SIAM J. COMPUT.
Vol. 13, No. 2, May 1984

012

SIMULATION OF PARALLEL RANDOM ACCESS MACHINES BY CIRCUITS*

LARRY STOCKMEYER† AND UZI VISHKIN‡

Abstract. A relationship is established between (i) parallel random-access machines that allow many processors to concurrently read from or write into a common memory including simultaneous reading or writing into the same memory location (CRCW PRAM), and (ii) combinational logic circuits that contain AND's, OR's and NOT's, with no bound placed on the fan-in of AND-gates and OR-gates. Parallel time and number of processors for CRCW PRAM's are shown to correspond respectively (and simultaneously) to depth and size for circuits, where the time-depth correspondence is to within a constant factor and the processors-size correspondence is to within a polynomial. By applying a recent result of Furst, Saxe and Sipser, we obtain the corollary that parity, integer multiplication, graph transitive closure and integer sorting cannot be computed in constant time by a CRCW PRAM with a polynomial number of processors. This is the first nonconstant lower bound on the parallel time required to solve these problems by a CRCW PRAM with a polynomial number of processors. We also state and outline the proof of a similar result, due to W. L. Ruzzo and M. Tompa, that relates time and processor bounds for CRCW PRAM's to alternation and space bounds for alternating Turing machines.

Key words. synchronous parallelism, parallel time complexity, circuit complexity, relating complexity measures, alternating Turing machines

1. Introduction and statements of results. Our main motivation for this work was the goal of proving nontrivial lower bounds on the time required by a parallel random-access machine to solve certain problems when the parallel RAM model allows simultaneous reading from and writing into a common memory. Following Snir [25], we call this model a *CRCW PRAM* (for concurrent-read concurrent-write parallel RAM). A CRCW PRAM has a sequence of RAM's R_1, R_2, R_3, ... operating synchronously in parallel. Each individual RAM is similar to a standard one-processor RAM (cf. [1, Chap. 1]). Each RAM has its own *local* infinite random-access memory and has instructions for addition, subtraction, conditional branches based on the predicates = and <, and reading and writing into its local memory. (For the moment we assume that there are no multiplication, division or Boolean instructions.) The RAM's also have access to a *common memory*, and each RAM has instructions for reading from and writing into the common

*Received by the editors June 3, 1982, and in revised form July 29, 1983. Portions of this paper have been reprinted, with permission, from "A complexity theory for unbounded fan-in parallelism" by A. K. Chandra, L. J. Stockmeyer and U. Vishkin, appearing in the Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, 1982, pp.1-13, © 1982 IEEE. This paper was typeset at the IBM San Jose Research Laboratory.

†Computer Science Department, IBM Research Lab. K51/281, San Jose, California 95193. Work performed in part at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

‡Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. Work performed while the author was visiting the IBM Thomas J. Watson Research Center, while on leave from the Department of Computer Science, Technion, Haifa, Israel.

memory using one of its local registers to specify the common memory address. If more than one processor attempts to write into the same location in common memory at the same time, the lowest numbered processor succeeds. Each processor R_i has an instruction which loads its number i into a specified local register. All processors have the same program. Without loss of generality we assume that instructions are of the following forms, where M_1, M_2, M_3, ... denote local memory registers, and res (result), op1 and op2 (operand 1 and 2) are positive integers.

$M_{\text{res}} \leftarrow$ constant
$M_{\text{res}} \leftarrow$ processor number
$M_{\text{res}} \leftarrow M_{\text{op1}}$
$M_{\text{res}} \leftarrow M_{\text{op1}} \,@\, M_{\text{op2}}$, $@ \in \{+, -\}$
$M_{\text{res}} \leftarrow {}^*M_{\text{op1}}$ {local,common} (indirect memory READ)
The contents of the {local,common} location whose address is in register M_{op1} is read into local register M_{res}.
${}^*M_{\text{res}} \leftarrow M_{\text{op1}}$ {local,common} (indirect memory WRITE)
The contents of local register M_{op1} is written into the {local,common} register whose address is in M_{res}.
GOTO label
GOTO label if $M_{\text{op1}} \,@\, M_{\text{op2}}$, $@ \in \{=, <\}$
HALT

By convention, zero is the value read by an indirect READ using a nonpositive address. An indirect WRITE using a nonpositive address has no effect. In addition to the program, another part of the specification of a particular CRCW PRAM is a function $P(n)$ from positive integers to positive integers called the *processor bound*. An input of *size* n consists of n binary words, each of length at most n. A CRCW PRAM is given an input of size n by placing the n words in the first n locations of common memory, and the first $P(n)$ processors $R_1, \ldots, R_{P(n)}$ are started. Each instruction takes one time unit (uniform cost criterion). The computation halts when $R_1, \ldots, R_{P(n)}$ have all halted. The machine *operates in time* $T(n)$ if it halts within $T(n)$ steps on any input of size n. Output conventions are not critical, but we assume that when the computation halts, the output is in an initial contiguous block of common memory of length at most n locations.

This definition is based largely on the definition of Shiloach and Vishkin [24] (simultaneous writing as defined above is defined in [11], [26]); they use this model to give upper bounds on parallel time for several problems. [24] contains many references to other papers that give algorithms that can be implemented on a CRCW PRAM or restricted versions of it. The model is essentially identical to the SIMDAG of Goldschlager [11] and similar to the P-RAM of Fortune and Wyllie [9]. The P-RAM of [9] allows simultaneous reading but no simultaneous writing to the same common memory location; we call this model a *CREW PRAM* (for concurrent-read exclusive-write). (Whereas [9] and [11] characterized the power of these models when the number of processors grows exponentially in n, we are concerned mainly with the case of a polynomially bounded number of processors.) The model is also mentioned by Schwartz [23];

although he notes the physical difficulties of implementing large fan-in, he does state that such models "can play a useful role as theoretical yardsticks for measuring the limits of parallel computation."

