INFORMATION AND COMPUTATION **126**, 161–168 (1996)
ARTICLE NO. 0043

Sensitive Functions and Approximate Problems

Shiva Chaudhuri

Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany
E-mail: shiva@mpi-sb.mpg.de

Properties of functions that are good measures of the CRCW PRAM complexity of computing them are investigated. While the *block sensitivity* is known to be a good measure of the CREW PRAM complexity, no such measure is known for CRCW PRAMs. It is shown that the complexity of computing a function is related to its *everywhere sensitivity*, introduced by Vishkin and Wigderson. Specifically, the time required to compute a function $f: D^n \to R$ of everywhere sensitivity $es(f)$ with P processors and unbounded memory is $\Omega(\log[\log es(f)/(\log(|D| + 4P/es(f)))])$. This improves results of Azar and of Vishkin and Wigderson. This lower bound is used to derive new lower bounds for some *approximate problems*. These problems can often be solved faster than their exact counterparts and for many applications, it is sufficient to solve the approximate problem. It is shown that *approximate selection*, *approximate counting*, *approximate compaction*, and *padded sorting* all require time $\Omega(\log\log n)$ with a linear number of processors, if the level of accuracy desired is moderately high. For these levels of accuracy, no lower bounds were known for these problems on the PRAM model. The lower bounds for some of the problems are tight. © 1996 Academic Press, Inc.

1. INTRODUCTION

The computation of Boolean functions by circuits leads naturally to their study in all models of parallel computation. Much work has been done on investigating properties of Boolean functions which are measures of the difficulty of computing the function. One such measure is the *sensitivity* of a function. Let $f: \{0, 1\}^n \to \{0, 1\}$ be a Boolean function, let $x \in \{0, 1\}^n$ and let $x^{(r)} \in \{0, 1\}^n$ denote the bit vector that differs from x exactly on the rth co-ordinate. Then the sensitivity of f on x, written $s_f(x)$ is the number of distinct co-ordinates r, such that $f(x) \neq f(x^{(r)})$. The sensitivity of f, written s_f, is the maximum, over all inputs x, of $s_f(x)$. The sensitivity of Boolean functions has been extensively studied [Si83, Tu84, CDR86, Ni91]. Cook *et al.* [CDR86] show that the complexity of computing a function f on a CREW PRAM is related to the sensitivity of f. More precisely, they prove a lower bound of $\Omega(\log s_f)$ on the time required to compute f. Nisan considered a generalization of sensitivity, called the *block sensitivity* [Ni91]. He showed that the time required to compute a function f on a CREW PRAM is $\Theta(\log bs_f)$, where bs_f is the block of sensitivity f. Thus, complexity of computing Boolean functions on CREW PRAMs is well characterized.

The AND function has sensitivity n and therefore takes $\Theta(\log n)$ time on CREW PRAM. However, AND can be computed in constant time on a CRCW PRAM. Thus, sensitivity and block sensitivity are not appropriate measures of CRCW PRAM complexity. The reason is that both measures can critically depend on the value of the function on a single input. A CRCW PRAM can use its concurrent writing property to check, in a single step, if its input is special. Thus any measure whose value depends on a small number of inputs is doomed to failure.

A measure that avoids dependence on a small set of inputs is *everywhere sensitivity* defined as follows. Let D and R be finite sets and let $f: D^n \to R$ be a function. An input $x \in D^n$ is *q-sensitive* if for every subset $J \subseteq \{1, ..., n\}$, $|J| = q - 1$, there exists an input l, which agrees with x on the co-ordinates in J and for which $f(x) \neq f(l)$. The everywhere sensitivity of x is the largest integer q such that x is *q-sensitive*. The everywhere sensitivity of f is the *minimum*, over all inputs x, of the eveywhere sensitivity of x. An alternative way of thinking of the everywhere sensitivity of a function is the maximum number of co-ordinates whose values can be safely revealed without revealing the value of the function. Vishkin and Widgerson showed that a CRCW PRAM with m memory cells requires time $\Omega(\sqrt{q/m})$ to compute a function of everywhere sensitivity q [VW85]. For a special class of functions, Azar improved this bound to $\Omega(q/m)$ [Az92]. However, this does not yield nontrivial bounds for $m = \Omega(n)$.

We investigate the role of everywhere sensitivity in determining the CRCW PRAM complexity of a function. Our main result is that computing a function $f: D^n \to R$ of everywhere sensitivity $es(f)$ requires time $\Omega(\log[\log es(f)/(\log(|D| + 4P/es(f)))])$ on a CRCW PRAM with P processors and unbounded memory. For computing, with n processors, a Boolean function of everywhere sensitivity n, for instance, PARITY, this gives a lower bound of $\Omega(\log\log n)$. This is weaker than the bound of $\Omega(\log n/\log\log n)$ obtained by Beame and Håstad [BH89]. However, surprisingly, for n processors and everywhere sensitivity $\Omega(n)$, the bound is tight. Goldberg and Zwick give an $O(\log\log n)$ time algorithm that computes a function of everywhere sensitivity $\Omega(n)$ optimally [GZ94].

As applications of the above bound we derive new lower bounds for other problems. These problems have the common feature that they are all *approximate* versions of problems. For some applications, it is enough to solve the approximate version, which can often be solved faster than the exact version [HR92, HR93, CHR93, MV91]. For each approximate problem, there is an *accuracy parameter* $\lambda \geqslant 1/(n+1)$. In approximate selection, the task is to find, from n elements, an element whose rank differs from a specified rank by at most λn. In approximate counting, given a bit vector, the goal is to compute an integer that lies between $s/(\lambda+1)$ and $s(\lambda+1)$, where s is the number of 1's in the input vector. In *approximate compaction*, the goal is to place all the 1's in the input into the initial $s(\lambda+1)$ memory locations, at most one 1 to a memory cell. We prove the following bounds: approximate selection with accuracy $\lambda \leqslant 1/4$ with Cn processors requires $\Omega(\log[\log n/\log(C+2)])$ time. Approximate counting and approximate compaction with accuracy λ using Cn processors require $\Omega(\log[\log n/\log(C\lambda+2)])$ time. These bounds are easily seen to imply lower bounds for other appoximate problems such as *interval allocation* and *approximate prefix summation* [Hag93, HR93].

