INFORMATION AND COMPUTATION **138**, 1–22 (1997)
ARTICLE NO. IC972654

The Complexity of Parallel Prefix Problems on Small Domains

Shiva P. Chaudhuri

Max-Plack-Institut für Informatik, Im Stadtwald, 6600 Saarbrücken, Germany
E-mail: shiva@mpi-sb.mpg.de

and

Jaikumar Radhakrishnan

Tata Institute of Fundamental Research, Homi Bhabha Road, Colaba, Mumbai 400005, India
E-mail: jaikumar@tcs.tifr.res.in

We establish non-trivial lower bounds for several prefix problems in the CRCW PRAM model. The chaining problem is, given a binary input, for each 1 in the input, to find the index of the nearest 1 to its left. Our main result is that for an input of n bits, solving the chaining problem using $O(n)$ processors requires inverse-Ackerman time. This matches the previously known upper bound. We also give a reduction to show that the same lower bound applies to a parenthesis matching problem, again matching the previously known upper bound. We also give reductions to show that similar lower bounds hold for the prefix maxima and the range maxima problem. © 1997 Academic Press

1. INTRODUCTION

Lower bounds in parallel computation often depend critically on the domain size of the problem that is being solved. Typically, these lower bounds use Ramsey theoretic arguments to force the algorithms to behave in a structured manner on some subset of the inputs. It is then argued that this subset of inputs is rich enough so that this structured behavior cannot find a quick solution. Examples of lower bounds that use this method can be found in [Sn85, MW87, B89, BBG89]. However, applying Ramsey theoretic arguments necessitates assuming an unrealistically large domain size, often an iterated exponential in the size of the problem. These lower bounds become invalid when considering smaller domains. Thus, a major thrust of parallel complexity is to prove lower bounds for problems defined on smaller domains.

The need for small domain lower bounds is further emphasized by the fact that in recent years, algorithms have been presented that, on small domains, actually beat the lower bounds proven for large domains. A good example is the problem of finding the maximum of n integers using a CRCW PRAM with n processors. For

0890-5401/97 $25.00
Copyright © 1997 by Academic Press
All rights of reproduction in any form reserved.

a sufficiently large domain, the problem has a lower bound of $\Omega(\log\log n)$ [MW87]. However, if all the integers are drawn from $\{1, ..., n^c\}$, then it is possible to find the maximum in $O(c)$ time [FRW88].

In this paper, we investigate the complexity of some related problems defined on small domains. Each problem is to be solved on a PRIORITY CRCW PRAM with n processors. (See JáJá's book [JaJa91] for information on the various models of PRAMs.)

*Unordered Chaining.*Given $(a_1, a_2, ..., a_n) \in \{0, 1\}^n$, compute values $(b_1, b_2, ..., b_n)$, such that there exist distinct integers $i_1, i_2, ..., i_q \in \{1, 2, ..., n\}$ satisfying

1. $a_i = 1$ if and only if $i = i_j$ for some j;
2. $b_{i_1} = 0$;
3. $b_{i_j} = i_{j-1}$, for $j = 2, 3, ..., q$.

In other words, we link the non-zeros into a chain. The stronger *ordered* version requires linking the 1s into a chain in the order in which they appear.

Ordered Chaining. Given values $(a_1, a_2, ..., a_n) \in \{0, 1\}^n$, compute $(b_1, b_2, ..., b_n)$, such that

$$b_i = \begin{cases} 0 & \text{if } a_i = 0 \\ \max\{j \mid a_j = 1, j < i\} & \text{otherwise,} \end{cases}$$

where we define $\max\{\ \} = 0$.

Prefix Maxima. Given $(a_1, a_2, ..., a_n) \in \{1, ..., n^c\}^n$, compute, for $i = 1, 2, ..., n$, the value $b_i = \max\{a_j : 1 \leqslant j \leqslant i\}$.

Range Maxima. Given $(a_1, a_2, ..., a_n) \in \{1, ..., n^c\}^n$, preprocess the data so that one processor can quickly answer any question of the form "What is the maximum of $\{a_i, a_{i+1}, ..., a_j\}$?", for $1 \leqslant i \leqslant j \leqslant n$.

Parenthesis Matching with Nesting Level. Given a legal sequence of matched parentheses and the nesting level of each, find the match of each parenthesis.

Results. Our main result is an $\Omega(\alpha(n))$ lower bound on the running time of every CRCW PRAM algorithm with n processors solving the unordered chaining problem. ($\alpha(n)$ is the inverse of Ackerman's function and is a very slowly growing function. See Section 3.1.) This implies the same lower bound for ordered chaining, solving an open problem in [BJK90, R93, Rr90].

Using reductions, we show similar lower bounds for the other problems. By reducing the ordered chaining problem to the prefix maxima problem, we obtain a lower bound of $\Omega(\alpha(n))$ even when the domain is $\{1, 2, ..., n\}$. Consequently, finding prefix maxima is strictly harder than just finding the maximum. This aspect is discussed further in Section 6. Prefix maxima can, in turn, be reduced to range maxima. This shows that there exists a constant $c > 0$ such that any algorithm that preprocesses over the domain $\{1, 2, ..., n\}$ with n processors so that a single processor can answer a query in $c\alpha(n)$ steps, requires $\Omega(\alpha(n))$ time. By reducing the ordered chaining problem to the parenthesis matching problem, we obtain a lower bound of $\Omega(\alpha(n))$ even when the depth of nesting is at most 2.

1.1. Relation to Previous Work

The problems considered in this paper appear frequently as subproblems in parallel algorithms. Examples are integer sorting, merging, lowest common ancestor, and compaction (see [BDH89, MV91, BJK90, GR86, R93]). Hence these problems have received considerable attention, and, in recent years, there has emerged a body of literature on very fast parallel algorithms [BV93, GMV91, H92].

For the chaining problem, Berkman and Vishkin [BV93] and, independently, Ragde [R93] gave ingenious parallel algorithms that run in $O(\alpha(n))$ time. For a restricted class of algorithms called *oblivious* algorithms, Chaudhuri [Cha94] proved that ordered chaining requires $\Omega(\alpha(n))$ time. However, in the general case, no lower bound was previously known. Our bound is one of very few lower bounds that hold for constant size domains. In fact, it appears that the only other such bound for CRCW PRAMs is the $\Omega(\log n/\log\log n)$ lower bound for PARITY shown by Beame and Håstad [BH89]. Also, the only other lower bound we are aware of for a problem that can be solved in $o(\log\log n)$ time is a lower bound of $\Omega(\log^* n)$ for a load balancing problem, due to MacKenzie [Mac92].

For the prefix maxima problem, Gil and Rudolph [GR86] give an algorithm that runs in $O(\log\log n)$ time. Berkman *et al.* [BJK90] give an algorithm for prefix maxima that is sensitive to the size of the domain. On the domain $\{1, ..., s\}$, their algorithm runs in $O(\log\log\log s)$ time with n processors. If s is small, this beats the lower bound for large domains.

For the range maxima problem, Berkman *et al.* [BBG89] give a preprocessing algorithm for range maxima that runs in $O(\log\log n)$ time; answering a query then takes constant time. Berkman and Vishkin [BV93] give a preprocessing algorithm that runs in $O(\alpha(n))$ time for a restricted class of inputs in which the difference between two adjacent numbers is at most a constant. This implies a prefix maxima algorithm with the same performance for this class of inputs.

Berkman and Vishkin [BV93] give an $O(\alpha(n))$ algorithm for parenthesis matching with nesting level. Without the nesting level information, PARITY can be reduced to this problem; hence it requires $\Omega(\log n/\log\log n)$ time [BH89].

Our lower bound argument for chaining is based on the work of Dolev *et al.* [DDPW83], who used a clever and versatile averaging argument to show that a *weak superconcentrator* with a linear number of edges must have $\Omega(\alpha(n))$ depth. Chaudhuri [Cha94] adapted their method to obtain the lower bound in the oblivious case. Our proof is a further extension of this method.

1.2. Organization of the Paper

In our lower bound argument, we fix parts of the input to limit the ability of the algorithm to gather information. The *computation graph*, described in Section 2, enables us to express these restrictions in graph theoretic terms; in particular, the degrees of the vertices in the graph reflect the power of the algorithm. In the *regularized* computation graph, described in Section 2.1, the degrees of the vertices are maintained below certain bounds, thereby limiting the power of the algorithm. To get the desired result, we need to select these bounds carefully. This is accomplished using special sequences, called *Ackerman sequences*; these sequences are described in Section 3. Using the properties of these sequences, our main result, the lower bound for chaining, is derived in Section 4. The reductions leading to lower bounds for the other problems are described in Section 5. Finally, in Section 6, the consequences of the results in this paper and the problems left open are discussed.

