ERCW PRAMs and Optical Communication

Philip D. MacKenzie* Vijaya Ramachandran†

Dept. of Computer Sciences
University of Texas
Austin, TX 78712-1188

June 7, 1996

Abstract

This paper presents algorithms and lower bounds for several fundamental problems on the Exclusive Read, Concurrent Write Parallel Random Access Machine (ERCW PRAM) and some results for unbounded fan-in, bounded fan-out (or 'BFO') circuits. Our results for these two models are of importance because of the close relationship of the ERCW model to the OCPC model, a model of parallel computing based on dynamically reconfigurable optical networks, and of BFO circuits to the OCPC model with limited dynamic reconfiguration ability.

Topics: Parallel Algorithms, Theory of Parallel and Distributed Computing.

*This research was supported by Texas Advanced Research Projects Grant 003658480. (philmac@cs.utexas.edu)

†This research was supported in part by Texas Advanced Research Projects Grants 003658480 and 003658386, and NSF Grant CCR 90-23059. (vlr@cs.utexas.edu)

1 Introduction

In this paper we develop algorithms and lower bounds for fundamental problems on the Exclusive Read Concurrent Write (ERCW) Parallel Random Access Machine (PRAM) model. The ERCW PRAM model has not received much attention, due in part to a general belief that concurrent writing does not add much power to a model without concurrent reading. We show that this is not always the case by presenting algorithms that solve problems on the ERCW PRAM much faster than they could be solved on the EREW PRAM. (See [34] for more details on the different PRAM models.) We further motivate the ERCW PRAM by its relation to parallel computers with optical communication networks. Since there is no 'queue' delay in optical communication networks, the ERCW PRAM is a better model for parallel machines with such networks than the recently proposed QRQW (or ERQW) model [23].

Many results for the ERCW PRAM follow directly from results for the EREW PRAM or CRCW PRAM. For instance, the global OR of n bits can be found in constant time on an n processor ERCW PRAM, as on a CRCW PRAM, but broadcasting 1 bit to n processors requires $\Theta(\log n)$ steps, as on an EREW PRAM. The result for broadcasting implies that computing the prefix sums of n inputs and merging two lists of size n both require $\Theta(\log n)$ time also. However, some results obtained directly from EREW PRAM and CRCW PRAM results do not give tight bounds. For instance, the problem of computing the parity of n bits on the ERCW PRAM has a lower bound of $\Omega(\log n/\log\log n)$ from the result for the CRCW PRAM, and an upper bound of $O(\log n)$ from the EREW PRAM. Tight bounds are not known for the ERCW PRAM. Furthermore, tight bounds are not known for many other problems, including the problems of compaction and finding the maximum. In this paper, however, we make significant progress towards developing tighter bounds for these and other problems.

Our results for the ERCW PRAM (here n is the size of the input, and all algorithms perform linear work except as noted) include a k-compaction algorithm that runs in $O(\log\log n + \log k)$ time; a randomized algorithm for k-compaction that runs in $O(\log k)$ expected time; a randomized algorithm for approximate k-compaction that runs in $O(\log\log k)$ time, with failure probability $1/k$; an algorithm for finding the maximum of inputs in the range $[1, n]$ that runs in $O(\log\log n)$ time; an algorithm for chaining that runs in $O(\log\log n)$ time; an algorithm for integer chain-sorting (linear-size integers) that runs in $O(\log\log n)$ time; and an algorithm for integer sorting (polynomial-size integers) that runs in $O(\log n)$ using almost linear work.

We present two lower bounds results for the ERCW PRAM: a lower bound of $\Omega(\sqrt{\log\log n})$ time for solving compaction, and a lower bound of $\Omega(\sqrt{\log n})$ for finding the maximum of general inputs. (The former was discovered independently by Goldberg and Jerrum, and led to the $\Omega(\sqrt{\log\log n})$ lower bound on h-relation routing in Goldberg, Jerrum and MacKenzie [25].)

Finally, we consider unbounded fan-in, bounded fan-out (BFO) circuits. The computations on such circuits can be mapped optimally onto an ERCW PRAM as oblivious algorithms. We show that any BFO circuit for adding two n-bit integers, merging a bit into an n bit sorted sequence, sorting n bits, computing the prefix sums or parity of n bits requires $\Omega(\log n)$ depth. Let $TH_{k,n}$ denote the threshold function which outputs 1 if and only if at least k of the inputs are equal to 1. We show that $TH_{k,n}$ can be computed by a linear size, $O(\log\log n + \log k)$ depth circuit, and that any BFO circuit which computes $TH_{k,n}$ requires $\Omega(\log\log n + \log k)$ depth.

As further motivation for studying the ERCW PRAM model we show how it is related to a model of massively parallel computing based on dynamically reconfigurable optical networks. Specifically, we show that the ERCW PRAM (using the 'Tolerant' protocol for resolving write

conflicts) with n global memory cells and unlimited local memory is computationally equivalent to the OCPC (Optical Communication Parallel Computer) model [1, 21, 22, 24, 43] on n processors. This is in contrast to the statement given in [1] that the OCPC model is equivalent to an EREW PRAM with n global memory cells. Since the OCPC model uses full dynamic reconfiguration which is not yet technically feasible, we are interested in developing oblivious ERCW PRAM algorithms, which only require partial dynamic reconfiguration. This motivates the study of BFO circuits, which provide these oblivious ERCW PRAM algorithms.

The current interest in the OCPC model, the close relation between the OCPC model and the ERCW PRAM model, and the richness of results obtained so far on the OCPC, the ERCW PRAM, and the BFO circuit model, all indicate that these are important models of parallel computation which should be studied further.

In Section 2, we define the ERCW PRAM and discuss different write conflict protocols. Section 3 gives lower and upper bounds for compaction problems, and Section 4 gives lower and upper bounds for computing the maximum. In Section 5, we give algorithms for chaining and integer sorting. Section 6 gives lower and upper bounds for computing certain functions on unbounded fan-in, bounded fan-out circuits. In Section 7 we give relations between the different ERCW models, and in Section 8, we describe the relationship of the ERCW PRAM to the OCPC model.

2 Preliminaries

An Exclusive Read, Concurrent Write (ERCW) PRAM consists of a collection of processors, each with infinite local memory, which operate synchronously and communicate through a global memory. Each read or write to global memory takes one time step. Only one processor can read from any memory cell at any time step, but multiple processors may write to a memory cell in a single time step. Write conflicts are handled according to one of the following standard collision resolution protocols: Priority, Arbitrary, Common, Collision, Tolerant and Robust [31], or according to the Nice Robust protocol, in which the value of a cell after a concurrent write is either unchanged or equal to the value written by an arbitrary processor participating in the concurrent write. (Since the standard OCPC model uses the Tolerant protocol, we will be most concerned with developing ERCW PRAM algorithms using the Tolerant protocol. We define the OCPC model in Section 8.)

The ERCW(ack) PRAM is an ERCW PRAM with the added feature that a processor which successfully writes to a cell receives an acknowledgement. To retain the spirit of the Common model, we assume no processor receives an acknowledgement in the Common model. To retain the spirit of the Robust model, we assume that false "successful" writes could cause bogus acknowledgements to be sent.

Often we would like to separate the issues of using the global memory as storage for inputs and outputs, and using the global memory for communication. In these cases, we will assume that inputs and outputs are spread evenly among the local memories of the processors. For instance, given p processors and n inputs, we will assume each processor contains n/p inputs in its local memory. With this assumption, we will be free to design algorithms which use less than n cells of global memory.

In our algorithms we do not require that all processors learn the output of an algorithm, for this would force a trivial $\Omega(\log n)$ time lower bound on all our algorithms.

Lemma 2.1 *An n processor ERCW(ack) PRAM with m global memory cells can be simulated on a $\max\{n, m\}$ processor ERCW PRAM with $2m+n$ global memory cells with the same write conflict*

protocol (except Robust).

