FMW87.pdf

LOWER BOUNDS FOR PARALLEL RANDOM-ACCESS MACHINES WITH UNBOUNDED SHARED MEMORY

Faith E. Fich, Friedhelm Meyer auf der Heide and Avi Wigderson

ABSTRACT

A parallel random-access machine (PRAM) consists of n synchronized processors and a shared memory. In this paper, we allow an infinite number of memory cells. The processors have arbitrary computational power and read and write access to the shared memory. In the literature, conflicts arising when several processors want to write into the same cell are resolved by various rules. We consider two such rules. In the COMMON PRAM, these processors have to write the same value and, in the PRIORITY PRAM, the processor with the smallest index succeeds. No lower bounds were previously known for these models. In this paper we prove two lower bounds.

Advances in Computing Research, vol. 4, pages 1–15
Copyright © 1987 JAI Press Inc.
All rights of reproduction in any form reserved
ISBN: 0-89232-682-4

First, we show that the COMMON PRAM requires $\Omega(\log\log\log(n))$ steps to solve the element distinctness problem on n integers. As this can be done in constant time by a PRIORITY PRAM, this lower bound implies a separation between these two models.

Second, the PRIORITY PRAM is shown to require $\Omega(\log\log(n))$ steps to compute the maximum of n integers, matching an upper bound due to Shiloach and Vishkin. This is the first tight result for this model. The lower bound generalizes a similar lower bound, obtained by Valiant, for parallel comparison trees with n processors.

Our proof technique uses Ramsey theoretic arguments. For this reason, we need the input integers to be large ($2^{O(n\log\log(n))}$ bits). Both problems remain open for small inputs (e.g., $O(\log(n))$ bits).

1. INTRODUCTION

The parallel random-access machine (PRAM) is an important and widely used model of parallel computation. A PRAM consists of n synchronized processors $P_1, \ldots, P_n$ which have read and write access to a shared memory. In this paper we assume that there are an infinite number of shared memory cells, all initially containing the value 0. Every time step consists of three successive phases. In the compute phase, each processor may perform some local computation of arbitrary complexity. In the write phase, each processor may write a value to an arbitrary shared memory cell. In the read phase, each processor may read from an arbitrary shared memory cell.

A concurrent-write PRAM is one which allows several processors to write simultaneously into the same memory cell. In this case, we must adopt a convention to resolve the conflict; namely, we must define the value in the cell after this write phase. Two popular methods of conflict resolution used in the literature are the following. The COMMON PRAM [10] requires that all processors which simultaneously access the same cell have to write the same value. For example, the parallel algorithms described in [5], [9], and [13] use this model. In the PRIORITY PRAM [6], the value in a cell will be the one written by the processor with the lowest index among those which write to this cell. This model is used in [14] and [1].

Let N denote the set of positive integers. At the beginning of a PRAM computation, each processor P_i contains an input variable x_i. We shall say that the PRAM computes a function $f: \mathbf{N}^n \to \mathbf{N}$ if, after the computation ends, the value $f(x_1, \ldots, x_n)$ is stored in some fixed shared memory cell.

Until now, all lower bounds on these models were obtained under one of the following two restrictions: *restricted communication width* [3, 16], where it is assumed that the size of the shared memory is bounded, or

restricted computational power [4, 11], where it is assumed that, at each step, each processor can only perform one operation from a small set of arithmetic operations.

In this paper we do not impose either of these restrictions. We allow an infinite number of shared memory cells. We also allow each processor to compute, in one step, any function of the information in its local memory.

Lower bounds for these strong models relate purely to the amount of interprocessor communication required to compute a given function. In this context, the question of whether PRIORITY and COMMON PRAMs are equivalent is particularly interesting, as it pertains to the power of communicating using different write conflict resolution mechanisms. Intuitively, in the PRIORITY PRAM, the hardware itself resolves write conflicts by picking the processor with lowest index to succeed. In the COMMON PRAM, the program has to ensure that two different values are never simultaneously written into the same cell.

Our first lower bound provides a separation between these two models. For this purpose, we consider the element distinctness problem. We show that a COMMON PRAM requires $\Omega(\log\log\log(n))$ steps to solve this problem, whereas a PRIORITY PRAM can do it in constant time. The reason for this difference in power is that indirect addressing can be used in a very powerful way by a PRIORITY PRAM, but only in a much weaker way by a COMMON PRAM. The same separation can be obtained between the COMMON PRAM and two other models, the ARBITRARY PRAM and the COLLISION PRAM [3]. These models have weaker write conflict resolution schemes than the PRIORITY PRAM, but they are still strong enough to solve the element distinctness problem in constant time.

