MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

The Complexity of Parallel Prefix Problems on Small Domains

S. Chaudhuri J. Radhakrishnan

MPI-I-93-147 Oktober 1993

INFORMATIK

Im Stadtwald
66123 Saarbrücken
Germany

The Complexity of Parallel Prefix Problems on Small Domains

S. Chaudhuri J. Radhakrishnan

MPI-I-93-147 Oktober 1993

The Complexity of Parallel Prefix Problems on Small Domains

Shiva P. Chaudhuri
Max-Plack-Institut für Informatik
Im Stadtwald
6600 Saarbrücken
Germany
shiva@mpi-sb.mpg.de

Jaikumar Radhakrishnan
Tata Institute of Fundamental Research
Homi Bhabha Road, Colaba
Bombay 400005
India
jaikumar@tcs.tifr.res.in

Abstract

We show non-trivial lower bounds for several prefix problems in the CRCW PRAM model. Our main result is an $\Omega(\alpha(n))$ lower bound for the chaining problem, matching the previously known upper bound. We 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 problems.

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. Then it is shown that these structured algorithms cannot solve the problem quickly. Examples of lower bounds that use this method can be found in [27, 23, 2, 3]. 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 a sufficiently large domain, the problem has a lower bound of $\Omega(\log\log n)$ [23]. However, if all the integers are drawn from $\{1,\ldots,n^c\}$ then it is possible to find the maximum in $O(c)$ time [16].

In this paper, we investigate the complexity of some related problems defined on small domains. Each problem is to be solved on a a PRIORITY CRCW PRAM with n processors (see JáJá's book [20] for information on the various models of PRAMs).

> *Unordered Chaining.* Given $(a_1, a_2, \ldots, a_n) \in \{0,1\}^n$, compute values $(b_1, b_2, \ldots, b_n)$, such that there exist integers $i_1, i_2, \ldots, i_q$ satisfying
>
> 1. $a_i = 1$ iff $i = i_j$ for some j;
> 2. $b_{i_1} = 0$;
> 3. $b_{i_j} = i_{j-1}$, for $j = 2, 3, \ldots, q$.

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

Ordered chaining. Given values $(a_1, a_2, \ldots, a_n) \in \{0,1\}^n$, compute $(b_1, b_2, \ldots, b_n$, such that $a_i = 0 \rightarrow b_i = 0$ and $b_i = \max\{j \mid a_j = 1, j < i\}$ otherwise (define $\max\{\} = 0$).

Prefix maxima. Given $(a_1, a_2, \ldots, a_n) \in D^n$, for some ordered domain D, compute, for $i = 1, 2, \ldots, n$, the value $b_i = \max\{a_j : 1 \leq j \leq i\}$.

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

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 [6, 25, 26].

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 $\{1, 2, \ldots, n\}$. 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 $\{1, 2, \ldots, n\}$ 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, for example, in integer sorting, merging, lowest common ancestor and compaction (see [4, 22, 6, 18, 25]). Hence these problems have received considerable attention, and, in recent years, there has emerged a body of literature on very fast parallel algorithms [8, 17, 19].

For the chaining problem, Berkman and Vishkin [8] and, independently, Ragde [25] gave ingenious parallel algorithms that run in $O(\alpha(n))$ time. For a restricted class of algorithms called *oblivious* algorithms, Chaudhuri [11] 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 [5]. 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 [21].

For the prefix maxima problem, Gil and Rudolph [18] an algorithm that runs in $O(\log\log n)$ time. Berkman, JáJá, Krishnamurthy, Thurimella and Vishkin [6], give an algorithm for prefix maxima that is sensitive to the size of the domain. On the domain $\{1, \ldots, s\}$, 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, Breslauer, Galil, Scheiber and Vishkin [3] 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 [8] 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 [8], give an $O(\alpha(n))$ algorithm for parenthesis matching with nesting level. Without the nesting level given, PARITY can be reduced to this problem; hence it requires $\Omega(\log n/\log\log n)$ time [5].

Our lower bound argument for chaining is based on the work of Dolev, Dwork, Pippenger and Wigderson [15], 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 [11] 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 curtail 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, A will be an algorithm solving the unordered chaining problem. For inputs of size n, let 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, $X(b) = \{x \in \{0,1\}^n : \textit{for } i = 1,\ldots,n,\ b_i \neq * \rightarrow b_i = x_i\}$. 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 of b. Let

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

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

We shall find it convenient to model the computation of A on a graph. Let b be a partial input of size n. The *computation graph* of A on b, $G(b)$, is defined as follows.