Proof: On the Common model, the ERCW(ack) PRAM and ERCW PRAM are the same, so the simulation is trivial. Otherwise, let E_1 be the ERCW(ack) PRAM and let E_2 be the ERCW PRAM. The first m cells of E_2 will correspond to the m cells of E_1, the second m cells of E_2 will be used for finding the processor that succeeds in writing to the corresponding cell of E_1, and the last n cells will be used for writing acknowledgements to the successfully writing processors. We simulate a read of cell j by processor i of E_1, by having processor i read cell j of E_2. We simulate a write step as follows. First, every processor j $(0 \leq j \leq m-1)$ writes n to cell $m+j$, and then every processor i $(0 \leq i \leq n-1)$ writes 0 to cell $2m+i$. For any processor i of E_1 that writes some value v_i to any cell c_i, processor i of E_2 writes i to cell $m+c_i$. Then processor j $(1 \leq j \leq m)$ reads cell $m+j$, and if the value read, say v, is not n or "collision", processor j writes 1 to cell $2m+v$. Now, for each processor i of E_1 writing to some cell c_i, processor i of E_2 reads cell $2m+i$. If it reads a 1, then it writes v_i to cell c_i (with no contention). Note that we require E_2 to have the same write conflict protocol as E_1, so that the processor that succeeds in writing to cell $m+j$ in E_2 (for some j) is the same as the processor that succeeds in writing to cell j in E_1. (Note: The simulation for the Collision model is slightly different. We omit the details.) □

The following lemma will be useful in designing Robust ERCW PRAM algorithms.

Lemma 2.2 *An n processor Nice Robust ERCW(ack) PRAM with m global memory cells can be simulated on a $\max\{n,m\}$ processor Robust ERCW PRAM with $m(n+2)$ global memory cells.*

Proof: Let E_1 be the ERCW(ack) PRAM and let E_2 be the ERCW PRAM. For each cell of E_1, we associate 1 extra cell to test for the processor which is successful, and n extra cells to be used for writing acknowledgements. We simulate a read of cell j by processor i of E_1, by having processor i read cell j of E_2. We simulate a write step as follows. First, every processor j $(0 \leq j \leq m-1)$ writes n to cell $m+j$. Then for any processor i of E_1 that writes some value v_i to any cell c_i, processor i of E_2 writes i to cell $m+c_i$, and writes 0 to cell $m(2+i)+c_i$. Then processor j $(1 \leq j \leq m)$ reads cell $m+j$, and if the value read, say v, is such that $0 \leq v \leq n-1$, processor j writes 1 to cell $m(2+v)+j$. Now, for each processor i of E_1 writing to some cell c_i, processor i of E_2 reads cell $m(2+i)+c_i$. If it reads a 1, then it writes v_i to cell c_i (with no contention). □

3 Compaction problems

In this section, we study the problems of k-compaction and approximate k-compaction on the ERCW PRAM. The *k-compaction* problem takes an array of size n with k marked elements, and places the marked elements into an array of size k. The *approximate k-compaction* problem takes an array of size n with k marked elements, and places the marked elements into an array of size $O(k)$. Compaction and approximate compaction are important subproblems in processor reallocation and load balancing.

3.1 Lower Bounds

Here we will show that 2-compaction on the Robust, Nice Robust, Tolerant, Collision, or Common ERCW PRAM requires $\Omega(\sqrt{\log\log n})$ time and that k-compaction on the Priority or Arbitrary ERCW PRAM requires time k for $k \leq \sqrt{(\log\log n)/2} - 1$. We do not place any restrictions on the number of global memory cells, or the number of processors. (We assume that each of the first n global memory cells contains an input.) We will assume that concurrent writes on the Robust

ERCW PRAM are resolved using the Tolerant protocol. We will assume that concurrent writes on the Arbitrary ERCW PRAM are resolved using the Priority protocol.

Wlog, we assume that each input is tagged by a pair (i, b), where i is its index (from 1 to n) and b is 0 if the input is unmarked, 1 if the input is marked. Then we will show a lower bound on solving compaction for the simpler problem of performing compaction on the tags which are marked with 1s. This will obviously imply a lower bound for the general compaction problem.

Let 2COMP be an algorithm for 2-compaction on the Tolerant, Collision, or Common ERCW PRAM. Let COMP be an algorithm for k-compaction on the Priority ERCW PRAM. We will use an adversary argument for our lower bound proof. A step will consist of a write followed by a read. At each step, the adversary will designate some of the inputs as unmarked (by setting $b = 0$) or marked (by setting $b = 1$). Let V_t be the set of indices of inputs which have not been designated by step t. These will be the live inputs. $V_0 = \{1, \ldots, n\}$. Let p_t be the number of processors which could be affected by a given live input. Let c_t be the number of cells which could be affected by a given live input. Let $k_t = \max\{c_t, p_t\}$.

Lemma 3.1 *We can construct an adversary such that after step t of 2COMP and COMP,* **(1)** $k_t \leq 4^t$; **(2)** $|V_t| \geq |V_{t-1}|^{1/k_{t-1}}/152k_{t-1}^2$; **(3)** *each processor and cell is affected by at most one live input; and* **(4)** *in COMP, at most t items have been designated as marked.*

Proof: We prove this by induction. First, $p_0 = 0 < 2^0$, $c_0 = 1 = 2^0$, each processor is affected by no inputs, and each cell is affected by at most one input.

Now assume the lemma is true up to step t. Then we show how to make it hold for step $t+1$. Let $k = k_t$. Let a writing function of a processor be defined as the cell to which it writes depending if the live input it knows is marked or unmarked. A processor zero-writes to a cell if a processor writes to the cell if its live input is unmarked. A processor one-writes to a cell if a processor writes to the cell if its live input is marked.

First we deal with zero-writes. We say the adversary *zeros* a live input if it designates it as unmarked. Note that once an adversary zeros a live input, the input is not live anymore. We will make the adversary zero some of the live inputs so that each cell is affected by at most one live input whose processor zero-writes to it in this step. To do this, we describe a simple procedure for the adversary. Until each cell has at most one live input whose processor zero-writes to it, the adversary arbitrarily chooses a remaining live input and for each cell which the chosen input's processors could write to, the adversary zeros the other live inputs whose processors zero-write to the same cell, up to two per cell. Notice that once we zero two live inputs whose processors zero-write to a cell, the cell is fixed, and no information about live inputs is written to it. Using this procedure, the adversary sets at most $2p_t$ live inputs for each one chosen. Thus we are left with $|V_t|/(2p_t + 1)$ live inputs. Let $m = |V_t|/(2k + 1)$.

Now we deal with one-writes. We need to find a set of live inputs of size $m^{1/k}/k$ such that if one of these inputs is one-written to a cell by a processor, then either each other live input is also one-written to the same cell by a processor, or none are one-written to the same cell. This is equivalent to finding a "sunflower" in a group of sets, where each set contains the cells one-written to by processors which know a given live input. By the Erdös-Rado Theorem [14], there must be a sunflower of size $(m/k!)^{1/k} \geq m^{1/k}/k$. Let $m' = m^{1/k}/k$. If we are on the Priority ERCW PRAM, we will designate the lowest numbered live input processor as marked. Note that the cells which are one-written to by a processor for each live input are now only affected by the lowest

numbered processor knowing a marked input, with the possible addition of at most one zero-writing processor. On any of the other ERCW PRAM models, note that the cells which are one-written to by a processor for each live input are fixed, because two of the live inputs must be marked. Also note that $m' \geq |V_t|^{1/k}/2k$.

Now at most $c' = c_t + 2p_t$ cells are affected by a live input after this write step. Also, since each cell can be read by at most one processor, after the read step at most $p' = c' + p_t$ processors will be affected by a live input. Each cell could be affected by at most three live inputs and each processor could be affected by at most four live inputs. The adversary must zero some inputs so that each cell and processor is only affected by one live input. To do this, we construct a graph in which the vertices are the live inputs and edges between vertices exist if the live inputs are known to the same cell or processor. There are at most $9p'm'$ edges in this graph, so by Turan's Theorem, we can find an independent set of vertices of size $(m')^2/(m' + 18p'm') \geq m'/(18p'+1) \geq |V_t|^{1/k}/152k^2$. The inputs corresponding to vertices not in the independent set are then zeroed by the adversary.

