One, Two, Three ... Infinity: Lower Bounds for Parallel Computation

Faith E. Fich
University of Washington

Friedhelm Meyer auf der Heide
IBM Research, San Jose

Prabhakar Ragde
University of California at Berkeley

Avi Wigderson
IBM Research, San Jose

ABSTRACT

In this paper we compare the power of the two most commonly used concurrent-write models of parallel computation, the COMMON PRAM and the PRIORITY PRAM. These models differ in the way they resolve write conflicts. If several processors want to write into the same shared memory cell at the same time, in the COMMON model they have to write the same value. In the PRIORITY model, they may attempt to write different values; the processor with smallest index succeeds.

We consider PRAM's with n processors, each having arbitrary computational power. We provide the first separation results between these two models in two extreme cases: when the size m of the shared memory is small ($m \leq n^{\epsilon}, \epsilon < 1$), and when it is infinite.

In the case of small memory, the PRIORITY model can be faster than the COMMON model by a factor of $\Theta(\log n)$, and this lower bound holds even if the COMMON model is probabilistic. In the case of infinite memory, the gap between the models can be a factor of $\Omega(\log\log\log n)$.

We develop new proof techniques to obtain these results. The technique used for the second lower bound is strong enough to establish the first tight time bounds for the PRIORITY model, which is the strongest parallel computation model. We show that finding the maximum of n numbers requires $\Theta(\log\log n)$ steps, generalizing a result of Valiant for parallel computation trees.

Introduction

The parallel random access machine (PRAM) is an important and widely used model of parallel computation. It consists of a set of n processors $P_1, \ldots P_n$, each of which is a random access machine. The processors communicate via a shared memory, whose size is called the *communication width* [VW]. The PRAM is synchronous. It computes a function $f : \Sigma^n \rightarrow \Sigma$, if initially each processor contains one input value, and at the end of the computation the function value is stored in the first shared memory cell.

One cycle of computation consists of three phases. In the compute phase, each processor may perform an arbitrary amount of computation. In the write phase, each processor may write into an arbitrary shared memory cell. In the read phase, each processor may read an arbitrary shared memory cell.

In the write phase it may happen that many processors try to access the same cell, that is, a write conflict occurs. When one restricts the model in a way that such simultaneous writes are forbidden, one gets the Exclusive-Write model [FW]. This model is very

Support for this research was provided by an IBM Faculty Development Award, NSF Grant MCS-8402676, DARPA Contract No. N00039-82-C-0235, an NSERC postgraduate scholarship, and the University of Washington Graduate School Research Fund.

© 1985 ACM 0-89791-151-2/85/005/0048 $00.75

weak. In [CD], Cook and Dwork proved an $\Omega(\log n)$ lower bound for computing the n-way OR function in this model. This function can be computed in depth 1 by an unbounded fan-in circuit and also in constant time by all other PRAM models considered in this paper, even when they only have one shared memory cell.

If simultaneous writes are not forbidden, the following two write conflict resolutions are most commonly used in literature.

- **COMMON**: If several processors want to write into the same memory cell at the same time, they have to write a common value [Ku].
- **PRIORITY**: If several processors want to write into the same memory cell at the same time, the one with smallest index wins [Go].

Both models are widely used for designing parallel algorithms (For example, [SV], [Ga], and [KR] use COMMON; [VT] uses PRIORITY.). There are also some lower bounds known for concurrent-write PRAM's (mentioned below), but the proof techniques are not sensitive enough to separate these two models.

Simultaneous access to the same cell on the COMMON model can be easily implemented by depth 1 OR circuits. This is so easy because the main part of the conflict resolution is done by the program. The program has to make sure that different processors never try to simultaneously write different values in the same cell. In the PRIORITY model, however, write conflicts are resolved entirely by the machine. The extra hardware required to implement PRIORITY conflict-resolution leads one to consider simulating this model by the COMMON model.

It is known that one step of the PRIORITY model can be simulated in constant time by the COMMON model, if we square the number of processors and sufficiently enlarge the shared memory [Ku]. However, in order to understand the difference in the power in the two conflict resolution schemes, we restrict both models to have the same number n of processors and the same communication width.