2. PARTIAL INPUTS AND THE COMPUTATION GRAPH

In the following, $\mathscr{A}$ will be an algorithm solving the unordered chaining problem. For inputs of size n, let $\mathscr{A}$ use $P = P(n)$ processors and take $k = k(n)$ steps. (A step is defined as one round of reads followed by writes.)

A *partial input* is an element of $\{0, 1, *\}^n$. For a partial input b, we denote by $X(b)$ the set of inputs consistent with b. That is, $X(b) = \{x \in \{0, 1\}^n : \textit{for } i = 1, ..., n, b_i \neq * \rightarrow b_i = x_i\}$. The positions with value 1 in b will be called the *blockers* of b and the positions with value 0 will be called the *passers* of b. Let

$$\mathsf{Bl}(b) = |\{j : b_j = 1\}|; \qquad \mathsf{Pa}(b) = |\{j : b_j = 0\}|.$$

For partial inputs a and b, we say a is a refinement of b if $X(a) \subseteq X(b)$.

It will be convenient to model the computation of $\mathscr{A}$ using a graph. Let b be a partial input of size n. The *computation graph* of $\mathscr{A}$ on b, $G(b)$, is defined as follows:

$$V(G(b)) = \{(c, i) : \quad c \text{ is a cell of memory and } 0 \leqslant i \leqslant k\}.$$

That is, we have $k+1$ levels; in each level, we have one vertex for each cell in the memory. The set of vertices in level i will be called V_i. The directed edges go from vertices at one level to the vertices at the next level. Every edge is labeled by a processor. If on some input in $X(b)$, processor p reads cell c and writes to cell d in step $i+1$, then we have the edge $((c, i), (d, i+1))$ with label p. When we say that a processor p reads from cell (c, i) and writes to cell $(d, i+1)$, we mean that in step $i+1$ of the computation of the algorithm $\mathscr{A}$, p reads cell c and writes to cell d. We use $f_v(b)$ to denote the indegree of vertex v in the graph $G(b)$. Initially, bit i of the input is assumed to be in cell i; finally, component i of the output is assumed to be in cell i. We refer to vertex $(i, 0)$ as α_i (the *input* vertices) and vertex (i, k) as β_i (the *output* vertices).

Let $a \in \{0, 1\}^n$. We shall associate with each vertex of $G(a)$ a content. The content of cell $(c, 0)$ is the value stored in cell c at the beginning of the algorithm. For $i \geqslant 1$, the content associated with (c, i) is the content of the cell c after step i (which is maintained until the write of step $i+1$ changes it) in the computation of $\mathscr{A}$ on input a. We call this content $\mathsf{content}(a, (c, i))$. Similarly, for a processor p and an input $a \in \{0, 1\}^n$, $\mathsf{state}(a, (p, i))$ is the state in which processor p *enters step* i in the computation of $\mathscr{A}$ on input a. Thus, $\mathsf{state}(a, (p, i))$ is the state the processor p

assumes after the read of step $i-1$ and maintains until the read of step i. In particular, if $G(a)$ has an edge $((c, i-1), (d, i))$ with label p, then processor p enters step i in state $\mathsf{state}(a, (p, i))$, reads the value $\mathsf{content}(a, (c, i-1))$ from cell c, assumes state $\mathsf{state}(a, (p, i+1))$ and writes a value to cell d; thereafter it enters step $i+1$ in state $\mathsf{state}(a, (p, i+1))$.

For a partial input b, let

$$\mathsf{contents}(b, (c, i)) = \{\mathsf{content}(x, (c, i)) \colon x \in X(b)\};$$

$$\mathsf{states}(b, (p, j)) = \{\mathsf{state}(x, (p, i)) \colon x \in X(b)\}.$$

We say that (c, i) is a *fixed* vertex if $|\mathsf{contents}(b, (c, i))| = 1$; otherwise we say that (c, i) is a *free* vertex. Similarly, if $|\mathsf{states}(b, (c, i))| = 1$, we say the state of p at the beginning of step i is *fixed*. Note that these terms depend on the algorithm $\mathscr{A}$ and the partial input b. We will use these terms within the context of an algorithm and a partial input. Which algorithm and which partial input are meant should be clear from the context.

For an input $x \in \{0, 1\}^n$, we denote by $x^{(j)}$ the input that differs from x only in the jth coordinate. We will need the following fact.

FACT 2.1. *Let $\phi \colon \{0, 1\}^n \to \{0, 1\}$ and $b \in \{0, 1, *\}^n$. If $\phi(x) = \phi(x^{(j)})$ for all j and all $x, x^{(j)} \in X(b)$, then ϕ is constant over $X(b)$.*

Proof. We can think of ϕ as a two-colouring of the vertices of the n-dimensional cube in a natural way—the colour of vertex $x \in \{0, 1\}^n$ is $\phi(x)$. The vertices corresponding to inputs in $X(b)$ define a subcube of this cube. The condition $\phi(x) = \phi(x^{(j)})$ for all j and all $x, x^{(j)} \in X(b)$ is equivalent to saying that any two adjacent vertices in this subcube have the same colour, which implies that ϕ is constant over the subcube, because the subcube is a connected graph. ▮

For a processor, p, let

$$\mathsf{affect}(b, (p, i)) = \{j \colon \exists x, x^{(j)} \in X(b)\ \mathsf{state}(x, (p, i)) \neq \mathsf{state}(x^{(j)}, (p, i))\}.$$

By Fact 2.1, we conclude that if in addition to the blockers and passers of b, all input bits corresponding to positions in $\mathsf{affect}(b, (p, i))$ are set (to 0 or 1), then the state in which p enters step i is fixed. Similarly, for a cell c, we define

$$\mathsf{affect}(b, (c, i)) = \{j \colon \exists x, x^{(j)} \in X(b)\ \mathsf{content}(x, (c, i)) \neq \mathsf{content}(x^{(j)}, (c, i))\}.$$

That is, if b' is obtained from b by setting all input bits corresponding to positions in $\mathsf{affect}(b, (c, i))$, then (c, i) is a fixed vertex in $G(b')$.

The following lemma shows a lower bound on the number of different values that the output vertices may have, based on how refined the partial input is.

LEMMA 2.1. *Let b be a partial input of size n. Let β_i, $i = 1, ..., n$, be the output vertices in the computation graph $G(b)$. Then*

$$\sum_{i=1}^{n} |\mathsf{contents}(b, \beta_i)| \geqslant \frac{(n - Pa(b))^2}{2(Bl(b) + 1)}.$$

Proof. Construct a graph H with $n - \mathsf{Pa}(b)$ vertices corresponding to the $n - \mathsf{Pa}(b)$ positions of the non-zero bits in the partial input b. Put the directed edge (i, j) in, if $b_j = *$ and $j \in \mathsf{contents}(b, \beta_i)$. For a vertex i of H, the different edges (i, j) going out of i correspond to different contents of β_i. Note that, on the input obtained by setting all the stars of b to 0, the content of an output vertex, β_i, is the index of some blocker of b, or 0. The content of β_i on this input does not correspond to any edge of H, since all edges of H point to stars. Suppose H had S edges. Then, accounting for the content of each output cell not represented by any edge of H, we get

$$\sum_{i=1}^{n} |\mathsf{contents}(b, \beta_i)| \geqslant S + n. \tag{1}$$

We have thus related the quantity $\sum_{i=1}^{n} |\mathsf{contents}(b, \beta_i)|$ to the number of edges in H. We will now show that if the number of blockers and passers is small, H must have a large number of edges. The basic intuition behind this fact is as follows. The graph H is the union of all the different chains that the algorithm may construct on different inputs (omitting edges that point to blockers of b). Since different inputs give rise to different chains, if the number of ways to extend b to a complete input is large, then H cannot be sparse. In particular, we have the following claim.

Claim. *H has no independent set of size* $\mathsf{Bl}(b) + 2$.

Proof of Claim. Suppose H has an independent set of size $\mathsf{Bl}(b) + 2$. Consider the input formed by setting these $\mathsf{Bl}(b) + 2$ positions to 1 and all other stars to 0. When this input is chained by the algorithm, at least $\mathsf{Bl}(b) + 1$ of these positions have a pointer to some 1 in the input. Since none has a pointer to another in the independent set, these pointers may point only to blockers of b. But there are only $\mathsf{Bl}(b)$ blockers. Hence, H has no independent set of size $\mathsf{Bl}(b) + 2$. ▮

Turán's theorem [AS92, p. 81] implies that a graph with v vertices and e edges has an independent set of size $v^2/(v + 2e)$. Using this, we can show that since H does not have a large independent set, it must have many edges. H has $n - \mathsf{Pa}(b)$ vertices and S edges; hence,

$$\frac{(n - \mathsf{Pa}(b))^2}{n - \mathsf{Pa}(b) + 2S} \leqslant \mathsf{Bl}(b) + 1,$$

$$\text{i.e., } \frac{(n - \mathsf{Pa}(b))^2}{\mathsf{Bl}(b) + 1} \leqslant n - \mathsf{Pa}(b) + 2S.$$

Using (1), we get

$$\sum_{i=1}^{n} |\mathsf{contents}(b, \beta_i)| \geqslant \frac{(n - \mathsf{Pa}(b))^2}{2(\mathsf{Bl}(b) + 1)}. \quad ▮$$

The central idea of the proof is that we think of the quantity $\sum_{i=1}^{n} |\mathsf{contents}(b, \beta_i)|$ as a measure of the difficulty of the task that the algorithm has to accomplish.

