Conf-951069--1

SAND95-1081C

A Lower Bound for the QRQW PRAM

Philip D. MacKenzie*

May 2, 1995

Abstract

The queue-read, queue-write (QRQW) parallel random access machine (PRAM) model is a shared memory model which allows concurrent reading and writing with a time cost proportional to the contention. This is designed to model currently available parallel machines more accurately than either the CRCW PRAM or EREW PRAM models. Many algorithmic results have been developed for the QRQW PRAM. However, the only lower bound results have been fairly simple reductions from lower bounds for other models, such as the EREW PRAM or the "few-write" CREW PRAM.

Here we present a lower bound specific to the QRQW PRAM. This lower bound is on the problem of Linear Approximate Compaction (LAC), whose input consists of at most m marked items in an array of size n, and whose output consists of the m marked items in an array of size $O(m)$. There is an $O(\sqrt{\log n})$ expected time randomized algorithm for LAC on the QRQW PRAM. We prove a lower bound of $\Omega(\log\log\log n)$ expected time for any randomized algorithm for LAC. This bound applies regardless of the number of processors and memory cells of the QRQW PRAM. The previous best lower bound was $\Omega(\log^* n)$ time, taken from the known lower bound for LAC on the CRCW PRAM.

1 Introduction

The PRAM model of computation has been the most widely used model for the design and analysis of parallel algorithms. The PRAM is a model of parallel computation in which processors communicate by performing synchronous unit time reads and writes to shared memory. For the most part, it has been assumed that contention for reading and writing to memory locations should be handled either by the "exclusive" rule (there can be no concurrent access to a memory location) or by the "concurrent" rule (concurrent accesses are allowed and can be processed in unit time). In the case of the concurrent rule, different conflict resolution protocols exist to handle concurrent writes.

As argued in Gibbons, Matias, and Ramachandran [5], neither the exclusive or concurrent rules accurately model current parallel architectures. They propose the "queue" rule as a more realistic alternative. In the queue rule, concurrent accesses are allowed, but take time proportional to the number of concurrent accesses. The machines which have contention properties well-approximated by the queue rule include the CRAY T3D, IBM SP2, Intel Paragaon, Thinking Machines CM-5 (data network), and others. We refer the interested reader to [5] for a more complete list of machines and measurements which justify their approximation by the queue rule.

*This work was performed at Sandia National Laboratories and was supported by the U.S. Department of Energy under contract DE-AC04-76DP00789. Address: Sandia National Laboratories, Albuquerque, NM 87185-1110. Email: philmac@cs.sandia.gov

The model proposed in [5] is the QRQW PRAM model. In that paper and in [4], they explore this model, developing many deterministic and randomized algorithms for standard problems. They also are able to give two lower bounds for the QRQW PRAM. One is for computing the OR of n bits. They show using a reduction from the lower bound for computing the OR function on the "few-write" CREW PRAM model in Dietzfelbinger, Kutylowski, and Reischuk [2] that computing the OR function on the QRQW PRAM requires $\Omega(\log n/\log\log n)$ time. (This provides a separation between the QRQW and CRCW PRAM.) The other lower bound is for broadcasting a bit to n processors. They show using a reduction from the lower bound on broadcasting a bit on an EREW PRAM model in Beame, Kik, and Kutylowski [1] that broadcasting a bit on the QRQW PRAM requires $\Omega(\log n)$ time. They extend this to an $\Omega(\log n)$ lower bound on the expected time for any randomized algorithm to broadcast a bit. A corollary to this is that Load Balancing requires $\Omega(\log n)$ expected time, assuming one processor has load n, and the other processors have no load [4].

Another problem considered in [5] is that of linear approximate compaction (LAC). In the LAC problem, an array of size n with at most m marked elements is given, and the marked elements must be moved into an array of size $O(m)$. LAC was a major part of the automatic processor allocation routines given in [5] for many general types of algorithms, and consequently, time improvements in LAC would result in time improvements for many algorithms. A randomized algorithm solving LAC in $O(\sqrt{\log n})$ expected time is given in [5], but no lower bound is given. Thus the only lower bound known for this problem was the $\Omega(\log^* n)$ expected time lower bound for LAC on the CRCW PRAM given in MacKenzie [6]. In this paper we give an $\Omega(\log\log\log n)$ expected time lower bound. This is the first lower bound specifically designed for the QRQW PRAM.