Padded sorting is the problem of placing n elements in sorted order in an output array of size at most $(\lambda+1)n$. The unused locations should contain a special *null* value. Our bounds for everywhere sensitive functions can be used to derive a lower bound of $\Omega(\log[\log n/\log((\lambda+2)(C+1))])$ for padded sorting with accuracy λ using Cn processors.

When $\lambda = 1/(n+1)$ each of the above problems reduces to its exact version, which is known to require $\Omega(\log n/\log\log n)$ time. It has been shown that if one can solve any of the approximate problems with an accuracy of λ using p processors in time t, then one can, with the same resources, solve a MAJORITY problem on $1/\lambda$ elements [Has]. The lower bound of Beame and Håstad then implies an $\Omega(\log(1/\lambda)/\log\log n)$ time lower bound for solving any of these problems with a polynomial number of processors. While this is good for small values of λ, it does not give a nontrivial bound when $\lambda = \Omega(1/(\log n)^c)$, for constant c. Our bounds improve the above for this range of λ, where, in fact, fast algorithms are known for each of the problems. In particular, for $\lambda = 1/4$, with n processors, algorithms for approximate selection, approximate counting and approximate compaction are known that run in time $\mathcal{O}(\log\log n)$[GZ94]. With n^2 processors padded sorting can be solved in $\mathcal{O}((\log\log n)^2)$ time [GZ94].

The methods used to prove the lower bounds are of independent interest. In Section 3, we prove some lemmas applicable to *any* PRAM algorithm. Roughly speaking these lemmas state that it is possible to bound the number of possible states of an algorithm by carefully setting a small number of inputs. This makes it possible to prove a lower bound for any problem merely by showing that if a small number of inputs is set a large number of output possibilities still remain. The method of bounding the number of states is general enough to have applications to other computational models.

2. PRELIMINARIES

2.1. *The Model*

We prove the lower bound on a strong model of the PRIORITY CRCW PRAM (see [JaJa92]). This model is sometimes referred to as the "Ideal" or "Full-Information" PRAM [Be89]. In this model each processor is assumed to keep track of the entire history of its own computation. Each processor has an *initial state*, and its state at step i is defined by its initial state and it history through step $i-1$. We consider deterministic algorithms, so the action of a processor is completely determined by its state. We make no other assumptions about the algorithm in particular, non-uniform algorithms are allowed.

We assume an infinite number of memory cells with infinite wordsize. Since there is no restriction on the wordsize, whenever a processor writes, it may as well write the entire history of its computation hence the name Full-Information PRAM. Lower bounds on this model depend crucially on limiting the amount of information that processors can communicate to each other through the shared memory. Lower bounds proved on this model carry over to more realistic models and give insight into the intrinsic difficulty of solving problems in parallel.

2.2. *Partial Inputs and the Computation Graph*

In the following, A will be an algorithm computing a function $f\colon D^n \to R^s$. For inputs of size n, let A use $P(n)$ processors and take $k(n)$ steps. We will use P and k for $P(n)$ and $k(n)$ respectively, from now on.

A *partial input* is an element of $(D \cup \{*\})^n$. For a partial input b, we denote by $X(b)$ the set of inputs consistent with b. That is, $X(b) = \{x \in D^n\colon i = 1, ..., n,\ b_i, \neq * \to b_i = x_i\}$. For partial inputs a and b, we say a is a refinement of b, and write $a \leqslant b$, if $X(a) \subseteq X(b)$.

For a given partial input b, consider a processor p at time t. The set of inputs consistent with b defines a set of states that p may be in, on inputs consistent with b. This set of states, in turn, defines the possible actions of p at time t; in particular, it defines the set of memory locations that p may read from, or write to. This gives us a way to identify (and consequently, limit) the amount of information that p may read from, or write to, the shared memory. We formalize this by modelling the computation of A on a graph. Let b be a partial input of size n. The *computation graph* of A on b, $G(b)$, is defined as follows:

$$V(G(b)) = \{(c, i)\colon c \textit{ is a memory cell and } 0 \leqslant i \leqslant k\}.$$

We have $(k+1)$ levels; in each level we have one vertex for each cell in the memory. The set of vertices in level i will be called V_i. The directed edges go from vertices at one level to the vertices of the next level. Every edge is labelled by a processor. $E(G(b))$ contains the edge $((c, i), (d, i+1))$ labelled p if on some input in $X(b)$, processor p reads cell c and writes to cell d in step $(i+1)$. (We assume that in each step of the computation, the processors first read and then write. Intuitively, the edges between levels $i-1$ and i represent the possible actions of the processors during step i.) Initially, variable i of the input is assumed to be in cell i; finally, output value i is assumed to be in cell i. We refer to vertex $(i, 0)$ as α_i for $1 \leqslant i \leqslant n$ (the *input* vertices), and vertex (i, k) as β for $1 \leqslant i \leqslant s$ (the *output* vertices).