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

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 labelled 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. 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, bit 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 associated with (c,i) is the content of the cell c after step i (that is, just before the write of step $i+1$ changes it) in the computation of A on the input a. We call this content $content(a,(c,i))$. Similarly, for a processor p and an input $a \in \{0,1\}^n$, $state(a,(p,i))$ is the state of processor p just before the write of step $i+1$ in the computation of A on input a. For a partial input b, let

$$\begin{aligned} contents(b,(c,i)) &= \{content(x,(c,i)) : x \in X(b)\}; \\ states(b,(p,j)) &= \{state(x,(p,i)) : x \in X(b)\}. \end{aligned}$$

We say that (c,i) is a *fixed* vertex if $|contents(b,(c,i))| = 1$; otherwise we say (c,i) is a *free* vertex. Similarly, if $|states(b,(c,i))| = 1$, we say the state of p is fixed. Note that the above definitions depend on the algorithm A and the size of input n. These parameters will be clear from the context where they are used.

For an input $x \in \{0,1\}^n$, we denote by $x^{(j)}$ the input that differs from x only in the j-th coordinate. We shall need the following fact, which we state without proof.

Fact 1 *Let $f : \{0,1\}^n \to \{0,1\}$ and $b \in \{0,1,*\}^n$. Then, if for all j and all $x, x^{(j)} \in X(b)$ $f(x) = f(x^{(j)})$, then f is constant over $X(b)$.*

Let

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

By Fact 1, we conclude that if in addition to the blockers and passers of b, all positions in $affect(b,(p,i))$ are fixed (at any value whatsoever), then the state of processor p after the read of step $i+1$ is fixed. Similarly, we define

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

That is, if b' is obtained from b by fixing all input positions in $affect(b,(c,i))$, then (c,i) is a fixed vertex in $G(b')$.

We model the computation of the algorithm A on the computation graph as follows. We say that a processor p reads from cell (c,i) and writes to cell $(d,i+1)$ when we mean that in the step $i+1$ of the computation of the algorithm A, p reads cell c and writes to cell d.

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

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

Proof. Construct a graph G with $n - Pa(b)$ vertices corresponding to the $n - Pa(b)$ non-zero positions in the partial input b. Put the directed edge (i,j) in, if $b_j = *$ and $j \in contents(b,\beta_i)$. For a vertex i of G, the different edges (i,j) going out of i correspond to different contents of β_i. Note that the content of an output vertex β_j on the input obtained from b by setting all the

stars to 0 does not correspond to any edge of G. Suppose G had S edges. Then, accounting for the n contents not represented by any edge of G, we get

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

Suppose G has an independent set of size $Bl(b)+2$. Consider the input formed by setting these $Bl(b)+2$ positions to 1 and all other stars to 0. When this input is chained by the algorithm, there must be at least $Bl(b)+1$ links in the chain. By our construction of G, these links may point only to blockers of b. But there are only $Bl(b)$ blockers. Hence G has no independent set of size $Bl(b)+2$. By Turán's theorem [1, page 81], we get

$$\begin{aligned} \frac{(n - Pa(b))^2}{n - Pa(b) + 2S} &\leq Bl(b) + 1 \\ \text{i.e.} \quad \frac{(n - Pa(b))^2}{Bl(b) + 1} &\leq n - Pa(b) + 2S. \end{aligned}$$

Using (1), we get

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

■

The central idea of the proof is that we think of the quantity $\sum_{i=1}^{n} |contents(b, \beta_i)|$ as a measure of the difficulty of the task that the algorithm has to accomplish. Lemma 2 bounds this measure from below. When we refine a partial input, we potentially reduce the set of contents of cells and states of processors, thus restricting the algorithm. We will find a partial input such that the algorithm is severely restricted, although the difficulty of the remaining task is still high. This allows us to 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 [14].

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 3 *Let $D = (d_0, d_1, \ldots, d_k)$ 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 *($l \leq k$) if every free vertex of $G(b)$ at level i, $i = 1, 2, \ldots, l$, 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.*

For a partial input b, let

$$\begin{aligned} Y_i(b) &= \max\{|contents(b, (c, i))| : c \text{ is a memory cell}\}; \\ Z_i(b) &= \max\{|states(b, (p, i))| : p \text{ is a processor}\}; \\ M_i(b) &= \max\{|affect(b, (c, i))| : c \text{ is a memory cell}\}; \\ N_i(b) &= \max\{|affect(b, (p, i))| : p \text{ is a processor}\}. \end{aligned}$$

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

(a) For $i = 0, 1, \ldots, l$,

$$Y_i(b), Z_i(b), M_i(b), N_i(b) \leq d_i^2.$$

(b) If $l \leq k-1$, then

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

Proof. We write G, Y_i, Z_i, M_i, N_i instead of $G(b), Y_i(b), Z_i(b), M_i(b), N_i(b)$ respectively.

