Tight Bounds on Parallel List Marking*

Sandeep N. Bhatt
Bell Communications Research
Morristown, NJ 07960 USA

Gianfranco Bilardi
Dipartimento di Elettronica e Informatica
Università di Padova
I-35131 Padova, Italy
and
Department of Electrical Engineering and Computer Science
University of Illinois at Chicago
Chicago, IL 60607 USA

Kieran T. Herley
Department of Computer Science
University College Cork
Cork, Ireland

Geppino Pucci
Dipartimento di Elettronica e Informatica
Università di Padova
I-35131 Padova, Italy

Abhiram Ranade
Department of Computer Science and Engineering
Indian Institute of Technology
Powai, Mumbai 400076 India

*A preliminary version of this paper appeared in *Proc. EURO-PAR'95 Parallel Processing*, Springer LNCS 966, Stockolm, S, 1995, pp. 231–242.

Proposed Running Head

Tight Bounds on Parallel List Marking

Contact Author

Dr. **Geppino Pucci**
Dipartimento di Elettronica e Informatica
Università di Padova
Via Gradenigo, 6/A
I-35131 Padova, Italy

Phone: +39 (49) 827 7830
Fax: +39 (49) 827 7826
E-mail: geppo@artemide.dei.unipd.it

Abstract

The list marking problem involves marking the nodes of an ℓ-node linked list stored in the memory of a (p, n)-PRAM, when only the position of the head of the list is initially known, while the remaining list nodes are stored in arbitrary memory locations. Under the assumption that cells containing list nodes bear no distinctive tags distinguishing them from other cells, we establish an $\Omega\left(\min\{\ell, n/p\}\right)$ randomized lower bound for ℓ-node lists and present a deterministic algorithm whose running time is within a logarithmic additive term of this bound. Such result implies that randomization cannot be exploited in any significant way in this setting. For the case where list cells are tagged in a way that differentiates them from other cells, the above lower bound still applies to deterministic algorithms, while we establish a tight $\Theta\left(\min\left\{\ell, \ell/p + \sqrt{(n/p)\log n}\right\}\right)$ bound for randomized algorithms. Therefore, in the latter case, randomization yields better performance for a wide range of parameter values.

Key words: List Marking; List Ranking; Linked Structures; Shared-Memory Machines; Parallel Algorithms; Randomized Algorithms; Lower Bounds.

List of Symbols

ℓ	Letter ell
$\langle, \rangle$	Angled Brackets
δ, μ	Lowercase greek letters (delta, mu)
Ω, Θ	Uppercase greek letters (Omega, Theta)
O, o	Big-oh, little-oh
Pr	Probability

1 Introduction

Linked structures are widely used in non numerical as well as sparse numerical computations. Therefore, it is important to ascertain whether parallelism can be exploited to process such structures effectively.

In this paper, we focus on lists, possibly the simplest type of linked structures, and on a very basic operation, which we call *marking*, consisting of writing a given value in each node of a given list. The essence of marking is that each node in the list has to be affected, while other structures stored in the same memory have to stay untouched. This feature is instrumental to the realization of several basic list operations such as searching an element or ranking all nodes (determining their distance from the head). Marking itself is used in important practical applications, such as garbage collection, for identifying active structures in a large memory heap. The complexity of parallel list operations crucially depends on the list representation, and is often affected by features that are irrelevant to sequential complexity. When managing lists in parallel, a favourable case arises if the the growth process affords keeping all list nodes in a compact region of memory. In this case the list can be represented as an array of ℓ records, each record corresponding to a list node, with a field storing the array index of its successor. Indeed, most list-based parallel algorithms in the literature (*e.g.*, searching and ranking [2]) do assume such compact representation. In other scenarios, unfortunately, list nodes become naturally scattered throughout a portion of memory whose size n is much larger than the length of the list. This case arises, for instance, when a sequence of concatenations and splittings is performed on a set of lists. In the present study, we consider the size ℓ of the list and the size n of the memory region

which is known *a priori* to contain the list as independent parameters.

We also distinguish between *tagged* and *untagged* lists, a tagged list being one where each node contains a *tag* that uniquely identifies the list. Tags can be maintained with small overhead if the list is modified only by insertion and deletion of nodes. However, the overhead is not negligible if other operations, such as concatenation and splitting, are allowed. Note that a node of a tagged list can be recognized as such by simply checking the tag stored with the node. In contrast, cells storing untagged list nodes are indistinguishable from other cells by simple inspection.

We investigate the extent to which parallelism, randomization, and tagging can be profitably exploited to improve upon sequential performance when lists are scattered throughout the memory. Specifically, we develop deterministic and randomized upper and lower bounds for marking a tagged or untagged list of ℓ nodes stored in the memory of a p-processor PRAM with n memory cells, when only the position of the head of the list is initially known and the remaining list nodes are scattered among arbitrary memory locations.

