Randomized Parallel List Ranking For Distributed Memory Multiprocessors *

Frank Dehne
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
dehne.at.scs.carleton.ca

Siang W. Song
Dept. of Computer Science - IME
University of São Paulo
05508-900 São Paulo, SP, Brazil
song.at.ime.usp.br

Abstract

We present a randomized parallel list ranking algorithm for distributed memory multiprocessors, using a BSP like model. We first describe a simple version which requires, with high probability, $\log(3p) + \log\ln(n) = \tilde{O}(\log p + \log\log n)$ communication rounds (h-relations with $h = \tilde{O}(\frac{n}{p})$) and $\tilde{O}(\frac{n}{p})$ local computation. We then outline an improved version which requires, with high probability, only $r \le (4k+6)\log(\frac{2}{3}p) + 8 = \tilde{O}(k\log p)$ communication rounds where $k = \min\{i \ge 0 | \ln^{(i+1)} n \le (\frac{2}{3}p)^{2i+1}\}$.

Note that $k < \ln^*(n)$ is an extremely small number. For $n \le 10^{10^{100}}$ and $p \ge 4$, the value of k is at most 2. Hence, for a given number of processors, p, the number of communication rounds required is, for all practical purposes, independent of n.

For $n \le 1,500,000$ and $4 \le p \le 2048$, the number of communication rounds in our algorithm is bounded, with high probability, by 78, but the actual number of communication rounds observed so far is 25 in the worst case. For $n \le 10^{10^{100}}$ and $4 \le p \le 2048$, the number of communication rounds in our algorithm is bounded, with high probability, by 118, and we conjecture that the actual number of communication rounds required will not exceed 50.

Our algorithm has a considerably smaller number of communication rounds than the list ranking algorithm used in Reid-Miller's empirical study of parallel list ranking on the Cray C-90 [21]. To our knowledge, [21] was the fastest list ranking implementation so far. Therefore, we expect that our result will have considerable practical relevance.

Key words: parallel algorithms, list ranking, coarse grained multicomputer.

*Research partially supported by NSERC (Natural Sciences and Engineering Research Council of Canada), FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo, Proc. No. 95/0767-0, 95/1367-5), CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico, Proc. No. 523112/94-7 and PROTEM II), and the Commission of the European Community (Project ITDC-207).

1 Introduction

The Model

Speedup results for theoretical PRAM algorithms do not necessarily match the speedups observed on real machines [3] [22]. Given sufficient slackness in the number of processors, Valiant's BSP approach [24] simulates PRAM algorithms optimally on distributed memory parallel systems. Valiant points out, however, that one may want to design algorithms that utilize local computations and minimize global operations [23] [24]. The BSP approach requires that g (= local computation speed / router bandwidth) is low, or fixed, even for increasing number of processors. Gerbessiotis and Valiant [14] describe circumstances where PRAM simulations can not be performed efficiently, among others if the factor g is high. Unfortunately, this is true for most currently available multiprocessors. The algorithm presented here considers this case for the list ranking problem. Furthermore, as pointed out in [24], the cost of a message also contains a constant overhead cost s. The value of s can be fairly large and the total message overhead cost can have a considerable impact on the speedup observed (see e.g. [8]).

We are therefore using a slightly enhanced version of the BSP model, referred to as *coarse grained multicomputer* model [8], [9], [10]. It is comprised of a set of p processors $P_1, \ldots, P_p$ with $O(n/p)$ local memory per processor and an arbitrary communication network (or shared memory). All algorithms consist of alternating local computation and global communication rounds. Each communication round consists of routing a single h-relation with $h = \tilde{O}(n/p)$ [1], i.e. each processor sends $\tilde{O}(n/p)$ data and receives $\tilde{O}(n/p)$ data. We require that all information sent from a given processor to another processor in one communication round is packed into one message. In the BSP model, a computation/communication round is equivalent to a superstep with $L = \frac{n}{p} g$ (plus the above "packing requirement").

Finding an optimal algorithm in the coarse grained multicomputer model is equivalent to minimizing the number of communication rounds as well as the total local computation time. This considers all parameters discussed above that are affecting the final observed speedup and it requires no assumption on g. Furthermore, it has been shown that minimizing the number of supersteps also leads to improved portability across different parallel architectures ([23] [24] [13]). The above model has been used (explicitly or implicitly) in parallel algorithm design for various problems ([6], [8], [9], [11], [12], [16], [10]) and shown very good practical timing results.

The List Ranking Problem

Consider a linear linked list consisting of a set S of n nodes and, for each node $x \in S$, a pointer $(x \rightarrow next(x))$ to its successor, $next(x)$, in the list. Let $\lambda \in S$ be the last list element and $next(\lambda) = \lambda$. The list ranking problem consist of computing for each $x \in S$ the distance of x to λ, referred to as $dist(x)$.

We assume that, initially, every processor stores n/p nodes and, for each of these nodes the pointer $(x \rightarrow next(x))$ to the next list element. See Figure 1. As output we require that every processor stores for each of its n/p nodes $x \in S$ the value $dist(x)$.

