SIAM J. DISC. MATH.
Vol. 1, No. 3, August 1988

THE PARALLEL COMPLEXITY OF ELEMENT DISTINCTNESS IS $\Omega(\sqrt{\log n})$*

PRABHAKAR RAGDE†, WILLIAM STEIGER‡, ENDRE SZEMERÉDI§, AND AVI WIGDERSON¶

Abstract. We consider the problem of element distinctness. Here n synchronized processors, each given an integer input, must decide whether these integers are pairwise distinct, while communicating via an infinitely large shared memory.

If simultaneous write access to a memory cell is forbidden, then a lower bound of $\Omega(\log n)$ on the number of steps easily follows (from S. Cook, C. Dwork, and R. Reischuk, *SIAM J. Comput.*, 15 (1986), pp. 87-97.) When several (different) values can be written simultaneously to any cell, then there is an simple algorithm requiring $O(1)$ steps.

We consider the intermediate model, in which simultaneous writes to a single cell are allowed only if all values written are equal. We prove a lower bound of $\Omega((\log n)^{1/2})$ steps, improving the previous lower bound of $\Omega(\log\log\log n)$ steps (F.E. Fich, F. Meyer auf der Heide, and A. Wigderson, *Adv. in Comput.*, 4 (1987), pp. 1-15).

The proof uses Ramsey-theoretic and combinatorial arguments. The result implies a separation between the powers of some variants of the PRAM model of parallel computation.

Key words. parallel computation, lower bounds, parallel random access machines

AMS(MOS) subject classification. 68Q10

1. Introduction.

The parallel random access machine (PRAM) is the most widely used theoretical model of parallel computation. In this machine, n synchronized processors $P_1, P_2, \ldots, P_n$ have read and write access to a shared memory $\{M_i | i \in \mathbb{N}\}$. Each cell M_i of shared memory is initialized to zero. Each processor is a (possibly infinite) state machine. One step of computation consists of two phases. In the write phase, each processor may write an integer value to some shared memory cell. All writes take place simultaneously. In the read phase, each processor may read some shared memory cell. Based on what it has seen, each processor then assumes a new state.

A PRAM may be restricted to disallow simultaneous read or write access by several processors to the same cell. We allow concurrent reads and writes. If several processors attempt to simultaneously write different values into the same cell, a write conflict arises. Here, we discuss two methods of write-conflict resolution that have appeared in the literature. In the COMMON model [K], the algorithm itself must ensure that a write conflict never occurs; all processors simultaneously writing into the same cell must be writing the same value. In the ARBITRARY model, the machine will resolve the write conflict arbitrarily; one of the values being written will appear in the cell, but it is impossible in advance to know which value. Algorithms on the ARBITRARY model must work regardless of who wins each competition to write.

* Received by the editors January 8, 1987; accepted for publication (in revised form) February 12, 1988.

†Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4. This work was performed while the author was at the University of California, Berkeley, California 94720, and was supported by an IBM Graduate Fellowship and a Natural Sciences and Engineering Research Council of Canada Postgraduate Scholarship.

‡Rutgers University, New Brunswick, New Jersey 08903.

§Hungarian Academy of Sciences, Budapest, Hungary.

¶The Hebrew University, Jerusalem, Israel. This work was done while the author was at the Mathematical Sciences Research Institute, Berkeley, California 94720, and was supported by National Science Foundation grant MCS-8120790

The input to the PRAM is an n-tuple $(x_1, x_2, \ldots, x_n)$, where the input variable x_i affects the initial state of P_i. Thus the input variables are initially distributed one to a processor. (This state of affairs is equivalent to storing the values in the first n cells of memory at the start of the computation.) We will choose input variables from $\mathbb{N}$ (the positive integers). We denote the set $\{1, 2, \ldots, k\}$ by $[k]$. All logarithms are to the base 2.

The element distinctness problem demonstrates the weakness of the COMMON model, relative to the ARBITRARY model, when the size of shared memory is infinite. In this problem, processor P_1 must halt in state "not-distinct" if there exist indices i, j (where $i \neq j$) such that $x_i = x_j$; otherwise, it must halt in state "distinct". In §2, we show how to solve this problem in $O(1)$ steps on the ARBITRARY model, and discuss two algorithms to solve it on the COMMON model in $O(\log n)$ steps. In §3, we state and prove a generalization of a result by Hensel concerning the covering of a clique by bipartite subgraphs, which is needed for the next section. Finally, in §4 we present a lower bound of $\Omega(\sqrt{\log n})$ steps for solving element distinctness on the COMMON model. The lower bound proceeds by using Ramsey theory to restrict the set of inputs under consideration and thus simplify the actions of processors to the point where they transfer information only by a combination of the two methods discussed in §2. Then an adversary argument is given, in which the result from §3 and other combinatorial theorems are utilized.

2. Upper bounds. We first show how to solve element distinctness in constant time on the ARBITRARY model, and then give two different $O(\log n)$ algorithms for the COMMON model. All of these algorithms appear in [FMW].

THEOREM 1. *Element distinctness can be solved in $O(1)$ steps on the* ARBITRARY *model.*

Proof. Consider the following algorithm, which uses all of the infinite memory as a hash table. In the first step, each processor P_i writes the value i into cell M_{x_i}. Each processor then reads the cell into which it just wrote. If all variables are distinct, then each processor will see the value it wrote. But if two variables are equal, then at least two processors will attempt to write into the same cell, and at least one will see a value different from the one it wrote. If the latter occurs, P_1 can be informed in one more step, by having any such processor write the value "not-distinct" into M_1. $\square$

Notice that this algorithm will work on any model that allows different values to be simultaneously written to the same cell, and that the infinite memory is essential to the algorithm.

Obviously, this algorithm will not work on the COMMON model. On this model, however, shared memory can be used as a hash table in a more indirect way. Given two nonintersecting sets of indices S_1 and S_2 (both subsets of $[n]$), each P_i $(i \in S_1)$ writes 1 into cell M_{x_i}. Then each P_j $(j \in S_2)$ reads cell M_{x_j}. The sets of values $\{x_i | i \in S_1\}$ and $\{x_j | j \in S_2\}$ are distinct if and only if each processor P_j $(j \in S_2)$ reads the value zero.

