Improved Upper and Lower Time Bounds for Parallel Random Access Machines without Simultaneous Writes

Ian Parberry

Department of Computer Science,
333 Whitmore Laboratory,
The Pennsylvania State University,
University Park, Pa. 16802.

Pei Yuan Yan

Department of Mathematics,
407 McAllister Building,
The Pennsylvania State University,
University Park, Pa. 16802.

ABSTRACT

The time required by a variant of the PRAM (a parallel machine model which consists of sequential processors which communicate by reading and writing into a common shared memory) to compute a certain class of functions called *critical* functions (which include the Boolean OR of n bits) is studied. Simultaneous reads from individual cells of the shared-memory are permitted, but simultaneous writes are not. It is shown that any PRAM which computes a critical function must take at least $0.5 \log n - O(1)$ steps, and that there exists a critical function which can be computed in $0.57 \log n + O(1)$ steps. These bounds represent an improvement in the constant factor over those previously known.

January 31, 1990

Improved Upper and Lower Time Bounds for Parallel Random Access Machines without Simultaneous Writes

Ian Parberry

Department of Computer Science,
333 Whitmore Laboratory,
The Pennsylvania State University,
University Park, Pa. 16802.

Pei Yuan Yan

Department of Mathematics,
407 McAllister Building,
The Pennsylvania State University,
University Park, Pa. 16802.

1. Introduction.

A PRAM consists of an infinite collection of sequential processors connected to a common shared memory. In each time interval, or step, each processor reads a value from the shared-memory, moves into a new state and writes this new state back into the shared-memory. Simultaneous reads of a single cell in the shared-memory by many processors are permitted, but simultaneous writes are not. Computation of a function on n inputs proceeds by placing the n input values into the first n cells of the shared-memory, starting the PRAM with all processors in a pre-defined initial state, and waiting for all processors to halt. When the computation is over, the output can be found in the first shared-memory cell. The running-time of the PRAM is defined to be the number of steps taken, expressed as a function of n.

The lower-bound techniques discussed in this paper are based on a communication argument, and thus hold even if the processors have infinite computational power. Many of the ‘‘standard’’ models which have become popular in the recent literature (for example, in Fortune and Wyllie [3], Goldschlager [4, 5], and Shiloach and Vishkin

[11]) limit the local computational power of the individual processors. More details on the effect that this can have on the computational ability of PRAMs can be found in Parberry [6-8]. Our upper-bound is intended to illustrate the limits of the lower-bound technique by taking advantage of the unlimited computing power of the processors, and is not intended as a practical algorithm. We will, however, use a number of processors which is only a polynomial in the size of the input.

A Boolean function is said to be *critical* if there exists an input I with the property that changing any single bit of I changes its output. One example of a critical function is the Boolean OR of n bits (consider the all-zero input). The obvious parallel algorithm for computing the Boolean OR of n bits uses ''successive doubling'' and takes time $\lceil \log_2 n \rceil$. (From this point on, all logarithms will be to base 2 unless otherwise indicated). This intuitively seems to be a lower-bound, based on the number of bits that a processor can ''know about'' at each point in time. Independently, Cook and Dwork [2] and Reischuk [10] noticed that this ''obvious'' lower-bound is incorrect. These authors combined their results in [1]. From this reference we learn that any PRAM which computes a critical function on n inputs must take time at least $\log_b n$ where $b = 0.5(5+\sqrt{21})$ is slightly greater than 4.79. We will show that it must take time at least $\log_4 n - O(1)$. From it we also learn that there is a critical function that can be computed by a PRAM in time $\log_3 n + 1$. We will show that there is a critical function which can be computed by a PRAM in time $\log_c n + O(1)$ where $c = 2+\sqrt{2}$ is slightly greater than 3.41. The relationship between these results is summarized in Tables 1.1 and 1.2. Other lower-bounds for PRAMs without simultaneous writes can be found in Simon [12] and Snir [13].

	Lower-Bound	Upper-Bound
Previous	0.44	0.64
Current	0.5	0.57

Table 1.1 Previous best-known upper and lower-bounds on the time required for a PRAM to compute critical functions. Each entry in the table shows the constant c where the dominant term in the bounds is of the form c log n.

	Lower-Bound	Upper-Bound
Previous	3	4.79
Current	3.41	4

Table 1.2 Previous best-known upper and lower-bounds on the number of input symbols of a critical function that a PRAM can compute in t steps. Each entry in the table shows the constant c where the dominant term in the bounds is a constant times c^t.

The remainder of the paper is divided into four sections. The first describes the PRAM model in more detail, whilst the second contains some preliminary results concerning exclusive-write PRAMs. The lower and upper-bounds are contained in the third and fourth sections, respectively. A preliminary version of the results in this paper appeared as part of the second author's Ph. D. thesis [14], and in Parberry and Yan [9].

