New Lower Bounds for Parallel Computation

MING LI AND YAACOV YESHA

The Ohio State University, Columbus, Ohio

Abstract. Lower bounds are proven on the parallel-time complexity of several basic functions on the most powerful concurrent-read concurrent-write PRAM with unlimited shared memory and unlimited power of individual processors (denoted by PRIORITY(∞)):

(1) It is proved that with a number of processors polynomial in n, $\Omega(\log n)$ time is needed for addition, multiplication or bitwise OR of n numbers, when each number has n^{ϵ} bits. Hence even the *bit complexity* (i.e., the time complexity as a function of the total number of bits in the input) is logarithmic in this case. This improves a beautiful result of Meyer auf der Heide and Wigderson [22]. They proved a $\log n$ lower bound using Ramsey-type techniques. Using Ramsey theory, it is possible to get an upper bound on the number of bits in the inputs used. However, for the case of polynomially many processors, this upper bound is more than a polynomial in n.
(2) An $\Omega(\log n)$ lower bound is given for PRIORITY(∞) with $n^{O(1)}$ processors on a function with inputs from $\{0, 1\}$, namely for the function $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i a^i$ where a is fixed and $x_i \in \{0, 1\}$.
(3) Finally, by a new efficient simulation of PRIORITY(∞) by unbounded fan-in circuits, that with less than exponential number of processors, it is proven a PRIORITY(∞) cannot compute PARITY in constant time, and with $n^{O(1)}$ processors $\Omega(\sqrt{\log n})$ time is needed. The simulation technique is of independent interest since it can serve as a general tool to translate circuit lower bounds into PRAM lower bounds.

Further, the lower bounds in (1) and (2) remain valid for probabilistic or nondeterministic concurrent-read concurrent-write PRAMs.

Categories and Subject Descriptors: C.1.2 [**Processors Architectures**]: Multiple Data Stream Architectures—*parallel processors*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*parallelism*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computation on discrete structures*

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: Addition, computational complexity, lower bounds, parallel random access machine, parity

---

Most of the results of this paper appeared in their preliminary forms in LI, M., AND YESHA, Y. New lower bounds for parallel computation. In *Proceedings of the 18th ACM Annual Symposium on Theory of Computing* (Berkeley, Calif., May 28–30). ACM, New York, 1986, pp. 177–187.

Results (1)–(3) listed in the abstract were also obtained independently, via totally different proofs, by Paul Beame [3, 4]. Beame also published his findings in the *Proceedings of the 18th ACM Symposium on Theory of Computing.*

This work was supported in part by the National Science Foundation under grant DCR 86-06366. M. Li was also supported in part by Office of Naval Research grant N00014-85-K-0445, ARO grant DAAL03-86-K-0171 at Harvard University, and NSERC operating grant OGP 0036747 at York University.

Authors' present addresses: M. Li, Department of Computer Science, York University at North York, Ontario M3J 1P3, Canada; Y. Yesha, Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210.

1. *Introduction*

A parallel random access machine (PRAM) consists of processors P(i), i = 1, 2, ..., and shared memory cells C(i), i = 1, 2 .... Each step of the parallel computation consists of three phases as follows: Each processor (1) reads from some shared memory cell, (2) may attempt writing into some shared memory cell, and (3) changes state. The state of each processor before step t is a function of its state before step $t - 1$, and the value read from the shared memory at step $t - 1$. The actions of each processor at step $t - 1$ are functions of its state before step $t - 1$.

The input $(x_1, \ldots, x_n)$ is initially placed in the shared memory. The value x_i is placed in the location C(i) for $i = 1, \ldots, n$. We say that the PRAM *computes the function* f if the following is true: Whenever $f(x_1, \ldots, x_n) = (a_1, \ldots, a_m)$, the computation terminates with a_i in the ith shared memory cell C(i).

The various variants of the PRAM differ in the way they handle *read or write conflicts*:

(1) *EREW (Exclusive Read Exclusive Write).* Read or write conflicts cannot occur.
(2) *CREW (Concurrent Read Exclusive Write).* Write conflicts cannot occur.
(3) *COMMON.* All the processors simultaneously writing into the same cell write the same value.
(4) *ARBITRARY.* An arbitrary processor succeeds in writing.
(5) *PRIORITY.* The processor with the minimum index succeeds in writing.