This method can be used to solve element distinctness in $\lceil \log n \rceil + 1$ steps on the COMMON model. At the tth step $(t \leq \lceil \log n \rceil)$, the set S_1 consists of those indices i for which the tth binary digit of i is 1, and S_2 consists of those indices j for which the tth binary digit of j is zero. There is a slight technical problem because memory is not re-initialized to zero, but this can be easily overcome by having each processor write the value t (instead of 1) at the tth step. In the last step, any processor who has detected nondistinctness informs P_1 by writing the value 'not-distinct' into M_1. Since any distinct indices i, j differ in at least one binary digit, the distinctness of every pair

(x_i, x_j) will be verified.

Throughout the above algorithm, processors are only aware of the value of the variable that they are given at the beginning of the algorithm. The algorithm does not use the local computational power of processors, nor does it require memory cells to hold large values. But again, the fact that memory is infinite is essential, as it is not the values written that are important, but the location into which they are written.

Another method of answering the question is to allow processors to learn the values of other variables. If a processor knows two or more variables (as reflected in the state of that processor), then it knows if those variables are distinct.

This idea can be used to give a different algorithm for solving element distinctness on the COMMON model, in $\lceil \log n \rceil$ steps. One cell of shared memory is dedicated to each processor, to serve as its mailbox. Initially, each processor knows just one variable. In the tth step, each processor P_i computes an encoding of all the variables it knows and places this encoding in the mailbox of processor P_j, where j differs from i only in the tth binary digit. Each processor then reads its own mailbox and learns the values stored there. A straightforward analysis shows that at each step, the number of distinct variables that any processor knows will double. Thus after $\lceil \log n \rceil$ steps, each processor will know the value of all n variables, and processor P_1 can halt in the desired state.

This second method illustrates the power of the PRAM model; it shows that *any* function of n variables can be computed in $O(\log n)$ steps. It uses only n cells, but the capacity of those cells must be large, and the local computational power of each machine is used in computing encodings. Indirect access to memory is not used, nor are concurrent reads and writes; it is the values written that are important, as the location into which each processor writes at each step is independent of the input.

In §4 we shall see, in the course of proving an $\Omega(\sqrt{\log n})$ lower bound for element distinctness on the COMMON model, that these two methods are essentially the only two ways in which processors can verify distinctness. The temptation to conjecture that the complexity of element distinctness on the COMMON model is $\Theta(\log n)$ is misleading; in [FRW] a general simulation result is given, which has as a corollary an algorithm for element distinctness on this model requiring $O(\log n/ \log\log n)$ steps.

3. A combinatorial theorem.

In this section we prove a combinatorial result that, in addition to aiding in the proof of the subsequent lower bound, is of independent interest.

A *semi-bipartition* of a set X is a pair of sets $\{A, B\}$, where $A, B \subseteq X$, and $A \cap B = \phi$. For $y, z \in X$, we say the pair $\{y, z\}$ is covered by $\{A, B\}$, if either $y \in A$ and $z \in B$, or $y \in B$ and $z \in A$. (We always assume that the elements of a pair are distinct.) The size of a semi-bipartition $\{A, B\}$ is $|A| + |B|$. For a collection C of semi-bipartitions, we define $\text{size}(C) = \sum_{\{A,B\} \in C}(|A| + |B|)$.

Hansel [H] provides a very short and elegant proof of the following theorem. Pippenger [P] gives an interesting proof using information-theoretic arguments.

THEOREM 2. *Let X be a set of size n, and let C be a collection of semi-bipartitions over X such that every pair of elements of X is covered by some semi-bipartition in C. Then size$(C) \geq n \log n$.*

We generalize this result in two ways: first, we relax the requirement that every pair of elements be covered, and second, we generalize the notion of covering by semi-bipartitions.

The following theorem is implied by a theorem of Fredman and Komlós [FK]; it admits short proofs in the styles of [H] and [P]. We give a proof that is a generalization of Hansel's.

THEOREM 3. *Let X be a set of size n, and let E be a set of pairs of elements from X. Let α be the largest number of elements from X that can be chosen so that no two of them are in a pair of E. If $\mathcal{C}$ is a collection of semi-bipartitions over X such that every pair in E is covered by some semi-bipartition in $\mathcal{C}$, then size$(\mathcal{C}) \geq n\log(n/\alpha)$.*

Proof. Let $\mathcal{C} = \{\{A_i, B_i\}, i = 1, 2, \ldots, g\}$. For $x \in X$, let c_x be the number of semi-bipartitions $\{A_i, B_i\}$ such that $x \in A_i \cup B_i$. Then size$(\mathcal{C}) = \sum_{x \in X} c_x$.

Let U be the set of all tuples u of length g, where $u_i \in \{A_i, B_i\}$. We say that a tuple $u \in U$ is *consistent* with x if, for $1 \leq i \leq g$, either $x \in u_i$ or $x \notin A_i \cup B_i$. For each $x \in X$, there are $2^{g - c_x}$ tuples consistent with x. A tuple may be consistent with more than one element of X. If a tuple u were consistent with $\alpha + 1$ elements of X, then two of those elements (say x and y) would appear in a pair of E. But since $\mathcal{C}$ covers all pairs in E, there must exist some semi-bipartition $\{A_i, B_i\}$ such that, say, $x \in A_i$, $y \in B_i$. This means that $u_i = A_i$ by the consistency of u with x, and $u_i = B_i$ by the consistency of u with y, a contradiction. Hence any tuple is consistent with at most α elements of X.

Thus $\alpha 2^g = \alpha|U| \geq \sum_{x \in X}$ (# tuples consistent with x) $= \sum_{x \in X} 2^{g - c_x}$, and so, $\alpha \geq \sum_{x \in X} 2^{-c_x}$.