2. The PRAM Model.

A PRAM consists of an infinite number of processors and an infinite shared memory. The shared-memory is divided into *cells*, each of which is capable of holding a natural number (for the purposes of this paper, we count 0 as being a natural number). Each processor has a *state*, which is also a natural number. The cells and the processors are numbered consecutively from 0. More formally, a PRAM M con-

sists of a triple (r,s,w), where r,w:$\mathbf{N}^3 \rightarrow \mathbf{N}$, and s:$\mathbf{N}^4 \rightarrow \mathbf{N}$. An input to M consists of a finite sequence of natural numbers $I = (x_0, x_1, ..., x_{n-1})$. Each x_i is called an *input symbol*. At the start of the computation, x_i is placed into cell i of the shared-memory, $0 \leq i < n$, whilst cell i for $i \geq n$ is set to 0. Each processor is placed in state 0. The computation on input I proceeds in a sequence of discrete *steps*. During the t^{th} step, $t \geq 1$, each processor p, $p \geq 0$ does the following simultaneously. Suppose that p was in state $q \in \mathbf{N}$ at the end of the $(t-1)^{th}$ step (where the 0^{th} step refers to a point in time immediately before the computation began).

1. Read a value v from shared-memory cell r(p,t,q).
2. Compute $v' = s(p,t,q,v)$. The state of p at the end of the t^{th} step is v'.
3. Write the value v' into cell $w(p,t,v')$.

Simultaneous reads of a single shared-memory cell by many processors are permitted, but simultaneous writes into a single cell are not. The output of M will be found in shared-memory cell 0 when the processors have terminated. The *time* required by M is the maximum over all inputs of size n of the number of steps needed to process that input, expressed as a function of n.

For the purposes of our lower-bounds, we will be more lenient than is customary in our definition of what it means for a PRAM to compute a function, in that we will allow some pre-processing of the inputs before the computation begins, and post-processing of the output after the computation ends. We say that a PRAM *computes* f:$\mathbf{N}^* \rightarrow \mathbf{N}^*$ if there exist functions $\alpha,\beta:\mathbf{N} \rightarrow \mathbf{N}$ such that the PRAM on input $\alpha(x_0), \alpha(x_1), \ldots, \alpha(x_{n-1})$ produces output $y \in \mathbf{N}$ where $\beta(y) = f(x_0, ..., x_n)$. For our upper-bounds we will insist that no extra pre- or post-processing can be done, that is,

α and β are the identity functions.

We will say that two PRAMs are *functionally equivalent* if they compute the same function, and that they are *equivalent* if they are functionally equivalent and have the same running time.

3. Persistence and Predictability in PRAMs.

Definition 3.1 Let $I=(x_0,...,x_{n-1})\in\{0,1\}^n$ be an input string. Define $I(u)=(x_0,...,x_{u-1},\overline{x}_u,x_{u+1},...,x_{n-1})$, where $\overline{0}=1$ and $\overline{1}=0$, to be the input which is identical to I except in the u^{th} bit.

Definition 3.2 (Cook, Dwork and Reischuk [1]) If $0\le u<n$, index u *affects* shared-memory cell c at time t on input I if the contents of c at time t is different on inputs I and I(u).

Definition 3.3 (Cook, Dwork and Reischuk [1]) If $0\le u<n$, index u *affects* processor p at time t on input I if the state of p at time t is different on inputs I and I(u).

Definition 3.4 A PRAM is called *persistent* if the following two properties hold:

1. If index u affects processor p at time t–1 on input I, then index u affects processor p at time t on input I.
2. If index u affects cell c at time t–1 on input I and processor p reads cell c at time t on input I(u), then index u affects processor p at time t on input I.

The following two definitions are mutually recursive.

Definition 3.5 If $0 \leq u < n$, index u *counsels* shared-memory cell c at time t on input I if either $t = 0$ and c is the i^{th} shared-memory cell, or $t > 0$ and either:

(1) no processor writes into c on input I at time t and either:

  (i) u counsels c at time t–1 on input I, or

  (ii) there exists a processor p which writes into c at time t on input I(u), or

(2) some processor p writes into cell c at time t on input I, and either:

  (i) no processor writes into cell c at time t on input I(u), or

  (ii) some processor p′ writes into cell c at time t on input I(u) and either:

    (a) $p' \neq p$, or

    (b) $p' = p$ and u counsels processor p at time t on input I.

Definition 3.6 For $0 \leq u < n$, index u *counsels* processor p at time t on input I if either:

(1) $t > 1$ and u counsels p at time t–1 on input I, or

(2) $t \geq 1$, u counsels some cell c at time t–1 on input I and p reads c at time t on input I.

The definition of *counseling* is weaker than that of *affecting* in the sense the only way that an index u can affect a cell or a processor at time t on input I is to counsel that cell or processor at time t on input I. The converse is not necessarily the case (that is, not all counselling leads to an affectation) since, for example, in Definition 3.5 (1) (ii), processor p may at time t (fortuitously) write into cell c the same value that it contained at time t-1.

Definition 3.7 A PRAM is called *predictable* if

(i) it is persistent, and

(ii) every index u which counsels a cell or processor at time t on input I also affects that cell or processor at time t on input I, for all inputs I and $t \geq 1$, and

(iii) if the state of processor p_1 at time t_1 on input I_1 is the same as the state of processor p_2 at time t_2 on input I_2 then $t_1 = t_2$ and $p_1 = p_2$.