For N any of the above models, let $N(m)$ be the model with m shared memory cells. The *time* of the PRAM is the number of parallel steps used.

The PRIORITY model is the strongest one out of the above models. The PRIORITY(∞) (i.e., PRIORITY with unlimited shared memory) is the strongest known PRAM. The lower bounds we prove in this paper for PRIORITY(∞) PRAMs also apply to all other models.

All the above models are widely used for implementing parallel algorithms. For example, Hirschberg et al [15] and Preparata [26] used CREW (actually Preparata [26] also used an even weaker model, EREW, in which concurrent read is not allowed), Shiloach and Vishkin [28] and Galil [12] used COMMON, Shiloach and Vishkin [30] used ARBITRARY, and Awerbuch and Shiloach [2] used ARBITRARY and PRIORITY.

The purpose of this paper is to gain better understanding of the power of concurrent read and concurrent write PRAMs. Although many practical algorithms have been developed based on the various PRAM models, the limitation of what a PRAM *can* do and what a PRAM *cannot* do is still not clear. Several authors have obtained significant results in this direction, including [7]–[10], [21], [22], and [32]. (See also [19] and a survey paper by Reischuk [28].) In this paper we continue this research and obtain several nontrivial lower bounds on several very basic functions like addition of n integers and parity of n Boolean bits. We also develop two basic new methods for obtaining lower bounds for PRAMs.

2. $\Omega(\log n)$ *Lower Bound on the Bit Complexity of ADDITION and Related Functions*

Recently, Fich et al [9] and Meyer auf der Heide and Wigderson [22] proved several important lower bounds on PRIORITY(∞), including MAX, SORTING, ADDITION, MULTIPLICATION, using Ramsey theorems. (The lower bound on addition was also independently obtained by A. Israeli and S. Moran (private communication) and Parberry [23]). But all these lower bounds depend on inputs from infinite (or very large) domains. In practice, we are often interested in small inputs. For example, using a technique due to Reif [27], addition of n $n^{1/\log\log n}$ bit numbers can be done in $O(\log n/\log\log n)$ time with $n^{O(1)}$ processors which is less than the $\Omega(\log n)$ lower bound of [22]. This is not a contradiction, since the lower bound in [22] is for a PRAM, which is capable of adding arbitrarily large numbers. We improve the result of A. Israeli and S. Moran (private communication) and [22] and [23] to numbers of polynomial size (i.e., polynomial number of bits) by a new concept known as Kolmogorov complexity. We successfully use the remarkable notion of Kolmogorov complexity to obtain parallel lower bounds (and trade-offs) for a large class of functions with arguments in small domains (even with binary bit inputs) on PRIORITY(∞). Kolmogorov complexity has been fruitfully used to study sequential complexity [13, 17, 20, 24, 25], particularly in restricted Turing machine lower bounds. In this section we demonstrate how to use Kolmogorov complexity in obtaining general and optimal parallel lower bounds.

For the purpose of this paper, we fix an enumeration of oracle Turing machines. The Kolmogorov complexity of a string X relative to an oracle A, $K^A(X)$, is the size of smallest oracle Turing machine with oracle A which prints X. X is *random* relative to A if $K^A(X) \geq |X|$. A simple and well-known counting argument shows that for any oracle A and every large enough n, there exist strings of length n that are random relative to A. A function $f(x_1, \ldots, x_n)$ is *invertible* if x_i (for all i) can be computed from $(f(x_1, \ldots, x_n), x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$.

For each string w, the string $\bar{w}$ is obtained by doubling each letter in w. For integer i, bin(i) is the binary representation of i. Let $w' = \overline{\text{bin}(|w|)}01w$. The string w' is called the *self-delimiting* version of w. So "110011010101011" is the self-delimiting version of "01011." The self-delimiting binary version of a positive integer n requires $\log n + 2\log\log n + 2$ bits and the self-delimiting version of a binary string w requires $|w| + 2\log|w| + 2$ bits. All logarithms are base 2 unless otherwise noted.

The following theorem improves (and simplifies) results in [21] and [22]. Meyer auf der Heide and Reischuk [21] proved an $\Omega(\log n)$ lower bound for addition on a restricted PRIORITY(∞): A PRIORITY(∞) with a particular instruction set. Our results, however, do not depend on a particular instruction set, and allow each processor to have arbitrary power. Meyer auf der Heide and Wigderson [22] prove an $\Omega(\log n)$ lower bound for addition on the model we use. However, their result does not imply a polynomial upper bound on the size of the integers needed to achieve the lower bound.