Let $a \in D^n$. We shall associate with each vertex of $G(a)$ a content. The content associated with (c, i) is the content of the cell c after step i (that is, just before the write of step $(i+1)$ changes it) in the computation of A on the input a. We call this *content*$(a, (c, i))$. Similarly, for a processor p, *state*$(a, (p, i))$ is the state of processor p just before the write of step $(i+1)$ in the computation of A on input a. For a partial input b, let

$$\textit{contents}(b, (c, i)) = \{\textit{content}(x, (c, i)) \colon x \in X(b)\};$$
$$\textit{states}(b, (p, i)) = \{\textit{state}(x, (p, i)) \colon x \in X(b)\}.$$

We say that (c, i) is a *fixed* vertex if $|\textit{contents}(b, (c, i))| = 1$ and (p, i) is a *fixed* processor if $|\textit{states}(b, (p, i))| = 1$; otherwise, we say $(c, i)((p, i))$ is a *free* vertex (processor). Note that the above definitions depend on the algorithm A and the size of input n. These parameters will be clear from the context where they are used.

We model the computation of the algorithm A on the computation graph as follows. We say that a processor p reads from cell (c, i) and writes to cell $(d, i+1)$ when we mean that in the step $(i+1)$ of the computation of the algorithm A, p reads cell c and writes to cell d.

3. REGULARIZED COMPUTATION GRAPHS

Intuitively, if a cell is written to by a small number of processors, then it can only be affected be a small number of input variables, namely those that affect the processors that write to it. Similarly, it can only have a small number of contents, namely, the ones that each processor may write. Thus computation graphs in which no cell is written to by many processors are of special interest, which motivates the following definition.

Definition. Let S be a sequence of positive integers $(d_0, d_1, d_2, \ldots)$. For a partial input b, we say $G(b)$ is *S-regularized up to level j* if every free vertex in $G(b)$ at level i, $1 \leqslant i \leqslant j$ has indegree less than d_i. If $G(b)$ has k levels and is S-regularized up to level k, we simply say $G(b)$ is S-regularized. If $G(b)$ is S-regularized, then we call b an *S-regularizing* input.

The above definition implies that at level i, at most $d_i - 1$ processors may write to any free vertex. We would expect that this property ensures a bound (dependent on i) on the number of contents that a cell at level i may have. This is indeed true, as shown by the following lemma.

Lemma 3.1. *Let $S = (d_0, d_1, d_2, \ldots)$ be a sequence of positive integers. Define Y_i, Z_i, M_i and N_i, for $i \geqslant 0$ by $Y_0 = Z_0 = d_0$, $M_0 = N_0 = 1$ and*

$$Y_i = Y_{i-1} + (d_i - 1)\, Z_{i-1}, \qquad Z_i = Z_{i-1}\, Y_i;$$
$$M_i = M_{i-1} + (d_i - 1)\, N_{i-1}, \qquad N_i = N_{i-1} + M_i.$$

Let $G(b)$ be a computation graph, on partial input b for an algorithm which takes inputs from D^n, where $|D| \leqslant d_0$. Suppose G is S-regularized upto level j. Then, for each i, $1 \leqslant i \leqslant j$, for any memory cell c and processor p:

1. (a) $|\textit{contents}(b, (c, i))| \leqslant Y_i$.

(b) $|\textit{states}(b, (p, i))| \leqslant Z_i$.

2. *Let $x \in D^n$ be an input consistent with b. Then:*

(a) *c can be fixed to content$(x, (c, i))$ by setting the values of at most M_i additional input co-ordinates consistently with x.*

(b) *p can be fixed to state$(x, (p, i))$ by setting the values of at most N_i additional input co-ordinates consistently with x.*

Proof. The proof is by induction on i. For $i = 0$ we have the following. Initially, each cell has one of $|D|$ possible values written in it, and each processor, after the first read, can be in one of $|D|$ possible states. A cell can be fixed to any of its possible contents by setting one input co-ordinate and a processor to any of its possible states by setting the cell it reads.

As the inductive hypothesis, assume 1(a), 1(b), 2(a), and 2(b) hold for $i' \leqslant i-1$. Consider level $i \leqslant k$.

Consider a vertex $(c, i)(i > 0)$ in the graph $G(b)$. Let $d < d_i$ be the indegree of (c, i). Let $p_1, \ldots, p_d$ be the processors that label the d edges. Let the number of states in which processor p_j writes to (c, i) be S_j. The content of (c, i) is determined by the state of the processor that succeeds in writing to (c, i), or, if no processor writes to (c, i), by the content of $(c, i-1)$. Thus, we have

$$|\textit{contents}(b, (c, i))| \leqslant \sum_{k=1}^{d} S_k + |\textit{contents}(b, (c, i-1))|.$$

By the inductive hypothesis, $S_k \leqslant Z_{i-1}$ for each k and $|\textit{contents}(b, (c, i-1))| \leqslant Y_{i-1}$. Thus, for $i \geqslant 1$,

$$|\textit{contents}(b, (c, i))| \leqslant Y_i = Y_{i-1} + (d_i - 1)\, Z_{i-1}$$

The number of states of a processor after the ith read is at most the product of the number of states it had after the $(i-1)$th read and the number of contents of the cell it read at the ith read. Thus

$$|states(b, (p, i))| \leqslant Z_i = Z_{i-1} Y_i.$$

In the following, let x be some input consistent with b.

