range $[1, n]$ uniformly at random. This is analogous to throwing the n balls into the n bins. In other words, the n keys are "thrown" into the n stations at random. The stations with IDs in $[1, \frac{n}{\log n}]$ correspond to red bins. Suppose that the n keys are partitioned into $\frac{n}{8(\log n)^2}$ groups $B(1), B(2), \ldots, B(\frac{n}{8(\log n)^2})$ of $8(\log n)^2$ keys each such that the keys in $B(i+1)$ is larger than those in $B(i)$ for every i $(1 \leq i \leq \frac{n}{8(\log n)^2})$. That is, $B(i)$ is a set of keys with ranks in $[8(\log n)^2(i-1)+1, 8(\log n)^2 i]$. From Lemma 3.2, with probability at least $1-n^{-2}$, every $B(i)$ contains at least $\log n$ samples. If this is the case, then every $B(i)$ includes at least one pivot. It follows that, with probability at least $1-n^{-2}$, every $Q(i)$ has no more than $16(\log n)^2$ keys Thus, we have the following lemma:

Lemma 5.1 *The probability that $|Q(i)| \leq 16(\log n)^2$ for all i $(0 \leq i \leq \frac{n}{(\log n)^2})$ is at least $1 - n^{-2}$.*

We use the algorithm for Lemma 4.6 to initialize the stations storing $Q(i)$ for each i $(0 \leq i \leq \frac{n}{(\log n)^2})$. If $|Q(i)| \leq 16(\log n)^2$, then using k channels $(k \leq \log n)$, stations storing keys in $Q(i)$ can be initialized in $O(\frac{(\log n)^2}{k})$ time slots with no station being awake for more than $O(\log n)$ time slots with high probability. Suppose that we assign k $(\leq \log n)$ channels to each group use the algorithm in Lemma 4.6 to initialize each group. Since we have $\frac{n}{(\log n)^2}$ groups, all of the groups succeed in being initialized in $O(\frac{(\log n)^2}{k})$ time slots $(k \leq \log n)$ with no station being awake for more than $O(\log n)$ time slots with probability at least $1 - n^{-2} \cdot (\frac{n}{(\log n)^2} + 1) > 1 - n^{-1}$. We assume that all of the groups succeed in being initialized and evaluate the time slots and awake time slots as follows.

Case 1: $\frac{n}{(\log n)^2} + 1 \leq k \leq \frac{n}{\log n}$. We assign $k' = \frac{k}{\frac{n}{(\log n)^2}+1}$ channels to initialize a group. All of the groups can be initialized in $O(\frac{(\log n)^2}{k'}) = O(\frac{n}{k})$ time slots with no station being awake for more than $O(\log n)$ time slots.

Case 2: $k \leq \frac{n}{(\log n)^2}$. We assign $\frac{\frac{n}{(\log n)^2}+1}{k}$ groups to a channel to initialize them. Each channel is used to initialize these groups by the algorithm for Lemma 4.6, in turn, using one channel. Each group is assigned $O((\log n)^2)$ time slots and the stations having the keys in the group execute the initialization algorithm using one channel assigned. The stations in a group are asleep while the initialization for the other groups is executed. Thus, every station is awake for $O(\log n)$ time slots. Further, all the groups can be initialized in $\frac{n}{k(\log n)^2} \cdot O((\log n)^2) = O(\frac{n}{k})$ time slots.

Thus, in Step 4, all the groups can be initialized in $O(\frac{n}{k})$ time slots with no station being awake for more than $O(\log n)$ time slots with probability $1 - n^{-1}$.

Since every station in $Q(i)$ has unique ID in the range $[1, |Q(i)|]$ we can use a deterministic sorting to compute the local ranks. Using the algorithm for Corollary 4.3 with $\epsilon = \frac{1}{2}$, we can sort $16(\log n)^2$ keys in $O(\frac{(\log n)^2}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$ time slots. In Step 5, we sort the groups concurrently using the k channels similarly to the initialization in Step 4. If $k > \frac{n}{(\log n)^2}$, then all the groups can be sorted in $O(\frac{(\log n)^2}{k'} + \log n) = O(\frac{n}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$ time slots. If $k \leq \frac{n}{(\log n)^2}$, every group can be sorted in $\frac{n}{k(\log n)^2} \cdot O((\log n)^2) = O(\frac{n}{k})$ time slots with no station being awake for more than $O(\log n)$ time slots. Thus, Step 5 can be done in $O(\frac{n}{k} + \log n)$ time slots with no station begin awake for $O(\log n)$ time slots.

