Computational Bounds for Fundamental Problems on General-Purpose Parallel Models*

Philip D. MacKenzie†
Dept. of Mathematics and Computer Science
Boise State University
Boise, ID 83712
philmac@cs.idbsu.edu

Vijaya Ramachandran‡
Department of Computer Sciences
University of Texas
Austin, TX 78712
vlr@cs.utexas.edu

April 21, 1998

Abstract

We present lower bounds for time needed to solve basic problems on three general-purpose models of parallel computation: the shared-memory models QSM *and* s-QSM*, and the distributed-memory model, the* BSP*. For each of these models, we also obtain lower bounds for the number of rounds needed to solve these problems using a randomized algorithm on a p-processor machine. Our results on 'rounds' is of special interest in the context of designing work-efficient algorithms on a machine where latency and synchronization costs are high. Many of our lower bound results are complemented by upper bounds that match the lower bound or are close to it.*

1 Introduction

Recently, there has been a great deal of interest in developing general-purpose models of parallel computation that incorporate features of real machines such as bandwidth limitations and the resulting cost for global memory accesses. The BSP [24] and LogP [5] models are distributed memory models of this type, and the QSM [10] and s-QSM are shared-memory models of this type.

As a shared-memory model, the QSM can be viewed as a generalization of the PRAM model [13] with the QRQW memory access rule [9] (which is intermediate between the EREW and CRCW rules), a parameter g to capture bandwidth limitations, and 'bulk-synchrony' in place of synchronization after each step as in the PRAM model. In a bulk-synchronous computation, individual processors can execute a number of steps in an asynchronous manner before a global synchronization step.

In this paper we study the time needed to solve basic problems on the QSM and the s-QSM. We will prove many of our lower bounds on a more generalized shared-memory model, the GSM, and those results will also translate into lower bounds for the BSP model.

In addition to studying the time needed to solve a problem regardless of the number of processors used, we also obtain bounds on the number of *rounds* needed in a computation with p processors.

*An extended abstract of this work will appear in *Proc. 1998 ACM Symp. on Parallel Algorithms and Architectures* [19].

†Part of this work was performed at Sandia National Laboratories and was supported by the U.S. Department of Energy under contract DE-AC04-76DP00789. Part of this work was performed while visiting the University of Texas.

‡Supported in part by NSF Grant CCR/GER 90-23059.

A round in a computation is a sequence of steps between two global synchronizations in which the total work performed by the algorithm is linear in n, the size of the input. It is desirable to minimize the number of global synchronization steps in an algorithm that runs on a machine that has high latency or synchronization costs. In addition, if one is interested in algorithms that perform linear work, then it becomes desirable to require the algorithm to compute in rounds, since any linear-work algorithm must compute in rounds. In this paper, we present bounds on the number of rounds needed for basic problems on a p-processor machine, where $p \leq n$.

Our lower bound results are tabulated in four tables in Table 1. A brief description of some upper bounds is given in Section 8. We study the following three classes of problems.
(1.) *Linear Approximate Compaction (LAC).* In many cases our lower bounds for this problem also hold for related problems such as load balancing and padded sort.
(2.) *Computing the OR.* Our lower bound on time for randomized algorithms uses the Random Adversary Technique [15] in a novel way, which could be of independent interest. Although the lower bound is quite weak – on the order of $\log^* n$ – it establishes that the OR function cannot be computed in time independent of the size of the input by any randomized algorithm on these models.
(3.) *Parity.* The lower bounds for this problem imply lower bounds for other important problems such as list ranking and sorting.

There are a large number of lower bound results known for computation on the traditional PRAM models as well as some for the QRQW PRAM [9]. However, we are not aware of many lower bounds results for the general-purpose models considered in this paper. A tight lower bound on the time needed for broadcasting on the QSM and the BSP is given in [1]. A lower bound for the number of rounds needed on the BSP to compute the OR by a deterministic algorithm is given in [11]. Among the results we present is the same lower bound as the one in [11], but one that holds for randomized algorithms as well.

Many of our results build on techniques and results developed earlier for the CRCW PRAM [3, 15], QRQW PRAM [9, 16, 17, 18, 14], EREW PRAM [16], and the few-write PRAM [6].

The rest of this paper is organized as follows. In Section 2 we define the QSM, s-QSM and BSP models, as well as our lower-bound model, the GSM, and we present some basic results on mapping lower bounds from the GSM to the other models. In Section 3 we present our lower bound results for Parity. In Section 4 we review the Random Adversary technique [16]. In Section 5 we develop a general lower bound proof based on the Random Adversary for the GSM, which we use in Section 6 to derive lower bounds for Linear Approximate Compaction (LAC) and related problems. In Section 7 we present a modified version of the Random Adversary and show how it can be used to derive a lower bound for OR (in Sections 6 and 7 we also present some deterministic lower bounds for LAC and OR). Section 8 presents our upper bound results.

2 Definitions and Basic Results

2.1 General-purpose Models

1. **The QSM model.** [10]

The *Queuing Shared Memory (QSM)* model consists of a number of identical processors, each with its own private memory, communicating by reading and writing locations in a shared memory. Processors execute a sequence of synchronized phases, each consisting of an arbitrary interleaving of the following operations:

Shared-memory reads: Each processor i copies the contents of r_i shared-memory locations into its private memory. The value returned by a shared-memory read can only be used in a subsequent phase.
Shared-memory writes: Each processor i writes to w_i shared-memory locations.
Local computation: Each processor i performs c_i RAM operations involving only its private state and private memory.

Concurrent reads or writes (but not both) to the same shared-memory location are permitted in a phase. In the case of multiple writers to a location x, an arbitrary write to x succeeds in writing the value present in x at the end of the phase.

The *maximum contention* of a QSM phase is the maximum, over all locations x, of the number of processors reading x or the number of processors writing x. A phase with no reads or writes is defined to have maximum contention one.

Consider a QSM phase with maximum contention κ. Let $m_{op} = \max_i\{c_i\}$ and $m_{rw} = \max\{1, \max_i\{r_i, w_i\}\}$ for the phase. Then the *time cost* for the phase is $\max(m_{op}, g \cdot m_{rw}, \kappa)$. The *time* of a QSM algorithm is the sum of the time costs for its phases.

The particular instance of the Queuing Shared Memory model in which the gap parameter, g, equals 1 is the Queue-Read Queue-Write (QRQW) PRAM model defined in [9].

2. **The s-QSM Model**. [10]

This is essentially the QSM, except that there is a gap parameter with value g for processing each access at memory, in addition to the gap parameter at processors. Thus the time cost of an s-QSM phase with maximum contention κ, and with $m_{op} = \max_i\{c_i\}$ and $m_{rw} = \max\{1, \max_i\{r_i, w_i\}\}$ is $\max(m_{op}, g \cdot m_{rw}, g \cdot \kappa)$.