We can conclude from the observation preceding Definition 3.7 that not every PRAM is predictable. However, the following result will enable us to take advantage of the extra structure which is present in a predictable PRAM.

Lemma 3.8 For every PRAM there exists an equivalent predictable PRAM.

Proof. Let M be a T(n) time-bounded PRAM. Define a new PRAM M′ as follows. M′ simulates M, with processor p of M′ at time t simulating processor p of M at time t, subject to the following modifications. Let $p_0 = 2$ and for $i \geq 1$, p_i be the smallest prime number exceeding p_{i-1}.

(1) Suppose processor p of M′ was in state q at time t–1. If $q \neq 0$ then it factors q into a product of prime powers:

$$Q = p_0^{t-1} p_1^{p} \prod_{i=1}^{t-1} p_{i+1}^{q_i}$$

where $q_i \in \mathbf{N}$ for $1 \leq i \leq t-1$. The state of processor p of M at time t-1 can be computed from the sequence of values that it read during the first t-1 steps of the computation. The sequence $q_1,...,q_{t-1}$ is used for that purpose. If $q = 0$, then the state of processor p of M is taken to be 0.

(2) Using the state of processor p of M at time t-1 determined in step (1), it ascertains which cell in the shared memory that processor p of M will read from at time t, and reads a value v from that cell. It factors v into a product of prime powers:

$$v = p_0^{t'} p_1^{p'} \prod_{i=1}^{t'} p_{i+1}^{v'_i}$$

where $v'_i \in \mathbf{N}$ for $1 \le i \le t-1$. The value v was written there by processor p′ at time t′. The value that processor p′ of M would have written into that cell can be obtained by the sequence of values read by processor p′ of M in the first t′ steps of the computation. The sequence of values $v'_1,...,v'_{t'}$ is used for that purpose.

(3) Using the state of processor p of M at time t-1 determined in (1), and the value which processor p of M read from the shared-memory at time t determined in (2), it computes the place in the shared-memory into which the new state w of processor p of M at time t should be written.

(4) Finally, it computes its own new state $2qp_{t+1}^{v}$ and writes this into the shared-memory in place of w.

The correctness of the simulation can be verified by induction on t. Some extra preprocessing of the inputs to M is necessary before they can be presented to M′. Each input symbol x should be replaced by 5^x. Some post-processing of the output of M′ is also necessary to obtain the output of M. The output of M′ is be decomposed into a product of prime powers:

$$p_0^{T(n)} p_1^{p} \prod_{i=1}^{T(n)} p_{i+1}^{q_i}$$

where $q_i \in \mathbf{N}$ for $1 \le i \le t-1$. The output of M is then $q_{T(n)}$.

We claim that M′ is predictable. Condition (1) of Definition 3.7 requires that M′ be persistent. Suppose index u affects processor p of M′ at time t-1 on input I. Then state q is different on inputs I and I(u) in (1) above, which implies that the new state $2qp^{v}_{t+1}$ of processor p of M′ computed in (4) above is also different on inputs I and I(u), that is, u affects processor p of m′ at time t on input I. Now suppose that index u affects cell c of M′ at time t-1 on input I and that processor p of M′ reads cell c at time t on input I(u). There are two cases to consider.

Case 1. Processor p does not read cell c at time t on input I. Since it *does* read from there on input I(u), it follows that u must affect processor p at time t on input I.

Case 2. Processor p reads cell c at time t on input I. The value v read in (2) above must be different on I and I(u). Thus the new state $2qp^{v}_{t+1}$ computed in (4) must differ on I and I(u), that is, index u affects processor p at time t on input I. Thus we conclude that M′ is persistent.

Next we must demonstrate that property (ii) of Definition 3.7 holds, that is, all t and I, if index u counsels shared-memory cell c or processor p at time t on input I, then u affects shared-memory cell c or processor p, respectively, at time t on input I. The proof is by induction on t. The hypothesis is certainly true for $t=0$. Now suppose that $t>0$ and that the hypothesis is true at time $t-1$. Let I be an arbitrary input, and u an arbitrary index.

Case 1 Let p be a processor, and suppose that index u counsels p at time t on input I. Then by Definition 3.6, either:

Case 1.1 u counsels p at time t-1 on input I, which implies by the induction hypothesis that u affects p at time t-1 on input I. Since M′ is persistent, it follows that u affects p

at time t on input I.

Case 1.2 u counsels some cell c at time t-1 on input I and p reads cell c at time t on input I. By the induction hypothesis, u affects cell c at time t-1 on input I. Since M′ is persistent, this implies that u affects p at time t on input I.

Case 2 Let c be a shared-memory cell, and suppose that index u counsels c at time t on input I. Then by Definition 3.5 either:

Case 2.1 No processor writes into c at time t on input I and either:

Case 2.1(i) u counsels c at time t-1 on input I, which by the induction hypothesis implies that u affects c at time t-1 on input I, and so u must affect c at time t on input I.