A restricted version of the list marking problem was introduced and analyzed by Luccio and Pagli in [7]. The authors prove a deterministic $\Omega\left(\min\{\ell, n/p\}\right)$ lower bound for marking tagged lists and provide a tight upper bound when $p = O(\ell/\log\ell)$ and $n = O(\ell\log\ell)$. In this paper, after formalizing the problem in Section 2, we improve and generalize these results in the following directions:

1. In Section 3, we prove that an $\Omega\left(\min\{\ell, n/p\}\right)$ lower bound also holds for any randomized algorithm for marking an untagged list. Moreover, we give a deterministic algorithm optimal to within a logarithmic additive term, therefore showing that ran-

domization can not be exploited in any significant way in this setting.

2. In Section 4, we establish a tight, randomized $\Theta\left(\min\left\{\ell, \ell/p + \sqrt{(n/p)\log n}\right\}\right)$ bound for marking a tagged list, showing that, for a wide range of list lengths, considerable speedups can be attained by means of randomization.

A synopsis of results for the marking problem is given in Section 5.

2 Problem Formulation

We will assume that each memory cell has the same format and contains a memory address which will be interpreted as a pointer (called the *successor pointer*) to another cell, a *tag field*, capable of holding a distinctive symbol, a *data field*, and a small constant amount of additional space, called *scratch space*. The head of the list, denoted by h, occupies cell 0 and its data field contains some arbitrary symbol which we will refer to as the *signature* of the list. Finally, each list node points to its successor in the list, and the pointer field of the last node r contains the address of cell 0, which we will interpret as a *nil* pointer.

We identify two variants of the problem. The list is *untagged* if list nodes bear no distinctive mark or symbol that renders them instantly identifiable as such. The list is *tagged* if each list node bears a distinctive symbol in its tag field which no non-list node bears, thus allowing list nodes to be identified by inspection. In both cases, the goal of the list marking problem is to copy the signature into the data field of every node in the list; the data fields of all other nodes should remain unchanged. A node is said to be *marked* once its data field bears the appropriate signature.

Since each memory cell contains a successor pointer, the entire memory can be interpreted as a directed graph G of n nodes. Note that each node has outdegree zero or one, but a node (including list nodes) may have indegree zero (*leaves*), one (*unary nodes*), or higher. Such a graph is known as a *pseudoforest*. (These structures feature in some connected component algorithms, see for example [5].) Within the pseudoforest, the chain of list nodes forms a directed path in a structure T that we may interpret as a tree, the edges of which are oriented from child to parent. The node h is a leaf of T and the list nodes are the ancestors of h in T located along the directed path from h to r, the root of T. In fact, G consists of T plus a collection of node-disjoint components each of which is either a tree or one or more trees joined by a cycle connecting their roots (see Figure 1 for an example). In this setting, the objective is to mark all those nodes in G that are ancestors of h in T.

The algorithms and the lower bound arguments presented in the paper all assume the ARBITRARY CRCW variant of the PRAM model of shared-memory computation [5]. Thus, concurrent reads and writes are permitted. Whenever a number of processors attempt to write simultaneously to a cell, one of them, chosen arbitrarily, succeeds, while the others fail. For convenience, we will refer to a PRAM with p processors and n cells of memory as a (p, n)-PRAM and will assume throughout that $p \leq n$. We will also assume that each processor has a private area of $O(1)$ storage for workspace.

We want to remark that the lower bound arguments on list marking apply to the less powerful EREW and CREW variants of the PRAM model, while the algorithms presented in the paper can be ported to these weaker models with an $O(\log p)$ extra factor in their running time. In fact, for many choices of the relevant parameters, the slowdown incurred in

running either the deterministic or the randomized marking algorithm on the EREW PRAM can be reduced to a constant factor. The technicalities involved are quite standard and are omitted.

3 Marking Untagged Lists

In this section we determine the complexity of the untagged variant of the list marking problem. In Subsection 3.1, we prove that any randomized *Las Vegas* [8] algorithm for the problem requires $\Omega(\min\{\ell, n/p\})$ time with high probability. In Subsection 3.2, we give a deterministic algorithm whose running time is within an additive logarithmic term of the lower bound, showing that randomization cannot be significantly exploited in this case.

3.1 A Randomized Lower Bound

The intuition behind the lower bound is that a list element becomes distinguishable from a non-list element only when every element along the directed path from the head of the list to that element has been identified. As a consequence, random probes of the memory cells will not speed-up the computation in any significant way. This argument is formalized in the following theorem.

Theorem 1 *Suppose that with probability* $1 - o(1)$ *a randomized parallel algorithm on a* (p, n)*-PRAM marks every element of an untagged list of length* ℓ *within* t *time steps. Then*

$$t = \Omega\left(\min\left\{\ell, \frac{n}{p}\right\}\right).$$

Proof: Observe that a randomized algorithm can be seen as one chosen uniformly at random from a set $\mathcal{D}$ of deterministic algorithms (each deterministic algorithm being characterized by the outcome of a sequence of unbiased, independent random binary trials). In order to prove our lower bound for the untagged case, we construct a set of inputs with the property that *every* algorithm in $\mathcal{D}$ fails to mark the list in less than the time prescribed by the lower bound for a constant fraction of the inputs. From this, it immediately follows that, for some input in the set, a constant fraction among all the deterministic algorithms fail to mark the list in the prescribed time. Therefore, the failure probability of a randomly chosen algorithm, on that particular input, is bounded below by a positive constant. This lower bound technique was introduced by Yao in [11].