The QRQW PRAM is the same as the s-QSM with gap parameter 1.

3. **The BSP Model.** [24]

The BSP model consists of p processor/memory components that communicate by sending point-to-point messages. The interconnection network supporting this communication is characterized by a bandwidth parameter g and a latency parameter L. An input of size n is partitioned uniformly among the p components so that each component is assigned either $\lceil n/p \rceil$ or $\lfloor n/p \rfloor$ of the inputs. A BSP computation consists of a sequence of "supersteps" separated by bulk synchronizations. In each superstep the processors can perform local computations and send and receive a set of messages. Messages are sent in a pipelined fashion, and messages sent in one superstep will arrive prior to the start of the next superstep. It is assumed that in each superstep messages are sent by a processor based on its state at the start of the superstep. The time charged for a superstep is calculated as follows. Let w_i be the amount of local work performed by processor i in a given superstep. Let s_i (r_i) be the number of messages sent (received) by processor i, and let $w = \max_{i=1}^{p} w_i$. Let $h = \max_{i=1}^{p}(\max(s_i, r_i))$; h is the maximum number of messages sent or received by any processor, and the BSP is said to route an *h-relation* in this step. The *cost*, T, of a superstep is defined to be $T = \max(w,\ g \cdot h,\ L)$. The time taken by a BSP algorithm is the sum of the costs of the individual supersteps in the algorithm. We will assume $L \geq g$ throughout this paper.

2.2 A Lower Bound Model

Generally speaking, the BSP is a lower level model than the QSM or s-QSM since it is a distributed-memory model, and it has an additional parameter, L, that is not present in the QSM models. However, due to the message-passing mode of communication used in the BSP, in certain situations it is more powerful than the QSM or s-QSM. For instance , if several different processors send values to a given processor to be placed in an array (in any order), the BSP processor can fill in the array

by simply picking out the elements from its input buffer. On a QSM this computation involves compaction, since each value needs to be tagged with an explicit location within the array in which it needs to be placed.

In order to derive lower bounds for the three models we consider, we define below the GSM model, which is stronger than all three of these models (and also stronger than the QSM(g,d) [10, 21], which is a generalization of the QSM and s-QSM.) We derive most of our lower bounds on this stronger model, and then obtain the lower bounds for the three models of interest as corollaries of the lower bound on the GSM.

The GSM Model

The *Generalized Shared Memory (*GSM*)* model consists of a number of identical processors, each with its own private memory, communicating by reading and writing cells in a shared memory. These shared-memory cells can hold an arbitrarily large amount of information. Processors execute a sequence of synchronized phases, each consisting of an arbitrary interleaving of shared-memory reads and shared-memory writes. Concurrent reads or writes (but not both) to the same shared-memory location are permitted in a phase. In the case of multiple writers, *all information from all writing processors is transferred to the cell, and added to the information already present in the cell.* We refer to this as the *strong queuing model.* The maximum contention for a phase is defined as in the QSM.

The GSM model has three parameters, α, β, and γ. We will always take $\mu = \max\{\alpha, \beta\}$, $\lambda = \min\{\alpha, \beta\}$. At the beginning of an algorithm, we assume that each cell contains information about up to γ inputs (disjoint from other cells). A GSM phase operates in integral units of *big-steps* which take $\mu = \max\{\alpha, \beta\}$ time each. The number of big-steps in a phase with maximum contention κ and maximum number of reads and writes by any processor m_{rw} is $b = \max\{\lceil m_{rw}/\alpha \rceil, \lceil \kappa/\beta \rceil\}$. The *time cost* of a GSM phase with b big-steps is $\mu \cdot b$. The *time* of a GSM algorithm is the sum of the time costs of its phases. Note that a single big step can "handle" α reads and writes, and β contention at any cell.

2.3 Rounds

A *round* in a computation (deterministic or randomized) with p processors on an n-element input is a phase in a QSM or s-QSM computation that takes $O(gn/p)$ time. On a BSP a round is a superstep in which the BSP routes an $O(n/p)$-relation (i.e., $h = O(n/p)$) and performs $O(gn/p + L)$ local computation at any processor.

A *round* in a GSM computation with p processors on an n-element input, where $p \leq n$ and $\gamma \leq n/p$, is a phase which takes $O(\mu n/\lambda p)$ time.

A p-processor algorithm on the QSM or s-QSM performs *linear work* if the processor-time product is $O(g \cdot n)$, where n is the size of the input. As shown in [10], $\Omega(g \cdot n)$ is a lower bound on this product for most nontrivial problems. A p-processor algorithm on a GSM performs linear work if the processor-time product is $O(\mu n/\lambda)$; this is again a lower bound for most nontrivial problems.

Under the above definitions, it should be clear that any linear-work algorithm must compute in rounds, and that an r-round computation on an input of size n performs at most $O(rgn)$ work on a GSM, QSM or s-QSM. On a p-processor BSP this computation has an upper bound of $O(r(gn + Lp))$ on work. For small r these upper bounds are close to linear work.

As noted in the introduction, it is desirable to minimize the number of rounds in a computation, if one wants to design efficient algorithms for parallel machines in which latency is high or global synchronization is a costly operation (or both).

2.4 Mapping GSM Bounds to the Other Models

We now show that lower bound results for the GSM translate into corresponding lower bound results for the QSM, s-QSM, and BSP.

Claim 2.1 *Let $T_{\text{GSM}}(n, \alpha, \beta, \gamma)$ be the time bound of a fastest algorithm on the GSM for a computational problem, and let $R_{\text{GSM}}(n, \alpha, \beta, \gamma, p)$ be the minimum number of rounds needed to perform the computation with p processors, $p \leq n$, on the GSM. Let $T_{\text{QSM}}(n, g)$, $R_{\text{QSM}}(n, g, p)$, $T_{s-\text{QSM}}(n, g)$, $R_{s-\text{QSM}}(n, g, p)$, $T_{\text{BSP}}(n, g, L, p)$, and $R_{\text{BSP}}(n, g, L, p)$ be the corresponding quantities on the QSM, s-QSM, and BSP, respectively.*
For lower bounds that do not consider local computations we have

