List Ranking on Interconnection Networks

Jop F. Sibeyn

Max-Planck-Institut für Informatik,
Im Stadtwald, 66123 Saarbrücken, Germany.
E-mail: jopsi@mpi-sb.mpg.de, Fax: +49-681-3025401.

Abstract

The list-ranking problem is considered for parallel computers which communicate through an interconnection network. Each PU holds k nodes of a set of singly linked lists. An easy randomized algorithm gives a considerable improvement over earlier ones.

For a large class of networks, the algorithm takes only twice the number of steps required by a k-k routing. The only conditions are that: (1) $k = \omega(k^*)$, where k^* is so large that the time consumption of k^*-k^* routing is determined by the bisection bound, and (2) the routing time slightly increases with the number of PUs in the network.

For special networks we can prove stronger results. Particularly, for $n \times \cdots \times n$ meshes, the list ranking problem is solved in $(1/2 + o(1)) \cdot k \cdot n$ steps, if $k = \omega(1)$. For hypercubes with N PUs, assuming all-port communication, the algorithm requires only $(2 + o(1)) \cdot k$ steps, if $k = \omega(\log^2 N)$.

We show that list ranking requires at least the time required for k-k routing. So, the results are within a factor two from optimal. For meshes we even match the lower bound up to lower-order terms.

Keywords: parallel algorithms, interconnection networks, list ranking, randomization.

1 Introduction

Lists. A *linked list*, hereafter just *list*, is a basic data structure: it consists of nodes which are linked together, such that every node has precisely one predecessor and one successor, except for the *initial node*, which has no predecessor, and the *final node*, which has no successor. Lists play a central role, both in sequential and in parallel algorithms. For example, they are used in the Euler-tour technique (see [8]), which is of outstanding importance in the theory of parallel computation: it is applied as a subroutine in many parallel algorithms for problems on trees.

List Ranking. An important problem connected to the use of lists is the *list ranking* problem: the determination of the rank of a node within its list. Once the nodes have been ranked, the lists can be transformed into arrays, on which many parallel operations can be performed more efficiently. For example, the parallel computation of 'prefix sums' is much easier on arrays than on lists. Also, the Euler-tour technique involves ranking lists.

In sequential computation the list ranking problem goes unnoticed, because it is trivial to solve: following the links, a list of length N is ranked in $\mathcal{O}(N)$ time. In particular, list-ranking is easier than sorting. In parallel computation it is much more important, and often the time for the list ranking determines the time of the algorithm in which it is applied.

Related Work. List-ranking has been studied intensively on various parallel computer models. Here we only mention the most relevant works, a more complete description is given in the full version of this paper (there one can also find the omitted proofs and alternative algorithms) [16].

On PRAMs optimal deterministic list ranking is achieved in [4]: time $\mathcal{O}(\log N)$ and work $\mathcal{O}(N)$. More realistic than PRAMs are parallel computers consisting of N processing units, *PUs*, that communicate through an interconnection network. On such parallel computers, it is hard to achieve anything worth mentioning: by its nature, list ranking is an extremely non-local problem. For hypercubes Ryü and JáJá [15] have shown that linear speed-up can be achieved if

every PU holds at least $k = N^\epsilon$ nodes, for $\epsilon > 0$ a constant. Though the time *order* is optimal, $\mathcal{O}(k)$ in the all-port model, the leading constant strongly increases with $1/\epsilon$, limiting its practicality. Randomizedly, the problem has been solved in $(7 + o(1)) \cdot k$, if $k = \omega(\log^2 N)$ [16]. The list-ranking problem on meshes has been considered in [2, 5, 16]. The fastest algorithm for $n \times n$ meshes (randomized) takes $(3^1/_2 + o(1)) \cdot k \cdot n$ steps [16].

Reid-Miller [12] analyzes list-ranking algorithms for a 'vector processor', the CRAY C-90. She advocates the use of an algorithm, which follows the same basic approach as ours. Both algorithms can be viewed as simplified, randomized versions of the earlier algorithm by Anderson and Miller [1]. We have added various novel ideas, and for the first time we

- precisely formulate the communication complexity of list ranking for a large class of networks, allowing for general application and prediction of its cost;
- allow for a set of lists, rather than one long list, without needing extra routing steps;
- give nearly optimal results for list ranking on meshes and hypercubes, thereby solving two important open problems.