Since the computation graph is regularized, a vertex, c, at level i has at most $d_i - 1$ processors that can write to it. By the inductive hypothesis, each processor that can write to c at step i can be fixed to its state on input x, by fixing N_{i-1} input co-ordinates consistently with x. In case no processor writes to c at step i, the state of c at step $i-1$ can be fixed to be its state at step $i-1$ on input x, by fixing M_{i-1} inputs consistently with x. After this, the vertex at level i is fixed to its state on input x, and the number of co-ordinates set, J, satisfies

$$J \leqslant M_i = M_{i-1} + (d_i - 1)\, N_{i-1}.$$

By the inductive hypothesis, the state of a processor before the read of step i can be fixed to its state before the read of step i on input x by setting N_{i-1} co-ordinates consistently with x. After this, the cell that this processor reads at step i is determined, and its contents can be fixed to be its contents on input x by setting M_i co-ordinates consistently with x. Now, after the read of step i, the processor is in the state it would be in on input x. Thus, the number of co-ordinates set, K, satisfies

$$K \leqslant N_i = N_{i-1} + M_i.$$

This proves the lemma. ▮

3.1. *Making a Computation Graph Regularized*

In this section we show how to regularize a computation graph by setting a small number of inputs. The idea is to fix all the vertices at level i, for $i = 1, 2, ...,$ that have indegree at least d_i. When we have done this for levels 1 through $i-1$, the computation graph is S-regularized upto level $i-1$. Then, by Lemma 3.1, the number of input variables that can affect a processor at level i is small. Lemma 3.1 shows that by appropriately setting the variables that affect a processor, we can fix a processor to any desired state. Let p be the highest priority processor that can write to a cell c at level i. If we fix p to the state in which it writes to c, then, since all other processors that may write to c have a lower priority, c will always have the contents written by p, and will therefore be fixed.

In this fashion, we may fix every vertex at level i that has indegree at least i. In Lemma 3.2 we show that the total number of input variables set is small.

Lemma 3.2. *For any algorithm using P processors and taking inputs from D^n, and for any $m \geqslant 1$, there is a partial input, b^*, with at most P/m co-ordinates fixed, such that, after t steps*:

1. (a) *Each memory cell has one of at most g_t different contents consistent with b^*.*

(b) *Each processor is in one of at most g_t different states consistent with b^*.*

2. *Let $x \in D^n$ be an input consistent with b^*. Then*:

(a) *A cell can be fixed to have contents consitent with x at time t by setting the values of at most g_t additional input co-ordinates consistently with x.*

(b) *A processor can be fixed to have a state consistent with x at time t by setting the values of at most g_t additional input co-ordinates consistently with x.*

Here $g_t = a^{c^{t+1}}$, *where* $a = \max\{4, |D|, m\}$ *and* $c \geqslant (5+\sqrt{5})/2$.

Proof. Fix $c \geqslant (5+\sqrt{5})/2$, and define $d_0 = \max\{4, |D|, m\} = a$ and $d_i = a^{c^i}$. Let $S = (d_0, d_1, ...)$ and define Y_i, Z_i, M_i and N_i as in Lemma 3.1. We will find an S-regularizing input with at most P/m co-ordinates set. We describe a simple procedure to find such a S-regularizing input. Our strategy is to proceed level by level, refining the current partial input at each level. When we are finished with level i, the current partial input will be such that at levels $j \leqslant i$, all vertices of indegree $\geqslant d_j$ will be fixed.

Suppose we have finished with levels 1, ..., $i-1$, and are currently at level i. Let b denote the current partial input. Consider the computation graph on b and let (c, i) be a free vertex of indegree $\geqslant d_i$. Let p be the highest priority processor that could write to (c, i). Then $\exists x \in D^n$, $x \leqslant b$, an input on which p writes to (c, i). Use Lemma 3.1 2(b) to fix this processor to $state(x, (p, i-1))$, setting at most N_{i-1} additional inputs. If b' is the partial input so obtained, on all inputs consistent with b', p will write the same value to (c, i), so this vertex is fixed. We set the current input to be b' and repeat the process.

Since we are continuously refining the input, the degree of a vertex cannot increase. Thus, the procedure will eventually fix all the vertices in level i with indegree $\geqslant d_i$.

It remains to bound the number of input variables set. When we are at level i, the current input is such that all free vertices at levels $j < i$ have indegree $\leqslant d_j - 1$. By Lemma 3.1 1(b), each processor writing to a cell at level i may be in at most Z_{i-1} states, and hence may contribute at most this many edges to the graph. Thus, the number of edges between levels $i-1$ and i is at most $Z_{i-1}P$, implying that the number of vertices with indegree $\geqslant d_i$ is at most $Z_{i-1}P/d_i$. To fix each such vertex, we set at most N_{i-1} variables. At level i, therefore, at most $N_{i-1}Z_{i-1}P/d_i$ input variables are set to values in D. We will show that this is at most $P/m2^i$. Summing for all i yields the bound on the total number of variables set.

We now show that $N_i Z_i/d_{i+1} \leqslant 1/m2^{i+1}$, for each $i \geqslant 0$. From the recurrences defining Y_i, Z_i, M_i and N_i, it is easy to derive the following inequalities:

$$Y_i \leqslant Z_i \ , \quad Z_i \leqslant \prod_{j=0}^{i} a^{c^j 2^{i-j}};$$
$$M_i \leqslant N_i \ , \quad N_i \leqslant 2^i \prod_{j=0}^{i} a^{c^j}.$$

Taking logarithms, showing that $N_i Z_i/d_{i+1} \leqslant 1/m2^{i+1}$ is equivalent to showing

$$\left(i + \log a \left[\sum_{j=0}^{i} (c^j 2^{i-j} + c^j) - c^{i+1} \right] \leqslant -\log m - (i+1).\right.$$

Rearranging, we have

$$\log a \left[\frac{c^{i+1} - 2^{i+1}}{c-2} + \frac{c^{i+1}-1}{c-1} - c^{i+1} \right] \leqslant -\log m - (2i+1)$$

or

$$\log a \left[c^{i+1} \left(\frac{1}{c-2} + \frac{1}{c-1} - 1 \right) - \frac{2^{i+1}}{c-2} - \frac{1}{c-1} \right]$$
$$\leqslant -\log m - (2i+1).$$

Since $a \geqslant m$ and $a \geqslant 4$, it can be verified that this inequality holds if $1/(c-2) + 1/(c-1) - 1 \leqslant 0$, i.e. if $c^2 - 5c + 5 \geqslant 0$. The latter function has roots $(5 \pm \sqrt{5})/2$, and is positive for $c > (5 + \sqrt{5})/2$, in particular, the inequality holds for the chosen value of c.