To prove the lower bound, we use the Random Adversary Technique, first developed in MacKenzie [6], combined with QRQW PRAM specific ideas which to our knowledge have not been used before. Specifically, we use the Random Adversary to restrict the inputs so as to force the time required for memory accesses to be either more than the desired lower bound or as high as the total possible contention. In this way, we are able to charge the algorithm for the amount of information it tries to transmit at a given read or write step.

We feel that even though the lower bound we prove does not seem to be tight, the ideas used should be useful for proving other lower bounds on the QRQW PRAM, and thus furthering our understanding of the role of contention in parallel computation.

In Section 2 we give explicit definitions of the models and problems we study. In Section 3, we review the Random Adversary technique. In Section 4, we give the general form of the QRQW PRAM lower bound. In Section 5 we give an application of this general lower bound to the problem of Linear Approximate Compaction.

2 Definitions

Our definition for the QRQW PRAM is taken from [5].

Consider a single step of a PRAM, consisting of a read substep, a compute substep, and a write substep. The *maximum contention* of the step is the maximum, over all memory cells x of the number of processors reading x or the number of processors writing x.

The QRQW PRAM model consists of a number of processors, each with its own private memory, communicating by reading and writing to locations in a shared memory. Processors execute steps synchronously. Each step consists of the following three substeps:

1. Read substep: Each processor i eads r_i shared memory locations, where the locations are known at the beginning of the substep.

2. Compute substep: Each processor i performs c_i RAM operations involving only its private state and private memory.
3. Write substep: Each processor i writes to w_i shared memory locations, where the locations and values written are known at the beginning of the substep.

Concurrent reads and writes to the same location are permitted in a step. In the case of multiple writes to a location x, an arbitrary write to x succeeds in writing the value present in x at the end of the step.

If a QRQW PRAM step has maximum contention k, and if $m = \max_i\{r_i, c_i, w_i\}$ for this step, then the time cost of the step is $\max\{m, k\}$. The *time* of a QRQW PRAM algorithm is the sum of the time costs for its steps. The *work* of a QRQW PRAM algorithm is its processor-time product.

Our lower bound proof does not need to consider c_i, so local computations are "free" for any algorithm. We will assume that there are separate steps for reading and writing. This could only increase the time of an algorithm by a factor of 2, and thus our asymptotic lower bounds will still hold in the general case.

We give a formal definition of the LAC problem here.

Linear Approximate Compaction Given an array of n cells with at most h containing one item each and all others being empty, insert the items into an array of size $O(h)$.

3 Random Adversary Technique

The Random Adversary Technique allows one to prove a lower bound on the time required for a parallel randomized algorithm to solve a given problem. The first step of the technique is to decide on an input distribution for the problem. By Yao's Theorem (see below), a lower bound on deterministic algorithms over this distribution provides the same lower bound for randomized algorithms.

The next step is to create a Random Adversary that proceeds through the given deterministic algorithm step by step, fixing some of the inputs in order to ensure some desired properties. (As shown below, this entails filling in the details of a procedure called REFINE.) Note that the Random Adversary is similar to a standard deterministic adversary in most parallel lower bound proofs. However, unlike deterministic adversaries that can fix inputs arbitrarily, the Random Adversary must fix inputs according to the chosen input distribution, i.e., using the procedure RANDOMSET, as described below. Also, depending on how RANDOMSET fixes the inputs, the desired properties might not hold. Therefore, it is possible that the Random Adversary might have to make repeated calls to RANDOMSET to ensure the desired properties.

The final step is to show that these desired properties (such as knowledge about the inputs still being widely dispersed among the processors, and that the number of inputs left unset is still large) hold with some given probability.

In the rest of this section we formalize this method.

3.1 Definitions

Let P be a problem and I the set of inputs to P. Let Q be the set of possible values to which each input could be set. Define a *partial input map* to be a function f from I to $\{\{*\} \cup Q\}$. Here '$*$' will denote a "blank" or "unset" input. A partial input map is an *input map* if no inputs are mapped to '$*$'. Let f_* denote the partial input map which maps every input to '$*$'. A partial input map f' is called a *refinement* of a partial input map f if for all $i \in I$, and $q \in Q$, $f(i) = q$ implies $f'(i) = q$. (We denote this by $f' \leq f$.)

