International Journal of Foundations of Computer Science
© World Scientific Publishing Company

# Time and Energy Optimal List Ranking Algorithms on the $k$-Channel Broadcast Communication Model with no Collision Detection [*]

KOJI NAKANO
*School of Information Science, Japan Advanced Institute of Science and Technology*
*Tatsunokuchi, Ishikawa 923-1292, Japan*

Received (received date)
Revised (revised date)
Communicated by Editor's name

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 a data packet to one of $k$ distinct communication channels. We assume that the stations run on batteries and expand 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. The main contribution of this paper is to present time and energy optimal list ranking algorithms on the BCM. We first show that the rank of every node in an $n$-node linked list can be determined in $O(n)$ time slots with no station being awake for more than $O(1)$ time slots on the single-channel $n$-station BCM with no collision detection. We then extend this algorithm to run on the $k$-channel BCM. For any small fixed $\epsilon > 0$, our list ranking algorithm runs in $O(\frac{n}{k})$ time slots with no station being awake for more than $O(1)$ time slots, provided that $k \leq n^{1-\epsilon}$. Clearly, $\Omega(\frac{n}{k})$ time is necessary to solve the list ranking problem for an $n$-node linked list on the $k$-channel BCM. Therefore, our list ranking algorithm on the $k$-channel BCM is time and energy optimal.

## 1. Introduction

A *Broadcast Communication Model* (BCM, for short) is a distributed system with no central arbiter populated by $n$ processing units referred to as *stations* $S(1), S(2), \ldots, S(n)$. The fundamental characteristic of the model is the broadcast nature of communications. A data packet broadcast on a channel can be received by every station that has tuned to the channel. The nature of end units is immaterial: they can well be processors in a parallel computing environment or radio transceivers in a wireless network. Likewise, the nature of the transmission channel 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 com-

[*] Work supported in part by the Telecommunications Advancement Foundations.

puting environment, local area networks, and single-hop radio networks. Although the BCM is assumed to operate in synchronous mode, we do not prescribe a particular synchronization mechanism. We feel that this is best left to the particular application. For example, in radio networks, synchronization may be provided by an interface to a commercially-available Global Positioning System [17].

We employ the commonly-accepted assumption that when two or more stations are broadcasting on a channel in the same time slot, the corresponding packets *collide* and are lost. The BCMs has two kinds of assumptions in terms of *collision detection* (CD) capability [20, 21, 22, 29]. In the BCM with CD, the status of a radio channel is:

**NULL:** if no station broadcast on the channel in the current time slot,

**SINGLE:** if exactly one station broadcast on the channel in the current time slot,

**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. In the BCM with no collision detection the status of a radio channel is:

**NOISE:** if either no station broadcasts or two or more stations broadcast on the channel in the current time slot, and

**SINGLE:** if exactly one station broadcasts on the channel in the current time slot.

In other words, the BCM with no CD cannot distinguish between no broadcast on the channel and the result of two or more stations broadcasting on the channel. Several workers have argued that from a practical standpoint the no CD assumption makes a lot of sense since in many situations, especially in the presence of noisy channels, the stations cannot distinguish between the no transmission case and the collision of several packets that arises when several stations attempt to broadcast at once [4, 5, 24]. On the other hand, many other radio or cellular networks including AMPS, GSM, ALOHA-net, as well as the well-known Ethernet are systems where collision detection is possible [1, 2, 6, 7, 19]. Our algorithms presented in this paper run on the BCM with no CD.

We assume that broadcasting/receiving data packets in the BCM is very costly. For example, if the stations run on batteries and, therefore, 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 its transceiver is active, that is, while transmitting or receiving a packet [12, 14, 23, 25, 27, 28]. It is perhaps surprising at first that a station expends power when receiving a packet. Consequently, we are interested in developing algorithms that allow stations to power their transceiver off (i.e. go to sleep) to the largest extent possible. Accordingly, we judge the goodness of an algorithm by the following two yardsticks:

*running time slots*: the overall number of time slots required by the algorithm to terminate

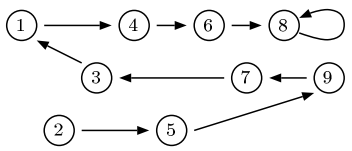

Figure 1: An example of a linked list

*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.

The challenge is to strike a sensible balance between the two, by designing algorithms that take a small number of time slots to terminate while being, at the same time, as energy-efficient as possible.

A linked list is a basic data structure frequently used in many processing tasks. *A linked list* $L$ of $n$ nodes is specified by an array $p$ such that $p(i)$ contains a pointer to the next node of node $i$ in the list of $L$. Figure 1 illustrates an example of a linked list. A node $i$ is *the end* of the list if $p(i) = i$. Further, if there exists no node $j$ such that $p(i) = j$, then node $i$ is *the top* of the list. Nodes 2 and 8 in Figure 1 are the top and the end nodes of the list, respectively. The list ranking problem asks to determine *the rank* of every node $i$ $(1 \leq i \leq n)$, which is the distance to the end of the list. The list ranking has many applications. For example, a number of algorithms such as computing a preorder/postorder numbering of nodes in a tree and finding lowest common ancestor of nodes use the list ranking as a key ingredient [16, 15].

The list ranking problem has been solved in several contexts [10, 11, 15, 26]. It is well known that the list ranking problem can be solved in $O(\log n)$ time using $n$ processors on the PRAM [13, 16]. This parallel list ranking algorithm uses *the pointer jumping technique* [3, 16], which repeatedly changes each pointer such that a new pointer is the successor of the successor. The pointer jumping is repeated until all pointer points at the end node of the list. Further, it is known that the number of processors can be reduced to $\frac{n}{\log n}$ without increasing the computing time [11, 16].

Suppose that a pointer $p(i)$ is stored in station $S(i)$ on the BCM. If energy-efficiency is not an issue, the list ranking problem can be solved on the BCM easily by traversing the list from the end of the list toward the top as follows: First, a unique station $S(i)$ satisfying $p(i) = i$ broadcasts $i$ on the channel. Every station receives $i$. Clearly, if $p(j) = i$, then the rank of node $j$ is 1. Next, station $S(j)$ broadcasts $j$ on the channel, and every station receives it. Again, if $p(j') = j$, then the rank of node $j'$ is 2. Continuing similarly, the rank of every node can be determined in $n - 1$ time slots. However, this algorithm is not energy efficient. Station $S(i)$ storing pinter $p(i)$ of the top node $i$ must be awake for $n - 1$ time slots.

