Incomparability In Parallel Computation
Extended Abstract

Vince Grolmusz
University of Chicago*
Eötvös University, Budapest

Prabhakar Ragde
University of Toronto

1. Abstract

We consider concurrent-write PRAMs with n processors of unlimited computational power and infinite shared memory. (I.e. the number of processors is equal to the number of input variables.) Several different models (COMMON, ARBITRARY, PRIORITY) have been used for algorithm design in the literature; these models differ in their method of write-conflict resolution. Their relative powers has recently been studied by several authors [CDR], [FMW], [FRW1,2,3], [RSSW]. The significance of such problems is in studying the barriers to information propagation under various constraints. We consider the COLLISION model, where simultaneous attempt to write in the same cell results in a special mark in the cell. We compare this model with other known models. Our main result is that the COMMON and COLLISION models are incomparable. The harder part of this result is based on an $\Omega(\log\log\log n)$ lower bound on the COLLISION model for the following task: write input x_i in cell numbered x_i (trivially solved in $O(1)$ time on the COMMON model). The proof uses combinatorial arguments including Turán's Theorem for graphs and the Sunflower Theorem of Erdős and Rado for set systems.

2. Introduction

The concurrent-read concurrent-write parallel random access machine (often denoted CRCW PRAM) has proved a useful model in the design of highly parallel algorithms. In this model, n processors $P_1, P_2, \ldots, P_n$ are allowed synchronous read/write access to a shared memory consisting of cells $M_1, M_2, \ldots$. Each step of computation consists of three phases. In the read phase, each processor reads from one cell; simultaneous reads from the same cell are permitted. In the compute phase, each processor performs local computation. For the purposes of lower bounds, we make no assumptions about the size of the local memory of a processor, or about its instruction set; an arbitrary amount of local computation is allowed. In the write phase, each processor may write into one cell; simultaneous writes are also allowed, and write-conflicts are resolved by one of several methods.

Many methods of write-conflict resolution appear in the literature, and algorithm designers tend to use the variation that best allows their particular techniques to succeed. We list some of these variations below.

*mailing address: Department of Computer Science, University of Chicago, 1100 E. 58th Street, CHICAGO IL 60637

- COMMON: Write conflicts are disallowed; if several processors want to write into the same cell simultaneously, they must all be writing the same value. This model was defined in [Ku]; examples of its use include [SV], [Ga], [KR].
- COLLISION: If two or more processors simultaneously attempt to write to a cell, a special *collision* symbol appears in the cell. This is a natural generalization of the Ethernet [Gr], first defined in [FRW2].
- ARBITRARY: If several processors simultaneously write different values, one of the values written appears in the cell, but it is impossible to predict in advance which value will appear. This model was defined in [V], and is used in [CV] and [Gaz].
- PRIORITY: If several processors simultaneously write different values, the value that appears is the one written by the processor of lowest index. This model was defined in [Go], and is used in [TV].

We introduce a new model, which is similar to COLLISION in that write conflicts can be detected, even if none of the values written can be recovered.

- TOLERANT: If several processors attempt to simultaneously write to a cell, the contents of that cell do not change.

All of these models have the ability to compute the OR of n bits in $O(1)$ steps; [CDR] showed that a PRAM in which simultaneous writes are forbidden required $\Omega(\log n)$ time to do this. The relationships between the various CRCW models are less clear. It is simple to show that ARBITRARY is at least as powerful as COMMON, COLLISION, and TOLERANT, and that PRIORITY is at least as powerful as ARBITRARY. [FRW1] and [FRW2] show various separations between these models; in particular, the separation is $\Theta(\log n)$ between COMMON or COLLISION and ARBITRARY, or between ARBITRARY and PRIORITY, if the number of memory cells is restricted to $O(n^{1-\epsilon})$ for some $\epsilon > 0$. (Note that if the number of memory cells is restricted, inputs must be initially distributed among the local memories of each processor.)