We first obtain bounds for Y_i and Z_i. Consider a free vertex (c, i) $(0 < i \leq l)$ in the graph G. Let the indegree of (c, i) be d (note that $d < d_i$). Let $p_1, p_2, \ldots, p_d$ be the labels on the edges coming into (c, i). Let the number of states in which processor p_j writes to (c, i) be S_j. Now, the content of (c, i) is determined by the state of the processor that succeeds in writing to (c, i), or, if no processor writes to (c, i), by the content of $(c, i-1)$. Thus, we have

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

From the definition of Z_{i-1} and Y_{i-1}, we have $S_j \leq Z_{i-1}$ and $|contents(b, (c, i-1))| \leq Y_{i-1}$. Thus, for $i = 1, 2, \ldots, l$, we have

$$Y_i \leq (d_i - 1) Z_{i-1} + Y_{i-1}. \tag{2}$$

The number of states a processor may assume after the read of step $i+1$ is at most the product of the number of states it had before the read and the number of possible contents of the cell it reads. Thus, for $i = 1, 2, \ldots, l$, we have

$$Z_i \leq Z_{i-1} Y_i. \tag{3}$$

We have $Y_0, Z_0 \leq 2$, since a memory cell can have at most two contents initially and a processor after the first read can be in at most two states.

Now let $\Delta_i = 2^{2^i} \prod_{j=0}^{i} d_j^{2^{i-j}}$. Note that for $i = 1, 2, \ldots, k$, $\Delta_i = d_i \Delta_{i-1}^2$. We shall show by induction that, for $i = 0, 1, \ldots, l$, $Z_i \leq \Delta_i$ and $Y_i \leq d_i \Delta_{i-1}$. The basis case, $i = 0$, is trivial. For $i \geq 1$, we obtain from (2) and the induction hypothesis that

$$Y_i \leq (d_i - 1)\Delta_{i-1} + d_{i-1}\Delta_{i-2} \leq (d_i - 1)\Delta_{i-1} + \Delta_{i-1} \leq d_i \Delta_{i-1},$$

and then from the induction hypothesis and (3) that

$$Z_i \leq d_i \Delta_{i-1}^2 = \Delta_i.$$

Next we bound M_i and N_i. Consider the vertex (c, i) $(0 < i \leq l)$ and let the labels on the edges coming into it be $p_1, p_1, \ldots, p_d$ $(d < d_i)$. If all the inputs that affect the states of these processors after the read of step i are fixed, and the inputs that affect the content of cell $(c, i-1)$ are fixed, then the content of cell (c, i) is fixed. Hence, for $i = 1, 2, \ldots, l$, we have

$$M_i \leq M_{i-1} + (d_i - 1) N_{i-1}. \tag{4}$$

Similarly, if all inputs that affect the state of the processor p before the read of step $i+1$ are fixed, and all the inputs 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 Z_{i-1} possible cells it could read in step i+1, we have

$$N_i \leq N_{i-1} + Z_{i-1}M_i. \tag{5}$$

The contents of an input cell and the state of a processor after the first read can be fixed by setting at most one bit of the input; hence $M_0, N_0 \leq 1$.

We now show by induction that, for $i = 0, 1, \ldots, l$, $M_i \leq 2^{i-1}d_i\Delta_{i-1}$ and $N_i \leq 2^i\Delta_i$. The basis case is trivial. For $i \geq 1$, we obtain from (4) and the induction hypothesis that

$$M_i \leq 2^{i-2}d_{i-1}\Delta_{i-2} + (d_i - 1)2^{i-1}\Delta_{i-1} \leq 2^{i-1}\Delta_{i-1} + (d_i - 1)2^{i-1}\Delta_{i-1} \leq 2^{i-1}d_i\Delta_{i-1},$$

and then from (5), the induction hypothesis and the bound shown above for Z_{i-1} that

$$N_i \leq 2^{i-1}\Delta_{i-1} + 2^{i-1}d_i\Delta_{i-1}^2 \leq 2^{i-1}\Delta_i + 2^{i-1}\Delta_i \leq 2^i\Delta_i.$$

We have thus shown that $Y_i, M_i, M_i, N_i \leq 2^i\Delta_i$. To prove part (a) of the lemma, it suffices to show that $2^{2^i}\Delta_i \leq d_i^2$, for $i = 0, 1, \ldots, l$. We use induction on i. The basis case is trivial, for $d_0 \geq 4$. For $i \geq 1$, we have, using the induction hypothesis and the fact that $d_i \geq d_{i-1}^4$, that

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

To obtain part (b), we observe that a processor in step $l+1$ can be in at most $Z_l(b)$ states, and can therefore appear as a label in at most $Z_l(b)$ edges of $G(b)$. The inequality follows from this since, by part (a), $Z_l(b) \leq d_l^2$. ∎

3 The Ackerman sequences

3.1 The Ackerman functions

The Ackerman functions are defined as follows.

$$\begin{aligned} A_1(x) &= 2x; \\ A_{i+1}(x) &= A_i^{(x)}(1). \end{aligned}$$

Here $A_i^{(x)}(1)$ is A_i applied x times to 1. That is, $A_2(x) = 2^x$ and $A_3(x) = \mathit{Tower}(x)$. The k-th inverse Ackerman function, I_k, is defined by $I_k(n) = \max\{i : A_k(i) \leq n\}$. 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, $\alpha(n)$, is given by

$$\alpha(n) = \min\{k : A_k(3) \geq n\}.$$

3.2 The Ackerman tree

We now construct certain sequences that we refer to as *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 tree defined inductively. Each edge of the tree has a label. We denote by $B_i(x)$ the biggest 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 collapsing the roots of the following x^5 trees of depth $i+1$.