1. $T_{\text{QSM}}(n, g) = \Omega(T_{\text{GSM}}(n, 1, g, 1))$
2. $T_{s-\text{QSM}}(n, g) = \Omega(g \cdot T_{\text{GSM}}(n, 1, 1, 1))$
3. $T_{\text{BSP}}(n, g, L, p) = \Omega(g \cdot T_{\text{GSM}}(n, L/g, L/g, n/p))$
4. $R_{\text{GSM}}(n, \alpha, \beta, \gamma, p) = \Omega(\frac{T_{\text{GSM}}(n, \alpha n/\lambda p, \beta n/\lambda p, \gamma)}{\mu n/\lambda p})$
5. $R_{\text{QSM}}(n, g, p) = \Omega(R_{\text{GSM}}(n, 1, g, 1, p))$
6. $R_{s-\text{QSM}}(n, g, p) = \Omega(R_{\text{GSM}}(n, 1, 1, 1, p))$
7. $R_{\text{BSP}}(n, g, L, p) = \Omega(R_{\text{GSM}}(n, 1, 1, n/p, p))$

Proof: The results for the time bounds follow from the observation that a phase of a QSM with time cost $\max\{m_{op}, gm_{rw}, \kappa\}$ can be executed in the same time cost or less on a GSM with $\alpha = 1$ and $\beta = g$; a phase of an s-QSM with time cost $\tau = \max\{m_{op}, g \cdot m_{rw}, g\kappa)$ can be executed on a GSM with $\alpha = 1$ and $\beta = 1$ in time at most τ/g; and a superstep of an BSP with time cost $\tau = \max\{w, g \cdot h, L\}$ can be executed on a GSM with $\alpha = L/g$ and $\beta = L/g$ in time at most τ/g. The result relating the number of rounds on the GSM to time on the GSM comes from considering a round as a phase on a GSM containing a single big-step that takes $O(\mu n/\lambda p)$ time, but operations take the same amount of time as on the original GSM. The remaining results follow from the observation that a round of a QSM can be executed in one round on a GSM with $\alpha = 1$ and $\beta = g$; a round of an s-QSM can be executed in one round on a GSM with $\alpha = 1$ and $\beta = 1$; and a round of a BSP can be executed in two rounds (one for writing and one for reading) on a GSM with $\alpha = 1$, $\beta = 1$, and $\gamma = n/p$. □

In the sections that follow, we will derive our lower bounds on the GSM and then translate them to the QSM, s-QSM, and BSP, using the above claim.

We also note that a similar claim can be obtained for the QSM(g, d) model [10, 21], and can be used to derive lower bounds for this model using the results we derive for the GSM.

Claim 2.2 *Let $T_{\text{GSM}}(n, \alpha, \beta, \gamma)$ be the time bound of a fastest algorithm on the GSM for a computational problem, and let $R_{\text{GSM}}(n, \alpha, \beta, \gamma, p)$ be the minimum number of rounds needed to perform the computation with p processors, $p \leq n$, on the GSM. Let $T_{g>d-\text{QSM}}(n, g, d)$ and $R_{g>d-\text{QSM}}(n, g, d, p)$ be the corresponding quantities on the QSM(g, d) when $g > d$ and let $T_{d>g-\text{QSM}}(n, g, d)$ and $R_{d>g-\text{QSM}}(n, g, d, p)$ be the corresponding quantities on the QSM(g, d) when $g < d$.*

For lower bounds that do not consider local computations we have

1. $T_{g>d-\text{QSM}}(n, g) = \Omega(d \cdot T_{\text{GSM}}(n, 1, g/d, 1))$

2. $T_{d>g-\mathrm{QSM}}(n,g) = \Omega(g \cdot T_{\mathrm{GSM}}(n, d/g, 1, 1))$

3. $R_{g>d-\mathrm{QSM}}(n,g,p) = \Omega(R_{\mathrm{GSM}}(n, 1, g/d, 1, p))$

4. $R_{d>g-\mathrm{QSM}}(n,g,p) = \Omega(R_{\mathrm{GSM}}(n, d/g, 1, 1, p))$

2.5 Boolean Functions

Let $x_1, \ldots, x_n$ denote boolean variables, $a_1, \ldots, a_n$ denote elements of $\{0,1\}$ and a denote an element of $\{0,1\}^n$. Let B_n denote the set $\{f | f : \{0,1\}^n \to \{0,1\}\}$. For $S \subseteq \{1, \ldots, n\}$ let m_S be the positive monomial $\prod_{i \in S} x_i$ and $m_S(a) = \prod_{i \in S} a_i$.

Fact 2.1 ([22]) *Every $f \in B_n$ can be written $f = \sum_S \alpha_S(f) m_S$ for unique integer coefficients $\alpha_S(f)$.*

For $f \in B_n$, let $\deg(f) = \max\{|S| \ | \alpha_S(f) \neq 0\}$.

Fact 2.2 ([6]) *For arbitrary $f, g \in B_n$, the following hold:*

1. $\deg(f \wedge g) \leq \deg(f) + \deg(g)$,
2. $\deg(\overline{f}) = \deg(f)$,
3. $\deg(f \vee g) \leq \deg(f) + \deg(g)$,
4. *If $g \subseteq f$, i.e., g results from f by fixing some inputs to 0 or 1, then* $\deg(g) \leq \deg(f)$.

Let $C(f)$ be the maximum of the numbers

$$\min\{k | \exists_{S \subseteq \{1,\ldots,n\}, |S|=k} f(a) = f(b) \text{ if } \forall_{i \in S} a_i = b_i\}$$

taken over all input maps a. (This was called the certificate complexity in Nisan [20].)

Fact 2.3 ([6]) $C(f) \leq (\deg(f))^4$.

For $A \subseteq \{0,1\}^n$, we denote the characteristic function of A by χ_A or $\chi(A)$. For $\mathcal{A}$ a class of subsets of $\{0,1\}^n$, let $\deg(\mathcal{A}) = \max\{\deg(\chi_A) | A \in \mathcal{A}\}$.

2.6 Yao's Theorem

The following theorem relates the success probability of a randomized algorithm to the success probability of a deterministic algorithm for a given distribution on the inputs. This theorem can be proved in a manner similar to the proof of Yao's Theorem [25].

Theorem 2.1 *Let S_1 be the success probability of a T step randomized algorithm solving problem P, where the success probability is taken over the random choices made by the algorithm, and minimized over all possible inputs. Let S_2 be the success probability over a distribution $\mathcal{D}$ of inputs, maximized over all possible T step deterministic algorithms to solve P. Then $S_1 \leq S_2$.*

This theorem greatly simplifies 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 Parity and Related Problems

Given a Boolean n-array, the *Parity* problem returns a 1 if the number of 1's in the input is odd, and returns a 0 otherwise. In this section we present algorithms and lower bounds for parity and related problems.

We start with a lower bound for deterministic algorithms on the GSM, even if unit-time concurrent reads are allowed. Our proof is an adaptation of a method used by [14] to obtain a lower bound for parity for the SIMD-QRQW model with concurrent reads (which is more restrictive than even the CRQW model) but with the added feature of 'latency detection', which is not present in the QRQW or QSM model.

