INFORMATION AND COMPUTATION 76, 13–28 (1988)

Limits on the Power of Concurrent-Write Parallel Machines*

PAUL BEAME†

Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, Massachusetts 02139

The computation of specific functions using the most general form of concurrent-read–concurrent-write parallel RAM is considered. It is shown that such a machine can compute any function of Boolean inputs in time $\log n - \log\log n + O(1)$ given a polynomial number of processors and memory cells and that this bound is tight for integer addition. Despite this evidence of the power of the model we show that a very simple function, namely parity, requires time $\Omega(\sqrt{\log n})$ to compute given a polynomial bound on the number of processors, independent of the number of memory cells.

1. INTRODUCTION

Parallel random access machines (PRAMs) are well-accepted as good models for parallel computation. The procedural nature of the way in which they compute and the relatively natural way of enforcing uniformity conditions on them have made them popular for the design of parallel algorithms. They consist of many processors acting in consort and communicating through some shared memory. They operate much like familiar sequential RAMs except for the rules for concurrently accessing the shared memory. There are three main classifications of these rules: exclusive read–exclusive write (EREW), concurrent read–exclusive write (CREW), and concurrent read–concurrent write (CRCW). The most powerful of these machines (CRCW) can be simulated by the weakest (EREW) with a delay per step proportional to the logarithm of the number of processors.

Cook, Dwork, and Reischuk (1984) have shown that either the EREW or the CREW PRAMs require $\Theta(\log n)$ time to compute the OR of n bits. Their lower bound holds independently of not only the number of processors or the size of the shared memory but also the way the program is specified and the instruction set of a processor. It really says something about the restrictive nature of the communication itself.

* This work was done while the author was at the University of Toronto.

† Current address: Department of Computer Science, FR-35, University of Washington, Seattle, Washington 98195.

0890-5401/88 $3.00

The OR of n bits can easily be computed in constant time on a CRCW PRAM and it is easy to see how with sufficiently many processors and cells any Boolean function can be computed in constant time. It is natural then to consider CRCW PRAMs which have resources bounded by a polynomial in the size of the input and try to prove lower bounds similar to those for PRAMs which have only exclusive writes. Previous lower bounds for CRCW PRAMs have approached this in different ways. Some have put extremely stringent restriction on the number of processors or the size of the shared memory, e.g., Vishkin and Wigderson (1985), or Fich, Meyer auf der Heide, Ragde, and Wigderson (1985). Other bounds which hold with a polynomial number of processors or cells but which put severe restrictions on the instruction sets of the PRAMs are due to Stockmeyer and Vishkin (1984) and Meyer auf der Heide and Reischuk (1984). Other more powerful primitives than those they allow seem to be perfectly reasonable since the cost of concurrent accesses is presumably significantly greater than that of local computations. A symptom of the restricted nature of these CRCW PRAMs is that most Boolean functions require time $\Omega(2^{n/2})$ to compute given a polynomial number of processors and cells whereas all Boolean functions can be computed in $O(\log n)$ time by the EREW or CREW machines of Cook, Dwork, and Reischuk using only n processors and cells.

We consider computation of specific functions for the most general form of CRCW machine which we call the Abstract CRCW PRAM. We first show that such a machine can compute any function of Boolean inputs in time $\log n - \log\log n + O(1)$ given a polynomial number of processors and cells and that this bound is tight. Essentially the same machines are considered by Meyer auf der Heide and Wigderson (1985) in which the authors prove an $\Omega(\sqrt{\log n})$ time lower bound for sorting n integers and a $\Theta(\log n)$ lower bound for adding n integers. Their results rely on Ramsey theory and as a consequence their bounds only hold for integers which are extremely large—so large that the problems in question are not polynomial time computable given a sequential machine with an honest complexity measure like the log-cost RAM.

We show that on the Abstract CRCW PRAM the sum of n numbers requires $\Theta(\log n)$ time even if for example the numbers have only n bits each. Using a result of Hastad (1986) concerning unbounded fan-in circuits, the main bound we prove is that on this extremely powerful model some very simple functions on $\{0, 1\}^n$ require time $\Omega(\sqrt{\log n})$ to compute given a polynomial bound on the number of cells and processors. These functions include computing the parity of, sorting, or adding n bits as well as multiplying two $n/2$-bit integers. These results are the first non-trivial lower bounds for computing Boolean functions on CRCW PRAMs which do not rely on restricted instruction sets of processors or resources smaller

than the size of the problem input. They show that there is something inherent in the ways in which processors communicate with each other which limits their computational power.

2. THE ABSTRACT CRCW PRAM

DEFINITION. An (*Abstract*) *CRCW PRAM* is a shared memory machine with processors $P_1, ..., P_{p(n)}$ which communicate through memory cells $C_1, ..., C_{c(n)}$. The input is initially stored in the first n cells of memory, $C_1, ..., C_n$. Initially all cells other than the input cells contain the value 0. The output of the machine is the value in the cell C_1 at termination time $T(n)$.