One of the straightforward strategies to design an energy-efficient algorithm on the BCM is to simulate known PRAM algorithms. It is known that the list ranking problem can be solved in $O(\log n)$ time using $n$ processors on the PRAM [13, 16]. We are going to show that an energy-efficient list raking algorithm on the BCM can be obtained by simulating a known PRAM list ranking algorithm as follows: first, it should be clear that any single step communication performed on the $n$-processor $O(n)$-memory-cell PRAM can be simulated by the $n$-station $n$-channel BCM in $O(1)$ time slots. This can be done by assigning $O(1)$ memory cells to each station. Then, read/write operations on the PRAM can be simulated using communication channels on the BCM in obvious way. Hence, any algorithm running in $O(\log n)$ time using $n$ processors and $O(n)$ memory cells on the PRAM can be simulated by the $n$-station $n$-channel BCM in $O(\log n)$ time slots. Clearly, no station is awake for more than $O(\log n)$ time slots in this simulation. Further, communication using $n$ channels in a single time slot can be simulated on the $k$-channel BCM ($k \leq n$) in $O(\frac{n}{k})$ time slots. Thus, the list ranking problem can be solved in $O(\frac{n \log n}{k})$ time slots with each station being awake for at most $O(\log n)$ time slots. However this algorithm is not time and energy optimal.

The main contribution of this paper is to present time and energy optimal list ranking algorithms on the BCM with no CD. Surprisingly, stations are awake for only $O(1)$ time slots in our list ranking algorithms. We first show that the rank of every node in an $n$-node linked list can be done in $O(n)$ time slots with no station being awake for more than $O(1)$ time slots on the single-channel $n$-station BCM with no CD. We then extend this algorithm to run on the $k$-channel BCM. For every small fixed $\epsilon > 0$, our algorithm runs in $O(\frac{n}{k})$ time slots with no station being awake for more than $O(1)$ time slots, provided that $k \leq n^{1-\epsilon}$. Clearly, every $p(i)$ must be broadcast at least once. Hence, $\Omega(\frac{n}{k})$ time is necessary to solve the list ranking problem for an $n$-node linked list on the $k$-channel BCM. Therefore, our algorithm is time and energy optimal.

This paper is organized as follows: Section 2 shows basic techniques used for our energy-efficient list ranking and shows a list ranking algorithm on the single-channel BCM with no CD. This algorithm runs in $O(n \log n)$ time slots with at most $\frac{n}{2^\alpha}$ stations ($1 \leq \alpha \leq \log n$) being awake for $O(\alpha)$ time slots. In Section 3, we modify this list ranking algorithm to run in $O(n)$ time slots with each station being awake for $O(1)$ time slots on the single-channel BCM with no CD. In Section 4, we extend this list ranking algorithm to run on the $k$-channel BCM. Section 5 offers concluding remarks and open problems.

## 2. List ranking using list shrink

The main purpose of this section is to show fundamental techniques used in our time and energy optimal list ranking algorithm. We first assume that the BCM has the collision detection capability. Later, we show how to avoid using the CD capbablity.

We use a fundamental technique for solving the list ranking problem as follows: This technique uses two arrays of variables $q[i]$ and $r[i]$ for every $i$ ($1 \leq i \leq n$). Initially, $q[i]$ is storing pointer $p(i)$ for every $i$, and $r[i] = 0$ if node $i$ is the end of the list, and $r[i] = 1$ otherwise. During the execution of list ranking algorithms, $r[i]$ is always storing the distance from node $i$ to $q[i]$. When the list ranking algorithm terminates, for every $i$, $q[i]$ is storing the pointer to the end of the list. Thus, each

$r[i]$ is storing the rank of node $i$. This technique is used in the pointer jumping [16], which repeats operations $q[i] \leftarrow q[q[i]]$ and $r[i] \leftarrow r[i]+r[q[i]]$ for every $i$ $(1 \leq i \leq n)$ in parallel. After $\log n$ iterations, every $q[i]$ is storing the pointer to the end of the list, thus, $r[i]$ is storing the rank of node $i$. The correctness of the pointer jumping can be easily seen as follows. Suppose that every $r[i]$ $(1 \leq i \leq n)$ is storing the distance from node $i$ to node $q[i]$. Then, the distance from node $i$ to node $q[q[i]]$ is the sum of $r[i]$ and $r[q[i]]$. Thus, after executing $q[i] \leftarrow q[q[i]]$ and $r[i] \leftarrow r[i]+r[q[i]]$ in parallel, $r[i]$ is storing the distance from node $i$ to $q[i]$ in the initial list. After $\log n$ iterations of these operations, every $q[i]$ is storing the index of the end node and $r[i]$ is the rank of node $i$.

Our list ranking algorithm uses two arrays $q$ and $r$. For a current linked list stored in array $q$, *a left sublist* is a sequence $\langle i_1, i_2, \ldots, i_m \rangle$ of nodes such that $i_1 > i_2 > \cdots > i_m$ and $i_{j+1} = q[i_j]$ for every $j$ $(1 \leq j \leq m-1)$. A left sublist is *a maximal left sublist* if no other left sublist contains it. We say that nodes $i_1$ and $i_m$ are *the head* and *the tail* of the maximal left sublist $\langle i_1, i_2, \ldots, i_m \rangle$. Similarly, we can define *a right sublist*, *a maximal right sublist*, and their *head* and *tail nodes*. In Figure 1, $\langle 9,7,3,1 \rangle$ is a maximal left sublist, and both $\langle 1,4,6,8 \rangle$ and $\langle 2,5,9 \rangle$ are maximal right sublists. Further, node 1 is the tail node of $\langle 9,7,3,1 \rangle$ as well as the head of $\langle 1,4,6,8 \rangle$.

Our list ranking algorithm repeats shrinking maximal left and right sublists. Further, every leaf node that has no predecessor is eliminated. More precisely, our algorithm repeats `list-shrink` described as follows:

```
list-shrink
Step 1: shrink left sublists by procedure left-shrink.
Step 2: eliminate leaf nodes by procedure leaf-elimination.
Step 3: shrink right sublists by procedure right-shrink.
Step 4: eliminate leaf nodes by procedure leaf-elimination.
```

Figure 2 illustrates each step of `list-shrink` executed for the linked list in Figure 1. Somewhat surprisingly, `list-shrink` can be done in $O(n)$ time slots with no station being awake for more than $O(1)$ time slots. Further, `list-shrink` eliminates at least half of the nodes.