[1] $\tilde{O}(n)$ denotes $O(n)$ "with high probability". More precisely, $X = \tilde{O}(f(n))$, if and only if $(\forall c > c_0 > 1)$ $Prob\{X \geq cf(n)\} \leq \frac{1}{n^{g(c)}}$ where c_0 is a fixed constant and $g(c)$ is a polynomial in c with $g(c) \rightarrow \infty$ for $c \rightarrow \infty$ [19].

A trivial sequential algorithm solves the list ranking problem in optimal linear time by traversing the list. Several PRAM list ranking algorithms have been proposed [15] [20]. Wyllie [25] proposed a non-optimal $O(\log n)$ time algorithm with total work greater than O(n). The first optimal $O(\log n)$ EREW PRAM algorithm is due to Cole and Vishkin [7]. Another optimal deterministic algorithm is given by Anderson and Miller [2]. Parallel list ranking algorithms using randomization were proposed by Miller and Reif [17] [18]. The algorithms use $O(n)$ processors. The optimal algorithm by Anderson and Miller [1] improves this by using an optimal number of processors. A $O(\sqrt(n))$ time mesh algorithm is described in [4]. Reid-Miller [21] presented an empirical study for the Cray C-90 which will be discussed in the next subsection. See Section 6 for some of the many applications of list ranking

The Results

We present a randomized parallel list ranking algorithm for the coarse grained multicomputer model discussed above. We first describe a simple version which requires, with high probability, $\log(3p) + \log\ln(n) = \tilde{O}(\log p + \log\log n)$ communication rounds. Then, we outline an improved version which requires, with high probability, only $r \leq (4k+6)\log(\frac{2}{3}p) + 8$ $= \tilde{O}(k \log p)$ communication rounds where $k = \min\{i \geq 0 \,|\, \ln^{(i+1)} n \leq (\frac{2}{3}p)^{2i+1}\}$.

We observe that $k < \ln^*(n)$ is an extremely small number. For $n \leq 10^{10^{100}}$ and $p \geq 4$, the value of k is at most 2. That is, for a given number of processors, p, the number of communication rounds required is, for all practical purposes, independent of n.

For $n \leq 10^{10^{100}}$ and $4 \leq p \leq 2048$, the number of communication round, r, is bounded, with high probability, by 118. See Table 1. Note that, the above is only an upper bound on the number of communication rounds. For $100,000 \leq n \leq 1,500,000$ and $4 \leq p \leq 2048$, with high probability, r is bounded by 78 in the worst case. See Table 2. We simulated 100 test runs of our algorithm for each of the n, p combinations shown in Table 2. The observed numbers of communication rounds actually required where always much lower, and never exceeded 25.

For $n \leq 10^{10^{100}}$ and $4 \leq p \leq 2048$, the number of communication rounds in our algorithm is bounded, with high probability, by 118, and we conjecture that actual number of communication rounds required will not exceed 50.

Our randomization technique is very different from the ones used in [1, 17, 18]. In the above model, our algorithm uses considerably fewer communication rounds than [1, 2, 4, 5, 7, 15, 17, 18, 20, 21, 25].

The simple version of our algorithm is a generalization of the algorithm used in Reid-Miller's [21] empirical study of parallel list ranking for the Cray C-90 in shared memory mode. The analysis of our simple list ranking algorithm improves the estimates on the load imbalance provided in [21]. Our improved algorithm also applies to the Cray C-90. Since it requires significantly fewer communication rounds than the algorithm used in [21], we expect that our result will considerably improve the running times observed in [21]. To our knowledge, [21] was the fastest list ranking implementation so far. Therefore, we expect that our result will have considerable practical relevance.

As in [21] we will, in general, assume that $n >> p$ (coarse grained), because this is usually the case in practice. Note, however, that our results hold for arbitrary ratios $\frac{n}{p}$.

Overview

In the remainder of this paper, we will first prove a result on random sampling in linear linked lists. In Section 3 we will then outline the simple version of our algorithm which is based on a single random sampling of list nodes. In Section 4 we will introduce an incremental method to improve the first sample. We present a considerably improved list ranking algorithm, which is the main result of this paper. In Section 5 we discuss the results of our simulation of the improved list ranking algorithm and, finally, in Section 6, we outline some applications.

2 Random Sampling in Linear Linked Lists

Consider a linear linked list with a set S of n nodes. In this section we will show that if we select $\frac{n}{p}$ random elements (pivots) of S then, with high probability, these pivots will split S into sublists whose maximum size is bound by $3p\ln(n)$; see Figure 2.

We recall the following Lemma from [6] in a slightly modified form for linked lists (rather than for arrays).