$(1/n)\sum_{x \in X} 2^{-c_x} \geq (\prod_{x \in X} 2^{-c_x})^{1/n}$, since the geometric mean does not exceed the arithmetic mean. Thus $(\prod_{x \in X} 2^{-c_x})^{1/n} \leq \alpha/n$. Taking logarithms, we have $\sum_{x \in X} c_x \geq n\log(n/\alpha)$, as required. $\square$

Next, we introduce a more general notion of covering. Let A be a set of tuples of length ℓ with components chosen from the set X. Let B be another such set. If $A \cap B = \phi$, we call $\{A, B\}$ a *tuple system* of length ℓ. For $y, z \in X$, we say that the tuple system $\{A, B\}$ covers the pair (y, z) if there exists a tuple $\alpha \in A$, a tuple $\beta \in B$, and a position i such that $\alpha_i = y$, $\beta_i = z$, and $\alpha_j = \beta_j$ for all $j \neq i$. That is, α, β differ only in position i, and y, z are the elements from the sets A, B in that position. If such a pair α, β of tuples exists, we say they are *similar* and cover (y, z).

If $\ell = 1$, this reduces to covering by semi-bipartitions. A collection of tuple systems $\mathcal{C}$ is a set $\{\{A_1, B_1\}, \{A_2, B_2\}, \ldots \{A_k, B_k\}\}$, where A_i, B_i are sets of tuples of length ℓ_i, and $A_i \cap B_i = \phi$. We define size$(\mathcal{C}) = \sum_{i=1}^{k}(|A_i| + |B_i|)$.

The following is the main result of this section.

THEOREM 4. *Given a set X of size n and a set E of pairs of elements from X, let α be the largest number of elements of X, no two of which are in a pair of E. Suppose $\mathcal{C}$ is a collection of tuple systems of length at most ℓ, and every pair of E is covered by a tuple system in $\mathcal{C}$. Then, for n sufficiently large, size$(\mathcal{C}) \geq n(\frac{1}{4}\log n - \frac{1}{2}\log(\alpha\ell) - \log\log n)$.*

As ℓ becomes larger, this bound becomes weaker. We conjecture that the lower bound on size is in fact $n\log(n/\alpha)$ regardless of the length of tuples. For our application, it will be the case that $\alpha\ell = O(n^{1/2-\epsilon})$ for some positive constant ϵ, and thus we obtain a lower bound of $\Omega(n\log n)$ on size$(\mathcal{C})$.

Proof of Theorem 4. Suppose we are given a tuple system $\mathcal{C}$ that covers E in the fashion described, and in addition, size$(\mathcal{C}) < n(\frac{1}{4}\log n - \frac{1}{2}\log(\alpha\ell) - \log\log n)$. For each tuple system $\{A_i, B_i\} \in \mathcal{C}$, we are going to construct semi-bipartitions that cover most of the edges that the tuple system covered, and then apply Theorem 3 to obtain a contradiction.

Let $\mathcal{C}'$ be an initially empty collection of semi-bipartitions, and E' an initially empty collection of pairs of distinct elements of X. E' will contain the pairs from E that $\mathcal{C}'$ does not cover.

Let a_i, b_i be the initial sizes of A_i, B_i respectively. We say the tuple system $\{A_i, B_i\}$ is *sparse* if there are at most $(|A_i||B_i|)/\sqrt{n}$ similar pairs in $\{A_i, B_i\}$. If

$\{A_i, B_i\}$ is not sparse, then there exists some $\alpha \in A_i$ which is similar to at least $|B_i|/\sqrt{n}$ tuples in B_i. There is some position p such that at least $|B_i|/\ell\sqrt{n}$ tuples in B_i differ from α only in position p, since α is a tuple of length at most ℓ. Let A'_i be the set that includes α and all tuples in A_i that differ from α only in position p, and B'_i be all tuples in B_i that differ from α only in position p. Then $|B'_i| \geq |B_i|/\ell\sqrt{n}$. Also, $A'_i \cap B'_i = \phi$.

Let A be the set of elements occurring at position p in tuples of A'_i, and let B be the set of elements occurring at position p in tuples of B'_i. We know that $|A| = |A'_i|$, and $|B| = |B'_i|$. Furthermore, since $A_i \cap B_i = \phi$ and all tuples in A'_i, B'_i differ only in position p, $A \cap B = \phi$. Thus $\{A, B\}$ is a semi-bipartition of X. We add $\{A, B\}$ to $\mathcal{C}'$. $\{A, B\}$ covers all pairs covered by $\{A'_i, B'_i\}$.

We remove A'_i from A_i and B'_i from B_i. We have not accounted for pairs covered by similarities between A'_i and $B_i - B'_i$, or between B'_i and $A_i - A'_i$. But any tuple in $B_i - B'_i$ is similar to at most one tuple in A'_i, for it can differ from a tuple in A'_i only in a position $p' \neq p$, and all tuples in A'_i have the same entry in position p' but different entries in position p. Similarly, any tuple in $A_i - A'_i$ is similar to at most one tuple in B'_i. Thus we have neglected at most $|A_i| + |B_i| \leq a_i + b_i$ similar pairs of tuples. We add the pairs of elements that these similar pairs of tuples cover to E'.

If $\{A_i, B_i\}$ is not yet sparse, we repeat this process. Each time, we shrink B_i by at least a factor of $\gamma = 1-(1/\ell\sqrt{n})$. In at most $\log_{(1/\gamma)} b_i$ iterations, $\{A_i, B_i\}$ becomes sparse (note that an empty tuple system is sparse). The total number of pairs we have added to E' is at most $\log_{(1/\gamma)} b_i(a_i + b_i) \leq \ell\sqrt{n}\log b_i(a_i + b_i)$.

When $\{A_i, B_i\}$ becomes sparse, we simply add to E' all pairs of elements that are covered by similarities in $\{A_i, B_i\}$. This adds at most $a_ib_i/\sqrt{n}$ pairs to E'.

When this process is completed, clearly size$(\mathcal{C}') \leq$ size$(\mathcal{C})$. Furthermore, $|E'| \leq \sum_i((a_ib_i/\sqrt{n}) + (a_i + b_i)\ell\sqrt{n}\log b_i)$. Since a_i,b_i, and $\sum_i(a_i + b_i)$ are all less than $\frac{1}{4}n\log n$ by the assumption on size$(\mathcal{C})$, it follows that for n sufficiently large, $|E'| \leq \frac{1}{3}\ell n^{3/2}\log^2 n$.