We will assume that ℓ is fixed and will restrict our attention to the following set of inputs. The contents of the memory are organized as a *circular list* of length n. The target list of length ℓ is a contiguous sublist of the circular list, and the address of the head and tail of the target list is given as input to the algorithm. (This formulation of the problem is essentially equivalent to that presented in the introduction, but more convenient in the current context.) There are $n!$ different inputs, corresponding to the $(n-1)!$ different circular lists of length n and the choice for the address of the head of the target list.

If $t > n/(2p)$, then the stated bound on t certainly holds. We can therefore concentrate on the case $t \leq n/(2p)$. At any time step, the algorithm probes a set of at most p memory cells. At each step $i \leq t$, we view the nodes on the circular list as grouped into k_i *sublists*, defined as maximal sets of adjacent probed nodes terminated by an unprobed node. Initially, there are n sublists, each consisting of a single distinct unprobed node. When a node v is

probed, its pointer to the next element v' in the list becomes known, and their corresponding sublists merge. The sublist containing the head of the target list, referred to as the *principal* sublist, contains a prefix of the target list, whose length is nondecreasing and becomes ℓ upon termination. Since each step of the algorithm causes at most p merges, at least $n - pi$ sublists remain after i steps.

Let n_j^i denote the number of nonprincipal sublists of length j at the beginning of the i-th step, so that $\sum_{j=1}^{n} n_i^j = k_i - 1$. For convenience, we assume that each step consists of a first substep during which $p - 1$ arbitrary cells are probed, followed by a second substep when the tail of the principal sublist is probed, which has the effect of grafting a single sublist onto the end of the principal sublist. Clearly, conforming to this discipline will not alter the running time of an algorithm by more than a constant factor. The merges provoked by the $p-1$ probes of the first substep yield k_{i+1} nonprincipal sublists. With the possible exception of the single sublist that will be grafted onto the principal sublist during the second substep, there are at most n_j^{i+1} sublists of length j. One of these sublists is grafted onto the principal sublist during this step, and because all input lists are equally likely, each of these sublists is equally likely to be chosen. Thus, the expected value of δ_i, the increase in the length of the principal sublist during step i, is bounded as follows

$$E[\delta_i] \leq \sum_{j=1}^{n} \frac{j n_j^{i+1}}{k_{i+1}} + \frac{n}{k_{i+1}} \leq \frac{2n}{n - pi} \leq 4,$$

since $k_{i+1} \geq n - pi \geq n/2$. The above summation accounts for all but one of the nonprincipal sublists, whose contribution is accounted for by the term n/k_{i+1}. Therefore, only constant progress is made, on average, at each step on the prefix of the list, and the stated result

follows. □

Note that the above bound is obtained under the optimistic assumption that a node belonging to the target list is recognized as such as soon as all the other nodes between the head of the target list and that node are probed, even though the algorithm, by the time it touches the corresponding memory cells, may not have sufficient information to determine that these cells actually contain list elements.

3.2 A Deterministic Upper Bound

We begin by outlining a relatively simple, slightly inefficient deterministic algorithm for the untagged list marking problem. We then provide a fast technique to transform the input instance into an equivalent, smaller one so that the running time of the algorithm on the new instance is within the desired bound. This "shrinking" process is accomplished by deleting nodes and rearranging edges of the pseudoforest underlying the original input instance.

Consider an untagged list of ℓ nodes stored in the memory of a (p, n)-PRAM, and let h be the (given) distinguished pointer to the head of the list. As we observed in Section 2, the nodes to be marked are the ancestors of h in a tree T whose root is r, the tail of the list. Suppose that we are given the preorder and postorder number of every node in T. Then, a particular node x is an ancestor of h if and only if $preorder(x) \leq preorder(h)$ and $postorder(x) \geq postorder(h)$ (see Fig. 2). Although the Euler-tour techniques of Tarjan and Vishkin [9] can be used to efficiently compute the preorder and postorder numberings in trees, these may not be applied immediately in the present context. Firstly, the presence of other components in the pseudoforest may complicate matters, and secondly the techniques

rely on an adjacency list representation for trees.

We circumvent the first difficulty as follows. Using a straightforward pointer-jumping technique, each node in T can identify the root r in $O(\log n)$ time per node or $O((n/p)\log n)$ time overall. Nodes not in T executing the same algorithm will "converge" on some node other than r and will hence be clearly identifiable as not belonging to T. All such nodes will remain dormant for the remainder of the algorithm.