3.2 Yao's Theorem

The following theorem shows that a lower bound on the average case time of a deterministic algorithm over random inputs implies the same lower bound bound for the expected time of any randomized algorithm over a worst case input.

Theorem 3.1 (Yao[7]) *Let T_1 be the expected running time for a randomized algorithm solving problem P over all possible inputs, where the expected time is taken over the random choices made by the algorithm. Let T_2 be the average running time over the distribution $\mathcal{D}$ of inputs, minimized over all possible deterministic algorithms to solve P. Then $T_1 \geq T_2$.*

(A nice proof of this theorem is given in Fich, Meyer auf der Heide, Ragde, and Wigderson [3].)

This theorem greatly simplify the problem of proving randomized lower bounds, as it converts the original problem to one where randomness only comes into play through the input distribution, and this can be set as one wishes. It is of course necessary to choose a distribution that will be difficult for any deterministic algorithm. Note that the input distribution cannot place all the probability on one input (i.e. a "worst case" input), since then a simple deterministic algorithm which checks for this input and outputs the precomputed answer will succeed with probability 1.

3.3 RandomSet Procedure

We will assume the distribution chosen is $\mathcal{D}$. Function RANDOMSET can be used to randomly generate an input map one input at a time. It is called with a partial input map f obtained through calls to RANDOMSET, and a set S of elements which are mapped to '$*$'. The elements in S are then randomly set one by one according to the distribution $\mathcal{D}$, conditional on f.

```
Function RANDOMSET($f, S$)
For each $i \in S$
    Set $f(i)$ according to the conditional distribution of $i$ given that
        the input is drawn from $\mathcal{D}$ and is a refinement of $f$
Return $f$
End RANDOMSET
```

Claim 3.1 *Assuming f generated solely by calls to RANDOMSET, then f will be generated according to the distribution $\mathcal{D}$.*

Proof: Straightforward. $\square$

3.4 REFINE and GENERATE

Say f is t-good if it satisifes certain properties, which will be defined with respect to the time t, the problem P, and the input distribution $\mathcal{D}$. (It will be the case that for $t' \geq t$, if f is t-good, then f is t'-good.) Say $T \leq n$ is the time that we are trying to show is a lower bound for solving the problem P. Let A be an algorithm which alledgely solves problem P over the input distribution $\mathcal{D}$ in time T.

Given this algorithm A, we create a procedure REFINE which tells the Random Adversary how to fix the inputs at each step. Formally, REFINE(t, f) takes a time t and a partial input map f and returns a pair (f', x) consists of a new partial input map f' that is a refinement of f and a lower bound x on the time of the next step. We need to prove that the procedure REFINE has two important properties, the first of which is concerned with preservation of "t-goodness".

Lemma 3.1 *If $t < T$ and* REFINE *is called with parameters (t, f), where f is t-good, then with probability at least $1 - n^{-2}$* REFINE *will return a pair (f', x) where either $t + x \geq T$ or f' is $(t + x)$-good.*

The second property is that REFINE is unbiased. Consider the function GENERATE defined below that starts with the partial input map $f_0 = f_*$, and calls REFINE until $t \geq T$ to generate a sequence of partial input maps $f_0 = f_* \geq f_{t_1} \geq \cdots \geq f_T \geq f$ where $(f_{t_i}, x) = \text{REFINE}(t_{i-1}, f_{t_{i-1}})$, $t_i = t_{i-1} + x$, f_{t_i} is a refinement of $f_{t_{i-1}}$, and f is an input map generated according to the conditional distribution over $\mathcal{D}$ from the set of refinements of f_T. Then we need to prove the following lemma.

Lemma 3.2 *The input map f returned by* GENERATE *is generated according to the distribution $\mathcal{D}$.*

In the REFINE procedure we construct in this paper, all inputs are set by calls to RANDOMSET. Consequently, by Claim 3.1, Lemma 3.2 will always hold.

```
Function GENERATE
    Let $f_0 = f_*$
    Let $f = f_0$
    Let $t = 0$
    While $t \leq T$ Do
        If for some $p$, $f(p) = $ '*' Then
            Let $(f, x) = \text{REFINE}(t, f)$
        Else
            Let $x = T - t$
        $t = \min\{t + x, T\}$
        $f_t = f$
    Let $P = \{p | f(p) = \text{'*'}\}$
    Return RANDOMSET$(f, P)$
End GENERATE
```

For concreteness, we say the partial input map f_t at any time t is simply the partial input map at the end of the last completed QRQW PRAM step.