Lemma 2.1 bounds this measure from below, relating it to the number of blockers and passers in a partial input. Intuitively, Lemma 2.1 states that if the partial input has few blockers and passers, then the algorithm still has a lot to do. When we refine a partial input, we potentially reduce the set of contents of cells and states of processors, thus restricting the algorithm. In Section 2.1 we describe a class of partial inputs that strongly restrict the algorithm. In Section 3.3 we show how to obtain such a partial input with a small number of blockers and passers. Such a partial input has the property that although the algorithm is severely restricted, the difficulty of the remaining task is still high. By carefully averaging over a number of such partial inputs, in Section 4, we conclude that if the algorithm runs in few steps, it must use many processors. We now make the idea of a restricted algorithm precise by introducing the notion of a regularized computation graph. The treatment below is taken from Chaudhuri [Cha96].

2.1. The Regularized Computation Graph

If a cell is written to by a small number of processors, then it can only have a small number of contents. Similarly, if a processor reads from a cell whose possible contents are limited, then the possible states it can attain after the read are also limited. In our analysis, we shall, guided by this intuition, strive to maintain bounds on the number of processors writing to cells, thus restricting the power of the algorithm.

Definition 2.1. Let $D=(d_0, d_1, ..., d_k)$ be a sequence of positive integers and let b be a partial input. We say that $G(b)$ is *D-regularized up to level l* $(l \leqslant k)$ if every free vertex of $G(b)$ at level i, $i=0, 1, 2, ..., l$, has indegree less than d_i. In this case, we say that b is *D-regularizing up to level l.* If $G(b)$ is D-regularized up to level k then we say that $G(b)$ is *D-regularized* and call b a *D-regularizing partial input.*

Let $D=(d_0, d_1, ..., d_k)$ satisfy $d_0 \geqslant 4$ and, for $i=1, 2, ..., k$, $d_i \geqslant d_{i-1}^4$. Let $\Delta_i = 2^{2^i} \prod_{j=0}^{i} d_j^{2^{i-j}}$. (We will also use Δ_i when $i=-1$: $\Delta_{-1}=\sqrt{2}$.) These quantities will prove useful in bounding certain parameters of D-regularized graphs in the remainder of this section. We will need the following estimates.

Fact 2.2. (a) *For $i=0, 2, ..., k$, $\Delta_i = d_i \Delta_{i-1}^2$.*

(b) *For $i=0, 1, ..., l$, $2^{2^i} \Delta_i \leqslant d_i^2$.*

Proof. Part (a) is easily verified using the definition.

To prove part (b), we use induction on i. The base case is trivial, because $d_0 \geqslant 4$. For $i \geqslant 1$, we have, using the induction hypothesis and $d_i \geqslant d_{i-1}^4$, that

$$2^{2^i} \Delta_i = 2^{2^i} d_i \Delta_{i-1}^2 = d_i (2^{2^{i-1}} \Delta_{i-1})^2 \leqslant d_i d_{i-1}^4 \leqslant d_i^2. \quad \blacksquare$$

For a partial input b, let

$$\mathsf{Maxcontents}_i(b) = \max\{ |\mathsf{contents}(b, (c, i))| : c \text{ is a memory cell}\};$$

$$\mathsf{Maxstates}_i(b) = \max\{ |\mathsf{states}(b, (p, i))| : p \text{ is a processor}\}.$$

Lemma 2.2. *Let $D=(d_0, d_1, ..., d_k)$ satisfy $d_0 \geqslant 4$ and, for $i=1, 2, ..., k$, $d_i \geqslant d_{i-1}^4$. Let b be a partial input such that $G(b)$ is D-regularized up to level l.*

(a) *For $i=1, 2, ..., l$, $\mathsf{Maxstates}_i(b) \leqslant \Delta_{i-2}$.*

(b) *For $i=0, 1, ..., l$, $\mathsf{Maxcontents}_i(b) \leqslant d_i \Delta_{i-1}$.*

Proof. We write G, $\mathsf{Maxcontents}_i$, $\mathsf{Maxstates}_i$ instead of $G(b)$, $\mathsf{Maxcontents}_i(b)$, $\mathsf{Maxstates}_i(b)$, respectively.

We have $\mathsf{Maxstates}_1 = 1 \leqslant \sqrt{2} = \Delta_{-1}$ and $\mathsf{Maxcontents}_0 = 2 \leqslant \Delta_0$. For the remaining cases, we shall use induction. We will verify that for $i=1, 2, ..., l$,

$$\mathsf{Maxstates}_{i+1} \leqslant \Delta_{i-1}$$

$$\mathsf{Maxcontents}_i \leqslant d_i \Delta_{i-1}.$$

First, we bound $\mathsf{Maxstates}_{i+1}$. The state of a processor at the beginning of step $i+1$ is completely determined by its state at the beginning of step i and the content of the cell it reads in step i. Now, there are $\mathsf{Maxstates}_i$ possibilities for its state at the beginning of step i, and for each such state, there are $\mathsf{Maxcontents}_{i-1}$ possibilities for the value it reads. Thus, using the induction hypothesis to bound $\mathsf{Maxstates}_i$ and $\mathsf{Maxcontents}_{i-1}$, we have

$$\mathsf{Maxstates}_{i+1} \leqslant \mathsf{Maxstates}_i \cdot \mathsf{Maxcontents}_{i-1} \leqslant \Delta_{i-2} \cdot d_{i-1} \Delta_{i-2} = \Delta_{i-1}.$$

(For the last equality we used Fact 2.2(a).)

Next, we bound $\mathsf{Maxcontents}_i$. Consider a free vertex (c, i) $(0 < i \leqslant l)$ in the graph G. Let the indegree of (c, i) be d (note that $d < d_i$). Let S_j be the number of states in which the processor on the jth edge coming into (c, i) writes to the cell. The content of (c, i) is determined by the state of the processor that succeeds in writing to it, or, if no processor writes, by the content of $(c, i-1)$. Thus, we have

$$|\mathsf{contents}(b, (c, i))| \leqslant \sum_{j=1}^{d} S_j + |\mathsf{contents}(b, (c, i-1))|.$$

Now $S_j \leqslant \mathsf{Maxstates}_{i+1}$, so $S_j \leqslant \Delta_{i-1}$. From the induction hypothesis, we have $|\mathsf{contents}(b, (c, i-1))| \leqslant \mathsf{Maxcontents}_{i-1} \leqslant d_{i-1} \Delta_{i-2}$. Thus, using Fact 2.2, we have

$$\mathsf{Maxcontents}_i \leqslant (d_i - 1)\, \Delta_{i-1} + d_{i-1} \Delta_{i-2} \leqslant (d_i - 1)\, \Delta_{i-1} + \Delta_{i-1} \leqslant d_i \Delta_{i-1}. \quad \blacksquare$$

For a partial input b, let

$$\mathsf{Maxcellaffect}_i(b) = \max\{|\mathsf{affect}(b, (c, i))| : c \text{ is a memory cell}\};$$

$$\mathsf{Maxprocaffect}_i(b) = \max\{|\mathsf{affect}(b, (p, i))| : p \text{ is a processor}\}.$$

Lemma 2.3. *Let $D=(d_0, d_1, ..., d_k)$ satisfy $d_0 \geqslant 4$ and, for $i=1, 2, ..., k$, $d_i \geqslant d_{i-1}^4$. Let b be a partial input such that $G(b)$ is D-regularized up to level l.*

(a) *For* $i = 0, 1, ..., l$, $\mathsf{Maxcellaffect}_i(b) \leqslant d_i^2$.

(b) *For* $i = 2, ..., l+1$, $\mathsf{Maxprocaffect}_i(b) \leqslant d_{i-2}^2$.

Proof. We write G, $\mathsf{Maxcellaffect}_i$, $\mathsf{Maxprocaffect}_i$ instead of $G(b)$, $\mathsf{Maxcellaffect}_i(b)$, $\mathsf{Maxprocaffect}_i(b)$, respectively.