Case 2.1(ii) There exists a processor p which writes into c at time t on input I(u). On input I(u), cell c contains at time t a value which is divisible by 2^t. This is not the case on input I. Therefore u affects c at time t on input I.

Case 2.2 Some processor p writes into c at time t on input I and either:

Case 2.2(i) No processor writes into c at time t on input I(u). The argument in this case is similar to Case 2.1(ii) above.

Case 2.2(ii) Some processor p′ writes into cell c at time t on input I(u) and either:

Case 2.2(ii)(a) $p' \neq p$. Without loss of generality, assume that $p > p'$. Then the value in cell c at time t on input I is divisible by 3^p, whereas the value in cell c at time t on input I(u) is not.

Case 2.2(ii)(b) $p' = p$ and u counsels processor p at time t on input I. By Case 1 above, u affects processor p at time t on input I, and hence the values written into c at

time t are different on inputs I and I(u).

In either case, index u affects cell c at time t on index I.

Finally, property (iii) of Definition 3.7 is easy to verify, since the state of all processors is uniquely ‘‘stamped’’ with its identity number and the time. This completes the proof that M′ is predictable. □

4. The Lower-Bound.

Lemma 4.1 Let M be a PRAM. Suppose $p_u \neq p_v$ are two processors, c is a shared-memory cell, and I an input. If p_u writes into cell c at time t on input I(u) and p_v writes into cell c at time t on input I(v), then either u affects p_v at time t on input I(v) or v affects p_u at time t on input I(u).

Proof. For a contradiction, assume the opposite and consider what happens at time t. On input I(u), p_u writes into c. Since by hypothesis v does not affect p_u at time t on input I(u), p_u must write into c at time t on input I(u)(v). A similar argument shows that p_v must also write into c at time t on input I(u)(v), which contradicts the fact that our PRAMs are exclusive-write. □

Definition 4.2

(i) K(p,t,I) is the set of indices which affect processor p at time t on input I.

(ii) $K(t) = \max_{p,I} |K(p,t,I)|$.

(iii) L(c,t,I) is the set of indices which affect cell c at time t on input I.

(iv) $L(t) = \max_{p,I} |L(p,t,I)|$.

Lemma 4.3 Let M be a predictable PRAM, c a shared-memory cell of M, and I an input. Suppose $u \in L(c,t,I)$. For all $v \in L(c,t,I)$, $u \neq v$, there exists a processor p such that one of the following holds. Either:

(i) $u \in L(c,t,I(v))$, or

(ii) $v \in L(c,t,I(u))$, or

(iii) $u,v \in K(p,t,I)$.

Proof. Suppose that hypotheses (i) and (ii) do not hold. Let $t_u \leq t$ be the last step before t in which some processor p_u writes into c on input I(u), $t_v \leq t$ be the last step before t in which some processor p_v writes into c on input I(v), and $t' \leq t$ be the last step before t in which some processor p′ writes into c on input I. If any of t_u,t_v,t' are undefined, set them equal to zero. Note that since $u \neq v$, at least one of t_u,t_v,t' is non-zero. There are two cases to consider:

Case 1 $t' \geq t_u,t_v$. Since u and v both affect c at time t on input I, they must both affect p′ at time t′ on input I, that is, $u,v \in K(p',t',I)$, which, since M is persistent, satisfies hypothesis (iii).

Case 2 $t' < \max(t_u,t_v)$. Without loss of generality, assume that $t_u \geq t_v,t'$. Let S(c,t,I) denote the contents of cell c at time t on input I. Then:

$$\begin{aligned} S(c,t_u,I(u)) &= S(c,t,I(u)) \\ &= S(c,t,I(u)(v)) \end{aligned}$$

since by hypothesis v does not affect c at time t on input I(u). Also, by a similar argument:

$$S(c,t_v,I(v)) = S(c,t,I(u)(v))$$

Therefore:

$$S(c,t_u,I(u)) = S(c,t_v,I(v))$$

Thus since M is predictable, we can conclude that $p_u = p_v$ and $t_u = t_v$. Thus $u,v \in K(p_u,t_u,I)$, which satisfies property (iii) since M is persistent. □

Lemma 4.4 Let $E \subseteq K(p,t,I)$, and $u \in E$. Let Y(u) be the set of indices $v \in E$ such that either u affects p at time t on input I(v), or v affects p at time t on input I(u). Then $|Y(u)| \geq |E| - K(t-1)$.

Proof. Let $E \subseteq K(p,t,I)$ and $u,v \in E$. There are two cases to consider:

Case 1 $u \in K(p,t-1,I)$. It is sufficient to demonstrate that if $v \notin Y(u)$ then $v \in K(p,t-1,I)$. Assume that $v \notin Y(u)$, and for a contradiction assume that $v \notin K(p,t-1,I)$. Let S(I) denote the state of processor p at time t–1 on input I. Then $S(I) = S(I(v))$ since, by hypothesis, v does not affect p at time t-1 on input I, and $S(I(v)) = S(I(u)(v))$ since M is persistent and (since $v \notin Y(u)$) u does not affect p at time t on input I(v). But $S(I(u)) = S(I(u)(v))$ by a similar argument, while $S(I) \neq S(I(u))$ since $u \in K(p,t-1,I)$. Thus:

$$S(I) = S(I(v)) = S(I(u)(v)) = S(I(u)) \neq S(I)$$

which is a contradiction. Therefore it must be the case that $v \in K(p,t-1,I)$.