Lemma 3.3 *With probability at least $1 - n^{-1}$, for any $t < T$, the partial input map f_t is t-good.*

Proof: Let Z_t be a binary random variable which is equal to 1 if f_t is t-good. Then the probability that f_t is not t-good can be bounded by

$$\sum_{i=1}^{t} \Pr(Z_t = 0 \mid Z_{t-1} = 1, \ldots, Z_1 = 1).$$

By Lemma 3.1, this is at most $Tn^{-2} \leq n^{-1}$. □

In summary, to fill in the Random Adversary framework for a specific problem P, we must specify

1. an input distribution $\mathcal{D}$,
2. a definition for t-good,
3. a function REFINE,
4. a time T, and
5. a proof for Lemma 3.1.

4 The Lower Bound

4.1 QRQW PRAM definitions

Let A be any deterministic algorithm for the QRQW PRAM. Let f be any input map. Trace$(p, 0, f)$ is defined to be the tuple $< p >$. Trace(p, t, f) (for $t > 0$) is defined to be the tuple $< p, \lambda_1, \ldots, \lambda_t >$ in which λ_j is a set of (cell, contents) pairs (if any) for each cell read in a QRQW PRAM step ending at time j, if any, and λ_j is the null symbol otherwise. Similarly, Trace$(c, 0, f)$ is defined to be the tuple $< c, \lambda_0 >$, where λ_0 is the initial value in cell c. Trace(c, t, f) (for $t > 0$) is defined to be the tuple $< c, \lambda_0, \ldots, \lambda_t >$ in which λ_j is the contents of the cell at time j.

For the following, assume that v is either a processor or cell. Let g be any partial input map. Define States(v, t, g) to be the set $\{$Trace$(v, t, f) : f$ is an input that refines $g\}$. Define Know(v, t, g) as the minimum set of inputs such that for any input maps f_1 and f_2 that refine g and have $f_1(q) = f_2(q)$ for all $q \in$ Know(v, t, g), Trace(v, t, f_1) is the same as Trace(v, t, f_2). (Intuitively, v is not dependent on inputs outside Know(v, t, g), since these could not affect its trace, and v is dependent on every input inside Know(v, t, g) by the fact that it is the minimum set of inputs which could affect its trace.) Let AffProc(i, t, g) contain each processor p in which $i \in$ Know(p, t, g). Let AffCell(i, t, g) contain each cell c in which $i \in$ Know(c, t, g).

4.2 General Lower Bound

Before we prove a lower bound on Load Balancing, we will prove a general lower bound on the amount of information which can be transferred between processors given a general random input.

Assume $|I| = n$ (i.e., the number of inputs is n). Without loss of generality, assume n is large enough so that our analysis holds. Assume the set of possible values for inputs $Q = \{v_1, \ldots, v_{|Q|}\}$. Let $K = |Q|$. We will use an input distribution with the property that given a partial input map which fixes any set of at most $Tn^{\frac{2}{3}}$ inputs, the probability that an unfixed input is assigned value v_j $(1 \leq j \leq K)$ is at least $q \geq (\log n)^{-1}$.

We define the following values for $i \geq 0$: $k_i = K^{2^{2i}}$, and $r_i = in^{\frac{2}{3}}$.

If $T \leq \frac{1}{4} \log\log\log n$ and $K \leq 2^{\sqrt{\log\log n}/2}$ then the following is easily proved.

Fact 4.1 *$k_T \leq \sqrt{\log n}$, and $r_T \leq Tn^{\frac{2}{3}}$.*

A partial input map f is called *t-good* if the following conditions are satisfied.

1. For each processor or cell v, $|$States$(v, t, f)| \leq k_t$.

2. For each processor or cell v, $|$Know$(v, t, f)| \leq k_t$.

3. For each input i, $|$AffProc$(i, t, f)| \leq k_t$ and $|$AffCell$(i, t, f)| \leq k_t$.

4. f maps at least $n - r_t$ inputs to '$*$'.