As a result, we have a collection $\mathcal{C}'$ of semi-bipartitions and a set E' of at most $\frac{1}{3}\ell n^{3/2}\log^2 n$ pairs such that every pair in $E - E'$ is covered by $\mathcal{C}'$. Let D be the largest set of elements of X such that no pair in $E - E'$ contains two elements of D. By Theorem 3, we know that size$(\mathcal{C}') \geq n\log(n/|D|)$. All pairs in E that contain two elements of D must also be pairs in E'. Hence the number of pairs in E that have two elements in D is at most $|E'|$. Let D' be the largest subset of D such that no pair in E contains two elements of D. We know that $|D'| \leq \alpha$ by the definition of α.

Turán's theorem [B, p. 282] states that given any set S and any collection P of pairs from S, there is a set S' such that no pair in P contains two elements from S', and $|S'| \geq |S|^2/(|S| + 2|P|)$. Applying this to D and the pairs of E with both elements in D, we know that $|D'| \geq |D|^2/(|D| + 2|E'|)$.

If $|D| > |E'|$, then $\alpha \geq |D'| \geq |D|^2/3|D|$, and so $|D| \leq 3\alpha$. If $|D| \leq |E'|$, then $\alpha \geq |D|^2/3|E'|$, and $|D| \leq n^{3/4}\log n\sqrt{\alpha\ell}$. In either case we have our contradiction, since we can conclude that size$(\mathcal{C}') \geq n(\frac{1}{4}\log n - \frac{1}{2}\log(\alpha\ell) - \log\log n)$, but size$(\mathcal{C}') \geq$ size$(\mathcal{C})$. □

4. The lower bound. In this section, we prove a lower bound for the element distinctness problem on the COMMON model, thus demonstrating a separation between the COMMON and ARBITRARY models when both are allowed infinite memory. The proof combines techniques from [FMW] and [MW] with the result from §3.

The following rather odd definition is motivated by technical problems that arise in the main result if overwriting a shared memory cell is allowed. A similar technique was used in [FMW].

DEFINITION. A *write-once* COMMON PRAM is a COMMON PRAM with the following modifications: if a cell in shared memory is written into at a particular step, then it may not be written into at any subsequent step. Also, at step t, each processor is allowed to simultaneously read t cells $M_{i_1}, M_{i_2}, \ldots, M_{i_t}$ of shared memory, instead of just one. The final restriction is that, if cell M_{i_j} defined above was written into at all, it must have been written into at step j.

LEMMA 5. *T steps of a* COMMON *PRAM with m cells of shared memory can be simulated by T steps of a write-once* COMMON *PRAM with m^2 cells of shared memory.*

Proof. Let $\langle \cdot, \cdot \rangle : [m] \times [m] \to [m^2]$ be any one-to-one function. At step t of the simulation, if P_i in the simulated machine writes into M_j and reads from M_k, then P_i in the write-once machine writes into $M_{\langle t,j \rangle}$, and reads $M_{\langle 1,k \rangle}, M_{\langle 2,k \rangle}, \ldots, M_{\langle t,k \rangle}$. From the contents of these t cells, P_i can determine what M_k in the simulated machine would have contained at step t. □

By Lemma 5, any lower bound for element distinctness on write-once COMMON with infinite memory is a lower bound on regular COMMON with infinite memory. We can now state and prove the main result of this paper.

THEOREM 6. *On a write-once* COMMON PRAM *with infinite memory, element distinctness requires $\Omega(\sqrt{\log n})$ steps.*

We introduce some definitions to be used in the proof. A (k, s, a)-*bundle* on X will be a partial order on the set X, where $ks + a = |X|$. Of the $ks + a$ elements, ks are in k disjoint antichains of size $s > 1$; the remaining a of the elements are in disjoint antichains of size 1, and there is a total ordering among all antichains. An antichain of size s will be called *nontrivial*. Figure 1 gives a simple example of such a partial order.

$$x_{10} < \begin{matrix} x_7 \\ x_3 \end{matrix} < \begin{matrix} x_1 \\ x_9 \end{matrix} < x_2 < x_4 < \begin{matrix} x_6 \\ x_5 \end{matrix} < x_8$$

FIG. 1. *A (3,2,4) bundle on* $\{x_1, x_2, \ldots, x_{10}\}$.

Let X denote the set of variables $\{x_1, x_2, \ldots, x_n\}$. Suppose we are given an algorithm that solves element distinctness. We will prove a lower bound of $T+1$ steps on the running time of this algorithm, where $T = \Omega(\sqrt{\log n})$.

The state of a processor P_i after step t (and consequently, the cells it chooses to write into and read from at step $t + 1$) can be a complicated function of the variables in X. The idea is to concentrate on a restricted set of inputs I_t which are "indistinguishable" to the algorithm through step t. For inputs in I_t, the state of a processor during the first t steps will be a function of a "small" number of variables. We can then apply Ramsey theory to simplify the functions describing access to shared memory. The input set I_t will be described in terms of objects Π_t, C_t, and S_t. We will display, for t ranging from zero to T, the following:

1. An infinite set S_t, where $S_t \subseteq \mathbb{N}$, and $S_t \subseteq S_{t-1}$. S_t is the set of possible values of a variable for inputs in I_t. We will define S_t so as to simplify all functions describing access to shared memory.

2. A (k_t, s_t, a_t)-bundle Π_t over X. Π_t will be a refinement of Π_{t-1}; that is, if x_i, x_j are comparable in Π_{t-1}, then they are comparable in Π_t. The bundle will satisfy the inequalities:

- $k_t \leq 3^{2t^2} + 5t + 3$;

• $a_t \leq \frac{n}{3} - \frac{n}{3^{t+2}} + 3^{2t^2+5t+4}$.

Pi_t will 'describe' inputs in I_t; the values of the variables in any input of I_t will satisfy the constraints of Π_t. We shall choose T so that it will always be the case that $a_t \leq 2n/3$ for $t \leq T$, that is, at least a third of the variables are in nontrivial antichains of Π_t.