Theorem 3.1 *Any deterministic algorithm for the n-element Parity problem on the* GSM *requires time $\Omega(\mu \log(n/\gamma)/\log \mu)$, even if unit-time concurrent reads are allowed.*

Proof: Let $r = n/\gamma$. Note that the problem reduces to computing Parity on an input of size r.

The proof bounds the degrees of functions describing the states of processors and contents of memory cells in each phase by restricting the set of inputs A_i we will consider in each phase i. $A_0 = \{0,1\}^r$ and for each $i > 0$, $A_i \subseteq A_{i-1}$. Let χ_{A_i} be the *characteristic function* of A_i, i.e., the function that evaluates to 1 for inputs in A_i and evaluates to 0 for inputs not in A_i. The sets A_i will be chosen so as to force the ith phase to take the maximum time.

We will prove by induction on j that the degrees of χ_{A_j}, and of the functions describing the states of processors and contents of cells at phase j are bounded by $b_j = (3 + \tau_j + 2\tau'_j)b_{j-1}$, where τ'_j is the maximum queue length at any memory cell in phase j, and τ_j is the maximum number of read or write requests issued by any processor in phase j for inputs in A_j. Note that $b_0 = 1$.

Assume the result holds for $j < i$ and consider phase i. For phase i, we define A_i to be the set of inputs in A_{i-1} that cause a processor to read from or write to the maximum possible number of cells, and subject to this constraint also cause a maximum number of writes to some cell. After the read phase, by Fact 2.2, $\deg(\chi_{A_i}) \leq b_{i-1} + b_{i-1} + \tau'_i \cdot b_{i-1}$, where the first term is from $\chi_{A_{i-1}}$, the second term comes from fixing the state of one processor so that a maximum number of cells are read or written, the third term comes from fixing the contents of up to τ'_i processors so that the maximum contention (τ'_i) occurs. So $\deg(\chi_{A_i}) \leq (2 + \tau'_i)b_{i-1}$.

The degree of a function describing the state of a processor is bounded by $b_{i-1} + \tau_i \cdot b_{i-1} + \deg(\chi_{A_i}) \leq (3 + \tau_i + \tau'_i)b_{i-1}$. Similary the degree of a function describing the contents of a memory location at the end of the ith phase is no more than $b_{i-1} + \tau'_i b_{i-1} + \deg(\chi_{A_i}) \leq (3 + 2\tau'_i)b_{i-1}$. Thus $b_i = (3 + \tau_i + 2\tau'_i)b_{i-1}$ is an upper bound on the degree of χ_{A_i} and the functions describing the states of processors and contents of cells at phase i.

Let $\tau''_i = \max\{\lceil \tau_i/\alpha \rceil, \lceil \tau'_i/\beta \rceil\}$. Assume the machine halts after phase l. Then the computation time $T \geq \mu(\tau''_1 + \ldots + \tau''_l)$. Since $\tau_i \geq 1$ and $\tau'_i \geq 1$, we have $T \geq l\mu$. Also, the degree of the function specifying the contents of the output cell should be at least r at termination since the degree of a function f that computes the parity of r bits is r [14]. Hence,

$$\begin{aligned} r &\leq b_l \\ &\leq \prod_{j=1}^{l} (3 + \tau_j + 2\tau'_j) \\ &\leq \prod_{j=1}^{l} (6\mu\tau''_j) \end{aligned}$$

$$\begin{aligned} &\leq \prod_{j=1}^{l} (6\mu)^{\tau_j''} \\ &\leq (6\mu)^{(\tau_1''+\tau_2''+\cdots+\tau_l'')} \\ &\leq (6\mu)^{T/\mu}. \end{aligned}$$

It follows that $T = \Omega(\mu \log r / \log \mu)$. □

Corollary 3.1 *Any deterministic algorithm for the n-element Parity problem requires time $\Omega((g/\log g)\log n)$ on the* QSM*, $\Omega(g \log n)$ time on the s-*QSM*, and $\Omega((L/\log(L/g))\log q)$ time on a p-processor* BSP*, where $q = \min\{n, p\}$.*

For randomized algorithms for parity, we obtain two types of lower bounds. The first is a lower bound on the GSM, which gives us the strongest result we are able to obtain for the BSP. The second type of result is an adaptation of CRCW lower bounds to the queuing models. We are able to obtain stronger bounds for the QSM and s-QSM by this latter method. Since this type of lower bound does not hold for a 'strong queuing' model, it does not adapt to the GSM or the BSP.

Our first lower bound is obtained by an adaptation of the lower bound for randomized parity on the SIMD-QRQW from [14].

Theorem 3.2 *Computing the parity of n bits on a* GSM *by a randomized algorithm requires $\Omega(\mu\sqrt{\frac{\log(n/\gamma)}{\log\log(n/\gamma)+\log\mu}})$ time.*

Proof: Let $r = n/\gamma$. We assume a uniform probability distribution on the inputs, and prove that any deterministic algorithm will take $\Omega(\mu \cdot \sqrt{\log r/(\log\log r + \log\mu)})$ time on more than half of the inputs. By Yao's theorem this will give us a lower bound on the running time of randomized algorithms for this problem.

We let $\tau = \sqrt{\log r/(\log\log r + \log\mu)}$ and let $\nu = \mu\tau$. Our goal is to prove a lower bound of $\Omega(\nu)$. Our lower bound proof will fix the values of certain input variables during the computation in order to maintain the following invariants at the end of phase t:

1. Each processor and memory cell knows at most one unfixed input variable.
2. The number of processors and memory cells that know an unfixed input variable x_i is no more than $k_t \leq \nu^t$.
3. Let V_t be the set of unfixed input variables at the end of phase t. Then, $|V_t| \geq |V_{t-1}|/(5\nu k_t)$.
4. The values revealed for variables not in V_t have been chosen to maximize the number of correct answers for settings of variables in V_t.

If the number of possible read-writes by a processor P is more than $\nu_1 = \alpha \cdot \tau$ in a given phase, then for at least half of the possible settings for the variables in V_t, P will perform more than ν_1 read-writes in that phase, and the running time of the algorithm will exceed the stated bound for these settings. By invariant 4, the algorithm will fail with probability at least 1/2. Hence we can assume that every processor performs no more than ν_1 read-writes in each phase.

Similarly, suppose the possible number of reads or writes at a memory cell m from processors that know different inputs is more than $\nu_2 = 2\beta\tau$ in a given phase. We assume that m is read or written to by a given processor at most once in a phase (since otherwise we can convert this algorithm into another one which performs the second access to m as a local operation and hence would run faster). Let B be the set of variables in V_t that are known by those processors accessing m in phase t. Thus, for at least half of the possible settings of the variables in V_t, at least half of the bits in B will be set so as to cause the corresponding processors to access memory cell m. Thus

the running time of the algorithm will exceed ν for these settings, and hence the algorithm will fail with probability at least 1/2. Hence we can assume that every memory cell is accessed by no more than ν_2 processors (that know different inputs) in each phase.