The second point is important, because in most applications, e.g. in parallel connected components algorithms the ranking has to be performed on a set of intermixed lists.

This Paper. We develop efficient randomized list-ranking algorithms. The main algorithm is formulated for arbitrary networks. Its versatility is demonstrated by giving modifications which perform very well for meshes and hypercubes. We also show that list ranking requires at least as many steps as routing. Because this paper aims to contribute theory with a practical impact, we concentrate on the case that every PU holds $k \gg 1$ nodes of a set of lists.

For hypercubes under the all-port assumption we obtain a running time of $(2 + o(1)) \cdot k$ steps for all $k = \omega(\log^2 N)$. On $n \times \cdots \times n$ meshes, for $k = \omega(1)$, a refined algorithm only requires $(1/2 + o(1)) \cdot k \cdot n$ steps. This equals the routing time, and hence the lower bound, up to lower-order terms.

We do not formulate our results that way, but the algorithm fits well into a BSP-like approach [17]. In fact, our algorithm can be viewed as a work-optimal PRAM algorithm with sub-optimal running time, which is very well suited for implementation on networks.

2 Preliminaries

Networks. There are various models for parallel computers. We consider distributed memory machines. That is, every PU has its own local memory, and data are exchanged over a fixed interconnection network. The number of PUs is N. The *degree* of a network is the maximum degree of any PU; its *diameter* is the maximal distance between any pair of PUs. The *bisection width*, $BW(N)$, is the minimum number of connections that has to be removed to obtain two disjoint networks with $\lfloor N/2 \rfloor$ and $\lceil N/2 \rceil$ PUs, respectively.

Meshes and Hypercubes. are among the best studied and most constructed machines with a fixed interconnection network, because of their regularity, programmability and other positive features. In a d-dimensional *mesh*, the N PUs are laid out in a d-dimensional $n \times \cdots \times n$ grid. Here and in the remainder, we use $n = N^{1/d}$. A PU is connected to its (at most) $2 \cdot d$ immediate neighbors. A *hypercube* with N PUs is a $\log N$-dimensional $2 \times \cdots \times 2$ grid.

Communication. We assume that the connections allow bi-directional communication. The PUs can communicate with all their neighbors at the same time: the *all-port model*. If on a real mesh or hypercube this assumption does not hold, then one step can trivially be split into several more elementary routing operations.

Performance Measure. The performance of an algorithm running on an interconnection network is measured by the maximum number of routing steps T it may take. This 'store-and-forward model' is very common in theoretical papers. It reflects the features of routing on SIMD machines (CM-2, MasPar). However, modern parallel computers tend rather to be MIMD machines consisting of larger and more powerful PUs (GCel, AP1000, iWarp, Touchstone Delta, Intel Paragon, J-Machine). With the applied 'wormhole routing' the distance between nodes plays a minor role (see [7, Sec. 7.3.2]). Fortunately, our algorithms are robust relative to such details of the model. In the general algorithm, the time consumption is expressed in terms of the time consumption of a corresponding routing operation.

Problem Definition. Initially, every PU holds k nodes. Every node has a unique index: node p, residing in memory position j, $0 \leq j < k$, of PU i, $0 \leq i < N$, has index $ind(p) = i \cdot k + j$. This index has nothing to do with its rank, but is used to compute the PU in which a node resides. The successor (predecessor) of a node p in the list is denoted $suc(p)$ ($pred(p)$). p knows the index of its successor, but, it does not know the index of its predecessor. The first and last nodes of every list know their status. The rank of p is denoted $r(p)$. It gives the 'distance' from the initial nodes. The rank of the initial nodes is fixed on 0. The described problem will be denoted $lrp(k, N)$ and the required number of steps by $T_{\text{rank}}(k, N)$.

Routing and Sorting. *k-k routing* is the routing problem, in which every PU is the source and destination of at most k packets. In k-k sorting, the packets must be sorted on a key from a totally ordered set with respect to a given indexing of the PUs. For a given network, $T_{\text{route}}(k, N)$ denotes the number of steps to perform k-k routing. $T_{\text{sort}}(k, N)$ is defined analogously. Let k^* be the smallest number such that for all $k \geq k^*$ the routing time increases linearly with k and equals the sorting time:

$$T_{\text{route}}(k, N) = k/k^* \cdot T_{\text{route}}(k^*, N), \quad (1)$$

$$T_{\text{sort}}(k, N) = T_{\text{route}}(k, N). \quad (2)$$

The existence of a k satisfying (1) is a basic assumption of the BSP model [17]. The existence of a k satisfying (2) is a consequence of (1): applying sample sort [14, 13], sorting can be reduced to routing. Generally, (1) and (2) are accurate only up to lower order-terms. To get a pleasant notation, *these* lower-order terms are omitted. The lower-order terms resulting from the list-ranking algorithm are taken into account.