Let COMMON(m) and PRIORITY(m) denote the respective models with m shared memory cells. The main results of this paper are two lower bounds on the common model, one on COMMON(1) and one on COMMON(∞). These lower bounds are sensitive to the COMMON conflict resolution scheme, so that we can use them to separate COMMON from PRIORITY.

Let $f : \{0,1\}^n \rightarrow S$ be a surjective function. Then a COMMON(1) requires at least $\log_3(|S|)$ steps to compute f. Even if the machine is probabilistic, the same lower bound holds for the expected number of steps. This result implies that it requires at least $\log_3 n$ steps to compute the smallest index of an input variable with value 1. Thus we get a $\log_3 n$ separation between COMMON(1) and PRIORITY(1), because this function can easily be computed by PRIORITY(1) in one step. The result can be generalized to obtain an $\Omega(\log n)$ separation between COMMON(m) and PRIORITY(m), if $m \leq n^\epsilon$, $\epsilon < 1$.

The second result is on the time required by COMMON(∞) to solve the element distinctness problem, that is, to decide whether all n integer inputs are distinct. We prove a lower bound of $\Omega(\log \log \log n)$ steps. This implies an $\Omega(\log \log \log n)$ separation between COMMON(∞) and PRIORITY(∞), because element distinctness can be solved in constant time on PRIORITY(∞).

Actually, our results extend to separating the COMMON model from weaker models than PRIORITY. More specifically, we refer to the ARBITRARY [SV] and ETHERNET [Gr] models. Both separation results extend to these models. In fact, we can demonstrate functions that can be computed in time a factor of $\Omega(\log n)$ faster on a deterministic ETHERNET model than on a probabilistic COMMON model. This is important, since ETHERNET becomes more powerful with randomization allowed.

The proof of the first lower bound is based on a new technique to handle the problem that – in contrast to the Exclusive-Write model – the "information fan-in" on the COMMON model may be arbitrarily large. Therefore the well-known information theory arguments used in [CD] do not apply in this case.

The difficulty in proving the second lower bound is that, due to the infinite shared memory, processors may use indirect addressing in very subtle ways. We handle differently the information that a processor acquires by direct and indirect addressing. In the second case, we use in a crucial way the fact that in the COMMON model, it is useless for a processor to read a cell it has just written into. Note that in the PRIORITY model, such behaviour is extremely useful.

Surprisingly, a simplified version of this proof technique yields a lower bound for PRIORITY(∞). Previously known lower bounds for PRIORITY impose restrictions either on the shared memory size ([VW]) or on the arithmetic power of the processors ([MR],

[FSS]). We show an $\Omega(\log\log n)$ lower bound for finding the maximum of n numbers on PRIORITY. This generalizes the result of Valiant [V] for parallel comparison trees. By an upper bound of Shiloach and Vishkin [SV], our result provides the first tight time bound on the PRIORITY model.

A Lower Bound for Small Memory

Consider a COMMON(1) with n processors, P_1 through P_n. The processors communicate through one cell M of shared memory, which can hold arbitrarily large values. We say such a machine computes a surjective function $f : \{0,1\}^n \rightarrow R$ for some range set R if at the beginning of the computation, P_i has the i^{th} argument (denoted by x_i) in its local memory and, at the end of the computation, the value of $f(x_1, x_2 \ldots x_n)$ appears in M. A particular vector of arguments $(x_1, x_2, \ldots x_n)$ is called an *input*. The variable x_i is called P_i's *private bit*.

Other ways to define the computation of a function appear in the literature. For example, in [VW] the arguments are located in read-only shared memory, one argument per cell. Our definition can be thought of as *public computation*, since the answer must appear in shared memory. It is sometimes useful to define *private computation*, in which each processor is required compute a private answer bit a_i. For the step-by step simulation of a PRIORITY(1) by a COMMON(1) appearing in [FRW], for example, x_i would be 1 if P_i wished to write, and $a_i = 1$ if and only if i is the processor of least index with $x_i = 1$. A good lower bound for public computation can lead to a good lower bound for private computation if $(a_1, a_2, \ldots a_n)$ can be made public in a small number of steps. This is the case with the simulation example, as the unique processor with $a_i = 1$ can take one more step and write i into M.

The following theorem is the main result of this section, giving a lower bound on the number of steps required to publicly compute any function. The lower bound depends only on the number of function values that are possible.