We will show the details of each step of `list-shrink`. Let $q$ and $r$ be the arrays storing pointers of a linked list and the distance as explained above. Procedure `left-shrink` is described as follows.

```
left-shrink
for $i \leftarrow 1$ to $n$ do
  $S(i)$ broadcasts $q[i]$ and $r[i]$ on the channel.
  $S(j)$ satisfying $q[j] = i < j$ receives them and sets $q[j] \leftarrow q[i]$ and
    $r[j] \leftarrow r[j] + r[i]$.
```

Clearly, `left-shrink` takes $n$ time slots. Further, each $S(i)$ is awake at time slots $i$ and $q[i]$ and is asleep for remaining time slots. Thus, every station is awake for at most two time slots. Suppose that `left-shrink` is executed for the linked list in

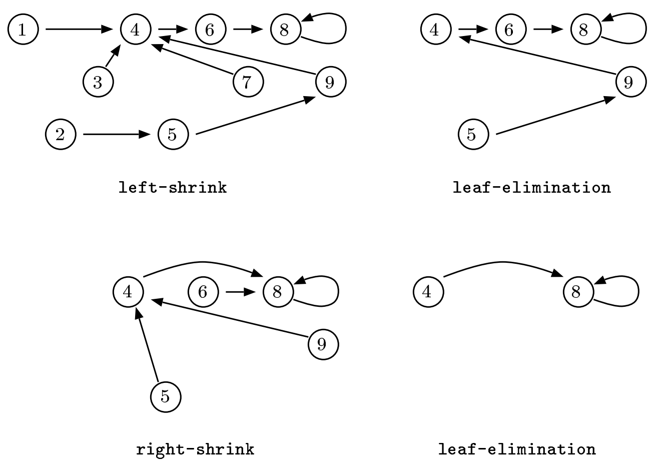

Figure 2: Each step of `list-shrink`

Figure 1. Note that $r[8] = 0$, and $r[i] = 1$ for all $i$ ($i \neq 8$). In time slot $i = 1$, station $S(1)$ broadcasts $q[1](= 4)$ and $r[1](= 1)$ on the channel. Station $S(3)$ receives them because $q[3] = 1 < 3$. It sets $q[3] \leftarrow q[1](= 4)$ and $r[3] \leftarrow r[3]+r[1](= 2)$. No station receives in time slot $i = 2$. In time slot $i = 3$, station $S(3)$ broadcasts $q[3](= 4)$ and $r[3](= 2)$ on the channel. Station $S(7)$ receives them and sets $q[7] \leftarrow q[3](= 4)$ and $r[7] \leftarrow r[7] + r[3](= 3)$. No station receives in time slots $i = 4, 5$, and 6. In time slot $i = 7$, station $S(7)$ broadcasts $q[7](= 4)$ and $r[7](= 3)$ on the channel. Station $S(7)$ receives them and sets $q[9] \leftarrow q[7](= 4)$ and $r[9] \leftarrow r[9] + r[7](= 4)$. Finally, no station receives in time slots $i = 8$ and 9. It is easy to see that, after executing `left-shrink`, $q[i_1] = q[i_2] = \cdots = q[i_m]$ holds for each left sublist $\langle i_1, i_2, \ldots, i_m \rangle$, Table 1 illustrates for the values of of $q$ and $r$ after each step of `list-shrink`. In the table, the values of $q$ and $r$ are with underlines when the corresponding nodes are eliminated. Further, they are blank if the corresponding station is asleep.

After `left-shrink`, the graph may have several leaves, which is a node having no predecessor. Clearly, we obtain a new shrunk list by removing the leaves. The following procedure `leaf-elimination` finds all leaves.

```
leaf-elimination
for $i \leftarrow 1$ to $n$ do
  $S(j)$ broadcasts $j$ if $j = q[i]$.
  $S(i)$ monitors the channel.
  If the status of the channel is NULL then node $i$ is a leaf.
```

Note that `leaf-elimination` uses the collision detection capability. Later, we will modify `leaf-elimination` to run on the BCM with no CD. In `leaf-elimination`, station $S(i)$ is awake at time slots $i$ and $q[i]$. Hence, no station is awake for more than two time slots.

Table 1: The values of local variables for the list in Figure 1

| node | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $q$ | $r$ | $q$ | $r$ | $q$ | $r$ | $q$ | $r$ | $q$ | $r$ | $q$ | $r$ | $q$ | $r$ | $q$ | $r$ | $q$ | $r$ |
| initial input | 4 | 1 | 5 | 1 | 1 | 1 | 6 | 1 | 9 | 1 | 8 | 1 | 3 | 1 | 8 | 0 | 7 | 1 |
| `left-shrink` | 4 | 1 | 5 | 1 | 4 | 2 | 6 | 1 | 9 | 1 | 8 | 1 | 4 | 3 | 8 | 0 | 4 | 4 |
| `leaf-elimination` | $\underline{4}$ | $\underline{1}$ | $\underline{5}$ | $\underline{1}$ | $\underline{4}$ | $\underline{2}$ | 6 | 1 | 9 | 1 | 8 | 1 | $\underline{4}$ | $\underline{3}$ | 8 | 0 | 4 | 4 |
| `right-shrink` | | | | | | | 8 | 2 | 4 | 5 | 8 | 1 | | | 8 | 0 | 4 | 4 |
| `leaf-elimination` | | | | | | | 8 | 2 | $\underline{4}$ | $\underline{5}$ | $\underline{8}$ | $\underline{1}$ | | | 8 | 0 | $\underline{4}$ | $\underline{4}$ |
| `left-shrink` | | | | | | | 8 | 2 | | | | | | | 8 | 0 | | |
| `leaf-elimination` | | | | | | | $\underline{8}$ | $\underline{2}$ | | | | | | | 8 | 0 | | |
| `right-shrink` | | | | | | | | | | | | | | | 8 | 0 | | |
| `leaf-elimination` | | | | | | | | | | | | | | | 8 | 0 | | |
| `rewind` $(T=4)$ | | | | | | | | | | | | | | | 8 | 0 | | |
| `rewind` $(T=3)$ | | | | | | | $\underline{8}$ | $\underline{2}$ | | | | | | | 8 | 0 | | |
| `rewind` $(T=2)$ | | | | | | | 8 | 2 | $\underline{8}$ | $\underline{7}$ | $\underline{8}$ | $\underline{1}$ | | | 8 | 0 | $\underline{8}$ | $\underline{6}$ |
| `rewind` $(T=1)$ | $\underline{8}$ | $\underline{3}$ | $\underline{8}$ | $\underline{8}$ | $\underline{8}$ | $\underline{4}$ | 8 | 2 | 8 | 7 | 8 | 1 | $\underline{8}$ | $\underline{5}$ | 8 | 0 | 8 | 6 |

Procedure `right-shrink` performs the same operation in the opposite order as `left-shrink`. Since each step of `list-shrink` can be done in $n$ time slots with every station being awake for two time slots, we have,

**Lemma 1** *Procedure* `list-shrink` *takes* $4n$ *time slots with no station being awake for more than* $8$ *time slots.*