Mathematical Issues. Let f and g be positive functions on $\mathbb{R}$. **f is of larger order than g,** denoted $f = \omega(g)$, if for all $c > 0$, there is a $C(c)$, such that for all $x > C(c)$, $f(x) > c \cdot g(x)$. For estimates on the binomial distribution, we use

Lemma 1 (Chernoff Bounds) [6] *Let $X_1, \ldots, X_m$ be independent Bernoulli trials, with $P[X_i = 1] = p$. Let $Z = \sum_i X_i$. Then for any $t > 0$,*

$$P[Z \geq p \cdot m + t] \leq e^{-t^2/(3 \cdot p \cdot m)}.$$

For more general estimates the following inequality by McDiarmid is very useful:

Lemma 2 (Azuma Inequality) [11] *Let $X_1, \ldots, X_m$ be independent random variables. For each i, X_i takes values in a set A_i. Let $f : \prod_i A_i \to \mathbb{R}$ be a measurable function satisfying $|f(x) - f(y)| \leq c$, when x and y differ only in a single coordinate. Let Z be the random variable $f(X_1, \ldots, X_m)$. Then for any $t > 0$,*

$$P[|Z - E[Z]| \geq t] \leq 2 \cdot e^{-2 \cdot t^2/(c^2 \cdot m)}.$$

All results in this paper hold with *high probability*: their probability on failure is bounded by n^{-c}, for some constant $c > 0$.

Pointer Jumping. Suppose we have a set of lists of total length S. Suppose that for a final node p, $suc(p) = p$. The following process of repeatedly doubling is called *pointer jumping*:

repeat $\lceil \log S \rceil$ **times**
for all p **do** $suc(p) := suc(suc(p))$.

Hereafter, for all p, $suc(p)$ gives the final node of the list of p (see [8] for a proof). The algorithm can easily be modified to compute functions like the rank. Implementing each of the $\lceil \log S \rceil$ concurrent reads with a constant number of routing and sorting operations, gives

Lemma 3 *Pointer jumping can be performed in* $\mathcal{O}(\log S \cdot T_{sort}(k, N))$ *steps.*

3 Lower Bounds

For routing or sorting it is clear which packets have to go where: distance and bisection arguments give strong lower bounds. For list-ranking, there is no trivial lower bound; a priori it is not clear what information actually has to be exchanged. By counting the 'information content' of the list-ranking problem, we prove that it is not easier than routing.

Lemma 4 *lrp*(k, N) *takes at least as many steps as transferring an unknown number* $m \in \{1, 2, \ldots, (k \cdot N/2 - 1)!\}$ *over the bisection.*

Proof: Consider a single list, of which all nodes with even rank are stored on the 'left' of the bisection of the network. These nodes are stored in some order in $k \cdot N/2$ memory positions. Suppose that the initial node, whose rank is already known, is stored in position 0. The remaining arrangement is one out of $(k \cdot N/2-1)!$ permutations: there is a one-one correspondence between the numbers in $\{1, 2, \ldots, (k \cdot N/2-1)!\}$, and arrangements. Suppose that on the right side the encoded number is known. Communication and computation within each half are free. That is, we assume a two-processor system, with infinitely powerful PUs that communicate through a connection with capacity equal to the sum of the capacities of the connections through the bisection. Clearly, solving the actual list-ranking problem takes at least as long as solving it on this two-processor system. Solving the list-ranking problem in T steps gives a method to transfer the encoded number in T steps as well: if the ranks of all nodes on the left are known, then the number can be computed instantaneously. □

Lemma 5 *If at most* l *bits go over the bisection in a step, then any protocol for transferring numbers* $m \in \{1, 2, \ldots, M\}$ *over the bisection takes at least* $\lceil \log M / \log(2^l + 1) \rceil$ *steps.*