3. A collection $\mathcal{C}_t$ of tuple systems over X, where $\mathcal{C}_t \supseteq \mathcal{C}_{t-1}$, size$(\mathcal{C}_t) \leq n(t+1)^2$, and the size of tuples in $\mathcal{C}_t$ is at most 3^t.

4. For each processor P_i, a set $V_t^i \subseteq X$ with the properties:

• $V_{t-1}^i \subseteq V_t^i$ and $|V_t^i| \leq 3^t$;

• All the variables in V_t^i are comparable in Π_t, that is, V_t^i is a chain of Π_t;

• The state of P_i after step t will be a function of *only* the variables in V_t^i, for all inputs in I_t.

We may now describe the set of inputs I_t. $\hat{x} \in I_t$ if and only if:

1. $\hat{x} \in S_t^n$;
2. $\hat{x}$ is consistent with Π_t;
3. There is at most one pair (j,k) such that $\hat{x}_j = \hat{x}_k$;
4. If $\hat{x}_j = \hat{x}_k$, (x_j, x_k) is not covered by any tuple system in $\mathcal{C}_t$. Note that the containment properties of $\mathcal{C}_t, \Pi_t$, and S_t ensure that $I_t \subseteq I_{t-1}$.

Suppose there is at least one pair (x_j, x_k) such that (x_j, x_k) is not covered by any tuple system in $\mathcal{C}_t$, and x_j, x_k are in the same antichain of Π_t. We know that at most one of x_j, x_k can affect the state of P_1 after step t, since V_t^1 is a chain in Π_t. Suppose x_k does not affect the state of P_1 after step t. Let $\hat{x}$ be an input in I_t such that all $\hat{x}_i$ are distinct. Let $\hat{x}'$ be the input such that $\hat{x}'_i = \hat{x}_i$ for $i \neq k$, but $\hat{x}'_k = \hat{x}'_j$. By choice of (x_j, x_k), $\hat{x}'$ is also an input in I_t, and by construction, not all the components of $\hat{x}'$ are distinct. Furthermore, since the state of P_1 is not a function of x_k, P_1 cannot distinguish these inputs after step t. Thus we obtain a lower bound of $t+1$ steps. We now show how to construct the objects described above, up to index T, where $T = \Omega(\sqrt{\log n})$.

Before step 1 we let Π_0 be the empty order, that is, $k_0 = 1$, $s_0 = n$, $a_0 = 0$. We also set $\mathcal{C}_0 = \phi$, $S_0 = \mathbb{N}$, $I_0 = (\mathbb{N})^n$, $V_0^i = \{x_i\}$. Now suppose we have constructed all the required objects up to index t. We show how to extend the construction to index $t+1$, by consideration of what happens at step $t+1$ for inputs in I_t.

We say a function $f(y_1, y_2, \ldots, y_\ell)$ is *defined on increasing tuples over S* if the domain of f consists of the increasing tuples over S. A tuple $\hat{y}$ is increasing if $\hat{y}_i < \hat{y}_j$ for $i < j$.

For a step $t' \leq t+1$, let $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ be the variables in $V_{t'-1}^i$, where the indices are chosen so that $x_{i_1} < x_{i_2} < \cdots < x_{i_k}$ in Π_t. We know this is possible because $V_{t'-1}^i$ is totally ordered in Π_t. The state of P_i just before step t', for inputs in I_t, is a function of $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$. Thus the indices of the cells that P_i writes into and reads from at step t' are also a function of these variables.

Let $w_i^{t'}(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$ be the *write index function* of P_i at step $t' \leq t+1$, for inputs in I_t. The value of $w_i^{t'}$ is the index of the cell into which P_i writes, or zero if the processor does not write at step t', for any setting of the input variables. We know that $w_i^{t'}$ is defined on increasing tuples over S_t, since for every input $\hat{x}$ in I_t, $\hat{x}_{i_1} < \hat{x}_{i_2} < \cdots < \hat{x}_{i_k}$.

Similarly, let $r_{i,j}^{t+1}$ be the jth *read index function* of P_i at step $t+1$ for inputs in I_t. The value of $r_{i,j}^{t+1}$ is the index of the jth cell that P_i reads from at step $t+1$, for inputs in I_t. $r_{i,j}^{t+1}$ is also defined on increasing tuples over S_t. We use the following lemma by Meyer auf der Heide and Wigderson [MW], which is based on a theorem by Erdős and Rado [ER]:

LEMMA 7. *Let $\mathcal{F}$ be a finite collection of functions defined on increasing tuples over S, for S infinite. The functions in $\mathcal{F}$ may be functions of differing numbers of variables. Then there exists an infinite $\tilde{S} \subseteq S$ such that, restricted to increasing tuples over $\tilde{S}$, each $f \in \mathcal{F}$ is one-to-one on the variables it depends on. Furthermore, two distinct $f, f' \in \mathcal{F}$ have the property that, restricted to increasing tuples over $\tilde{S}$, they either have disjoint ranges, or are identical.*

We apply Lemma 7 to the collection $\mathcal{F}$, where

$$\mathcal{F} = \{w_i^{t'} | 1 \le t' \le t+1, 1 \le i \le n\} \cup \{r_{i,j}^{t+1} | 1 \le i \le n, 1 \le j \le t+1\}.$$

Let $\tilde{S}$ be the infinite subset of S_t with the properties mentioned in Lemma 7. For $f \in \mathcal{F}$, when f takes on the value zero for some input, it means that processors using f at some step choose not to write (or read) at that step for that input. We would like to simplify the situation by ensuring that, at a particular timestep, processors either write for all inputs or do not write for any input. We can do this by reducing $\tilde{S}$. For each $f \in \mathcal{F}$, if f is not constant when restricted to increasing tuples over $\tilde{S}$, let $\{v_1, v_2, \dots, v_k\}$ be the variables on which f depends. Since f is 1-1 on those variables, if zero is in the range of f, then there are unique values $\hat{v}_1, \hat{v}_2, \dots, \hat{v}_k$ such that setting $v_i = \hat{v}_i$ for $i = 1, 2, \dots, k$ makes $f = 0$. In this case, we remove the values $\hat{v}_1, \hat{v}_2, \dots, \hat{v}_k$ from $\tilde{S}$. Then the range of f when restricted to $\tilde{S}$ does not include zero. If this is done for all $f \in \mathcal{F}$, then for inputs in I_{t+1}, processors will either never write (read) or always write (read) at any given time step among the first $t+1$ steps.