Lemma 1 *$xk \leq n$ randomly chosen elements of S (pivots) partition list S into sublists S_i such that the size of the largest sublist is at most $\frac{n}{x}$ with probability at least*

$$1 - 2x(1 - \frac{1}{2x})^{xk}$$

Proof. (Analogous to [6]) Assume that the nodes of S are sorted by their rank. This sorted list can be viewed as $2x$ segments of size $\frac{n}{2x}$. If every segment contains at least one pivot (chosen element), then $\max_{1 \leq j \leq xk} |S_j| \leq \frac{n}{x}$. Consider one segment. Since the pivots are chosen randomly, the probability that a specific pivot is not in the segment is $(1 - \frac{1}{2x})$. Since xk pivots are selected independently, the probability that none of the pivots are in the segment is $(1 - \frac{1}{2x})^{xk}$. Therefore, even assuming mutual exclusion, the probability that there exists a segment which contains no pivot is at most $2x(1 - \frac{1}{2x})^{xk}$. Hence, every segment contains at least one pivot with probability at least $1 - 2x(1 - \frac{1}{2x})^{xk}$. ◻

Corollary 1 *$xk \leq n$ randomly chosen pivots partition list S into $xk+1$ sublists S_i such that there exists a sublist S_i of size larger than $c\frac{n}{x}$ with probability at most $\frac{2x}{c}(1 - \frac{c}{2x})^{xk} \leq \frac{2x}{c} e^{-\frac{1}{2}ck}$.*

Lemma 2 *Consider $xk \leq n$ randomly chosen pivots which partition S into $xk + 1$ sublists S_i, and let $m = \max_{0 \leq i \leq xk} |S_i|$. If $k \geq \ln(x) + 2\ln(n)$ then Prob$\{m > c\frac{n}{x}\} \leq \frac{1}{n^c}$, $c > 2$.*

Proof. Corollary 1 implies that

$$\text{Prob}\{m > c\frac{n}{x}\} \leq \frac{2x}{c} \mathrm{e}^{-\frac{1}{2}ck}$$

We observe that, for $c > 2$,

$$\ln(x) + 2\ln(n) \leq k$$

$$\Rightarrow \frac{2}{c}\ln(\frac{2x}{c}) + 2\ln(n) \leq k$$

$$\Rightarrow \ln(\frac{2x}{c}) + c\ln(n) \leq \frac{ck}{2}$$

$$\Rightarrow \frac{2x}{c} n^c \leq e^{\frac{ck}{2}}$$

$$\Rightarrow \text{Prob}\{m > c\frac{n}{x}\} \leq n^{-c}$$

□

Theorem 1 *$\frac{n}{p}$ randomly chosen pivots partition S into $\frac{n}{p}+1$ sublists S_j with $m = \max_{0 \leq j \leq p} |S_j|$ such that*

$$Prob\{m \geq c3p\ln(n)\} \leq \frac{1}{n^c}, c > 2$$

Proof. Let $x = \frac{n}{3p\ln(n)}$, $k = \ln(x) + 2\ln(n) = 3\ln(n) - \ln(3p\ln(n))$.

Then $xk = \frac{n}{p}\frac{3\ln(n)-\ln(3p\ln(n))}{3\ln(n)} \leq \frac{n}{p}$, and Theorem 1 follows from Lemma 2. □

3 A Simple Algorithm Using A Single Random Sample

In this section we will present a simple list ranking algorithm which requires, with high probability, at most $\log(3p) + \log\ln(n) = \tilde{O}(\log p + \log\log n)$ communication rounds. This algorithm is based on a single random sample of nodes. We will later improve the performance of the algorithm by improving the sample through a sequence of sampling rounds.

Consider a random set $S' \subset S$ of pivots. For each $x \in S$ let $nextPivot(x, S')$ refer to the closest pivot following x in the list S. (W.l.o.g. assume that the last element, λ, of S is selected as a pivot and let $nextPivot(\lambda, S') = \lambda$. Note that for $x \neq \lambda$, $nextPivot(x, S') \neq x$.) Let $distToPivot(x, S')$ be the distance between x and $nextPivot(x, S')$ in list S. Furthermore, let $m(S, S') = \max_{x\in S} distToPivot(x, S')$.

The *modified list ranking problem* for S with respect to S' refers to the problem of determining for each $x \in S$ its next pivot $nextPivot(x, S')$ as well as the distance $distToPivot(x, S')$. The input/output structure for the modified list ranking problem is the same as for the list ranking problem.

Algorithm 1

(1) Select a set $S' \subset S$ of $\tilde{O}(\frac{n}{p})$ random pivots as follows: Every processor P_i makes for each $x \in S$ stored at P_i an independent biased coin flip which selects x as a pivot with probability $\frac{1}{p}$.

(2) All processors solve collectively the *modified list ranking problem* for S with respect to S' (details will be discussed later).

(3) Using an all-to-all broadcast, the values $nextPivot(x, S')$ and $distToPivot(x, S')$ for all pivots $x \in S'$ are broadcast to all processors.

(4) Using the data received in Step 3, each processor P_i can solve the list ranking problem for the nodes stored at P_i sequentially in time $\tilde{O}(\frac{n}{p})$.