Before each step t processor P_i is in state q_i^t. At time step t, depending on q_i^t, processor P_i reads some cell C_j of shared memory, then, depending on the contents of C_j and q_i^t, assumes a new state q_i^{t+1}, and depending on this state, writes a value $v = v(q_i^{t+1})$ into some cell.

When several processors are attempting to write into a single cell at the same time step the one that succeeds will be the lowest numbered processor.

A processor may write anything into its cells when it writes, including a full description of the portion of the history of the computation which it knows, as well as its own processor number. When there is no danger of confusion we will assume that a CRCW PRAM is an Abstract CRCW PRAM.

In the definition of an Abstract CRCW PRAM it becomes clear that, except for the final step of an algorithm in which the machine must output function values, the actual values of a processor's state or the contents of a cell are not important. What is important is the set of inputs which lead to the cell contents or state. The computation then may be viewed as operating not on actual values as much as on the partitions associated with them.

DEFINITION. Let M be a CRCW PRAM. For any processor P_i the *processor partition*, $P(M, i, t)$, of the input set at time step t is defined so that two inputs are in the same equivalence class of $P(M, i, t)$ if and only if they lead to the same state of processor P_i at the end of time step t.

For any cell C_j the *cell partition*, $C(M, j, t)$, of the input set at time t is defined so that two inputs are in the same equivalence class of $C(M, j, t)$ if and only if they lead to the same contents of cell C_j at the end of time step t.

The idea that these processor and cell partitions are the crucial aspects of

a computation is implicit in much of the lower bound work in this area. Snir (1985), for example, has given a formal explanation of the actions of a most powerful EREW PRAM which is essentially a method of describing the ways that the processor and cell partitions of that machine may be combined during a computation.

It will be useful to use the following restrictions of the Abstract CRCW PRAM in order to simplify our discussions of these machines.

DEFINITION. We say that a CRCW PRAM is a *common-write* CRCW PRAM if whenever several processors are attempting to write into a single cell at the same time step, the values that they are attempting to write are the same.

LEMMA 2.1. [Kucera (1982)]. *Let M be a CRCW PRAM with $p(n)$ processors and $c(n)$ memory cells, and taking $T(n)$ time. Then a common-write CRCW PRAM can simulate M by using $O(p(n)^2)$ processors, using $c(n)+p(n)$ memory cells, and taking $4T(n)$ time.*

Finally, for notational convenience we will assume unless otherwise stated that all logarithms are to the base 2.

3. The Computation of Arbitrary Boolean Functions and Integer Addition

By an argument similar to that of the Shannon bounds on combinatorial circuit complexity (see Savage (1976)), Ruzzo (1986) has shown that almost all Boolean functions require depth $\Omega(2^{n/2})$ to be computed by polynomial size unbounded fan-in circuits. Then by the simulations in Stockmeyer and Vishkin (1984) we see that restricted CRCW PRAMs require time $\Omega(2^{n/2})$ to compute most Boolean functions given a polynomial number of processors. The standard trick used for the following lemma shows that Abstract PRAMs are much more powerful than PRAMs with restricted instruction sets.

LEMMA 3.1. *For any function f of n inputs there is an Abstract (EREW) PRAM M that computes f in time $\lceil \log n \rceil + 1$ using $\lceil n/2 \rceil$ processors an n cells.*

Proof. The following algorithm computes any function of n inputs. In the first step processor P_i reads cell C_i. In the second step processor P_i reads cell C_{i+1} and writes the pair (x_i, x_{i+1}) into cell C_i. At the end of step $t+1$, $t \geqslant 0$, processor P_i will know and cell C_i will contain the 2^t inputs $(x_i, ..., x_{i+2^t-1})$. During step $t+1$, processor P_i reads cell C_{i+2^t}—thus

reading inputs $(x_{i+2^t}, ..., x_{i+2^{t+1}-1})$—and writes the encoding of the inputs $(x_i, ..., x_{i+2^{t+1}-1})$ which it knows into cell C_i. During step $\lceil \log n \rceil + 1$ processor P_1 knows the entire input so instead of writing the encoding of the input into cell C_1 it writes the function value into that cell.

Only half the processors in a given step need to continue on to the next step since their actions never affect the final output. In fact, from step t on, only processors whose indices are congruent to 1 modulo 2^t ever need to be active. ▮

The above algorithm uses the power of the processors to remember large portions of the input. If, for example, it is necessary to compute symmetric functions of Boolean inputs then machines with much less local memory would suffice. We shall refer to the algorithm as the binary fan-in tree algorithm. We now see that an Abstract CRCW PRAM can even beat the binary fan-in tree algorithm for inputs in $\{0, 1\}^n$.