Theorem 1. *In the COMMON(1) model, any algorithm that publicly computes a surjective function* f : $\{0,1\}^n \rightarrow R$ *requires at least* $\log_3 |R|$ *steps on some input.*

Although the theorem as stated applies to the case of a single shared memory cell, it is powerful enough to use in a more general setting. We note that a COMMON(1) can simulate one step of a COMMON(m) in at most m steps, which leads to the following corollary.

Corollary 1. *A COMMON(m) that publicly computes a surjective function* $f : \{0,1\}^n \rightarrow R$ *requires at least* $\frac{\log_3 |R|}{m}$ *steps.*

By specifying a particular function, we can separate the PRIORITY and COMMON models, with the separation varying as a function of the size of shared memory.

Corollary 2. *Simulating one step of a PRIORITY(m) requires* $\Omega(\log n - \log m)$ *steps on a COMMON(m). In particular, when* $m = O(n^\epsilon), \epsilon < 1$, $\Omega(\log n)$ *steps are required.*

Proof of Corollary 2: Divide the input positions into m groups of size $\lfloor n/m \rfloor$ or $\lfloor n/m \rfloor + 1$, and consider the function f whose value is an m-tuple $(a_1, a_2 \ldots a_m)$ such that $a_i = \min\{j | j \text{ is in } i^{th} \text{ group and } x_j = 1\}$. This function can be computed in one step on a PRIORITY(m); in fact, it can be viewed ([FRW]) as a special case of simulating a write step of a PRIORITY(m). The function f has at least $(\frac{n}{m})^m$ possible values. Applying corollary 1 gives a lower bound of $\Omega(\log n - \log m)$ for a COMMON(m). ■

This implies that logarithmic time is required for COMMON to simulate one step of PRIORITY when the size of the shared memory is $O(n^\epsilon)$.

We introduce some terminology to be used in the proof of Theorem 1. Given a particular input, the *history of a computation through step t* is the sequence of values $\{H_0, H_1, \ldots, H_t\}$, where H_i is the contents of the shared memory cell after step i. H_0 is the initial content of M, which we can assume is 0. The *tree of possible computations* has nodes that intuitively correspond to the different states that the PRAM can be in during the course of the computation. Formally, we associate with a node v at depth i a history $\{H_0, H_1, \ldots, H_i\}$, and the set I_v of all inputs that generate this history through step i. An input is said to *reach* node v if it is a member of I_v. The children of v correspond to all possible extensions to the history at v; each child is labelled with a different extension $\{H_0, H_1, \ldots, H_i, H_{i+1}\}$. The last entry in the history associated with a leaf of the tree will be the function value for all inputs that reach that leaf.

With each node v in the computation tree, we can associate a formula f_v in conjunctive normal form, whose variables are the private input bits x_i. This formula will have the property that the set of inputs I_v associated with this node is exactly the set of inputs that satisfy the formula f_v. The construction of these formulas will proceed by induction on the depth of a node.

For the root r of the computation tree, we define f_r to be the empty formula. Now suppose we have a node w with associated history $\{H_0, H_1, \ldots H_{t-1}\}$ and associated formula f_w. Suppose, furthermore, that w has a child v and that the history at v is the history at w extended by the value H_t. This means that for some inputs in I_w, the content of M after the d^{th} step is the value H_t. The action of any processor at step t for an input in I_w is completely determined by the history through step $t-1$ (the history associated with w, which is the same for all inputs in I_w) and by the processor's private bit. Thus, it is possible to determine which private bit values would cause processors to write H_t. At least one of this possibilities must occur; thus inputs with history $\{H_0, H_1, \ldots H_{t-1}, H_t\}$ must satisfy f_w and also a clause consisting of the OR of these possible bit values. For example, if P_1 writes H_4 when $x_1 = 1$, and P_2 does so when $x_2 = 0$, then the added clause would be $(x_1 \vee \overline{x_2})$. In two cases it is not necessary to add a clause: when one processor P_i writes regardless of what his private bit is, and when no processor writes, i.e., $H_t = H_{t-1}$. Since there is only one memory cell, each processor reads its content during every read phase. Therefore, we can assume, without loss of generality, that processors write into the memory cell M only to change its value.