Proof: Consider the two-processor system from the proof of Lemma 4. Let A be a protocol for transferring any number $m \in \{1, 2, \ldots, M\}$, from one PU to the other. A is known by each of them. The action of A can be described by a $(2^l + 1)$-ary tree. The protocol starts at the root, at level 0. With every packet sent, it goes one level deeper. The additional term 1 is due to the case 'no-packet-transferred'. The nodes represent the set of remaining possibilities for the number m. At the root we find the whole set of cardinality M. For some inputs, the protocol may have a fast strategy, expressed by a leaf close to the root. On the other hand, it is easy to prove by induction, that at level i, there is a set with cardinality at least $\lceil M/(2^l + 1)^i \rceil$. Hence, on any level i with $i < \lceil {}^{2^l+1}\log M \rceil = \lceil \log M / \log(2^l + 1) \rceil$ there is a set with more than one element: there are still numbers among which A cannot distinguish. □

Combining these lemmas and applying Stirling's formula gives an interesting result:

Theorem 1 *If at most* $\log(k \cdot N) + \mathcal{O}(1)$ *bits go over any connection in a step, then*

$$T_{rank}(k, N) \geq (1 - o(1)) \cdot k \cdot N/(2 \cdot BW(N)).$$

4 General Algorithm

The only conditions for efficient list ranking with the algorithm of this section are

$$k = \omega(k^*), \tag{3}$$
$$T_{\text{route}}(k, \ln N) = o(T_{\text{route}}(k, N)). \tag{4}$$

Let $f = f(N)$ be the largest function such that

$$f^6 \cdot \ln f \leq k/k^*, \tag{5}$$
$$f^2 \cdot \ln f \leq T_{\text{route}}(k, N)/T_{\text{route}}(k, \ln N). \tag{6}$$

Algorithm. The algorithm, is described and analyzed step by step. Some readers may prefer to read through the clearly marked steps, before reading the remarks and lemmas. Due to a lack of space all proofs had to be omitted (see [16]). $T_i(k, N)$ denotes the number of routing steps taken by Step i.

Step 1. Every non-initial node is selected uniformly and independently as *ruler* with probability f^{-1}. Every initial node p sets $r(p) := 0$.

Lemma 6 $T_1(k, N) = 0$.

Step 2. Each ruler p initiates a 'sending wave': It sends a message $(ind(p), 1)$ to $suc(p)$. Upon reception of a packet $(ind(p), r)$, a node p' assigns $r(p') := r$. If p' is not a final node or a ruler, and if $r < f \cdot \ln f$, then it sends a packet $(ind(p), r + 1)$ to $suc(p')$.

Notice that we do *not* perform pointer jumping: every node sends at most one packet in Step 2. The sending of the packets $(*, r)$, $1 \leq r \leq f \cdot \ln f$, is called *sending round* r. The sending rounds should have barrier synchronizations between them: no packet $(*, r+1)$ should be send before all packets $(*, r)$ have arrived. The following operations are added to Step 2, to increase the efficiency of the algorithm:

Step 2^+. At the beginning of each sending round r, every initial node p that was not selected before, is designated as *pseudo-ruler* with probability $f^{-1} \cdot (1 - f^{-1})^{r-1}$. Upon selection, p behaves as a ruler, and initiates a sending wave.

At the beginning of each sending round r, the packets $(*, r)$ are spread within blocks of $\ln N$ PUs. In general, this can be done easily. Similarly, at the end of it, the packets are first sent to preliminary destinations within their $\ln N$ block.

Lemma 7 *In total over all rounds, the spreadings in Step 2 can be implemented such that the required number of steps is at most* $4/f \cdot T_{route}(k, N)$.

We analyze the sending rounds. The following lemma plays a central role in our analysis. It essentially goes back to the special way the initial nodes are designated as pseudo-rulers. Let $P_r(p)$ be the probability that a node p sends a packet in round r.

Lemma 8 *For all* p *and* r, $1 \leq r \leq f \cdot \ln f$, $P_r(p) = f^{-1} \cdot (1 - f^{-1})^{r-1}$.

Let $k_r = k \cdot P_r$, be the expected number of packets sent by a PU in round r.

Lemma 9 *In round* r, $1 \leq r \leq f \cdot \ln f$, *the number of packets sent by a PU is bounded by* $k'_r = k_r + \mathcal{O}(k_r/f)$.