It can be shown by induction that $Z_i \leqslant a^{c^{i+1}}$; now, 1(a), 1(b), 2(a), and 2(b) follow from Lemma 3.1. ∎

4. EVERYWHERE SENSITIVE AND ELUSIVE BOOLEAN FUNCTIONS

We give an alternate, but equivalent definition of everywhere sensitivity.

Recall from Section 2.2 that a partial input $b \in D \cup \{*\}^n$ and $X(b)$ is the set of all inputs consistent with b. Define the *length* of a partial input, written $|b|$, to be the number of values in it that are not $*$'s. For a function $f: D^n \to R$, and a partial input b, let $R(b)$ denote the set of possible output values on inputs in $X(b)$. That is, $R(b) = \{r: r \in R$ and $\exists x \in X(b)$ such that $f(x) = r\}$.

Then we define the everywhere sensitivity of f to be $\max\{k: \forall$ partial inputs $b, |b| \leqslant k \Rightarrow |R(b)| > 1\}$. It may be verified that this definition is equivalent to the one in Section 1. Thus we may view the everywhere sensitivity of a function as the maximum number of input variables that an adversary may reveal, without revealing the value of the function. This is precisely the view that we will use in our proofs. We now obtain a lower bound through the following simple argument.

THEOREM 4.1. *Let $f: D^n \to R$ be a function with everywhere sensitivity $es(f)$. Let*

$$k = \left\lceil \frac{1}{\log c} \log \left(\frac{\log es(f) - 1}{\log(|D| + 4P/es(f))} \right) \right\rceil - 2,$$

where $c = (5 + \sqrt{5})/2$. Then, any CRCW PRAM algorithm computing f with P processors requires k steps.

Proof. Assume that $P \geqslant n$. The function cannot be computed faster by using less processors, hence the lower bound for $P = n$ also holds for $P < n$.

Choose $m = 4P/es(f)$ so that $P/m \leqslant es(f)/4$; note that $m \geqslant 4$. Let $a = \max\{|D|, m\}$. Define $c = (5 + \sqrt{5})/2$. The choice of k in the theorem ensures that $a^{c^{k+1}} < es(f)/2$.

Suppose there is an algorithm that computes the function in less than k steps. Consider the computation graph of the algorithm. By Lemma 3.2 there is an with at most P/m variables set to values in D such that the conclusions of the lenama hold. Let b be this partial input and consider the output cell of the computation graph, (β, k). Choose any input $x \in D^n$, $x \leqslant b$. Let b' be the input obtained from b by fixing (β, k) to $content(x, (\beta, k))$, setting at most $a^{c^{k+1}}$ additional input co-ordinates as promised by the lemma. On any input $x' \in D^n$, $x' \leqslant b'$, the algorithm outputs the same value, that is, $content(x, (\beta, i))$.

However, since at most $P/m + d_k^2 \leqslant es(f)/4 + es(f)/2 < es(f)$ variables in b' are set to values in D, there must be two inputs consistent with b' on which the function has different values. This gives us a contradiction. ∎

We now introduce a measure that allows us to quantify the complexity of a function more accurately. Everywhere sensitivity is more robust than sensitivity in that its value is unaffected by small numbers of inputs. However, it errs in the other direction, that is, it is often insensitive to large numbers of variables. Consider the Boolean function $f(x_1, ..., x_n) = x_1 \vee (\bar{x}_1 \wedge PARITY(x_2, ..., x_n))$. It is easy to see that $es(f) = 0$. Clearly this function has a low everywhere sensitivity, but is hard to compute. This motivates our definition of another measure.

For a partial input b and a function $f: D^n \to R, f|_b$ is the function obtained by replacing input variable x_i with the value assigned to it by b, where a value $*$ indicates that the variable may assume any value in D.

The *elusiveness* of a function f, written $E(f)$ is $\max\{es(f|_b): b$ is a partial input$\}$. Clearly, $E(f) \geqslant es(f)$. On the other hand, the difference between the two may be arbitrarily large. This is demonstrated by the function above, which has everywhere sensitivity 0 and elusiveness $n-2$.

We may now strengthen the above theorem as follows. If f is a function of elusiveness $E(f)$, then there is a restriction σ such that $es(f|\sigma) = E(f)$. Applying Theorem 4.1 now yields:

THEOREM 4.2. *Let $f\colon D \to R$ be a function. Let*

$$k = \left\lceil \frac{1}{\log c} \log\left(\frac{\log E(f) - 1}{\log(|D| + 4P/E(f))}\right)\right\rceil - 2,$$

where $c = (5 + \sqrt{5})/2$. Then, any CRCW PRAM algorithm computing f with P processors requires k steps.

5. APPLICATIONS

We use our bounds for everywhere sensitive functions to prove lower bounds for the following problems.

Approximate Selection: Given n elements from an ordered universe, an integer $r \in \{1, ..., n\}$ and an accuracy parameter $\lambda \geqslant 1/(n+1)$, find an element with rank between $r - \lambda n$ and $r + \lambda n$.

Approximate Counting: Given an input from $\{0, 1\}^n$, and an accuracy parameter $\lambda \geqslant 1/(n+1)$ compute an integer b, $s/(\lambda+1) \leqslant b \leqslant s(\lambda+1)$, where s is the number of 1's in the input.