We are going to prove that no more than $\frac{n}{2}$ nodes are in the list at the end of `list-shrink`. Suppose that a list has $s$ maximal right sublists $L_1, L_2, \ldots, L_s$ in this order. For example, in Figure 1, $s = 2$ and $L_1 = \langle 1, 4, 6, 8 \rangle$ and $L_2 = \langle 2, 5, 9 \rangle$. For simplicity, we assume that $L_1$ and $L_s$ contain the top and the end nodes of the whole list, respectively. We can show the proof similarly when this is not the case. From the definition, each maximal right sublist $L_i$ has at least two nodes. Further, no node is contained in two or more maximal right sublists. Hence, we have $2s \leq n$. At the end of Step 2, every node that is not in maximal right sublists are removed. For example, in Figure 2, nodes 3 and 7 are removed. Further, the head node (nodes 1 and 2 in Figure 2) of each maximal right sublist is removed. Clearly, the list obtained after Step 2 has at most $s$ maximal right sublists. Note that the lists may have less than $s$ maximal right sublists, because two or more adjacent maximal sublists may be merged into one. Since every left sublist has two nodes, every reamining nodes is in one of the maximal right sublists at the end of Step 2. Let $L'_1, L'_2, \ldots, L'_{s'}$ $(s' \leq s)$ denote the maximal sublists obtained at the end of Step 2. We can evaluate the number of nodes in the list obtained after Step 4 as follows. At the end of Step 4, no more than two nodes in $L'_s$ remain. The two nodes are the end of the whole list and the head of $L'_s$. For example, in Figure 2, nodes 8 and 4 remain. Further, no node in $L'_i$ $(2 \leq i \leq s')$ but its tail remain. All nodes in $L'_1$ are eliminated in Step 4. Thus, at the end of Step 4, the list has at most $s'$ nodes. Since $s' \leq s \leq \frac{n}{2}$, we have the following lemma.

**Lemma 2** *After executing* `list-shrink` *on a list of* $n$ *nodes, the resulting list has no more than* $\frac{n}{2}$ *nodes.*

Lemma 2 implies that all nodes but the end of the list are eliminated by repeating `list-shrink` for $\log n$ times. After that, the rank of every node can be computed by rewinding the $2\log n$ iterations of `leaf-elimination` as follows. For each node $i$, let $t(i)$ denote an integer such that node $i$ has been eliminated in the $t(i)$-th $(1 \leq i \leq 2\log n)$ `leaf-elimination`. For convenience, let $t(j) = 2\log n$ for the end node $j$ of the whole list. Suppose that node $q[i]$ is the end of the list and $r[i]$ is storing the rank of node $i$. Then, for every node $j$ satisfying $q[j] = i$, its rank is the sum of $r[j]$ and $r[i]$. Using this fact, the rank of node $i$ is stored in $r[i]$ by the following procedure.

```
rewind
for $T \leftarrow 2\log n$ downto 1 do
  for $i \leftarrow 1$ to $n$ do
    if $t(i) \leq T$ then $S(i)$ broadcasts $q[i]$ and $r[i]$
    every $S(j)$ satisfying $t(j) = T$ and $q[j] = i$ receives $q[i]$ and $r[i]$ and
      sets $q[i] \leftarrow q[j]$ and $r[j] \leftarrow r[j] + r[i]$.
```

Table 1 shows the values of $q$ and $r$ during the execution of `rewind`. They have underlines, when the corresponding station is awake and change its $q$ and $r$.

If $q[j] = i$ when `rewind` starts, $t(i) > t(j)$ holds. Hence, each station $S(j)$ always succeeds in receiving $q[i]$ and $q[j]$. It is easy to see that `rewind` runs in $2n\log n$ time slots. When `rewind` terminates, every $q[i]$ is storing the end of the list, and thus $r[i]$ is the rank of node $i$. Further, each station $i$ is awake for at most $t(i)$ time slots while `rewind` is running. Recall that station $i$ is asleep after the $t(i)$-th `leaf-elimination` until it is awake for `rewind`. Thus, station $i$ is awake for $O(t(i))$ time slots for list ranking. From Lemma 2, the number of nodes $i$ satisfying $t(i) \geq 2\alpha$ is at most $\frac{n}{2^\alpha}$ for every $1 \leq \alpha \leq \log n$. Therefore, we have,

**Lemma 3** *The list ranking problem can be solved in $O(n\log n)$ time slots with at most $\frac{n}{2^\alpha}$ $(1 \leq \alpha \leq \log n)$ station being awake for $O(\alpha)$ time slots on the single-channel BCM.*

Recall that, in the list ranking algorithm for Lemma 3, only `leaf-elimination` uses the collision detection capability. In what follows, we are going to show how we perform `leaf-elimination` without CD.

For each node $i$, let $q^{-1}[i]$ denote its predecessor in the cunnret linked list. A node $i$ is eliminated in Step 2 iff the values of $q[i]$ and $q^{-1}[i]$ before starting Step 1 satisfy one of the following three conditions:

- $q^{-1}[i]$ does not exists (i.e. node $i$ is the top of the whole list),
- $q[i] < i < q^{-1}[i]$ (i.e. node $i$ is in a maximal left sublist but it is neither the head nor the tail), or
- $q[i] > i$ and $i < q^{-1}[i]$ (i.e. node $i$ is the tail of a maximal left sublist).

Hence, once every station $S(i)$ learns the value of $p^{-1}[i]$, it can determine if node $i$ is eliminated in Step 2. It is easy to see that every $S(i)$ can learn $p^{-1}[i]$ in $n$ time slots with each station being awake for two time slots on the single-channel BCM with no CD. Consequently, procedure `leaf-elimination` is performed without the collision detection capability.

## 3. Time and energy optimal list ranking

Recall that, in each `list-shrink`, at most half of the nodes remain in the list. By renumbering the remaining nodes after each `leaf-elimination`, we can reduce the running time slots. In other words, we give a unique number in the range $[1, n']$ to each remaining node, where $n'$ is the number of remaining nodes.

For this purpose, we use an energy optimal prefix-sums algorithm described as follows: Suppose that we have an array $a$ of $n$ numbers. Each $a(i)$ $(1 \leq i \leq n)$ is stored in $S(i)$. *The prefix-sums problem* asks to compute the $i$-th prefix-sum $\mathit{prefix}(i) = a(1) + a(2) + \cdots + a(i)$ for every $i$. The prefix-sums problem can be solved in $n-1$ time slots with every station being awake for at most two time slots. The details of the algorithm are spelled out as follows.

```
prefix-sums
$S(1)$ sets $\mathit{prefix}(1) \leftarrow a(1)$.
for $i \leftarrow 1$ to $n-1$ do
  $S(i)$ broadcasts $\mathit{prefix}(i)$
  $S(i+1)$ receives $\mathit{prefix}(i)$ and sets $\mathit{prefix}(i+1) \leftarrow \mathit{prefix}(i) + a(i)$.
```

It is easy to see that every $S(i)$ learns $\mathit{prefix}(i)$ when `prefix-sums` terminates. Further each station $S(i)$ is awake for at time slots $i$ and $i-1$. Thus, we have

**Lemma 4** *All the prefix-sums can be computed in $n-1$ time slots with no station being awake for more than two time slots.*

Using `prefix-sums`, all of the remaining nodes can be renumbered. Suppose that `list-shrink` is executed for a list of $n$ nodes. Let $a(i) = 1$, if node $i$ is remaining, and $a(i) = 0$ if node $i$ is eliminated. By computing the prefix-sums of $a$, we assign each remaining node $i$ new ID $\mathit{prefix}(i)$. Then, every remaining node has a unique ID in the range $[1, n_1]$, where $n_1$ is the number of remaining nodes.