Set $S_{t+1} = \tilde{S}$. If we also set $\Pi_{t+1} = \Pi_t$, and $\mathcal{C}_{t+1} = \mathcal{C}_t$, this defines a set of inputs I_{t+1} such that for inputs in this set, the read index and write index functions of all processors through step $t+1$ will have the properties mentioned in the lemma. The fact that $f, f' \in \mathcal{F}$ either have disjoint ranges or are identical means that, when a processor uses f as a write index function, it is communicating only with those processors that use f as a read index function at subsequent steps. Thus we have considerably simplified the pattern of communication in the first $t+1$ steps, by restricting consideration to inputs in I_{t+1}. In what follows, our goal is to define V_{t+1}^i for all processors P_i. To do this, we will add tuple systems to $\mathcal{C}_{t+1}$; this only reduces I_{t+1}.

We also set $Q_i = \{\{V_t^i\}\}$ for $1 \le i \le n$. The set Q_i will represent the variables that P_i knows about at the end of step $t+1$. We will add to Q_i sets of variables that P_i could learn at step $t+1$, by considering the access functions of P_i in $\mathcal{F}$. At the conclusion of this process, Q_i will be a collection of at most $t+2$ sets of variables. Each set in Q_i will be totally ordered in Π_t, and the state of P_i at time $t+1$ will, by construction, depend only on variables in the union of those sets. We will then refine Π_{t+1} in such a way that this union is totally ordered in Π_{t+1}, thus ensuring our conditions after step $t+1$ and completing the induction step.

We consider each function $f \in \mathcal{F}$ in turn, and deal with processors that use this f as an access function.

Case 1: f constant. If f is constant, then it represents the index of a single cell that processors using f read from or write to. Let t' be the unique step at which processors write into that cell. If no such step exists, let $\mathcal{W} = \phi$; otherwise, let $\mathcal{W} = \{P_i | w_i^{t'} = f\}$. Let $\mathcal{R}_j = \{P_i | r_{i,j}^{t+1} = f\}$. $\mathcal{R}_j$ is the set of processors that access cell f as the jth of the $t+1$ cells that they read at step $t+1$.

If $\mathcal{W} = \phi$, then all processors in $\mathcal{R}_j$, $1 \le j \le t+1$, read zero from cell f at step $t+1$, and so no information is transmitted. If $\mathcal{W} \ne \phi$, the value written into cell f at step t' must be a function of variables that the processors in $\mathcal{W}$ "have in common"; that is, it must depend only on variables in $V = \bigcap_{P_i \in \mathcal{W}} V_{t'-1}^i$. Recall that processors using f always write on all inputs in I_{t+1}; if the value depended on some variable not in V, then there would be an input for which two processors in $\mathcal{W}$ attempt to write different values into the cell indexed by f, a violation of the COMMON model.

Thus, the state of any processor $P_i \in \mathcal{R}_j$ at time $t+1$ may be affected by variables in V, for any input in I_{t+1}. The variables in V are totally ordered in Π_t, and $|V| \leq 3^{t'-1}$. For each reading processor $P_i \in \mathcal{R}_j$, $1 \leq j \leq n$, we add V to Q_i.

Case 2: f not constant. Let $y_1, y_2, \ldots, y_\ell$ be the variables that f depends on (considered as free variables, not members of X). We note that if f is used as a write index function by some processor at time t', then $\ell \leq 3^{t'-1}$. Let $\mathcal{R}_j = \{P_i | r_{i,j}^{t+1} = f\}$, and $\mathcal{W}_{t'} = \{P_i | w_i^{t'} = f\}$, for $t' \leq t+1$.

Consider $P_i \in \mathcal{W}_{t'}$ and $P_j \in \mathcal{R}_h$. To say that P_i uses f as a write index function at time t' means that $w_i^{t'}$ is equal to f with $y_1, y_2, \ldots, y_\ell$ replaced by some variables in $V_{t'-1}^i$. If the replacement variables for P_i are exactly the same as the replacement variables for P_j using f as its hth read function, then P_j at step $t+1$ will read as its hth cell the same cell that P_i wrote into at time t'. Let $\mathcal{W}_j'$ be the set of processors in $\mathcal{W}_{t'}$ that have the same set of replacement variables for f as P_j. As in case 1, we may set $V = \bigcap_{P_k \in \mathcal{W}_j'} V_{t'-1}^k$, and note that the value written is a function of V. As before, it is important that processors in $\mathcal{W}_j'$ always write on all inputs in I_{t+1}. We also have $|V| \leq 3^{t'-1}$. We add V to Q_j and, having dealt with what P_j could learn using function f, remove P_j from $\mathcal{R}_h$. If we do this for all such P_i, P_j, then no processor in any $\mathcal{W}_{t'}$ will have exactly the same replacement variables as any processor in any $\mathcal{R}_h$.

For the remaining processors, our goal is to ensure that no processor using f as a read index function at step $t+1$ accesses the same cell as any processor using f as a write index function at step $t+1$ or earlier. In order to ensure this, we add tuple systems to $\mathcal{C}_{t+1}$. Again, consider $P_i \in \mathcal{W}_{t'}$ and $P_j \in \mathcal{R}_h$. Let $x_{i_1}, x_{i_2}, \ldots, x_{i_\ell}$ be the replacement variables of P_i for $y_1, y_2, \ldots, y_\ell$, and $x_{j_1}, x_{j_2}, \ldots, x_{j_\ell}$ be the replacement variables of P_j. At most two variables are equal for any input in I_{t+1}. If there are positions $1 \leq a < b \leq \ell$ such that $i_a \neq j_a$ and $i_b \neq j_b$, then for every input in I_t, either $x_{i_a} \neq x_{j_a}$ or $x_{i_b} \neq x_{j_b}$. It follows that, for every input in I_t, the fact that f is one-to-one ensures that $f(x_{i_1}, \ldots, x_{i_\ell}) \neq f(x_{j_1}, \ldots, x_{j_\ell})$, and so P_j will read a different cell from the one into which P_i wrote.