All possible bit values that would have resulted in something other than H_t being written will result in additional clauses. For example, if P_3 would have written H_t' different from H_t if $x_3 = 1$, we add the clause $(\overline{x_3})$, since it is known that $x_3 = 0$. We can also substitute these known values into other clauses. In our example, a clause containing the literal x_3 would have that literal removed; a clause containing the literal $\overline{x_3}$ would be entirely removed.

We call a clause *nontrivial* if it contains more than one literal. Note that at most one nontrivial clause is added to f_w to create the formula at the child of w. The following lemma provides an important bound on the accumulation of nontrivial clauses.

Lemma 1. *If a node w with q children has a formula f_w with c nontrivial clauses, the formula at each child of w has at most $c + 3 - q$ nontrivial clauses.*

Proof: If $q \leq 2$ this follows from the construction, as at most one nontrivial clause is added. Thus we may assume $q > 2$. There are q possible extensions of the computation history at this node. One of them could correspond to the case where no one writes $(H_t = H_{t-1})$, but there are at least $q - 1$ different values that could be written at the next step. No processor may write more than one of these values, for otherwise that processor would always write, and those two values would be the only possibilities. For each value written, we can arbitrarily select one processor that writes it; assume without loss of generality that for $i = 1, 2, \ldots q - 1$, value V_i is written by P_i at this step if literal l_i is true. (Note that l_i is either x_i or $\overline{x_i}$.)

The formula f_w implies that at most one of the literals $l_1, l_2 \ldots l_{q-1}$ is true. Otherwise, there would exist an input in I_w for which two different processors would attempt to simultaneously write different values, a violation of the COMMON model.

Now consider the formula f_v at the child v of w that corresponds to V_{q-1} being written. This is created by first adjoining one nontrivial clause to f, and also some trivial clauses as a result of the knowledge that $\{l_1, l_2, \ldots, l_{q-2}\}$ are all false. This knowledge also results in some substitutions. Let $\beta = (\beta_1, \beta_2, \ldots \beta_n)$ be an input in I_v which makes l_{q-1} true. β satisfies f_w, since I_v is a subset of I_w, and makes $\{l_1, l_2, \ldots, l_{q-2}\}$ false.

For $j = 1 \ldots q - 2$ let β^j be the input obtained from β by complementing β_j (i.e., β^j makes both l_j and l_{q-1} true). The input β^j cannot satisfy f_w, because it makes two literals in $\{l_1, l_2, \ldots, l_{q-2}\}$ true. Let C_j be some clause in f_w that β^j does not satisfy. Since there exists an input in I_w which makes l_j true, and another that makes l_j false, C_j must be nontrivial. The only difference between β and β^j is in the value of the j^{th} bit. Thus C_j must contain the literal $\overline{l_j}$. and $l'_j = x_j$ otherwise. Furthermore, $\overline{l_j}$ is the only literal in C_j that β makes true.

We can now see that for $1 \leq i < j \leq q - 2$, the clauses C_i and C_j are distinct. This follows from the fact that β^i satisfies C_j (it contains the literal $\overline{l_j}$, and β^i makes l_j false) but not C_i. Consider the creation of f_v. The substitutions that follow from the knowledge that $\{l_1, l_2, \ldots l_{q-2}\}$ are false will remove the nontrivial clauses C_i. Thus f_v can have at most $c - (q - 2) + 1$ nontrivial clauses, as required. A similar argument works for the other children of v; in fact, the child that corresponds to the case of no one writing will have at most $c + 2 - q$ nontrivial clauses. ∎

The importance of lemma 1 is that, although we cannot bound the degree of a node in the computation tree, high degree requires accumulating and then destroying nontrivial clauses, and only one nontrivial clause is accumulated per level. We make this idea more precise in the following fashion. Let $L(s,h)$ be the maximum number of leaves in a subtree of height h whose root formula has s clauses.

Lemma 2. $L(s,h) \leq (3 + s/h)^h$. *In particular,* $L(0,h) \leq 3^h$.

Proof: By lemma 1, we have

$$L(s,1) \leq s+3$$
$$L(s,h) \leq \max_{2 \leq q \leq s+3} \{q \cdot L(s+3-q, h-1)\}$$

This can be shown by induction to satisfy the statement of the lemma. ∎

Theorem 1 then follows from the fact that each leaf of the computation tree can be labelled with at most one function value. All function values must appear, so the tree has at least $|S|$ leaves. By Lemma 2, a computation tree of height h has at most 3^h leaves.