Having eliminated all nodes not in T, we may now construct an adjacency list representation for T. Label the i-th cell C_i with the pair $< s, i >$ where s is the address of its parent in T. By sorting the cells lexicographically using Cole's algorithm [1], the children of each node occupy adjacent positions, and so they may be easily linked together in an adjacency list for that node. (These linking pointers are distinct from the successor pointers for the nodes and are stored in the scratch storage associated with the nodes in question.) The cells are then resorted with respect to their original addresses in order to reconstruct the original structure of T and to attach adjacency lists to the appropriate tree nodes. The implementation details are straightforward. The sorting steps dominate the running time and so the adjacency list representation for T can be constructed in $O((n/p)\log n)$ time.

Given the adjacency list representation for T, the preorder and postorder numberings can be computed in $O((n/p)\log n)$ time [9]. The identification and marking of the ancestors of h can be completed within the same time bound. Interleaving this $O((n/p)\log n)$ algorithm with the obvious $O(\ell)$ sequential algorithm, we obtain the following result.

Proposition 1 *An untagged list of length ℓ in the memory of an (p, n)-PRAM can be marked deterministically in $O\left(\min\{\ell, (n/p)\log n\}\right)$ time.*

We now proceed to develop a more efficient algorithm, which, by a sequence of *pruning* steps, will reduce the pseudoforest G to one, G', significantly smaller, and such that the ancestors of h in G' be among the ancestors of h in G. Marking of G' can be fast, due to small size, and then extended to mark all the ancestors of h in G. Each pruning step reduces the number of nodes in the pseudoforest by a constant factor by deleting some nodes and rearranging pointers among the active (undeleted) nodes. The basis for pruning steps is provided by the following lemma.

Lemma 1 *There is a constant $\mu < 1$ such that, for any pseudoforest $G = (V, E)$ containing h, there is a subset $W \subseteq V - \{h\}$ of leaves and unary nodes, such that:*

(i) *No pair of nodes in W are neighbors in G (W is an independent set).*

(ii) $|W| \geq \mu|V| - 1$*;*

Moreover, set W can be computed in $O(|V|/p + \log n/\log\log n)$ time.

Proof: Let $G = (V, E)$ be a pseudoforest. The nodes eligible for inclusion in W are the nodes in G of indegree zero or one, apart from h. This set induces a set of maximal node-disjoint linear chains (*i.e.*, either simple paths or cycles) of eligible nodes. Apply the following operation to each such chain. If the chain is of length one or two, then select one node; if the chain has length three or more, then select a subset of the nodes such that (i) no two adjacent nodes are selected, and (ii) the maximum number of consecutive unselected nodes on the chain is two. The union of the selected nodes for the various chains forms the desired set W.

The identification of nodes in G of indegree at most one is and the selection of one node in each chain of length at most two are straightforward and require only constant work per node.

The selection of nodes belonging to chains of length three or greater is instead accomplished by applying the 2-ruling algorithm of Cole and Vishkin [2] and selecting the nodes in the ruling. (Their algorithm is formulated in terms of circular lists, but this restriction can be easily relaxed.) This latter step can be completed in $O(|V|/p + \log n/\log\log n)$ time.

Consider a chain of length k. If $k \leq 2$, then $\lceil k/2 \rceil$ of the nodes on the chain are selected. If $k \geq 3$, then each pair of selected nodes is separated by at most two unselected nodes. Allowing for the possibility that the first two nodes on a chain might be unselected, we see that in this case the number of selected nodes is at least $\lceil (k-2)/3 \rceil \geq (1/9)k$.

If we let s denote the number of nodes in G of indegree at most one (including h), it is clear that $s \geq (|V|+1)/2$. The $s-1$ eligible nodes are arranged into chains of length one, two, or greater, and by the previous argument we are guaranteed that at least $(1/9)(s-1) \geq (1/18)|V| - 1$ nodes are selected. This establishes our claim with $\mu = 1/18$. □

A possible choice of set W for the pseudoforest of Figure 1 is shown in Figure 3. Notice that it is easy to delete a node w in W from G by redirecting the only edge incident on w (if any) to point to w's parent (or *nil*, if w has outdegree zero). By the above lemma, the graph G' thus obtained contains significantly fewer nodes than G (at most $(1-\mu)|V|+1$). It is also easy to verify that for every pair of nodes x and y in H', x is an ancestor of y in G' if and only if x is an ancestor of y in G. Thus, while smaller in size, the graph G' retains some of the ancestor-descendent information of the original graph G. The effect of the pruning step on the pseudoforest of Figure 1 is shown in Figure 4.

Before we describe the implementation of the pruning step in greater detail, we must

introduce some auxiliary data structures employed by the algorithm, the role of which will become clear in due course.

Each processor maintains a private stack that is empty before the first pruning. When a processor deletes a node during a pruning step, it pushes that node onto its stack. This facilitates the reconstruction of the graph at a later stage in the algorithm. Each processor also has a private list called its *work list*. Collectively, the p work lists hold all the active nodes in the graph. Between pruning steps, nodes are redistributed among work lists to ensure that each processor's work list contains an equal number of items. A processor is responsible for performing whatever operations are required for the nodes on its work list during a pruning step.