Surprisingly, a simplified version of the proof technique used for the above result can be applied to obtain a lower bound for PRIORITY PRAMs. A PRIORITY PRAM needs $\Omega(\log\log(n))$ steps to compute the maximum of n integers. This result matches an upper bound by Shiloach and Vishkin [13], making our result the first tight lower bound in this model. A previous lower bound for this problem is due to Valiant [15]. He considered parallel comparison trees with n processors and obtained the same bound. Our result shows that the computation of this function cannot be sped up by either the ability to indirectly access the memory or the ability to compute arbitrary functions (instead of simply comparing inputs).

An essential part of our proof is simplifying the structure of the computation using Ramsey theoretic arguments, by successively restricting the input domain. Therefore our proofs require that the inputs are taken from a large enough domain (e.g., each input integer can have $2^{O(n\log\log(n))}$ bits). The complexity of the two problems for a small input domain remains open.

2. UPPER BOUNDS

The *element distinctness problem* on n variables is to compute the function $\delta_n: \mathbf{N}^n \to \{0, 1\}$, where $\delta_n(x_1, \ldots, x_n) = 0$ if and only if $x_i = x_k$ for some $i \neq k$. Initially, each of the n processors P_i has one integer x_i. The processors have to determine whether all of these integers are distinct or whether two of them are the same.

We first show how to compute δ_n in constant time on a PRIORITY PRAM. In fact, this algorithm also works on an ARBITRARY or COLLISION PRAM. We assume that the output appears in a special shared memory cell called *answer*.

Algorithm 1

Processor P_1 writes 1 into *answer*.
For all i, processor P_i writes i into memory cell x_i.
For all i, processor P_i reads memory cell x_i.
For all i, if processor P_i does not read the value i, then it writes 0 into *answer*.

This algorithm uses three steps to compute δ_n.

To determine that $\delta_n(x_1, \ldots, x_n) = 1$, an algorithm must verify that all the inequalities $x_i \neq x_k$ are true. We now consider two algorithms for computing δ_n on a COMMON PRAM that collect this information in different ways. In the lower-bound proof in the next section, we show that these are essentially the only ways this information can be collected. Both algorithms require $O(\log(n))$ steps. For simplicity, it is assumed that n is a power of 2 in the description of the algorithms.

The idea of Algorithm 2 is that the processors can communicate the values of their input variables to a single processor, which then does all of the necessary comparisons locally. The pattern of communication among processors is a binary tree.

Algorithm 2

Processor P_n accumulates the values of the variables $x_1, \ldots, x_n$ as follows:
  If $n > 1$, then
    in parallel,
      processor $P_{n/2}$ recursively accumulates the values of the variables $x_1, \ldots, x_{n/2}$ and
      processor P_n recursively accumulates the values of the variables $x_{(n/2)+1}, \ldots, x_n$
    processor $P_{n/2}$ writes an encoding of the values of the variables $x_{(n/2)+1}, \ldots, x_n$ into a memory cell
    processor P_n reads from that memory cell.
  Processor P_n computes $\delta_n(x_1, \ldots, x_n)$ and writes it into *answer*.

The computational power of the processors allows each processor to compute an encoding in one step. Once it has accumulated all inputs, a processor can compute the output in one step. Algorithm 2 uses $O(\log(n))$ steps and can be performed on an exclusive-read-exclusive-write PRAM. In fact, this algorithm shows that every function $f: \mathbf{N}^n \to \mathbf{N}$ can be computed in $O(\log(n))$ steps. A straightforward analysis shows that, during the tth time step, $n2^{t-2}$ new inequalities are verified. Notice that this number increases as the computation progresses.

The next algorithm does not use any of the computational power of the processors, but it uses the concurrent-write ability of COMMON PRAMs in a fundamental way. Essentially, we divide the variables into two groups, check that the groups have no element in common, and recursively check element distinctness for these two groups in parallel.