Now we can set $p_{t+1} = c_t + 3p_t \leq 4^{t+1}$, $c_{t+1} = c_t + 2p_t \leq 4^{t+1}$, V_{t+1} to be the remaining set of live inputs, where $|V_{t+1}| \geq |V_t|^{1/k}/152k^2$. □

Theorem 3.1 *Solving 2-compaction on a Robust, Common, Collision, or Tolerant ERCW PRAM requires $\Omega(\sqrt{\log\log n})$ steps, and for $k \leq \sqrt{(\log\log n)/2} - 1$, solving k-compaction on a Priority or Arbitrary ERCW PRAM requires at least k steps.*

Proof: From lemma 3.1, we can see that

$$|V_t| \geq \frac{|V_0|^{\prod_{0\leq i\leq t} 1/4^i}}{4^{2t}152^t} \geq \frac{n^{\prod_{0\leq i\leq t} 1/4^i}}{3000^t} \geq \frac{n^{4^{-\sum_{0\leq i\leq t} i}}}{3000^t} \geq \frac{n^{4^{-i(i+1)/2}}}{3000^t} \geq \frac{n^{2^{-(t+1)^2}}}{3000^t}$$

Thus after $T = \sqrt{(\log\log n)/2)} - 1$ steps, we will have $|V_T| \geq \Omega(2^{\sqrt{\log n}}/3000^{\sqrt{\log\log n}/2})$. For large n, $|V_T| \geq \max\{k+3, 5\}$. For the case of 2-compaction on the Robust, Nice Robust, Collision, Common, or Tolerant models, there will be 3 live inputs which do not affect either of the first two global memory cells. However, these cells contain the indices for only 2 live inputs. Therefore the adversary could designate as marked at least one of the 3 live inputs that do not affect either of the two cells, and the compaction would have failed. For the case of k-compaction on the Priority or Arbitrary models, after $k-1$ steps, at most $k-1$ entries have been designated as marked. There will be 3 live inputs which do not affect any of the first k cells. Therefore, $k-1$ of the cells will contain indices for the $k-1$ entries already designated as marked, and whatever index the other cell contains, the adversary could designate one of the other inputs as marked, and the compaction would have failed. □

Since this lower bound holds for any number of processors and global memory cells, it also holds for the ERCW(ack) PRAM with the same write conflict resolution protocols.

3.2 Upper Bounds

First we note that there is a simple algorithm which solves k-compaction in $O(k)$ time on an Arbitrary ERCW PRAM. However, this algorithm will not work unless some processor can succeed in each write. For the other write conflict resolution protocols we need a different approach.

We construct an algorithm which runs in $O(\log\log n + \log k)$ time on a Tolerant ERCW PRAM. This is an adaptation of an $O(\log k)$ time algorithm for k-compaction on the Robust CRCW PRAM

given in Fich *et al.* [17]. If $k \geq n^{1/5}$, we use a simple EREW prefix sums algorithm to perform the compaction in $O(\log k)$ time. Otherwise, as in [17] we partition the input cells into groups of l cells, where

$$l = \begin{cases} 2k(k-1) & \text{if } k \leq \frac{\log n}{4 \log\log n} \\ \frac{(k-1)\log n}{3\log\log n - 1} & \text{if } \frac{\log n}{4\log\log n} < k \leq \log n, \text{ and} \\ \frac{(k-1)\log n}{3\log k - 1} & \text{if } \log n < k < n^{1/5}. \end{cases}$$

We solve k-compaction within each group in $O(\log l) = O(\log k)$ time using the simple EREW algorithm.

Let $y_j = j$ if the jth group contains a non-zero entry, and let $y_j = 0$ otherwise. As in [17], if we solve the k-compaction problem for $y_1, \ldots, y_{n/l}$, we can solve the original k-compaction problem in $O(\log l) = O(\log k)$ more steps. To solve the k-compaction problem on $y_1, \ldots, y_{n/l}$, we proceed as in [17], and reduce the problem in $O(\log l)$ (i.e., $O(\log k)$) time on a Tolerant or Collision ERCW PRAM to k-compaction in an array of size $l2n^{(k-1)/l}$. If $k > \log n/4\log\log n$, then $l2n^{(k-1)/l} = k^{O(1)}$, and we can easily solve this k-compaction problem in $O(\log k)$ time. If $k \leq \log n/4\log\log n$, then in [17] the problem is solved in constant time using a CRCW technique, but on the ERCW PRAM we will, instead, solve the problem recursively on an array of size $l2n^{(k-1)/l} \leq l2n^{1/2k}$. For $k \leq \log n/4\log\log n$, $2l \leq n^{1/k}$, so we can bound the time of this recurrence by

$$T(n) \leq T(n^{3/(2k)}) + O(\log k) = O\left(\frac{\log\log n}{\log k}(\log k)\right) = O(\log\log n).$$

Hence, using the fact that the Tolerant protocol is a Nice Robust protocol, and using Lemma 2.2, we obtain the following theorem.

Theorem 3.2 *Let $t(n,k) = \log\log n + \log k$. The k-compaction problem can be solved in $O(t(n,k))$ time on an $n/t(n,k)$ processor Collision or Tolerant ERCW PRAM with $n/t(n,k)$ global memory cells, and on an $n/t(n,k)$ processor Robust ERCW PRAM with $O((n/t(n,k))^2)$ global memory cells. It can also be solved in $O(k)$ time on an n/k processor Arbitrary ERCW PRAM with 1 global memory cell.*

3.2.1 Randomized

We present two results for randomized algorithms for compaction. Both results are obtained by having processors hash into random locations in an array. We will assume the inputs are given in the local memories of the processors.

Our first result is an $O(\log k)$ expected time randomized algorithm for compaction on the Robust ERCW PRAM with n processors and n memory locations. If $k > n^{1/16}$, we use the simple $O(\log n) = O(\log k)$ time parallel prefix algorithm to perform the compaction. Otherwise, let A be an array of size k^4. Clear this array, and let each processor representing some marked element write its processor number to a random location of A. We use an $O(\log k)$ time prefix operation to check if k processors succeeded without collision. If so, we compact them into the first k locations in the array using simple parallel prefix, and inform the marked processors of their success. If any processor doesn't receive notice of success, (and so, by construction no processor receives notice of success) it simply retries the procedure. It is easily shown that the probability of failure decreases geometrically with the number of attempts. Then using Lemma 2.2, we obtain the following theorem. (We omit some details.)

Theorem 3.3 *An $n/\log k$ processor Robust ERCW PRAM with no more than $n/\log k$ global memory cells can solve k-compaction in $O(\log k)$ expected time.*

Now we describe an $O(\log\log k)$ algorithm for approximate compaction on an n processor Nice Robust ERCW(ack) PRAM which works with probability $1-\frac{1}{k}$ and uses only $O(k)$ global memory locations. Each processor with a marked element writes it to a random location in an array of size $8k$. If a processor receives an acknowledgement, it idles. If not, the processor writes its element into an array of size $4k$. This procedure continues for a total of $\log\log k$ steps as the array size reduces by half each time. Then we attempt for three steps to write the remaining elements into arrays of size k.

It is not difficult to see, using a Chernoff bound, that the number of remaining elements after step t is at most $\max\{k2^{-(2^t+t-1)}, k^{1/4}\}$ with probability $1-te^{-k^{1/4}/4}$. The probability of any element colliding in the last three steps is $k^{1/4}(1/k^{3/4})^3 \leq 1/k^2$. Since $(\log\log k)e^{-k^{1/4}/4} \leq 1/k^2$ for sufficiently large k, we can bound the total probability of not succeeding by $1/k$. Then using Lemma 2.1, we obtain the following theorem. (We omit some details.)