THEOREM 3.1. *A common-write Abstract CRCW PRAM which has* $p(n) \geq 2n$ *and* $c(n) \geq p(n)/\log(p(n)/n)$ *can compute any function of inputs* $\{0, 1\}^n$ *in time* $\log n - \log\log(p(n)/n) + O(1)$.

Proof. Suppose there are $p(n) = n2^k$ processors ($k \leq n$). We exhibit an algorithm which computes the function in $\log n - \log k + 2$ steps using $p(n)/k$ memory cells. Break up the input into chunks of k bits each. Assign $k2^k$ processors to each chunk, and associate a label (i, α) with each processor for each $\alpha \in \{0, 1\}^k$ and each $i = 1, ..., k$.

(1) In the first step each processor with label (i, α) reads the ith bit within its chunk and writes a 1 into a cell labelled α for its chunk if and only if the value it read disagreed with the ith bit of α.

(2) In the second step one processor for each α within each chunk reads cell α and if it reads a 0 it writes α into a single cell designated for that chunk.

The input has now been effectively compressed from n bits in n cells to n/k k-bit integers in n/k cells. The remainder of the algorithm now uses the binary fan-in tree algorithm on the n/k cells which contain the description of the input to compute the function. This takes $\lceil \log n/k \rceil$ steps since there is already one processor which knows the contents of each cell. Since $\log n/k = \log n - \log k$ the running time is as claimed. ▮

For the CRCW PRAM, a problem which a number of authors have considered is that of integer addition. Namely, given n integers stored in n cells of memory compute their sum as an output in a single cell.

The first to consider this problem for CRCW PRAMs were Meyer auf der Heide and Reischuk (1984) who showed that if the instruction set of

the machines is limited to operations like addition, indirect addressing, basic logical operations, and conditionals as well as integer multiplication and the input integers are sufficiently large then the sum requires time $\log n$. Their results depend on the fact that all functions computed by such machines are polynomials of the original inputs of a certain form and so the instruction set comes into the proof in a crucial way. The integers for which they obtained the lower bound require $\Omega(np(n))$ bits to be represented where $p(n)$ is the number of processors in the machine.

Subsequently two papers, independent of each other, extended this result to CRCW PRAMs of the general form with which we are concerned in this paper. Meyer auf der Heide and Wigderson (1985) used the Erdös–Rado theorem to show that if the integers are sufficiently large then the sum requires $\log n$ time to compute on a general CRCW PRAM. Like many other Ramsey-theoretic arguments, in order to be sufficiently large, the integers involved must be huge—certainly containing exponentially many bits in the number of processors and memory cells. At about the same time Parberry (1986) used a more direct technique to show a better result, that the $\log n$ lower bound holds even if the integers have a number of bits which is a small degree polynomial in n and the number of processors and cells.

We are interested in obtaining lower bounds which hold for all machines with reasonable limitations on their resources in terms of the size of the problem instance. With this in mind, we observe that even the lower bound for integer addition which Parberry obtains requires the integers in the problem definition to have many more bits than the amount of resources allowed to solve the problem. This is not entirely satisfactory since it means that, in order to find a single family of instances which is hard for all machines with polynomially bounded resources, the integers must have a number of bits which grows faster than any polynomial in n.

We consider a function, related to the integer addition problem, which is, in a sense we describe, the hardest function to compute on $\{0, 1\}^n$. We give tight lower bounds for this function which yield tight lower bounds of $\log n$ for the integer addition problem with input integers having relatively few bits. Independently, Li and Yesha (1986) have obtained essentially the same lower bounds using quite different techniques which are similar to those of Parberry but which also use Kolmogorov complexity.

DEFINITION. Let A be a partition of a set $I \subseteq \{0, 1\}^n$. Define the *size* of A, $\#(A)$, to be the number of equivalence classes in A.

Let $b \geq 2$ and $\Sigma = \{0, 1, ..., b-1\}$. Let Encode_b be the function on Σ^n which converts an input string into the integer which it represents in base b. For inputs from Σ^n this is the hardest function to compute on the

CRCW PRAM since its output provides a complete description of the input and in one further step any processor can read the first cell and compute any other function.

THEOREM 3.2. *Any CRCW PRAM to compute the function* Encode_b *described above requires* $T(n) \geq \log n + 1 - \log[1 + \log_b 2p(n)]$.