Algorithm 3

Processor P_1 writes 1 into *answer*.
If $n > 1$, then
  for $1 \leq i \leq n/2$, processor P_i writes 1 into memory cell x_i
  for $n/2 < i \leq n$, processor P_i reads cell x_i and, if its value is 1, writes 0 into *answer*
  for $1 \leq i \leq n$, processor P_i reads *answer*
  if *answer* $\neq 0$, then, in parallel,
  processors $P_1, \ldots, P_{n/2}$ compute $\delta_{n/2}(x_1, \ldots, x_{n/2})$ and
  processors $P_{(n/2)+1}, \ldots, P_n$ compute $\delta_{n/2}(x_{(n/2)+1}, \ldots, x_n)$.

This algorithm also uses $O(\log(n))$ steps. But, in this case, the number of new inequalities verified in one step decreases as the computation proceeds; in step t, $n^2/2^{t+1}$ inequalities are verified.

In the lower-bound proof presented in the next section, we distinguish between information received via direct storage access (as in Algorithm 2) and via indirect storage access (as in Algorithm 3).

3. A LOWER BOUND FOR THE COMMON PRAM

In this section we prove the following result:

THEOREM 1. *A COMMON PRAM requires* $\Omega(\log\log\log(n))$ *steps to compute* δ_n.

Together with Algorithm 1, this implies an $\Omega(\log\log\log(n))$ separation between COMMON and PRIORITY PRAMs.

In order to prove this theorem, we first introduce a variant of the COMMON PRAM which we call the m-read COMMON PRAM. These models differ in two respects. Firstly, no processor may write into a cell where any processor has already written. This restriction is essential for our lower-bound proof. In compensation, in the read phase of each step of an m-read COMMON PRAM, each processor is allowed to read up to m memory cells in parallel, instead of just one. This will guarantee that, for large enough m, an m-read COMMON PRAM is not weaker than a COMMON PRAM.

LEMMA 1. *T steps of a COMMON PRAM can be simulated by an m-read COMMON PRAM in T steps, if $m \geq T$.*

PROOF. Let M be a COMMON PRAM executing T steps. We modify M to obtain a T-read COMMON PRAM M^* as follows. We subdivide the infinite shared memory of M into T infinite parts. If a processor of M writes to cell w at step t, then, at step t, the corresponding processor of M^* writes to the wth cell of the tth part of memory. This ensures that no processor writes into a previously accessed cell. When a processor of M reads cell r, the corresponding processor M^* reads the rth cell of each part of the shared memory. Because each processor of M^* reads (among other things) the value read by the corresponding processor of M, we have shown that M^* simulates M. □

Theorem 1 now follows directly from the next lemma, taking m to be $\log\log(n)$.

LEMMA 2. *An m-read COMMON PRAM needs $\Omega(\log\log\log(n) - \log\log(m))$ steps to compute δ_n.*

PROOF. This proof is an adversary argument. As the computation proceeds, the adversary fixes the value of certain variables and maintains a set of allowed inputs such that, after each step, each processor only knows one live variable (i.e., a variable whose value has not been fixed). The precise meaning of the statement "processor P_i only knows x_j after step t" is that for every possible value a of x_j, the configuration of P_i after step t is the same for all allowed inputs $(b_1, \ldots, b_n)$ with $b_j = a$. □

Consider the situation arranged by the adversary after step t of some algorithm. We use $V_t \subseteq \{1, \ldots, n\}$ to denote the set of indices of live variables and $[V_t]^2$ to denote the set of all unordered pairs of elements from V_t. The set $E_t \subseteq [V_t]^2$ describes those pairs of live variables which the adversary has declared to be distinct. The graph $G_t = (V_t, E_t)$ with vertex set V_t and edge set E_t is called the distinctness graph. Live variables are restricted, by the adversary, to take values from an infinite subset $S_t \subseteq \mathbf{N}$. The indexed set $F_t = \{a_i \mid i \in \{1, \ldots, n\} - V_t\}$, describes the adversary's assignment of values to the fixed variables. These values are distinct elements of $\mathbf{N} - S_t$.

The set $I(V_t, E_t, S_t, F_t)$ of allowed inputs consists of all n-tuples $(b_1, \ldots, b_n)$ satisfying the following properties:

1. $b_i = a_i$ for all $i \in \{1, \ldots, n\} - V_t$.
2. $b_i \in S_t$ for all $i \in V_t$.
3. $b_i \neq b_j$ for all $i, j \in V_t$ with $\{i, j\} \in E_t$.