We can extend this result, and obtain a theorem similar to Theorem 1 for probabilistic algorithms. In the probabilistic COMMON model, each processor is allowed to make random choices to determine its behaviour at each step. We insist that no sequence of choices results in two processors attempting to write different values into the same cell at the same time. Theorem 2 gives a bound on the expected number of steps to compute a function in terms of the size of its range.

Theorem 2. *In the probabilistic COMMON(1) model, any algorithm that publicly computes a surjective function* $f : \{0,1\}^n \rightarrow R$ *has an expected running time of at least* $\lfloor \log_3 |R| \rfloor$ *steps on some input.*

As in Corollary 2, we obtain a logarithmic separation between the probabilistic COMMON(m) model and the deterministic PRIORITY(m) model, for $m = O(n^\epsilon)$ where $\epsilon < 1$. This separation can also be shown between probabilistic COMMON and models weaker than PRIORITY, such as the ARBITRARY model defined in [FRW]. For these cases, randomization does not help the COMMON model to simulate more powerful models.

Theorem 2 is proved using the following two lemmas.

Lemma 3. *The sum of the root-leaf distances to any set S of leaves in a tree of possible computations is at least* $|S|\lfloor\log_3|S|\rfloor$.

Proof: Let us define a *tree skeleton* to be a tree whose nodes can be labelled with nonnegative integers, such that the root is labelled with zero, and any node labelled with s that has q children has each child labelled no higher than $s + 3 - q$. Lemma 2 is actually a statement about tree skeletons; any computation tree leads in a natural way to a tree skeleton, where the label of a node is just the number of nontrivial clauses in its formula. Let S be our set of chosen leaves, and Q be the sum of the root-leaf distances. We can prune away everything but the root-leaf paths to leaves in S. This still leaves a tree skeleton, for after deleting a node, the labelling at its brothers is still valid. The pruning also leaves Q unchanged.

We can then transform the tree skeleton in a way that will never increase the sum of the root-leaf distances to leaves in S. Suppose we can find two leaves v_1 and v_2, where v_i is at depth t_i and $t_2 - t_1 \geq 2$. We add two children v_1', v_2' to v_1, label them with the same number as v_1, and delete v_2. We remove v_1, v_2 from S and add v_1', v_2'.

Continuing in this fashion, we can obtain a tree skeleton and a set S' of leaves, where $|S| = |S'|$, and all leaves in S' are at depth t or $t-1$. Furthermore, the sum of root-leaf distances to leaves in S' is less than or equal to Q. But Lemma 2 says that t is at least $\lceil \log_3 |S| \rceil$, and the result follows. ∎

Lemma 4. *Let T_1 be the expected running time for a given probabilistic algorithm solving problem P, maximized over all possible inputs. Let T_2 be the average running time for a given input distribution, minimized over all possible deterministic algorithms to solve P. Then $T_1 \geq T_2$.*

Lemma 4 was stated by Yao ([Y]) in a stronger form; the weak form here can be proved in a few lines. We can consider a probabilistic algorithm as a probabilistic distribution of deterministic algorithms. Let A be our set of deterministic algorithms, and I our set of inputs. Let $r[A_i, I_j]$ be the running time of algorithm A_i on input I_j. Suppose our given probabilistic algorithm chooses to run deterministic algorithm A_i with probability p_i, and that our given

2) For $1 \leq i \leq n/2$, processor P_i writes 1 into cell x_i.

3) For $(n/2)+1 \leq i \leq n$, processor P_i reads cell x_i and, if its value is 1, writes 0 into *answer*.

4) For $1 \leq i \leq n$, processor P_i reads *answer*.

5) If *answer* $\neq 0$ then, in parallel,

   5.1) Processors $P_1, \ldots, P_{n/2}$ compute $ED_{n/2}(x_1, \ldots, x_{n/2})$

   5.2) Processors $P_{(n/2)+1}, \ldots, P_n$ compute $ED_{n/2}(x_{(n/2)+1}, \ldots, x_n)$.

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; at the t^{th} execution of step 3, $\frac{n^2}{2^{t+1}}$ 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).

A Lower Bound for COMMON(∞)

In this section we prove the following result.

Theorem 2. *Computing ED_n on a COMMON(∞) requires $\Omega(\log\log\log n)$ steps.*

Together with ALGORITHM 1 this implies an $\Omega(\log\log\log(n))$ separation between the COMMON and PRIORITY models with infinite memory.