Proof. The important fact about the definition of Encode_b is that when it has been computed the contents of the first cell must induce a partition of the inputs which has b^n distinct classes. That is any M that computes Encode_b must satisfy $\#(C(M, 0, T(n))) = b^n$. Let p_t and c_t for $t \geq 0$ satisfy the following recurrences:

$$p_0 = 1$$

$$c_0 = b$$

$$p_{t+1} = p_t c_t$$

$$c_{t+1} = p(n)\, p_{t+1} + c_t.$$

CLAIM. p_t (respectively c_t) is an upper bound over all processors (cells) of the size of the processor (cell) partition induced on the input set at the end of time step t, i.e.,

$$p_t \geq \max_i \#(P(M, i, t))$$

$$c_t \geq \max_j \#(C(M, j, t)).$$

Since the input is initially present in the n cells, $C_1, \ldots, C_n$ and since the processors initially have no access to it, it is clear that

$$\#(P(M, i, 0)) = 1 = p_0$$

$$\#(C(M, j, 0)) \leq b = c_0.$$

The cell which a processor reads during time step $t+1$ can depend only on its state at time t so that on elements of one class in the partition only one cell may be read. Thus during time step $t+1$ each class in the processor partition at the end of step t may be split into at most the number of classes in the partition of the cell from which it read. It follows that

$$\#(P(M, i, t+1)) \leq \#(P(M, i, t)) \max_j \#(C(M, j, t)) \leq p_t c_t = p_{t+1}.$$

A cell may have all $p(n)$ processors writing into it during step $t+1$ and each processor may communicate the entire partition of the portion of the

input on which it succeeds in writing into the cell. Also the cell may still maintain the partition of the portion of the input on which no processor writes. Thus the number of classes into which the contents of the cell may resolve the input satisfies

$$\#(C(M, j, t+1)) \leqslant \sum_{i=1}^{p(n)} \#(P(M, i, t+1)) + \#(C(M, j, t))$$
$$\leqslant p(n)\, p_{t+1} + c_t = c_{t+1}.$$

Thus the claim follows.

Easy calculation shows that for $t \geqslant 1$ we can bound c_t above by $(2p(n)b)^{2^{t-1}}$. In order to compute Encode_b in T steps, $c_T \geqslant b^n$ is needed. Therefore $2^{T-1}[1 + \log_b 2p(n)] \geqslant n$ and so $T \geqslant \log n + 1 - \log[1 + \log_b 2p(n)]$. ∎

By choosing $b = 2$ in Theorem 3.2 we see that the bound in Theorem 3.1 is asymptotically tight. Using Theorem 3.2 it is an easy step to prove a lower bound for the addition of integers.

COROLLARY 3.1. *The sum of n integers of $n \log b$ bits each requires time at least $\log n + 1 - \log[1 + \log_b 2p(n)]$ to compute on a CRCW PRAM with $p(n)$ processors.*

Proof. The function Encode_b is exactly the function computed by considering the ith input x_i as $x_i b^{i-1}$ and adding the resulting integers. The corollary follows immediately since the largest such integer is bounded above by b^{n-1} and so requires only $n \log b$ bits. ∎

COROLLARY 3.2. *For n sufficiently large, the sum of n integers with $\omega(n \log n)$ bits each requires time $\log n$ to compute on a CRCW PRAM given a number of processors polynomial in n.*

COROLLARY 3.3. *The sum of n integers with n bits each requires time $\Omega(\log n)$ to compute on a CRCW PRAM with as many as 2^{n^ε} processors for any $\varepsilon < 1$.*

There is something apparently contradictory about these results. The sum of n integers with polynomially many bits can be computed in $O(\log n)$ depth using combinatorial circuits. Thus, using a general of Chandra, Stockmeyer, and Vishkin (1984), the sum can be computed in $O(\log n/\log\log n)$ depth using polynomial size unbounded fan-in circuits. How then can we reconcile the $\log n$ lower bound with the simulation of Stockmeyer and Vishkin?

The simulation of Stockmeyer and Vishkin shows how each output of an unbounded fan-in circuit can be computed on a CRCW PRAM in time equal to the depth of the circuit. Since circuits can only present each bit of an integer as a single output, each bit of the sum can in fact be computed in time $O(\log n/\log\log n)$ on a CRCW PRAM. It is merely the requirement that the entire output must appear in one cell which is reponsible for the additional complexity.

This illustrates a general phenomenon of function computation on CRCW PRAMs. In the following sense it really consists of decision problem computation plus computation of an appropriate version of the Encode function.

THEOREM 3.3. *For each n let f_n be a function defined on $\{0,1\}^n$ with range $R = R(n)$. Since $|R(n)| \leqslant 2^n$ for each n there is a binary encoding of R which uses at most n bits. Let d_i: $R(n) \to \{0,1\}$ be the function which yields the ith bit of this binary encoding function on $R(n)$. Let $p(n)$ and $c(n)$ grow as $n^{O(1)}$. Then*

$$\begin{aligned}&\log\log|R| - \log\log n + O(1)\\ &\quad\leqslant T_f \leqslant \max_i T_{d_i\circ f} + \log\log|R| - \log\log n + O(1)\\ &\quad\leqslant T_f + \log\log|R| - \log\log n + O(1),\end{aligned}$$

where $T_{d_i\circ f}$ is the time required to compute $d_i\circ f$ on a CRCW PRAM with $p(n)$ processors and $c(n)$ memory cells, and T_f is the time to compute f on a CRCW PRAM with $np(n)$ processors and $nc(n)$ cells.