For some models of parallel computation, fan-in considerations lead easily to a lower bound of $\Omega(\log n)$ on parallel time. For the CREW PRAM, the fan-in argument is considerably subtle, but Cook and Dwork [7] and Reischuk [19] succeed in proving $\Omega(\log n)$ lower bounds for problems such as computing the OR of n bits or the minimum of n numbers. For the CRCW PRAM, arguments based on bounded fan-in no longer work. For example, Shiloach and Vishkin [24] show that a CRCW PRAM with $O(n^2)$ processors can find the minimum of n numbers in constant time. However, Vishkin and Wigderson [28] recently proved nonconstant lower bounds in the case where the size of the common memory is limited. In fact, they prove a trade-off of $mT^2 = \Omega(n)$ for problems such as parity of n bits in the CRCW PRAM, where m is the size of the common memory, T is the parallel time and the input is in a read-only common memory. This result holds for any number of processors.

To aid our understanding of the power of CRCW PRAM's and to facilitate proofs of lower bounds, a characterization in terms of circuits would seem useful. There is considerable precedent for circuit characterizations of other computational models. Pippenger and Fischer [16] show a correspondence between serial (e.g., Turing machine) time and circuit size, and Borodin [2] shows a correspondence between serial space and circuit depth. Correspondences involving simultaneous resource bounds are given by Dymond and Cook [8], Hong [12], Pippenger [15], and Ruzzo [20]. Lev [13] shows a correspondence between time on a parallel RAM that does not allow simultaneous writing and circuit depth (unlike our result, the time-depth correspondence involves a $\log n$ factor rather than a constant factor). Whereas these correspondences use circuits with fan-in bounded by 2, the "correct" circuit analogue for CRCW PRAM's is the unbounded fan-in circuit studied by Furst, Saxe and Sipser [10]. These are acyclic circuits containing AND-gates, OR-gates, and NOT-gates.

More precisely, a *circuit* is an acyclic directed graph. Each node of the graph is labeled as either an input node, an AND-gate, an OR-gate, or a NOT-gate. Input nodes must have fan-in zero, and NOT-gates must have fan-in one. In addition, certain nodes are designated as output nodes. An assignment of Boolean values to all input nodes extends, in the obvious way, to Boolean values associated with all nodes. The *size* of a circuit is the number of edges (i.e., wires). The *depth* of a circuit is the length of a longest path from some input to some output.

Our main result is the following:

THEOREM 1. *There is a constant c and a function $q(P, T, n)$ bounded above by a polynomial in P, T, n, such that the following holds. Let R be a* CRCW PRAM *with processor bound $P(n)$ that operates in time $T(n)$. There is a constant d_R and, for each n, a circuit C_n of size $d_R \cdot q(P(n), T(n), n)$ and depth $c \cdot T(n)$ such that C_n realizes the input-output behavior of R on inputs of size n.*

Remarks.

1. $q(P, T, n) = O(PTL(L^2 + PT))$ where $L = O(n + T + \log P)$.

2. Theorem 1 remains true if the PRAM has instructions for bitwise Boolean operations and, or, negation, etc., as in vector machines [17].

3. Theorem 1 is not true if the individual RAM's have a unit cost multiplication instruction, since [10] shows that multiplication of n-bit numbers cannot be realized by a circuit of polynomial size and constant depth. However, Theorem 1 is true (with a different polynomial q) if multiplication and division are restricted to $O(\log n)$-bit numbers.

4. As noted above, concurrent write conflicts into the same common memory location are resolved by allowing the lowest numbered processor to write. One can imagine other reasonable alternatives. For example, one can assume that the program works correctly no matter which processor succeeds in writing, or that the program is such that whenever more than one processor attempts to write into the same location concurrently they are all writing the same thing (this latter assumption is used in [24]). Clearly Theorem 1 holds for CRCW PRAM's with either of these alternate assumptions. Theorem 1 holds for the CREW PRAM as well.

One advantage of circuits is that lower bounds may be easier to prove in the context of the fixed structure of circuits rather than the dynamic nature of a program. In this regard, since Furst, Saxe, and Sipser [10] have shown that polynomial-size and constant-depth circuits cannot compute parity, an immediate corollary is that a CRCW PRAM with a polynomially-bounded number of processors cannot compute parity in constant time. The same corollary holds for any function to which parity is reducible by a polynomial-size constant-depth circuit. Two examples of such functions from [10] are integer multiplication and graph transitive closure. Other examples from [5], [6] are determining whether a graph has a perfect matching and sorting (binary representations of) positive integers.

COROLLARY 1. *A* CRCW PRAM *with a polynomially-bounded number of processors that operates in constant time cannot compute parity, multiply integers, find the transitive closure of a graph, determine whether a graph has a perfect matching, or sort integers.*

Even given the elegant proof technique of [10], a direct proof of Corollary 1 would probably be very cumbersome.