Approximate Compaction: Given an input from $\{0, 1\}^n$, and an accuracy parameter $\lambda \geqslant 1/(n+1)$ compute $b_1, ..., b_n$ such that if the input had k 1's, then for some $S \subseteq \{1, ..., \min\{(1+\lambda)k, n\}\}$, with $|S| = k$, $b_j = 1 \Leftrightarrow j \in S$. (*In other words, all the* 1's *in the input are compacted into k distinct locations among the first*$(1+\lambda)$ *k locations.*) A lower bound for this problem was first proved in [Cha93].

Padded Sorting: Given n elements from an ordered universe, and an array A of size $\lfloor(\lambda+1)n\rfloor$, place the input elements in the array A, in any order such that, if $a_i < a_j$ are two input elements, then a_i is placed to the left of a_j in the array. The unused locations should contain a special *null* value.

THEOREM 5.1. *Approximate selection problems with accuracy $\lambda \leqslant 1/4$ using Cn processors requires time*

$$\Omega\left(\log \frac{\log n}{\log(C+2)}\right).$$

Proof. We assume that $C \geqslant 2$ and $\lambda = 1/4$; clearly, the problem cannot be solved faster by using less processors or by solving for smaller λ.

In order to prove a lower bound, we consider a restricted version of the problem where each element has a value in $\{0, 1\}$ and the problem is to approximately select the median. Any deterministic algorithm that solves this problem with accuracy λ can be viewed as computing a Boolean function $f\colon \{0, 1\}^n \to \{0, 1\}$.

We will show that the function f has everywhere sensetivity at least $n/2 - \lambda n - 1$. Consider any partial input b of length less than $n/2 - \lambda n$. Let b_0 and b_1 be elements of $X(b)$ obtained by setting all the $*$'s in b to 0 and 1 respectively. Clearly, $f(b_0) = 0$ and $f(b_1) = 1$. Thus $es(f) \geqslant n/2 - \lambda n - 1 \geqslant (n/2)(1/2 - \lambda)$, when $n \geqslant 4$. Using Theorem 4.1 with $P = Cn$ and $\lambda = 1/4$ gives us the claimed bound. ∎

THEOREM 5.2. *Approximate counting with accuracy λ using Cn processors requires time*

$$\Omega\left(\log \frac{\log n}{\log(C\lambda+2)}\right).$$

Proof. We prove the bound for $\lambda \geqslant 1$; clearly, the problem cannot be solved faster by solving for a smaller λ.

Any deterministic algoritm that solves the approximate counting problem can be viewed as computing a function $g\colon \{0, 1\}^n \to \{0, ..., n\}$, where, for $x \in \{0, 1\}^n$, $g(x)$ is the value output by the algorithm. Note that if x has s 1's, $s/2\lambda \leqslant g(x) \leqslant 2s\lambda$, since $\lambda \geqslant 1$.

We will show that the function g has everywhere sensitivity at least $n/6\lambda^2$. Consider any partial input b, of length less than n/a, for some real a, whose value will be determined later. Let $b_0 \in X(b)$ be the partial input obtained from b by setting all the $*$'s in b to 0, and let $b_i \in X(b)$ be the input obtained by setting all the $*$'s to 1. Since b_0 has at most n/a 1's and b_1 has at least $n - n/a$ 1's, we have $g(b_0) \leqslant n(\lambda+1)/a$ and $n(1-1/a)/(\lambda+1) \leqslant g(b_1)$. For $a > (\lambda+1)^2 + 1$, we have $n(\lambda+1)/a < n(1-1/a)/(\lambda+1)$, hence, in this range of values for a, $g(b_0) \neq g(b_1)$. Setting $a = 6\lambda^2 > (\lambda+1)^2 + 1$ (since $\lambda \geqslant 1$), we find $es(g) \geqslant n/6\lambda^2$.

Applying Theorem 4.1 with $P = Cn$ yields the stated bound. ∎

THEOREM 5.3 [Cha93]. *Approximate compaction with accuracy λ using Cn processors requires time*

$$\Omega\left(\log \frac{\log n}{\log(C\lambda+2)}\right).$$

Proof. We reduce approximate counting to approximate compaction. Let $a_1, ..., a_n$ be an instance of approximate counting with accuracy λ. First solve the approximate compaction problem with accuracy λ on this input. Let $b_1, ..., b_n$ be the output produced. Then, find the index, b, of the rightmost 1 in the output, which can be done in constant time [FRW88]. Then, the solution to the approximate counting problem is b. This is because, if the input had k 1's, then the index of the rightmost 1 in the output is at least k and at most $(\lambda+1)k$. ∎

As approximate compaction has applications to various important problems, it has been studied in great detail. A particularly important case is the range of λ for which the problem can be solved in constant time. Radge showed that for $\lambda = k^3$, λ-Approximate Compaction can be solved in constant time with n processors [Rag90]. Hagerup extended this by giving a constant time algorithm for $\lambda = k^{\varepsilon}$ for any $\varepsilon > 0$, where the constant depends upon ε [Hag92]. Below, we show that when $\varepsilon < 1$ and $\lambda = n^{\varepsilon}$, the time required is $\Omega(\log 1/\varepsilon)$, for sufficiently large n (note that since $k \leqslant n$, this automatically implies a lower bound for $\lambda = k^{\varepsilon}$). Thus as $\varepsilon \to 0$, the time required grows to infinity. The following is obtained by simply substituting the value of λ in the bound for approximate compaction.

COROLLARY 5.1. *For $\varepsilon \to 0$ and $\lambda = n^{\varepsilon}$, approximate compaction takes time $\Omega(\log(1/\varepsilon))$.*

THEOREM 5.4. *Padded sorting with accuracy λ with Cn processors requires*

$$\Omega\left(\log \frac{\log n}{\log((\lambda+2)(c+1))}\right)$$

time. (*See Acknowledgments*).