Proof. The first inequality follows from the bound, $c_t \leqslant (4p(n))^{2^{t-1}}$, given in the proof of Theorem 3.2 and the third inequality follows trivially since $d_i \circ f$ can be computed in one step given the value of f. The second inequality follows from the algorithm which computes each bit of the encoding of R separately and then encodes the result using the algorithm given in Theorem 3.1. ∎

COROLLARY 3.4. *Given f, R, d_i, T_f, and $T_{d_i\circ f}$ as defined in the theorem above*

$$T_f = \Theta(\max_i T_{d_i\circ f} + \log\log|R| - \log\log n).$$

This corollary really implies that the only thing which is interesting about function computation on the CRCW PRAM as opposed to decision problem computation is the Encode function. We consider decision problem computation on the CRCW PRAM in Section 5.

4. Partitions, Degrees, and Restrictions

We need a few definitions which will allow us to define a measure of progress for CRCW PRAM computation. Since these have a somewhat independent interest we put these in a separate section.

We may describe each equivalence class in a partition of the input set $I=\{0,1\}^n$ by a Boolean formula which expresses the characteristic function of that equivalence class. When this formula is written in disjunctive normal form (DNF), the subset of the inputs which it describes may be written as the union of inputs which satisfy each clause.

Definition. For any partition A of $J\subseteq I=\{0,1\}^n$ let the *degree* of A, $\delta(A)$, be the length of the longest clause in a minimal DNF formula for the characteristic function of each equivalence class in A when considered as a subset of J.

Remark. If a partition B is a refinement of partition A then $\delta(A)\leqslant\delta(B)$.

A *restriction* π of the input set I is a map

$$\pi\colon\{1,2,\ldots,n\}\to\{0,1,*\},$$

where

$$\pi(i)=\begin{cases}1 & \text{means } x_i \text{ is set to 1}\\ 0 & \text{means } x_i \text{ is set to 0}\\ * & \text{means } x_i \text{ is unset.}\end{cases}$$

Definition. For any partition A of I and any restriction π let $A\lceil_\pi$ be the restriction of A to the set $I\lceil_\pi=\{x\in I\colon \forall x_i \text{ set by } \pi,\ x_i=\pi(i)\}$. If F is a Boolean formula then $F\lceil_\pi$ is the formula obtained by replacing each literal corresponding to an input which π sets by the truth value assigned by $\pi(i)$.

Lemma 4.1. *Let π be a restriction and A a partition of I. If F is a DNF formula for the characteristic function of some equivalence class $C\in A$ then $F\lceil_\pi$ is a DNF formula for the characteristic function of the corresponding equivalence class $C\lceil_\pi\in A\lceil_\pi$ considered as a subset of $I\lceil_\pi$.*

Definition. A *random restriction* ρ chosen from R_q is a function which independently assigns 0, 1, or $*$ to each $i\in\{1,2,\ldots,n\}$, where

$$\Pr[\rho(i)=*]=q$$
$$\Pr[\rho(i)=1]=\tfrac{1}{2}(1-q)$$
$$\Pr[\rho(i)=0]=\tfrac{1}{2}(1-q).$$

Let π be a restriction of the input set I. A *random q-specialization* π' *of* π is a restriction which agrees with π on all the inputs set by π and which takes the values set by a randomly chosen $\rho \in R_q$ on the remaining inputs.

An important measure of the progress in the computation of a CRCW PRAM will be $\delta(A\lceil_\pi)$, where A is $P(M, i, t)$ or $C(M, j, t)$ and π is an appropriate restriction.

5. Lower Bounds for Some Simple Functions

The parity function is the function on binary values $x_1, ..., x_n$ which produces their sum modulo 2. Furst, Saxe, and Sipser (1984) gave an $\Omega(\log^* n)$ lower bound on the depth of polynomial size unbounded fan-in circuits computing parity. Ajtai, extending the results in Ajtai (1983), and Babai (1984), improved the depth lower bound for polynomial size parity circuits to $\Omega(\sqrt{\log n})$. Independently, by applying and modifying the techniques of Furst, Saxe, and Sipser (1984), we were able to derive an intermediate lower bound (Beame (1985)) which states that any CRCW PRAM computing the parity function and having $p(n)$ unbounded but $c(n) = n^{O(1)}$ requires time $T(n) = \Omega(\sqrt{\log \log n})$.

The results for unbounded fan-in circuits were based on efforts to achieve exponential lower bounds for constant depth circuits computing parity. Yao, in a breakthrough paper (Yao, 1985), was able to give exponential lower bounds for constant depth parity circuits but his results do not imply anything better than $\Omega(\sqrt{\log n})$ lower bounds for polynomial size parity circuits. Also, because of the notion of approximation, Yao's proofs do not appear to translate well to the machine model with which we are concerned.