Theorem 3.4 *An $n/(\log\log k)$ processor Nice Robust ERCW PRAM with $n/(\log\log k)$ global memory cells can solve approximate k-compaction in time $O(\log\log k)$, with probability $1-1/k$.*

4 Maximum

Finding the maximum of n inputs requires $\Theta(\log n)$ time on an EREW or CREW, even when the inputs are restricted to be either 0 or 1 [11]. Finding the maximum of n inputs on a Priority CRCW with n processors requires $\Theta(\log\log n)$ time if the inputs come from a large range and $O(k)$ time if the inputs are restricted to the range $[1, n^k]$ [19]. In this section we will show that finding the maximum requires $\Omega(\sqrt{\log n})$ time on an ERCW PRAM. If input values are restricted to the range $[1, s]$, $s \leq n$ we will show that the maximum can be found in $O(\log\log s)$ time on the Common or Tolerant ERCW PRAM, and that $\Omega(\sqrt{\log\log s})$ time is required to find the maximum.

4.1 Lower Bounds

Our first lower bound will be for the case of unrestricted input domain. Wlog, we will assume that all the inputs are distinct. Let MAX be an algorithm on the Priority ERCW PRAM which finds the maximum of n inputs stored one per processor in the first n processors. For concreteness, assume that the output of MAX is stored in the first global memory cell. Consider step t of MAX. Let $V_t \subseteq \{1,\ldots,n\}$ be the set of indices of inputs which could still be the maximum. These will be called the live inputs. Let $S_t \subseteq \{1,2,3,\ldots\}$ be the possible values for the live variables, as restricted by the adversary. Let $F_t = \{f_i | i \in \{1,\ldots,n\} - V_t\}$ be the adversary's assignment of values to fixed variables. Let k_t be the number of processors which know any live input.

As the computation proceeds, the adversary fixes the values of certain inputs and maintains a set of allowed inputs, such that, after each step, each processor knows at most one live variable. Initially $V_0 = \{1,\ldots,n\}$, $S_0 \subseteq \{1,2,3,\ldots\}$ and is infinite, and $F_0 = \emptyset$. (S_0 will be defined explicitly later.)

Lemma 4.1 *We can construct an adversary such that after step t of MAX, the following properties hold:* **(1)** *$V_t \subseteq V_{t-1}$ and $|V_t| \geq \frac{|V_{t-1}|}{3^{t+1}}$;* **(2)** *each processor knows at most one input in V_t;* **(3)** *$k_t \leq 3^t$;* **(4)** *$S_t \subseteq S_{t-1}$ and S_t is infinite;* **(5)** *$F_{t-1} \subseteq F_t \subseteq \{1,2,3,\ldots\} - S_t$; and* **(6)** *an input in V_t is known by at most 3^t processors.*

Proof: Define a processor's read (write) function at step t to be a function which maps the input this processor knows about at step t to the location which it reads(writes). As in [18], we use Ramsey Theoretical arguments to restrict the possible inputs such that **(1)** for each step t, a processor either writes for all inputs or for no inputs; **(2)** for each step t, the processors read and write functions are either constant or one-to-one; and **(3)** all the read and write functions are either identical or disjoint. (Let S_0 is the set of possible inputs after this restriction. The fact that S_0 is still infinite is shown in [18].) After this restriction, we can see that, because we are assuming the inputs are distinct, one-to-one read and write functions will be useless. Thus we will only consider the constant read and write functions. Since reading and writing locations are now fixed at each step, a processor reading a location knows exactly which single live input it will learn about, if any.

Assume the lemma is true for all steps up to $t-1$. Consider step t of MAX. Each processor knows at most 1 live input. It writes to a specified location, and then reads from a specified location. These locations have already been fixed by the adversary. Since we are using the Priority model, only the lowest numbered processor writing to a given cell will succeed, and thus the only information obtained from that cell would be the information known by that single processor. Each processor can thus learn about at most one other processor when it reads a cell. Form a graph with the live inputs (V_{t-1}) as vertices and an edge between two vertices if a processor knows those two live inputs (one from previously, and one from the cell just read). Take the largest independent set in the graph and let V_t be the inputs associated with this independent set. Fix the smallest distinct values f_i from S_{t-1} to variables $i \in V_{t-1} - V_t$ and remove them from S_{t-1} to obtain S_t. Add these values to F_{t-1} to obtain F_t. By Turán's Theorem, we can choose a set V_t such that $|V_t| \geq \frac{|V_{t-1}|^2}{|V_{t-1}|+2e}$, where e is the number of edges in the graph. The number of edges is at most the number of processors which can know a given live input. Since the communication has been fixed, the number of processors which knows a given live input can at most triple at this time step. (If it has affected at most p processors and $c < p$ cells at step $t-1$, then it can affect $p + c < 2p$ cells and $2p + p < 3p$ processors at step t.) Then a live input is known by at most $3k_{t-1} = 3(3^{t-1}) = 3^t$ processors after step t. Thus $e \leq |V_{t-1}|3^t$, so $|V_t| \geq \frac{|V_{t-1}|^2}{|V_{t-1}|3^{t+1}} \geq \frac{|V_{t-1}|}{3^{t+1}}$. From this, it is easy to see that $|V_t| \geq n/3^{(t+1)+t+(t-1)+\cdots+2} \geq n/3^{(t+2)(t+1)/2}$. □

Theorem 4.1 *Finding the maximum of n inputs on a Priority ERCW PRAM requires $\Omega(\sqrt{\log n})$ communication steps.*

Proof: By Lemma 4.1, at step $T = \sqrt{2\log n/\log 3} - 2$, $V_T \geq 2$, and the first cell is affected by at most one of these inputs. Then the adversary can simply set the other input to be higher than the value stored in the first global memory cell. □

For the case of inputs restricted to a specific range, we can prove the following lower bound.

Theorem 4.2 *Finding the maximum of n inputs drawn from the range $[0, s]$, for $s < n$, requires $\Omega(\sqrt{\log\log s})$ time on a Robust, Nice Robust, Tolerant, Collision, or Common ERCW PRAM.*

Proof: Consider an input array of size n which consists of all zeros except for two entries at locations $i, j \in [1, s]$, which contain the values i and j, respectively. Solving 2-compaction in this array can easily be reduced to finding the maximum of the n inputs, and thus the $\Omega(\sqrt{\log\log s})$ lower bound on 2-compaction applies to the problem of finding the maximum. □

4.2 Upper Bounds

We first show a doubly logarithmic time algorithm for Rightmost One problem, and then show an algorithm for Maximum which is doubly logarithmic in the range, up to a range of size n.

Theorem 4.3 *The rightmost one of n bits can be found on an $n/\log\log n$ processor Common, Tolerant, or Collision ERCW PRAM in $O(\log\log n)$ time, or on an n processor Priority ERCW PRAM in constant time.*

Proof Sketch: The algorithm for the Priority model is trivial. For the other models, we divide the array into subarrays of size $\sqrt{n}$ and recursively find the rightmost subarray which contains a 1 and the rightmost one in each subarray. Note that on the CRCW PRAM, the recursion is unnecessary, since n processors can find the rightmost one in an array of size $\sqrt{n}$ in constant time. □

Theorem 4.4 *The maximum of n inputs in the range $[0, s]$ can be found on a $\max\{n, s\}/\log\log s$ processor Common, Tolerant, or Collision ERCW PRAM in $O(\log\log s)$ time.*

Proof Sketch: We create an array of size s, place 1's at positions in the array which correspond to input values, and find the rightmost one in $O(\log\log n)$ time. □

Using an algorithm similar to one in [16], we obtain the following result for finding the maximum of binary inputs (i.e., the global OR) on a Robust ERCW PRAM.