— End of Algorithm —

For the correctness of Step 1, we recall the following

Lemma 3 *[19] Consider a random variable X with binomial distribution. Let n be the number of trials, each of which is successful with probability q. The expectation of X is $E(X) = nq$ and*

$$Prob\{X > cnq\} \leq e^{-\frac{1}{2}(c-1)^2 nq}, \text{for any } c > 1$$

In order to implement Step 2, we simply simulate the standard recursive doubling technique. (For all x in parallel: WHILE $next(x) \neq nextPivot(x, S')$ DO $next(x) := next(next(x))$.) From Theorem 1 it follows that, with high probability, $m(S, S') \leq 3p \ln(n)$. Hence, Step 2 requires, with high probability, at most $\log(3p \ln(n)) = \log(3p) + \log\ln(n)$ communication rounds. Step 3 requires 1 communication round, and Step 4 is straightforward. In summary, we obtain

Theorem 2 *Algorithm 1 solves the list ranking problem using, with high probability, at most $1 + \log(3p) + \log\ln(n)$ communication rounds and $\tilde{O}(\frac{n}{p})$ local computation.*

We observe that, if $\frac{n}{p} \leq \mathrm{e}^{(3p)^\alpha}$ for some $\alpha > 1$ then,

$$\ln(n) \leq \ln(p) + (3p)^\alpha$$

$$\Rightarrow \log\ln(n) \leq \log(\ln(p) + (3p)^\alpha) \leq \log(2(3p)^\alpha)$$

$$\Rightarrow \log\ln(n) \leq 1 + \alpha \log(3p)$$

$$\Rightarrow \log(3p) + \log\ln(n) \leq 1 + (\alpha + 1)\log(3p)$$

This implies

Corollary 2 *If $\frac{n}{p} \leq e^{(3p)^\alpha}$, for some constant $\alpha > 1$, then the number of communication rounds required by Algorithm 1 is bounded by $2 + (\alpha + 1)\log(3p) = \tilde{O}(\log p)$.*

4 Improving The Maximum Sublist Size

We will now present an algorithm which improves the maximum sublist size obtained in Algorithm 1 and solves the list ranking problem by using, with high probability, only $r \leq (4k + 6)\log(\frac{2}{3}p) + 8$ communication rounds and $\tilde{O}(\frac{n}{p})$ local computation where

$$k := \min\{i \geq 0 | \ln^{(i+1)} n \leq (\frac{2}{3}p)^{2i+1}\}.$$

Note that $k < \ln^*(n)$ is an extremely small number (see Table 1). Figure 3 illustrates $\ln^{(i+1)} n$ and $(\frac{2}{3}p)^{2i+1}$ as functions of i, as well as their intersection point k.

The basic idea of the algorithm is that any two pivots should not be closer than $O(p)$ because this creates large "gaps" elsewhere in the list. If two pivots are closer than $O(p)$, then one of them is "useless" and should be "relocated". The non-trivial part is to perform the "relocation" without too much overhead and such that the new set of pivots has a considerably better distribution. The algorithm uses three colors to mark nodes: *black* (pivot), *red* (a node close to a pivot), and *white* (all other nodes).

Algorithm 2

(1) Perform Step 1 of Algorithm 1. Mark all selected pivots *black* and all other nodes *white.*

(2) For $i = 1, \ldots, k$ do

(2a) For each *black* node x, all nodes which are to the right of x (in list S) and have distance at most $\frac{2}{3}p$ are marked *red.* Note: previously *black* nodes (pivots) that are now marked *red* are no longer considered pivots.

(2b) For each *black* node x, all nodes which are to the left of x (in list S) and have distance at most $\frac{2}{3}p$ are marked *red.*

(2c) Every processor P_i makes for each *white* node $x \in S$ stored at P_i an independent biased coin flip which selects x as a new pivot, and marks it *black*, with probability $\frac{1}{p}$.

(2d) Every processor P_i marks *white* every *red* node $x \in S$ stored at P_i.

(3) Let $S' \in S$ be the subset of *black* nodes obtained after Step 2. Continue with Steps 2 – 4 of Algorithm 1.

— End of Algorithm —

Observe that Steps 2a and 2b have to be performed in a left-to-right scan, respectively, as if executed sequentially. We can simulate this sequential scanning process in the parallel setting because the number of pivots is bounded by n/p. For Step 2a, we build linked lists of pivots by computing for each of them a pointer to the next pivot of distance at most 2 $p/3$, if any, and the distance. These linked lists of pivots are compressed into one processor and we run on these lists a sequential left-to-right scan to mark pivots red. We return the pivots to their original location and mark every non-pivot red for which there exists a non-red pivot that attempts to mark it red. Step 2b is performed analogously. Note that each node x requires a pointer to its predecessor $prev(x)$ in the linked list. All $prev(x)$ values can be easily computed with one communication round and $O(\frac{n}{p}$ local computation.