[CSV] shows that if the instruction set of a processor is restricted (for example, multiplication is not permitted) and the size of shared memory is bounded by a polynomial in n, then any of these PRAM models are depth-equivalent to unbounded fan-in circuits modulo a polynomial blowup in hardware (processors and cells, or gates). Thus all circuit lower bounds can be carried over. [BH] shows tight bounds on computing the parity of n bits on all of these models. [LY] extends many of these results to the case where the input is stored in a special read-only memory.

[Ku] showed that any one of these models could simulate any other in constant time if the simulating machine was allowed n^2 processors. This is evidence that proving lower bounds separating these machines (when the number of processors is held constant) can be very difficult without assuming restrictions on instruction set or size of shared memory. Furthermore, by "gathering variables", these machines (without using simultaneous writes or reads) can compute *any* function of the input variables in $O(\log n)$ steps.

The only separation result known that does not assume such restrictions is the lower bound demonstrated in [FMW] of $\Omega(\log\log\log n)$ for solving element distinctness (testing whether n integers are distinct) on COMMON. This bound is improved in [RSSW] to $\Omega(\sqrt{\log n})$. It is natural to conjecture a separation of $\Theta(\log n)$ between any of these models;

however, [FRW3] showed a surprising simulation of one step of PRIORITY by $O\left(\frac{\log n}{\log\log n}\right)$ steps of COMMON, if the size of shared memory on the COMMON machine is increased by a multiplicative factor of n. This increase, of course, has no effect if the size of shared memory is unbounded.

In this paper we contribute separation results in which no restrictions are made on the size of shared memory or on the instruction sets of processors. We prove a lower bound of $\Omega(\log\log\log n)$ for a particular problem on the COLLISION model. This, when taken in conjunction with the element distinctness result and a further reduction to another problem, implies that the powers of COMMON and COLLISION are incomparable. That is, there can exist no simulation of one step of one of these models by $O(1)$ steps of the other. This answers an open question posed in [FRW2]. In contrast, that paper showed that if the size of shared memory is restricted to one cell, one step of COMMON could be simulated by $O(1)$ steps of COLLISION, while $\Theta(\log n)$ steps were required in the reverse direction.

We also separate COLLISION from TOLERANT, and ARBITRARY from COLLISION. Finally, we show that the powers of TOLERANT and COMMON are also incomparable. These results show the usefulness of Ramsey-theoretic arguments in proving lower bounds on highly parallel machines, and contribute towards our growing understanding of how processors communicate in such a powerful environment.

3. The Main Lower Bound

The input to a PRAM will be an n-tuple of positive integers $(x_1, x_2, \ldots, x_n)$, where x_i is initially stored in the local memory of processor P_i. (Since memory is unbounded, this is equivalent to the situation where the input variables are stored in shared memory, one to a cell.) The output of the PRAM will also be an n-tuple of positive integers $(b_1, b_2, \ldots, b_n)$, and we can assume that b_i is stored in the local memory of processor P_i.

Our main result concerns a problem we call the PAIR problem. We call x_i a *pair* of x_j if $x_i = x_j$. The PAIR problem asks each processor to discover a pair of their input value, if one exists. More precisely, the output vector satisfies:

$$b_i = \begin{cases} j, & \text{if } x_j = x_i \text{ for some } j \neq i \\ 0, & \text{otherwise.} \end{cases}$$

Theorem 1. *The PAIR problem can be solved in $O(1)$ steps on* ARBITRARY.

Proof: At the first step, each processor P_i writes i into cell M_{x_i}. Each processor then reads the cell it just wrote into. If a processor reads its own index, then it learns nothing; but if it reads a value it did not write, then that value is the index of a pair of its own value. The write is then repeated, except that processors that read their own index do not participate. Finally, each processor P_i that did not participate reads cell M_{x_i}. At this point, every processor either knows a pair of its input variable, or knows that no pair exists. ∎

Notice that the infinite memory is required in this algorithm. The algorithm cannot work on COLLISION; after the first step, a processor may learn that its input variable has a pair, but will not know what the index of that pair is. The lower bound for element distinctness on COMMON took advantage of the fact that, in that machine, a processor cannot discover, after a write step, if any other processors wrote into the same cell. In COLLISION, a processor can make that discovery, but it cannot know who attempted to write to the same cell. Thus, intuitively, the PAIR problem is hard for COLLISION. The following lower bound formalizes this idea; it is our main result.