It should be emphasized that the only space overhead for these stacks and work lists is $O(1)$ per processor for a header pointer: the objects in these structures are nodes linked by pointers. These linking pointers are distinct from the successor pointers of the nodes in question and are represented within the scratch space of the nodes.

In summary, the pruning step applied to a pseudoforest $G = (V, E)$ may be described as follows.

1. Identify the set W.

2. Perform the following step for each active node x in W: mark x deleted, label the node with the current time and the name of its lone child, and push the node onto the local stack. For each active node c whose parent x is in W do the following: redirect c's successor pointer to point to x's parent, or to *nil* if x has no parent. (Each processor is responsible for the nodes on its own work list.)

3. Update the work lists.

Let a_i denote the total number of active nodes at the beginning of the i-th pruning step, for $i \geq 0$, that is, the number of nodes in the current pseudoforest. From Lemma 1 it follows that Step 1 above is completed in $O(a_i/p + \log n/\log\log n)$ time. Assuming that every processor work list holds $O(a_i/p)$ active nodes at the start of the pruning step, it is easy to see that Step 2 requires $O(a_i/p)$ time. To update the work lists in Step 3, each processor scans through its own list removing the deleted nodes and counting the active nodes. Finally, using a straightforward combination of parallel prefix [3] and routine pointer manipulations, it is possible to redistribute the active nodes among the work lists so that each processor's list receives at most $\lceil a_{i+1}/p \rceil$ nodes in $O(a_i/p + \log n/\log\log n)$ time. Thus, the i-th pruning step can be completed in $O(a_i/p + \log n/\log\log n)$ time.

The following recurrence provides an upper bound for the a_i's:

$$a_i \leq \begin{cases} (1-\mu)a_{i-1} + 1, & \text{for } i > 0, \\ n, & \text{for } i = 0. \end{cases}$$

Thus, $a_i \leq (1-\mu)^i n + \sum_{j=0}^{i-1}(1-\mu)^j$, which is bounded above by $\mu^{-1}n/\log n$ when $i \geq \lceil \log\log n/\log(1-\mu)^{-1} \rceil$. Selecting k to be this latter quantity, we see that the number of active nodes can be reduced to at most $\mu^{-1}n/\log n$ in time

$$O\left(\sum_{i=0}^{k-1} \frac{(1-\mu)^i n + \mu^{-1}}{p} + k\frac{\log n}{\log\log n}\right) = O\left(\frac{n}{p} + \log n\right).$$

In conclusion, the overall algorithm is as follows.

1. Apply k pruning steps to the initial pseudoforest G to produce a pseudoforest G' with at most $\mu^{-1}n/\log n$ nodes.

2. Apply the algorithm of Proposition 1 to mark all nodes in G' that are ancestors of h.

3. Reincorporate the nodes deleted during Step 1 in the reverse order to which they were deleted. In other words, first undo the deletions of the k-th pruning, then those of the $(k-1)$-st pruning, and so on. For each reinserted node, mark it if its child is marked.

We have already noted that Step 1 runs in $O(n/p + \log n)$ time. Step 3 basically performs the same operations as Step 1, but in reverse order, therefore it has a comparable running time. Finally, by Proposition 1, Step 2 is completed in $O(((\mu^{-1}n/\log n)/p)\log n) = O(n/p + \log n)$ time.

The correctness of the above algorithm can be proved as follows. Since an ancestor of a node x in G' is also an ancestor of x in G, and only leaves and unary nodes are deleted, all of the nodes marked by the algorithm are ancestors of h in G. On the other hand, suppose that some node along the directed path from h to r is not marked by the algorithm, and let x be the first such node. This node must have been deleted during one of the pruning operations in Step 1, otherwise it would have been marked during Step 2. Suppose that c was the lone child of x at the time that x was deleted. By assumption, node c is marked by the algorithm, and so when x is reinserted during Step 3, it too would be marked.

The main result of this section is summarized in the following theorem:

Theorem 2 *An untagged list of length ℓ stored in the memory of a (p,n)-PRAM can be marked deterministically in $O(\min\{\ell, n/p + \log n\})$ time.*

4 Marking Tagged Lists

Recall that in a tagged list each node carries a special symbol in its tag field so that it can be distinguished from non-list elements by inspection. The results of [7] show that tagging cannot lead to deterministic list-marking algorithms significantly faster than stated in Theorem 2. However, in this section, we sketch a simple randomized strategy which takes advantage of tags, and then we establish its optimality.

The randomized algorithm is quite simple and proceeds in two stages. In the first stage, each processor randomly accesses q memory locations and retains the addresses of those locations that contain list elements. Successful probes split the original list into sublists of nodes whose heads are marked and randomly distributed among the processors. Our intuition is that the qp random probes in the first stage will select list elements so that g, the length of longest sublist, is sufficiently small. Once the list has been split in this way, in the second stage we invoke a straightforward adaptation to the CRCW-PRAM of the well known randomized backtrack search algorithm of Karp and Zhang [6] that marks all the list nodes while balancing the work among the processors in $O(\ell/p + g)$ time, with high probability. Variants of this splitting technique are employed by Greene and Knuth [4] for graph traversal, and by Ullman and Yannakakis [10] for graph searching.