Corollary 1 *The number of steps required for the routing in round* r *is bounded by* $T_{route}(k_r + \mathcal{O}(k_r/f), N)$.

Lemma 10 $T_2(k,N) = (1+\mathcal{O}(f^{-1}))\cdot T_{route}(k,N)$.

Some nodes have not been reached during Step 2: nodes at the end of a section between two rulers which lie more than $f\cdot\ln f$ out of each other, and some nodes at the beginning of the lists. Instead of expensively walking step by step, we proceed with

Step 3. Perform a list-ranking algorithm on all nodes that were not reached in Step 2. The nodes that were reached in round $f\cdot\ln f$ and the initial nodes that were not designated as pseudo-rulers act as initial list elements. The unreached rulers act as final elements. If k is still large enough, recursion may be applied. Pass along the data related to the ruler or initial node, and add appropriate values to the computed ranks.

First the participating nodes are spread within $\ln N$ blocks.

The participating nodes in Step 3 are precisely those which did not send during Step 2. By Lemma 8, the probability that a node did not send during Step 2 can be estimated by $(1-f^{-1})^{f\cdot\ln f}\leq f^{-1}$. Hence, the expected number of nodes participating in Step 3 is less than k/f in every PU.

Lemma 11 $T_3(k,N) = (1+o(f^{-1}))\cdot T_{rank}(k/f,N)$.

Initial nodes and nodes reached by waves from them know their ranks. They do not participate further. A non-ruler node, reached by a wave coming from a ruler p, is said to be a *subject* of p. Rulers and their subjects remain active.

Step 4. If a ruler p has been reached by a wave coming from a ruler p', then p sends a packet holding $ind(p)$ to p'. If p has been reached by a wave from an initial node, it marks itself as an initial node in the new set of lists.

First the packets are spread within $\ln N$ blocks.

A ruler p, receiving a packet holding $ind(p')$, marks p' as its new successor. If p does not receive a packet, it marks itself as a final node.

Lemma 12 $T_4(k,N) = (f^{-1}+o(f^{-1}))\cdot T_{route}(k,N)$.

Step 5. Perform a weighted list ranking (prefix sum) on the rulers. The weight of a ruler is the number r from the packet $(*,r)$ by which it was reached during Step 2 or Step 3. If k is still large enough, recursion may be applied.

First the rulers are spread within $\ln N$ blocks.

Lemma 13 *Rulers know their ranks after Step 5.* $T_5(k,N) = (1+o(f^{-1}))\cdot T_{rank}(k/f,N)$.

Step 6.a, 6.b. Spread the ranks of the rulers back to their subjects, by performing obvious modifications of Step 2 and 3.

Lemma 14 $T_6(k,N) = (1+\mathcal{O}(f^{-1}))\cdot T_{route}(k,N)+(1+o(f^{-1}))\cdot T_{rank}(k/f,N)$.

Summing over all steps, we obtain:

Theorem 2 *If (5) and (6) hold, then*

$$T_{rank}(k,N)\leq(2+\mathcal{O}(f^{-1}))\cdot T_{route}(k,N)+(3+o(f^{-1}))\cdot T_{rank}(k/f,N).$$

The result is very general. The recurrency cannot be solved without knowing more about the network, or without imposing additional conditions on k.

Consequences. For sufficiently large k, we now achieve within a factor two from optimal:

Theorem 3 *There is a constant x such that, if (4) holds, and $k \geq \log\log^x N \cdot \log N \cdot k^*$, then*

$$T_{rank}(k, N) \leq (2 + o(1)) \cdot T_{route}(k, N).$$

The value of the x in Theorem 3 strongly depends on the analysis and on the precise formulation of Theorem 2. The term $(3 + o(f^{-1})) \cdot T_{\text{rank}}(k/f, N)$ in Theorem 2 comes from two sources: ranking the rulers, and ranking the tails. It can be reduced to $(1 + o(f^{-1})) \cdot T_{\text{rank}}(k/f, N)$, if the 'walking' along the paths is carried out for f^2 instead of $f \cdot \ln f$ steps: the number of nodes in a tail becomes extremely small. A refined analysis shows that then $k = \log\log\log^x N \cdot \log N \cdot k^*$ is sufficient for list ranking in $(2 + o(1)) \cdot T_{\text{route}}(k, N)$ steps.

k can be taken smaller, if a known algorithm achieves the optimal time order:

Theorem 4 *Let $T_{rank}(k, N) = \mathcal{O}(T_{route}(k, N))$, for all $k \geq k^{**}$. If (4) holds, then for all $k = \omega(\max\{k^*, k^{**}\})$,*

$$T_{rank}(k, N) \leq (2 + o(1)) \cdot T_{route}(k, N).$$

Theorem 4 is very general: for $k \geq \log N \cdot k^*$, the only condition for an optimal time-order list-ranking algorithm is a parallel prefix algorithm running in $\mathcal{O}(T_{\text{route}}(k, N))$ steps [16]. This will be granted on any reasonably structured network.