For two disjoint sets A and B, let $K(A, B)$ denote the complete bipartite graph on A and B. Let $G = (V, E)$ be a simple graph. A family $C = \{(A_l, B_l) \mid l = 1, \ldots, r\}$ with $A_l \cap B_l = \emptyset$ is a bipartite cover of G if every edge $\{i, j\} \in E$ belongs to some graph $K(A_l, B_l)$. The size $s(C)$ of C is $\sum_{l=1}^{r} (|A_l| + |B_l|)$. The bipartite complexity of G is defined to be $\beta(G) = \min\{s(C) \mid C \text{ is a bipartite cover of } G\}$. An upper bound of $q\lceil\log(q)\rceil$ is easy to obtain for the bipartite complexity of K_q, the complete graph on q vertices. The following lower bound is due to Hansel [8] and Pippenger [12].

THEOREM 2. $\beta(K_q) \geq q \log(q)$.

We shall measure the "complexity" of a set of allowable inputs in terms of the number of live variables and the bipartite complexity of the distinctness graph.

MAIN LEMMA. *Consider an m-read COMMON PRAM. Assume that, before step t, the set of allowed inputs is $I(V_{t-1}, E_{t-1}, S_{t-1}, F_{t-1})$ and each processor P_i only knows one variable x_{j_i} with $j_i \in V_{t-1}$. Then, the adversary can define a new set of allowed inputs $I(V_t, E_t, S_t, F_t) \subseteq I(V_{t-1}, E_{t-1}, S_{t-1}, F_{t-1})$ such that, after step t, the following properties are satisfied:*

1. *$V_t \subseteq V_{t-1}$ and $|V_t| \geq |V_{t-1}|^2/(|V_{t-1}| + 2nm)$.*
2. *Each processor P_i knows exactly one variable with index in V_t.*
3. *$S_t \subseteq S_{t-1}$ and S_t is infinite.*
4. *$E_t \supseteq E_{t-1} \cap [V_t]^2$ and $\beta(G_t) \leq \beta(G_{t-1}) + n(t + m)$, where $G_t = (V_t, E_t)$.*
5. *$F_{t-1} \subseteq F_t \subseteq \mathbf{N} - S_t$.*

We now complete the proof of Lemma 2. Before the computation starts (i.e., after step 0), $V_0 = \{1, \ldots, n\}$, $E_0 = \emptyset$, $S_0 = \mathbf{N}$, and $j_i = i$ together satisfy

the conditions of the Main Lemma. Suppose that the computation terminates after T steps. Then $G_T = K_{|V_T|}$, the complete graph on V_T; otherwise there would be two allowed inputs, one in which all the elements are distinct and another containing two equal elements, between which the algorithm could not distinguish, although they have different images under δ_n. Thus, by Theorem 2, $\beta(G_T) = \beta(K_{|V_T|}) \geq |V_T| \log(|V_T|)$.

From condition 4, we get that $\beta(G_T) \leq n \sum_{t=1}^{T} (t+m) + \beta(G_0) \leq nT(T+m)$. Since $|V_{t-1}| \leq n$, it follows from condition 1 that

$$|V_t| \geq \frac{|V_{t-1}|^2}{|V_{t-1}| + 2nm} \geq \frac{|V_{t-1}|^2}{3nm}.$$

By induction, this implies

$$|V_T| \geq \frac{|V_0|^{2^T}}{(3nm)^{2^T-1}} \geq \frac{3n}{(3m)^{2^T}}.$$

Combining these inequalities, we get

$$nT(T+m) \geq \frac{3n}{(3m)^{2^T}} \log\left[\frac{3n}{(3m)^{2^T}}\right].$$

Thus $T = \Omega(\log\log\log(n) - \log\log(m))$. □

To prove the Main Lemma, we first state three results: two "Ramsey-like" and one graph-theoretical. They will be extensively used in the proof.