Proof. Let A be the output array into which the elements are placed; $|A| = \lfloor n(\lambda+1) \rfloor$. We prove the lower bound for Cn processors, where $Cn \geqslant |A|$; clearly, the problem cannot be solved faster by using less processors.

Write $d = \lceil \lambda \rceil + 2$ and consider the action of the padded sorting algorithm on inputs from $D^n = \{1, ..., d\}^n$. On input $x \in D^n$, let y_i be the minimum index in A where the value i appears, for $1 \leqslant i \leqslant d$. If i does not appear in x, then y, is 0. Define $F\colon D^n \to \{0, 1, ..., |A|\}^d$, by $F(x) = \langle y_1, y_2, ..., y_d \rangle$.

Now, F must have everywhere sensitivity at least $n - \lfloor |A|/d \rfloor$. If not, there is a partial input b, of length less than $n - \lfloor |A|/d \rfloor$, such that $\forall x \in X(b)$, $F(x) = \langle y_1, ..., y_d \rangle$, where $\langle y_1, ..., y_d \rangle \in \{0, ..., |A|\}^d$. First, note that since b has at least one $*$, $y_i \neq 0$ for each $i = 1, ..., d$ since there is an input in $X(b)$ containing the value i. Since the output is in sorted order, $y_1 < y_2 < \cdots < y_d$. Let $y_{d+1} = |A| + 1$. Then $\exists i \in D$ such that $y_{i+1} - y_i \leqslant \cdots \lfloor |A|/d \rfloor$. Consider the input x' obtained from b by setting all the $*$'s to the value i. Then x' has at least $\lfloor |A|/d \rfloor + 1$ elements with value i which must all be placed between indices y_1 and y_{i+1}, that is in at most $\lfloor |A|/d \rfloor$ output positions, which is impossible.

We show that F can be computed in time $T + \log d + 1$, where T is the time taken by the padded sorting algorithm. Let B be an array of size d. Modify the padded sorting algorithm so that after the output is written into A, processor p_i reads $A(i)$, for each i. If it reads the value $j \neq 0$, it writes i into $B(j)$ If it reads a null value, it does not write. After this step, $B(i)$ contains the minimum index of A in which i appears, for $i = 1, ..., d$ (this is because the PRIORITY rule automatically selects the processor of minimum index that writes to $B(i)$). F can now be computed in $\log d$ further steps in by encoding all the information in B into half the number of cells at each step.

Observe that $n - \lfloor |A|/d \rfloor \geqslant n - n(\lambda+1)/(\lambda+2) = n/(\lambda+2)$. Applying Theorem 4.1 now gives

$$T + \log d + 1 \geqslant \Omega \log \left[\frac{\log(n/(\lambda+2))}{\log[(\lambda+2) + 4C(\lambda+2)]} \right]\Bigg).$$

The stated bound follows from simple estimations. ∎

6. DISCUSSION

The technique used to prove the lower bounds in this paper is a *restriction* technique. We restrict the set of inputs under consideration by setting the values of some of the input co-ordinates. This technique has been used in the past to prove lower bounds [FSS88, BH89, Mac92]. The method of restricting is typically to set the values of a random subset of the inputs. Here, we determine which co-ordinates to set based on the algorithm, in a manner similar to [Mac92]. The method of random restrictions [FSS88, BH89] yields the following results for PRAMs. A memory cell in a PRAM algorithm that uses P processors can be fixed by setting the values of $n - n/\log(P)$ input co-ordinates. Thus, random restrictions cannot be used to give lower bounds for functions with everywhere sensitivity smaller than, say, $n - \sqrt{n}$. On the other hand, the technique can be applied for any sub-exponential value of P. In contrast, our method fixes a cell by fixing $p^{1-\varepsilon}$ input co-ordinates for some ε, $0 < \varepsilon < 1$. If P is close to linear, this method can be used for functions with everywhere sensitivity as low as $n^{1-\varepsilon}$. On the other hand, as soon as P becomes larger than $n^{1+\delta}$, for some δ, the method yields only trivial bounds. Thus, the first method appears to give lower bounds that deteriorate slowly as the number of processors grows, but rapidly as the everywhere sensitivity of the function decreases. Our method gives bounds that deteriorate rapidly as the number of processors increases, but slowly as the everywhere sensitivity decreases. It would be interesting to combine the two methods to get bounds that are robust with respect to both parameters.

The method of bounding the number of states achievable by a processor or a memory cell has been used before in [BeSS, LY89, Bel89]. While we set input co-ordinates to limit the number of achievable states, there is also a natural limit, even when no input co-ordinates are set, determined by the number of processors the algorithm uses. In particular, analogously to Lemma 3.1, if we use $\hat{Y}_i$ and $\hat{Z}_i$ to denote the number of contents and states of a memory cell and processor respectively, then the corresponding recurrences would be

$$\hat{Y}_i \leqslant \hat{Y}_{i-1} + P\hat{Z}_{i-1}\ , \quad \hat{Z}_i \leqslant \hat{Z}_{i-1}\hat{Y}_i,$$

where P is the number of processors used by the algorithm. This is the approach implicit in [Be88, LY89], which yields $\hat{Z}_i \leqslant P^{2^i + O(1)}$, Our method yields $Z_i \leqslant m^{4^i}$. Beame [Be88] obtains a lower bound of $\log n - \log \log P$ for the problem of writing an encoding of n bits into one cell. The n bits are initially written, one bit to a cell, in n cells. The upper bound is $\log n - \log \log(P/n)$; i.e. the second term is smaller. Using our method yields a lower bound of $0.5 \log n - \log \log(P/n)$, where the second term has a similar form; however, the first term is weaker by a factor of 1/2.