After assigning new IDs to the remaining nodes, we first arrange them to stations $S(1), S(2), \ldots S(n_1)$ such that each $S(i)$ $(1 \leq i \leq n_1)$ is storing remaining node with new ID $i$. After that, pointers are changed according to the new ID using `node-transfer` as follows:

```
node-transfer
for $i \leftarrow 1$ to $\frac{n}{2}$ do
  if node $j$ is remaining and $\mathit{prefix}(j) = i$ then $S(j)$ broadcasts $q[j]$ and $r[j]$.
  $S(i)$ receives them. Let $q'(i)$ denote the value of $q[j]$.
for $i \leftarrow 1$ to $n$ do
  $S(i)$ broadcasts $\mathit{prefix}(i)$.
  $S(j)$ such that $q'(j) = i$ receives and store it in $q[j]$.
```

After executing `list-shrink` on the list in Figure 1, two nodes 4 and 8 remain. By `prefix-sums`, nodes 4 and 8 receives new IDs $\mathit{prefix}(4) = 1$ and $\mathit{prefix}(8) = 2$, respectively. Thus, after executing `node-transfer`, $S(1)$ and $S(2)$ are storing new nodes 1 and 2, respectively. Further, $q[1]=2, r[1]=2$ and $q[2]=2, r[2]=0$.

From Lemma 2, $n_1 \leq \frac{n}{2}$ holds. Thus, `node-transfer` correctly moves node $i$ to $S(\mathit{prefix}(i))$ and runs in $\frac{3}{2}n$ time slots. If $S(i)$ has remaining node $i$, it is awake at time slots $\mathit{prefix}(i)$ and $\frac{n}{2}+i$. Further, for every $S(i)$ $(1 \leq i \leq n_1)$, it is awake at time slots $i$ and $n+q'(i)$. Thus no station is awake for more than four time slots.

After `node-transfer`, we execute `list-shrink` on the new list with $n_1$ nodes. Suppose that we have $n_2$ nodes after executing the second `list-shrink`. We use `prefix-sums` and `node-transfer` to move the $n_2$ nodes to $n_2$ stations $S(\frac{n}{2}+1), S(\frac{n}{2}+2), \ldots, S(\frac{n}{2}+n_2)$. Continuing similarly, the list ranking problem can be solved. In general, after the $i$-th `list-shrink` $(1 \leq i \leq \log n - 1)$, the $n_i$ remaining nodes are moved to $n_i$ stations $S(n-\frac{n}{2^{i-1}}+1), S(n-\frac{n}{2^{i-1}}+2), \ldots S(n-\frac{n}{2^{i-1}}+n_i)$. Using the $n_i$ stations, `list-shrink` is executed on the new list of $n_i$ nodes. This takes $O(\frac{n}{2^i})$ time slots and stations storing the remaining nodes are awake for $O(1)$ time slots. From Lemma 2, $n_i \leq \frac{n}{2^i}$ holds for every $i$ $(1 \leq i \leq \log n)$. Hence, no station is working for two more iterations after the first iteration of `list-shrink`. Thus, the $\log n$ iterations of `list-shrink`, `prefix-sums`, and `node-transfer` can be done in $O(n+\frac{n}{2}+\frac{n}{4}+\cdots+1)=O(n)$ time slots with each station being awake for $O(1)$ time slots. We can modify `rewind` according to new position of nodes. Finally, we have the following important theorem.

**Theorem 1** *The list ranking of an $n$-node linked list given to $n$ stations can be done in $O(n)$ time slots with no station being awake for more than $O(1)$ time slots on the single-channel BCM with no CD.*

## 4. List Ranking on the $k$-channel BCM

This section is devoted to show that the list ranking can be done in $O(\frac{n}{k})$ time slots on the $k$-channel BCM with no station being awake for $O(1)$ time slots. Our idea is to simulate the single-channel list ranking algorithm on the $k$-channel BCM. We first show a list ranking algorithm on the BCM which has exactly $\sqrt{n}$ channels. We then go on to generalize this algorithm to run on the $k$-channel BCM.

We first demonstrate how we simulate `left-shrink` on the $\sqrt{n}$-channel BCM. Imagine that nodes are partitioned into $\sqrt{n}$ groups such that the $i$-th $(1 \leq i \leq \sqrt{n})$ group consists of nodes in the range $[(i-1)\sqrt{n}+1, i\sqrt{n}]$. Each maximal sublist is partitioned into segments so that a segment consists of nodes in the same group. Procedure `left-shrink` is simulated by two subprocedures `left-shrink1` and `left-shrink2` that we describe next. In `left-shrink1`, each segment is shrunk. After removing leaf nodes, `left-shrink2` shrinks each maximal left sublist. Figure 3 illustrates `left-shrink1` and `left-shrink2`.

In `left-shrink1`, segments consist of nodes in the range $[(i-1)\sqrt{n}+1, i\sqrt{n}]$ are shrunk using channel $i$ $(1 \leq i \leq \sqrt{n})$. For any pair $i, j$ $(1 \leq i, j \leq \sqrt{n})$, let $\|i,j\|$ denote $(i-1)\sqrt{n}+j$. The details of `left-shrink1` are spelled out as follows:

```
left-shrink1
for $i \leftarrow 1$ to $\sqrt{n}$ do in parallel
```