Theorem 4.5 *An $n/\log\log n$ processor Robust ERCW PRAM can find the global OR of n bits in $\Theta(\log\log n)$ time with error probability $\frac{1}{n}$.*

5 Chaining and Integer Sorting

Our goal is to obtain a fast ERCW PRAM algorithm to sort integers from a polynomial range. To do this, we first develop algorithms for Chaining, that is, given n bits as inputs, finding for each 1 input the position of the nearest one to its left.

Theorem 5.1 *The Chaining problem on n bits can be solved on an $n/\log\log n$ processor Common, Tolerant, or Collision ERCW PRAM in $O(\log\log n)$ time.*

Proof: First partition the input array into consecutive groups of $\log^2 n$ bits and solve the Nearest Ones problem in these groups using simple prefix operations in $O(\log\log n)$ time. Now we assign a 1 to each group which contained a 1, and solve the Chaining problem on $n/\log^2 n$ bits. Once this is done, the processor associated with the leftmost 1 bit in each group can simply read the position of the rightmost 1 bit in the nearest group to the left which contains a 1, and write it to the output array.

To solve the Chaining problem on $n/\log^2 n$ bits, notice that in $O(\log\log n)$ steps, we can broadcast each bit to $\log n$ processors, so we have $\log n$ processors working for each bit. Imagine a complete binary tree formed over the $n/\log^2 n$ bits. For each bit, associate one of its associated processors with each of its ancestors. Now for each node in the tree, solve the Rightmost One and Leftmost One problems. Each processor will then know if its bit is the rightmost or leftmost at that node. For each bit, use a simple prefix operation over the processors associated with that bit

to find the lowest node (closest to the leaves) for which the bit is not the leftmost bit. Then that processor can look at the left child of that node to find the rightmost bit. This is the nearest one to the left. There will be no read conflict, because each node can have at most one child with a leftmost bit which becomes not the leftmost. □

The following theorem addresses the Chaining problem in the case when there is a processor associated with each non-zero element in the input.

Theorem 5.2 *Let $A[1..n]$ be an array of zeros and ones, with a processor associated with each $A[i] = 1$ (hence the number of processors is equal to the number of ones in the input). Let the priorities of the processors decrease with the position within A of the element to which a processor is associated. The Chaining problem on this input can be solved in $O(\log\log n)$ time on a PRIORITY ERCW(ack) PRAM.*

Proof: The following algorithm solves the problem within the stated bounds. We create an auxiliary array of size $\sqrt{n}$ and divide the input array A into $\sqrt{n}$ blocks of size $\sqrt{n}$. All processors assigned to elements in the ith block perform a concurrent write of their element's position within A into location i of the auxiliary array. The processors that succeed delete their entry in A and recursively solve the problem in the auxiliary array. The remaining processors recursively solve the problem within their blocks. The recursive solutions are then combined into a solution for the original problem in constant time. Since all of the recursive subproblems are of size $\sqrt{n}$, the overall algorithm runs in $O(\log\log n)$ time. □

We can now perform a stable sort of n integers in the range $[0..n-1]$ in $O(\log n)$ time with $n\log\log n/\log n$ processors and $O(n^2)$ space on a PRIORITY ERCW(ack) PRAM as follows. As in the CRCW algorithm of [29] we use an $n \times n$ array (which is assumed to be initialized to zero). For each index i in the input, if element i has value j then a 1 is written into position (i,j) of the array. We then solve the chaining problem on the $n \times n$ array (interpreted as a $1 \times n^2$ array) to obtain the sorted elements in a linked list. This portion of the algorithm runs in $O(\log\log n)$ time using n processors using the algorithm in the proof of Theorem 5.2. To obtain the sorted list in an array form, we perform list ranking to find the position of each element in the output array. Since the sort is stable this allows us to sort n integers in the range $[0..n^k-1]$, for any constant k, within the same processor-time bounds. It also allows us to reduce the space requirement to $n^{1+\epsilon}$, for any constant $\epsilon > 0$, by viewing each value as the sum of powers of n^ϵ. This gives us the following theorem.

Theorem 5.3 *Integer chain-sorting can be performed on n integers in the range $[0..n-1]$ in $O(\log\log n)$ time with n processors on a Priority ERCW(ack) PRAM. Integer sort into an array can be performed on n integers in the range $[0..n^k]$ in $O(\log n)$ time with $n\log\log n/\log n$ processors and $n^{1+\epsilon}$ space on a Priority ERCW(ack) PRAM.*

6 Unbounded Fan-in, Bounded Fan-out Circuits

Since fully dynamic reconfiguration between a large number of processors in optical networks does not yet seem to be technically feasible, we would like to find ways of reducing the need for it. One way is to design oblivious algorithms. In oblivious algorithms the pattern of transmissions is known prior to the start of the algorithm (i.e., it is not dependent on the inputs). Therefore, we would

be able to fix or preset the transmission elements, and we may avoid some of the reconfiguration costs.

A special type of oblivious algorithm is given by a BFO circuit. We assume standard definitions for circuits and formulas [5]. A BFO circuit with size s and depth d can be simulated in a straightforward way by an s processor, d step oblivious OCPC algorithm. Just as unbounded fan-in, unbounded fan-out circuits correspond closely to the CRCW PRAM [6], and the study of bounded fan-in circuits often sheds light on problems on the CREW and EREW PRAM, we believe that the study of BFO circuits should enhance the understanding of the ERCW PRAMs

We now give some results on solving some fundamental problems on BFO circuits.

6.1 Lower Bounds

Our first result shows how to transform a BFO circuit into something resembling a formula, so that we can obtain a lower bound the depth of the circuit using known lower bounds on formula size.

Theorem 6.1 *Let f be a Boolean function over n variables. If a circuit of depth d with fan-out at most c (with one input corresponding to each variable) computes f, then there is a Boolean formula of size at most nc^d which computes f.*

Proof: Let C be a depth d circuit in which each gate has fan-out at most c. Let C' be the same circuit, but with every gate with some fan-out $c' > 1$ replaced by a gate with a single output leading into a "fan-out" gate which fans out the output to c' other gates. Then C' has depth at most $2d$. Now consider the following percolate operation. Assume a gate g has an output which enters a c'-fan-out gate. The percolate operation replaces this with a c'-fan-out gate at each input which fans out each input into c' duplicates of gate g. This has the effect of percolating the gate g up in the circuit.

We perform percolate operations on C' until all standard gates are above all fan-out gates. Notice that we have not changed the result nor the depth of the circuit. Call this new circuit C''. Notice that the standard gates of C'' all have output 1, and thus correspond to a formula for f. Let F correspond to this formula, i.e., the circuit consisting of the standard gates of C'', with the inputs corresponding to every input into a gate which is an actual input or an output from one of the fan-out gates. Since there are at most d levels of fan-out gates, and each of those gates has fan-out c, each input can be fanned out to at most c^d inputs of F. Thus there are at most nc^d inputs to F. □

Corollary 6.1 *Any BFO circuit which computes parity requires $\Omega(\log n)$ depth.*

Proof: By Khrapchenko [35], any formula for parity must have size $\Omega(n^2)$. By the previous lemma, $nc^d = \Omega(n^2)$, and since c is a constant, $d = \Omega(\log n)$. □

Let $TH_{k,n}$ denote the threshold function which outputs 1 if and only if at least k of the inputs are equal to 1.

Corollary 6.2 *Any BFO circuit which computes $TH_{k,n}$ requires $\Omega(\log k + \log\log n)$ depth.*

Proof: By Khrapchenko [35] any formula for $TH_{k,n}$ must have size $\Omega(k(n-k+1))$. By Krichevskii [36] any formula for $TH_{k,n}$ must have size $\Omega(n \log n)$. By the previous lemma, $nc^d = \max\{\Omega(k(n-k+1)), \Omega(n\log n)\}$, and since c is a constant, $d = \Omega(\log k + \log\log n)$. □

We next consider the computation of multiple-valued Boolean functions.