Theorem 2. *There exists an input on which* COLLISION *requires* $\Omega(\log\log\log n)$ *steps to solve the PAIR problem.*

As a corollary to the two theorems above, we have an $\Omega(\log\log\log n)$ separation between COLLISION and ARBITRARY, and thus between COLLISION and PRIORITY. In the remainder of this section we briefly outline the main ideas of the proof of Theorem 2.

Our lower bound proof starts by assuming a COLLISION PRAM that solves the PAIR problem. We proceed to construct, for each step t, a set of "allowable" inputs such that the machine cannot answer correctly after step t for some allowable input. As long as there exists an allowable input, we have a lower bound of t steps.

In our allowable inputs, every input variable x_i will be equal to exactly one other input variable x_j; in fact, the lower bound holds even if inputs are restricted to be of this form. (This will be important in the next section.) The problem then reduces to determining the (unique) pair of each input variable. However, in order to fully describe the set of allowable inputs, we will require some additional sets, which are described below.

- A set $\mathcal{U}_t$ of *free* variables. This set will be partitioned into sets $\{U_1, U_2, \ldots\}$ of equal size. We refer to each one of these sets as a *U-set.* We denote the total number of variables in $\mathcal{U}_t$ as v_t, and we denote the number of variables in any U-set as u_t. In any allowable input, the values of two variables in different U-sets must be different. Since every variable has a pair, the members of each pair must be contained in the same U-set.

Intuitively, the algorithm has succeeded, after t steps, of determining only that the pair of any input variable x_i in a U-set is some other variable in that U-set.

- An infinite set S_t of integers. The allowable inputs will have values for the free variables chosen from S_t.
- A set $\mathcal{M}_t$ of *fixed* variables. Any variable that is not free will be fixed. A fixed variable has the same value in any allowable input. It is set to some value that is smaller than any value in S_t, and its pair (another fixed variable) remains the same over all allowable inputs.

We can now state three inductive hypotheses that hold by construction.

1) The state of each processor up to and after step t, when we restrict attention to allowable inputs, is a function of at most one free variable. For a given processor P, this variable, if it exists, is the same over all allowable inputs. We say that the processor *knows* that variable.

If this condition holds, and a U-set has at least four variables, then any processor knows at most one of these, and so no processor can know the pair of any free variable.

Because of condition 1), the choice of which cell processor P_i reads at a given step is also a function of the one free variable that P_i knows. We call this the *read access function* of P_i. When we think of a read access function, we consider it as a function of some variable z that can take on values from S_t; to use that function, different processors may substitute different free variables for z. Similarly, the write access function of P_i is a function of that one free variable.

2) For every step $t' \leq t$, a processor either does not write at step t' for all allowable inputs or always writes. Any read or write access function at step t' is either constant or 1–1; any two such functions used before or at step t are either identical, or have disjoint ranges.

Condition 1) ensures that a processor cannot "know" anything about the pair of the variable it knows, apart from the fact that it is somewhere in the U-set of that variable. Condition 3) will ensure that it can find very little information in memory, also.

3) Consider any write access function f used at step $t' \leq t$. Let V be the set of free variables used by processors at step t' as arguments to f. For any U-set, the intersection of V and that U-set is either empty, contains only one variable, or is the whole U-set.

Intuitively, f could be used to convey information about variables in the intersection. But as a result of condition 3), f conveys either information about only one variable, or it conveys no information at all, since two processors will write to every place accessed.

We can slightly modify the COLLISION PRAM, without decreasing its power. We allow the processors to read $t-1$ cells at step t, simultaneously. But those cells, if they were written into at all, must have been written into at steps $1, 2, \ldots, t-1$ respectively. Furthermore, we disallow overwriting – that is, a cell may be written into only once. One can prove easily that for infinite memory, this does not decrease the power of the PRAM.