LEMMA 3. *Let f: $W \to D$ be any function defined on an infinite domain W. Then there exists an infinite subset $W' \subseteq W$ such that $f|_{W'}$ is either constant or 1-1. In particular, if D is finite, then $f|_{W'}$ is constant.*

LEMMA 4. *Let f, g: $W \to D$ be two functions defined on an infinite domain W. Then there exists an infinite subset $W' \subseteq W$ such that $f|_{W'}$ and $g|_{W'}$ are either identical or have disjoint ranges.*

PROOF. Suppose that there is no infinite subset of W on which f and g are identical. Let $W' \subseteq W$ be an infinite subset of elements on which f and g disagree. By Lemma 3, we may assume that $f|_{W'}$ and $g|_{W'}$ are constant or 1-1. If either function is constant, then they must have disjoint ranges. Therefore, suppose that both functions are 1-1. Consider the graph (W', E), where $\{w_1, w_2\} \in E$ if and only if $f(w_1) = g(w_2)$ or $g(w_1) = f(w_2)$. Because f and g are 1-1, each vertex has degree at most 2. Hence, there is a 3 coloring of the graph. Let W'' be an infinite monochromatic subset of W'. Then, by construction, $f|_{W''}$ and $g|_{W''}$ have disjoint ranges. □

LEMMA 5. *Let $H(U, L)$ be a finite graph, and let $\alpha(H)$ denote the size of a maximum independent set in H. Then $\alpha(H) \geq |U|^2/(|U| + 2|L|)$.*

The proofs of Lemmas 3 and 5 can be found in [7, p. 112] and [2, p. 282], respectively.

PROOF OF THE MAIN LEMMA. Consider the sequence of writes performed by some processor P_i up to and including step t. In step $p \leq t$, P_i decides whether or not to write according to some predicate d_i^p. If it writes, it writes a value v_i^p to a cell w_i^p. Now consider the m reads P_i executes in step t. It reads from cells $r_{i,1}, \ldots, r_{i,m}$. Since P_i only knows x_{j_i}, it follows that d_i^p, v_i^p, w_i^p, and $r_{i,h}$ are functions of only this one variable. Thus d_i^p: $S_{t-1} \to \{0, 1\}$ and v_i^p, w_i^p, $r_{i,h}$: $S_{t-1} \to N$. Note that the reads are executed in parallel. Therefore, no processor can use the information obtained in one read to determine the other read functions it uses in the same step. □

Our adversary simplifies this structure by restricting the set of allowed inputs.

CLAIM 1. Without loss of generality, at every step $p \leq t$, each processor either writes for all allowed inputs or does not write for any allowed input.

PROOF. Apply Lemma 3 successively to d_i^p for $i = 1, \ldots, n$ and $p = 1, \ldots, t$ to obtain an infinite subset $S' \subseteq S_{t-1}$. The claim follows when the adversary restricts the set of allowed inputs to be $I(V_{t-1}, E_{t-1}, S', F_{t-1})$. □

We use the term *address function* to denote any read function $r_{i,h}$ used at step t or any write function w_i^p used at step $p \leq t$.

CLAIM 2. Without loss of generality, every address function is either constant or 1-1.

PROOF. Apply Lemma 3 successively to all address functions. The result is an infinite subset $S'' \subseteq S'$ on which every address function is either constant or 1-1. The adversary restricts the set of allowed inputs to be $I(V_{t-1}, E_{t-1}, S'', F_{t-1})$. □

CLAIM 3. Without loss of generality, if P_i and P_k access the same cell then they use the same address function.

PROOF. Apply Lemma 4 successively to every pair of address functions. The result is an infinite subset $S''' \subseteq S''$ such that every pair of address

functions (which we now consider to be functions on S^m) are either identical or have disjoint ranges. The adversary therefore restricts the set of allowed inputs to be $I(V_{t-1}, E_{t-1}, S^m, F_{t-1})$. □

The next observation depends on the the fact that we are using an m-read COMMON PRAM. It is the only part of the proof of the Main Lemma which does so.

CLAIM 4. Without loss of generality, if P_i and P_k know the same variable (that is, $j_i = j_k$), then $w_i^p \neq r_{k,h}$ for all $p = 1, \ldots, t$ and $h = 1, \ldots, m$. In particular, $w_i^t \neq r_{k,h}$.

PROOF. We may assume that every processor has copies of the programs of all other processors. Then P_k can compute $w_i^p(x_{j_i})$, the address P_i writes to in step p, and $v_i^p(x_{j_i})$, the value P_i writes at that step. By the definition of an m-read COMMON PRAM, all processors writing into this cell at time p must write the same value and thereafter that value is never changed. Thus cell $w_i^p(x_{j_i})$ will still contain the value $v_i^p(x_{j_i})$ when P_k reads it. Since P_k can compute this value, it does not have to read cell $w_i^p(x_{j_i})$. □

Notice that if processors P_i and P_k know different variables, then P_k may learn information by reading a cell P_i has written into. Furthermore, as shown in Algorithm 1, the argument given in the proof of Claim 4 is completely fallacious for the PRIORITY PRAM.