We now show how we fix variables in V_t after the reads and writes in phase $t+1$ while maintaining our four invariants. The first invariant is violated at a processor if it reads a cell that knows a variable that is different from the one known by the processor; it is violated at a cell if it is written into by a processor that knows a different variable than that known by the cell. We construct an undirected graph G on the vertex set V_t such that there is an edge between vertices corresponding to variables x_i and x_j iff a processor that knows x_i (x_j) either reads or writes a cell that knows x_j (x_i). The degree of any vertex in this graph is

$\leq 2\max(2\nu_1 k_t, \nu_2 k_t) \leq 4\nu k_t$.

Here the term $2\nu_1 k_t$ within the max represents the number of edges inserted for processors that know a variable x_i, and the term $\nu_2 k_t$ represents the number of edges inserted for memory cells that know x_i, and the factor of 2 arises from the case when we interchange x_i and x_j.

Let $u_t = 5\nu k_t$. The maximum degree of vertices in G is no more than $4\nu k_t$, hence this graph has an independent set of size at least $|V_t|/(\deg(G)+1) \geq |V_t|/u_t$. We find an independent set I of this size, and we fix the values of all variables in $V_t - I$ with values such that the number of correct answers for variables in I is maximized. Let $k_{t+1} = \nu \cdot k_t$. The number of processors and memory cells that know a given variable in I after the read step is no more than k_{t+1}, and each processor and memory cell knows at most one variable in I. Let $V_{t+1} = I$.

We note the following properties hold at the end of phase $t+1$:

1. Each processor and memory cell at the end of phase $t+1$ knows at most one variable in V_{t+1}.

2. The number of processors and memory cells that know a given variable in V_{t+1} is at most $k_{t+1} = \nu k_t$.

3.

$$\begin{aligned} |V_{t+1}| &\geq \frac{|V_t|}{(5\nu k_t)} \\ &\geq \frac{r}{((5\nu)^{t+1} \cdot \prod_{i=1}^{t} k_i)} \\ &\geq \frac{r}{((5\nu)^{t+1} \prod_{i=1}^{t} \nu^i)} \\ &\geq \frac{r}{((5\nu)^{t+1} \nu^{t(t+1)/2})}. \end{aligned}$$

4. We fixed the variables in $V_t - V_{t+1}$ to maximize the number of correct answers for settings of variables in V_{t+1}.

Hence we have re-established the four invariants at the end of phase $t+1$.

The output cell cannot contain the correct answer with probability greater than half as long as $|V_T| > 1$. Hence,

$$r \leq (5\nu^{(1+t/2)})^{t+1} \quad \text{or} \quad t = \Omega(\sqrt{\frac{\log r}{\log \nu}})$$

Since each phase takes time at least μ, any randomized GSM algorithm that solves the parity problem with probability at least $1/2+\epsilon$, $\epsilon > 0$ must take time $\Omega(\mu\sqrt{\log r/(\log\log r + \log\mu)})$. □

Corollary 3.2 *Computing parity of n bits on a* BSP *by a randomized algorithm that succeeds with probability at least $1/2+\epsilon$, $\epsilon > 0$, requires $\Omega(L\sqrt{\frac{\log q}{\log\log q + \log(L/g)}})$ time, where $q = \min\{n, p\}$.*

For the QSM and s-QSM we have the following stronger lower bounds, which are obtained by using the result in [1] on adapting CRCW PRAM lower bounds to the QSM and the lower bound of Beame and Hastad [3] on computing parity on the CRCW PRAM.

Theorem 3.3 *Any randomized algorithm for the n-element Parity problem on a p-processor* QSM *requires time* $\Omega(\frac{g \log n}{\log\log n + \min(\log\log p,\ \log\log g)})$.

Proof: By using the result in [1] on adapting CRCW PRAM lower bounds to the QSM and the lower bound of Beame and Hastad [3] on computing parity on the CRCW PRAM using p processors, we obtain a lower bound of $\Omega(g \log n / \log\log p)$ for computing parity with a randomized algorithm on a QSM. This gives the stated lower bound if p is polynomial in n. If p is superpolynomial in n, we note that in a sequence of $T = \Theta(\log n / \log\log n)$ memory request steps by processors (each of which takes g units of time), at most g^T processors can obtain information about any single input bit. Thus the number of processors q that are affected by any input bit is bounded by $q \leq n \cdot g^{\Theta(\log n / \log\log n)}$. By adaptation of the Beame-Hastad [3] lower bound to QSM given in [1], this implies a lower bound of $\Omega(\log n / \log\log q)$, i.e., $\Omega(\frac{g \log n}{\log\log n + \log\log g})$ to compute parity by a randomized algorithm on the QSM. We obtain the desired result by combining the results for the case when p is polynomial in n and the case when p is not polynomial in n. □

Corollary 3.3 *Any randomized algorithm for the n-element Parity problem on an s-*QSM *requires time* $\Omega(g \log n / \log\log n)$.

For the number of rounds needed to compute parity, the results obtained for the OR function (later in this paper) in Theorem 6.2 and its corollary apply. However, in the following theorem and its corollary we are able to strengthen the lower bound for the number of rounds on a QSM.

Theorem 3.4 *The number of rounds needed to compute parity of n bits on a p processor* QSM*, $p < n$, by a deterministic algorithm is* $\Omega(\frac{\log n}{\log(n/p) + \min\{\log g,\ \log\log p\}})$

Proof: Note that for $p < \sqrt{n}$, the lower bound is constant, and thus trivially satisfied.

For $p \geq \sqrt{n}$, first we prove a lower bound of $\Omega(\frac{\log n}{\log(n/p) + \log\log p})$. To do this, we change the proof of Theorem 4.1, part (b) of Beame and Hastad [3]. The major changes are that the degrees of the partitions of the processors after a read become $(1 + n/p)s$, instead of $2s$, and the number of different cells that could be accessed becomes $2^d(n/p)p$ instead of $2^d p$, where d is the degree of the processor partitions. We perform two reductions, one with $q = 1/(96(n/p)s)$ after the read step to reduce processor partitions to degree s, and one with $q = 1/(48s)$ after the write step to reduce cell partitions to degree s. We need $s = 2\log 4p$ for the probabilities to work out as in [3], along with the fact that $p > \sqrt{n}$. After T steps we must have $s \geq \frac{1}{48} n((96(n/p)s)(48s))^{-(T-1)}$, and the lower bound of $T = \Omega(\frac{\log n}{\log(n/p) + \log\log p})$ follows.