To provide evidence that we have found the "correct" circuit analogue of CRCW PRAM's we can give a converse to Theorem 1. To avoid the usual mismatch of uniform programs with nonuniform circuits, we now allow programs to be nonuniform. In the nonuniform case, each R_i can have a different program, and the programs can depend on the size of the input. Theorem 1 remains true for nonuniform CRCW PRAM's although the polynomial q now depends also on the size of the programs, where the *size* of a program is the number of bits needed to write the program when constants and indices of registers mentioned in the program are written in binary.

THEOREM 2. *There are constants c_1, c_2 and c_3 such that the following holds. Let C be a circuit of size S and depth T that computes a function f having n inputs and at most n outputs. There is a nonuniform* CRCW PRAM *with $c_1(S+n)$ processors and program size $c_2\log(S+n)$ that runs in time $c_3(T+1)$ and computes f.*

Remark. Theorem 2 remains true under the two alternate concurrent write assumptions for CRCW PRAM's mentioned in Remark 4 above. However, we do not see how to prove Theorem 2 for the CREW PRAM. Since [7], [19] shows that an n-bit OR requires time $\Omega(\log n)$ on a CREW PRAM, Theorem 2

does not hold literally for the CREW PRAM; however, a time bound of $O(T + \log n)$ would not contradict [7], [19]. We leave circuit characterizations of the CREW PRAM and EREW PRAM [14], [25] as interesting open questions (the EREW PRAM allows neither simultaneous reading nor writing to the same location). If natural circuit characterizations of the CREW PRAM and EREW PRAM cannot be found, this may strengthen [26], [27] in supporting the CRCW PRAM model of computation if some kind of simultaneous access to a common memory is to be allowed.

Before giving the proofs, a few remarks on our view of the significance of this work should be made. First, a few words should be said about how realistic is the CRCW PRAM model. Certainly any physically realizable computer using current technology cannot have completely unbounded fan-in. On the other hand, fan-in bounded by 2 is probably too restrictive. For example, a realistic situation could have many processors writing onto a bus; this would be, in effect, an OR with fan-in equal to the number of processors. Lower bounds on CRCW PRAM time are very strong since the model is so powerful. Second, it should be noted that Furst, Saxe, and Sipser were led to unbounded fan-in circuits by the desire to separate the polynomial-time hierarchy from PSPACE by an oracle. Starting from a quite different motivation, we were led to exactly the same type of circuits. This provides further evidence that lower bounds on the depth and size of unbounded fan-in circuits is an important area of study. Finally, a common thread running throughout the history of theoretical computer science is the search for the "right" computational models. One way to support a model or models as being right is to show equivalence of seemingly different models. This has occurred in attempts to model "effective computation" (Turing machines, recursion equations, RAM's, etc. are equivalent), "serial time" (Turing machine time, serial RAM time, and circuit size are equivalent to within a polynomial), and "parallel time" (vector machine time, SIMDAG time, alternating Turing machine time, and circuit depth are equivalent to within a polynomial). By the equivalence between CRCW PRAM's and circuits stated in Theorems 1 and 2, these two models support each other as being right models of unbounded fan-in parallelism.

Another model of parallelism is the alternating Turing machine [4], although when viewed as a model of parallelism the fan-in is bounded. Ruzzo and Tompa [22] have shown that if one considers the alternation depth of the alternating machine, that is, the number of times that the machine switches from an existential state to a universal state or vice versa along any computation path, then this complexity measure corresponds quite closely to parallel time in the unbounded fan-in sense. Specifically, if $T(n)$ and $S(n)$ are suitably well behaved functions with $S(n) \geq \log n$ and $\log T(n) \leq S(n) \leq T(n)$, then the class of languages accepted by alternating Turing machines that are simultaneously $O(T(n))$ alternation bounded and $O(S(n))$ space bounded is precisely the class of languages accepted by CRCW PRAM's that are simultaneously $O(T(n))$ time bounded and $2^{O(S(n))}$ processor bounded. An advantage of this result, as compared to Theorems 1 and 2, is that both models are uniform. A disadvantage is that the relationship does not hold for constant $T(n)$; the parity function is computable by a deterministic Turing machine in space $\log n$ (i.e., $T(n) \equiv 1$ and $S(n) = \log n$), but as noted in Corollary 1 above, parity is not computable in constant time by a CRCW PRAM

with a polynomial number of processors. In § 3, we review the definition of alternating Turing machines, state Ruzzo and Tompa's result more precisely, and outline the proof.

2. Proofs of Theorems 1 and 2.

Proof of Theorem 1. We give the proof for a nonuniform CRCW PRAM. We have made no special attempt to control the size of the constant c or the polynomial q in the statement of the theorem; we have opted instead for a simple description. Fix an input size n. Let P, T and S be the processor bound, running time and program size, respectively. We can take the word length, i.e., the maximum length of the binary representations of all addresses of registers and values stored in registers, to be

$$L = \max(n, S, \log P) + T + 1.$$

This is true because initially the length of all words in the memory or the program is $\max(n, S, \log P)$, and each instruction can at most double the magnitude of a value (i.e., add one to its length). One extra bit is added to L to allow for negative numbers. So that addition and subtraction can be treated uniformly, arithmetic is done modulo 2^{L+1} and a negative number $-z$ is stored as $(2^{L+1}-z) \bmod 2^{L+1}$.