Equipped with these claims, our adversary is ready to continue. First, the 1-1 address functions are handled by determining a graph of low bipartite complexity and adding its edges to the distinctness graph. Following this, the adversary deals with the constant address functions by fixing some of the live variables.

1-1 Address Functions

Suppose that $g_1, \ldots, g_r : S^m \to \mathbf{N}$ are the 1-1 address functions. For each g_l, define

$$A_l = \{q \in V_{t-1} \mid x_q = x_{j_i} \text{ and } w_i^p = g_l \text{ for some } i \in \{1, \ldots, n\} \text{ and } p \leq t\}$$

and

$$B_l = \{q \in V_{t-1} \mid x_q = x_{j_i} \text{ and } r_{i,h} = g_l \text{ for some } i \in \{1, \ldots, n\} \text{ and } h \in \{1, \ldots, m\}\}.$$

Intuitively, $q \in A_l$ if a processor that knows x_q uses g_l as a write function in some step $p \leq t$. Similarly, $q \in B_l$ if some processor that knows x_q uses g_l as one of its read functions in step t.

The next result is a corollary to Claim 4.

CLAIM 5. $A_l \cap B_l = \emptyset$.

Since every processor can contribute at most t times to the A's and at most m times to the B's, $\sum_{l=1}^{r} (|A_l| + |B_l|) \leq n(t + m)$. Let $G' = (V_{t-1}, E')$ be the graph with $E' = \bigcup_{l=1}^{r} K(A_l, B_l)$ and let $G'' = (V_{t-1}, E'')$ be the graph with $E'' = E_{t-1} \cup E'$. Then $\beta(G') \leq n(t + m)$. This implies the following result.

CLAIM 6. $\beta(G'') \leq \beta(G) + n(t + m)$.

CLAIM 7. For inputs in $I(V_{t-1}, E'', S^m, F_{t-1})$, we may assume, without loss of generality, that no processor uses a 1-1 read function in step t.

PROOF. Suppose processor P_k uses a 1-1 read function, that is, $r_{k,h} = g_l$ for some $h \in \{1, \ldots, m\}$ and $l \in \{1, \ldots, r\}$. If the same cell was written into by some processor P_i at time $p \leq t$, then $w_i^p(x_{j_i}) = r_{k,h}(x_{j_k})$. Claim 3 implies that $w_i^p = r_{k,h} = g_l$. Now $j_i \in A_l$, $j_k \in B_l$, and $\{j_i, j_k\} \in E' \subseteq E''$. It follows from the definition of $I(V_{t-1}, E'', S^m, F_{t-1})$ that $x_{j_i} \neq x_{j_k}$. This contradicts the fact that g_l is 1-1. Thus, every processor that uses a 1-1 read function reads the initial contents of the cell, namely 0. Since the read imparts no information about any input known to be in $I(V_{t-1}, E'', S^m, F_{t-1})$, we may assume, without loss of generality, that the read was not performed. □

Constant Address Functions

From this point on, we assume that the set of allowed inputs is $I(V_{t-1}, E'', S^m, F_{t-1})$.

CLAIM 8. If w is accessed by a constant address function, then the contents of cell w after step t depends on at most one variable from V_{t-1}.

PROOF. To prove this claim, we do not need any of the properties of the m-read COMMON PRAM. In fact, for cells which are written via constant address functions, the PRIORITY write conflict resolution scheme can be used. We can also allow these cells to be written into more than once. Consider the last step $p \leq t$ in which cell w was accessed for writing on an allowed input. Let P_i be the processor with lowest index writing into cell w at step p on an allowed input. By Claim 1, P_i always writes to cell w at step p. Therefore, the contents of cell w can only depend on x_{j_i}. □

Consider the graph $H(V_{t-1}, L)$, where $\{i, k\} \in L$ if some processor knowing x_i uses a constant read function to access cell w in step t, and the contents of cell w depends on x_k. Thus, after step t, this processor "knows" both x_i and x_k. Note that $|L| \leq mn$, the maximum number of reads that can be performed in step t. Apply Lemma 5 to this graph to obtain an independent set of vertices $V_t \subseteq V_{t-1}$ such that $|V_t| \geq |V_{t-1}|^2/(|V_{t-1}| + 2nm)$. The adversary restricts the set of allowable inputs to be $I(V_t, E'', S''', F_{t-1})$ and we get the following result.