On the other hand, suppose there is only one position a such that $i_a \neq j_a$. If, for some input $\hat{x}$, P_j reads the same cell as P_i, it "knows" that $\hat{x}_{i_a} = \hat{x}_{j_a}$. We would like the PRAM to be unable to answer after $t+1$ steps, for inputs in I_{t+1}. To ensure that P_i and P_j access different cells, we must ensure that $x_{i_a} \neq x_{j_a}$ for all inputs in I_{t+1}. If this is done, then $P_i \in \mathcal{R}_h$ will read the value zero for all inputs in I_{t+1}, and it will gain no additional information as a result of that read.

Let A_f be the set of all tuples $(x_{i_1}, \ldots, x_{i_\ell})$ that are the replacement variables for f of some processor P_i in some $\mathcal{W}_{t'}$ $(t' \leq t+1)$. Similarly, B_f is the set of all tuples $(x_{j_1}, \ldots, x_{j_\ell})$ that are the replacement variables for f of some processor P_j in some $\mathcal{R}_h$ $(h \leq t+1)$. $\{A_f, B_f\}$ forms a tuple system, since our actions above ensured that no processor in $\mathcal{W}_t'$ has the same set of replacement variables as any processor in $\mathcal{R}_h$. Furthermore, the pairs of variables that we require to be distinct are precisely those covered by this tuple system. Adding $\{A_f, B_f\}$ to $\mathcal{C}_{t+1}$ will ensure (by the definition of I_{t+1}) that no reader using f as a read index function at time $t+1$ will read a cell written into by any writer using a different write index function or a different set of replacement variables than the reader. This concludes the description of what is done in the case of one function $f \in \mathcal{F}$.

After this has been done for all f, $\mathcal{C}_{t+1}$ has been defined. Each processor has $t+1$ read index functions at time $t+1$, and hence can contribute at most $t+1$ tuples to some set B_f as described above. Each processor also has at most $t+1$ write index functions, at steps $1, 2 \ldots, t+1$, and so can contribute at most $t+1$ tuples to sets A_f above.

It follows that $\sum_{f \in \mathcal{F}} |B_f| \leq n(t+1)$ and $\sum_{f \in \mathcal{F}} |A_f| \leq n(t+1)$. Thus

$$\begin{aligned} \text{size}(\mathcal{C}_{t+1}) &\leq \text{size}(\mathcal{C}_t) + \sum_{f \in \mathcal{F}} (|A_f| + |B_f|) \\ &\leq (t+1)^2 n + 2(t+1)n \leq (t+2)^2 n \text{ as required.} \end{aligned}$$

At most one set V is added to Q_i for each read index function that P_i uses at time $t+1$, so $|Q_i| \leq t+2$. Furthermore, a set $V \in Q_i$ added as a result of a read index function that reads a cell written into at step t' is of size at most $3^{t'-1}$. It follows that Q_i contains at most one set of size $3^{t'-1}$ for $t' < t+1$, and at most two sets of size 3^t (one is V_t^i, and the other may be added as a result of reading a cell written at step $t+1$). Thus, if we set $V_{t+1}^i = \bigcup_{V \in Q_i} V$, then $|V_{t+1}^i| \leq 2 \cdot 3^t + \sum_{t' < t+1} 3^{t'-1} \leq 3^{t+1}$ as required.

We must still ensure that V_{t+1}^i is ordered in Π_{t+1}. Each $V \in Q_i$ is ordered in Π_t, so each antichain in Π_t contains at most $t+2$ elements that appear in sets $V \in Q_i$. Let R be the set of pairs (x_j, x_k) such that x_i, x_j are in the same antichain of Π_t but in different sets of some Q_i. By counting, we can show that $|R| \leq n3^{2t+1}$. If we refine Π_{t+1} so that every pair $(x_j, x_k) \in R$ is comparable in Π_{t+1}, then $V_i^{t+1} = \bigcup_{V \in Q_i} V$ will be totally ordered in Π_{t+1}, as required.

LEMMA 8. *There exists* Π_{t+1} *a refinement of* Π_t *such that all pairs in* R *are comparable in* Π_{t+1}, *and further,* Π_{t+1} *satisfies the required conditions on* k_{t+1}, a_{t+1}.

Proof. Each pair in R consists of two variables in the same nontrivial antichain of Π_t. At most $k_t/3^{t+2}$ antichains contain more than $3^{3t+3}n/k_t$ pairs in R, because $|R| \leq n3^{2t+1}$. For each such antichain, choose an arbitrary total order for its elements, in effect converting the nontrivial antichain to s_t trivial antichains. This increases a_{t+1} by at most $k_t s_t / 3^{t+2} \leq n/3^{t+2}$.

No remaining nontrivial antichain contains more than $3^{3t+3}n/k_t$ pairs in R. From each nontrivial antichain, take the $s_t/3^{t+2}$ elements that are in the most pairs in R, choose an arbitrary total order for them, and make them all larger than every element of the antichain. As a result, any element in any nontrivial antichain is in less than $3^{4t+6}n/k_t s_t$ pairs of R. If this were not true for some antichain, then counting occurrences of deleted elements in pairs would lead to a total of $3^{3t+4}n/k_t$ occurrences, a contradiction since each pair contains two elements. The deletions increase a_{t+1} by at most $k_t s_t / 3^{t+2} \leq n/3^{t+2}$.

The following theorem by Hajnal and Szemerédi [HS] is useful at this point.

