Time and Energy Optimal List Ranking Algorithms 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 a data packet to one of k distinct communication channels. 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. 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 have tuned to the channel. The nature of end units is immaterial: they can 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 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. For example, in radio networks, synchronization may be provided by an interface to a commercially-available Global Positioning System [9].

For simplicity, we assume that the BCM has the following collision detection capability [11]: 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. Although our algorithms presented in this paper use the collision detection capability, it is not difficult to modify them such that they do not use this capability.

We assume that broadcasting/receiving data packets in the BCM is very costly. 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 in the radio network expends power while its transceiver is active that is, while transmitting or receiving a packet [5, 12]. 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 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. As we are going to show later, the goals of optimizing these parameters are, of course, conflicting. It is relatively straightforward to minimize overall completion time at the expense of energy consumption.

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 node following 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. The list ranking problem asks to determine *the rank* of every node i ($1 \le i \le n$), which is the distance to the end of the list. Nodes 2 and 8 in Figure 1 are the top and the end nodes of the list, respectively.

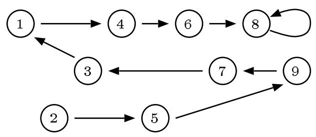

Fig. 1. An example of a linked list

The list ranking problem has been solved in several contexts [4, 7, 13]. It is well known that the list ranking problem can be solved in $O(\log n)$ time using n processors on the (EREW) PRAM [6, 8]. This parallel list ranking algorithm uses *the pointer jumping technique* [1, 8], which repeats changing 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 of nodes. Further, it is known that the number of processors can be reduced to $\frac{n}{\log n}$ [4,8].

One of the straightforward strategies to design an energy-efficient algorithm on the BCM is to simulate a known PRAM algorithm. 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. 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. Further, communication using n channels can be simulated on the k-channel BCM ($k \leq n$) in $O(\frac{n}{k})$ time slots. It is known that the list ranking problem can be solved in $O(\log n)$ time using n processors on the PRAM [6,8]. 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 ranking algorithms on the BCM. 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. 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.

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 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, $q[i]$ is storing the pointer to the end of the list for every i. Thus, each $r[i]$ is storing the rank of node i. This technique used in the pointer jumping [8], 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 reader should have no difficulty to confirm the correctness of the pointer jumping.

Our list ranking algorithm also uses the 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 $(1 \le j \le m-1)$. 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.

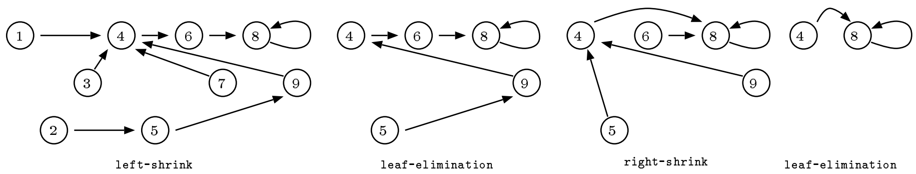

Fig. 2. Each step of `list-shrink`

We will show the details of each step of `list-shrink`. Let q and r be 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 $p[i]$ and is asleep for the other time slots. Suppose that `left-shrink` is executed on the linked list in Figure 1. Note that $r[8] = 0$, and $r[i] = 1$ for all i ($i \neq 8$). 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 q and r after each step of `list-shrink`. In the figure, 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 are nodes 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 $j = p[i]$.
  $S(i)$ monitors the channel.
    If the status of the channel is NULL then node i is a leaf.