THEOREM 2.1. *It requires $\Omega(\min[\log(b(n)/\log q), \log n])$ time to compute an invertible function $f(x_1, \ldots, x_n)$, where $|x_i| \leq b(n)$ for all i and $\log n = o(b(n))$, on a PRIORITY(∞) with q processors.*

PROOF. Suppose that a PRIORITY(∞) M with q processors computes $f(x_1, \ldots, x_n)$ in $o(\min[\log(b(n)/\log q), \log n])$ steps for infinitely many n's. We are actually talking about an infinite family of functions f_n, one for each n, and we assume that for each f_n we have a different PRIORITY(1) with a number of processors q, which is a function of n. However, for simplicity of notation we write, for instance $f(x_1, \ldots, x_n)$ instead of $f_n(x_1, \ldots, x_n)$. To prove the theorem it suffices to assume that $\log(b(n)/\log q) \leq \log n$. The programs (maybe infinite) of M can be encoded into an oracle A. The oracle, when queried about (i, l), returns the initial section of length l of the program for P(i). Fix a string $X \in \{0, 1\}^{nb(n)}$ such that $K^A(X) \geq |X|$. Equally divide X into n parts $x_1, x_2, \ldots, x_n$. Then consider the (*fixed*) computation of M on input $(x_1, \ldots, x_n)$. We inductively define (with respect to X) a processor to be *alive* at step t in this computation if

(1) it writes the output; or
(2) it succeeds in writing something at some step $t' \geq t$ which is read at some step $t'' > t'$ by a processor who is alive at step t''.

An input component is *useful* if it is read at some step t by a processor alive at step t. (Note that this is similar to the "communication pattern" used in [22].) It is easily seen, by induction on the step number, that for a computation that consists of T steps the number of useful input components and the number of processors ever alive are both $O(2^T)$.

It is not difficult to see that, given all the useful input components and the set $ALIVE = \{(P(i), t_i) \mid P(i)$ was alive until step $t_i > 0\}$, we can simulate M to uniquely reconstruct the output $f(x_1, \ldots, x_n)$.

Since $T = o(\log(b(n)/\log q))$ and $\log(b(n)/\log q) \leq \log n$, we know $2^T = o(n)$. Hence, there is an input component x_{i_0}, which is not useful. We need $O(2^T \log q) = o(b(n))$ bits to represent $ALIVE$. To represent $(x_1, \ldots, x_{i_0-1}, x_{i_0+1}, \ldots, x_n)$ we need $(n-1)b(n) + \log n$ bits, where $\log n$ bits are needed to indicate the index i_0 of the missing input component. The total number of bits needed is then

$$J = nb(n) - b(n) + \log n + o(b(n)) < nb(n).$$

(Since $\log n = o(b(n))$.) But from these J bits we can find $f(x_1, \ldots, x_n)$ by simulating M using the oracle A, and then reconstruct x_{i_0} from $f(x_1, \ldots, x_n)$ and $(x_1, \ldots, x_{i_0-1}, x_{i_0+1}, \ldots, x_n)$. This contradicts the randomness of X. □

An immediate application of the above theorem is to provide a lower bound for addition of small numbers. Note that addition is an invertible function.

COROLLARY 1. *For a PRIORITY(∞) with $n^{O(1)}$ processors, it requires*

(*a*) *$\Omega(\log n/\log\log n)$ time to add n $O(n^{1/\log\log n})$-bit numbers*
(*b*) *$\Omega(\log n)$ time to add n $O(n^{\epsilon})$-bit numbers*
(*c*) *$\Omega(\log\log n)$ time to add n $O(\log^k n)$-bit numbers, for $k > 1$.*

The result (b) has been independently obtained by Beame [3] using a different method that does not depend on Kolmogorov complexity. The result (c) is weaker than the results contained in Section 3.

We note that when the input size is small, the $\Omega(\log n)$ lower bound does not always hold. For example, for numbers up to size $n^{1/\log\log n}$, using a method of Reif [27], the following can be shown:

THEOREM 2.2. *Addition of n numbers of $O(n^{1/\log\log n})$ bits can be done in $O(\log n/\log\log n)$ time by a PRAM with $n^{O(1)}$ processors.*[1]

We next present an $\Omega(\log n)$ lower bound for a function with input components from $\{0, 1\}$. We actually prove a much more general result. We say that a function $f(x_1, \ldots, x_n)$ is *almost* 1–1 if for every y $|\{\bar{x} \mid f(\bar{x}) = y\}| \leq 2^{o(n)}$. All 1–1 functions are almost 1–1 functions.