Consider a free vertex (c, i) $(0 < i \leqslant l)$. Let W be the set of labels on the edges coming into this vertex (i.e., W is the set of processors that write to cell c at the end of step i for some input in $X(b)$); since G is D-regularized, $|W| \leqslant d_i - 1$. Recall that what the processor p writes at the end of step i is determined completely by the state it assumes after the read of step i. This in turn is completely determined by the input bits corresponding to positions in $\mathsf{affect}(b, (p, i+1))$ (by the remark after Fact 2.1). Similarly, the content of cell $(c, i-1)$ is determined completely by the input bits corresponding to positions $\mathsf{affect}(b, (c, i-1))$. Thus

$$\mathsf{affect}(b, (c, i)) \subseteq \bigcup_{p \in W} \mathsf{affect}(b, (p, i+1)) \cup \mathsf{affect}(b, (c, i-1)).$$

It follows that

$$\mathsf{Maxcellaffect}_i \leqslant (d_i - 1)\mathsf{Maxprocaffect}_{i+1} + \mathsf{Maxcellaffect}_{i-1}. \tag{2}$$

Similarly, if all input bits that affect the state of the processor p before the read of step $i+1$ are fixed and all the input bits that affect any of the cells that it could read in step $i+1$ are fixed, then the state of processor p after the read of step $i+1$ is fixed. Since there are at most $\mathsf{Maxstates}_{i+1}$ possible cells it could read in step i, we have

$$\mathsf{Maxprocaffect}_{i+2} \leqslant \mathsf{Maxprocaffect}_{i+1} + \mathsf{Maxstates}_{i+1} \cdot \mathsf{Maxcellaffect}_i. \tag{3}$$

We will now show that for $i = 0, 1, 2, ..., l$, $\mathsf{Maxcellaffect}_i \leqslant 2^{i-1} d_i \Delta_{i-1}$ and $\mathsf{Maxprocaffect}_{i+2} \leqslant 2^i \Delta_i$. The contents of a cell at the beginning of the first step and the state of a processor after the first read are determined by at most one bit of the input; hence $\mathsf{Maxcellaffect}_0$, $\mathsf{Maxprocaffect}_2 \leqslant 1$. This shows that the claim holds when $i = 0$. For $i \geqslant 1$, we shall use induction.

First, we consider $\mathsf{Maxcellaffect}_i$. Using Fact 2.2(a), we obtain from (2) and the induction hypothesis that

$$\begin{aligned}\mathsf{Maxcellaffect}_i &\leqslant (d_i - 1)\, 2^{i-1} \Delta_{i-1} + 2^{i-2} d_{i-1} \Delta_{i-2} \\ &\leqslant (d_i - 1)\, 2^{i-1} \Delta_{i-1} + 2^{i-1} \Delta_{i-1} = 2^{i-1} d_i \Delta_{i-1}.\end{aligned}$$

Next, we consider $\mathsf{Maxprocaffect}_{i+2}$. From (3), the induction hypothesis and the bound in Lemma 2.2 for $\mathsf{Maxstates}_{i+1}$, we obtain

$$\mathsf{Maxprocaffect}_{i+2} \leqslant 2^{i-1} \Delta_{i-1} + \Delta_{i-1} \cdot 2^{i-1} d_i \Delta_{i-1} = 2^{i-1} \Delta_i + 2^{i-1} \Delta_i = 2^i \Delta_i.$$

(For the second to last equality, we used Fact 2.2(a).)

Thus, we have established that for $i = 0, 1, ..., l$, $\mathsf{Maxcellaffect}_i \leqslant 2^{i-1} d_i \Delta_{i-1}$ and $\mathsf{Maxprocaffect}_{i+2} \leqslant 2^i \Delta_i$. The lemma now follows from Fact 2.2. ∎

Recall that $f_v(b)$ is the indegree of vertex v in graph $G(b)$.

LEMMA 2.4. *Let $D = (d_0, d_1, ..., d_k)$ satisfy $d_0 \geqslant 4$ and, for $i = 1, 2, ..., k$, $d_i \geqslant d_{i-1}^4$. Let b be a partial input such that $G(b)$ is D-regularized up to level l. If $l \leqslant k-1$, then*

$$\sum_{v \in V_{l+1}} f_v(b) \leqslant d_l^2 P.$$

Proof. We observe that a processor after the read of step $l+1$ can be in at most $\mathsf{Maxstates}_{l+2}(b)$ states, and can therefore appear as a label in at most $\mathsf{Maxstates}_{l+2}(b)$ edges of $G(b)$. The inequality follows from this since, by Lemma 2.2 and Fact 2.2(b), we have $\mathsf{Maxstates}_{l+2}(b) \leqslant \Delta_l \leqslant d_l^2$. ∎

3. THE ACKERMAN SEQUENCES

3.1. The Ackerman Functions

The Ackerman functions are defined as follows:

$$A_1(x) = 2x;$$

$$A_{i+1}(x) = A_i^{(x)}(1).$$

We use the notation $A_i^{(x)}(1)$ to mean A_i applied x times to 1. That is, $A_2(x) = 2^x$ and $A_3(x) = \mathsf{Tower}(x)$. The kth inverse Ackerman function, I_k, is defined by

$$I_k(n) = \max\{i\colon A_k(i) \leqslant n\}.$$

It can be verified that $A_k(1) = 2$ and $A_k(2) = 4$, for all k; in contrast, $A_k(3)$ is a very fast growing function of k. The Ackerman inverse of n is given by

$$\alpha(n) = \min\{k\colon A_k(3) \geqslant n\}.$$

3.2. The Ackerman Tree

We now construct certain sequences that we call *Ackerman sequences.* These sequences play a central role in our analysis of the computation graph. To help picture these sequences we first introduce a tree called the *Ackerman tree.* The Ackerman sequences will then be obtained from the labels on the paths of this tree.

The tree $T_i(x)$ is an ordered rooted tree defined inductively. Each edge of the tree has a label. We denote by $B_i(x)$ the largest label appearing in $T_i(x)$.

- $T_1(x)$ is a tree of depth 2. The root has one outgoing edge with label x, and the child of the root has x^5 edges, each with label x^{10}. Thus $B_1(x) = x^{10}$.

- $T_{i+1}(x)$ has depth $i+2$. The root has one outgoing edge with label x. The child of the root is formed by merging the roots of the following x^5 trees of depth $i+1$.

$$T_i(x^{10}), T_i(B_i(x^{10})), ..., T_i(B_i^{(x^5-1)}(x^{10})).$$

Thus $B_{i+1}(x) = B_i^{(x^5)}(x^{10})$, since the largest label in $T_i(B_i^{(x^5-1)}(x^{10}))$ has value $B_i(B_i^{(x^5-1)}(x^{10})) = B_i^{(x^5)}(x^{10})$. (Note that the function $B_i(x)$ has the same *doubly recursive* character as the Ackerman function, albeit with different values of the parameters.)

The Ackerman tree $\Gamma(k, l)$ has depth $k+1$. It is obtained by merging the roots of the following l trees:

$$T_k(256), T_k(B_k(256)), ..., T_k(B_k^{(l-1)}(256)).$$

The tree $\Gamma(k, l)$ may alternatively be described as follows.

T1. All leaves of the tree are at distance $k+1$ from the root.

T2. The outdegree of the root is l.

T3. The label on the leftmost edge of the root is 256.

T4. If the label on the edge coming into a non-leaf node γ is d, then γ has d^5 children.

T5. If the label on the edge coming into the node γ is d and e is the leftmost edge coming out of γ, then the label on e is d^{10}.

T6. If $f = (\gamma, \gamma')$ is not the leftmost edge coming out of γ, then its label is obtained as follows. Let e be the edge coming out of γ immediately to the left of f. Then the label of f is the largest label that appears on an edge of a path starting with e and ending at a leaf.

We now show the properties of $\Gamma(k, l)$ that we use in the proof of our lower bound. These properties will be used in Section 3.3 while constructing a regularized computation graph and in Section 4 while proving the lower bound.

Let H be the set of leaves of the tree $\Gamma(k, l)$. For $h \in H$, and $i = 1, 2, ..., k+1$, $d_i(h)$ is the label on the ith edge from the root on the path connecting the root with the leaf h. We set $d_0(h) = 4$ and define $D(h) = (d_0(h), d_1(h), d_2(h), ..., d_k(h))$. The sequences $D(h)$ play a central role in our analysis. For each leaf h, we construct a $D(h)$-regularized computation graph and derive the lower bound by averaging over the leaves of the tree. Note that $d_{k+1}(h)$ is not part of $D(h)$. Thus, several leaves give rise to the same $D(h)$; the last level of vertices in the tree are used just to provide *weights* to the sequences while averaging.

The next two lemmas shows how the d values increase when one travels from top to bottom or from left to right in the tree.