In our circuit simulation we represent local and common memories by sets of triples

$$(a_\ell(p,k,t), v_\ell(p,k,t), w_\ell(p,k,t)) \text{ and } (a_c(k,t), v_c(k,t), w_c(k,t))$$

where $a_\ell(p,k,t)$, $v_\ell(p,k,t)$, $a_c(k,t)$ and $v_c(k,t)$ are L-bit binary words, and $w_\ell(p,k,t)$ and $w_c(k,t)$ are single bits. If $w_\ell(p,k,t) = 1$, the triple $(a_\ell(p,k,t),v_\ell(p,k,t),w_\ell(p,k,t))$ means that the register with address $a_\ell(p,k,t)$ in the local memory of processor p contains the word $v_\ell(p,k,t)$ at step t. If $w_\ell(p,k,t) = 0$, the triple means nothing. The triples $(a_c(k,t),v_c(k,t),w_c(k,t))$ mean the same for common memory. If for some a_0, p and t there is no k such that $w_\ell(p,k,t) = 1$ and $a_\ell(p,k,t) = a_0$, then local register a_0 of processor p contains zero at step t, and similarly for common memory triples. Since a processor can write into at most one local or common register at each step, we can take $1 \le k \le T$ for local triples and $1 \le k \le n+PT$ for common triples. Let $s(p)$ be the number of instructions in the program of processor p (certainly $s(p) \le S$). The circuit simulation also computes, for each processor p, each step t, and each j with $1 \le j \le s(p)$, a bit $\mathrm{ic}(p,j,t)$ (instruction counter) which is 1 iff processor p should execute the j^{th} instruction of its program at step t. Let $x_1, \ldots, x_n$ be the input words stored in the first n locations of common memory, each padded out to L bits. At the start of the computation $(t = 0)$ set

$$(a_c(k,0), v_c(k,0), w_c(k,0)) = (k, x_k, 1) \qquad \text{for } 1 \le k \le n,$$

$$\mathrm{ic}(p, j, 0) = \begin{cases} 1 & \text{if } j = 1 \\ 0 & \text{if } j \ne 1. \end{cases}$$

All other memory triples with t = 0 are set to (0,0,0).

The proof will be completed by showing that changes to the memory triples and ic bits caused by one step of the PRAM can be computed by a circuit of

constant depth and size polynomial in P, T, S and n. The inputs to this circuit are the local and common memory triples and the ic bits before the step and the outputs are the same information after the step. We restrict our description to a "projection" of the general circuit on one processor. The general outline of the circuit for one processor is shown in Fig. 1. We now describe the pieces of Fig. 1 and explain how each can be implemented by a constant-depth polynomial-size circuit. A subcircuit that is used often is the circuit EQ that takes two L-bit binary words, say y and z, and produces 1 (*true*) iff $y = z$. This circuit has depth 4 and size $O(L)$ since

$$\mathrm{EQ}(y, z) = \bigwedge_{i=1}^{L} (y_i z_i \vee (\sim y_i)(\sim z_i)).$$

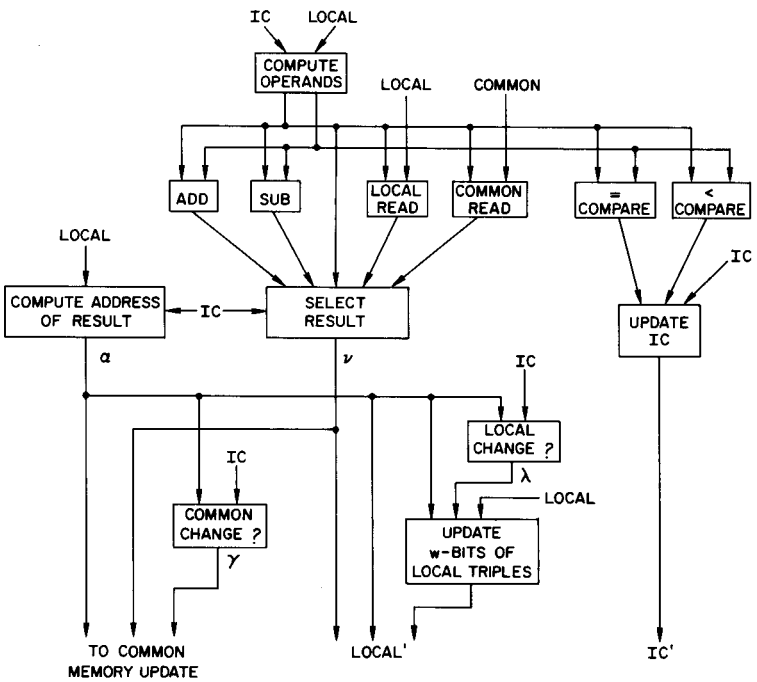

FIG. 1. *The high level outline of the circuit that simulates one step of one processor.*

Fix a processor p, $1 \le p \le P$, and a step t, $0 \le t < T$.

(1) Compute operands.

The outputs of this circuit are the values contained in local registers M_{op1} and M_{op2} for the instruction to be executed next. Let $\mathrm{op1}(p, j)$ be the L-bit binary representation of op1 in the j^{th} instruction of the program of processor p; for an instruction with no op1 (e.g. HALT) we can take $\mathrm{op1}(p, j) = 0$. The first operand is computed as

$$\bigvee_{j=1}^{s(p)} \bigvee_{k=1}^{T} \mathrm{ic}(p,j,t) \wedge \mathrm{EQ}(\mathrm{op1}(p,j), a_\ell(p,k,t)) \wedge w_\ell(p,k,t) \wedge v_\ell(p,k,t).$$

The rightmost $\wedge$ of this expression is computed bitwise over the bits of $v_\ell(p,k,t)$. This expression can be implemented as a circuit of constant depth and size $O(STL)$. The second operand is computed similarly.