CLAIM 9. After step t, every processor knows at most one variable in V_t.

We now complete the proof of the Main Lemma. Those processors which do not know any variable in V_t can be assigned an arbitrary one. The adversary fixes the variables x_i with $i \in V_{t-1} - V_t$ and assigns them distinct values $a_i \in S'''$. These values are added to F_{t-1} to obtain F_t and are removed from S''' to obtain S_t. Finally, let $G_t = (V_t, E_t)$ be the subgraph of G'' induced by V_t. Then $\beta(G_t) \leq \beta(G'')$. □

4. A LOWER BOUND FOR THE PRIORITY PRAM

In this section, we apply a simplified version of the proof method from the last section to obtain a lower bound for PRIORITY PRAMs.

THEOREM 3. *A PRIORITY PRAM requires $\Omega(\log\log(n))$ steps to compute the maximum of n numbers.*

PROOF. In contrast to solving the element distinctness problem, computing the maximum of n numbers does not become easier if these numbers are known to be distinct. Therefore, we will assume that they are distinct, because this will considerably simplify the treatment of 1-1 address functions. Essentially, in the context of the previous lower bound proof, such functions are always useless.

More formally, let M be a PRIORITY PRAM which finds the maximum of n numbers. Let V_t, S_t, and F_t be defined as in the previous section. However, the adversary's set of allowed inputs is now defined to be $J(V_t, S_t, F_t) = I(V_t, [V_t]^2, S_t, F_t)$. Specifically, all input values are required to be distinct. The following lemma plays the same role as the Main Lemma in the previous section. □

LEMMA 6. *Assume that, before step t, the set of allowed inputs is $J(V_{t-1}, S_{t-1}, F_{t-1})$ and that each processor only knows one variable with index in V_{t-1}. Then the adversary can define a new set of allowed inputs $J(V_t, S_t, F_t)$ such that after step t the following properties are satisfied:*

1. *$V_t \subseteq V_{t-1}$ and $|V_t| \geq |V_{t-1}|^2/(|V_{t-1}| + 2n)$.*
2. *Each processor knows only one variable with index in V_t.*
3. *$S_t \subseteq S_{t-1}$ and S_t is infinite.*
4. *$F_{t-1} \subseteq F_t \subseteq \mathbf{N} - S_t$.*

If M requires T steps then $|V_T| \leq 1$; otherwise no processor can determine the output. From Lemma 6 and the fact that $|V_0| = n$, we can derive $T = \Omega(\log\log(n))$. This concludes the proof of Theorem 3.

PROOF OF LEMMA 6. Consider the proof of the Main Lemma for $m = 1$. It is sufficient to show that, for $J(V_t, S_t, F_t)$, the analogue of Claim 7 (which takes care of 1-1 address functions) and the analogue of Claim 9 (which takes care of constant address functions) hold.

The proof of Claim 9 does not use the properties of the m-read COMMON PRAM. Since $J(V_t, S_t, F_t) \subseteq I(V_t, E_t, S_t, F_t)$, the analogous result for the PRIORITY PRAM holds. Similarly, Claim 3, used in the proof of Claim 7, is true.

However, the proof of Claim 7 also depends on Claim 4, which is not true for the PRIORITY PRAM. Fortunately, the rest of the proof of the analogue of Claim 7 can be derived from the fact that all input variables are assumed to have distinct values. Specifically, since g_t is 1-1, the values x_{j_i} and x_{j_k} are equal, and hence $j_i = j_k$. Thus processor P_k either reads the initial cell contents, 0, or reads a value written by a processor that knows the same variable. Because P_k can compute the cell contents for any input known to be in $J(V_{t-1}, S''', F_{t-1})$, it does not have to perform the read. □

5. LOWER BOUNDS ON FINITE INPUT DOMAINS

In the previous sections, we proved Theorems 1 and 3 under the assumption that the input integers can be arbitrarily large. An explicit bound on the size of the input domain required can be obtained with slightly modified versions of Lemmas 3 and 4. These result are easy to prove using the pigeonhole principle.