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

Thus $B_{i+1}(x) = B_i^{(x^5)}(x^{10})$.

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

$$T_k(20),\ T_k(B_k(20)), \ldots, T_k(B_k^{(l-1)}(20)).$$

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

1. All leaves of the tree are at distance $k+1$ from the root.
2. The outdegree of the root is l.
3. The label on the leftmost edge of the root is 20.
4. If the label on the edge coming into a non-leaf node γ is d, then γ has d^5 children.
5. 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}.
6. 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 denote by Γ_i the set of nodes of $\Gamma(k,l)$ at distance i from the root. The the set of children of the node γ is denoted by $succ(\gamma)$. $T(\gamma)$ denotes the subtree rooted at γ. Let H be the set of leaves of the tree $\Gamma(k,l)$ and $H(\gamma)$ the leaves in $T(\gamma)$; thus, if γ is not a leaf, then $H(\gamma) = \bigcup_{\gamma' \in succ(\gamma)} H(\gamma')$. For $h \in H$, and $i = 1, 2, \ldots, k+1$, $d_i(h)$ is the label on the i-th edge from the root on the path connecting the root with the leaf h; we set $d_0(h) = 4$. We define

$$\begin{aligned} D(h) &= (d_0(h), d_1(h), d_2(h), \ldots, d_k(h)); \\ E[a,b](h) &= \prod_{i=a}^{b} (d_i(h))^5; \\ E(h) &= E[1,k](h). \end{aligned}$$

(As usual, $E[a,b] = 1$ if $a > b$.) The sequences $D(h)$ play a central role in our analysis. Note that the label $d_{k+1}(h)$ is ignored in the the definitions of $D(h)$ and $E(h)$. Let $d_i^+(h)$ be the next bigger value of d_i after $d_i(h)$, that is,

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

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

Observe that if $\gamma \in \Gamma_i$, then the values $d_0(h), d_1(h), \ldots, d_i(h)$ and $d_i^+(h)$ are the same for all leaves $h \in H(\gamma)$. We refer to these values as $d_0(\gamma), d_1(\gamma), \ldots, d_i(\gamma)$ and $d_i^+(\gamma)$ respectively. Similarly, if $a, b \leq i$, then $E[a,b](h)$ is constant over $H(\gamma)$; we refer to this value as $E[a,b](\gamma)$.

We now state the properties of $\Gamma(k,l)$ that we use in the proof of our lower bound. These properties will be used when we obtain $D(h)$-regularizing partial inputs in section 3.3 and prove the lower bound in section 4.

Lemma 5

(a) For $h \in H$ and $i = 1, 2, \ldots, k$, $d_i(h) \geq (d_{i-1}(h))^4$.

(b) $d_i^+(h) \geq (d_k(h))^{10}$.

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

(d) $\sum_{\gamma \in \Gamma_i} 1/d_i(\gamma) \leq 2^{-i}$.

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

(f) $\sum_h 1/E[1,k] = l$.

(g) $H(\Gamma(k,l)) \leq A(4k,1)$.

Proof.

(a) For $i \geq 2$, we have $d_i(h) \geq (d_{i-1}(h))^{10}$ and the claim follows easily. For $i = 1$, we have $d_1(h) \geq 20$ and $d_0(h) = 4$, and again the claim holds.

(b) We have, from our definitions, $d_i^+(h) \geq d_{k+1}(h) \geq (d_k(h))^{10}$.

(c) For $\gamma \in \Gamma_k$, the number of leaves in $H(\gamma)$ is exactly $(d_k(\gamma))^5$. Let $\gamma' \in \Gamma_k$ be the parent of h. By part (b), $d_k^+(\gamma) > d_k(\gamma)$; hence the number of the nodes in Γ_k up to γ' is at most $d_k(\gamma) = d_k(h)$. Thus the total number of leaves up to h is at most $d_k(h)(d_k(h))^5 = (d_k(h))^6$.