After the operands are computed, the various operations are applied to the operands.

(2) Addition and subtraction.

Addition of two L-bit numbers $y = y_L \ldots y_2 y_1$ and $z = z_L \ldots z_2 z_1$ can be computed by a circuit of size $O(L^3)$ and constant depth. First compute the carry *generate* and *propagate* bits as $g_i = y_i \wedge z_i$ and $p_i = y_i \vee z_i$, respectively. The i^{th} carry bit c_i is 1 iff there is a $j < i$ such that $g_j = 1$ and $p_k = 1$ for all k with $j < k < i$. Therefore, all the carry bits can be computed by a circuit of size $O(L^3)$ and constant depth. Finally, the i^{th} bit of the sum is $y_i \oplus z_i \oplus c_i$. For subtraction, $y - z$, the binary representation of $2^{L+1} - z \bmod 2^{L+1}$ can be computed by complementing the bits of z and then adding 1. (Recently, Chandra, Fortune and Lipton [3] have significantly improved the $O(L^3)$ upper bound for the constant depth computation of addition.)

(3) Read from memory.

Let a_0 be the first operand computed in (1) above. The value read from common memory is

$$\bigvee_{k=1}^{n+PT} \mathrm{EQ}(a_0, a_c(k,t)) \wedge w_c(k,t) \wedge v_c(k,t).$$

The size is $O(L(n+PT))$. Reading from local memory is similar; the size is $O(LT)$.

(4) Comparison.

For nonnegative numbers y and z (i.e., $y_L = z_L = 0$), $y < z$ iff there is an i such that $z_i = 1$ and $y_i = 0$ and $y_j = z_j$ for all $j > i$. This can be computed by a circuit of size $O(L^2)$ and constant depth. There are three other cases depending on whether y or z or both are negative. Testing whether $y = z$ is done by EQ.

(5) Update instruction counter.

An example should suffice. If part of the program is

5: GOTO 7 if $M_{\text{op1}} < M_{\text{op2}}$
6: not a GOTO,

if 7 is not mentioned in any other GOTO's, and if c is the output of the $<$-comparison circuit, then

$$\mathrm{ic}(p, 7, t+1) = (\mathrm{ic}(p, 5, t) \wedge c) \vee \mathrm{ic}(p, 6, t).$$

The total size is $O(S)$.

(6) Select result.

Based on which $\mathrm{ic}(p,j,t)$ is 1, the correct L-bit result $v(p,t)$ is selected from the outputs of (1), (2) or (3). For instructions that load a constant or load the processor number, the proper constant is selected. The constant-depth implementation should be obvious at this point. The size is $O(LS)$.

(7) Compute address of result.

For instructions other than memory writes, the L-bit address $\alpha(p,t)$ is given by the number "res" in the instruction being executed at step t. For memory writes, the address is computed as in (1). The size is $O(STL)$.

(8) Local change? and common change?

The local (common) change bit $\lambda(p,t)$ ($\gamma(p,t)$) is 1 iff processor p changes the local (common) memory at step t. Let $C(p)$ be the set of instruction numbers of common memory WRITE instructions in the program of processor p. Let POS(α) be a circuit that computes 1 iff the L-bit α is positive. Recall the convention that an indirect WRITE using a nonpositive address has no effect. Then

$$\gamma(p,t) = \Big(\bigvee_{j \in C(p)} \mathrm{ic}(p,j,t) \Big) \wedge \mathrm{POS}(\alpha(p,t)).$$

$\lambda(p,t)$ is computed similarly. The size is $O(S+L)$.

(9) Update local memory.

Local memory of processor p is updated by setting

$$\begin{aligned} a_\ell(p,t+1,t+1) &= \alpha(p,t), \\ v_\ell(p,t+1,t+1) &= v(p,t), \\ w_\ell(p,t+1,t+1) &= \lambda(p,t), \end{aligned}$$

$$\begin{aligned} a_\ell(p,k,t+1) &= a_\ell(p,k,t) \quad \text{for } 1 \le k \le t, \\ v_\ell(p,k,t+1) &= v_\ell(p,k,t) \quad \text{for } 1 \le k \le t. \end{aligned}$$

Also, for each $1 \le k \le t$, $w_\ell(p,k,t+1)$ should be set to 0 if processor p is changing its local memory at this step ($\lambda(p,t)=1$) and the address $a_\ell(p,k,t)$ matches the address $\alpha(p,t)$ being changed at this step. Each w_ℓ can be updated by a constant depth circuit of size $O(L)$, or $O(TL)$ in all.

The circuitry of (1)-(9) must be replicated for each processor, so this gives size $O(P(STL + L^3 + L(n+PT)))$.

(10) Update common memory.

Next, the common memory triples are updated using $\alpha(p,t)$, $v(p,t)$ and $\gamma(p,t)$ from all the processors. First, $\gamma(p,t)$ must be set to 0 if there is a processor q, with $q < p$, attempting to write into the same address.

$$\gamma'(p,t) = \gamma(p,t) \wedge \sim\Big(\bigvee_{q<p} \gamma(q,t) \wedge \mathrm{EQ}(\alpha(q,t), \alpha(p,t)) \Big).$$

The total size for updating the γ's is $O(P^2L)$. The updating of common memory triples is similar to (9) except that a new block of P triples comes into play. For $1 \le p \le P$,

$$\begin{aligned} a_c(n+tP+p,t+1) &= \alpha(p,t), \\ v_c(n+tP+p,t+1) &= v(p,t), \\ w_c(n+tP+p,t+1) &= \gamma'(p,t). \end{aligned}$$

The computation of $w_c(k,t+1)$ for $k \leq n+tP$ is similar to (9). The size is $O(PL(n+PT))$.