Let r be the number of communication rounds required by Algorithm 2. We will now show that, with high probability,

$$r \leq (4k+6)\log(\frac{2}{3}p) + 8 = \tilde{O}(k \log p).$$

Let n_i be the maximum length of a contiguous sequence of *white* nodes after the i^{th} execution of Step 2b, and define $n_0 = n$.

Let S_i be the set of *black* nodes after the i^{th} execution of Step 2c, $1 \leq i \leq k$, and let S_0 be the set of *black* nodes after the execution of Step 1. Note that, in Step 3, $S' = S_k$. Define $m_i = m(S_i)$ for $0 \leq i \leq k$.

Lemma 4 *With high probability, the following holds:*
(a) $n_0 = n$ *and* $n_i \leq 3p\ln(n_{i-1}), 1 \leq i \leq k$
(b) $m_i \leq 3p\ln(n_i), 0 \leq i \leq k$

Proof. It follows from Theorem 1 that, with high probability,

$$\begin{aligned} n_0 &= n \\ m_0 &\leq 3p\ln(n) \end{aligned}$$

and, for a fixed $1 \leq i \leq k$

$$\begin{aligned} n_i &\leq m_{i-1} \\ m_i &\leq 3p\ln(n_i). \end{aligned}$$

Since $k \leq \ln^*(n)$ and $\log^*(n)\frac{1}{n^c} \leq \frac{1}{n^{c-\epsilon}}, \epsilon > 0$, the above bounds for n_i and m_i hold, with high probability, for all $1 \leq i \leq k$. ◻

Lemma 5 *With high probability, for all $1 \leq i \leq k$,*
(a) $n_i \leq 3p(2\ln(3p) + \ln^{(i)}(n))$
(b) $m_i \leq 6p\ln(3p) + 3p\ln^{(i+1)}(n)$

Proof.
(a) Applying Lemma 4 we observe that

$$\begin{aligned} n_1 &\leq 3p\ln(n) \\ n_2 &\leq 3p\ln(3p\ln(n)) \\ &= 3p(\ln(3p) + \ln\ln(n)) \\ n_3 &\leq 3p\ln(n_2) \\ &\leq 3p(\ln(3p) + \ln(\ln(3p) + \ln\ln(n))) \\ &\leq 3p(\ln(3p) + \ln\ln(3p) + \ln\ln\ln(n)) \\ n_4 &\leq 3p\ln(n_3) \\ &\leq 3p(\ln(3p) + \ln\ln(3p) + \ln\ln\ln(3p) + \ln\ln\ln\ln(n)) \\ &\vdots \\ n_i &\leq 3p(2\ln(3p) + \ln^{(i)}(n)) \end{aligned}$$

(b) It follows from Lemma 4 that $m_i \leq 3p\ln(n_i) \leq 3p\ln(3p(2\ln(3p)+\ln^{(i)}(n))) \leq 3p(\ln(3p)+\ln(2) + ln^{(2)}(3p) + ln^{(i+1)}(n)) \leq 6p\ln(3p) + 3p\ln^{(i+1)}(n)$. ◻

Theorem 3 *With high probability, Algorithm 2 solves the list ranking problem with $r \leq (4k+6)\log(\frac{2}{3}p) + 8 = \tilde{O}(k\log p)$ communication rounds and $\tilde{O}(\frac{n}{p})$ local computation.*

Proof. With high probability, the total number of communication rounds in Algorithm 2 is bounded by

$$\begin{aligned} & 2k\log(\tfrac{2}{3}p) + \log(m_k) + 1 \\ \leq\ & 2k\log(\tfrac{2}{3}p) + \log(6p) + \log\ln(3p) + \log(3p) + \log\ln^{(k+1)}(n) + 1 \end{aligned}$$

$$\begin{aligned}
&\leq (2k+3)\log(\tfrac{2}{3}p) + \log 9 + \log 4.5 + \log \ln^{(k+1)}(n) + 1 \\
&\leq (2k+3)\log(\tfrac{2}{3}p) + \log \ln^{(k+1)}(n) + 8 \\
&\leq \log((\tfrac{2}{3}p)^{2k+3}) + \log \ln^{(k+1)}(n) + 8 \\
&\leq 2\log((\tfrac{2}{3}p)^{2k+3}) + 8 \text{ if } (*) \ln^{(k+1)}(n) \leq (\tfrac{2}{3}p)^{2k+3} \\
&\leq (4k+6)\log(\tfrac{2}{3}p) + 8 = \tilde{O}(k \log p)
\end{aligned}$$

Condition (*) is true because we selected $k = \min\{i \geq 0 | \ln^{(i+1)} n \leq (\frac{2}{3}p)^{2i+1}\}$. Note that, this bound is not tight. □

5 Simulation Results

We simulated the behaviour of Algorithm 2. In particular, we simulated how our above method improves the sample by reducing the maximum distance, m_i, between subsequent pivots. We examined the range of $4 \leq p \leq 2048$ and $100,000 \leq n \leq 1,500,000$ as shown in Table 2 and applied Algorithm 2 for each n, p combination shown 100 times with different random samples. Table 2 shows the values of k and the upper bound R on the number of communication rounds required according to Theorem 3. We then measured the maximum distance, m_k^{obs}, observed between two subsequent pivots in the sample chosen at the end of the algorithm, as well as the number, r^{obs}, of communication rounds actually required. Each of the numbers shown is the worst case observed in the respective 100 test runs.