Using some of the techniques in Yao (1985), Hastad (1986) has obtained improved lower bounds for parity circuits. His improved results yield $\Omega(\log n/\log \log n)$ lower bounds for polynomial size parity circuits which match the upper bounds for such circuits given by Chandra, Stockmeyer, and Vishkin (1984). He also eliminates the necessity for approximation as used by Yao. This makes his proofs amenable to modification and application to obtain the following result which has a much shorter proof than that in Beame (1985).

THEOREM 5.1. *If M is a CRCW PRAM which computes the parity function with $p(n) = n^{O(1)}$ and $c(n)$ unbounded then $T(n) = \Omega(\sqrt{\log n})$.*

During the course of the computation of a CRCW PRAM its processor and cell partitions are becoming more and more complex. The basic idea of

the proof of this theorem is that despite this if we set the values of some of the input variables then the processor and cell partitions have small degree on the remaining variables. Of course the number of variables whose values are set must grow over time as the partitions become more complex. The result will follow because any partition for the restricted parity function requires degree equal to the number of the remaining variables.

Proof of Theorem 5.1. We assume without loss of generality by Lemma 2.1 that M satisfies the common-write rule since the simulation only squares the number of processors. We will also assume that $p(n) \geqslant n$ and when processors write a value they tag it with the time step during which they are writing. This does not conflict with the common write and can only transmit additional information.

We will define a restriction π_t for each step t of the computation such that after step t and after π_t is applied, the cell partitions will all have degree less than the number of unset bits. The lower bound will follow since parity has minimal DNF clauses which depend on *all* the unset bits.

Define π_0 so that $\pi_0(i) = *$ for all i (i.e., all bits are unset) and $\pi_{-1} = \pi_0$. π_t will be chosen from the random q_t-specializations of π_{t-1}, where q_t will be defined later. For $t \geqslant 0$, let $p_t = \max_i \delta(P(M, i, t)\restriction_{\pi_{t-1}})$, $c_t = \max_j \delta(C(M, j, t)\restriction_{\pi_t})$, and m_t be the number of bits unset by π_t.

Let $b_t = 3^t \log_n p(n)$ and $q_t = (1/20b_t)\, p(n)^{-1/b_t} = (1/20b_t)\, n^{-3^{-t}}$.

CLAIM. For $t \geqslant 0$ we can choose π_t so that $p_t \leqslant b_t$, $c_t \leqslant 2b_t$, and $m_t \geqslant n2^{-t} \prod_{i=1}^{t} q_i$.

We show this by induction. The base case is $p_0 = 0 < c_0 = 1 \leqslant b_0$ and $m_0 = n$. The induction step is the following. Let $t \geqslant 1$. Since the cell which a processor reads during step t is dependent solely upon its current state, the new state which the processor assumes will depend only on the old state and the equivalence class in which the value in the cell read lies. Since the clauses in the cell partition have only c_{t-1} literals corresponding to unset input bits, the longest DNF clause needed to describe the new state's equivalence class will have at most $p_{t-1} + c_{t-1}$ literals corresponding to input bits which were unset after step t. Thus it follows that

$$p_t \leqslant p_{t-1} + c_{t-1} \leqslant 3b_{t-1} = 3^t \log_n p(n) = b_t.$$

From this recurrence it is easy to see that the concurrent reads do not give the CRCW PRAM much of its power. An identical recurrence would also hold for CREW PRAMs. It is the concurrent writes which cause all the difficulty.

We now try to bound c_t. Cells into which no processor writes during step t to any input $I\restriction_{\pi_{t-1}}$ will be unaffected by the write step and so will

have equivalence classes with clauses bounded by c_{t-1}. Thus we need only consider cells into which processors may write on some input in $I\lceil_{\pi_{t-1}}$. We say that such cells are used during write step t.

For the cells used during write step t, we first consider equivalence classes which correspond to values written by processors during step t. Becuase M satisfies the common-write rule, the set of inputs on which some value v is written into a particular cell is the union of the sets of inputs on which each processor writes v into that cell. In fact, because of the tagging by time step during writes, the equivalence class in the cell corresponding to v is exactly such a union. Since the set of inputs on which a particular processor performs any action is a union of equivalence classes in that processor's partition, the set of inputs on which some processor writes v into a particular cell is a union of equivalence classes of processors. Thus the equivalence class corresponding to v is a union of classes with DNF clauses bounded by p_t, and so its DNF clauses have maximum length at most $p_t \leqslant b_t$.