Lemma 6.1 *Let $f : R^n \to R^m$ be a Boolean function. Consider the jth input variable for some $j, 1 \leq j \leq n$. Let O be a set of output variables with the property that for each $o \in O$ there is some n-bit input I such that the value of o is complemented when the jth bit in I is complemented. Then any bounded fan-out circuit that computes f will require depth $\Omega(\log |O|)$.*

Proof: The circuit must contain a path from the jth input node to each of the output nodes in O. Since the circuit has bounded fan-out, the lemma follows. □

Corollary 6.3 *Any bounded fan-out circuit for adding two n-bit integers, merging a bit into an n bit sorted sequence, sorting n bits, or computing the prefix sums of n bits requires $\Omega(\log n)$ depth.*

6.2 Upper bounds

There are well known bounded fan-in circuits with $O(\log n)$ depth and linear size for parity, addition, merging, sorting binary inputs, and prefix sums on binary inputs. By [33], these circuits can be converted into bounded fan-out circuits of the same size and depth. By Corollaries 6.1 and 6.3, these are optimal.

Next we present a BFO circuit which computes the threshold function $TH_{k,n}$ in optimal size n, and optimal depth $O(\log k + \log\log n)$. Our construction makes use of an optimal logarithmic depth circuit for computing (in binary) the sum of n bits [39] and two constructions for monotone formulas due to Valiant [42] and Friedman [20] which we sketch below.

The monotone formula construction of Valiant [42] shows that any monotone symmetric function on n variables can be written as a monotone formula of size $O(n^{5.3})$. Implicit in the construction of this formula, is a monotone BFO circuit of size $O(n^{5.3})$ and depth $O(\log n)$.

The monotone formula construction of Friedman [20] shows that $TH_{k,n}$ can be written as a monotone formula of size $O(k^{12.6} n \log n)$. This construction uses Valiant's construction on threshold functions with $4k^2$ inputs, and thus has depth $O(\log k)$. The threshold function developed by Friedman has the form

$$TH_k(y_1, \ldots, y_n) = \bigvee_{j=1}^{k^4 \log n} TH_k\left(\bigvee_{i \in A_1^j} y_i, \bigvee_{i \in A_2^j} y_i, \ldots, \bigvee_{i \in A_{4k^2}^j} y_i\right),$$

where for each j, $A_1^j, \ldots, A_{4k^2}^j$ is a partition of the n inputs. Thus each of the n inputs must be fanned out to $k^4 \log n$ of Valiant's threshold circuits. This can be done in $O(\log k + \log\log n)$ depth and $O(nk^4 \log n)$ size. The total size of all of the Valiant circuits are then $k^4 \log n$ times $O((4k^2)^{5.3})$.

We use these results for our circuit as follows. First we place the n inputs into groups of size $k^{15} \log n$ and use the addition circuits to find the sum of the number of ones. This takes linear size in each group, and thus linear size overall. The depth required is $O(\log k + \log\log n)$. Then using a simple comparison circuit, each group can check to see if it has more than k 1's. The output to this circuit goes into a final OR gate which determines the final outcome. Along with this, each circuit fans out each of its first $\lceil \log k \rceil$ outputs into the appropriate number of ones, so that we will have at most $2k$ outputs from each group, with the number of ones output equal to the number of ones in the group (assuming the number of ones is at most k). The outputs of all the groups can now be input into the Friedman circuit. Since there are $2kn/(k^{15} \log n)$ inputs, the size of the

Friedman circuit will be $O(n)$. The depth will be $O(\log k + \log\log n)$. The output to the Friedman circuit is then ORed with the output of the comparator circuit at each of the groups. If any group had more than k inputs, then the output of the total circuit will be 1. If not, then the output from each group will be the correct number of ones in the group, and the output of the total circuit will be the output of Friedman's circuit, which will be 1, if and only if the number of ones in the input is at least k.

This circuit implies the following theorem

Theorem 6.2 *There is a size $O(n)$, depth $O(\log k + \log\log n)$ BFO circuit which computes $TH_{k,n}$.*

7 Relations between ERCW Models

We now discuss the relative powers of the different write conflict resolution protocols on the ERCW(ack) PRAM. Many of our results parallel those on the CRCW PRAM. Using the results from Section 8, some of these results can be generalized to the ERCW PRAM model and the OCPC model.

We use the notation Protocol(m) to denote a collision protocol on an ERCW(ack) PRAM with m global memory cells. If we let $X \leq Y$ mean "X conflict resolution protocol can be simulated on Y conflict resolution protocol with constant slowdown", then it is not hard to see that

$$\text{Robust}(m) \leq \text{Collision}(m) \leq \text{Arbitrary}(m) \leq \text{Priority}(m) \text{ , and}$$
$$\text{Robust}(m) \leq \text{Nice Robust}(m) \leq \text{Tolerant}(m) \leq \text{Collision}(2m).$$

(The last simulation simply associates an extra memory cell with each memory cell of the Tolerant ERCW(ack) PRAM, to test whether there will be a collision at that cell, so that the value of the cell is not overwritten if there is a collision.) In addition, $\text{Common}(m) \leq \text{Arbitrary}(m)$.

In the next two subsections, we describe some less obvious simulation results.

7.1 Separations

In our lower bounds, we will always assume the simulating machine has infinite memory. Boppana [4] showed that solving Element Distinctness on the Common CRCW PRAM (and thus the Common ERCW(ack) PRAM) requires $\Omega(\log n/\log\log n)$ time. However on any of the other ERCW(ack) models, Element Distinctness can be solved in constant time. This provides a separation of $\Omega(\log n/\log\log n)$ between the Common and any other model. This separation is tight for the CRCW PRAM, but so far the best algorithm for Element Distinctness on the Common ERCW(ack) PRAM requires $\Omega(\log n)$ time.

Grolmusz and Ragde [27] provide separations between some other CRCW PRAM models, which can be easily transferred to the ERCW(ack) PRAM. From these we get separations of $\Omega(\log\log\log n)$ time between the Collision and Arbitrary models, between the Collision and Common models, and between the Tolerant and Collision models. Chaudhuri improves the first and third separations to $\Omega(\log\log n)$, and these results also translate to the ERCW(ack) PRAM, giving separations of $\Omega(\log\log n)$ time between the Collision and Arbitrary models and between the Tolerant and Collision models.

7.2 Simulations

The simulation of Priority(m) CRCW PRAM on Arbitrary(mn) CRCW PRAM given by Chlebus *et al.* [9] can be followed almost exactly to give a simulation of a Priority(m) ERCW(ack) PRAM

on an Arbitrary(mn) ERCW(ack) PRAM. This simulation runs in $O(\log\log n)$ steps.

To simulate an Arbitrary(m) ERCW(ack) PRAM on a Tolerant(mn) ERCW(ack) PRAM, we use a partition algorithm from [9], which is run with a subset of processors, and results in either one processor being marked, or at least one but at most half of the processors being marked. This uses $O(n)$ memory cells and takes $O(\log\log n)$ time. An additional feature of this algorithm is that each marked processor has an associated unmarked processor, and thus if k processors are initially assigned to each processor in the subset, then the algorithm can assign $2k$ processors to each marked processor, k from the marked processor, and k from its associated unmarked processor. (This partition algorithm was written for the Collision CRCW PRAM, but can also be run on the Tolerant ERCW(ack) PRAM.)

For our simulation, we will divide the processors into groups according to the cells they write to, and run the partition algorithm $\sqrt{\log n/\log\log n}$ times (each subsequent application on the marked elements from the previous application). Then if there is more than a single processor remaining in a group, each of the remaining processors will be assigned $2^{\sqrt{\log n/\log\log n}}$ processors.