5 Large-Bisection Networks

For hypercubes with all-port communication, and other networks with a large bisection width, (4) does not hold: for $k \geq k^*$, routing or sorting in a subcube is as expensive as on the whole hypercube. Thus, the given algorithm cannot be applied: the spreadings in the $\ln N$ blocks would dominate the total time consumption. What determined the condition on k, (3), and the definition of f, (5)?

- In order to apply (1) and (2), the number of packets sent by each PU should be at least k^* in all routing operations.
- The maximum number of packets sent by a PU should never exceed the expected number by more than a factor $1 + \mathcal{O}(f^{-1})$.

Both conditions become most critical at the end of Step 2, were each PU sends only k/f^2 packets. Now we want to apply the same algorithm, but without spreadings. Our new condition and choice of $f = f(N)$ are inspired by the above two demands:

$$\begin{aligned} k &= \omega(\max\{\ln N, k^*\}), \\ k &= f^6 \cdot \ln f \cdot \max\{\ln N, k^*\}. \end{aligned} \tag{7}$$

Here k^* only needs to satisfy (1): no sorting subroutines are applied. For such k the number of packets in a single PU is as large as in an $\ln N$ block before (we never used $k^* > 1$). This gives

Theorem 5 *If (7) holds, then*

$$T_{rank}(k, N) \leq (2 + \mathcal{O}(f^{-1})) \cdot T_{route}(k, N) + (3 + o(f^{-1})) \cdot T_{rank}(k/f, N).$$

Theorem 5 has important consequences:

Theorem 6 *There is a constant x such that, if $k \geq \log\log^x N \cdot \log N \cdot \max\{\ln N, k^*\}$, then*

$$T_{rank}(k, N) \leq (2 + o(1)) \cdot T_{route}(k, N).$$

Theorem 7 *Let $T_{rank}(k, N) = \mathcal{O}(T_{route}(k, N))$, for all $k \geq k^{**}$. For all $k = \omega(\max\{\ln N, k^*, k^{**}\})$,*

$$T_{rank}(k, N) \leq (2 + o(1)) \cdot T_{route}(k, N).$$

From [3] the following result can be derived:

Lemma 15 *For a hypercube with all-port communication, if* $k = \omega(\log N)$,

$$T_{route}(k, N) = (1 + o(1)) \cdot k.$$

Thus, $k^* = \omega(\log N)$. This implies $T_{\text{rank}}(k, N) = \mathcal{O}(T_{\text{route}}(k, N))$, for $k \geq \log^2 N$, and thus

Theorem 8 *For a hypercube with all-port communication, if* $k = \omega(\log^2 N)$,

$$T_{rank}(k, N) \leq (2 + o(1)) \cdot k.$$

6 Meshes

Meshes are slow in comparison with other architectures, but often the loss is partially gained back because 'local' and 'sparse' operations can be performed almost for free. The spreadings are local operations: for meshes (4) holds. The operation needed at the end of the algorithm to distribute the computed ranks of the rulers back to their subjects is sparse: only a small amount of information has to be made available. Generally, it is a problem that the subjects of a ruler may be scattered over the network. On meshes, however, the ranks of the rulers can be broadcast to suitable submeshes, and then a subject can 'look-up' its rank locally. In this way the total routing time is reduced by a factor of two and becomes optimal: $(1 + o(1)) \cdot T_{\text{route}}$.