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

Lemma 5. *T steps of a COMMON(∞) can be simulated by a T-read COMMON(∞) in T steps.*

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

Theorem 2 now follows directly from the next lemma, by taking k to be $\log\log n$.

Lemma 6. *An k-read COMMON(∞) requires $\Omega(\log\log\log n - \log\log k)$ steps to compute ED_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_i} after step t" is that the configuration of P_i after step t is the same for all allowable inputs with the same value of x_{j_i}.

Let $[n] = \{1, \ldots, n\}$. Consider the situation arranged by the adversary after step t of some algorithm. We use $V_t \subseteq [n]$ 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 simple 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 $S_t \subseteq N$. The indexed set $F_t = \{f_i | i \in [n] - V_t\}$, describes the adversary's assignment of values to the fixed variables. These values are distinct elements of $N - S_t$.

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) $b_i = f_i$ for all $i \in [n] - V_t$,

2) $b_i \in S_t$ for all $i \in V_t$, and

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 $C = \{(A_\ell, B_\ell) | \ell = 1, \ldots, r\}$ with $A_\ell \cap B_\ell = \phi$ is a bipartite cover of G if every edge $\{i, j\} \in E$ belongs to some graph $K(A_\ell, B_\ell)$. The size $s(C)$ of C is $\sum_{\ell=1}^{r}(|A_\ell| + |B_\ell|)$. The bipartite complexity of G is defined to be $\beta(G) = \min\{s(C) | C \text{ is a bipartite cover of } G\}$. For the complete graph on q vertices, K_q, the bipartite complexity is known.

Theorem 4. ([H],[P]) $\beta(K_q) \geq q \log q$.

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.

Now we are ready to formulate the main lemma.

Main Lemma. *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 that each processor P_i only knows one variable with index in V_{t-1}, namely x_{j_i}. 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. (Recall that each processor can read k cells in a step.)*

1) $V_t \subseteq V_{t-1}$ *and* $|V_t| \geq \frac{|V_{t-1}|^2}{|V_{t-1}| + 2nk}$.
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) $E_t \supsetneq E_{t-1} \cap |V_t|^2$ *and* $\beta(G_t) \leq \beta(G_{t-1}) + n(t+k)$, *where* $G_t = (V_t, E_t)$.
5) $F_{t-1} \subseteq F_t \subseteq N - S_t$.

We now complete the proof of lemma 6. Before the computation starts (i.e. after step 0), $V_0 = [n]$, $E_0 = \phi$, $S_0 = N$, and $j_i = i$ 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 with different images under ED_n between which the algorithm could not distinguish. Thus, by Theorem 4, $\beta(G_T) = |V_T| \log(|V_T|)$.

Now, from condition 4, we get that $\beta(G_T) = \beta(K_{|V_T|}) \leq nT(k+T)$.

Since $|V_t| \leq n$, it follows from condition 1 that $|V_t| \geq \frac{|V_t|^2}{|V_t| + 2nk} \geq \frac{|V_t|^2}{3nk}$. By induction, this implies $|V_T| \geq \frac{|V_0|^{2^T}}{(3nk)^{2^T - 1}}$ and $|V_T| \geq \frac{3n}{(3k)^{2^T}}$.

Combining these inequalities, we get

$$nT(k+T) \geq \frac{3n}{(3k)^{2^T}} \log(\frac{3n}{(3k)^{2^T}})$$

Thus $T = \Omega(\log\log\log n - \log\log k)$. ∎

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 7. *Let $f : W \rightarrow D$ 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 8 . *Let $f, g : W \rightarrow D$ 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.*

Lemma 9 . *Let $H(U, L)$ be a finite graph, and let $\alpha(H)$ denote the size of a maximum independent set in H. Then $\alpha(H) \geq \frac{|U|^2}{|U| + 2|L|}$.*

Lemma 7 can be found in [GRS, p. 112], Lemma 8 follows directly from Lemma 7, and Lemma 9 can be found in [B p. 282].