Next, a lower bound of $\Omega(\frac{\log n}{\log(n/p) + \log g})$ follows from the lower bound given in Corollary 7.3 for the number of rounds need to compute OR. □

Corollary 3.4 *The number of rounds needed to compute parity of n bits on a p processor* QSM*, $p < n$, by a randomized algorithm is* $\Omega(\frac{\log n}{\log(n/p) + \min\{\log g,\ \log\log p\}})$

Proof: A lower bound of $\Omega(\log n / \log(gn/p))$ follows from Corollary 7.3 (for the OR function). The remaining part of the lower bound is obtained using [3] and extending it to randomized algorithms using either the Random Adversary technique, or [2]. □

We note that the lower bounds we have obtained for the Parity problem imply corresponding lower bounds for other problems such as list ranking and sorting, since there are simple size-preserving reductions from parity to these other problems.

4 The Random Adversary

Our lower bounds for Load Balancing and the OR function use the Random Adversary technique [16].

We start by reviewing this method.

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, 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.

4.1 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 $f' \leq f$.)

4.2 RandomSet Procedure

We will assume the distribution chosen is $\mathcal{D}$. Function RANDOMSET is 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

Fact 4.1 ([17]) *Assuming f generated solely by calls to RANDOMSET, then f will be generated according to the distribution $\mathcal{D}$.*

4.3 REFINE and GENERATE

Say f is t-good if it satisfies certain properties, which will be defined with respect to the time t, the problem P, and the input distribution $\mathcal{D}$. 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 allegedly 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) consisting 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".

Assertion 4.1 *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 $f_0 = f_* \geq f_{t_1} \geq \cdots \geq f_{t_j} \geq f$ where $(f_{t_i}, x) = \text{REFINE}(t_{i-1}, f_{t_{i-1}})$, $t_i = t_{i-1} + x$, $t_{j-1} < T \leq t_j$, 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_j}. Then we need to prove the following lemma.

Lemma 4.1 *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 Fact 4.1, Lemma 4.1 will always hold.

```
Function GENERATE
    Let f_0 = f_*
    Let f = f_0
    Let t = 0
    While t ≤ T Do
        Let (f, x) = REFINE(t, f)
        Let t = t + x
        Let f_t = f
    Let P = {p | f(p) = '*'}
    Return RANDOMSET(f, P)
```

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 parallel step.

Lemma 4.2 ([17]) *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 Assertion 4.1, 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 Assertion 4.1.

5 A General GSM Lower Bound

Here we prove a general lower bound that will be used in proving lower bounds on Load Balancing and related problems.

5.1 GSM Definitions

Recall that a big-step of a GSM takes μ time. For our lower bound proofs, we may assume that each phase of a GSM takes at least one big-step, since computation will be considered free.

Let A be any deterministic algorithm for the GSM. 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 by p in a GSM step ending at big-step j, if any, and λ_j is the null symbol otherwise. Trace(c, t, f) is defined to be the tuple $< c, \lambda_t >$, where λ_t is the contents of cell c at big-step t.

Let e be any partial input map. Let v be a processor or cell. For each possible trace x, let $S(v, t, e, x) = \{f | f \subseteq e$ and Trace$(v, t, f) = x\}$ (i.e. $S(v, t, e, x)$ is the set of input maps which cause the trace x for v). Let States$(v, t, e) = \{S(v, t, e, x) | S(v, t, e, x) \neq \emptyset\}$. Note that |States$(v, t, e)$| is the number of different traces of v at big-step t with input maps refined from e. Define Know(v, t, e) as the minimum set of inputs such that for any input maps f_1 and f_2 that refine e and have $f_1(q) = f_2(q)$ for all $q \in$ Know(v, t, e), 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, e), since these could not affect its trace, and v is dependent on every input inside Know(v, t, e) by the fact that it is the minimum set of inputs which could affect its trace.) Let AffProc(i, t, e) contain each processor p in which $i \in$ Know(p, t, e). Let AffCell(i, t, e) contain each cell c in which $i \in$ Know(c, t, e). Define Cert(v, t, f) as the minimum set of inputs (and lexicographically smallest, if there are more than one) such that for any input map f' such that $f(q) = f'(q)$ for all $q \in$ Cert(v, t, f), Trace(v, t, f) is the same as Trace(v, t, f'). Note that when $f \subseteq e$, Cert$(v, t, f) \subseteq$ Know(v, t, e), but they are not necessarily the same.

5.2 General Lower Bound

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

Assume $\mu < (\log n)^{1/8}$ and $\gamma < \log n$. (If not, the lower bound that we wish to prove reduces to $\Omega(\mu)$, which is trivial to prove, since at least one read is required to solve the problem.) Assume the set of possible values for inputs is $\{0, 1\}$. Assume $|I| = n\rho$, i.e., the number of input bits is $n\rho$, and assume that each input bit initially affects exactly one cell, and each cell is initially affected by at most $\nu = \gamma\rho$ input bits. Without loss of generality, assume n is large enough so that our analysis holds.

Consider an input probability distribution with the following properties: (1) every input map is possible, and (2) 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 a given value is at least $q \geq (\log n)^{-1}$.

We define the following values for $i \geq 0$: $d_i = \nu(\mu + 1)^{2i}$, $k_i = 2^{\nu(\mu+1)^{4(i+1)}}$, and $r_i = in^{\frac{2}{3}}$.

If $T \leq \frac{(1/8)\log\log n - \log \nu}{2\log(\mu+1)}$ then the following is easily proved.

Fact 5.1 *$d_T \leq (\log n)^{1/8}$, $k_T \leq 2^{\sqrt{\log n}}$, 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, deg(States$(v, t, f)) \leq d_t$,

(2) For each processor or cell v, $|\text{States}(v,t,f)| \leq k_t$,
(3) For each processor or cell v, $|\text{Know}(v,t,f)| \leq k_t$,
(4) For each input i, $|\text{AffProc}(i,t,f)| \leq k_t$ and $|\text{AffCell}(i,t,f)| \leq k_t$, and
(5) f maps at most r_t inputs to something other than '$*$'.

One can verify that for each processor or cell v, $\deg(\text{States}(v,0,f_*)) \leq \nu$, $|\text{States}(v,0,f_*)| \leq 2^\nu$, and $|\text{Know}(v,0,f_*)| \leq 2^\nu$, and for each input i, $|\text{AffProc}(i,0,f_*)| = 0$ and $|\text{AffCell}(i,0,f_*)| \leq 2^\nu$. Therefore the input map f_* is 0-good.