LEMMA 3.1. *For $h \in H$ and $i = 1, 2, ..., k+1$, $d_i(h) \geqslant (d_{i-1}(h))^4$.*

Proof. If $i \geqslant 2$, then from parts T5 and T6 of the alternative definition of the tree we have $d_i(h) \geqslant (d_{i-1}(h))^{10}$ and the claim follows easily. For $i=1$, we have $d_1(h) \geqslant 256$ and $d_0(h)=4$, and again the claim holds.

Let $d_i^+(h)$ be the next largest value of d_i after $d_i(h)$; that is,

$$d_i^+(h) = \min\{d_i(h'): d_i(h') > d_i(h)\}.$$

$(\min\{\ \} = \infty.)$

LEMMA 3.2. *For $i=1, ..., k-1$, $d_i^+(h) \geqslant (d_k(h))^{10}$.*

Proof. If $d_i^+(h) = \infty$, then we are done immediately. Otherwise, from part T6 of the alternate definition of the tree, we have that $d_i^+(h) \geqslant d_{k+1}(h)$, and from parts T5 and T6, we have $d_{k+1}(h) \geqslant (d_k(h))^{10}$. ∎

The set of children of the node γ is denoted by $\mathsf{succ}(\gamma)$. $T(\gamma)$ denotes the subtree rooted at γ and $H(\gamma)$ the leaves in $T(\gamma)$; thus, if γ is not a leaf, then $H(\gamma) = \bigcup_{\gamma' \in \mathsf{succ}(\gamma)} H(\gamma')$. We denote by Γ_i the set of nodes of $\Gamma(k, l)$ at distance i from the root. Observe that if $\gamma \in \Gamma_i$, then the values $d_0(h)$, $d_1(h)$, ..., $d_i(h)$, and $d_i^+(h)$ are the same for all leaves $h \in H(\gamma)$. We refer to these values as $d_0(\gamma)$, $d_1(\gamma)$, ..., $d_i(\gamma)$, and $d_i^+(\gamma)$, respectively.

LEMMA 3.3. *The number of leaves up to h (h and those to the left of h) is at most $(d_k(h))^6$.*

Proof. For $\gamma \in \Gamma_k$, the number of children of γ is exactly $(d_k(\gamma))^5$. Let $\gamma' \in \Gamma_k$ be the parent of h. By Lemma 3.2, $d_k^+(\gamma) > d_k(\gamma)$, i.e., at the kth level, the labels on the edges strictly increase from left to right. Hence the number of the nodes in Γ_k up to γ' is at most $d_k(\gamma') = d_k(h)$. Since each of these nodes has at most as many children as γ', i.e., $d_k(h)^5$, the total number of leaves up to h is at most $d_k(h)(d_k(h))^5 = (d_k(h))^6$. ∎

LEMMA 3.4. *For $i=1, ..., k$, $\sum_{\gamma \in \Gamma_i} 1/d_i(\gamma) \leqslant 2^{-(i+1)}$.*

Proof. It follows from Lemma 3.1 that $\min\{d_i(\gamma): \gamma \in \Gamma_i\} \geqslant 2^{i+2}$. Then, using Lemma 3.2, we have $d_i^+(\gamma) \geqslant (d_i(\gamma))^{10} \geqslant 2d_i(\gamma)$. Thus the sum is at most the sum of a decreasing geometric series starting with the value $2^{-(i+2)}$. ∎

For $h \in H$, define $E[a, b](h) = \prod_{i=a}^{b} (d_i(h))^5$ and $E(h) = E[1, k](h)$. ($E[a, b] = 1$ if $a > b$.) If $\gamma \in \Gamma_i$ and $a, b \leqslant i$, then $E[a, b](h)$ is constant over $H(\gamma)$; we refer to this value as $E[a, b](\gamma)$.

LEMMA 3.5. *Suppose $i \geqslant 1$ and $\gamma \in \Gamma_i$. Then $\sum_{h \in H(\gamma)} 1/E[1, k](h) = 1/E[1, i-1](\gamma)$.*

Proof. We use reverse induction on i. For $i = k+1$, the claim is obvious. For $1 \leqslant i \leqslant k$, we split the sum by $\gamma' \in \mathsf{succ}(\gamma)$ and use induction to obtain

$$\sum_{\gamma' \in \mathsf{succ}(\gamma)} \sum_{h \in H(\gamma')} \frac{1}{E[1,k](h)} = \sum_{\gamma' \in \mathsf{succ}(\gamma)} \frac{1}{E[1,i](\gamma')} = \frac{|\mathsf{succ}(\gamma)|}{E[1,i-1](\gamma)(d_i(\gamma))^5}$$
$$= \frac{1}{E[1,i-1](\gamma)},$$

where the last equation holds since $|\mathsf{succ}(\gamma)| = (d_i(\gamma))^5$. ∎

LEMMA 3.9. $\sum_{h \in H} 1/E[1,k](h) = l$.

Proof. Using Lemma 3.5, we have

$$\sum_{h \in H} \frac{1}{E(h)} = \sum_{\gamma \in \Gamma_1} \sum_{h \in H(\gamma)} \frac{1}{E[1,k](h)} = \sum_{\gamma \in \Gamma_1} \frac{1}{E[1,0](\gamma)} = |\Gamma_1| = l. \quad ∎$$

LEMMA 3.7. $H(\Gamma(k,l)) \leqslant A_{4k}(l)$.

Proof. We first show by induction that, for $x \geqslant 5$, $B_i(x) \leqslant A_{3i}(x)$. For the base case, we have $B_1(x) = x^{10}$ and $A_3(x) = \mathsf{Tower}(x)$, and $\mathsf{Tower}(x) \geqslant x^{10}$, for $x \geqslant 5$.

For the induction step, we have the following routine derivation:

$$\begin{aligned}
A_{3(i+1)}(x) &= A_{3i+2}^{(x)}(1) = A_{3i+2}(A_{3i+2}^{(x-1)}(1)) = A_{3i+2}(A_8^{(x-1)}(1)) \\
&\geqslant A_{3i+2}(\mathsf{Tower}(x-1)) \geqslant A_{3i+2}(x^5 + x) \\
&\geqslant A_{3i+1}^{(x^5+x)}(1) = A_{3i+1}^{(x^5)}(A_{3i+1}^{(x)}(1)) \\
&\geqslant A_{3i+1}^{(x^5)}(x) \geqslant B_{3i}^{(x^5)}(x) \\
&\geqslant B_{i+1}(x).
\end{aligned}$$

To obtain the second line from the first, we used $A_8(y) \geqslant \mathsf{Tower}(y)$ and $\mathsf{Tower}^{(x-1)}(1) = \mathsf{Tower}(x-1)$; the second to last step was obtained by induction.

It follows from Lemma 3.3 that the number of leaves in $\Gamma(k,l)$ is at most the largest label appearing in $\Gamma(k,l)$, raised to the sixth power, i.e., $(B_k^{(l)}(256))^6$. We show that for $l \geqslant 5$, $(B_k^{(l)}(256))^6 \leqslant A_{4k}(l)$. For $k=1$, we have $(B_1^{(l)}(256))^6 = (256^{10^l})^6 \leqslant A_4(l)$.

For $k \geqslant 2$, we have another routine derivation:

$$\begin{aligned}
A_{4k}(l) &= A_{4k-1}^{(l)}(1) = A_{4k-1}(A_{4k-1}^{(l-1)}(1)) \\
&\geqslant A_{4k-1}(256 + l + 1) \qquad \text{since} \quad l \geqslant 5 \\
&\geqslant A_{4k-2}^{(256+l+1)}(1) = A_{4k-2}^{(l+1)}(A_{4k-2}^{(256)}(1)) \\
&\geqslant A_{4k-2}^{(l+1)}(256) \geqslant (A_{3k}^{(l)}(256))^6 \\
&\geqslant (B_k^{(l)}(256))^6.
\end{aligned}$$

The last step follows from the previous derivation. ∎

3.3. Obtaining a $D(h)$-Regularizing Partial Input

We now analyze the computation graph of algorithm $\mathscr{A}$. Let h be a leaf of the tree $\Gamma(k, l)$, and consider the sequence $D(h)$ defined in Section 3.2. We shall associate with h a $D(h)$-regularizing partial input $b(h)$ with a small number of blockers and passers. This will enable us to apply Lemma 2.1.

The partial input $b(h)$ is produced in stages. The intermediate partial inputs produced will be called $b_0(h), b_1(h), ..., b_k(h)$. The partial input $b_i(h)$ will be $D(h)$-regularizing up to level i. At the end we set $b(h) = b_k(h)$. From now on we will omit the parameter h when referring to $b(h)$, the intermediate partial inputs $b_i(h)$, or the values $d_i(h)$, if the value of h is clear from the context.