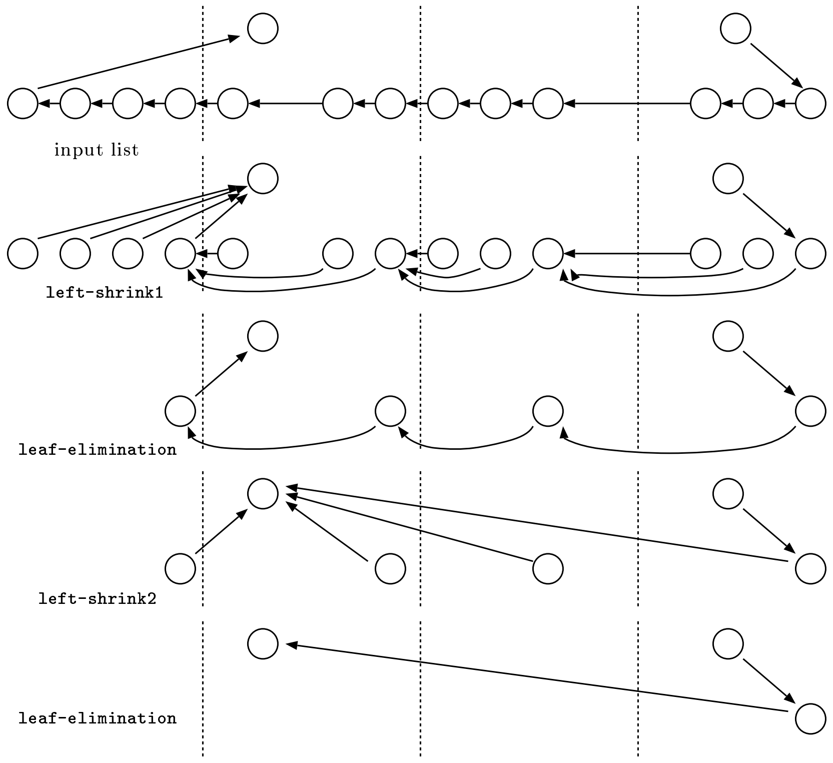

Figure 3: Left shrink on the $k$-channel

```
  for $j \leftarrow 1$ to $\sqrt{n}$ do
    $S(\|i,j\|)$ broadcasts $q[\|i,j\|]$ and $r[\|i,j\|]$ on channel $i$.
    $S(\|i,j'\|)$ satisfying $q[\|i,j'\|] = \|i,j\| < \|i,j'\|$ receives them from channel $i$
        and sets $q[\|i,j'\|] \leftarrow q[\|i,j\|]$ and $r[\|i,j'\|] \leftarrow r[\|i,j'\|] + r[\|i,j\|]$.
```

Clearly, `left-shrink1` runs in $\sqrt{n}$ time slots with each station being awake for at most two time slots. After executing `left-shrink1`, $q[\|i,j\|]$ is storing a new pointer, which is the successor of the tail node of the segment. After that, leaves are eliminated using the $\sqrt{n}$ channel in a similar way as follows:

```
leaf-elimination
for $i \leftarrow 1$ to $\sqrt{n}$ do in parallel
  for $j \leftarrow 1$ to $\sqrt{n}$ do
  $S(\|i',j'\|)$ broadcasts $\|i',j'\|$ on channel $i$ if $\|i,j\| = q[\|i',j'\|]$.
  $S(\|i,j\|)$ monitors channel $i$.
  If the status of the channel is NULL then node $\|i,j\|$ is a leaf.
```

Clearly, all nodes in the maximal left sublists but the tails of the segments are removed by `leaf-elimination`. Note that the remaining tails in the same maximal left sublist are in distinctive groups. Using this fact, `left-shrink2` shrinks maximal left sublists. Recall that, in `left-shrink1`, nodes $||i,1||, ||i,2||, \ldots, ||i,\sqrt{n}||$ are broadcast on channel $i$ in this order. In `left-shrink2`, $\sqrt{n}$ nodes $||1,j||, ||2,j||, \ldots, ||\sqrt{n},j||$ are broadcast on channel $j$. This broadcast enables us to shrink maximal left sublists. The details are spelled out as follows:

```
left-shrink2
for $j \leftarrow 1$ to $\sqrt{n}$ do in parallel
  for $i \leftarrow 1$ to $\sqrt{n}$ do
    $S(||i,j||)$ broadcasts $q[||i,j||]$ and $r[||i,j||]$ on channel $j$.
    $S(||i',j||)$ satisfying $q[||i',j||] = ||i,j|| < ||i',j||$ receives them from channel $j$
      and sets $q[||i',j||] \leftarrow q[||i,j||]$ and $r[||i',j||] \leftarrow r[||i',j||] + r[||i,j||]$.
```

Clearly, `left-shrink2` runs in $\sqrt{n}$ time slots with each station being awake for at most two time slots. It is easy to see that, after executing `left-elimination` again, all nodes but one in each maximal sublist are removed. Similarly, we can perform `right-shrink` on the $k$-channel BCM.

Again, `leaf-elimination` on the $k$-channel BCM uses the collision detection capability. We avoid using this capbability similarly to the single-channel BCM case. A node $||i,j||$ is a leaf node at the end of `left-shrink1` iff the values of $q[||i,j||]$ and $q^{-1}[||i,j||]$ before starting `left-shrink1` satisfy one of the following three conditions:

- $q^{-1}[||i,j||]]$ does not exists,
- $q[||i,j||] < ||i,j|| < q^{-1}[||i,j||]$ and nodes $||i,j||$ and $q^{-1}[||i,j||]$ are in the same group or
- $q[||i,j||] > ||i,j||$ and $||i,j|| < q^{-1}[||i,j||]$

Hence, once every station $S(||i,j||)$ learns the value of $p^{-1}[||i,j||]$, it can determine if node $||i,j||$ is eliminated in `leaf-elimination`. It is easy to see that every $S(||i,j||)$ can learn $p^{-1}[||i,j||]$ in $O(\sqrt{n})$ time slots with each station being awake for two time slots on the single-channel BCM with no CD. Consequently, procedure `leaf-elimination` executed after `left-shrink1` can be done without the collision detection capability. Procedure `leaf-elimination` after `left-shrink2` can be done without CD similarly to the single-channel case. Thus, `list-shrink` running in $O(n)$ time slots on the single-channel BCM can be simulated in $O(\sqrt{n})$ time on the $\sqrt{n}$-channel BCM.

Next, we are going to show how we implement `prefix-sums` for $n$ numbers in the $\sqrt{n}$-channel BCM. Suppose that the input array $a$ is partitioned into $\sqrt{n}$ groups of $\sqrt{n}$ numbers. It is easy to see that, using a single channel the (*local*) prefix-sums within each group can be computed in $\sqrt{n}$ time slots. Let $A(i)$ $(1 \leq i \leq \sqrt{n})$ be the sum of numbers in group $i$. After that, the (*global*) prefix-sums of $A(1), A(2), \ldots, A(\sqrt{n})$ is computed using a single channel in $\sqrt{n}$ time slots. Each prefix-sum of array $a$ is computed by adding a local prefix-sum and a global prefix-sum in obvious way. Thus, the `prefix-sums` of $n$ numbers can be computed using $\sqrt{n}$ channels in $O(\sqrt{n})$ time slots. It is easy to see that no station being awake for more than $O(1)$ time slots. Further, we can implement `node-transfer` to run in $\sqrt{n}$ time slots with no station being awake more than $O(1)$ time slots in a obvious way. Consequently, `list-shrink`, `prefix-sums`, and `node-transfer` for an $n$-node list can be implemented on the $\sqrt{n}$-channel BCM to run in $O(\sqrt{n})$ time slots with no station being awake for more than $O(1)$ time slots.

```
left-shrink1
for $i \leftarrow 1$ to $\sqrt{n}$ do in parallel
  for $j \leftarrow 1$ to $\sqrt{n}$ do
    $S(\|i,j\|)$ broadcasts $q[\|i,j\|]$ and $r[\|i,j\|]$ on channel $i$.
    $S(\|i,j'\|)$ satisfying $q[\|i,j'\|] = \|i,j\| < \|i,j'\|$ receives them from channel $i$
        and sets $q[\|i,j'\|] \leftarrow q[\|i,j\|]$ and $r[\|i,j'\|] \leftarrow r[\|i,j'\|] + r[\|i,j\|]$.
```