We now describe algorithm REFINE which is called with a time t and a partial input map f, and which returns a pair (f', x) consisting of a partial input map f' which is a random refinement of f, and a lower bound x on the time of the step taken. Note that x is the true time of the step if $t + x < T$. The random refinement is based on the action of algorithm A on the step.

The intuition behind this REFINE procedure is the following. First, in lines (3) through (8), we force some processor that possibly accesses many cells in this step to actually access those cells. Since each state of a processor has a reasonably high probability, we can force this without fixing the inputs of too many processors. This gives the lower bound RWCount on the time of the step.

Next, in lines (10) through (21), we force some cell that is possibly accessed by many processors, to actually be accessed by those processors, or at least $\log \log n$ of them if there are more than $\log \log n$ that could access the cell for some input map. Since each state of a processor has a reasonably large probability, we can force many processors to actually access a cell without needing to consider too many cells (i.e., without fixing the inputs that affect processors that possibly write to those cells) Thus REFINE forces the time required for the step to correspond to the "amount of information exchanged" in the step, without fixing too many inputs.

We define MaxCell(t, g) as the cell with the maximum possible contention at time t for any input map which refines g. We define MaxRWC(c, t, g) as the maximum possible contention at c at time t for any input map which refines g. We define MaxProc(t, g) as the processor with the maximum possible number of reads or writes at time t for any input map which refines g. We define MaxRWP(p, t, g) as the maximum possible number of reads or writes p makes at time t for any input map which refines g. Let ACCESS(c, t, g) be the set of processors that read from or write to cell c at time t for any input map which refines g.

```
       Function REFINE($t, f$)
(1)        Let $g = f$
(2)        Let Done = FALSE
(3)        While not Done
(4)            Let $p$ = MaxProc($t, g$)
(5)            Let RWCount = MaxRWP($p, t, g$)
(6)            Let $g$ = RANDOMSET(Know($p, t, g$), $g$)
(7)            If $p$ accesses RWCount cells
(8)                Let Done = TRUE
(9)        Let Done = FALSE
(10)       While not Done
(11)           Let $c$ = MaxCell($t, g$)
(12)           Let $W$ = ACCESS($c, t, g$)
(13)           Choose a set $W' \subseteq W$ such that
                   $|W'| = \min\{|W|, m_t s_t 2^{\log^{\frac{3}{4}} n}\}$
(14)           Let $B = \min\{\text{MaxRWC}(c, t, g), \log\log n\}$
(15)           Let Count = 0
(16)           For each $p \in W'$
(17)               Let $g$ = RANDOMSET(Know($p, t, g$), $g$)
(18)               If $p$ accesses $c$
(19)                   Let Count = Count + 1
(20)           If Count $\geq B$
(21)               Let Done = TRUE
(22)       Let $f' = g$
(23)       Return ($f'$, max{Count, RWCount}
       End REFINE
```

Lemma 4.1 *If f is t-good and* REFINE(t, f) *returns (f', x), then either $t + x \geq T$ or (1) For each processor or cell v, $|States(v, t + x, f')| \leq k_{t+x}$, (2) For each processor or cell v, $|Know(v, t + x, f')| \leq k_{t+x}$, and (3) For each input i where $f'(i)$ = '$*$', $|AffProc(i, t + x, f')| \leq k_{t+x}$ and $|AffCell(i, t + x, f')| \leq k_{t+x}$.*

Proof: If line (21) of REFINE is executed with $B = \log\log n$, then $x \geq \log\log n$, and $t + x \geq T$. Otherwise, one can see that the time of the step will be x, and we obtain the bounds below.

The bounds we show below follow from the fact that for $x \geq 1$, $1 + x \leq 2^x$. It is possible to have $x = 0$, and it is easy to check that our bounds hold in that case also.

We analyze this by read step, then write step. For the read step, processor v reads one of at most k_t sets of at most x cells at step t, each one being in one of at most k_t states. Thus $|\text{States}(v, t+x, f')| \leq k_t^{x+1} \leq K^{2^{2(t+x)}} = k_{t+x}$.

The inputs that affect processor v are the m_t that originally affect it plus the m_t that affect each of the xk_t possible cells it reads. Thus $|\text{Know}(v, t+x, f')| \leq m_t + xk_tm_t \leq K^{2^{2(t+x)}} = m_{t+x}$.