(d) It follows from part (a) that $\min\{d_i(\gamma) : \gamma \in \Gamma_i\} \geq 2^{i+1}$. Then, using part (b), we have

$$\sum_{\gamma \in \Gamma_i} \frac{1}{d_i^+(\gamma)} \leq 2 \cdot \frac{1}{\min\{d_i(\gamma) : \gamma \in \Gamma_i\}} \leq 2^{-i}.$$

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

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

(f) Using part (e), we have

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

(g) We first show by induction that, for $x \geq 5$, $B_i(x) \leq A_{3i}(x)$. For the basis case, we have $B_1(x) = x^{10}$ and $A_3(x) = Tower(x)$, and $Tower(x) \geq x^{10}$, for $x \geq 5$.

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

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

It follows from part (c) that the number of leaves in $\Gamma(k,l)$ is at most the biggest label appearing in $\Gamma(k,l)$. By our definitions, the biggest label in $\Gamma(k,l)$ is $B_k^{(l)}(20)$. We show

that for $l \geq 5$, $B_k^{(l)}(20) \leq A_{4k}(l)$. For $k = 1$, we have $B_4^{(l)}(20) = 20^{20l} \leq Tower(l) \leq A_4(l)$. For $k \geq 2$, we have another routine derivation.

$$\begin{aligned} A_{4k}(l) &\geq A_{4k-1}^{(l)}(1) \geq A_{4k-1}(A_{4k-1}^{(l-1)}(1)) \\ &\geq A_{4k-1}(20+l) && \text{since } l \geq 5 \\ &\geq A_{4k-2}^{(l)}(20) \geq A_{3k}^{(l)}(20) \\ &\geq B_k^{(l)}(20). \end{aligned}$$

■

3.3 Obtaining a $D(h)$-regularizing partial input

We shall now analyze the computation graph of algorithm A. Let h be a leaf of the tree $\Gamma(k,l)$, and consider the sequence $D(h)$ defined in section 3. We shall associate with h a $D(h)$-regularizing partial input $b(h)$. We next show how such a partial input with a small number of blockers and passers can be obtained.

The partial input $b(h)$ is produced in stages. The intermediate partial inputs produced will be called $b^0(h), b^1(h), \ldots, b^k(h)$. The partial input $b^i(h)$ will be $D(h)$-regularizing up to level i. In the end we shall set $b(h) = b^k(h)$. From now on we shall not mention 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 $b^0 = *^n$. Observe that 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 $G(b^{i-1})$. A free vertex v at level i will be called a *high degree* vertex if $d_i \leq f_v < d_i^+$; it will be called a *very high degree* vertex if $f_v \geq d_i^+$. To obtain b^i we consider these high and very high degree vertices one by one, and, if necessary, refine the partial input to deal with them. The temporary partial input produced will be denoted by b'; initially $b' = b^{i-1}$.

(A) *High Degree Vertices.* Consider a vertex v that had 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 variables in $affect(b', (p, i-1))$ to 0. 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 $affect(b', (p, i-1))$ to the value they have in x. This fixes the state of processor p in step i so that it writes to v; consequently, 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. Let the number of inputs set in step A of *STAGE i* be $S_A^i(h)$, and the number of inputs in step B of *STAGE i* be $S_B^i(h)$. This completes the description of *STAGE i*.

For brevity, we write $M_i(h), N_i(h), Y_i(h)$, and $Z_i(h)$ instead of $M_i(b(h)), N_i(b(h)), Y_i(b(h))$, and $Z_i(b(h))$; when the parameter h is clear from the context, we shall further simplify the notation by dropping it.

Lemma 6

(a)

$$S_A^i(h) \leq \sum_{v \in V_i : d_i \leq f_v(b^{i-1}) < d_i^+} f_v(b^{i-1}) d_{i-1}^2.$$

(b)

$$S_B^i(h) \leq \sum_{v \in V_i : d_i^+ \leq f_v(b^{i-1})} d_{i-1}^2.$$

Proof. First observe, using Lemma 5 (a), that the sequence $D(h)$ satisfies the conditions in Lemma 4. Now consider the processing of a high degree vertex v. Let the partial input when v is processed be b'. Since $b' \preceq b^{i-1}$, we have that $G(b')$ is $D(h)$-regularized up to level $i-1$, $f_v(b') \leq f_v(b^{i-1})$ and, for all processors p, $affect(b', (p, i-1)) \subseteq affect(b^{i-1}, (p, i-1))$. It follows that

$$S_A^i(h) \leq \sum_{v \in V_i : d_i \leq f_v(b^{i-1}) < d_i^+} f_v(b^{i-1}) N_{i-1}.$$

Part (a) of the lemma follows from this because $N_{i-1} \leq d_{i-1}^2$ by Lemma 4 (a).