According to Theorem 3, for the range of test data used, the number of communication rounds in our algorithm should not exceed 78. This is an upper bound, though. The actual number of communication rounds observed in Table 2 is 25 in the worst case. The number of rounds observed is usually around 30% of the upper bound according to Theorem 3. We also observe that for a given p (i.e. in a vertical column), the values of m_k^{obs} and r^{obs} are essentially stable and show no monotone increase or decrease with increasing n.

6 Applications

The problem of list ranking is a special case of computing the suffix sums of the elements of a linked list. The above algorithm can obviously be generalized to compute prefix or suffix sums for associative operators (by replacing the addition operation for node distances by the respective associative operator). List ranking is a very popular tool for obtaining numerous parallel tree and graph algorithms [4] [5] [21].

An important application outlined in [4] is to use list ranking for applying Euler tour techniques to tree problems. As demonstrated in [4], once an efficient distributed memory parallel list ranking algorithm is available, it is easy to obtain efficient distributed memory parallel algorithms for the following problems for an undirected forest of trees: rooting every tree at a given vertex chosen as root, determining the parent of each vertex in the rooted forest, computing the preorder (or postorder) traversal of the forest, computing the level of each vertex, and computing the number of descendants of each vertex. All these problems can be easily solved with one or a small constant number of list ranking operations.

7 Conclusion

We presented a randomized parallel list ranking algorithm for distributed memory multiprocessors, using the coarse grained multicomputer model. The algorithm requires, with high probability, $r \leq (4k+6)\log(\frac{2}{3}p)+8 = \tilde{O}(k \log p)$ communication rounds. For all practical purposes, $k \leq 2$. The algorithm presented improves on the number of communication rounds required in Reid-Miller's [21] list ranking implementation for the Cray C-90 which was, to our knowledge, the fastest list ranking implementation to date. Therefore, we expect that our result will have considerable practical relevance.

References

[1] J. R. Anderson and G. L. Miller, "A simple randomized parallel algorithm for list ranking". *Information Processing Letters*, Vol. 33, No. 5, January 1990, pp. 269 – 273.

[2] J. R. Anderson and G. L. Miller, "Deterministic parallel list ranking", J. H. Reif (editor), *VLSI Algorithms and Architectures: 3rd Aegean Workshop on Computing, AWOC'88*, Springer Verlag, Lecture Notes in Computer Science, Vol. 319, 1988, pp. 81 –90.

[3] R.J. Anderson, and L. Snyder, "A Comparison of Shared and Nonshared Memory Models of Computation," in Proc. of the IEEE, 79(4), pp. 480-487.

[4] M. J. Atallah, S. E. Hambrusch, "Solving tree problems on a mesh-connected processor array," *Information and Control*, Vol. 69, 1986, pp. 168-187.

[5] S. Baase, "Introduction to parallel connectivity, list ranking, and Euler tour techniques". J. H. Reif (ed.) *Synthesis of Parallel Algorithms*. Morgan Kaufmann Publisher, 1993.

[6] G.E. Blelloch, C.E. Leiserson, B.M. Maggs, C.G. Plaxton, "A Comparison of Sorting Algorithms for the Connection Machine CM-2.," in Proc. ACM Symp. on Parallel Algorithms and Architectures, 1991, pp. 3-16.

[7] 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*, Vol. 17, No. 1, 1988, pp. 128 – 142.

[8] F. Dehne, A. Fabri, and A. Rau-Chaplin, "Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers," in Proc. ACM Symp. Computational Geometry, 1993, pp. 298-307.

[9] F. Dehne, A. Fabri, and C. Kenyon, "Scalable and Architecture Independent Parallel Geometric Algorithms with High Probability Optimal Time," in Proc. 6th IEEE Symposium on Parallel and Distributed Processing, 1994, pp. 586-593.

[10] F. Dehne, X. Deng, P. Dymond, A Fabri, A. A. Kokhar, "A randomized parallel 3D convex hull algorithm for coarse grained parallel multicomputers," in Proc. ACM Symp. on Parallel Algorithms and Architectures, 1995.

[11] X. Deng and N. Gu, “Good Programming Style on Multiprocessors,” in Proc. IEEE Symposium on Parallel and Distributed Processing, 1994, pp. 538-543.

[12] X. Deng, “A Convex Hull Algorithm for Coarse Grained Multiprocessors,” in Proc. 5th International Symposium on Algorithms and Computation, 1994.

[13] X. Deng and P. Dymond, “Efficient Routing and Message Bounds for Optimal Parallel Algorithms,” in Proc. Int. Parallel Proc. Symp., 1995.