LEMMA 3'. *Let $f: W \to D$ be any function defined on a finite domain W. Then there exists a subset W' of W with $|W'| \geq |W|^{1/2}$ such that $f|_{W'}$ is either constant or 1-1.*

LEMMA 4'. *Let $f, g: W \to D$ be any constant or 1-1 functions defined on a finite domain W. Then there exists a subset W' of W with $|W'| \geq |W|/4$ such that $f|_{W'}$ and $g|_{W'}$ are either identical or have disjoint ranges.*

Now observe that, in the proofs of Theorems 1 and 3, Lemma 3 is applied to d_i^p and w_i^p, for $i = 1, \ldots, n$ and $p = 1, \ldots, T$, and, for each time step, it is applied to $r_{i,h}$, for $i = 1, \ldots, n$ and $h = 1, \ldots, m$. But from Lemma 1, all the functions $r_{i,h}$, for $h = 1, \ldots, m$ behave in the same way. Therefore, Lemma 3′ only has to be applied a total of $O(nT)$ times. Since $T = O(\log\log(n))$, we can conclude that both theorems hold even when the inputs are restricted to be integers with at most $2^{O(n\log\log(n))}$ bits. For much smaller input domains, our method fails and the complexity of the two functions is an open question.

ACKNOWLEDGMENTS

Faith E. Fich's research was supported by an IBM Faculty Development Award, National Science Foundation Grant MCS-8402676, and the University of Washington Graduate School Research Fund.

This work was done while the authors Friedhelm Meyer auf der Heide and Avi Wigderson were at IBM Research Laboratory, San Jose.

REFERENCES

1. Atallah M, Vishkin U: Finding Euler tours in parallel. *J. Comp. and Syst. Sci.* 29(3):330–337, 1984.
2. Berge C: *Graphs and Hypergraphs.* North-Holland Publ., Amsterdam and New York, 1973.
3. Fich FE, Ragde PL, Wigderson A: Relations between concurrent write models of parallel computation. *Proc. 3rd ACM Symposium on Principles of Distributed Computing, 1984* pp. 179–189.
4. Furst M, Saxe JB, Sipser M: Parity, circuits, and the polynomial time hierarchy. *Proc. 22nd Symposium on Foundations of Computer Science, 1981* pp. 348–359.
5. Galil Z: Optimal parallel algorithms for string matching. *Proc. 16th ACM Symposium on Theory of Computing, 1984* pp. 240–248.
6. Goldschlager L: A unified approach to models of synchronous parallel machines. *J. ACM* 29(4):1073–1086, 1982.
7. Graham RL, Rothschild BL, Spencer JH: *Ramsey Theory.* Wiley, New York, 1980.
8. Hansel G: Nombre minimal de contacts de fermature nécessaires pour réaliser une fonction booléenne symétrique de n variables. *C. R. Acad. Sci. Paris* 258:6037–6040, 1964.
9. Kannan R, Miller G, Rudolph L: A sublinear parallel algorithm for computing the greatest common divisor of two integers. *Proc. 25th Symposium on Foundations of Computer Science, 1984* pp. 7–11.
10. Kucera L: Parallel computation and conflicts in memory access. *Information Processing Letters* 14(2):93–96, 1982.
11. Meyer auf der Heide F, Reischuk R: On the limits to speed up parallel machines by large hardware and unbounded communication. *Proc. 25th Symposium on Foundations of Computer Science, 1984* pp. 56–64.
12. Pippenger N: An information theoretic method in combinatorial theory. *J. Combinatorial Theory* [A] 23(1):99–104, 1977.

13. Shiloach Y, Vishkin U: Finding the maximum, merging, and sorting on parallel models of computation. *Journal of Algorithms* 2:88–102, 1981.
14. Tarjan RE, Vishkin U: Finding biconnected components and computing tree functions in logarithmic parallel time. *Proc. 25th Symposium on Foundations of Computer Science, 1984* pp. 12–20.
15. Valiant L: Parallelism in comparison problems. *SIAM J. Comp.* 4(3):348–355, 1975.
16. Vishkin U, Wigderson A: Trade-offs between depth and width in parallel computation. To appear in *SIAM J. Comp.* 14(2):303–314, 1985.