By interleaving the above strategy with the straightforward $O(\ell)$ sequential algorithm we can see that the list can be marked in $O(\min\{\ell, \ell/p+q+g\})$ time, with high probability. The following lemma illustrates the tradeoff between the parameters q and g, the two quantities that determine the running time of the algorithm.

Proposition 2 *Suppose that a tagged list of length ℓ is stored in the memory of a (p, n)-*

PRAM, into which t probes are made at random. Let random variable X denote the length of a longest contiguous subsequence of unprobed list elements. Then $\Pr(X > g) < \ell e^{-tg/n}$.

Proof: The probability that no probes are made within a fixed subsequence of length $g \leq \ell$ equals $(1 - g/n)^t < e^{-tg/n}$, and there are at most $\ell - g + 1$ such subsequences. □

With p processors making a total of pq probes, we have $\Pr(X > g) < \ell e^{-pqg/n}$. Setting $q = g = \sqrt{k(n/p)\log n}$ where k is an arbitrary constant greater than one, we can see that

$$\Pr\left(X > \sqrt{k\frac{n}{p}\log n}\right) < \ell n^{-k} \leq n^{-(k-1)}.$$

Thus our algorithm runs in $O(\min\left\{\ell, \ell/p + \sqrt{(n/p)\log n}\right\})$ time, with high probability.

Next, we establish a lower bound that is within a constant factor of this upper bound.

Theorem 3 *Suppose that with probability at least $1 - n^{-k}$, with $k > 0$, a randomized parallel algorithm on a (p, n)-PRAM marks every element of a tagged list of length ℓ within t time steps. Then*

$$t = \Omega\left(\frac{\ell}{p} + \min\left\{\ell, \sqrt{k\frac{n}{p}\log n}\right\}\right).$$

Proof: First note that ℓ/p is a trivial work-based lower bound for the problem. Next, assume that in i steps the first i list elements are marked, for all $1 \leq i \leq \ell$. This assumption does not weaken the argument for the lower bound.

Let W_j be the event that each of the first $j + 1$ list elements is marked within the first j steps. Also, let C_t be the event that every list element has been marked within t or fewer steps. Assume that $t < \ell$. Then $C_t \subseteq W_t$, which means that $\Pr(C_t) \leq \Pr(W_t)$ and, therefore $\Pr(\overline{C_t}) \geq \Pr(\overline{W_t})$.

Now, the probability that the $(i+1)st$ list element was touched by a random probe within the first i steps does not exceed pi/n. Hence, $\Pr(W_i|\overline{W_{i-1}}) \leq pi/n$, and consequently, $\Pr(\overline{W_i}|\overline{W_{i-1}}) \geq 1 - pi/n$. Combining this with the observation that $\Pr(\overline{W_1}) = 1 - p/n$, we can see that

$$\begin{aligned}
\Pr(\overline{W_t}) &\geq \prod_{i=1}^{t} \left(1 - \frac{pi}{n}\right) \\
&\geq \left(1 - \frac{pt}{n}\right)^t \\
&= e^{\Omega\left(-pt^2/n\right)},
\end{aligned}$$

for $t > 1$.

Thus, if $\Pr(\overline{C_t}) \leq n^{-k}$, it must be the case that $n^{-k} = e^{\Omega(-pt^2/n)}$, hence

$$t = \Omega\left(\sqrt{k\frac{n}{p}\log n}\right),$$

and the theorem follows. □

5 Conclusions

The results of this paper are summarized in Figure 5. The figure shows that in all cases, speed-ups over sequential performance can be obtained only for a number of processors larger than a certain threshold p_0. In the deterministic untagged case, $p_0 = \Theta(n/\ell)$ and $\log n = o(\ell)$; in the randomized tagged case, $p_0 = \Theta(n \log n/\ell^2)$. Moreover, in the untagged case, the deterministic upper bound and the randomized lower bound match except for

$p = \Omega(n/\log n)$, therefore randomization cannot be of substantial help. For tagged lists, however, with length in the range $\sqrt{(n/p)\log n} \leq \ell \leq n/p + \log n$, randomization affords considerable speedups.

Finally, preliminary investigations indicate that aspects of the above behavior remain when extending the algorithms for marking to other basic operations and/or to broader classes of linked structures.

Acknowledgments

The authors wish to thank the referees of EUROPAR'95 for their valuable feedback on the conference version of the paper, which resulted in improvements of the manuscript. This research was supported, in part, by the Istituto Trentino di Cultura through the Leonardo Fibonacci Institute, in Trento, Italy. Further research support was provided by MURST and CNR of Italy to G. Bilardi and G. Pucci , and by the ESPRIT contract No. 9072 (project GEPPCOM) to G. Bilardi, G. Pucci and K.T. Herley.