Case 2 $u \notin K(p,t-1,I)$. If $v \in K(p,t-1,I)$ then by an argument similar to Case 1 (interchanging u and v) we see that $u \in Y(v)$, or equivalently, $v \in Y(u)$. If $v \notin K(p,t-1,I)$, then since $u,v \notin K(p,t-1,I)$ and M is predictable, it must be the case that both u and v affect some cell c at time t-1 on input I and p reads cell c at time t on input I. Thus $u,v \in L(c,t-1,I)$, and so by Lemma 4.3, one of the following holds. Either:

(i) $u \in L(c,t-1,I(v))$, which, since M is predictable, implies that $u \in K(p,t,I(v))$; that is, $v \in Y(u)$.

(ii) $v \in L(c,t-1,I(u))$, which by a similar argument implies that $v \in Y(u)$.

(iii) there exists a processor p' such that $u,v \in K(p',t-1,I)$. Thus, although v may not be a member of Y(u), there are at most $K(t-1)$ choices for such a v.

Thus we have shown that there are at most $K(t-1)$ members of $K(p,t,I)$ which are not members of Y(u), which implies that $|Y(u)| \geq |E| - K(t-1)$. □

The proof of Lemma 4.4 relies heavily on the fact that the PRAM in question is predictable. The corresponding result for ordinary PRAMs, which is implicit in Cook, Dwork and Reischuk [1], gives $|Y(u)| \geq |E| - K(t)$. This is the crux of our improvement in the lower-bound.

Lemma 4.5 For a predictable PRAM, when $t \geq 1$, $L(t) \leq 3 \times 4^t$.

Proof. We will prove that for a predictable PRAM, $K(0) = 0$, $L(0) = 1$ and for $t \geq 0$

$$K(t) \leq K(t-1) + L(t-1) \quad (1)$$

$$L(t) \leq 3\,(K(t-1) + L(t-1)) \quad (2)$$

Once this is established, it can easily be verified by induction that for $t \geq 1$, $K(t) \leq 4^{t-1}$ and $L(t) \leq 3 \times 4^{t-1}$.

The proof is by induction on t. The hypothesis is certainly true for $t = 0$. Now suppose that the hypothesis is true at time t–1. If u affects processor p at time t on input I, then either u affects p at time t–1 on input I or u affects some shared-memory cell c at time t–1 on input I (which p subsequently reads). Therefore:

$$\begin{aligned} K(p,t,I) &\leq K(p,t-1,I) + L(c,t-1,I) \\ &\leq K(t-1) + L(t-1) \end{aligned}$$

which implies inequality (1) as required.

Suppose that index u affects a cell c at time t on input I. Then either:

Case 1 No processor writes into c at time t on input I. Let Y(c,t,I) be the set of indices u which cause some processor to write into c at time t on input I(u). Then clearly:

$$|L(c,t,I)| \leq |L(c,t-1,I)|+|Y(c,t,I)|$$

which implies that:

$$|L(c,t,I)| \leq L(t-1)+|Y(c,t,I)| \quad (3)$$

It remains to show a bound on $|Y(c,t,I)|$. For each $u \in Y(c,t,I)$, let p_u be the processor which writes into c at time t on input I(u). Let Z(c,t,I) be the set of ordered pairs (u,v) such that $u,v \in Y(c,t,I)$ and either u affects p_v at time t on input I(v) or v affects p_u at time t on input I(u). For each u there are at most K(t) choices of v which can affect p_u at time t on input I(u). Thus:

$$|Z(c,t,I)| \leq 2|Y(c,t,I)|K(t)$$

(the factor of two comes from the fact that each $(u,v) \in Z(c,t,I)$ has also been counted as (v,u)).

Let E(u) be the set of indices $v \in Y(c,t,I)$ which cause p_u to write into c at time t on input I(v). For every $u \in Y(c,t,I)$, $(u,v) \in Z(c,t,I)$ for all $v \notin E(u)$ by Lemma 4.1. In addition, there are at least $|E(u)|-K(t-1)$ choices of $v \in E(u)$ such that $(u,v) \in Z(c,t,I)$ by Lemma 4.4. Therefore:

$$\begin{aligned} |Z(c,t,I)| &\geq |Y(c,t,I)|((|Y(c,t,I)|-|E(u)|)+(|E(u)|-K(t-1))) \\ &= |Y(c,t,I)|(|Y(c,t,I)|-K(t-1)) \end{aligned}$$

Therefore we have:

$$|Y(c,t,I)|(|Y(c,t,I)|-K(t-1)) \leq |Z(c,t,I)| \leq 2|Y(c,t,I)|K(t)$$

which implies that:

$$|Y(c,t,I)| \leq 2K(t) + K(t-1)$$

which when substituted into inequality (3) tells us that:

$$|L(c,t,I)| \leq L(t-1) + 2K(t) + K(t-1)$$

from which we can easily deduce inequality (2) by application of inequality (1).