After executing `list-shrink`, `prefix-sums`, and `node-transfer` on an $n$-node list, we obtain the shrunk list with less than $n$ nodes. Let $n_1, n_2, \ldots, n_{\log n}$ be the number of nodes such that $n_i$ is the number of remaining $n$ nodes after the $i$-th iteration of `list-shrink` and `prefix-sums` on the $k$-channel BCM. We use $\frac{n}{2^i}$ $(\geq n_i)$ processors and $\sqrt{\frac{n}{2^i}}$ channels to perform the $i$-th iteration, which takes $O(\sqrt{\frac{n}{2^i}})$ time slots. Hence, $\log n$ iterations take at most $O(\sqrt{n} + \sqrt{\frac{n}{2^1}} + \sqrt{\frac{n}{2^2}} + \cdots + \sqrt{1}) = O(\sqrt{n})$ time slots. Thus, the list ranking can be done in $O(\sqrt{n})$ time slots with no station being awake for $O(1)$ time slots on the $\sqrt{n}$-channel BCM.

Next, let us consider the case when the BCM has less than $\sqrt{n}$ channels. Let $k$ $(\leq \sqrt{n})$ be the number of available channels. Communication using $\sqrt{n}$ channels can be simulated in $\frac{\sqrt{n}}{k}$ time slots in obvious way. Thus, the list ranking problem can be solved in $\frac{\sqrt{n}}{k} \times O(\sqrt{n}) = O(\frac{n}{k})$ time slots. Consequently, we have,

**Lemma 5** *The list ranking of an $n$-node linked list given to $n$ stations can be done in $O(\frac{n}{k})$ time slots with no station being awake for more than $O(1)$ time slots on the $k$-channel BCM with no CD provided that $k \leq \sqrt{n}$.*

In what follows, we show a list ranking algorithm on the BCM with more than $\sqrt{n}$ channels. Suppose that $n^{\frac{2}{3}}$ channels are available. We partition the nodes into $n^{\frac{2}{3}}$ groups such that the $i$-th $(1 \leq i \leq n^{\frac{2}{3}})$ group in the range $[(i-1)n^{\frac{1}{3}} + 1, in^{\frac{1}{3}}]$. Each maximal sublist is partitioned into segments so that a segment consists of nodes in the same group. We assign to each of the $n^{\frac{2}{3}}$ channels to a group, and shrink each segment in $O(n^{\frac{1}{3}})$ time slots in a similar way to `left-shrink1`. We then remove leaf nodes similarly to `leaf-elimination`. Clearly, at most one node remains in each segment.

Next, we repartition nodes into $n^{\frac{1}{3}}$ groups such that the $i$-th $(1 \leq i \leq n^{\frac{1}{3}})$ group consists of nodes in the range $[(i-1)n^{\frac{2}{3}} + 1, in^{\frac{2}{3}}]$. We also repartition each maximal left sublist into segments based on the repartitioned groups. We assign $n^{\frac{1}{3}}$ channels to each group and shrink each segment in a similar way to `left-shrink2`. This takes $O(n^{\frac{1}{3}})$ time slots. After that, all leaves are removed.

We then go on to shrink each maximal left sublist. For this purpose, every node $i$ $(1 \leq i \leq n^{\frac{2}{3}})$ in the 1st group broadcasts $q[i]$ and $r[i]$ on channels. Since $n^{\frac{2}{3}}$ nodes are in the 1st group, this is feasible. All nodes $j$ satisfying $q[j] = i < j$ receive them, and performs $q[i] \leftarrow q[j]$ and $r[j] \leftarrow r[j] + r[i]$. Next, every node

$i$ ($n^{\frac{2}{3}}+1 \leq i \leq 2n^{\frac{2}{3}}$) in the 2nd group broadcasts $q[i]$ and $r[i]$ on channels. All nodes $j$ satisfying $q[j]=i>j$ receive them and change $q[j]$ and $r[j]$ in the same way. Continuing similarly, every maximal left sublist can be shrunk. Using this technique, the list ranking problem can be solved in $O(n^{\frac{1}{3}})$ time on the $n^{\frac{2}{3}}$-channel BCM.

Finally, suppose that $n^{1-\frac{1}{c}}$ channels available for any fixed $c \geq 2$. We generaize `left-shrink1` and `left-shrink2` as follows: Let $\|i_{c-1}, i_{c-2}, \ldots, i_0\|$ denote an integer satisfying $\|i_{c-1}, i_{c-2}, \ldots, i_0\| = i_{c-1}n^{\frac{c-1}{c}} + i_{c-2}n^{\frac{c-2}{c}} + \cdots + i_0$.

```
left-shrink(m)
for all $1 \leq i_{c-1}, \ldots, i_{m+1}, i_{m-1}, \ldots, i_0 \leq n^{\frac{1}{c}}$ do in parallel
  for $i_m \leftarrow 1$ to $n^{\frac{1}{c}}$ do
    $S(\|i_{c-1}, i_{c-2}, \ldots, i_0\|)$ broadcasts $q[\|i_{c-1}, i_{c-2}, \ldots, i_0\|]$
    and $r[\|i_{c-1}, i_{c-2}, \ldots, i_0\|]$ on channel $\|i_{c-1}, \ldots, i_{m+1}, i_{m-1}, \ldots, i_0\|$.
    $S(\|i_{c-1}, \ldots, i_{m+1}, j, i_{m-1}, \ldots, i_0\|)$ satisfying
      $q[\|\imath_{c-1}, \ldots, i_{m+1}, j, i_{m-1}, \ldots, i_0\|] = \|i_{c-1}, i_{c-2}, \ldots, i_0\| <$
        $\|i_{c-1}, \ldots, i_{m+1}, j, i_{m-1}, \ldots, i_0\|$
      receives them from it and sets
      $q[\|i_{c-1}, \ldots, i_{m+1}, j, i_{m-1}, \ldots, i_0\|] \leftarrow q[\|i_{c-1}, i_{c-2}, \ldots, i_0\|]$ and
      $r[\|i_{c-1}, \ldots, i_{m+1}, j, i_{m-1}, \ldots, i_0\|] \leftarrow += r[\|i_{c-1}, i_{c-2}, \ldots, i_0\|]$.
```