THEOREM 2.3. *It requires $\Omega(\log n - \log\log q)$ time to compute an almost* 1–1 *function $f(x_1, \ldots, x_n)$ on a PRIORITY(∞) with q processors.*

[1] Reif [27] showed that adding n $O(\log n)$-bit numbers can be done in $O(\log n/\log\log n)$ time.

PROOF. Suppose that a PRIORITY(∞) M with q processors computes $f(x_1, \ldots, x_n)$ in $o(\log n - \log\log q)$ steps for infinitely many n's. The oracle A stores the programs of M as before. Fix a string $X \in \{0, 1\}^n$ such that $K^A(X) \geq |X|$. Let $x_i =$ ith bit of X. Then consider the (*fixed*) computation of M on input $(x_1, \ldots, x_n)$. We use all the definitions and facts from Theorem 2.1. Let $USEFUL = \{x_i \mid x_i \text{ is useful}\}$.

Using $ALIVE$, $USEFUL$, we can simulate M to uniquely reconstruct the output $f(x_1, \ldots, x_n)$. Since $T = o(\log(n/\log q))$, we know that $2^T \log q = o(n)$. Therefore $|USEFUL|$, $|ALIVE| < n/C$ for any constant C for n large enough. Now, to represent the elements in $USEFUL$ we need to describe the index of each $x_i \in USEFUL$. This information can be represented by

$$m, d_1, d_2, \ldots, d_m,$$

where $m = |USEFUL| < n/C$, and if $i = \sum_{j=1}^{k} d_j$ for $1 \leq k \leq m$, then $x_i \in USEFUL$. The values of the parameters m and the d_j's are coded self-delimiting. Also note that $\sum_{j=1}^{m} d_j \leq n$. Then by the convexity of the logarithm function, as long as we choose large enough C, the total number of bits needed to represent $USEFUL$ is no more than

$$m + 3m\left(\frac{\log n}{m} + \frac{2\log\log n}{m}\right) + O(\log n) \leq \frac{n}{3},$$

where the first term (and $O(\log n)$) is for all the x_i's (in self-delimiting coding), and the middle term is for representing the indices and m.

To represent $ALIVE$, since $2^T \log q = o(n)$, we need at most n/C bits for any fixed C. Now from $ALIVE$ and $USEFUL$ we compute $f(x_1, \ldots, x_n)$ using oracle A. Then, since f is almost 1–1, we reconstruct the entire input from f with an extra $o(n)$, say $n/4$, bits. Hence we can conclude that $K(X) < |X|$, contradicting the randomness of X. □

COROLLARY 2. *Let $f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i a^i$, where $x_i \in \{0, 1\}$ and $a > 1$ is fixed. Computing f on a PRIORITY(∞) with $n^{O(1)}$ processors requires $\Omega(\log n)$ time.*

PROOF. Because f is 1–1. □

Remark. All the results in this section are true for *nondeterministic*, and hence *probabilistic* PRIORITY(∞) as well. For the nondeterministic model, we have to supply for each pair $(P(i), t)$ a bit that determines which of the two nondeterministic choices available did processor $P(i)$ choose at step t. We can define $ALIVE$ as the collection of all triples $(P(i), t, b)$ such that processor $P(i)$ is alive at step t and b is its nondeterministic choice at this step. We select b's according to a fixed successful computation on the fixed input $(x_1, \ldots, x_n)$. $USEFUL$ is defined as before. Note that the size of $ALIVE$ is still $o(b(n))$. The rest of the proof is the same as the proof of the deterministic case.