Case 2 Some processor p writes into c at time t on input I and either:

Case 2(i) no processor writes into cell c at time t on input I(u). Then u must affect processor p at time t on input I, that is, $u \in K(p,t,I)$.

Case 2(ii) some processor p' writes into c at time t on input I(u) and either:

Case 2(ii)(a) $p' \neq p$. Then u must affect processor p at time t on input I, and again $u \in K(p,t,I)$.

Case 2(ii)(b) $p' = p$ and u affects processor p at time t on input I. Then immediately $u \in K(p,t,I)$.

In both Case 2(i) and 2(ii) we see that:

$$|L(c,t,I)| \leq |K(p,t,I)| \leq K(t) \leq K(t-1)+L(t-1)$$

(making use of inequality (1)), which implies that:

$$L(t) \leq K(t-1)+L(t-1) \leq 3(K(t-1)+L(t-1))$$

as required. □

Definition 4.6 Let $B=\{0,1\}$. If $f:B^* \rightarrow B$, $I \in B^n$ is called a *critical input* for f if for all $0 \leq u < n$, $f(I) \neq f(I(u))$. We call f a *critical function* if for all $n \geq 1$ there is an input in B^n which is critical for f.

Theorem 4.7 Any PRAM which computes a critical function on n inputs must take time at least $0.5 \log n - O(1)$.

Proof. Suppose M is a PRAM which computes a critical function f in time T(n). Without loss of generality we can assume that M is predictable (by Lemma 3.8). Let I be a critical input for f. Then $|L(0,T(n),I)| = n$. But Lemma 4.6 tells us that $|L(0,T(n),I)| \leq 3 \times 4^{T(n)}$. Therefore $T(n) \geq 0.5 \log n - O(1)$. □

5. The Upper-Bound.

In this section we will demonstrate a critical function which can be computed quickly on a PRAM. For the purposes of exposition we will present an algorithm which attempts to compute the Boolean OR of its inputs. The algorithm will fail for two reasons. Firstly, it will get the result wrong for many inputs. Secondly, some input symbols will be lost, that is, there will be indices u such that u does not affect the output cell when the algorithm has terminated on any input. However, we will demonstrate that the function computed by the algorithm is critical (the all-zero input will be critical), and that it is a function of sufficiently many input symbols for the required time-bound to hold.

The faulty algorithm for computing the OR of n bits proceeds as follows. Suppose that n is a power of four. During each step of the algorithm, the PRAM will attempt to reduce the number of bits to be processed by a factor of four by ORing

together groups of four bits. At time t there will be a set of cells C(t) which contain sub-results. $C(0)=\{0,1,...,n-1\}$. For each cell c at time t there will be a set of processors P(c,t) which write into c at time t on some input (although at most one member of P(c,t) will do so on any particular input). $P(c,0)=\varnothing$ for $c\geq 0$. We will ensure that $P(c_1,t)\cap P(c_2,t)=\varnothing$ when $c_1\neq c_2$. At time t, each cell c will contain a value v(c,t) which is either zero or of the form $2^{t'}$ where $t'\leq t$ is the last time that c was written into (t' is zero if c has not yet been written into).

In the first step of the algorithm, processor p reads the contents of cell p and writes two back there if the value read was one, in parallel for $0\leq p<n$. Thus $C(1)=C(0)$ and $P(c,1)=\{c\}$ for $0\leq c<n$. At time t, for $t>1$, the values from cells

$$c_i=a4^{t-1}+i4^{t-2}$$

for $0\leq i<4$ are examined, and an attempt is made to place a non-zero value into c_0 if at least one of their values is non-zero, in parallel for $0\leq a<n/4^{t-1}$. Thus by induction on t, for $t\geq 1$ it can be shown that

$$C(t)=\{a4^{t-1}\mid 0\leq a<n/4^{t-1}\}$$

At time t the following algorithm is used. Note that we adopt the convention that $\log_2 0=0$.

Algorithm 1.

Processors in $P(c_1,t-1)$ each perform the following:

Read a value v from cell c_3
$t_3:=\log_2 v$
if I wrote into cell c_1 at time t-1 **then**
if $t_3\neq t-1$ **then** write 2^t into cell c_0

Processors in $P(c_2,t-1)$ each perform the following:
    Read a value v from cell c_1
    $t_1:=\log_2 v$
    if I wrote into cell c_2 at time t-1 **then**
        if $t_1 \neq t-1$ **then** write 2^t into cell c_0
Processors in $P(c_3,t-1)$ each perform the following:
    Read a value v from cell c_2
    $t_2:=\log_2 v$
    if I wrote into cell c_3 at time t-1 **then**
        if $t_2 \neq t-1$ **then** write 2^t into cell c_0

We claim that there are no write-conflicts at time t. The proof is by induction on t. The hypothesis is certainly true for $t=1$. Now suppose that the hypothesis is true at time t-1. Thus we know that at most one processor from each of $P(c_i,t-1)$ wrote into cell c_i respectively at time t-1, for $1 \leq i \leq 3$. From Algorithm 1 we deduce that $P(c_0,t) = \bigcup_{i=1}^{3} P(c_i,t-1)$. Therefore the only possible write-conflicts may occur between some processors $p_1 \in P(c_1,t-1)$, $p_2 \in P(c_2,t-1)$, and $p_3 \in P(c_3,t-1)$.