Now we can use a technique from [10] to choose one of the remaining (marked) processors for each cell as follows. Let $k = 2^{\sqrt{\log n/\log\log n}}$. Assign each marked processor to a cell in an n element array according to its processor number. Form a k-ary tree T over this array. Assign the k processors of a marked processor P to the k leaves in T with the same parent as P. Now let P write a 0 to its own location. Then let P write a 1 to its own location, and let the auxiliary processors at positions greater than P also write 1 to their locations. Then let P read its location. P will only read a 1 if it is the first (lowest numbered) child of its parent which is writing. Say a marked processor "wins" this level if it succeeds in writing a 1. Assume P wins this level. Then P and its associated processors move up to the next level. Notice that on the ERCW(ack) PRAM, the processors that lose can't inform their associated processors that they've lost, so these associated processors will also move up to the next level. But they will simply mimic the associated processors of the winner, and this will still allow only the first child of each parent to succeed in writing a 1. We continue this procedure through the $\log n/\log k$ levels, until exactly one winning processor remains. This processor then writes without contention to the appropriate cell, completing the simulation of the Arbitrary Write step.

The time for performing $\sqrt{\log n/\log\log n}$ partitions is $O(\sqrt{\log n\log\log n})$ and the time to proceed through $\log n/\log k$ levels of the tree, each one taking constant time, is also $O(\sqrt{\log n\log\log n})$. Thus the time for the simulation step is $O(\sqrt{\log n\log\log n})$.

8 Optical Communication and ERCW PRAMS

Here we describe the technology for optical communication, the OCPC model which is derived from this technology, and its relation to the ERCW PRAM.

8.1 Optical Communication Technology

There are two basic types of optical interconnection networks, fiber optic networks, and free-space optic networks. The type of fiber optic network which allows unit time communication between any pairs of processors is the *Passive optical star coupler* network [13, 28]. In this network all processors are connected via optical fibers to a passive optical star coupler, which broadcasts messages sent from one processor to all other processors. To allow more flexible communication, time division multiplexing (TDM) or wavelength division multiplexing (WDM) is used. For unit

time communication, we must use (WDM). For dynamic reconfiguration ability, we must have tunable transmitters and/or receivers. Currently, tunable transmitters and receivers are too slow to be practical.

The other type of optical interconnection network is a free-space optic network. In this network, we do not use wires, but instead use the directional property of light to send messages to the correct destination. Free-space optics has the potential to reduce space requirements and to alleviate many topological difficulties associated with more conventional routing. There are two types of free-space optic networks which achieve unit time communication, the beam spreading/masking network, and the beam steering network. The beam spreading/masking network [32, 41] (also called the crossbar or matrix multiplication network) uses an $n \times n$ array of switching elements with each row assigned to a transmitting processor and each column assigned to a receiving processor. A transmitting processor spreads its light encoded message out to the row of optical switches, and those switches either block the message or send it on to the designated receiving processor. This network has the disadvantages of having n^2 switches, $1/n$ total power transfer, and slow switching time. A beam steering network can either be made up of reconfigurable holograms [26, 2] or acousto-optic deflectors [32]. The typical reconfigurable hologram method assumes the processors are sitting on a board, and there is some holographic material above the board. To transmit a message, a reflecting hologram is written into the holographic substrate and the processor transmits a light encoded message to that hologram, which then steers it to the correct receiving processor. In the acousto-optic deflector method, it is assumed that each processor is connected to a two-dimensional deflector which can be programmed to steer a light encoded message to any other processor. In terms of speed of reconfiguration, the acousto-optic deflector seems to be the fastest, although it is still not as fast as an electronic switch.

8.2 OCPC model

One abstraction of the beam steering model (which could also be considered an abstraction of the passive optical star coupler with tunable transmitters) was first considered by Anderson and Miller [1], and has since been studied in [12, 15, 21, 22, 24, 25, 37, 43]. Various names for this model have been proposed, including *Local Memory PRAM*, S*PRAM, OMC, OCP, and OCPC. We will use the term OCPC, denoting *Optical Communication Parallel Computer*.

An OCPC consists of a collection of processors, each with infinite local memory, which operate synchronously and communicate by transmitting messages to each other. At any step, a processor can transmit at most one message to another processor. The message will succeed in reaching the processor if it is the only message being sent to that processor at that step. Concurrent transmissions to the same processor will be handled according to one of the following standard collision resolution protocols: Priority, Arbitrary, Common, Collision, Tolerant and Robust [31]. (Note that the standard OCPC model uses the Tolerant protocol.)

The OCPC(ack) is an OCPC with the added feature that a processor which successfully transmits a message to another processor receives an acknowledgement as in the ERCW PRAM (see Section 2).

The following are some relationships between OCPC and ERCW PRAM models, with and without acknowledgements.

Lemma 8.1 *An n processor OCPC can be simulated on an n processor ERCW PRAM with n global memory cells with the same write conflict protocol.*

Proof: One step of the OCPC is simulated by a write and a read on the ERCW PRAM. We simulate an attempted transmission from processor i to processor j on the OCPC by processor i writing to cell j and processor j reading cell j. Since the write conflict resolution protocols are the same, processor j receives the same value on the ERCW PRAM as on the OCPC. □ Note that in the next proof, the simulation of an ERCW PRAM on an OCPC requires that the OCPC has at least as many processors as the ERCW PRAM has global memory cells. This is not necessarily bad, since only *global* memory cells are counted, and for many ERCW PRAM algorithms, only $O(n)$ global memory cells are required.

Lemma 8.2 *An n processor ERCW PRAM with m global memory cells can be simulated on a* $\max\{n, m\}$ *processor OCPC with the same write conflict protocol.*

Proof: Assign each processor of the OCPC to a memory cell and assign the first n processors also to the n processors of the ERCW PRAM. We simulate a write from processor i to cell j on the ERCW PRAM by processor i transmitting to processor j. Because the write conflict resolution protocols are the same, processor j receives the same value on the OCPC as is written to cell j on the ERCW PRAM. Processor j stores the value transmitted to it. We simulate processor i reading cell j in the ERCW PRAM by processor i transmitting a read request to processor j and processor j transmitting the value stored there to processor i. Note that there can never be any transmission conflicts when simulating the read step. □

Lemma 8.3 *An n processor OCPC(ack) can be simulated on an n processor OCPC with the same write conflict protocol (except Robust).*

Proof: On the Common model, the OCPC and the OCPC(ack) models are equivalent, since no processor ever succeeds in a write. On the other models we simulate a transmission of a message M from processor i to processor j by the following procedure. Processor i first transmits i to processor j. If processor j receives a value v, then it transmits an acknowledgement to processor v. If processor i receives an acknowledgement, then it transmits M to processor j. This will be guaranteed to succeed without collision. □

Lemma 8.4 *An n processor ERCW(ack) PRAM with m global memory cells can be simulated on an on a* $\max\{n, m\}$ *processor OCPC with the same write conflict protocol (except Robust).*

Proof: By Lemma 8.3, it suffices to prove the theorem for the OCPC(ack) model. Assign each processor of the OCPC to a memory cell and assign the first n processors also to the n processors of the ERCW(ack) PRAM. A write step of the ERCW(ack) PRAM is simulated by a transmission step of the OCPC(ack). We simulate a write from processor i to cell j on the ERCW(ack) PRAM by processor i attempting a transmission to processor j. If processor i is successful in writing to the cell, then it will receive an acknowledgement. Because the OCPC(ack) is using the same write conflict resolution protocol, processor i's transmission will succeed and processor i will receive an acknowledgement. Processor j now stores the value transmitted to it. A read step of the ERCW(ack) PRAM is simulated by two transmission steps of the OCPC(ack). We simulate processor i reading cell j in the ERCW(ack) PRAM by processor i transmitting a read request to processor j and processor j transmitting the value stored there to processor i. Note that there can never be any transmission conflicts when simulating the read step. □

References

[1] R. J. Anderson and G. L. Miller. Optical communication for pointer based algorithms. Technical Report CRI 88-14, University of Southern California, 1988.

[2] P. B. Berra, A. Ghafoor, M. Guiznani, S. J. Marcinkowski, and P. A. Mitkas. Optics and supercomputing. *Proceedings of the IEEE*, 77(12):1797–1815, 1989.