Clearly, `left-shrink(m)` uses $(n^{\frac{1}{c}})^{c-1} = n^{1-\frac{1}{c}}$ channels. Also, `left-shrink(1)` and `left-shrink(2)` correspond to `left-shrink1` and `left-shrink2` if $c=2$. Thus, the reader should have no difficulty to confirm that, a sequence of procedures, `left-shrink(1)`, `leaf-elimination` `left-shrink(2)`, `leaf-elimination`, …, `left-shrink(m)`, `leaf-elimination` simulates `list-shrink`. Further, all the other procedures including `right-shrink`, `leaf-elimination`, `prefix-sums`, `node-transfer`, and `rewind` can be simulated similarly on the $n^{1-\frac{1}{c}}$-channel BCM. Therefore, we have

**Theorem 2** *For every $c \geq 2$, the list ranking of an $n$-node linked list given to $n$ stations can be done in $O(cn^{\frac{1}{c}})$ time slots with no station being awake for more than $O(c)$ time slots on the $n^{1-\frac{1}{c}}$-channel BCM with no CD.*

Let $\epsilon = \frac{1}{c}$ be a small fixed real number. From above theorem, we have the following important corollary.

**Corollary 1** *For any small fixed $\epsilon > 0$, the list ranking of an $n$-node linked list can be done in $O(\frac{n}{k})$ time slots with no station being awake for more than $O(1)$ time slots on the $k$-channel $n$-station BCM with no CD provided that $k \leq n^{1-\epsilon}$.*

## 5. Concluding Remarks

We have shown that the rank of every node in an $n$-node linked list can be determined in $O(n)$ time slots with no station being awake for more than $O(1)$ time slots on the single-channel $n$-station BCM. We have extended this algorithm to run on the $k$-channel BCM. For any small fixed $\epsilon > 0$, our list ranking algorithm runs in $O(\frac{n}{k})$ time slots with no station being awake for more than $O(1)$ time slots, provided that $k \leq n^{1-\epsilon}$. Clearly, $\Omega(\frac{n}{k})$ time is necessary to solve the list ranking problem for an $n$-node linked list on the $k$-channel BCM. Therefore, our list ranking algorithm on the $k$-channel BCM is time and energy optimal. Since the number of channels is small in practice, our algorithm is always optimal. However, from a theoretical point of view, it remains open to show a list ranking algorithm running in $O(\log n)$ time slots with each station being awake for $O(1)$ time slots on the $\frac{n}{\log n}$-channel BCM.

## References

1. N. Abramson, Ed., *Multiple Access Communications: Foundations for Emerging Technologies*, IEEE Press, New York, 1993.
2. N. Abramson, Multiple access in wireless digital networks, *Proceedings of the IEEE*, 82, (1994), 1360–1370.
3. S. G. Akl, *Parallel Computation: Models and Methods* Prentice Hall, 1997.
4. R. Bar-Yehuda, O. Goldreich, and A. Itai, Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection, *Distributed Computing*, 5, (1991), 67–71.
5. R. Bar-Yehuda, O. Goldreich, and A. Itai, On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization, *Journal of Computer and Systems Sciences*, 45, (1992), 104–126.
6. D. Bertzekas and R. Gallager, *Data Networks*, Second Edition, Prentice-Hall, 1992.
7. R. Binder, N. Abramson, F. Kuo, A. Okinaka, and D. Wax, ALOHA packet broadcasting – a retrospect, *AFIPS Conf Proceedings*, May 1975, 203–216.
8. J. L. Bordim, J. Cui, T. Hayashi, K. Nakano, and S. Olariu, Energy-efficient initialization protocols for ad-hoc radio network, *IEICE Trans. on Fundamentals*, E83-A, 9, pp.1796-1803, 2000.
9. T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to algorithms*, MIT Press, 1994.
10. R. Cole and U. Vishkin, Deterministic Coin Tossing With Applications to Optimal Parallel List Ranking, *Information and Control*, 70,32–56, 1986.
11. R. Cole and U. Vishkin, Approximate parallel scheduling. Part I: The basic technique with applications to optimal parallel list ranking in logarithmic time, *SIAM J. Computing*, 17, 1, 128–142, 1988
12. K. Feher, *Wireless Digital Communications*, Prentice-Hall, Upper Saddle River, NJ, 1995.
13. A. Gibbons and W. Rytter, *Efficient parallel algorithms*, Cambridte University Press, 1988.
14. E. P. Harris and K. W. Warren, Low power technologies: a system perspective, *Proc. 3-rd International Workshop on Multimedia Communications*, Princeton, 1996.
15. T. Hayashi, K. Nakano, and S. Olariu, Efficient List Ranking on the Reconfigurable Mesh, with Applications, *Theory of Computing Systems*, 31, 593–611, 1998.
16. J. JáJá, *An introduction to parallel algorithms*, Addison-Wesley, 1992.
17. E. D. Kaplan, *Understanding GPS: principles and applications*, Artech House, Boston, 1996.
18. F. T. Leighton, *Introduction to parallel algorithms and architectures*, Morgan Kaufmann, 1992.

19. R. M. Metcalfe and D. R. Boggs, Ethernet: distributed packet switching for local computer networks, *Communications of the ACM*, 19, (1976), 395–404.

20. A. Micic, and I. Stojmenovic, A hybrid randomized Initialization protocol for TDMA in single-hop wireless networks, *Workshop on Advances in Parallel and Distributed Computational Models*, 2002.

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

22. K. Nakano and S. Olariu, Uniform Leader Election Protocols in Radio Networks, to appear in *IEEE Trans. on Parallel and Distributed Systems*, 2002

23. K. Nakano, S. Olariu, A. Y. Zomaya, Energy-Efficient Deterministic Routing Protocols in Radio Networks, *IEEE Trans. on Parallel and Distributed Systems*, 12, 6, 544–557, 2001.

24. 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.

25. R.A. Powers, Batteries for low-power electronics, *Proc. IEEE*, 83, pp.687–693, 1995.

26. M. Reid-Miller, *List Ranking and List Scan on the CRAY C-90*, ournal of Computer and System Sciences", (1996), 53, 3.

27. A. K. Salkintzis and C. Chamzas, An in-band power-saving protocol for mobile data networks, *IEEE Transactions on Communications*, COM-46, (1998), 1194–1205.

28. K. Sivalingam, M. B. Srivastava, and P. Agrawal, Low power link and access protocols for vireless multimedia networks, *Proc. IEEE Vehicular Technology Conference VTC'97*, Phoenix, AZ, May, 1997.

29. D. E. Willard, Log-logarithmic selection resolution protocols in a multiple access channel, *SIAM Journal on Computing*, 15, (1986), 468–477.