An input affects all the processors it originally affects, plus all the processors that read from the cells that it affects. Thus $|\text{AffProc}(v, t+x, f')| \leq s_t + xs_t \leq K^{2^{2(t+x)}} = s_{t+x}$.

For the write step, cell v is written to by at most x processors, and thus it could either be in the state it was originally or in one of the k_t states of the processors that write to it. Thus $|\text{States}(v, t+x, f')| \leq k_t + xk_t \leq K^{2^{2(t+x)}} = k_{t+x}$.

The inputs that affect a cell v are the at most m_t that originally affect it plus the m_t that affect each of the at most x processors that possibly write to it. Thus $|\text{Know}(v, t+x, f')| \leq m_t + xm_t \leq K^{2^{2(t+x)}} = m_{t+x}$.

An input affects all the cells it originally affects, plus the xk_t cells possibly written to by each processor that it affects. Thus $|\text{AffCell}(v, t+x, f')| \leq s_t + xk_ts_t \leq K^{2^{2(t+x)}} = s_{t+x}$. $\square$

We say that REFINE(t, f) is *successful* if it calls RANDOMSET with at most $n^{\frac{2}{3}}$ inputs.

Lemma 4.2 *If f is t-good and REFINE(t, f) is successful and returns (f', x), then either $t+x \geq T$ or f' is $(t+x)$-good.*

Proof: This follows from Lemma 4.1, the definition of successful, and the definition of t-good. $\square$

Lemma 4.3 *If f is t-good then* REFINE(t, f) *is successful with probability at least* $1 - n^{-2}$.

Proof: Consider the While loop at lines (3) through (8). Since the probability of a processor being in any state is at least $q^{\sqrt{\log n}} \geq 2^{-\log^{\frac{2}{3}} n}$, the probability of this executing more than $\sqrt{n}$ times is at most

$$(1 - 2^{-\log^{\frac{2}{3}} n})^{\sqrt{n}} \leq n^{-3}.$$

Then since at Know$(p, t, g) \leq \log n$, the probability of more than $\sqrt{n}\log n$ inputs being set is at most n^{-3}.

Now consider the While loop at lines (10) through (21). We argue that the probability of not finishing at one of the first $\sqrt{n}$ iterations of this loop is at most n^{-3}.

First we show that if for one of the first $\sqrt{n}$ iterations of the loop, $|W'| = k_t^2 2^{\log^{\frac{3}{4}} n}$, then the probability of not finishing on that iteration is less than n^{-3}. Since each processor is affected by at most k_t inputs, and each input affects at most k_t processors, we can find a set of processors $W'' \subseteq W'$ of size $2^{\log^{\frac{3}{4}} n}$ whose states are independent. Also, we use the fact that the probability of being in any state is at least $2^{-\log^{\frac{2}{3}} n}$. By a Chernoff bound, the probability that fewer than $\log\log n$ processors in W'' write to c is at most n^{-3}.

Assuming $|W'| < k_t^2 2^{\log^{\frac{3}{4}} n}$ for the first $\sqrt{n}$ iterations, we first find independent iterations. Each of the $\sqrt{n}$ cells that we consider is written to by at most $k_t^2 2^{\log^{\frac{3}{4}} n}$ processors. Each of these processors is affected by k_t inputs and each input affects at most k_t processors. Thus there is a set C of at least $\sqrt{n}/(k_t^4 2^{\log^{\frac{3}{4}} n}) \geq n^{\frac{1}{4}}$ cells which are written to by processors whose states are independent. For each cell $c \in C$, there is at least one input map g' such that a set W'' of B processors write to c. Since the processors in W'' are affected by a total of at most Bk_t inputs, the minimum probability of fixing inputs so that all these processors write to c is at least $q^{Bk_t} \geq 2^{-\log^{\frac{2}{3}} n}$. (Note that since $W' = W$, we actually will fix inputs affecting all the processors in W', and hence W''.)

Thus the probability of not finishing in any of the first $\sqrt{n}$ iterations is less than $(1-2^{-\log^{\frac{2}{3}} n})^{n^{\frac{1}{4}}} \leq n^{-3}$.

Note that if we do finish in the first $\sqrt{n}$ iterations, we set at most $\sqrt{n} k_t^3 2^{\log^{\frac{3}{4}} n} \leq n^{\frac{3}{5}}$ inputs.