For meshes, [10, 9] offer the first optimal deterministic k-k routing and sorting algorithms:

Lemma 16 *For a* d*-dimensional mesh, if* $k \geq 4 \cdot d$,

$$T_{route}(k, N), T_{sort}(k, N) = (1/2 + o(1)) \cdot k \cdot n.$$

Choice of Parameters. The algorithm as given is correct. But, for d-dimensional $n \times \cdots \times n$ meshes the spreadings can be performed in much larger blocks. This enables us to optimize the parameter choices. For k and f we assume

$$\begin{aligned} k &= 4 \cdot d \cdot f^{d+1}, \qquad (8) \\ f &= \omega(1). \end{aligned}$$

In Step 1, every non-initial node is selected as ruler with probability f^{-d}. In Step 2, the probability of selecting an initial node as pseudo-ruler is modified accordingly. The number of sending rounds becomes $f^d \cdot \ln f$. We assume that

$$k \leq n^{1/2 - 2 \cdot \epsilon}, \qquad (9)$$

$$\epsilon = 1/(2 \cdot d + 2). \qquad (10)$$

For larger k, good performance can be achieved with different choices. At the start and end of each sending round, the participating packets are spread in $n^a \times \cdots \times n^a$ submeshes, with

$$a = 1/2 + \epsilon. \qquad (11)$$

Step 1, ..., Step 5.

Lemma 17 *In total the spreadings take less than* $n^{-\epsilon} \cdot T_{route}(k, N)$ *steps.*

Lemma 18 *Let* k_r *be the expected number of packets sent by a PU in sending round* r, $1 \leq r \leq f^d \cdot \ln f$, *of Step 2. The number of packets sent by a PU in round* r *is bounded by*

$$k'_r = (1 + \ln^{1/2} n \cdot n^{-\epsilon}) \cdot k_r.$$

Lemma 19

$$\begin{aligned} T_2(k,N) &= (1+\ln^{1/2} n \cdot n^{-\epsilon}) \cdot T_{route}(k,N), \\ T_3(k,N) &= (1+\ln^{1/2} n \cdot n^{-\epsilon}) \cdot T_{rank}(k/f,N), \\ T_4(k,N) &= T_{route}(k/f^d+1,N), \\ T_5(k,N) &= T_{rank}(k/f^d+1,N). \end{aligned}$$

Alternative Step 6. We describe an alternative to Step 6, which runs in $\mathcal{O}(T_{\text{route}}(k/f,N))$.

The mesh is divided in $n/f \times \cdots \times n/f$ submeshes. After spreading, there are at most k/f^d+1 rulers in a PU. The rulers travel along all dimensions, and drop a copy in all positions that are shifted from their original positions by multiples of n/f.

Lemma 20 *Broadcasting the rulers to all submeshes takes $2 \cdot k \cdot n/(f \cdot d)$ steps. Afterwards every PU holds at most $k+f^d$ rulers.*

Now, in all submeshes the packets are sorted in a 'snake-like' order. Rulers get their index as key, subjects the index of their ruler $+ 1/2$.

Lemma 21 *In every submesh, rulers and subjects can be sorted together in $(2+f^{-1}) \cdot T_{sort}(k,N/f)$ steps. Afterwards, each ruler stands at the head of its subjects.*

The rulers now send their ranks 'along the snake' to their subjects. Those add their off-set, computed in Step 2 or 3, and are done.

Lemma 22 *Distributing the rank of a ruler to its subjects can be performed in $\log n$ steps.*

Summing the time consumptions of the substeps,

Lemma 23 $T_6(k,N) = (2+4/d+o(1)) \cdot T_{route}(k/f,N)$.

With Lemma 19, this gives our last main result:

Theorem 9 *For list ranking on d-dimensional meshes, with k satisfying (8),*

$$T_{rank}(k,N) \leq (1+\mathcal{O}(f^{-1})) \cdot T_{route}(k,N) = (1+\mathcal{O}(f^{-1})) \cdot k \cdot n/2.$$

7 Conclusion

We have given a complete analysis of the list ranking problem on networks. First we showed that the problem requires at least as many steps as routing. Then we gave a general algorithm, which uses only twice as many steps. A modification also achieves this for list ranking on hypercubes. For meshes we even match the lower bound up to a lower order term. We consider our algorithm to be an excellent candidate for actual implementation.

Two questions remain unanswered:

(1) Can the algorithm be made deterministic? We notice that the randomization in our algorithm is fundamentally different from the randomization in sorting algorithms. For sorting, a good sample can also be obtained by sorting subsets and taking regularly interspaced subsets thereof. The problem with list ranking is that there is no total order on the elements, and hence it is hard to deterministically select a small subset that more or less regularly subdivides the lists. Deterministic coin tossing [4] can be used to select an r-ruling set, but this mere selection is more expensive than our entire algorithm. In practice, taking the local minima of a simple deterministic coloring with $\log(k \cdot N)$ or with $\log\log(k \cdot N)$ colors (as it is applied in [1]) will work fine. But, there is no guarantee that the selected set of rulers is small, so along these lines no strong claims can be made.