Lemma 3. *If f is a function with infinite domain, then there exists an infinite subdomain such that f is either 1–1 or constant when restricted to the subdomain. If f, g are two 1–1 functions with infinite common domain, then there exists an infinite common subdomain such that f and g are either identical or have disjoint ranges on this subdomain.* ∎

This lemma was used in [FMW] to restrict the manner in which processors may communicate with each other. We apply this lemma to all pairs of read and write access functions used before or during step $t+1$, thereby further reducing S_{t+1}.

Now let us consider the information learned when some processor P knowing x_j uses it in some read access function f at step $t+1$ to read a cell that was written (if at all) at step t'. If P reads according to f, then P can only read values written by processors that wrote using f at step t'. First, we consider the case where f is 1–1. Let V be the set of variables substituted into f at that step by such processors, and use inductive condition 3).

If the intersection of V with the U-set that contains x_j is empty, then P reads 0 (the initial value in that cell). If it is the whole U-set, then some processor Q wrote using x_j at the same step. But the pair of x_j is in the set V, and so no matter what that pair is, some processor used it to substitute into f. Thus two processors wrote into the cell read by P, and P reads a collision symbol.

If the intersection is one element (say x_i), then P could learn something about x_i.

If f is constant, then the only way that P can gain information from the read is if only one processor Q wrote into the cell specified at time t'. In this case, P (knowing x_j) can learn something about the variable x_i known by Q.

Let us construct a graph whose nodes are the free variables; there is an edge between x_i and x_j if a processor knowing x_j learns something about x_i (in the sense described above). Each processor can contribute at most t edges to this graph, since it reads at most t cells at step $t+1$. Thus by Turán's theorem [B], there exists an independent set of size $\frac{v_t^2}{v_t+2nt} \geq \frac{v_t^2}{3nt}$. We throw away everything but the independent set. (When we throw away variables during the construction, we actually save them until the end of the step under consideration, and then add them to $\mathcal{M}_t$, suitably paired off.)

We do not want any one free variable to be known by too many processors. So if we throw away those free variables which are known by more the twice the average number of processors, we are left with at least $\frac{v_t^2}{6nt}$ free variables, each known by at most $\frac{6n^2t}{v_t^2}$ processors.

If we throw away all U-sets that contain less than $\frac{u_t v_t}{12nt}$ variables, we are left with at least $\frac{v_t^2}{12u_t nt}$ U-sets. We then throw away variables within those sets until they all have size $\frac{u_t v_t}{12nt}$.

This brings us to taking care of condition 3). For any write access function used at step $t+1$, let V be the set of variables that processors substitute into that write access function. We call such a set V an f-set. Condition 3) does not hold yet for such sets V; we must partition each U-set into many new U-sets such that it does hold.

Lemma 4. *Given any number of f-sets within a U-set of size m, with the property that any element of the U-set is in at most ℓ f-sets, we can partition the U-set into new U-sets of size $m^{\frac{1}{\ell+2}}$ (we may have to throw away $(\ell!)m^{\frac{\ell+1}{\ell+2}}$ elements) such that the intersection of any f-set with a new U-set is empty, has one element, or is the whole new U-set.*

The proof uses a theorem of Erdős and Rado [ER] which states that a family of at least $\ell! k^{\ell+1}$ (not necessarily different) sets of size at most ℓ, there is a sunflower formed by k sets. The details are omitted here. ∎

We define $\mathcal{U}_{t+1}$ by applying this lemma to each U-set from $\mathcal{U}_t$ (after all the throwing out of variables) and partitioning each U-set into many new U-sets. We can see, that if $t \leq \frac{1}{4}\log\log\log n$, then, for large enough n, the waste from partitioning a U-set does not amount to more than half the size of the original U-set; the waste is thrown away. We suitable pair off and fix the values of discarded variables, and define $\mathcal{M}_{t+1}$.