Proof of the main lemma : Consider the sequence of writes performed by some processor P_i up to and including step t. In step $p \leq t$, it 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 k reads P_i executes in step t. It reads from cells $r_{i,1}, \ldots, r_{i,k}$. 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 $d_i^p : S_{t-1} \rightarrow \{0,1\}$ and $v_i^p, w_i^p, r_{i,h} : S_{t-1} \rightarrow N$. 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 this 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 7 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 define an address function as any read function $r_{i,h}$ or any write function w_i^p which is actually used.

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

Proof: Apply Lemma 7 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_q access the same cell then they use the same address function.*

Proof: Apply Lemma 8 successively to every pair of address functions. The result is an infinite subset $S''' \subsetneq S''$ such that every pair of address functions (which we now consider to be functions on S''') are either identical or have disjoint ranges. The adversary therefore restricts the set of allowed inputs to be $I(V_{t-1}, E_{t-1}, S''', F_{t-1})$. ∎

The next observation depends on the the fact that we are using an k-read COMMON(∞) and is the only part of the proof which does so.

Claim 4. *Without loss of generality, if P_i and P_q know the same variable (i.e. $j_i = j_q$), then $w_i^p \neq r_{q,h}$ for all $p = 1, \ldots, t$ and $h = 1, \ldots, k$. In particular, $w^{t_i} \neq r_{i,h}$.)*

Proof: We may assume that every processor has copies of the programs of all other processors. Then P_q 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 k-read COMMON(∞), 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_q reads it. Since P_q can compute this value, it does not have to read cell $w_i^p(x_{j_i})$. (Note that this argument is completely fallacious for the PRIORITY(∞), as shown in ALGORITHM 1.) ∎

Equipped with these two 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 $g_1, \ldots, g_r$ are the 1-1 address functions. For each g_ℓ, define

$A_\ell = \{q \in V_{t-1} | x_q = x_{j_i}$ and $w_i^p = g_\ell$ for some $i \in [n]$ and $p \in [t]\}$

$B_\ell = \{q \in V_{t-1} | x_q = x_{j_i}$ and $r_{i,h} = g_\ell$ for some $i \in [n]$ and $h \in [k]\}$.

Intuitively, $q \in A_\ell$ if some processor that knows x_q uses g_ℓ as a write function. Similarly, $q \in B_\ell$ if some processor that knows x_q uses g_ℓ as its read function.

As a corollary to claim 4, we get the following result.

Claim 5. $A_\ell \cap B_\ell = \phi$.

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

Claim 6. $\beta(G'') \leq \beta(G) + (k+t)n$.

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

Proof: By claims 3 and 4, every processor that uses a 1-1 read function reads the initial contents of the cell, namely '0'. Thus the read imparts no information about any input known to be in $I(V_{t-1}, E'', S''', F_{t-1})$.

Constant address functions:

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

Claim 8. *If w is a constant address function, then the contents of cell w 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 k-read COMMON(∞). In fact, we can assume that, for cells which are accessed by constant address functions, the PRIORITY write conflict resolution scheme is used. Furthermore we can allow these cells to be accessed during more than one step. Consider the last step $p \leq t$ in which cell w was accessed for writing. Let P_i be the processor with lowest index writing into cell w in step p. By our construction, P_i always writes to cell w at step p. Therefore the contents of cell w can only depend on x_{j_i}. ∎

Thus we can assume that all constant address functions used up to and including step t are different.

Consider the graph $H(V_{t-1}, L)$, where $\{i, k\} \in L$ if some processor knowing x_i uses a constant read function w in step t, and the contents of w depends on x_q (i.e. after step t this processor "knows" both x_i and x_q). Note that $|L| \leq nk$, the total number of reads in step t. Apply Lemma 9 to this graph to obtain an independent set of vertices $V_t \subsetneq V_{t-1}$. such that $|V_t| \geq \dfrac{|V_{t-1}|^2}{|V_{t-1}| + 2nk}$. 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 $f_i \in S'''$. 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 $\beta(G_t) \leq \beta(G'')$. ∎

A Lower Bound for PRIORITY(∞)

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

Theorem 5. *A PRIORITY(∞) 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(∞) 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.

Lemma 10. *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 $|V_t| \geq \dfrac{|V_{t-1}|^2}{|V_{t-1}| + 2n}$,

2) Each processor knows only one variable with index in V_t.

3) $S_t \subsetneq S_{t-1}$ *and* S_t *is infinite, and*

4) $F_{t-1} \subseteq F_t \subseteq N - S_t$.