For each cell used during write step t it remains to consider the equivalence classes which correspond to the cases when no processor has actually written into the cell during step t. Each such class is the intersection of some old cell class and the set of inputs on which no processor writes into the cell during step t. We will choose π_t in such a way that the set of inputs on which no processor writes into the cell is described by a DNF formula whose clauses are bounded by b_t. Then each class within the cell will have DNF clauses bounded by $c_{t-1} + b_t$. To achieve this we note that the set of inputs on which some processor *writes* into a cell is a union of processor classes and so is already described by a DNF formula with clauses bounded by p_t. Then it follows that G, the formula describing the set of inputs on which *no* processor writes into the cell, is the negation of a formula with short DNF clauses and so has CNF clauses bounded by $p_t \leqslant b_t$. We now apply the following lemma which is essentially the main lemma of Hastad (1986).

LEMMA 5.1 [(Hastad (1986)]. *Let G be any CNF formula each of whose clauses has at most r literals. Let ρ be a random restriction chosen from R_q. Then*

$$\Pr[G\lceil_\rho \text{ has minimal DNF with any clause} \geqslant s] \leqslant (5qr)^s.$$

Thus if we choose a random q_t-specialization ρ of π_{t-1}

$$\begin{aligned}\Pr[\delta(C(M, j, t)\lceil_\rho) \geqslant c_{t-1} + b_t] &< (5q_t b_t)^{b_t} \\ &= (\tfrac{1}{4}p(n)^{-1/b_t})^{b_t} \\ &= 4^{-b_t}p(n)^{-1}.\end{aligned}$$

It is clearly only necessary to apply Hastad's lemma once per cell used during write step t. An upper bound on the number of cells used during write step t is certainly the number of possible states of all processors at the time of the write. This is just the number of classes in the processor partitions at the time of the write. Because the DNF clauses describing each equivalence class are bounded by $p_t \leqslant b_t$ and a class is the union of the set of inputs satisfying each DNF clause, each class contains a fraction of at least 2^{-b_t} of the possible inputs. The classes within a single processor partition are disjoint so there at most 2^{b_t} classes per processor. Thus at most $p(n)\,2^{b_t}$ cells are used during write step t. It follows that

$$\Pr[\exists \text{ a cell } C_j \text{ such that } \delta(C(M, j, t)\lceil_\rho) \geqslant c_{t-1} + b_t] < 2^{-b_t}. \tag{1}$$

If l is the number of bits unset by ρ then by Chebyshev's inequality we have

$$\Pr\left[l < \frac{m_{t-1}q_t}{2}\right] \leqslant \frac{4}{m_{t-1}q_t}. \tag{2}$$

Since the probabilities in (1) and (2) add up to less than 1 for n sufficiently large we may choose π_t to be one of the random q_t-specializations of π_{t-1} so that neither condition holds. In this case we have $c_t \leqslant c_{t-1} + b_t \leqslant 2b_t$ and $m_t \geqslant m_{t-1}q_t/2 \geqslant n2^{-t}\prod_{i=1}^{t} q_i$ by the inductive hypothesis. The claim follows by induction.

Now, parity requires t steps if $m_{t-1} > b_t$. This is satisfied provided

$$n2^{-(t-1)}\prod_{i=1}^{t-1} q_i > b_t$$

or provided

$$n^{1-(1/3+1/9+\cdots+1/3^{t-1})} > 40^{t-1}\prod_{i=1}^{t} b_i. \tag{3}$$

The right side of (3) is of the form $3^{t^2/2}(c_1 \log_n p(n))^t$ for some constant c_1. Since $p(n) = n^{O(1)}$, $\log_n p(n)$ is a constant. Condition (3) is then satisfied for some $t = \Theta(\sqrt{\log n})$. This in order to compute parity we must have $T(n) = \Omega(\sqrt{\log n})$. ∎

Remark I. It was not really necessary to require that $p(n) = n^{O(1)}$ to obtain the above lower bound. In fact one can show the same lower bound with $p(n)$ as large as $n^{2^{\varepsilon\sqrt{\log n}}}$ for some $\varepsilon > 0$. On the other hand one does not obtain a stronger lower bound using our techniques by polynomially bounding the number of memory cells as well as the number of processors.

Remark II. Using essentially the same argument as given above (in fact

a slightly simpler one) one can prove the same lower bounds for *common-write* CRCW PRAMs which have no restrictions on the number of processors but which have $c(n)$ bounded by $n^{2^{\varepsilon\sqrt{\log n}}}$ for some $\varepsilon > 0$.

Remark III. Methods found by Li and Yesha (1986) independently of this work can allow one to obtain results similar to those in Theorem 5.1 and Remark I. As actually stated in that paper the time bound is the same as that in Beame (1985); however, slightly tightening up the simulation of Abstract CRCW PRAMs by circuits and applying Hastad's instead of Yao's lower bounds for circuits will yield the improved bounds. It is not clear that one could obtain the results in Remark II by this method.

Using the reductions described by Chandra, Stockmeyer, and Vishkin (1984) we may derive the following bounds among others.