3. *Time-Processor Trade-Off for PARITY on PRIORITY(∞)*

Since Cook and Dwork [7] proved an $\Omega(\log n)$ lower bound for Boolean OR on concurrent-read exclusive-write PRAM (see also [8]), the following question remained open: Show for a natural *Boolean* function (with n 1-bit input and one 1-bit output, e.g., PARITY) that it requires nonconstant time on the most general and powerful concurrent-read concurrent-write PRAMs with subexponential number of processors, each having arbitrary computation power. A related problem for unbounded fan-in circuits was solved by Furst et al. [11] and Ajtai [1]. Yao [33] and Hastad [14] improved the results of [11] and [1]. For restricted PRAMs (where each processor is restricted to have only certain instructions), the problem was solved by Chandra et al. [6] and Stockmeyer and Vishkin [31]. Fich et al. [9] proved a lower bound for computing the maximum of n integers on a PRIORITY(∞) with n processors. However, until the results of the present paper, and the results independently obtained by Beame [3], no nontrivial lower bound was obtained for the PRIORITY(∞) for any Boolean function.

We resolve the above open question. We prove that indeed a PRIORITY(∞) (the most powerful and general PRAM) that computes PARITY of n bits requires more than constant time with a subexponential number of processors, and $\Omega(\sqrt{\log n})$ time with polynomial many processors. More generally, we obtain a trade-off between time, number of processors and input size from which the above lower bounds follow.

Our result is based on an efficient general simulation of a PRAM by circuits. Unlike the simulation of [6] and [31] (which assumes restrictions on the power of individual processors), our simulation does not assume any such restrictions.

THEOREM 3.1. *There exists a constant c such that if a Boolean function $f(x_1, \ldots, x_n)$ is computed on PRIORITY(∞) with q processors in time T, then f can be computed by an unbounded fan-in Boolean circuit of size $q^{O(2^T)}$ and depth $O(T)$.*

PROOF. Without loss of generality, we may consider the COMMON(∞) models. Kucera [16] proved that COMMON(∞) with q^2 processors can simulate PRIORITY(∞) with q processors with only a constant slow down. Lemma 1 below applies to the PRIORITY model as well. However, the description of the circuit is simpler if the COMMON model is used. Hence, it is enough to consider COMMON(∞). Consider the computation of a Boolean function by a COMMON(∞) with q processors. As before, $K^A(x)$ denotes the Kolmogorov complexity of x relative to the oracle A that describes the COMMON(∞) program.

By definition of the PRAM, there exists a program Q with the following properties:

(1) With oracle A, the program Q can compute and output the state of a processor before step t, given as input: (a) the state of the processor before step $t - 1$ and (b) the contents before step $t - 1$ of the shared memory cell from which the processor read at step $t - 1$.

(2) With oracle A, the program Q can compute and output the contents of a shared memory cell before step t, given as input: (a) the contents of that cell before step $t - 1$, (b) the state before step $t - 1$ of some processor that wrote into that cell at step $t - 1$ (if such a processor exists), and (c) one more bit which is 0 if no processor wrote into this cell at step $t - 1$, and 1 if some processor wrote into this cell at step $t - 1$.

Let $S(t)$ be the maximum over all processors of the Kolmogorov complexity (relative to A) of the state of a processor before step t, and let $M(t)$ be the maximum over all shared memory cells of the Kolmogorov complexity, relative to A, of the contents of a shared memory cell before step t. We prove

LEMMA 1. *At step t, $S(t) \leq (2^{t+1} - 1)(\log q + v)$ and $M(t) \leq (2^{t+1} - 1)(\log q + v)$, where v is a constant.*

PROOF. By induction on t. It is easy to see that the Lemma is true for $t = 0$. Now suppose that it is true for all steps up to $t - 1$. Then, by (1) above, $S(t) \leq S(t - 1) + M(t - 1)$. By (2) above, $M(t) \leq S(t - 1) + M(t - 1) + 1$. Hence, the lemma is true for step t. □

Now, note that the address accessed by some processor depends only on its state. Hence, the Kolmogorov complexity (relative to A) of any address that is accessed by step t is at most $(2^{t+1} - 1)(\log q + v)$. Hence, $(2^{t+1} - 1)(\log q + v)$ bits are needed to represent any relevant state, address, and memory cell contents.

By Lemma 1, at most $G = 2^{T+1}(\log q + v)$ bits are required in order to represent each state, address, and contents of any shared memory cell which is ever accessed by step T. Hence, at most $U = 2^G$ distinct shared memory cells are accessed by step T. We may rename the shared memory addresses: If an address is represented by the binary string $z(|z| \leq G)$, which is the binary representation of the number j, then we call it address j.