Initially, we set $b_0 = *^n$. Trivially, b_0 is $D(h)$-regularizing up to level 0. Now, in the graph $G(b_0)$, there may be free vertices at level 1 that have indegree d_1 or higher. In Stage 1 of our procedure, we refine b_0 to obtain b_1 so that, in $G(b_1)$, the indegree of every free vertex at level 1 will be less than d_1; that is, b_1 will be $D(h)$-regularizing up to level 1. In general, when we come to Stage i, we already have a partial input b_{i-1} that is $D(h)$-regularizing up to level $i-1$. Our task in Stage i is to obtain a refinement b_i of b_{i-1} so that, in $G(b_i)$, every free vertex at level i has indegree less than d_i. The indegree of a vertex cannot increase when the partial input is refined; hence, b_i is $D(h)$-regularizing up to level i.

Stage i. Consider the graph $G(b_{i-1})$. Recall that a *free* vertex is one which has more than one possible content. A free vertex v at level i will be called a *high degree* vertex if $d_i \leqslant f_v < d_i^+$; it will be called a *very high degree* vertex if $f_v \geqslant d_i^+$. To obtain b_i we consider these high and very high degree vertices one by one, and, if necessary, refine the partial input to eliminate them. The temporary partial input produced will be denoted by b'; at the beginning of Stage i, $b' = b_{i-1}$.

(A) *High Degree Vertices.* Consider a vertex v that has high degree in $G(b_{i-1})$. If v is not high degree in $G(b')$, then we do nothing. Otherwise, for each processor p that writes to v, we fix all input bits corresponding to $\mathsf{affect}(b', (p, i+1))$ to 0. Recall that the state of processor p while writing in step i is determined completely by the values of these bits. Thus, for the resulting partial input b', if any of these processors writes to v, then v is fixed; otherwise, v has indegree 0 in $G(b')$. Note that we created only passers in this case.

(B) *Very High Degree Vertices.* Assume that all the high degree vertices of b_{i-1} have been processed in Step A, and the resulting partial input is b'. Next, consider a vertex v that had very high degree in $G(b_{i-1})$. If the indegree of v in $G(b')$ is less than d_i, then we do nothing. Otherwise, let p be the processor of highest priority that writes to cell v. There is some input $x \in X(b')$ on which p writes to v. We set all inputs in $\mathsf{affect}(b', (p, i+1))$ to the value they have in x. This fixes the state of processor p in step i so that it writes to v. Since we are in the PRIORITY model, p will override any other processor that tries to write, and v is a fixed vertex in the graph of the resulting partial input. Note that we may create both passers and blockers in this case.

At the end of this process, all the free vertices at level i have indegree less than d_i. We call the resulting partial input $b_i(h)$. Let the number of inputs set in step A of Stage i be $S^i_A(h)$, and let the number of inputs in step B of Stage i be $S^i_B(h)$.

LEMMA 3.8. $S^i_A(h) \leqslant \sum_{v \in V_i:\, d_i \leqslant f_v(b_{i-1}(h)) < d_i^+} f_v(b_{i-1}(h))(d_{i-1}(h))^2$.

Proof. First observe, using Lemma 3.1, that the sequence $D(h)$ satisfies the conditions in Lemma 2.3. Now consider the processing of a high vertex v. Let the partial input when v is processed be b'. Since b' is a refinement of $b_{i-1}(h)$, we have that $G(b')$ is $D(h)$-regularized up to level $i-1$, $f_v(b') \leqslant f_v(b_{i-1}(h))$, and, for all processors p, $\mathsf{affect}(b', (p, i+1)) \subseteq \mathsf{affect}(b_{i-1}(h), (p, i+1))$. It follows that

$$S^i_A(h) \leqslant \sum_{\substack{v \in V_i: \\ d_i \leqslant f_v(b_{i-1}(h)) < d_i^+}} f_v(b_{i-1}(h)) \; \mathsf{Maxprocaffect}_{i+1}(b_{i-1}(h)).$$

The lemma follows from this because $\mathsf{Maxprocaffect}_{i+1}(b_{i-1}(h)) \leqslant (d_{i-1}(h))^2$ by Lemma 2.3(b). ∎

Similarly, we can show

LEMMA 3.9. $S^i_B(h) \leqslant \sum_{v \in V_i:\, d_i^+ \leqslant f_v(b_{i-1}(h))} (d_{i-1}(h))^2$.

We will use the following observation.

Observation 3.1. The partial input $b_i(h)$ constructed by the above procedure depends only on $d_1(h), d_2(h), ..., d_i(h)$. Therefore, if $\gamma \in \Gamma_i$, then for all leaves $h \in H(\gamma)$, $b_i(h)$ is the same. We denote this common value of b_i by $b_i(\gamma)$.

4. THE LOWER BOUND

In this section, we will show that no algorithm can solve the unordered chaining problem in constant time using a linear number of processors. We will use the partial input $b(h)$ described in the previous section. In fact, we will select roughly n different partial inputs, one for each leaf of a tree, $\Gamma(k, l)$, and use an averaging argument. In our calculations the number of passers and blockers in $b(h)$ will play an important role; for brevity, we denote them by $\mathsf{Pa}(h)$ and $\mathsf{Bl}(h)$ instead of $\mathsf{Pa}(b(h))$ and $\mathsf{Bl}(b(h))$.

THEOREM 4.1. *Let $\mathscr{A}$ be an algorithm that solves the unordered chaining problem for inputs of length n in k steps using P processors. Suppose $A_{4k}(5) \leqslant n$. Then $P = \Omega(nI_{4k}(n))$.*

Proof. Consider the tree $\Gamma(k, l)$ with $l = I_{4k}(n)$. We have $l \geqslant 5$, and by Lemma 3.7, the number of leaves in $\Gamma(k, l)$ is at most $A_{4k}(I(4k, n)) \leqslant n$.

Consider the partial input $b(h)$ associated with the leaf h. By Lemma 2.2, for each output vertex β_i, $|\mathsf{contents}(b(h), \beta_i)| \leqslant (d_k(h))^2$. Using Lemma 2.1 and $E(h) \geqslant (d_k(h))^2$, we then get

$$nE(h) \geqslant \frac{(n - \mathsf{Pa}(h))^2}{2(\mathsf{Bl}(h)+1)},$$

implying

$$2\left(\mathsf{Bl}(h)+\frac{\mathsf{Pa}(h)}{E(h)}+1\right)\geqslant\frac{n}{E(h)}.$$

Summing over all leaves h and using Lemma 3.6, we get

$$2\left(\sum_h\left(\mathsf{Bl}(h)+\frac{\mathsf{Pa}(h)}{E(h)}+1\right)\right)\geqslant\sum_h\frac{n}{E(h)}=nl\geqslant nI_{4k}(n).$$

We shall show (Lemma 4.1(b) and Lemma 4.2) that

$$\sum_h \mathsf{Bl}(h)\leqslant P \qquad \text{and} \qquad \sum_h\frac{\mathsf{Pa}(h)}{E(h)}\leqslant 2P.$$

Therefore, we have $6P+2n\geqslant nI_{4k}(n)$, that is, $P=\Omega(nI_{4k}(n))$. ▮

COROLLARY 4.1. *If $\mathscr{A}$ is an algorithm solving the unordered chaining problem with a linear number of processors, then $\mathscr{A}$ needs $\Omega(\alpha(n))$ time.* ▮

LEMMA 4.1.

(a) *For $i=1, 2, ..., k$, $\sum_h S^i_B(h)\leqslant P/2^i$.*

(b) $\sum_h \mathsf{Bl}(h)\leqslant P$.