It is interesting to note that these bounds do not apply to randomized algorithms. In fact, approximate counting and approximate compaction can be solved in time $O(\log^* n)$, using randomization [MV91], and this is the best possible for randomized algorithms, as proved in [Mac92]. Thus, this is one of the instances where randomization yields provably better algorithms.

ACKNOWLEDGMENTS

The proof of Theorem 5.4 is joint work with Torben Hagerup and Holger Bast. We thank Jaikumar Radhakrishnan for pointing out the application to Corollary 5.1. We especially thank Referee B for an exceptionally insightful referee report; the paper and the author, have benefited immeasurably from the comments therein. In particular, almost all the remarks in Section 6 are due to Referee B.

A preliminary version of this paper appeared in [C93].

Received August 31, 1993; final manuscript received February 21, 1996

REFERENCES

[Az92] Azar, I. (1992), Lower bounds for threshold and symmetric functions in parallel computation, *SIAM J. Comput*. **21**, No. 2, 329–338.

[Be88] Beame, P. (1988), Limits on the power of concurrent write parallel machines, *Inform. and Comput*. **76**, 13–28.

[Be89] Beame, P. (1989), "Lower Bounds in Parallel Machine Computation," Ph.D. thesis, Univ. of Toronto.

[BH89] Beame, P., and Håstad, J. T. (1989), Optimal bounds for decision problems on the CRCW PRAM, *J. Assoc. Comput. Mach*. **36**, 643–670.

[Bel89] Bellantoni, S. (1989), Parallel RAMs with bounded memory wordsize, *in* "Proc. of 1st ACM sypposium on Par. Alg. and Arch.," pp. 83–91.

[Cha93] Chaudhuri, S. (1993), A lower bound for linear appoximate compaction, *in* "Proc. 2nd Israel Symp. on Theory of Comp. and Sys.," pp. 25–32.

[C93] Chaudhuri, S. (1993), Sensitive functions and approximate problems, *in* "Proc. of 34th IEEE FOCS," pp. 186–193.

[CHR93] Chaudhuri, S., Hagerup, T., and Raman, R. (1993), Approximate and exact deterministic parallel selection, *in* "Proc. 18th Math. Fdtns. of Comp. Sci.," Lecture Notes in Computer Science, Vol. 711, pp. 352–361, Springer-Verlag, Berlin/New York.

[CDR86] Cook, S., Dwork, C., and Reischuk, R. (1996), Upper and lower time bounds for parallel random access machines without simultaneous writes, *SIAM J. Comput*. **15**, No. 1, 87–97.

[FRW88] Fich, F. E., Wigderson, A., and Ragde, P. (1988), Simulations among concurrent-write models of parallel computation, *Algorithmica* **3**, 43–51.

[FSS88] Furst, M., Saxe, J. B., and Sipser, M. (1988), Parity, circuits, and the polynomial time hierarchy, *Math. Systems Theory* **17**, 13–27.

[GZ94] Goldberg, T., and Zwick, U. (1995), Optimal deterministic approximate parallel prefix sums and their applications, *in* "Proc. Israel Symp. on Theory and Computing Systems (ISTCS '95)," pp. 220–228.

[Hag92] Hagerup, T. (1992), On a compaction theorem of Ragde, *Inform. Process. Lett*. **43**, 335–340.

[Hag93] Hagerup, T. (1993), Fast deterministic processor allocation, *in* "Proc. 4th ACM–SIAM SODA," pp. 1–10.

[HR92] Hagerup, T., and Raman, R. (1992), Waste makes haste: Tight bounds for loose parallel sorting, *in* "Proc. 33rd IEEE FOCS," pp. 628–637.

[HR92] Hagerup, T., and Raman, R. (1993), Fast approximate and exact parallel sorting, *in* "Proc. 5th Annual SPAA," pp. 346–355.

[Has] Håstad, J. (1992), Personal communication.

[JaJa92] JáJá, J., "An Introduction to Parallel Algorithms," Addison–Wesley, Reading, MA.

[LY89] Li, M., and Yesha, Y. (1989), New lower bounds for parallel computation, *J. Assoc. Comput. Mach*. **36**, No. 3, 671–680.

[Mac92] MacKenzie, M. D. (1992), Load balancing requires $\Omega(\log^* n)$ expected time, *in* "Proc. 3rd ACM–SIAM SODA," pp. 94–99.

[MV91] Matias, Y., and Vishkin, U. (1991), Converting high probability into nearly-constant time with applications to parallel hashing, *in* "Proc. 23rd Annual STOC," pp. 307–316.

[Ni91] Nisan, N. (1991), CREW PRAMs and decision trees, *SIAM J. Comput*. **20**.

[Rag90] Ragde, P. (1990), The parallel simplicity of compaction and chaining, *in* "Proc. 17th ICALP 1990," Lecture Notes in Computer Science, Vol. 443, pp. 744–751, Springer-Verlag, Berlin/New York.

[Si83] Simon, H.-U. (1983), A tight $\Omega(\log \log n)$ bound on the time for parallel RAM's to compute nondegenerated Boolean functions, *in* "Foundations of Computing Theory" (M. Karpinski, Ed.), Lecture Notes in Comput. Sci., Vol. 158, pp. 439–444, Springer, Berlin.

[TU84] Turan, G. (1984), The critical complexity of graph properties, *Inform. Process. Lett*. **18**, 151–153.

[VW85] Vishkin, U., and Wigderson, A. (1985), Trade-offs between depth and width in parallel computations, *SIAM J. Comput*. **14**, 303–314.