Case 1. p_1 writes into cell c_1 at time t-1 and p_2 writes into cell c_2 at time t-1. Then only p_1 writes into c_0 at time t.

Case 2. p_1 writes into cell c_1 at time t-1 and p_3 writes into cell c_3 at time t-1. Then only p_3 writes into c_0 at time t.

Case 3. p_2 writes into cell c_2 at time t-1 and p_3 writes into cell c_3 at time t-1. Then only p_2 writes into c_0 at time t.

Case 4. p_1 writes into cell c_1 at time t-1, p_2 writes into cell c_2 at time t-1 and p_3 writes into cell c_3 at time t-1. Then no processor writes into cell c_0 at time t.

The aim of Algorithm 1 is to make c_0 at time t contain the Boolean OR of c_0, c_1, c_2 and c_3 at time t-1, in the sense that the former is to be non-zero if any of the latter are.

This does not happen in Case 4, preventing the algorithm from computing the Boolean OR of n inputs. This is not a major concern, however, since our aim is to have the algorithm compute some critical function. Unfortunately, it does not compute a critical function of all n inputs. Let $F_1(t)$ be the number of indices which affect a cell $c \in C(t)$ at time t on the all-zero input (due to the symmetry of the algorithm, this value will be the same for all such c). Then cell $c_0 \in C(t)$ contains a non-zero value at time t if it contained a non-zero value at time t-1 or some processor wrote into cells c_1, c_2 or c_3 at time t-1. Cell c_1 is affected by $F_1(t-1)$ indices at time t-1, and is affected by $F_1(t-2)$ indices at time t-2. Therefore any processor which writes into it at time t-1 is affected by $F_1(t-1) - F_2(t-1)$ indices. This processor also writes into c_0 at time t. A similar argument holds for cells c_2 and c_3. Because the first step of the algorithm is a special case, we are justified in taking $F_1(0) = 0$ and $F_1(1) = 1$. For $t > 1$,

$$F_1(t) = 4F_1(t-1) - 3F_1(t-2)$$

Therefore $F_1(t) = (3^t - 1)/2$. Thus we compute a critical function on n inputs in $\log_3 n + O(1)$ steps, a bound which appears in Cook, Dwork and Reischuk [1].

Now suppose that the value in cells c_1, c_2 and c_3 at time t-2 is identical for $t > 1$. This can be achieved by making the input symbols which affect c_2 and c_3 at time t-2 copies of the input symbols which affect c_1 at time t-1. We modify Algorithm 1 as follows:

Algorithm 2.

Processors in $P(c_1,t-1)$ each perform the following:
Read a value v from cell c_3
$t_3:=\log_2 v$
if I wrote into cell c_1 at time t-1 **then**
if $t_3 = 0$ **then** write 2^t into cell c_0
else if $0 < t_3 \le t-2$ **then**
if I am the smallest-numbered processor in $P(c_1,t-1)$
then write 2^t into cell c_0

Processors in $P(c_2,t-1)$ each perform the following:
Read a value v from cell c_1
$t_1:=\log_2 v$
if I wrote into cell c_2 at time t-1 **then**
if $t_1 = 0$ **then** write 2^t into cell c_0

Processors in $P(c_3,t-1)$ each perform the following:
Read a value v from cell c_2
$t_2:=\log_2 v$
if I wrote into cell c_3 at time t-1 **then**
if $t_2 = 0$ **then** write 2^t into cell c_0

We claim that there are no write-conflicts at time t. The proof is by induction on t. The hypothesis is certainly true for $t = 1$. Now suppose that the hypothesis is true at time t–1. Let

$$v = v(c_1,t-2) = v(c_2,t-2) = v(c_3,t-2)$$

Case 1. $v > 0$. Since the values in each cell are monotonically nondecreasing with time, we know that $t_i > 0$ for $1 \le i \le 3$. Since $t_3 > 0$, the smallest-numbered processor in $P(c_1,t-1)$ may write into c_0 at time t, and if any other processor from $P(c_1,t-1)$ wrote into c_1 at time t–1, then it is prevented from writing into c_0 at time t. Since $t_1 > 0$, no processor in $P(c_2,t-1)$ writes into c_0 at time t, and since $t_2 > 0$, no processor in $P(c_3,t-1)$ writes into c_0 at time t

Case 2. $v = 0$. For $1 \le i \le 3$, either $t_i = 0$ or $t_i = t-1$. Therefore Algorithm 2 behaves in a manner identical to Algorithm 1, and so the same arguments prevent a write-conflict

from occurring in this case.