Let us denote by ℓ_t the maximum number of processors which may know a free variable after step t. The resulting recurrence equations are:

$$\ell_{t+1} \leq \frac{6n^2t}{v_t^2}$$

$$u_{t+1} \geq \left(\frac{u_t v_t}{12nt}\right)^{\frac{1}{\ell_{t+1}+1}} - 1$$

$$v_{t+1} \geq \frac{v_t^3}{288n^2t^2} - u_{t+1}$$

By easy estimations we can obtain inequalities of the form:

$$\ell_t \leq 2^{2^{O(t)}}$$
$$v_t \geq \frac{n}{2^{2^{O(t)}}}$$
$$u_t \geq n^{\frac{1}{2^{2^{O(t)}}}}$$

Since we require only that $u_T \geq 4$ for a lower bound of T steps, we obtain a lower bound of $T = \Omega(\log\log\log n)$. ■

We note that it is simple to prove finite version of Lemmas 3 and 4 using the above Erdős-Rado Theorem. We get the following:

Theorem 5. *There exists a polynomial $p(n)$ such that* COLLISION *requires $\Omega(\log\log\log n)$ steps to solve the PAIR problem even if inputs are restricted to numbers with at most $p(n)$ bits.* ■

This implies that if the size of shared memory is restricted to $2^{p(n)}$, ARBITRARY is still more powerful than COLLISION. As for upper bounds for the PAIR problem, the best known algorithm simply simulates the $O(1)$ ARBITRARY algorithm. Using the simulation from [FRW3], the PAIR problem can be solved in $O(\frac{\log n}{\log\log n})$ steps on COLLISION.

4. Further Results

Element distinctness can easily be reduced to the PAIR problem, and so the PAIR problem is hard for both COMMON and COLLISION. COLLISION can solve element distinctness in $O(1)$ steps, but COMMON cannot. We were unable to find a function or a decision problem that is easy for COMMON but hard for COLLISION. However, we can specify a task which COMMON can do quickly, but COLLISION cannot. We call this Writing Task 1, and it simply requires that the value of x_i must appear in cell M_{x_i}, where x_i is positive integer for all i. This can be done in one step on COMMON, but is hard for COLLISION. This does not imply that there exists a function or decision problem hard for COLLISION and easy for COMMON. It does show, however, that a step-by-step simulation (as all simulations so far have been) is impossible.

Theorem 6. *If a* COLLISION *PRAM can solve Writing Task in T steps, then there exists an infinite set R of positive integers such that a* COLLISION *PRAM can solve the PAIR problem in $T + 2$ steps for inputs restricted so that values are chosen from R and every variable has a unique pair.*

Corollary: Writing Task requires $\Omega(\log\log\log n)$ steps on COLLISION.

The proof of this theorem uses Ramsey's theorem [R] to aid in the reduction. Suppose we are given a COLLISION PRAM that solves Writing Task in T steps. Without loss of generality, we may assume the following: in each write phase, each processor P, instead of writing a word w to a cell c, writes its entire *history* (its name, the value of its input variable, and the contents of every shared memory cell it has read).

This modification does not decrease the power of the COLLISION PRAM, since if a processor Q reads this cell c then it can compute the word w, simply by simulating the computation of some other processors (processor P, and those processors whose history was read by P, and so on). Note that we cannot ensure this on COMMON.

We define the *communication pattern* of the PRAM on a given input to be the $n \times T$ matrix A, where

$$A_{ij} = \begin{cases} (P_k, t) & \text{if } P_i \text{ read a cell at step } j \text{ that was written by } P_k \text{ at step } t, \\ * & \text{if } P_i \text{ read a collision symbol at step } j, \\ 0 & \text{if } P_i \text{ read an empty cell at step } j \end{cases}$$

We say that a processor P *directly learns* variable x_i if either P gets x_i as input (in which case $P = P_i$), or if P has read a cell that was last written into by a processor that directly learned x_i.

It is easy to see that the history of each processors depend only on the communication pattern and on the values of the directly learned variables. For a COLLISION PRAM which solves a problem in T steps, there are only a finite number of different communication patterns. The communication patterns can be used to colour certain sets of integers in order to get an infinite input set by Ramsey's Theorem [R], such that for these inputs if a processor P writes x_i to cell M_{x_i}, then P directly learned x_i or a pair of x_i. Then P will know x_i's index as well, and the PRAM can solve the PAIR problem in two more steps, as we have seen in the proof of Theorem 1. The details are omitted. ■

This proves the incomparability of COMMON and COLLISION. It is easy to show that one step of TOLERANT can be simulated by $O(1)$ steps of COLLISION. Element distinctness can be solved quickly on TOLERANT but not on COMMON; we can demonstrate another task that can be solved quickly on COMMON but not on TOLERANT. This simply requires that a collision symbol be written to cell M_{x_i}, for all i. This can be done in $O(1)$ steps on COLLISION, but even if we include the collision symbol * in the alphabet of TOLERANT, it cannot be done quickly. The details are omitted here.