Since all the circuitry must be replicated T times, the total size of the simulating circuit is $O(PT(STL + L^3 + L(n+PT)))$.

Finally, in order to assign output nodes in the circuit, the contents of the i^{th} location of common memory, $1 \leq i \leq n$, is computed as in (3) where now a_0 is the L-bit representation of i. Output nodes are now assigned to the results of this final computation depending on the output conventions of the particular problem being solved by the CRCW PRAM. □

As mentioned in Remark 3 above, Theorem 1 remains true if the PRAM has instructions for multiplication and division of $O(\log n)$-bit numbers. In fact, *any* Boolean function with $d \cdot \log_2 n$ inputs and m outputs can be computed by a circuit of depth 3 and size $O(mn^{d+1})$ by computing each output from its disjunctive normal form.

Proof of Theorem 2. By DeMorgan's laws we can assume that the circuit contains only OR-gates and NOT-gates. Given a circuit with S wires and N nodes, number the nodes from 1 to N such that the n input nodes are numbered from 1 to n. Let C_1, C_2, ... denote the common memory registers. Initially the i^{th} input bit is placed in C_i for $1 \leq i \leq n$. In general, the Boolean value computed by node number i in the circuit is computed by the CRCW PRAM in register C_i, and the values of all nodes at a fixed depth d in the circuit are computed in parallel. There is a processor associated with every wire. Say that a wire is directed from node i to node j, and node j is at depth d in the circuit. The processor associated with this wire waits cd steps, where the constant c is large enough that the values computed by all nodes at depth $< d$ have already been computed by the PRAM before step cd. The waiting is done by incrementing and testing a counter. If node j is an OR-gate, the processor writes 1 in C_j if C_i contains 1, or does nothing if C_i contains 0. If node j is a NOT-gate, the processor writes the negation of the contents of C_i into C_j. Other processors are assigned to output nodes. After waiting cT steps, these processors in parallel move the output bits to an initial contiguous block of common memory. □

Regarding the first sentence of the remark following the statement of Theorem 2, note that if several processors write into C_j at the same step, then j is an OR-gate and all these processors are writing 1.

3. Alternating Turing machines and CRCW PRAM's. In this section, some familiarity with the concept of alternation is assumed; see, for example, [4]. An alternating Turing machine (*ATM*) is like a nondeterministic Turing machine except that the states are partitioned into existential states, universal states, accepting states and rejecting states. Accepting and rejecting states are halting states. We consider ATM's with a read-only input tape and one read/write worktape. A configuration of an ATM consists of the state, the positions of the heads on the input tape and worktape, and the nonblank contents of the worktape. We let $\vdash$ denote the one-step transition relation on configurations. Let $\text{INIT}_M(x)$ denote the initial configuration of machine M on input x. An ATM M *accepts* input x iff there is a tree, called an *accepting computation tree of M on input x*, such that the root of the tree is $\text{INIT}_M(x)$, all leaves are accepting

configurations, if the existential configuration α is a node of the tree then α has one son β and $\alpha \vdash \beta$, and if the universal configuration α is a node of the tree then the sons of α are all those β with $\alpha \vdash \beta$. If L is a set of words, *M accepts L* if, for all x, M accepts x iff $x \in L$. With no loss of generality we assume that transitions from existential configurations to universal configurations, or vice versa, occur only when the machine is in one of a distinguished class of states called *switching states*. If β is an existential switching (resp., universal switching) configuration then there is exactly one α such that $\alpha \vdash \beta$ and α is a universal (resp., existential) configuration; moreover, β is the only configuration such that $\alpha \vdash \beta$. An ATM M is *$S(n)$ space bounded* if any configuration reachable from $\text{INIT}_M(x)$ uses at most $S(|x|)$ cells on the worktape. An ATM M is *$T(n)$ alternation bounded* if any computation path

$$\text{INIT}_M(x) \vdash \alpha_1 \vdash \alpha_2 \vdash \ldots \vdash \alpha_m$$

has at most $T(|x|)-1$ switching configurations. We let

$$\text{ATM-ALT-SPACE}(T(n), S(n))$$

be the class of languages $L \subseteq \{1,2\}^*$ accepted by ATM's that are simultaneously $T(n)$ alternation bounded and $S(n)$ space bounded.

A CRCW PRAM M *accepts* input $x \in \{1,2\}^*$ iff, when started with the i^{th} digit of x in common register i for $1 \le i \le |x|$, M halts with 1 in common register 1. Let

$$\text{CRCW-TIME-PROC}(T(n), P(n)) \quad .$$

be the class of languages $L \subseteq \{1,2\}^*$ accepted by CRCW PRAM's that are simultaneously $T(n)$ time bounded and $P(n)$ processor bounded.

The pair $(T(n), S(n))$ is *suitable* if

(1) $S(n) \ge \log n$ and $\log T(n) \le S(n) \le T(n)$,

(2) there is a deterministic Turing machine which, when given an input of length n, lays off a block of $S(n)$ tape cells and computes the binary representation of $T(n)$, and

(3) there is a (serial, uniform cost criterion) RAM which, when given an input of length n, halts within $O(T(n))$ steps with $S(n)$ and $T(n)$ in two registers.