The circuit consists of layers. Each layer is divided into levels. The first level in layer t includes the binary representations of all the states and contents of shared memory cells 0 through U before step t. The purpose of the other levels is to compute the states and the contents of the shared memory before step $t + 1$. We now describe the operation of layer t. It includes the following components:

(1) *Selection of Reading Addresses.* For each processor P(l) ($l = 1, \ldots, q$), a binary vector $r_{l1}, \ldots, r_{lU}$ is computed as a function of the state s of the processor. $r_{lj} = 1$ if, being in state s, P(l) reads from address j. $r_{lj} = 0$, otherwise. Each state is represented by at most G bits. Using disjunctive normal form, we construct an unbounded fan-in circuit to compute each r_{lj}. The circuit has depth 2 and size $2^{O(G)}$, which is $q^{O(2^T)}$. Hence, the circuit which computes all the r_{lj}'s has depth 2 and size $q^{O(2^T)}$.

(2) *Selection of Writing Addresses.* For each processor P(l), a binary vector $w_{l1}, \ldots, w_U$ is computed as a function of the state s of the processor. $w_{lj} = 1$ if, being at state s, processor P(l) writes into address j. $w_{lj} = 0$, otherwise. This circuit is similar to the circuit in (1) above, and has depth 2 and size $q^{O(2^T)}$.

(3) *Computing the Value z_l Read by Processor* P(l), *for* $l = 1, \ldots, q$. This can be easily done in constant depth and polynomial size, as a function of the contents of the shared memory cell, and the vector r_{lj}.

(4) *Computing the Value v_l Written by* P(l), *for* $l = 1, \ldots, q$. This value is a function of the state of the processor. Using disjunctive normal form, this can be done in depth 2 and size $q^{O(2^T)}$. The analysis is similar to the analysis in (1) above.

(5) *Computing the State of Processor* P(l) *before Step $t + 1$, for* $l = 1, \ldots, q$. This next state depends on the state of P(l) before step t, and on z_l. Again, using disjunctive normal form, this can be done in depth 2 and size $q^{O(2^T)}$.

(6) *Computing the Contents of Shared Memory Cell j before Step* $t + 1$, *for* $j = 1, \ldots, U$. Let $d_j = V_{i=1}^{q} \overline{w_{ij}}$. Clearly $d_j = 1$ if no processor writes into cell j at step t. Otherwise, $d_j = 0$. Also let $h_j = V_{i=1}^{1} w_{ij} v_i$. (The bitwise OR of binary strings is used.) Then, since we have the COMMON conflict resolution scheme, the contents of address j before step $t + 1$ is $h_j V d_j s_j$, where s_j is the contents of cell j before step t. This value can clearly be computed in constant depth and polynomial size.

Since the whole circuit consists of T layers, its depth is $O(T)$ and its size is $2^{O(q^T)}$. □

Hastad [13] has proved the following result:

LEMMA 2. *There exists an absolute constant n_0 such that there are no depth k PARITY circuits of size $2^{(1/10)^{k/(k-1)} n^{1/(k-1)}}$ for $n > n_0^k$.*

Combining our simulation with the above depth-size trade-off, we obtain the following:

THEOREM 3.2. *There exists an absolute constant b such that if a PRIORITY(∞) with $q > b$ processors computes PARITY of $n > b^T$ bits in time T, then $bT(bT + \log\log q) > \log n$.*

PROOF. Suppose that a PRIORITY(∞) with q processors computes PARITY of n bits in time T steps. Using Theorem 3.1, there is an unbounded fan-in circuit of depth cT and size q^{c2^T} which computes PARITY of n bits. By Lemma 2, there is a constant n_0 such that for all $n > n_0^{cT}$,

$$q^{c2^T} > 2^{(1/10)^{cT/(cT-1)} n^{1/(cT-1)}}.$$

Hence

$$q^{c2^T} > 2^{(1/10)^{cT/(cT-1)} n^{1/(cT-1)}}.$$

Hence

$$1 + 2^{cT}\log q > \left(\frac{1}{10}\right)^{cT/(cT-1)} n^{1/(cT-1)}.$$

For T, q large enough $2^{cT+1}\log q > 1 + 2^{cT}\log q$, hence

$$2^{cT+1}\log q > \left(\frac{1}{10}\right)^{cT/(cT-1)} n^{1/(cT-1)}.$$

Hence, $1 + cT + \log\log q + cT/(cT - 1)\log 10 > \log n/(cT - 1)$. For T large enough, $cT > 1 + (cT/(cT - 1))\log 10$ and $2cT > cT - 1$. Hence

$$2cT + \log\log q > \log n/2cT.$$

Without loss of generality, we may assume that both T and q are increasing functions of n. Let n_1 be such that for all $n > n_1$, T and q are large enough as required above. Let $b = \max(2c, n_0^c, n_1)$. □