Part (b) can be obtained similarly. ∎

We shall make use of the following observation: *The partial input $b^i(h)$ constructed by the above procedure depends only on $d_1(h), d_2(h), \ldots, d_i(h)$. 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 $b^i(\gamma)$. Similarly, $M_i(h)$, $N_i(h)$, $Y_i(h)$ and $Z_i(h)$ are constant over $H(\gamma)$; we denote the respective values by $M_i(\gamma)$, $N_i(\gamma)$, $Y_i(\gamma)$ and $Z_i(\gamma)$.*

4 The lower bound

In this section, we shall show that no algorithm can solve the unordered chaining problem in constant time using a linear number of processors. We shall make use of the partial input $b(h)$ described in the previous section. In our calculations the number of passers and blockers in $b(h)$ will play an important role; for brevity, we denote them by $Pa(h)$ and $Bl(h)$ instead of $Pa(b(h))$ and $Bl(b(h))$.

Theorem 1 *Let A be an algorithm that solves the unordered chaining problem for inputs of length n in k steps using P processors. Suppose $A_{4k}(5) \leq n$. Then $P = \Omega(nI_{4k}(n))$.*

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

Consider the partial input $b(h)$ associated with the leaf h. By Lemma 4, for each output vertex β_i, $|contents(b(h), \beta_i)| \leq (d_k(h))^2$. Using Lemma 2 and $E(h) \geq (d_k(h))^2$, we then get

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

implying

$$2(Bl(h) + \frac{Pa(h)}{E(h)} + 1) \geq \frac{n}{E(h)}.$$

Summing over all leaves h and using Lemma 5 (f), we get

$$2(\sum_h (Bl(h) + \frac{Pa(h)}{E(h)} + 1)) \geq nI_{4k}(n).$$

We shall show (Lemma 8 (b) and Lemma 9) that

$$\begin{aligned} \sum_h Bl(h) &\leq P; \\ \text{and } \sum_h Pa(h) &\leq 2P; \end{aligned}$$

Therefore, we have

$$6P + 2n \geq nI_{4k}(n),$$

that is, $P = \Omega(nI_{4k}(n))$. ∎

Corollary 7 *If A is an algorithm solving the unordered chaining problem with a linear number of processors, then A needs $\Omega(\alpha(n))$ time.* ∎

Lemma 8

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

$$\sum_h S_B^i(h) \leq P/2^i.$$

(b)

$$\sum_h \mathrm{Bl}(h) \leq P.$$

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

$$\sum_h Bl(h) \leq \sum_h \sum_{i=1}^{k} S_B^i(h) = \sum_{i=1}^{k} \sum_h S_B^i(h).$$

Thus part(b) of the lemma follows easily from part (a).

We now show part (a). By Lemma 6,

$$\sum_h S_B^i(h) \leq \sum_h \sum_{v \in V_i:\, d_i^+ \leq f_v(b^{i-1})} d_{i-1}^2 \leq \sum_{v \in V_i} \sum_{h:\, d_i^+ \leq f_v(b^{i-1})} d_{i-1}^2.$$

(To simplify the notation, we write d_i^+ for $d_i^+(h)$ and $f_v(b^{i-1})$ for $f_v(b^{i-1}(h))$.)

As observed in section 3.3, for $\gamma \in \Gamma_{i-1}$, b^{i-1} is constant over $H(\gamma)$. Therefore, we may group the h by the value of γ and obtain

$$\sum_h S_B^i(h) \leq \sum_{v \in V_i} \sum_{\gamma \in \Gamma_{i-1}} \sum_{h: h \in H(\gamma) \wedge d_i^+ \leq f_v(b^{i-1})} d_{i-1}^2.$$

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

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

where $S(v,\gamma) = \{h \in H(\gamma) : d_i^+(h) \leq f_v(b^{i-1}(h))\}$. Let h' be the rightmost leaf in $S(v,\gamma)$, then by Lemma 5 (c)

$$|S(v,\gamma)| \leq (d_k(h'))^6.$$

Then, using Lemma 5 (b), we have

$$f_v(b^{i-1}(h')) \geq d_i^+(h') \geq (d_k(h'))^{10} \geq \frac{1}{2}(d_{i-1}(\gamma))^{40} |S(v,\gamma)| \geq (d_{i-1}(\gamma))^5 |S(v,\gamma)|.$$

Therefore, $(d_{i-1}(\gamma))^2 |S(v,\gamma)| \leq f(b^{i-1}(\gamma))/(d_{i-1}(\gamma))^3$, and

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

Now, by Lemma 4 (b), $\sum_{v \in V_i} f_v(b^{i-1}(\gamma)) \leq P(d_{i-1}(\gamma))^2$. Therefore,

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

Then, using Lemma 5 (d), we get the required bound

$$\sum_h S_B^i(h) \leq P/2^i.$$

■

Lemma 9

$$\sum_h \frac{\mathrm{Pa}(h)}{E(h)} \leq 2P.$$

Proof. Using the observation in section 3.3, we write

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

It follows from Lemma 8 (a) that

$$\sum_h \sum_{i=1}^{k} \frac{S_B^i(h)}{E(h)} \leq P. \tag{6}$$

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

$$\sum_h \sum_{i=1}^{k} \frac{S_A^i(h)}{E(h)} \leq \frac{P}{2^i} \tag{7}$$

It follows that

$$\sum_h \sum_{i=1}^{k} \frac{S_A^i(h)}{E(h)} \leq P. \tag{8}$$

The lemma then follows by combining (6) and (8).