COROLLARY 5.1. *Any CRCW PRAM which sorts or adds n input bits or computes the product of two n-bit integers requires time* $\Omega(\sqrt{\log n})$ *if it uses a number of processors which is bounded by a polynomial in n.*

6. SUMMARY AND OPEN PROBLEMS

The Abstract CRCW PRAM we have described seems to be a natural model in which to prove lower bounds about concurrent-read–concurrent-write machines.

It is natural to try to improve on the parity lower bound for Abstract CRCW PRAMs to match the upper bound of $\log n/\log\log n$ given by Chandra, Stockmeyer, and Vishkin (1984). Hastad (1986) was able to do this for unbounded fan-in circuits but the processors in an Abstract CRCW PRAM are accumulating information about the input in a very different way from these circuits and for this reason it may be possible that the upper bound for these machines is not optimal.

The simulations in Stockmeyer and Vishkin (1984) shows that with a restricted instruction set a CRCW PRAM with a polynomial number of processors does not need more than a polynomial number of memory cells. The bounds in Beame (1985) and Theorem 5.1 appear to be incomparable since they restrict different resources—one restricts processors but not cells and the other restricts cells but not processors. It seems worthwhile elaborating general relationships between the number of processors and the number of cells in the case of unrestricted instruction set machines.

The most interesting open questions concerning the Abstract CRCW PRAM are the following: Are there specific Boolean functions for which we can prove stronger lower bounds than those for parity? Are there non-trivial lower bounds which match the values for the best known algorithms?

ACKNOWLEDGMENTS

I thank Steve Cook for several discussions which helped initiate this work and which helped formalize some of the main ideas in it. I also extend thanks to Al Borodin and Faith Fich for carefully reading drafts of this paper and for helpful comments which have corrected and improved its presentation. I am grateful to Friedholm Meyer auf der Heide for pointing me to Corollary 3.1.

RECEIVED August 13, 1985; ACCEPTED January 8, 1987

REFERENCES

AJTAI, M. (1983), Σ_1^1-Formulae on finite structures, *Ann. Pure Appl. Logic* **24**, 1–48.

BABAI, L. (1984), private communication.

BEAME, P. W. (1985), Lower bounds for very powerful parallel machines, manuscript.

CHANDRA, A. K., STOCKMEYER, L. J., AND VISHKIN, U. (1984), Constant depth reducibility, *SIAM J. Comput.* **13** (2), 423–439.

COOK, S. A., DWORK, C., AND REISCHUK, R. (1984), Upper and lower time bounds for parallel random access machines without simultaneous writes, *SIAM J. Comput.* **15** (1), 87–97.

FICH, F. E., MEYER AUF DER HEIDE, F., RAGDE, P., AND WIGDERSON, A. (1985), One, two, three... infinity: Lower bounds for parallel computation, *in* "Proc. 17th, ACM-STOC," pp. 48–58.

FURST, M., SAXE, J. B., AND SIPSER, M. (1984), Parity, circuits, and the polynomial time hierarchy, *Math. Systems Theory*, **17** (1), 13–28.

HASTAD, J. (1986), Improved lower bounds for small depth circuits, *in* "Proc. 18th ACM-STOC," pp. 6–20.

KUCERA, L. (1982), Parallel computation and conflicts in memory access, *Inform. Process. Lett.* **14** (2), 93–96.

LI, M., AND YESHA, Y. (1986), New lower bounds for parallel computation, *in* "Proc. 18th ACM-STOC," pp. 177–187.

MEYER AUF DER HEIDE, F., AND REISCHUK, R. (1984), On the limits to speed up parallel machines by large hardware and unbounded communication, *in* "Proc. 25th FOCS," pp. 56–64.

MEYER AUF DER HEIDE, F., AND WIGDERSON, A. (1985), The complexity of parallel sorting, *in* "Proc. 26th IEEE-FOCS," pp. 532–540.

PARBERRY, I. (1986), On the time required to sum n semigroup elements on a parallel machine with simultaneous writes, *in* "Proc. Aegean Workshop on Computation."

RUZZO, W. L. (1986), private communication.

SAVAGE, J. E. (1976), "The Complexity of Computing," Wiley, New York.

SNIR, M. (1985), On parallel searching, *SIAM J. Comput.* **14** (3), 688–708.

STOCKMEYER, L. J., AND VISHKIN, U. (1984), Simulation of parallel random access machines by circuits, *SIAM J. Comput.* **13** (2), 404–422.

VISHKIN, U., AND WIGDERSON, A. (1985), Trade-offs between depth and width in parallel computation, *SIAM J. Comput.* **14** (2), 303–314.

YAO, A. C. (1985), Separating the polynomial-time hierarchy by oracles: Part I, *in* "Proc. 26th IEEE-FOCS," pp. 1–10.