References

[1] Cole, R. Parallel merge sort. *SIAM J. Comput.* **17**, 4 (August 1988), 770–785.

[2] Cole, R., and Vishkin, U. Deterministic coin tossing with applications to optimal parallel list ranking. *Inform. and Control* **70**, 1 (July 1986), 32–53.

[3] Cole, R., and Vishkin, U. Faster optimal prefix sums and list ranking. *Inform. and Comput.* **81**, 3 (June 1989), 344–352.

[4] Greene, D. H., and Knuth, D. E. *Mathematics for the Analysis of Algorithms.* Birkauser, Boston MA, 1982.

[5] JàJà, J. *An Introduction to Parallel Algorithms.* Addison-Wesley, Reading MA, 1992.

[6] Karp, R. M., and Zhang, Y. Randomized parallel algorithms for backtrack search and branch and bound computation *J. ACM* **40**, 3 (July 1993), 765–789.

[7] Luccio, F., and Pagli, L. A model of sequential computation with pipelined access to memory. *Math. Systems Theory* **26**, 4 (July 1993), 343–356.

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

[9] Tarjan, R. E., and Vishkin, U. Finding biconnected components and computing tree functions in logarithmic time. *SIAM J. Comput.*, **14**, 4 (August 1985), 862–874.

[10] Ullman J. D., and Yannakakis, M. High-probability parallel transitive closure algorithms. In *Proc.* 2*-nd Annual ACM Symposium on Parallel Algorithms and Architectures*, ACM Press, Crete, GR, 1990, pp. 200–209.

[11] A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc.* 18*-th Annual IEEE Symposium on Foundations of Computer Science*, IEEE Press, Providence, RI, 1977, pp. 222–227.

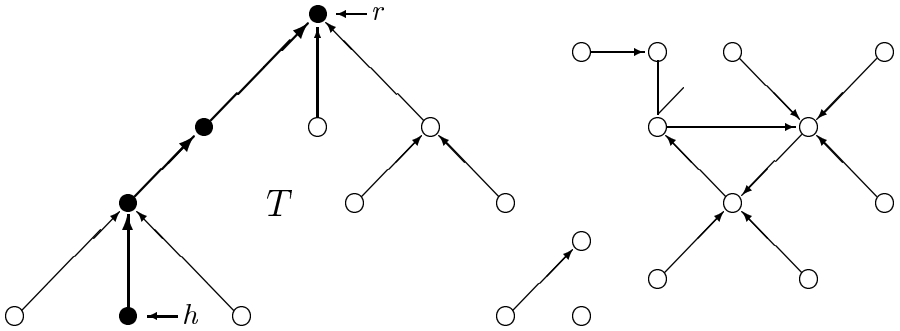

Figure 1: A pseudoforest. Thick lines and disks indicate the four-node list to be marked.

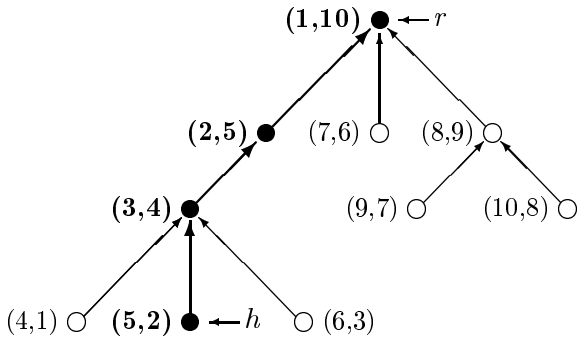

Figure 2: (*preorder*, *postorder*) numbers for nodes in T. A node x is an ancestor of h iff $preorder(x) \leq preorder(h)$ and $postorder(x) \geq postorder(h)$.

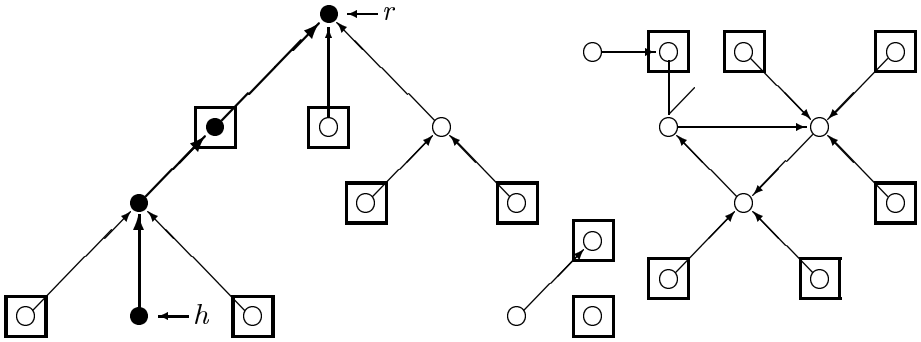

Figure 3: An independent set W (nodes enclosed into squares) for the pseudoforest of Figure 1.