To prove (7), we use Lemma 6 and write

$$\frac{S_A^i(h)}{E(h)} \leq \sum_h \sum_{v \in V_i : d_i \leq f_v(b^{i-1}) < d_i^+} \frac{d_{i-1}^2 f_v(b^{i-1})}{E(h)} \leq \sum_{v \in V_i} \sum_h \frac{d_{i-1}^2 f_v(b^{i-1})}{E(h)},$$

where, as usual, d_i, d_i^+, b^{i-1} and d_{i-1} stand for $d_i(h), d_i^+(h), b^{i-1}(h)$ and $d_{i-1}(h)$ respectively. As in the proof of Lemma 8 we compute the rightmost sum by grouping the leaves by $\gamma \in \Gamma_{i-1}$. Then

$$\begin{aligned} \sum_h \frac{S_A^i(h)}{E(h)} &\leq \sum_{v \in V_i} \sum_{\gamma \in \Gamma_{i-1}} \sum_{h \in H(\gamma) : d_i \leq f_v(b^{i-1}) < d_i^+} \frac{d_{i-1}^2 f_v(b^{i-1})}{E(h)} \\ &\leq \sum_{v \in V_i} \sum_{\gamma \in \Gamma_{i-1}} [(d_{i-1}(\gamma))^2 f_v(b^{i-1}(\gamma))(\sum_{h \in H(\gamma) : d_i \leq f_v(b^{i-1}) < d_i^+} \frac{1}{E(h)})]. \end{aligned} \tag{9}$$

Fix $v \in V_i$ and $\gamma \in \Gamma_i$, and consider the rightmost sum

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

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

$$\sum_{h \in H(\gamma) : d_i \leq f_v(b^{i-1}) < d_i^+} \frac{1}{E(h)} \leq \sum_{\gamma' \in succ(\gamma)} [\delta(v, \gamma') \sum_{h \in H(\gamma')} \frac{1}{E(h)}],$$

where $\delta(v, \gamma') = 1$ if $d_i(\gamma') \leq f_v(b^{i-1}(\gamma')) < d_i^+(\gamma')$, and $\delta(v, \gamma') = 0$ otherwise. By Lemma 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 $\gamma' \in succ(\gamma)$. Therefore

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

Returning to (9), we now have

$$\sum_h \frac{S_A^i(h)}{E(h)} \leq \sum_{v \in V_i} \sum_{\gamma \in \Gamma_i} \frac{(d_{i-1}(\gamma))^2 f_v(b^{i-1}(\gamma))}{E[1, i-1](\gamma)} \leq \sum_{\gamma \in \Gamma_{i-1}} [\frac{(d_{i-1}(\gamma))^2}{E[1, i-1](\gamma)} (\sum_{v \in V_i} f_v(b^{i-1}(\gamma)))].$$

Using Lemma 4 (b), $\sum_{v \in V_i} f_v(b^{i-1}(\gamma)) \leq (d_{i-1}(\gamma))^2 P$. Thus

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

The inequality (9) follows from this using Lemma 5 (d). ∎

5 Reductions

Theorem 2 *An algorithm that solves prefix maxima on domain $\{1,\ldots,n\}$ with n processors requires time $\Omega(\alpha(n))$.*

Proof. We reduce the chaining problem to a prefix maxima problem on domain $\{1,\ldots,n\}$. On input $a_1,\ldots,a_n$, compute $c_1,\ldots,c_n$, $c_i = 0$ if $a_i = 0$ and $c_i = i$ if $a_i = 1$. Then solve prefix maxima for $c_1,\ldots,c_n$. Let $d_1,\ldots,d_n$ be the prefix maxima. It is easy to see that $b_1 = 0$ and $b_i = d_{i-1}$, $2 \leq i \leq n$, is the required solution to the chaining problem. ∎

Theorem 3 *There exists a constant c such that an algorithm that preprocesses for range maximum on domain $\{1,\ldots,n\}$ with n processors so that a single processor can answer a query in $c\alpha(n)$ steps requires $\Omega(\alpha(n))$ time.*

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

Theorem 4 *Parenthesis matching with nesting level requires $\Omega(\alpha(n))$ time, even when the depth of nesting is at most 2.*

Proof. Given an input to the chaining problem, 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 original 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, Dwork, Pippenger and Wigderson [15] and Chaudhuri [11]. The techniques used in this paper have since been sharpened and applied to several other problems. In Chaudhuri [13], they have been used to obtain strong lower bounds for the problem of *approximate compaction*. In Chaudhuri [14], 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, Matias and Vishkin [7] 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 [26], gives a constant time randomized chaining algorithm that works if the number of 1's 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.