Note that Algorithm 2 does not compute the same function as Algorithm 1. Let $F_2(t)$ be the number of indices which affect a cell $c \in C(t)$ at time t on the all-zero input (due to the symmetry of the algorithm, this value will be the same for all such c). Then $F_2(0) = 0$, $F_2(1) = 1$ and for $t > 1$:

$$F_2(t) = 4F_2(t-1) - 2F_2(t-2)$$

Therefore $F_2(t) = a((2+\sqrt{2})^t - b^{-t})$ where $a = \frac{3\sqrt{2}+4}{4(2\sqrt{2}+3)}$ and $b = 1+\frac{1}{\sqrt{2}}$.

Theorem 5.1 There is a critical function which can be computed on $F_2(t)$ inputs in t+O(1) steps by a PRAM using 4^t processors.

Proof. Suppose we are given $F_2(t)$ input bits for some $t \geq 0$. These bits are expanded to give 4^t bits by making multiple copies of each bit according to the requirements of Algorithm 2. Each processor p, $0 \leq p < 4^t$ examines the tree structure of the algorithm and determines which input bit that cell p should be a copy of. It then reads that bit and writes it into cell p. This takes two PRAM steps. Algorithm 2 is then executed in parallel t times. The total run-time is thus t+O(1) steps. □

Corollary 5.2 There is a critical function which can be computed in time $\log_c n + O(1)$ on n^d processors where $c = 2+\sqrt{2}$ and $d = 2/\log c < 1.13$.

6. Conclusion and Open Problems.

We have moved the upper and lower-bounds for the computation of critical functions on PRAMs closer together. The major remaining open problem is to make them meet. Our lower-bound holds for PRAMs which have powerful processors and can read and write large values. Can a better lower-bound be found for PRAMs which

write only zeros and ones? Our upper-bound differs from that of Cook, Dwork and Reischuk [1] in that it makes use of large values. Is this necessary? Is it possible to obtain better bounds for the computation of Boolean OR (which appears to be the most interesting critical function)? The best known upper-bound (which appears in [1]) is approximately $0.73 \log n + O(1)$.

7. References

1. S. Cook, C. Dwork, and R. Reischuk, ‘‘Upper and lower time bounds for parallel random access machines without simultaneous writes,’’ *SIAM J. Comput.*, vol. 15, no. 1, pp. 87-97, Feb. 1986.

2. S. A. Cook and C. Dwork, ‘‘Bounds on the time for parallel RAMs to compute simple functions,’’ *Proc. 14th Ann. ACM Symp. on Theory of Computing*, pp. 231-233, San Francisco, CA, May 1982.

3. S. Fortune and J. Wyllie, ‘‘Parallelism in random access machines,’’ *Proc. 10th Ann. ACM Symp. on Theory of Computing*, pp. 114-118, 1978.

4. L. M. Goldschlager, ‘‘Synchronous parallel computation,’’ Ph. D. Thesis, Technical Report TR-114, Dept. of Computer Science, Univ. of Toronto, Dec. 1977.

5. L. M. Goldschlager, ‘‘A universal interconnection pattern for parallel computers,’’ *J. Assoc. Comput. Mach.*, vol. 29, no. 4, pp. 1073-1086, Oct. 1982.

6. I. Parberry, ‘‘A complexity theory of parallel computation,’’ *Ph. D. Thesis*, Dept. of Computer Science, Univ. of Warwick, May 1984.

7. I. Parberry, ‘‘Parallel speedup of sequential machines: a defense of the parallel computation thesis,’’ *SIGACT News*, vol. 18, no. 1, pp. 54-67, 1986.

8. I. Parberry, *Parallel Complexity Theory*, Research Notes in Theoretical Computer Science, Pitman Publishing, London, 1987.

9. I. Parberry and P. Y. Yan, ‘‘Improved upper and lower time bounds for parallel random access machines without simultaneous writes,’’ *Proc. 1989 International Conference on Parallel Processing*, vol. 3, pp. 226-233, St. Charles, IL, Aug. 1989.

10. R. Reischuk, ‘‘A lower time-bound for parallel random-access machines without simultaneous writes,’’ Research Report RJ3431, IBM Research, San Jose, Mar. 1982.

11. Y. Shiloach and U. Vishkin, ‘‘Finding the maximum, sorting and merging in a parallel computation model,’’ *J. Algorithms*, vol. 2, pp. 88-102, 1981.

12. H. Simon, ‘‘A tight Ω(log log n)-bound on the time for parallel RAM’s to compute nondegenerated Boolean functions,’’ *Inf. Control*, vol. 55, pp. 102-107, 1982.

13. M. Snir, ‘‘On parallel searching,’’ *SIAM J. Comput.*, vol. 14, no. 3, pp. 688-708, Aug. 1985.

14. P.Y. Yan, ‘‘Lower bound techniques in some parallel models of computation,’’ Ph.D. Thesis, Dept. of Mathematics, Penn State University, 1989.