THEOREM 9. *Let* $R \subseteq X^2$ *be a set of pairs such that no element of* X *occurs in* Δ *or more pairs in* R. *Then there is a number* s *such that* X *can be partitioned into* Δ *classes of size* s *or* $s+1$, *such that no pair in* R *contains two elements in the same class.*

Then, by Theorem 9, we can partition each nontrivial antichain into $3^{4t+6}n/k_t s_t$ classes such that no pair in R contains two elements from the same class, and each class is of size s or $s+1$. We remove one element from each classes of size $s+1$, choose an arbitrary total order for the removed elements, and make each larger than everything in their former antichain. Within each antichain, we choose an arbitrary total order for the classes, thus converting each antichain into $3^{4t+6}n/k_t s_t$ smaller nontrivial antichains, plus at most one new trivial antichain for each class. Thus

$$k_{t+1} \leq k_t \left\lceil \frac{3^{4t+6}n}{k_t s_t} \right\rceil \leq 3^{4t+7} k_t \text{ since } a_t \leq \frac{2n}{3}$$

$$\leq 3^{2(t+1)^2+5(t+1)+3} \text{ as required.}$$

We can thus define an appropriate Π_{t+1}. In the last step, we may have increased a_{t+1} by one element for each new class, and so in total,

$$\begin{aligned} a_{t+1} &\leq a_t + 2\left(\frac{n}{3^{t+2}}\right) + k_{t+1} \\ &\leq \frac{n}{3} - \frac{n}{3^{t+2}} + 3^{2(t+1)^2+5(t+1)+4} \text{ as required.} \end{aligned}$$

This concludes the description of the construction of objects of index $t+1$. We continue this construction to index $T = \epsilon\sqrt{\log n}$, for ϵ a suitably chosen small constant.

Let us choose ϵ sufficiently small ($\epsilon = 1/4$ will do), and n sufficiently large, so that the following conditions hold:

$$n > 3^{2T^2+6T+8}, \tag{10}$$

$$3^T \leq \frac{(n/3)^{1/6}}{\log(n/3)}, \tag{11}$$

$$k_T \leq \frac{(n/3)^{1/6}}{\log(n/3)}, \tag{12}$$

$$(T+1)^2 < \frac{1}{12}\log(\frac{n}{3}). \tag{13}$$

Condition (10) ensures that $a_t \leq 2n/3$ for $t \leq T$, which is required in order that the construction be possible up to index T. Conditions (11) and (12) ensure that Theorem 4 is applicable to the tuple system C_T and the nontrivial antichains of Π_T. To apply Theorem 4, we must know how many elements of X we can select so that no two are in the same nontrivial antichain (this is k_T, the number of nontrivial antichains) and we must know an upper bound on tuple length (this is 3^T). By the theorem, any tuple system that covers all pairs in all nontrivial antichains of Π_T must be of size at least $n(\frac{1}{4}\log(n/3) - \frac{1}{2}\log((n/3)^{1/3}/\log^2(n/3)) - \log\log(n/3))$, or at least $(1/12)n\log(n/3)$. But by condition (13), size$(C_T) < (1/12)n\log(n/3)$. Thus at least one pair of variables in some antichain of Π_T remains uncovered by C_T, thereby ensuring a lower bound of $T+1$ steps as discussed above. This concludes the proof of Theorem 6. $\square$

We conclude by mentioning some implications of this proof for separating the power of PRAM models with the same (finite) amount of shared memory. We can define a bounded version of the element distinctness problem, where the variables $\{x_i\}$ are chosen from $[m]$ for some $m \in \mathbb{N}$. An ARBITRARY PRAM with m shared memory cells can solve this version of element distinctness in $O(1)$ steps, by using the algorithm of Theorem 1. It is also possible to prove a bound of $\Omega(\sqrt{\log n})$ steps to solve bounded element distinctness on a COMMON PRAM with m memory cells in exactly the same fashion as given here, by using the finite version of the Erdős-Rado theorem from [ER2] to prove a finite version of Lemma 7. However, this only works for m larger than a very rapidly growing function of n, since a lower bound on the size of the set S mentioned in Lemma 7 is required, and this bound is expressed by a recurrence involving generalized Ramsey numbers. We can conclude, however, that for n sufficiently large, there exists $m = m(n)$ such that there is an $\Omega(\sqrt{\log n})$ separation between the COMMON and ARBITRARY PRAMs with m memory cells. This also held true for the $\Omega(\log\log\log n)$ separation given in [FMW], except that in that proof m was only required to grow as fast as $2^{2^{n^{O(1)}}}$

REFERENCES

[B] C. BERGE. *Graphs and Hypergraphs*. North-Holland, Amsterdam, 1973.

[CDR] S. COOK, C. DWORK, AND R. REISCHUK. *Upper and lower bounds for parallel random access machines without simultaneous writes*, SIAM J. Comput., 15 (1986), pp. 87-97.

[ER] P. ERDŐS AND R. RADO. *A combinatorial theorem*, J. London Math. Soc., 25 (1950), pp. 376-382.

[ER2] ———, *Combinatorial theorems on classifications of subsets of a given set*, Proc. London Math. Soc., 3 (1952), pp. 417-439.

[FK] M.L. FREDMAN AND J. KOMLÓS. *On the size of separating systems and families of perfect hash functions*, SIAM. J. Algebraic Discrete Methods, 5 (1984), pp. 61-68.

[FMW] F.E. FICH, F. MEYER AUF DER HEIDE, AND A. WIGDERSON. *Lower bounds for parallel random access machines with unbounded shared memory*, Adv. in Comput., 4 (1987), pp. 1-15.

[FRW] F.E. FICH, P. RAGDE, AND A. WIGDERSON. *Simulations Among Concurrent-Write PRAMs*, Algorithmica, 3 (1988), pp. 43-51.

[H] G. HANSEL. *Nombre minimal de contacts de fermeture nécessaires pour réaliser une fonction booléenne symmétrique de n variables*, Comptes Rendus Acad. Sci. Paris, 258 (1964), pp. 6037-6040.

[HS] A. HAJNAL AND E. SZEMERÉDI. *Proof of a conjecture of P. Erdős*, Combin. Theory Appl., 2 (1970), pp. 601-623.

[K] L. KUCERA. *Parallel computation and conflicts in memory access*, Inform. Process. Lett., 14 (1982), pp. 93-96.

[MW] F. MEYER AUF DER HEIDE AND A. WIGDERSON. *The complexity of parallel sorting*, Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, pp. 532-540.

[P] N. PIPPENGER. *An information-theoretic method in combinatorial theory*, J. Combin. Theory, ser. A, 23 (1977), pp. 99-104.