Proof. As observed earlier, blockers are created only in step B of the procedure described in Section 3.3. Hence,

$$\sum_h \mathsf{Bl}(h)\leqslant\sum_h\sum_{i=1}^{k} S^i_B(h)=\sum_{i=1}^{k}\sum_h S^i_B(h).$$

Thus, Part (b) of the lemma follows easily from Part (a).
We now show part (a). By Lemma 3.9,

$$\sum_h S^i_B(h)\leqslant\sum_h\sum_{\substack{v\in V_i:\\ d_i^+(h)\leqslant f_v(b_{i-1}(h))}}(d_{i-1}(h))^2\leqslant\sum_{v\in V_i}\sum_{\substack{h:\\ d_i^+(h)\leqslant f_v(b_{i-1}(h))}}(d_{i-1}(h))^2.$$

By Observation 3.1, for $\gamma\in\Gamma_{i-1}$, $b_{i-1}(h)$ is constant over $H(\gamma)$. Therefore, we may group the different h by the value of γ and obtain

$$\sum_h S^i_B(h)\leqslant\sum_{v\in V_i}\sum_{\gamma\in\Gamma_{i-1}}\sum_{\substack{h\in H(\gamma)\,\wedge\\ d_i^+(h)\leqslant f_v(b_{i-1}(h))}}(d_{i-1}(h))^2.$$

Since $(d_{i-1}(h))^2$ is constant for the innermost sum, it can be moved out. The sum then reduces to

$$\sum_{v \in V_i} \sum_{\gamma \in \Gamma_{i-1}} (d_{i-1}(\gamma))^2 \, |R(v, \gamma)|,$$

where $R(v, \gamma) = \{h \in H(\gamma)\colon d_i^+(h) \leqslant f_v(b_{i-1}(h))\}$. Let h' be the rightmost leaf in $R(v, \gamma)$. Then by Lemma 3.3,

$$|R(v, \gamma)| \leqslant (d_k(h'))^6.$$

Using Lemmas 3.2 and 3.1, we have

$$\begin{aligned} f_v(b_{i-1}(h')) &\geqslant d_i^+(h') \geqslant (d_k(h'))^{10} \geqslant (d_k(h'))^4 \, |R(v, \gamma)| \\ &\geqslant (d_{i-1}(\gamma)^{16} \, |R(v, \gamma)| \geqslant (d_{i-1}(\gamma))^5 \, |R(v, \gamma)|. \end{aligned}$$

Therefore, $(d_{i-1}(\gamma))^2 \, |R(v, \gamma)| \leqslant f_v(b_{i-1}(\gamma))/(d_{i-1}(\gamma))^3$, and

$$\sum_h S_B^i(h) \leqslant \sum_{v \in V_i} \sum_{\gamma \in \Gamma_{i-1}} \frac{f_v(b_{i-1}(\gamma))}{(d_{i-1}(\gamma))^3} = \sum_{\gamma \in \Gamma_{i-1}} \frac{1}{(d_{i-1}(\gamma))^3} \sum_{v \in V_i} f_v(b_{i-1}(\gamma)).$$

Now, by Lemma 2.4, $\sum_{v \in V_i} f_v(b_{i-1}(\gamma)) \leqslant P(d_{i-1}(\gamma))^2$. Therefore,

$$\sum_h S_B^i(h) \leqslant P \sum_{\gamma \in \Gamma_{i-1}} \frac{1}{d_{i-1}(\gamma)}.$$

Using Lemma 3.4, we then get the required bound $\sum_h S_B^i(h) \leqslant P/2^i$. ∎

LEMMA 4.2. $\sum_h (\mathsf{Pa}(h)/E(h)) \leqslant 2P$.

Proof. From the definitions in Section 3.3, we have

$$\mathsf{Pa}(h) \leqslant \sum_{i=1}^{k} (S_A^i(h) + S_B^i(h)).$$

It follows from Lemma 4.1(a) that

$$\sum_h \sum_{i=1}^{k} \frac{S_B^i(h)}{E(h)} \leqslant P. \tag{4}$$

We shall show that, for $i = 1, 2, ..., k$,

$$\sum_h \frac{S_A^i(h)}{E(h)} \leqslant \frac{P}{2^i} \tag{5}$$

It follows that

$$\sum_{i=1}^{k} \sum_{h} \frac{S_A^i(h)}{E(h)} \leqslant P. \tag{6}$$

The lemma then follows by combining (4) and (6).

To prove (5), we use Lemma 3.8 and write

$$\begin{aligned} \sum_h \frac{S_A^i(h)}{E(h)} &\leqslant \sum_h \sum_{\substack{v \in V_i:\\ d_i(h) \leqslant f_v(b_{i-1}(h)) < d_i^+(h)}} \frac{(d_{i-1}(h))^2 f_v(b_{i-1}(h))}{E(h)} \\ &\leqslant \sum_{v \in V_i} \sum_{\substack{h:\\ d_i(h) \leqslant f_v(b_{i-1}(h)) < d_i^+(h)}} \frac{(d_{i-1}(h))^2 f_v(b_{i-1}(h))}{E(h)}. \end{aligned}$$

As in the proof of Lemma 4.1 we compute the inner sum by grouping the leaves by $\gamma \in \Gamma_{i-1}$. Then

$$\begin{aligned} \sum_h \frac{S_A^i(h)}{E(h)} &\leqslant \sum_{v \in V_i} \sum_{\gamma \in \Gamma_{i-1}} \sum_{\substack{h \in H(\gamma):\\ d_i(h) \leqslant f_v(b_{i-1}(h)) < d_i^+(h)}} \frac{(d_{i-1}(h))^2 f_v(b_{i-1}(h))}{E(h)} \\ &\leqslant \sum_{v \in V_i} \sum_{\gamma \in \Gamma_{i-1}} (d_{i-1}(\gamma))^2 f_v(b_{i-1}(\gamma)) \left(\sum_{\substack{h \in H(\gamma):\\ d_i(h) \leqslant f_v(b_{i-1}(h)) < d_i^+(h)}} \frac{1}{E(h)} \right). \end{aligned} \tag{7}$$

Fix $v \in V_i$ and $\gamma \in \Gamma_{i-1}$, and consider the inner sum

$$\sum_{\substack{h \in H(\gamma):\\ d_i(h) \leqslant f_v(b_{i-1}(h)) < d_i^+(h)}} \frac{1}{E(h)}.$$

Note that the condition, $d_i(h) \leqslant f_v(b_{i-1}(h)) < d_i^+(h)$, now depends only on $d_i(h)$ and $d_i^+(h)$, which are constant over $H(\gamma')$, for each $\gamma' \in \mathsf{succ}(\gamma)$. Therefore, this time we group the different h based on γ' and obtain

$$\sum_{\substack{h \in H(\gamma):\\ d_i(h) \leqslant f_v(b_{i-1}(h)) < d_i^+(h)}} \frac{1}{E(h)} \leqslant \sum_{\gamma' \in \mathsf{succ}(\gamma)} \left[\delta(v, \gamma') \sum_{h \in H(\gamma')} \frac{1}{E(h)} \right],$$

where

$$\delta(v, \gamma') = \begin{cases} 1 & \text{if} \quad d_i(\gamma') \leqslant f_v(b_{i-1}(\gamma')) < d_i^+(\gamma') \\ 0 & \text{otherwise.} \end{cases}$$

By Lemma 3.5, $\sum_{h \in H(\gamma')} 1/E(h) = 1/E[1, i-1](\gamma)$. Observe that $\delta(v, \gamma') = 1$ for at most one $\gamma' \in \mathsf{succ}(\gamma)$, since the intervals $[d_i(\gamma'), d_i^+(\gamma') - 1]$ are disjoint for different γ'. Therefore

$$\sum_{\substack{h \in H(\gamma): \\ d_i(h) \leqslant f_v(b_{i-1}(h)) < d_i^+(h)}} \frac{1}{E(h)} \leqslant \frac{1}{E[1, i-1](\gamma)}.$$

Returning to (7), we now have

$$\begin{aligned} \sum_h \frac{S_A^i(h)}{E(h)} &\leqslant \sum_{v \in V_i} \sum_{\gamma \in \Gamma_i} \frac{(d_{i-1}(\gamma))^2 f_v(b_{i-1}(\gamma))}{E[1, i-1](\gamma)} \\ &\leqslant \sum_{\gamma \in \Gamma_{i-1}} \left[\frac{(d_{i-1}(\gamma))^2}{E[1, i-1](\gamma)} \left(\sum_{v \in V_i} f_v(b_{i-1}(\gamma)) \right) \right]. \end{aligned}$$

Using Lemma 2.4, $\sum_{v \in V_i} f_v(b_{i-1}(\gamma)) \leqslant (d_{i-1}(\gamma))^2 P$. Thus,

$$\sum_h \frac{S_A^i(h)}{E(h)} \leqslant \sum_{\gamma \in \Gamma_{i-1}} \frac{(d_{i-1}(\gamma))^4 P}{E[1, i-1](\gamma)} \leqslant \sum_{\gamma \in \Gamma_{i-1}} \frac{P}{d_{i-1}(\gamma)}.$$

The inequality (5) follows from this by using Lemma 3.4. ▮

5. REDUCTIONS

We now give easy reductions to obtain lower bounds for related problems.

THEOREM 5.1. *Every algorithm that solves prefix maxima on domain* $\{1, ..., n\}$ *with n processors requires time* $\Omega(\alpha(n))$.