[3] P. C. P. Bhatt, K. Diks, T. Hagerup, V. Prasad, T. Radzik, and S. Saxena. Improved deterministic parallel integer sorting. *Inform. and Comput.*, 94(1):29–47, September 1991.

[4] R. B. Boppana. Optimal separations between concurrent-write parallel machines. In *Proc. 21st Symp. on Theory of Computing*, pages 320–326, 1989.

[5] R. B. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, chapter 14, pages 757–804. MIT Press/Elsevier, 1990.

[6] A. K. Chandra, L. J. Stockmeyer, and U. Vishkin. A complexity theory for unbounded fan-in parallelism. In *Proc. 23th Symp. on Found. of Comp. Sci.*, pages 1–13, 1982.

[7] B. S. Chlebus. A parallel bucket sort. *Inform. Process. Lett.*, 27(2):57–61, February 1988.

[8] B. S. Chlebus. Parallel iterated bucket sort. *Inform. Process. Lett.*, 31(4):181–183, May 1989.

[9] B. S. Chlebus, K. Diks, T. Hagerup, and T. Radzik. Efficient simulations between CRCW PRAMs. In *Proc. 13th Symp. on Math. Found. of Comp. Sci.*, volume 324, pages 231–239. Springer Lecture Notes in Computer Science, 1988.

[10] B. S. Chlebus, K. Diks, T. Hagerup, and T. Radzik. New simulations between CRCW PRAMs. In *Proc. 7th Intl. Conf. on Fundamentals of Comp. Theory*, volume 380, pages 95–104. Springer Lecture Notes in Computer Science, 1989.

[11] S. Cook, C. Dwork, and R. Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.*, 15(1):87–97, February 1986.

[12] M. Dietzfelbinger and F. Meyer auf der Heide. Simple, efficient shared memory simulations. In *Proc. ACM Symp. on Para. Alg. and Arch.*, pages 110–119, 1993.

[13] P. Dowd. High performance interprocessor communication through optical wavelength division multiple access channels. In *Proc. 18th Symp. on Comp. Arch.*, pages 96–105, 1991.

[14] P. Erdös and R. Rado. Intersection Theorems for Systems of Sets. *J. London Math. Soc.*, 35:85–90, 1960.

[15] M. M. Eshaghian. Parallel algorithms for image processing on omc. *IEEE Trans. Comput.*, 40(7):827–833, 1991.

[16] F. Fich, R. Impagliazzo, B. Kapron, V. King, and M. Kutylowski. Limits on the power of parallel random access machines with weak forms of write conflict resolution. In *Proc. of 10th Symp. on Theor. Aspects of Comp. Sci.*, page unknown, 1993.

[17] F. Fich, M. Kowaluk, M. Kutylowski, K. Loryś, and P. Ragde. Retrieval of scattered information by EREW, CREW, and CRCW PRAMs. In *Proc. 3rd Scand. Workshop on Alg. Theory*, pages 30–41. Lec. Notes in Comp. Sci., Vol. 621, 1992.

[18] F. E. Fich, F. Meyer auf der Heide, P. Ragde, and A. Wigderson. One, two, three ...infinity: Lower bounds for parallel computation. In *Proc. 17th Symp. on Theory of Computing*, pages 48–58, 1985.

[19] F. E. Fich, P. Ragde, and A. Wigderson. Relations between concurrent-write models of parallel computation. *SIAM J. Comput.*, 17:606–627, 1988.

[20] J. Friedman. Construct $O(n \log n)$ size montone formulae for the kth threshold function of n boolean variables. *SIAM J. Comput.*, 15(3):641–654, 1986.

[21] A. V. Gerbessiotis and L. G. Valiant. Direct bulk-synchronous parallel algorithms. In *Proc. Scandinavian Workshop on Algo. Theory*, 1992.

[22] M. Geréb-Graus and T. Tsantilas. Efficient optical communication in parallel computers. In *Proc. ACM Symp. on Para. Alg. and Arch.*, pages 41–48, 1992.

[23] P. B. Gibbons, Y. Matias, V. Ramachandran. The Queue-Read Queue-Write PRAM model: Accounting for contention in parallel algorithms. In *Proc. ACM-SIAM Symp. on Discrete Algs.* 1994, *SIAM J Comput*, to appear.

[24] L. A. Goldberg, M. Jerrum, T. Leighton, and S. Rao. A doubly logarithmic communication algorithm for the completely connected optical communication parallel computer. In *Proc. ACM Symp. on Para. Alg. and Arch.*, pages 300–309, 1993.

[25] L. A. Goldberg, M. Jerrum, and P. D. MacKenzie. A lower bound for routing on a completely connected optical communication parallel computer. accepted to SPAA, 1994.

[26] J. W. Goodman, F. Leonberger, S. Y. Kung, and R. A. Athale. Optical interconnections for vlsi systems. *Proceedings of the IEEE*, 72(7):850–866, 1984.

[27] V. Grolmusz and P. Ragde. Incomparability in parallel computation. In *Proc. 28th Symp. on Found. of Comp. Sci.*, pages 89–98, 1987.

[28] I. M. I. Habbab, M. Kavehrad, and C. E. W. Sundberg. Protocols for very high-speed optical fiber local area networks using a passive star topology. *J. Lightwave Tech.*, LT-5:1782–1793, 1987.

[29] T. Hagerup. Towards optimal parallel bucket sorting. *Inform. and Comp.*, 75:39–51, 1987.

[30] T. Hagerup. Constant-time parallel integer sorting. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 299–306, 1991.

[31] T. Hagerup and T. Radzik. Every robust CRCW PRAM can efficiently simulate a Priority PRAM. In *Proc. 2nd ACM Symp. on Para. Alg. and Arch.*, pages 117–124, 1990.

[32] A. Hartmann and S. Redfield. Design sketches for optical crossbar switches intended for large-scale parallel processing applications. *Optical Eng.*, 28(4):315–327, 1989.

[33] H. J. Hoover, M. M. Klawe, and N. J. Pippenger. Bounding fan-out in logical networks. *J. Assoc. Comput. Mach.*, 31(1):13–18, 1984.

[34] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, chapter 17, pages 869–941. MIT Press/Elsevier, 1990.

[35] V. M. Khrapchenko. A method for determining lower bounds for the complexity of Π-schemes. *Mat. Zametki*, 10(1):83–92, 1971. (in Russian); English translation in: *Math. Notes* **10**(1) (1971) 474–479.

[36] R. E. Krichevskii. Complexity of contact circuits realizing a function of logical algebra. *Dokl. Akad. Nauk SSSR*, 151(4):803–806, 1963. (in Russian); English translation in: *Soviet Phys. Dokl.* **8**(8) (1964) 770–772.

[37] P. D. MacKenzie, C. G. Plaxton, and R. Rajaraman. On contention resolution protocols and associated probabilistic phenomena. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 153–162, 1994.

[38] P. D. MacKenzie and Q. F. Stout. Ultra-fast expected time parallel algorithms. In *2nd ACM-SIAM Symp. on Disc. Alg.*, pages 414–423, 1991. submitted to Journal of Algorithms.

[39] D. E. Muller and F. P. Preparata. Bounds to complexities of networks for sorting and for switching. *J. Assoc. Comput. Mach.*, 22(2):195–201, April 1975.

[40] S. Rajasekaran and J. H. Reif. Optimal and sublogarithmic time randomized parallel sorting algorithms. *SIAM J. Comput.*, 18(3):594–607, June 1989.

[41] A. A. Sawchuk, B. K. Jenkins, and C. S. Raghavendra. Optical crossbar networks. *IEEE Computer*, 20(6):50–60, 1987.

[42] L. G. Valiant. Short monotone formulae for the majority function. *J. Algorithms*, 5:363–366, 1984.

[43] L. G. Valiant. General purpose parallel architectures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, chapter 18, pages 945–971. MIT Press/Elsevier, 1990.