Recall that Step 4 initializes the stations in each $Q(i)$. Thus, at least one station knows the value of $|Q(i)|$. In Step 6, we first move the value of each $|Q(i)|$ to station $S(i)$ and then compute the the prefix-sums of $\frac{n}{(\log n)^2}$ integers $|Q(0)|, |Q(1)|, \ldots, |Q(\frac{n}{(\log n)^2} - 1)|$ using $S(0), S(1), \ldots, S(\frac{n}{(\log n)^2} - 1)$. From Corollary 4.4, the prefix-sums can be computed in $O(\frac{n}{k(\log n)^2} + \log n) < O(\frac{n}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$ time slots.

Step 7 can be done by broadcasting the prefix-sums in turn using k channels and by local computation. It takes $O(\frac{n}{k(\log n)^2}) < O(\frac{n}{k})$ time slots with each station being awake for $O(1)$ time slots.

Since every step can be done in $O(\frac{n}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$ time slots with probability at least $1 - O(n^{-1})$, finally, we have

Theorem 5.2 *The task of ranking n keys in the k-channel BCM with n stations can be done in $O(\frac{n \log n}{k} + \log n)$ time slots with no station being awake for more than $O(\log n)$ time slots with probability at least $1 - O(n^{-1})$.*

6 Lower bounds

This section is devoted to discuss the lower bounds for the sorting problem on the initialized BCM. Since the ranking problem can be reduced to the sorting problem easily, the same lower bounds hold for the ranking problem on the uninitialized BCM.

From the bandwidth argument, it is clear that any sorting algorithm takes at least $\Omega(\frac{n}{k})$ time, because every key must be broadcast at least once. We show the limit of acceleration under the following two assumptions in terms of the size of storage of stations: *limited storage*: Each station can store a constant number of keys in its local memory. *unlimited storage*: Each station can store unlimited number of keys in its local memory.

It is well known that any deterministic sorting algorithm needs $\Omega(n \log n)$ comparisons in the worst case as well as the average case [4, pp.172]. We briefly review this argument and related results. Let us consider the sorting problem on input $X = \{x_1, x_2, \ldots, x_n\}$. The sorted sequence of X is one of the possible $n!$ permutations of X. Any deterministic sorting algorithm on X can be represented by a *binary decision tree* such that the leaves are all possible $n!$ permutations and each internal node is associated with comparison of two keys in X. The sorting algorithm traverses the binary decision tree from the root to a leaf. At each internal node, the sorting algorithm moves to one of the two children by the result of the comparison. Clearly, the number of comparisons by the sorting algorithm on a particular input is the number of internal nodes on the path. Thus, the number of comparisons performed in the worst case is the number of internal nodes on the longest path. Since the binary decision tree has $n!$ leaves, there exists a path of length at least $\log n! = \Omega(n \log n)$. Hence, there exists an input X requiring $\Omega(n \log n)$ comparisons by the sorting algorithm. Next, we analyze the average comparisons over all input. We assume that all permutations have equal probability $\frac{1}{n!}$. It is not difficult to confirm that, on any binary tree with $n!$ leaves, the average length of all the $n!$ paths to the leaves is at least $\Omega(n \log n)$. Thus, if all permutations have equal probability, then any deterministic sorting algorithm need to perform expected $\Omega(n \log n)$ comparisons.

Using the *Yao's minimax principle* [16], the average analysis of deterministic algorithms gives a lower bound of the average case analysis of randomized algorithms. More precisely, the expected comparisons of the optimal deterministic sorting algorithm for an arbitrary chosen input distribution is a lower bound on the expected comparisons for the worst input performed by the optimal randomized algorithms. The readers should refer[10, pp.34] for the details of the Yao's minimax principle. It follows that any randomized sorting algorithm has the worst input on which it need to perform expected $\Omega(n \log n)$ comparisons.

On the BCM with limited storage, a station is awake and receives a new key, the station can compare the new key and keys in its local memory. Since it is storing a constant number of keys, it performs only a constant number of comparisons with the new key. In other words, a single awake time slot by a station can involve $O(1)$ comparison. It follows that $\Omega(n \log n)$ total awake time slots are necessary to solve the sorting problem. Hence, any randomized sorting algorithm with limited storage has at least one station that must be awake for expected $\Omega(\log n)$ awake time slots.