Acknowledgment

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 [12]. We thank Magnús Halldórsson for his comments on the presentation in this paper.

References

[1] N. Alon and J. Spencer, "The Probabilistic Method", John Wiley and Sons Inc., New York, 1992.

[2] R. Boppana, "Optimal Separations Between Concurrent Write Parallel Machines", *Proc. of the 21st ACM STOC*, 1989, 320–326.

[3] O. Berkman, D. Breslauer, Z. Galil, B. Scheiber and U. Vishkin, "Highly Parallelizable Problems", *Proc. of 21st ACM STOC*, 1989, 309–319.

[4] P. C. P. Bhatt, K. Diks, T. Hagerup, V. C. Prasad, T. Radzik and S. Saxena, "Improved Deterministic Parallel Integer Sorting", *Information and Computation 94*, 1991, 643–670.

[5] P. Beame and J. Håstad, "Optimal Bounds for Decision Problems on the CRCW PRAM", *Proc. of the 19th ACM STOC*, 1987, 83–93.

[6] O. Berkman, J. JáJá, S. Krishnamurthy, R. Thurimella and U. Vishkin, "Some Triply-Logarithmic Parallel Algorithms", *Proc. of 31st FOCS*, 1990, 871–881.

[7] O. Berkman, Y. Matias and U. Vishkin, "Randomized Range-Maxima in Nearly-Constant Parallel Time", *Computational Complexity*, 2, 1992, 350–373.

[8] O. Berkman and U. Vishkin, "Recursive Star-Tree Parallel Data Structure", *Proc. of 30th IEEE FOCS*, 1989, 196–202.

[9] A. K. Chandra, S. Fortune and R. J. Lipton, "Unbounded Fan-in Circuits and Associative Functions", *Proc. of the 15th ACM STOC*, 1983.

[10] A. K. Chandra, S. Fortune and R. J. Lipton, "Lower bounds for Constant Depth Circuits for Prefix Problems", *Proc. of the 10th Intl. Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, Springer-Verlag, 1983.

[11] S. Chaudhuri, "Tight Bounds on the Chaining Problem", *Proc. of 3rd ACM SPAA*, 1991, 62–70.

[12] S. Chaudhuri and J. Radhakrishnan, "The Complexity of Parallel Prefix Problems on Small Domains", *Proc. 33rd IEEE FOCS*, 1992, 638–647.

[13] S. Chaudhuri, "A Lower Bound for Linear Approximate Compaction", *Proc. of 2nd Israel Symp. on Theory of Comp. and Sys.*, 1993, 25–32.

[14] S. Chaudhuri, "Sensitive Functions and Approximate Problems", *Proc. of 34th IEEE FOCS*, 1993, (to appear).

[15] D. Dolev, C. Dwork, N. Pippenger and A. Wigderson, "Superconcentrators, Generalizers and Generalized Connectors with Limited Depth", *Proc. of the 15th ACM STOC*, 1983, 42–51.

[16] F. E. Fich, A. Wigderson and P. Ragde, "Simulations Among Concurrent-Write Models of Parallel Computation", *Algorithmica* 3, 1988, 43–51.

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

[18] J. Gil and L. Rudolph, "Counting and Packing in Parallel", *International Conference on Parallel Processing*, 1986, 1000-1002.

[19] T. Hagerup, "The Log-Star Revolution", *Proc. 9th Symposium on Theoretical Aspects of Computer Science* (1992), Springer Lecture Notes in Computer Science, Vol. 577, 259–278.

[20] J. JáJá. "An Introduction to Parallel Algorithms. Addison-Wesley, Reading, MA, 1992.

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

[22] Y. Matias and U. Vishkin, "On Parallel Hashing and Integer Sorting,*Proc. of 17th ICALP*, 1990, 729–743.

[23] F. Meyer auf der Heide and A. Wigderson, "The complexity of parallel sorting", *SIAM Journal on Computing*, v16, No 1, 1987, 100-107.

[24] I. Newman, P. Ragde and A. Wigderson, "Perfect Hashing, Graph Entropy and Circuit Complexity", *Proc. of 5th Ann. Conf. on Structure in Complexity Theory*, 1990, 91–99.

[25] P. Ragde, "The Parallel Simplicity of Compaction and Chaining", *Proc. 17th ICALP*, 1990, 744–751.

[26] R. Raman, "The Power of Collision: Randomized Parallel Algorithms for Chaining and Integer Sorting", *10th FST & TCS Conf.*, 1990, LNCS 472, 161–175.

[27] M. Snir, "On Parallel Searching", *SIAM J. of Computing, vol. 14, no. 2*, 1985, 688–708.