Theorem 3.2 has many corollaries:

COROLLARY 1. *PRIORITY(∞) with a number of processors that is subexponential in n cannot compute PARITY of n bits in constant time.*

COROLLARY 2. *A PRIORITY(∞) with $n^{O(1)}$ processors requires $\Omega(\sqrt{\log n})$ time to compute PARITY of n bits.*

COROLLARY 3. *Unbounded fan-in circuits of polynomial size and constant depth compute the same Boolean functions as a PRIORITY(∞) with polynomially many processors and constant time.*

Corollaries 1 and 2 have also been obtained independently by Paul Beame [3] using a different method. Theorem 3.1 and Corollary 3 are of independent interest and serve as a general tool to translate circuit lower bounds into PRAM lower bounds.

Remark 1. From the reducibilities mentioned in [6] and [31] and our theorem it then follows that the same time-processor trade-off applies to SORTING (even of binary bit inputs), MAJORITY, CONNECTIVITY and many other functions (see [6] and [31]). Furthermore, our proof is much shorter in comparison with the $\Omega(\sqrt{\log n})$ lower bound obtained for SORTING in [22].

Remark 2. After the results in this paper were reported in [18] and this paper was submitted for publication, a recent paper by Beame and Hastad [4] improves the lower bound for PARITY by providing an optimal $\Omega(\log n/\log \log n)$ lower bound on the time needed by a PRIORITY(∞), with a number of processors polynomial in n, to compute PARITY of n bits. Hence this improved our Corollary 1(a), (c) in Section 2 and Corollary 2 in Section 3.

ACKNOWLEDGMENTS. We thank Eitan Gurari, Xin He, and Tim Long for their generous help, and the anonymous referees for very helpful suggestions.

REFERENCES

1. AJTAI, M. $\sum_1^1$-formulae on finite structures. *Ann. Pure Appl. Logic 24* (1983), 1–48.
2. AWERBUCH, B., AND SHILOACH, Y. New connectivity and MSF algorithms for ultracomputers and PRAM. In *Proceedings of the IEEE Conference on Parallel Processing*. IEEE, New York, 1983 pp. 175–179.
3. BEAME, P. Limits on the power of concurrent-write parallel machines. In *Proceedings of the 18th ACM Symposium on Theory of Computing* (Berkeley, Calif., May 28–30). ACM, New York 1986, pp. 169–176.
4. BEAME, P. Lower bounds in parallel machine computation. Ph.D. dissertation. Univ. of Toronto, Toronto, Ont., Canada, 1986.
5. BEAME, P., AND HASTAD, J. Optimal bounds for decision problems on the CRCW PRAM. In *Proceedings of the 19th ACM Symposium on Theory of Computing* (New York, N.Y., May 25–27). ACM, New York, 1987, pp. 83–93.
6. CHANDRA, A., STOCKMEYER, L., AND VISHKIN, U. Constant depth reducibility. *SIAM J. Comput. 13* (1984), 324–429.
7. COOK, S. A., AND DWORK, C. Bounds on the time for parallel RAM's to compute simple functions. In *Proceedings of the 14th ACM Symposium on Theory of Computing*. ACM, New York, 1982, pp. 231–233.
8. COOK, S. A., DWORK, C., AND REISCHUK, R. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput. 15*, 1 (1986), 87–97.
9. FICH, F., MEYER AUF DER HEIDE, F., RAGDE, P., AND WIGDERSON, A. One, two three, ..., infinity: Lower bounds for parallel computation. In *Proceedings of the 24th ACM Symposium on Theory of Computing* (Providence, R.I., May 6–8). ACM, New York, 1985, pp. 48–58.
10. FICH, F., RAGDE, P., AND WIGDERSON, A. Relations between concurrent-write models of parallel computation. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing* (Vancouver, B.C., Canada, Aug. 27–29). ACM, New Yrok, 1984, pp. 179–184.
11. FURST, M., SAXE, J., AND SIPSER, M. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theor. 17*, 1 (1984), 13–28.
12. GALIL, Z. Optimal parallel algorithms for string matching. In *Proceedings of the 16th ACM Symposium on Theory of Computing* (Washington, D.C., Apr. 30–May 2). ACM, New York, 1984, pp. 240–248.
13. HARTMANIS, J. Generalized Kolmogorov-complexity and the structure of feasible computations. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*. IEEE, New York, 1983, pp. 439–445.
14. HASTAD, J. Almost optimal lower bounds for small depth circuits. In *Proceedings of the 18th ACM Symposium on Theory of Computing* (Berkeley, Calif., May 28–30). ACM, New York, 1986, pp. 6–20.
15. HIRSCHBERG, D., CHANDRA, A., AND SARWATE, D. Computing connected components on parallel computers. *Commun. ACM 22*, 8 (Aug. 1979), 461–464.
16. KUCERA, L. Parallel computation and conflicts in memory access. *Inf. Proc. Lett. 14*, 2 (1982), 93–96.
17. LI, M., AND VITANYI, P. B. M. Tape versus queue and stacks: The lower bounds. *Inf. Comput. 78*, 1 (July 1988), 56–85.