Next, we are going to show $\Omega(\frac{\log n}{\log\log n})$ lower bond of awake time slots for the unlimited storage BCM. We will evaluate how many meaningful comparisons can be performed after receiving a new key. Suppose that a station has been awake for s time slots and has received s keys. Thus, it is storing $s+1$ keys in its local storage. Let $y_0, y_1, \ldots, y_s$ ($y_0 < y_1 < \cdots < y_s$) denote the $s+1$ keys and z be a new key received by the station. By comparing z with the $s+1$ keys, the station can find i ($-1 \leq i \leq s$) such that $y_i < z < y_{i+1}$ where $y_{-1} = -\infty$ and $y_{s+1} = +\infty$. Similarly to the proof of the $\Omega(n \log n)$ lower bound, we associate this comparison of key z with the s keys with an internal node having $s+2$ children. Then, any deterministic sorting algorithm can be represented by a *decision tree*. Suppose that a deterministic sorting algorithm on the BCM solves the sorting problem with every station being awake for at most T time slots. Note that an awake time slot by a station is associated with an internal node of the decision tree. Clearly, no internal node has more than $T+1$ children. Further, the total awake time slot is nT, every path from the root to a leaf has nT internal nodes. Since the decision tree have $n!$ leaves, we have $(T+1)^{nT} \geq n!$. It follows that at least one station must be awake for $T = \Omega(\frac{\log n}{\log\log n})$ time slots. By applying the Yao's minimax principle to this lower bound, we have that any randomized sorting algorithm on the BCM with unlimited storage has at least one station that must be awake for expected $\Omega(\frac{\log n}{\log\log n})$ awake time slots. Further, since the running time slot cannot be smaller than the awake time slot any randomized algorithm must be run for expected $\Omega(\frac{n}{k} + \frac{\log n}{\log\log n})$ time slots. Finally, we have,

Theorem 6.1 *Any randomized algorithm that solves the sorting (or ranking) problem of n keys on the BCM with n stations must be run for*

- *expected $\Omega(\frac{n}{k} + \log n)$ time slots with at least one station being awake for expected $\Omega(\log n)$ time slots if storage is limited, and*
- *expected $\Omega(\frac{n}{k} + \frac{\log n}{\log\log n})$ time slots with at least one station being awake for expected $\Omega(\frac{\log n}{\log\log n})$ time slots if storage is unlimited.*

Note that our ranking algorithm is designed for the BCM with limited storage. Thus, our ranking algorithm is optimal under the limited storage assumption.

References

[1] M. Ajtai, J. Komlos, and E. Szemeredi, Sorting in $c \log n$ parallel steps, *Combinatorica*, 3, 1, pp.1–19, 1983.

[2] M. Adler, J. W. Byers, R. M. Karp, Parallel Sorting with Limited Bandwidth, *SIAM Journal on Computing* 29, 6, pp. 1997-2015, 2000

[3] J. L. Bordim, J. Cui, T. Hayashi, K. Nakano, and S. Olariu, Energy-efficient initialization protocols for adhoc radio network, *IEICE Trans. on Fundamentals*, E83-A, 9, pp.1796-1803, 2000.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to algorithms*, MIT Press, 1994.

[5] R. Cole, Parallel merge sort, *SIAM J. Computing*, 17, 4, 770-785, 1988.

[6] A. Gibbons and W. Rytter, *Efficient parallel algorithms*, Cambridte University Press, 1988.

[7] J. JáJá, *An introduction to parallel algorithms*, Addison-Wesley, 1992.

[8] D. Krizanc, A survey of randomness and parallelizm in comparison problems, *Advances in Randomized Parallel Computing*, Kluwer Academic Publishers, 25–39, 1999.

[9] F. T. Leighton, Tight bounds on the complexity of parallel sorting, *IEEE Trans. Computers*, 34, 4, 344-354, 1985.

[10] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

[11] K. Nakano, Optimal sorting algorithms on bus-connected processor arrays, *IEICE Trans. Fundamentals*, E-76A, 11, pp.2008–2015, 1994.

[12] K. Nakano, Optimal initializing algorithms for a reconfigurable mesh, *J. Parallel and Distributed Computing*, 24, pp.218–223, 1995.

[13] K. Nakano, S. Olariu, and J. L. Schwing, Broadcast-efficient protocols for mobile radio network, *IEEE Trans. parallel and Distributed Systems*, 10, 12, 1276-1289, 1999.

[14] K. Nakano and S. Olariu, Randomized initialization protocols for ad-hoc networks, *IEEE Trans. on Parallel and Distributed Systems*, 11, 7, pp. 749–759, 2000.

[15] K. Nakano, and S. Olariu, Energy-efficient initialization protocols for single-hop radio networks with no collision detection, *IEEE Trans. on Parallel and Distributed Systems*, 11, 8, 851–863, 2000.

[16] A. C. Yao, Probablistic Computations: Towards a unified measure of complexity, *Proc. of 17th Symposium on Foundations of Computer Science*, pp. 222-227, 1977.