In `leaf-elimination`, station $S(i)$ is awake at time slots i and $p[i]$. 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 half nodes are in the list after executing `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 $2s \leq n$. After Step 2, every node that are 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 are 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 adjacent sublists may be merged into one. Since every left sublist has two nodes, every node is in one of the maximal right sublists. Let $L'_1, L'_2, \ldots, L'_{s'}$ ($s' \leq s$) denote the maximal sublists after Step 2. We can evaluate the number of nodes in the list obtained after Step 4 as follows. The obtained list contains at most two nodes in $L'_{s'}$. The two nodes are the end of the whole list and the head of $L'_{s'}$. The list also contains the tail of each L'_i ($2 \leq i \leq s'$). All nodes in L'_1 are eliminated in Step 4. Thus, the list thus obtained has at most s' ($\leq s \leq \frac{n}{2}$) nodes. Therefore, 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 $2 \log n$ iterations of `leaf-elimination` as follows. For each node i but the end of the list, let $t(i)$ denote an integer such that node i has eliminated in $t(i)$-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 ← 2 log n downto 1 do
   for i ← 1 to n do
      if t(i) ≤ 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] ← q[j] and r[j] ← r[j] + r[i].
```

Table 1 also 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. Clearly, station i is awake for $O(t(i))$ time slots for list ranking. From Lemma 2, the number of nodes i satisfying $t(i) \geq 2A$ is at most $\frac{n}{2^A}$ for every $1 \leq A \leq \log n$. Therefore, we have,

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

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. More precisely, 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 next. 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 $S(i)$ leans $\mathit{prefix}(i)$ when `prefix-sums` terminates. Further each station $S(i)$ is awake for at time slots i and $i-1$ and is asleep for the other time slots.

The remaining nodes can be renumbered using `prefix-sums`. Suppose that `list-shrink` is executed on 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, each remaining node i is assigned 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 arrange them in 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 are remaining. By `prefix-sums`, nodes 4 and 8 receives new IDs $\mathit{prefix}(4) = 1$ and $\mathit{prefix}(8) = 2$, respectively.

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 $\frac{n}{2} + i$ and $\frac{n}{2} + q'(i)$. 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. After 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 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 more than two 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.*

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. Due to the stringent page limitation, we only demonstrate how we simulate `left-shrink` and `leaf-elimination` on the $\sqrt{n}$-channel BCM. The other procedures can be simulated similarly.

Imagine that nodes are partitioned into $\sqrt{n}$ groups such that 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.

In `list-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
  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 similarly as follows:

`leaf-elimination`
for $i \leftarrow 1$ to $\sqrt{n}$ in parallel
  for $j \leftarrow 1$ to $\sqrt{n}$ do
  $S(|i',j'|)$ broadcasts $|i',j'|$ on channel i if $|i,j| = p[|i',j'|]$.
  $S(|i,j|)$ 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. On the other hand, in `left-shrink2`, 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 $\sqrt{n}$ 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 $r[i',j] \leftarrow r[i',j] + r[i,j]$.

Again, `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 simulate `right-shrink` and `rewind`, `prefix-sums`, `node-transfer` on the k-channel BCM.

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 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 i-th iteration, which takes $O(\sqrt{\frac{n}{2^i}})$ time slots. It follows that $\log n$ 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. Hence, the list ranking problem can be solved in $\frac{\sqrt{n}}{k} \times O(\sqrt{n}) = O(\frac{n}{k})$ time slots. Thus, we have,

Lemma 4. *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 provided that $k \leq \sqrt{n}$.*

It is not difficult to generalize our list ranking algorithm on the $\sqrt{n}$-channel BCM to run on the $n^{1-\frac{1}{c}}$-channel BCM for any fixed $c \geq 2$. For example, procedure `left-shrink` can be simulated by executing the $n^{1-\frac{1}{c}}$-channel version of `left-shrink` for c times, each of which runs in $O(n^c)$ time slots. Consequently, we have the following theorem:

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.*

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 provided that $k \leq n^{1-\epsilon}$.*

References

1. S. G. Akl, *Parallel Computation: Models and Methods* Prentice Hall, 1997.
2. 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.
3. T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to algorithms*, MIT Press, 1994.
4. 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
5. K. Feher, *Wireless Digital Communications*, Prentice-Hall, Upper Saddle River, NJ, 1995.
6. A. Gibbons and W. Rytter, *Efficient parallel algorithms*, Cambridge University Press, 1988.
7. T. Hayashi, K. Nakano, and S. Olariu, Efficient List Ranking on the Reconfigurable Mesh, with Applications, *Theory of Computing Systems*, 31, 593–611, 1998.
8. J. JáJá, *An introduction to parallel algorithms*, Addison-Wesley, 1992.
9. E. D. Kaplan, *Understanding GPS: principles and applications*, Artech House, Boston, 1996.
10. F. T. Leighton, *Introduction to parallel algorithms and architectures*, Morgan Kaufmann, 1992.
11. K. Nakano and S. Olariu, Randomized initialization protocols for ad-hoc networks, *IEEE Trans. on Parallel and Distributed Systems*, 11, 7, 749–759, 2000.
12. R.A. Powers, Batteries for low-power electronics, *Proc. IEEE*, 83, pp.687–693, 1995.
13. M. Reid-Miller, *List Ranking and List Scan on the CRAY C-90*, Journal of Computer and System Sciences", (1996), 53, 3.