For example, letting $\log n$ abbreviate $\lceil \log_2(n+1) \rceil$, the function $\log n$ is computable by a RAM in time $O(\log n)$: starting with $I=1$, the RAM successively doubles the variable I until the I^{th} common location contains a zero. Therefore, $(\log n, \log n)$ is suitable.

THEOREM 3 (Ruzzo, Tompa [22]). *Let $(T(n), S(n))$ be suitable.*

$$\text{ATM-ALT-SPACE}(O(T(n)), O(S(n))) = \text{CRCW-TIME-PROC}(O(T(n)), 2^{O(S(n))}).$$

Proof. Our goal is just to outline enough of the proof to allow the interested reader to complete the details. (The following proof, which uses the results of the previous section as a foundation, is due to the first author. Ruzzo and Tompa's original proof does not use unbounded fan-in circuits *per se*.)

1. ($\subseteq$) Let M be an ATM which is $O(T(n))$ alternation bounded and $O(S(n))$ space bounded. If the constant d is such that $d^{S(n)}$ is an upper bound

on the number of configurations of M on inputs of length n, then M accepts within time $d^{S(n)}$ [4, Thm. 2.6]. Fix an input x. A *timed configuration* (on input x) is a pair (α, t) where α is an $S(|x|)$ space bounded configuration of M and t is an integer with $0 \le t \le d^{S(n)}$. The relation $\vdash$ is extended to timed configurations by

$$(\alpha, t) \vdash (\beta, t') \text{ iff } \alpha \vdash \beta \text{ and } t' = t + 1.$$

The CRCW PRAM that simulates M first constructs the adjacency matrix A of an acyclic directed graph whose nodes are the timed configurations. If u and v are timed configurations, there is an edge directed from v to u (i.e., $A(v,u) = 1$) iff $u \vdash v$. The matrix A is computed and stored in common memory. To compute A, each processor views its processor number as a pair (u,v) of timed configurations. Since u and v are strings of length $O(S(n))$, time $O(S(n))$ is sufficient for each processor to "decode" its number to a pair (u,v) and check whether $u \vdash v$; the processor then writes 1 in $A(v,u)$ iff $u \vdash v$. Note that $2^{O(S(n))}$ processors are sufficient for this, and recall that $S(n) \le T(n)$.

A timed configuration (u, t) is *special* if u is halting, u is switching, or $(u, t) = (\text{INIT}_M(x), 0)$. The next step is to partially transitively close the graph so that $A(v,u) = 1$ iff there is a sequence $w_1, \ldots, w_m$ of *nonspecial* timed configurations such that

$$(*) \qquad u \vdash w_1 \vdash w_2 \vdash \ldots \vdash w_m \vdash v.$$

To compute this closure, each processor decodes its processor number to a triple (u,w,v) of timed configurations and checks whether w is special; this takes time $O(S(n))$. If w is special, the processor does not participate in computing the closure. If w is not special, the processor executes a sequence of phases. During a given phase, the processor writes 1 in location $A(v,u)$ iff both $A(v,w) = 1$ and $A(w,u) = 1$; each phase takes constant time. After t phases, all paths (*) of length $\le 2^t$ will have been discovered, so $\log_2 d^{S(n)} = O(S(n))$ phases suffice.

Next, it is easy to transform this graph to a circuit as follows. Remove all nonspecial nodes, view halting configurations as input nodes, view existential switching (resp., universal switching) configurations as OR-gates (resp., AND-gates), and view $(\text{INIT}_M(x), 0)$ as the single output node. This circuit has size $2^{O(S(n))}$ and depth $O(T(n))$. If input nodes corresponding to accepting (resp. rejecting) halting configurations are given value 1 (resp., 0), the value computed at the output node is 1 iff M accept x. As shown in the proof of Theorem 2, a CRCW PRAM with $2^{O(S(n))}$ processors can simulate this circuit within time $O(T(n))$.

2. ($\supseteq$). Let M be a CRCW PRAM which is $O(T(n))$ time bounded and $P(n) = 2^{O(S(n))}$ processor bounded. For a given input x, consider the circuit described in the proof of Theorem 1. By the well known trick of computing, for each gate, both the value computed by the gate and the negation of that value, we can assume that the circuit does not contain NOT-gates. Removing NOT-gates in this way does not increase depth and at most doubles size. This circuit has depth $O(T(n))$, and since it has size polynomial in P, T, and n, the nodes of the circuit can be named by binary strings of length

$$O(\log P(n) + \log T(n) + \log n) \text{ which is } O(S(n)).$$

The construction of this circuit is sufficiently uniform that, for a particular naming of the nodes, there is an $O(S(n))$ space bounded deterministic Turing machine which, when given x and the names of nodes u and v, determines whether there is an edge (i.e., wire) of the circuit from u to v, and determines the types of u and v, that is, input node assigned value 0, input node assigned value 1, output node, AND-gate, or OR-gate. This Turing machine is used as a subroutine in the following ATM program for simulating the circuit.

$v \leftarrow$ the output node of the circuit;
while v is not an input node do
    begin
        if v is an AND-gate (resp., OR-gate) then universally (resp., existentially) choose a node u such that there is an edge from u to v in the circuit;
        $v \leftarrow u$;
    end
if the input node v is assigned value 0 (resp., 1) then reject (resp., accept).