We now describe algorithm REFINE which is called with a big-step 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 number of big-steps taken by the phase. Note that x will be the true number of big-steps taken by the phase if $t + x \leq T$. The random refinement is based on the action of algorithm A in the phase. The intuition behind this REFINE procedure is the following. First, in lines (4) through (10), we force some processor that possibly accesses many cells in this phase 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 on the number of big-steps in the phase. Next, in lines (12) through (21), we force some cell that is possibly accessed by many processors, to actually be accessed by those processors, or at least $\mu \log\log n$ of them if there are more than $\mu \log\log n$ 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) In this REFINE procedure we will force the number of big-steps required for the phase to correspond to the "amount of information exchanged" in the phase, without fixing too many inputs.

We define $\text{MaxCell}(t,e)$ as the cell with the maximum possible contention at big-step t for any input map which refines e. We define $\text{MaxRWC}(c,t,e)$ as the maximum possible contention at c at big-step t for any input map which refines e. We define $\text{MaxCertCell}(c,t,e)$ as the lexicographically smallest input map $f \leq e$ which causes the maximum possible contention at c at big-step t. We define $\text{MaxProc}(t,e)$ as the processor with the maximum possible number of reads or writes at big-step t for any input map which refines e. We define $\text{MaxRWP}(p,t,e)$ as the maximum possible number of reads or writes p makes at big-step t for any input map which refines e. We define $\text{MaxCertRWP}(p,t,e)$ as the lexicographically smallest input map $f \leq e$ which causes p to read or write $\text{MaxRWP}(p,t,e)$ cells at big-step t. Let $\text{ACCESS}(c,t,e)$ be the set of processors that read from or write to cell c at big-step t for any input map which refines e.

Function REFINE(t,f)
(1) Let $e = f$
(2) Let Done = FALSE
(3) Let MaxCountRW $= 0$
(4) While not Done
(5) Let $p = \text{MaxProc}(t,e)$
(6) Let $h = \text{MaxCertRWP}(p,t,e)$
(7) Let $e = \text{RANDOMSET}(\text{Cert}(p,t,h), e)$
(8) If for all $i \in \text{Cert}(p,t,h)$, $h(i) = e(i)$
(9) Let MaxCountRW $= \text{MaxRWP}(p,t,e)$
(10) Let Done = TRUE
(11) Let Done = FALSE
(12) While not Done
(13) Let $c = \text{MaxCell}(t,e)$

(14) Let $h = \text{MaxCertCell}(c, t, e)$
(15) Let $W = \text{ACCESS}(c, t, h)$
(16) Choose $W' \subseteq W$ such that $|W'| = \min\{|W|, \mu \log\log n\}$
(17) Let $V = \bigcup_{p \in W'} \text{Cert}(p, t, h)$
(18) Let $e = \text{RANDOMSET}(V, e)$
(19) If for all $i \in V$, $h(i) = e(i)$
(20) Let MaxContention $= |W'|$
(21) Let Done = TRUE
(22) Let $f' = e$
(23) Return $(f', \max\{\lceil \text{MaxContention}/\beta \rceil, \lceil \text{MaxCountRW}/\alpha \rceil\}$

Claim 5.1 *If f is t-good and* REFINE(t, f) *returns (f', x), then either $x \geq \log\log n$ and is a lower bound for the number of big-steps required for the phase, or x is the exact number of big-steps required for the phase.*

Proof: Lines (4) through (10) set MaxCountRW to the (exact) maximum number of cells read or written to by a processor during this step.

If the maximum contention is less than $\mu \log\log n$, lines (12) through (21) set MaxContention to the (exact) maximum contention at any cell. Then x is the maximum of $\lceil \text{MaxCountRW}/\alpha \rceil$ and $\lceil \text{MaxContention}/\beta \rceil$, which is the exact time required for the step. Otherwise (i.e., if the maximum contention is at least $\mu \log\log n$), the MaxContention will be $\mu \log\log n$ and x will be the maximum of MaxCountRW and $\log\log n$, which is at least $\log\log n$ and a lower bound on the time required for the step. □

Lemma 5.1 *If f is t-good and* REFINE(t, f) *returns (f', x), then either $t + x > T$ or*
(1) For each processor or cell v, $\deg(\mathit{States}(v, t+x, f')) \leq d_{t+x}$,
(2) For each processor or cell v, $|\mathit{States}(v, t+x, f')| \leq k_{t+x}$,
(3) For each processor or cell v, $|\mathit{Know}(v, t+x, f')| \leq k_{t+x}$, and
(4) For each input i where $f'(i) =$ '$*$', $|\mathit{AffProc}(i, t+x, f')| \leq k_{t+x}$ and $|\mathit{AffCell}(i, t+x, f')| \leq k_{t+x}$.

Proof: Assume that $t + x \leq T$, and thus $x < \log\log n$.

The bounds we show below follow from the fact that for $x \geq 1$, $\mu x + 1 \leq (\mu + 1)^x$. It is possible to have $x = 0$, and it is easy to check that our bounds hold in that case also. An additional note is that there are at most $\beta x k_t^2$ processors that possibly read (equivalently, possibly write to) any given cell for any input map that refines f'. Otherwise, there would be $\beta x + 1$ processors that were affected by totally disjoint sets of inputs that all possibly read (possibly write to) that cell, and thus some refinement of f' would force all $\beta x + 1$ processors to read (write to) that cell. This is impossible by Claim 5.1. For a read step, the trace of processor v depends on its original trace τ, plus the traces of the $y \leq \alpha x$ cells $c_1, \ldots, c_y$ that it read, $\tau_1, \ldots, \tau_y$. Say the new trace is τ'. Then $S(v, t+x, f', \tau') = S(v, t, f, \tau) \cap S(c_1, t, f, \tau_1) \cap \cdots \cap S(c_y, t, f, \tau_y)$, and

$$\begin{aligned} & \deg(\chi(S(v, t+x, f', \tau'))) \\ &= \deg(\chi(S(v, t, f, \tau)) \cdot \prod_{i=1}^{y} \chi(S(c_1, t, f, \tau_i))) \\ &\leq \deg(\chi(S(v, t+x, f, \tau'))) + \sum_{i=1}^{y} \deg(\chi(S(c_1, t, f, \tau_i))). \end{aligned}$$

By induction, each term is at most d_t, and this implies that $\deg(\text{States}(v, t+x, f')) \leq (\alpha x+1)d_t \leq d_{t+x}$.

Processor v reads one of at most k_t sets of at most αx cells at step t, each one being in one of at most k_t states. Thus $|\text{States}(v, t+x, f')| \leq k_t^{\alpha x+1} \leq 2^{\nu(\mu+1)^{4(t+x+1)}} = k_{t+x}$.

The inputs that affect processor v are the k_t that originally affect it plus the k_t that affect each of the $\alpha x k_t$ possible cells it reads. Thus $|\text{Know}(v, t+x, f')| \leq k_t + \alpha x k_t k_t \leq 2^{\nu(\mu+1)^{4(t+x+1)}} = k_{t+x}$.