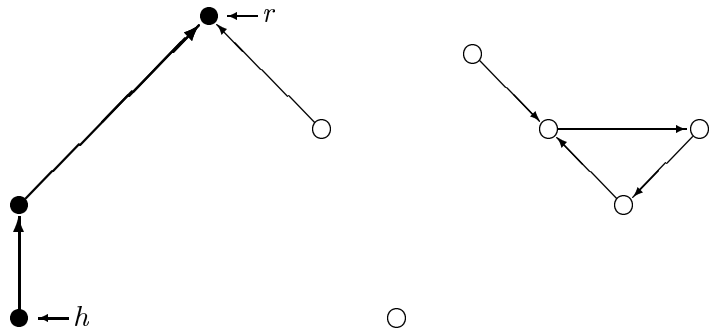

Figure 4: The pseudoforest of Figure 1 after the pruning step.

Deterministic Untagged Randomized Untagged	$O\left(\min\left\{\ell, n/p + \log n\right\}\right)$ $\Omega\left(\min\left\{\ell, n/p\right\}\right)$
Randomized Tagged	$\Theta\left(\min\left\{\ell, \ell/p + \sqrt{(n/p)\log n}\right\}\right)$

Figure 5: Summary of bounds for the marking problem.

Biographies

Gianfranco Bilardi received the Laurea (1978) (*summa cum laude*) in Electrical Engineering from the University of Padova and the Master (1982) and the PhD (1985) degrees, both in Electrical Engineering, from the University of Illinois at Urbana-Champaign. As a graduate student, he was awarded a Rotary International fellowship (1980) and the IBM predoctoral fellowship (1982-1984). From 1984 to 1990, he was an assistant professor of Computer Science at Cornell University, Ithaca, New York. In 1990, he joined the Department of Electronics and Informatics at the University of Padova, Italy, as a professor of Computer Science. Since 1996, he is also a professor of Electrical Engineering and Computer Science at the University of Illinois at Chicago. His research interests lie in the area of parallel and VLSI computing. He is the author of more than 60 publications in international journals and conferences. Dr. Bilardi is a member of the ACM and of the EATCS and is a senior member of the IEEE.

Sandeep N. Bhatt is Director of the Network Optimization and Computing research group at Bellcore, Morristown, New Jersey, and a research professor of Computer Science at Rutgers University. He received his SB, SM, and Phd (1984) from the Massachusetts Institute of Technology and was an associate professor of Computer Science at Yale University before joining Bellcore in 1982. During 1990, he was a visiting associate professor of Computer Science at Caltech. Dr. Bhatt's research interests include algorithmic models and architectures for parallel and distributed computing, high performance implementations of N-body algorithms for fluid dynamics, and network monitoring and surveillance. His research con-

tributions include new techniques and applications of graph embeddings, randomized algorithms for load-balancing, and the design of network architectures. His doctoral thesis on theoretical aspects of VLSI circuit layout provided a robust framework for solving different layout problems and presented efficient channel routing algorithms. Dr. Bhatt's broader interests include data structures, graph algorithms, and computational complexity.

Kieran T. Herley received his BSc (1982) and MSc (1983) both in Computer Science from University College Cork, Ireland. Further studies at Cornell University lead to an MS (1986) and a Phd (1990) in Computer Science. Since 1990, he has been a lecturer in the Department of Computer Science of University College Cork, Ireland. Dr. Herley's research interests include the design and analysis of parallel algorithms and parallel computational models. He is a member of ACM and EATCS.

Geppino Pucci received the Laurea (1987) (*summa cum laude*) and the Ph.D. (1993) degrees both in Computer Science from the University of Pisa, Italy. His Laurea thesis was awarded the IBM and the UNITEAM prizes for the best Italian theses in computer science. From 1988 to 1990 he was with the Computing Laboratory of the University of Newcastle-upon-Tyne, United Kingdom, as a research associate. In 1992, he joined the Department of Electronics and Informatics of the University of Padova, Italy, as an assistant professor. On leave from Padova, he spent 1993 at the International Computer Science Institute, Berkeley, California, as a postdoctoral fellow. His research interests include design and analysis of parallel algorithms, theory of computation, and probabilistic modeling. Dr. Pucci is a member of ACM.

Abhiram Ranade received the BTech degree in Electrical Engineering from the Indian Institute of Technology, Bombay, in 1981 and the doctorate in Computer Science from Yale University in 1989. Soon after graduation, he joined the Computer Science Division of the University of California at Berkeley as an assistant professor. Currently, he is with the Department of Computer Science and Engineering, Indian Institute of Technology, Bombay. His research interests include parallel architectures and algorithms, parallel programming techniques, and data structures.

Figure legends

Figure 1: A pseudoforest. Thick lines and disks indicate the four-node list to be marked.

Figure 2: (*preorder*, *postorder*) numbers for nodes in T. A node x is an ancestor of h iff $preorder(x) \leq preorder(h)$ and $postorder(x) \geq postorder(h)$.

Figure 3: An independent set W (nodes enclosed into squares) for the pseudoforest of Figure 1.

Figure 4: The pseudoforest of Figure 1 after the pruning step .

Figure 5: Summary of bounds for the marking problem.