This ATM is $O(T(n))$ alternation bounded and $O(S(n))$ space bounded, and it accepts x iff M does. □

Whereas Theorem 1 was useful in proving nonconstant lower bounds on CRCW PRAM time, Theorem 3 appears to be more useful in proving upper bounds. An interesting class is the class of languages accepted by CRCW PRAM's in time $O(\log n)$ with a polynomial number of processors. By Theorem 3, this class is precisely

$$\text{ATM-ALT-SPACE}(O(\log n), O(\log n)).$$

It is implicit in a paper of Ruzzo [21, Example 1, Thm. 2] that any context free language is in ATM-ALT-SPACE$(O(\log n), O(\log n))$. The following corollary is immediate.

COROLLARY 2 (RUZZO). *If L is context free then L is accepted in time $O(\log n)$ by a* CRCW PRAM *with a polynomial number of processors.*

It is interesting to compare this corollary with the result of Reif [18] that any *deterministic* context free language is accepted by a CREW PRAM (i.e., no concurrent writing to the same location) in time $O(\log n)$ using a polynomial number of processors.

Acknowledgment. We thank Larry Ruzzo and Martin Tompa for allowing us to include their result relating alternating Turing machines to CRCW PRAM's.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
[2] A. BORODIN, *On relating time and space to size and depth*, this Journal, 6 (1977), pp. 733-744.
[3] A. K. CHANDRA, S. FORTUNE, AND R. J. LIPTON, *Unbounded fan-in circuits and associative functions*, Proc. 15th ACM Symposium on Theory of Computing, 1983, pp. 52-60.

[4] A. K. CHANDRA, D. C. KOZEN, AND L. J. STOCKMEYER, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114-133.

[5] A. K. CHANDRA, L. J. STOCKMEYER, AND U. VISHKIN, *A complexity theory for unbounded fan-in parallelism*, Proc. 23rd IEEE Symposium on Foundations of Computer Science, 1982, pp. 1-13.

[6] A. K. CHANDRA, L. J. STOCKMEYER, AND U. VISHKIN, *Constant depth reducibility*, this Journal, this issue, pp. xxx-xxx.

[7] S. COOK AND C. DWORK, *Bounds on the time for parallel RAM's to compute simple functions*, Proc. 14th ACM Symposium on Theory of Computing, 1982, pp. 231-233.

[8] P. W. DYMOND AND S. A. COOK, *Hardware complexity and parallel computation*, Proc. 21st IEEE Symposium on Foundations of Computer Science, 1980, pp. 360-372.

[9] S. FORTUNE AND J. WYLLIE, *Parallelism in random access machines*, Proc. Tenth ACM Symposium on Theory of Computing, 1978, pp. 114-118.

[10] M. FURST, J. B. SAXE, AND M. SIPSER, *Parity, circuits and the polynomial-time hierarchy*, Proc. 22nd IEEE Symposium on Foundations of Computer Science, 1981, pp. 260-270.

[11] L. M. GOLDSCHLAGER, *A universal interconnection pattern for parallel computers*, J. Assoc. Comput. Mach., 29 (1982), pp. 1073-1086.

[12] J.-W. HONG, *On similarity and duality of computation*, Proc. 21st IEEE Symposium on Foundations of Computer Science, 1980, pp. 348-359.

[13] G. LEV, *Size bounds and parallel algorithms for networks*, Doctoral Thesis, Report CST-8-80, Dept. of Computer Science, University of Edinburgh, 1980.

[14] G. LEV, N. PIPPENGER, AND L. G. VALIANT, *A fast parallel algorithm for routing in permutation networks*, IEEE Trans. on Computers, C-30 (1981), pp. 93-100.

[15] N. PIPPENGER, *On simultaneous resource bounds*, Proc. 20th IEEE Symposium on Foundations of Computer Science, 1979, pp. 307-311.

[16] N. PIPPENGER AND M. J. FISCHER, *Relations among complexity measures*, J. Assoc. Comput. Mach., 26 (1979), pp. 361-381.

[17] V. R. PRATT AND L. J. STOCKMEYER, *A characterization of the power of vector machines*, J. Comput. Sys. Sci., 12 (1976), pp. 198-221.

[18] J. REIF, *Parallel time* $O(\log n)$ *acceptance of deterministic CFLs*, Proc. 23rd IEEE Symposium on Foundations of Computer Science, 1982, pp. 290-296.

[19] R. REISCHUK, *A lower time bound for parallel random access machines without simultaneous writes*, Report RJ 3431, IBM Research Lab., San Jose, CA, 1982.

[20] W. L. RUZZO, *On uniform circuit complexity*, Proc. 20th IEEE Symposium on Foundations of Computer Science, 1979, pp. 312-318.

[21] W. L. RUZZO, *Tree-size bounded alternation*, J. Comput. Sys. Sci., 21 (1980), pp. 218-235.

[22] W. L. RUZZO AND M. TOMPA, personal communication.

[23] J. T. SCHWARTZ, *Ultracomputers*, ACM Trans. on Programming Languages and Systems, 2 (1980), pp. 484-521.

[24] Y. SHILOACH AND U. VISHKIN, *Finding the maximum, merging and sorting in a parallel computation model*, J. Algorithms, 2 (1981), pp. 88-102.

[25] M. SNIR, *On parallel searching*, Proc. SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottawa, Canada, 1982, pp. 242-253.

[26] U. VISHKIN, *Implementation of simultaneous memory address access in models that forbid it*, J. Algorithms, 4 (1983), pp. 45-50.

[27] U. VISHKIN, *A parallel-design space distributed-implementation space general purpose computer*, Report RC 9541, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1982.

[28] U. VISHKIN AND A. WIGDERSON, *Trade-offs between depth and width in parallel computation*, Proc. 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 146-153.