[14] A.V. Gerbessiotis and L.G. Valiant, “Direct Bulk-Synchronous Parallel Algorithms,” in Proc. 3rd Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science, Vol. 621, 1992, pp. 1-18.

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

[16] Hui Li, and K. C. Sevcik, “Parallel Sorting by Overpartitioning,” in Proc. ACM Symp. on Parallel Algorithms and Architectures, 1994, pp. 46-56.

[17] G. L. Miller and J. H. Reif, “Parallel tree contraction part 1: Fundamentals”. *Advances in Computing Research*, Vol. 5, 1989, pp. 47 –72.

[18] G. L. Miller and J. H. Reif, “Parallel tree contraction part 1: Further applications”. *SIAM J. Computing*, Vol. 20, No. 6, December 1991, pp. 1128 – 1147.

[19] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice Hall, New York, NY, 1993.

[20] M. Reid-Miller, C. L. Miller, F. Modugno, “List ranking and parallel tree compaction”. J. H. Reif (ed.) *Synthesis of Parallel Algorithms.* Morgan Kaufmann Publisher, 1993.

[21] M. Reid-Miller, “List ranking and list scan on the Cray C-90,” in Proc. ACM Symp. on Parallel Algorithms and Architectures, 1994, pp. 104-113.

[22] L. Snyder, ‘’Type architectures, shared memory and the corollary of modest potential,” *Annu. Rev. Comput. Sci. 1*, 1986, pp. 289-317.

[23] L.G. Valiant, “A Bridging Model for Parallel Computation,” *Communications of the ACM*, 33, 1990, pp. 103–111.

[24] L.G. Valiant *et. al.*, “General Purpose Parallel Architectures,” *Handbook of Theoretical Computer Science*, Edited by J. van Leeuwen, MIT Press/Elsevier, 1990, pp.943-972.

[25] J. C. Wyllie, “The complexity of parallel computation”, Technical Report TR 79-387, Department of Computer Science, Cornell University, 1979.

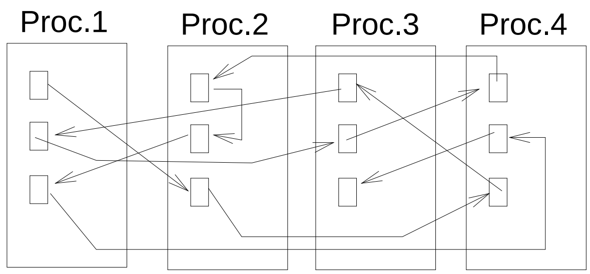

Figure 1: A Linear Linked List Stored In A Distributed Memory Multiprocessor

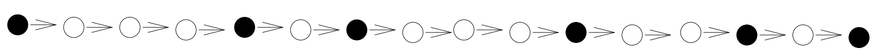

Figure 2: A Linear Linked List With Random Pivots

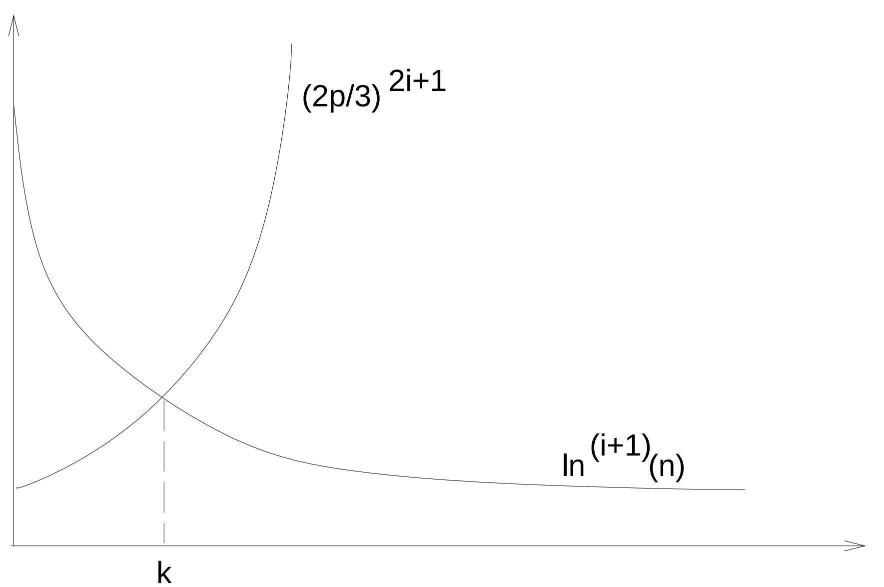

Figure 3: $\ln^{(i+1)} n$ and $(\frac{2}{3}p)^{2i+1}$ As Functions Of i, And Their Intersection Point k.