(2) Can we get rid of the condition that the initial elements of the lists are known? This condition is required only if the average list length is constant. Even then it is not hard to modify the main algorithm such that the theorems in Section 4 and Section 5 still hold. The real problem is to achieve $T_{\text{rank}}(k,N) = (1+o(1)) \cdot T_{\text{route}}(k,N)$ for meshes. This implies that we should walk along most links only once. The trick with the rulers works fine in the case of long lists, but they are not useful for lists of length $\mathcal{O}(1)$.

References

[1] Anderson, R.J., G.L. Miller, 'Deterministic Parallel List Ranking,' *Algorithmica*, 6, pp. 859–868, 1991.

[2] Atallah, M.J., S.E. Hambrusch, 'Solving Tree Problems on a Mesh-Connected Processor Array,' *Information and Control*, 69, pp. 168–187, 1986.

[3] Chang, Y., J. Simon, 'Continuous Routing and Batch Routing on the Hypercube,' *Proc. 18th Symp. on Theory of Computing*, pp. 272–281, ACM, 1986.

[4] Cole, R., U. Vishkin, 'Deterministic Coin Tossing and Accelerated Cascades: Micro and Macro Techniques for Designing Parallel Algorithms,' *Proc. 18th Symp. on Theory of Computing*, pp. 206–219, ACM, 1986.

[5] Gibbons, A.M., Y. N. Srikant, 'A Class of Problems Efficiently Solvable on Mesh-Connected Computers Including Dynamic Expression Evaluation,' *Information Processing Letters*, 32, pp. 305–311, 1989.

[6] Hagerup, T., C. Rüb, 'A Guided Tour of Chernoff Bounds,' *Information Processing Letters*, 33, 305–308, 1990.

[7] Hwang, K., *Advanced Computer Architecture; Parallelism, Scalability, Programmability*, McGraw-Hill, Inc., 1993.

[8] JáJá, J., *An Introduction to Parallel Algorithms*, Addison-Wesley Publishing Company, Inc., 1992.

[9] Kaufmann, M., J.F. Sibeyn, T. Suel, 'Derandomizing Routing and Sorting Algorithms for Meshes,' *Proc. 5th Symp. on Discrete Algorithms*, pp. 669–679, ACM-SIAM, 1994.

[10] Kunde, M., 'Block Gossiping on Grids and Tori: Deterministic Sorting and Routing Match the Bisection Bound,' *Proc. European Symp. on Algorithms*, LNCS 726, pp. 272–283, Springer-Verlag, 1993.

[11] McDiarmid, C., 'On the Method of Bounded Differences,' in *Surveys in Combinatorics*, J. Siemons, editor, 1989 London Mathematical Society Lecture Note Series 141, pp. 148–188, Cambridge University Press, 1989.

[12] Reid-Miller, M., 'List Ranking and List Scan on the Cray C-90,' *Proc. 6th Symp. on Parallel Algorithms and Architectures*, pp. 104–113, ACM, 1994.

[13] Reif, J., L.G. Valiant, 'A logarithmic time sort for linear size networks,' *Journal of the ACM*, 34(1), pp. 68–76, 1987.

[14] Reischuk, R., 'Probabilistic Parallel Algorithms for Sorting and Selection,' *SIAM Journal of Computing*, 14, pp. 396–411, 1985.

[15] Ryu, K.W., J. JáJá, 'Efficient Algorithms for List Ranking and for Solving Graph Problems on the Hypercube,' *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 1, pp. 83–90, 1990.

[16] Sibeyn, J.F., 'List Ranking on Interconnection Networks,' *Techn. Rep. 11/1995, SFB 124-D6*, Universität Saarbrücken, Saarbrücken, Germany, 1995. Preliminary version in *Proc. Computing Science in the Netherlands*, pp. 271–280, SION, Amsterdam, 1994. Submitted to *Acta Informatica*.

[17] Valiant, L.G., 'A Bridging Model for Parallel Computation,' *Communications of the ACM*, 33(8), pp. 103–111, 1990.