18. LI, M., AND YESHA, Y. New lower bounds for parallel computation. In *Proceedings of the 18th ACM Annual Symposium on Theory of Computing* (Berkeley, Calif., May 28–30). ACM, New York, 1986, pp. 177–187.
19. LI, M., AND YESHA, Y. Separation and lower bounds for ROM and nondeterministic models of parallel computation. *Inf. Comput. 73*, 2 (May 1987), 102–128.
20. MAASS, W. Quadratic lower bounds for deterministic and nondeterministic one-tape Turing machines. *Trans. AMS 292*, 2 (Dec. 1985), 675–693.
21. MEYER AUF DER HEIDE, F., AND REISCHUK, R. On limits to speed up parallel machines by large hardware and unbounded communication. In *Proceedings of the 25th IEEE Symposium on Foundation of Computer Science.* IEEE, New York, 1984, pp. 56–64.
22. MEYER AUF DER HEIDE, F., AND WIGDERSON, A. The complexity of parallel sorting. In *Proceedings of the 26th IEEE Annual Symposium on Theory of Computing.* IEEE, New York, 1985, pp. 532–540.
23. PARBERRY, I. On the time required to sum *n* semi group elements on a parallel machine with simultaneous write. In *Proceedings of Aegean Workshop on Computation.* 1986.
24. PAUL, W. Kolmogorov complexity and lower bounds. In *Proceedings of the 2nd International Conference on Fundamentals of Computation Theory.* 1979.
25. PAUL, W., SEIFERAS, J., AND SIMON J. An information-theoretic approach to time bounds for on-line computations. *J. Comput. Syst. Sci. 23* (1981), 108–126.
26. PREPARATA, F. P. New parallel-sorting schemes. *IEEE Trans. Comput. C-27*, 669.
27. REIF, J. An optimal algorithm for integer sorting. In *Proceedings of the 26th IEEE Symposium on Foundation of Computer Science.* IEEE, New York, 1985, pp. 494–504.
28. REISCHUK, R. Parallel machines and their communication theoretical limits. In *Lecture Notes in Computer Science*, vol. 210. Springer-Verlag, New York, pp. 359–368.
29. SHILOACH, Y., AND VISHKIN, U. Finding the maximum, merging and sorting on parallel models of computation. *J. Algorithms 2* (1981), 88–102.
30. SHILOACH, Y., AND VISHKIN, U. An $O(\log n)$ parallel connectivity algorithm. *J. Algorithms 3* (1982), 57–67.
31. STOCKMEYER, L., AND VISHKIN, U. Simulation of random access machines by circuits. *SIAM J. Comput. 13* (1984), 409–422.
32. VISHKIN, U., AND WIGDERSON, A. Trade-offs between depth and width in parallel computation. *SIAM J. Comput. 14*, 2 (May 1985), 303–314.
33. YAO, A. C.-C. Separating the polynomial-time hierarchy by oracles. In *Proceedings of the IEEE 26th Symposium on Foundations of Computer Science* (Portland, Ore.). IEEE, New York, 1985, pp. 1–10.

RECEIVED AUGUST 1986; REVISED DECEMBER 1987 AND OCTOBER 1988; ACCEPTED NOVEMBER 1988