$p =$	4	8	16	32	64	128	256	512	1024	2048
n	$k; R$	$k; R$	$k; R$	$k; R$	$k; R$	$k; R$	$k; R$	$k; R$	$k; R$	$k; R$
10^{10}	1;18	0;26	0;32	0;38	0;44	0;50	0;56	0;62	0;68	0;74
10^{100}	1;18	1;38	0;32	0;38	0;44	0;50	0;56	0;62	0;68	0;74
10^{1000}	1;18	1;38	1;48	0;38	0;44	0;50	0;56	0;62	0;68	0;74
$10^{(10^4)}$	1;18	1;38	1;48	1;58	0;44	0;50	0;56	0;62	0;68	0;74
$10^{(10^5)}$	1;18	1;38	1;48	1;58	1;68	0;50	0;56	0;62	0;68	0;74
$10^{(10^6)}$	1;18	1;38	1;48	1;58	1;68	1;78	0;56	0;62	0;68	0;74
$10^{(10^7)}$	1;18	1;38	1;48	1;58	1;68	1;78	1;88	0;62	0;68	0;74
$10^{(10^8)}$	1;18	1;38	1;48	1;58	1;68	1;78	1;88	1;98	0;68	0;74
$10^{(10^9)}$	1;18	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	0;74
$10^{(10^{10})}$	1;18	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{11})}$	1;18	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{12})}$	1;18	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{13})}$	1;18	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{14})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{15})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{16})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{17})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{18})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{19})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{20})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{30})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{40})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{50})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{60})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{70})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{80})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{90})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118
$10^{(10^{100})}$	2;22	1;38	1;48	1;58	1;68	1;78	1;88	1;98	1;108	1;118

Table 1: Values Of k and $R := (4k + 6)\log(\frac{2}{3}p) + 8$ [Upper Bound On r] For Various Combinations Of n And p.

$p =$	4	8	16	32	64	128	256	512	1024	2048
n	k	k	k	k	k	k	k	k	k	k
	R	R	R	R	R	R	R	R	R	R
	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}
	r^{obs}	r^{obs}	r^{obs}	r^{obs}	r^{obs}	r^{obs}	r^{obs}	r^{obs}	r^{obs}	r^{obs}
100,000	1	1	1	1	1	0	0	0	0	0
	18	38	48	58	68	50	56	62	68	74
	28	59	119	238	409	1400	2421	5900	9136	17158
	8	13	16	19	22	12	13	14	15	16
200,000	1	1	1	1	1	0	0	0	0	0
	18	38	48	58	68	50	56	62	68	74
	35	72	127	283	444	1690	3023	5447	11047	17921
	9	14	16	20	22	12	13	14	15	16
300,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	31	65	130	235	468	859	3038	5076	9432	19636
	8	14	17	19	22	25	13	14	15	16
400,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	35	75	134	226	441	925	3497	6394	11627	17252
	9	14	17	19	22	25	13	14	15	16
500,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	32	72	117	264	474	860	3150	6144	11179	21552
	8	14	16	20	22	25	13	14	15	16
600,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	33	78	132	246	458	1015	2934	6295	11409	26526
	9	14	17	19	22	25	13	14	15	16
700,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	32	69	122	244	467	882	3420	6605	11622	21028
	8	14	16	19	22	25	13	14	15	16

Table 2: k, $R := (4k+6)\log(\frac{2}{3}p) + 8$, m_k^{obs} and r^{obs} For Various Combinations of n and p, Where m_k^{obs} and r^{obs} Are The Observed *Worst Case* Values Of m_k and r, Respectively. (For each shown combination of n and p, the m_k^{obs} and r^{obs} shown are the worst case values observed during 100 test runs.)

$p =$	4	8	16	32	64	128	256	512	1024	2048
n	k	k	k	k	k	k	k	k	k	k
	R	R	R	R	R	R	R	R	R	R
	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}	m_k^{obs}
	r^{obs}	r^{obs}	r^{obs}	r^{obs}	r^{obs}	r^{obs}	r^{obs}	r^{obs}	r^{obs}	r^{obs}
800,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	35	79	147	260	510	989	4216	5905	11098	28814
	9	14	17	20	22	25	14	14	15	16
900,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	33	76	132	240	536	887	3023	6909	12244	24516
	9	14	17	19	23	25	13	14	15	16
1,000,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	40	69	127	264	440	851	3406	7924	11861	21552
	9	14	16	20	22	25	13	14	15	16
1,100,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	38	83	136	241	531	996	3469	6120	11938	23631
	9	14	17	19	23	25	13	14	15	16
1,200,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	36	75	134	279	510	974	3412	6394	11627	22720
	9	14	17	20	22	25	13	14	15	16
1,300,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	33	76	133	254	605	1011	4216	6390	11258	20613
	9	14	17	19	23	25	14	14	15	16
1,400,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	32	70	141	259	605	924	3722	6394	11627	22720
	8	14	17	20	23	25	13	14	15	16
1,500,000	1	1	1	1	1	1	0	0	0	0
	18	38	48	58	68	78	56	62	68	74
	33	89	172	270	551	903	3893	6120	11938	23631
	9	14	17	20	23	25	13	14	15	16

Table 3: Continuation of Table 2.