Proof. We reduce the ordered chaining problem to a prefix maxima problem on domain $\{1, ..., n\}$. On input $a_1, ..., a_n$, compute $c_1, ..., c_n$, where $c_i = 0$ if $a_i = 0$ and $c_i = i$ if $a_i = 1$. Then, solve prefix maxima for $c_1, ..., c_n$. Let $d_1, ..., d_n$ be the prefix maxima. The solution to the ordered chaining problem is given by $b_1 = 0$ and for $2 \leqslant i \leqslant n$, $b_i = 0$ if $d_{i-1} \in \{0, d_i\}$, and $b_i = d_{i-1}$ otherwise. ▮

THEOREM 5.2. *There exists a constant* $c > 0$ *such that every algorithm that preprocesses for range maximum on domain* $\{1, ..., n\}$ *with n processors, so that a single processor can answer a query in* $c\alpha(n)$ *steps, requires* $\Omega(\alpha(n))$ *time for preprocessing.*

Proof. We reduce the prefix maxima problem to the range maxima problem. On input $a_1, ..., a_n$, first preprocess for range maxima and then assign n processors, one to find the maximum of $[1, i]$, $1 \leqslant i \leqslant n$. ▮

THEOREM 5.3. *Parenthesis matching with nesting level using n processors requires* $\Omega(\alpha(n))$ *time, even when the depth of nesting is at most* 2.

Proof. Given an n-bit input to the ordered chaining problem, we will produce an input to the parenthesis matching problem with $2n + 2$ symbols. We replace each 0 with "()" and assign a nesting level of 2 to both parentheses; replace each 1 with

")(" and assign a nesting level of 1 to both parentheses. Add a "(" before and a ")" after the whole sequence, both with nesting level 1. Note that every ")" with nesting level 1 corresponds to some 1 in the original input. The "(" that matches it corresponds to the 1 preceding it in the original input. Thus, after solving the parenthesis matching problem, it is easy to recover the solution to the ordered chaining problem in constant time. ▮

6. CONCLUDING REMARKS

We have presented lower bounds for chaining, prefix maxima, range maxima, and parenthesis matching on small domains. The bounds are tight for the chaining problem and parenthesis matching, but we do not know about the other two problems. Our work extends the techniques developed in Dolev *et al.* [DDPW83] and Chaudhuri [Cha94]. The techniques used in this paper have since been sharpened and applied to several other problems. In Chaudhuri [Cha93a], they have been used to obtain $\Omega(\log\log n)$ lower bounds for the problem of *approximate compaction*, which is the problem of relocating a distinguished subset of the input values into an initial segment of approximately the same size. In Chaudhuri [Cha96], these methods have been placed in a general setting and shown to be applicable to an entire class of sensitive functions rather than just isolated cases, as in earlier works.

In the literature, several fast *randomized* solutions have been proposed for the problems considered in this paper. Berkman *et al.* [BMV92] give randomized preprocessing algorithms for the range maxima problem that run in $O(\log^* n)$ time; each query can then be answered in constant time. Raman [Rr90] gives a constant time randomized chaining algorithm that works if the number of 1s in the input is not too large. However, no non-trivial lower bounds have been reported for any of these problems. Is there an $\Omega(\alpha(n))$ lower bound for chaining, even if randomization is permitted? We have not succeeded in extending our methods to obtain such a lower bound.

We intuitively expect prefix maxima to be harder than just finding the maximum; however, the two problems often have the same complexity. For example, with one processor the complexity is $\Theta(n)$, and with n processors and a sufficiently large domain, $\Theta(\log\log n)$. Our lower bound is the only instance known to us where the two are shown to have different complexities. This suggests that the difference arises because of restricting the domain size. However, if we restrict the domain size further, to a constant, then both have complexity $O(1)$. It is an interesting open question to determine when the two problems have different complexities.

ACKNOWLEDGMENTS

We are grateful to Vince Grolmusz for introducing us to the problem of chaining and to Ravi Boppana for suggesting the approach taken in this paper. This work was reported earlier in the conference paper [CR92]. We thank Magnús Halldórsson for his comments on the presentation in this paper. We thank the referees and Faith Fich for several suggestions, which greatly helped in improving the presentation in the paper.

Received October 21, 1993; final manuscript received May 12, 1997

REFERENCES

[AS92] Alon, N., and Spencer, J. (1992), "The Probabilistic Method," Wiley, New York.

[B89] Boppana, R. (1989), Optimal separations between concurrent write parallel machines, *in* "Proc. of the 21st ACM STOC," pp. 320–326.

[BBG89] Berkman, O., Breslauer, D., Galil, Z., Scheiber, B., and Vishkin, U. (1989), Highly parallelizable problems, *in* "Proc. of the 21st ACM TOC," pp. 309–319.

[BDH91] Bhatt, P. C. P., Diks, K., Hagerup, T., Prasad, V. C., Radzik, T., and Saxena, S. (1991), Improved deterministic parallel integer sorting, *Inform. and Comput.* **94**, 643–670.

[BH89] Beame, P., and Håstad, J. (1989), Optimal bounds for decision problems on the CRCW PRAM, *J. Assoc. Comput. Mach.* **36** (3), 643–670.

[BJK90] Berkman, O., JáJá, J., Krishnamurthy, S., Thurimella, R., and Vishkin, U. (1990), Some triply-logarithmic parallel algorithms, *in* "Proc. of 31st FOCS," pp. 871–881.

[BMV92] Berkman, O., Matias, Y., and Vishkin, U. (1992), Randomized range-maxima in nearly-constant parallel time, *Comput. Complexity* **2**, 350–373.

[BV93] Berkman, O., and Vishkin, U. (1993), Recursive star-tree parallel data structure, *SIAM J. Comput.* **22** (2), 221–242.

[CFL] Chandra, A. K., Fortune, S., and Lipton, R. (1985), Unbounded fan-in circuits and associative functions, *J. Comput. System Sci.* **30**, 222–234.

[CFL83b] Chandra, A. K., Fortune, S., and Lipton, R. J. (1983), Lower bounds for constant depth circuits for prefix problems, *in* "Proc. of the 10th Intl. Colloqium on Automata, Languages and Programming," Lecture Notes in Computer Science, Springer-Verlag, Berlin/New York.

[Cha94] Chaudhuri, S. (1994), Tight bounds on oblivious chaining, *SIAM J. Comput.* **23** (6), 1253–1265.

[CR92] Chaudhuri, S., and Radhakrishnan, J. (1992), The complexity of parallel prefix problems on small domains, *in* "Proc. 33rd IEEE FOCS," 638–647.

[Cha93a] Chaudhuri, S. (1993), A lower bound for linear approximate compaction, *in* "Proc. of 2nd Israel Symp. on Theory of Comput. and System.," pp. 25–32.

[Cha96] Chaudhuri, S. (1993), Sensitive functions and approximate problems, *Inform. and Comput.* **126** (2), 161–168.

[DDPW83] Dolev, D., Dwork, C., Pippenger, N., and Wigderson, A. (1983), Superconcentrators, generalizers and generalized connectors with limited depth, *in* "Proc. of the 15th ACM STOC," pp. 42–51.

[FRW88] Fich, F. E., Wigderson, A., and Ragde, P. (1988), Simulations among concurrent-write models of parallel computation, *Algorithmica* **3**, 43–51.

[GMV91] Gil, J., Matias, Y., and Vishkin, U. (1991), Towards a theory of nearly constant time parallel algorithms, *in* "Proc. of 32nd IEEE FOCS," pp. 698–710.

[GR86] Gil, J., and Rudolph, L. (1986), Counting and packing in parallel, *in* "International Conference on Parallel Processing," pp. 1000–1002.

[H92] Hagerup, T. (1992), The log-star revolution, *in* "Proc. 9th Symposium on Theoretical Aspects of Computer Science," Lecture Notes in Computer Science, Vol. 577, pp. 259–278, Springer-Verlag, Berlin/New York.

[JaJa91] JáJá, J. (1992), "An Introduction to Parallel Algorithms," Addison–Wesley, Reading, MA.

[Mac92] MacKenzie, P. D. (1992), Load balancing requires $\Omega(\log^* n)$ expected time, *in* "Proc. of 3rd ACM–SIAM SODA," pp. 94–99.

[MV91] Matias, Y., and Vishkin, U. (1990), On parallel hashing and integer sorting, *J. Algorithms* **12**, 573–606.

[MW87] Meyer auf der Heide, F., and Wigderson, A. (1987), The complexity of parallel sorting, *SIAM J. Comput.* **16** (1), 100–107.

[NRW89] Newman, I., Ragde, P., and Wigderson, A. (1990), Perfect hashing, graph entropy and circuit complexity, *in* "Proc. of 5th Ann. Conf. on Structure in Complexity Theory," pp. 91–99.

[R93] Ragde, P. (1990), The parallel simplicity of compaction and chaining, *J. Algorithms* **14**, 371–380.

[Rr90] Raman, R. (1990), The power of collision: Randomized parallel algorithms for chaining and integer sorting, *in* "10th FST & TCS Conf.," Lecture Notes in Computer Science, Vol. 472, Springer-Verlag, Berlin/New York, pp. 161–175.

[Sn85] Snir, M. (1985), On parallel searching, *SIAM J. Comput.* **14** (2), 688–708.