In total, REFINE(t, f) fails with probability at most $n^{-3} + n^{-3} \leq n^{-2}$. □

Lemma 4.4 *If $t < T$ and* REFINE *is called with parameters (t, f), where f is t-good, then with probability at least $1 - n^{-2}$* REFINE *will return a pair (f', x) where either $t + x \geq T$ or the partial input map f' is $(t + x)$-good.*

Proof: This follows from Lemma 4.3 and Lemma 4.2. □

Corollary 4.1 *With probability at least $1 - n^{-1}$, for any processor or cell v, and any time $t \leq T$, $|Know(v, t, f_t)| \leq k_T \leq \sqrt{\log n}$; for any unset input i and any time $t \leq T$, $|AffProc(i, t, f_t)| \leq k_T \leq \sqrt{\log n}$, $|AffCell(i, t, f_t)| \leq k_T \leq \sqrt{\log n}$; and the number of inputs set by RANDOMSET is at most $r_T \leq Tn^{\frac{2}{3}}$.*

5 Linear Approximate Compaction

Let $m = \log\log n$. Let $\mathcal{D}$ be an input distribution that consists of an array of n cells, with $\frac{n}{m}$ items located at uniformly random cells in the array. (In other words, $Q = \{v_1, v_2\}$ where input i is v_1 if cell i in the input array contains an item, otherwise input i is v_2. Note that this distribution satisfies the property we required above.) Without loss of generality we assume each item is tagged with the cell it originally occupies. Say the *enhanced LAC* problem (ELAC) is for each nonempty cell in the input array to contain a pointer to the location in the output array where its item was inserted.

Claim 5.1 *Given an algorithm for the LAC problem on a QRQW PRAM that uses expected time t, there exists an algorithm for the ELAC problem on a QRQW PRAM that uses expected time $t + 1$.*

Proof: One can simply run the algorithm solving the LAC problem, and then assign one processor per output cell to read the tag of the item inserted there and write that location into the original cell containing the item. □

Lemma 5.1 *A deterministic algorithm which solves the ELAC problem on a QRQW PRAM requires $\Omega(\log\log\log n)$ expected time.*

Proof: Consider a deterministic algorithm that alledgedly solves the ELAC problem in expected time $t < T$. Let C be the constant such that the algorithm places the $\frac{n}{m}$ items into an array of size $\frac{Cn}{m} \leq \frac{n}{8}$, for large n. From Corollary 4.1, with probability $1 - n^{-1}$, for any processor or cell v, $|\text{Know}(v, t, f_t)| \leq k_t \leq \sqrt{\log n}$; for any unset input i $|\text{AffCell}(i, t, f_t)| \leq k_t \leq \sqrt{\log n}$; and the number of inputs set by RANDOMSET is at most $Tn^{\frac{2}{3}} \leq \frac{n}{2}$. For now, we assume these conditions hold.

Consider an input cell c whose input has not been fixed by the Random Adversary. Consider the contents of c (the pointer to the location in the output array) assuming that the input map f which was refined from f_t assigned items to all inputs in $\mathrm{Know}(c, t, f_t)$. Do this for all input cells whose inputs have not been fixed. This defines a *potential pointer map* F. Then F is a function with a domain of size at least $\frac{n}{2}$ and a range of size at most $\frac{n}{8}$. By a simple counting argument, we can find $\frac{n}{8}$ disjoint pairs of input cells, each of which point to the same output cells.

Let S be one of the disjoint pairs of input cells that we have just found. Each $c \in S$ has $\mathrm{Know}(c, t, f_t) \leq k_t \leq \sqrt{\log n}$, and thus at most $2\sqrt{\log n}$ inputs affect the contents of these cells. Each of these inputs i has $\mathrm{AffCell}(i, t, f_t) \leq k_t \leq \sqrt{\log n}$, so at most $2 \log n$ cells are affected by the same inputs that affect the cells in S. Thus from the $\frac{n}{8}$ disjoint pairs of input cells which are mapped by F to the same output cell, we can find a subset of $B = \frac{n}{16 \log n}$ pairs whose cell contents are completely independent. Number these pairs from 1 to B.