5. Further Work and Open Problems

We believe that all lower bounds of the form $\Omega(\log\log\log n)$ described in this paper can be improved by the use of techniques from [MW] and [RSSW] to permit processors to learn more than one variable. However, this requires the use of stronger Ramsey theory, and will no doubt be complicated.

It is still possible that any function computable by COMMON in t steps can be computed by COLLISION in $O(t)$ steps, and vice versa. We have only shown that settling this question affirmatively would require something like a characterization of the functions thus computable, and not merely a step-by-step simulation. It would also be nice to be able to separate these models with 0/1 inputs, instead of unbounded inputs.

Finally we note that one step of PRIORITY can be simulated by $O\left(\frac{\log n}{\log\log n}\right)$ steps of COMMON or COLLISION. The simulation fails to work on TOLERANT. Can we do better than the trivial $O(\log n)$ simulation?

6. References

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

[BH] Beame, P., and Hastad, J. *Optimal Bounds for Decision Problems on the CRCW PRAM*, Proc. 19th ACM Symposium on Theory of Computing, 1987.

[CDR] Cook, S.A., Dwork, C., and Reischuk, R. *Upper and Lower Bounds for Parallel Random Access Machines without Simultaneous Writes*, SIAM J. Computing, vol. 15, no. 1, 1986, pp. 87-97.

[CSV] Chandra, A., Stockmeyer, L., and Vishkin, U. *A Complexity Theory of Unbounded Fan-in Parallelism*, Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science, 1982, pp. 1-13.

[CV] Cole, R., and Vishkin, U. *Approximate and Exact Parallel Scheduling With Applications to List, Tree, and Graph Problems*, Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1986.

[ER] Erdős, P. and Rado, R. *Intersection Theorems for Systems of Sets*, J. London Math. Soc. , vol. 35, 1960, pp. 85-90.

[FMW] Fich, F.E., Meyer auf der Heide, F., and Wigderson, A. *Lower Bounds for Parallel Random-Access Machines with Unbounded Shared Memory*, Advances In Computing Research, 1986.

[FRW1] Fich, F.E., Ragde, P.L., and Wigderson, A.*Relations Between Concurrent-Write Models of Parallel Computation (preliminary version)*, Proc. 3rd Annual ACM Symposium on Principles of Distributed Computing, 1984, pp. 179-189.

[FRW2] Fich, F.E., Ragde, P.L., and Wigderson, A. *Relations Between Concurrent-Write Models of Parallel Computation*, manuscript, 1986.

[FRW3] Fich, F.E., Ragde, P.L., and Wigderson, A. *Simulations Among Concurrent-Write PRAMs*, manuscript, 1986.

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

[Gaz] Gazit, H. *An Optimal Randomized Algorithm for Finding Connected Components in Graphs*, Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1986.

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

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

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

[LY] Li, M., and Yesha, Y. *New Lower Bounds for Parallel Computation*, Proc. 18th Annual ACM Symposium on Theory of Computing, 1986, pp. 177-187.

[R] Ramsey, F.P. *On A Problem of Formal Logic*, Proc. London Math. Soc., ser. 2, vol. 30, 1930, pp. 264-286.

[RSSW] Ragde, P.L., Szemerédi, A., Steiger, W., and Wigderson, A. *The Parallel Complexity of Element Distinctness is* $\Omega(\sqrt{\log n})$, manuscript, 1986.

[SV] Shiloach, Y., and Vishkin, U. *Finding the Maximum, Merging, and Sorting on Parallel Models of Computation*, J. Algorithms, vol. 2, 1981, pp. 88-102.

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

[V] Vishkin, U. *Implementation of Simultaneous Memory Address Access in Models That Forbid It*, Tech. Rept. 210, Israel Institute of Technology, July 1981.