If M requires T steps then $|V_T| = 1$; otherwise no processor can determine the output. From Lemma 10 and the fact that $|V_0| = n$, we can derive $T = \Omega(\log\log n)$. This concludes the proof of Theorem 5. ∎

Proof of lemma 10 : Consider the proof of the main lemma for $k = 1$. It is sufficient to show that for $J(V_t, S_t, F_t)$ the analogues of claim 7 (which takes care of 1-1 address functions) and claim 9 (which takes care of constant address functions) hold.

The proof of claim 9 does not use the properties of the k-read COMMON(∞). Since $J(V_t, S_t, F_t) \subseteq I(V_t, E_t, S_t, F_t)$, the analogous result for the PRIORITY (∞) is true.

However, claim 5 is derived from claim 2 which, in turn, depends on the properties of the k-read COMMON(∞). Fortunately, the analogue of claim 7 follows directly from claim 3, since all input variables are assumed to have distinct values. Because the proof of claim 3 does not use the properties of the k-read COMMON(∞), its analogue also holds for the PRIORITY(∞). ∎

References

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

[CD] Cook, S.A., and Dwork, C. *Bounds on the Time for Parallel RAMs to Compute Simple Functions*, Proc. 14th Annual ACM Symposium on Theory of Computing, 1982, pp.231-233.

[FRW] Fich, F.E., Ragde, P.L., and Wigderson, A. *Relations Between Concurrent-Write Models of Parallel Computation.* Proc. 3rd Annual ACM Symposium on Principles of Distibuted Computing, 1984, pp. 179-189.

[FW] Fortune,S., and Wyllie, J. *Parallelism in Random Access Machines*, Proc. 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 114-118.

[Ga] Galil, Z. *Optimal Parallel Algorithms for String Matching*, Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 240-248.

[Go] Goldschlager, L. *A Unified Approach to Models of Synchronous Parallel Machines*, Journal of the ACM, vol. 29, no. 4, 1982, pp. 1073-1086.

[GRS] Graham, R.L., Rothschild, B.L., and Spencer, J.H. *Ramsey Theory*, Wiley and Sons, 1980.

[Gr] Greenberg, A. *Efficient Algorithms for Multiple Access Channels*, Ph.D Thesis, University of Washington, 1983.

[Ha] Hansel, G. *Nombre minimal de contacts de fermature nessecaires pour realiser une fonction booleenne symetrique de n variables*, C. R Acad. Sci. Paris 258,1964, pp. 6037-6040.

[KMR] Kannan, R., Miller, G., and Rudolph, L. *Sublinear Parallel Algorithm for Computing the Greatest Common Divisor of Two Integers*, Proc. 25th Annual Symposium on Foundations of Computer Science, 1984, pp. 7-11.

[Ku] Kucera, L. *Parallel Computation and Conflicts in Memory Access*, Information Processing Letters, vol. 14, no. 2, 1982, pp. 93-96.

[MR] Meyer Auf der Heide, F., and Reischuk, R. *On the Limits to Speed Up Parallel Machines by Large Hardware and Unbounded Communication*, Proc. 25th Annual Symposium on Foundations of Computer Science, 1984, pp. 56-64.

[Pi] Pippenger, N. *An Information-Theoretic Method in Combinatorial Theory*, Journal of Combinatorial Theory, vol. 23, no. 1, July 1977, pp. 99-104.

[SV] Shiloach, Y., and Vishkin, U. *Finding The Maximum, Merging and Sorting On Parallel Models of Computation*, J.Alg, v.2, 1981, pp. 88-102.

[TV] Tarjan, R.E, Vishkin, U. *Finding Biconnected Components and Computing Tree Functions in Logarithmic Parallel Time*, Proc. 25th Annual Symposium on Foundations of Computer Science, 1984, pp. 12-20.

[V] Valiant, L. *Parallelism in Computation Problems*, SIAM J. Comput., vol. 4, no. 3, 1975, pp. 348-355.

[VW] Vishkin, U., and Wigderson, A. *Trade-offs Between Depth and Width in Parallel Computation*, to appear in SIAM J. Computing.

[Y] Yao, A. Probabilistic Computations: *Towards a Unified Measure of Complexity*, Proc. 18th Annual Symposium on Foundations of Computer Science, 1977, pp.222-227.