Claim 5.2 *With very high probability, at least one pair from B uses the same pointers as F.*

Proof: We can see that the probability of both cells in one of these pairs using the pointers from F is at least the probability that f assigns items to all inputs in $\mathrm{Know}(c, t, f_t)$ for each c in the pair. Even conditioned on any values of inputs from the other $B - 1$ pairs and the inputs fixed by the Random Adversary, the probability of this event is at least $(2m)^{-2\sqrt{\log n}}$. (Note that the number of inputs we are conditioning on is at most $\frac{n\sqrt{\log n}}{16 \log n} + \frac{1}{4} n^{\frac{2}{3}} \log\log\log n \leq \frac{n}{2 \log\log n}$.)

The probability that this does not happen for any of the B pairs is at most

$$(1 - (2m)^{-2\sqrt{\log n}})^B \leq e^{B(2m)^{-2\sqrt{\log n}}} \leq e^{-\sqrt{n}},$$

for sufficiently large n Thus, with very high probability, at least one pair of input cells will be mapped to the same output cell. □

By Claim 5.2, with very high probability the mapping provided by the algorithm at this point will not be a valid solution to ELAC. This is assuming that the conditions implied by Corollary 4.1 hold, but these hold with high probability. Thus with high probability, the mapping provided by the algorithm at this point will not be a valid solution to ELAC. Then since $T = \Omega(\log\log\log n)$, this proves the lemma. □

Corollary 5.1 *A deterministic algorithm which solves the LAC problem on a QRQW PRAM requires $\Omega(\log\log\log n)$ time.*

Proof: From Claim 5.1, given a deterministic algorithm that solves the LAC problem in $t' = o(\log\log\log n)$ steps, we could solve the ELAC problem in $t' + 1 = o(\log\log\log n)$ steps, which is impossible by Lemma 5.1. □

Theorem 5.1 *Solving the* LAC *problem on a Randomized QRQW PRAM requires* $\Omega(\log\log\log n)$ *expected time.*

Proof: Assume there is a randomized algorithm that solves LAC on the QRQW PRAM in expected time t. Then by Yao's Theorem, for the distribution $\mathcal{D}$ given above, there is a deterministic algorithm which solves LAC over that distribution in expected time t. Then by Corollary 5.1, $t = \Omega(\log\log\log n)$. □

6 Conclusions

In this paper we have presented some new ideas for proving lower bounds on the QRQW PRAM, and shown how these ideas lead to an improved lower bound for Linear Approximate Compaction. The lower bounds we obtain pertain to randomized algorithms, which in many cases (and in particular for the case of LAC) seem to be very useful.

There is still a large gap between the lower and upper bounds for LAC. Unfortunately, the technique we use does not seem able to completely bridge this gap. The reason is that computing LAC for the input distribution we give requires only $O(\log\log n)$ expected time, even on an EREW PRAM. (This can be done by performing deterministic LAC in groups of polylogarithmic size, and noting that with high probability, no group contains too many marked inputs.) We leave open the problem of reducing this gap.

References

[1] P. Beame, M. Kik, and M. Kutylowski. Information broadcasting by exclusive-read prams. *Para. Process. Lett.*, 4:159–169, 1994.

[2] M. Dietzfelbinger, M. Kutylowski, and R. Reischuk. Exact lower time bounds for computing boolean functions on CREW PRAMs. *J. Comput. System Sci.*, 48:231–254, 1994.

[3] F. E. Fich, F. Meyer auf der Heide, P. Ragde, and A. Wigderson. One, two, three ...infinity: Lower bounds for parallel computation. In *Proc. 17th Symp. on Theory of Computing*, pages 48–58, 1985.

[4] P. B. Gibbons, Y. Matias, and V. Ramachandran. Efficient low-contention parallel algorithms. In *Proc. 6th ACM Symp. on Para. Alg. and Arch.*, pages 236–247, 1994.

[5] P. B. Gibbons, Y. Matias, and V. Ramachandran. The qrqw pram: Accounting for contention in parallel algorithms. In *5th ACM-SIAM Symp. on Disc. Alg.*, pages 638–648, 1994.

[6] P. D. MacKenzie. Load balancing requires $\Omega(\log^* n)$ expected time. In *3rd ACM-SIAM Symp. on Disc. Alg.*, pages 94–99, 1992. submitted to SIAM Journal on Computing.

[7] A. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 18th Symp. on Found. of Comp. Sci.*, pages 222–227, 1977.