An input affects all the processors it originally affects, plus all the processors that possibly read from the cells that it affects. Thus $|\text{AffProc}(v, t+x, f')| \leq k_t + \beta x k_t^2 k_t \leq 2^{\nu(\mu+1)^{4(t+x+1)}} = k_{t+x}$.

Say the trace of cell v after the step is τ, and let $h \in S(v, t+x, f', \tau)$. Let $G = \{S(p, t, f', \tau^*) | \tau^* = \text{Trace}(p, t, h')$ and p writes to v at time t on input map $h' \leq f'\}$. We consider two cases.

Case 1 $h \in S$ for some $S \in G$. Let $P = \{p : p \text{ writes to } c \text{ on input map } h\}$. For $p \in P$, say $h \in S(p, t, f', \tau_p^*) \in G$, and let $S_p^* = S(p, t, f', \tau_p^*)$. Let $G' = \{S(p', t, f', \tau') \in G | p' \notin P\}$. Then $S(v, t+x, f', \tau) = \bigcap\{S_p^* : p \in P\} \cap \bigcap\{\overline{S} | S \in G'\}$. By the inclusion-exclusion principle,

$$\chi(\bigcap\{\overline{S} | S \in G'\}) = \sum (-1)^{\mathcal{A}} \prod_{S \in \mathcal{A}} \chi(S),$$

where the sum is over all $\mathcal{A} \subseteq G'$. Further, since at most βx processors write to v, for all $h' \in \bigcap\{S_p^* : p \in P\}$, at most $\beta x - |P|$ many $S \in G$ contain h'. Thus $\prod_{p \in P} \chi(S_p^*) \cdot \prod_{S \in \mathcal{A}} \chi_S = 0$ if $|P| + |\mathcal{A}| \geq \beta x$. Thus

$$\begin{aligned} &\deg(\chi(S(v, t+x, f', \tau))) \\ &\leq \deg(\prod_{p \in P} \chi(S_p^*) \sum (-1)^{\mathcal{A}} \prod_{S \in \mathcal{A}} \chi_S) \\ &\leq \max\{\sum_{p \in P} \deg(\chi(S_p^*) + \sum_{S \in \mathcal{A}} \deg(\chi_S)\}, \end{aligned}$$

where the sum in the first inequality and the max in the second inequality is taken over all $\mathcal{A}$ where $|\mathcal{A}| \leq \beta x - |P|$. By induction, each term is at most d_t, so $\deg(\text{States}(v, t+x, f')) \leq \beta x d_t \leq d_{t+x}$.

Case 2 $h \notin S$ for any $S \in G$. By a similar argument, $\deg(\text{States}(v, t+x, f')) \leq \deg(\text{States}(v, t, f)) + \beta x d_t \leq (\beta x + 1)d_t \leq d_{t+x}$.

Cell v is possibly written to by at most $\beta x k_t^2$ 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 possibly write to it. Thus $|\text{States}(v, t+x, f')| \leq k_t + \beta x k_t^2 k_t \leq 2^{\nu(\mu+1)^{4(t+x+1)}} = k_{t+x}$.

The inputs that affect a cell v are the at most k_t that originally affect it plus the k_t that affect each of the at most $\beta x k_t^2$ processors that possibly write to it. Thus $|\text{Know}(v, t+x, f')| \leq k_t + \beta x k_t^2 k_t \leq 2^{\nu(\mu+1)^{4(t+x+1)}} = k_{t+x}$.

An input affects all the cells it originally affects, plus the $\alpha x k_t$ cells possibly written to by each processor thatit affects. Thus $|\text{AffCell}(v, t+x, f')| \leq k_t + \alpha x k_t k_t \leq 2^{\nu(\mu+1)^{4(t+x+1)}} = k_{t+x}$. □

Lemma 5.2 *If f is t-good and REFINE(t, f) is successful and returns (f', x), then either $t+x > T$ or f' is $(t+x)$-good.*

Proof: This follows from Lemma 5.1, the definition of successful, and the definition of t-good. □

Claim 5.2 *If f is t-good then the probability of a processor or cell being in any given state at time $t < T$ is at least $q^{\sqrt{\log n}}$.*

Proof: Consider a processor or cell v, a state of v, and an input map $h \leq f$ which causes the trace corresponding to that state. By setting each input $i \in \text{Cert}(v,t,h)$ to $h(i)$, the state would occur. The probability of this state occurring is at least q^β where $\beta = |\text{Cert}(v,t,h)|$. But by Fact 2.3, $|\text{Cert}(v,t,h)| \leq (\deg(\text{States}(v,t,f)))^4 \leq d_t^4 \leq \sqrt{\log n}$. □

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

Lemma 5.3 *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 (4) through (10). Since by Claim 5.2, 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 executing this loop more than $\sqrt{n}$ times is at most

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

Then since $\text{Cert}(p,t,h) \leq \sqrt{\log n}$ for any input map h, the probability of more than $\sqrt{n}\log n$ inputs being set is at most n^{-3}.

Now consider the While loop at lines (12) 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}.

For any iteration of this While loop, the probability of finishing is the probability that at most $(\mu \log\log n)\max_p\{|\text{Cert}(p,t,f^*)|\}$ inputs are set according to f^*. The probability of this is at least $q^{(\mu\log\log n)\sqrt{\log n}} \geq 2^{-\log^{\frac{2}{3}} n}$. Thus the probability of not finishing in any of the first $\sqrt{n}$ iterations is less than $(1 - 2^{-\log^{\frac{2}{3}} n})^{\sqrt{n}} \leq n^{-3}$.

Note that if we do finish in the first $\sqrt{n}$ iterations, we set at most $\sqrt{n}(\mu\sqrt{\log n}\log\log n)$ inputs.

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

Lemma 5.4 *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 5.3 and Lemma 5.2. □

Corollary 5.1 *With probability at least $1 - n^{-1}$, for any processor or cell v, and any $t \leq T$, $\deg(States(v,t,f_t)) \leq d_T \leq (\log n)^{1/8}$; $|Know(v,t,f_t)| \leq k_T \leq 2^{\sqrt{\log n}}$; for any unset input i and any time $t \leq T$, $|AffProc(i,t,f_t)| \leq k_T \leq 2^{\sqrt{\log n}}$, $|AffCell(i,t,f_t)| \leq k_T \leq 2^{\sqrt{\log n}}$; and the number of inputs set by RANDOMSET is at most $r_T \leq Tn^{\frac{2}{3}}$.*

6 Load Balancing

Consider the following variant of the load-balancing problem:

Chromatic Load Balancing (CLB) Let $m \geq 1$, and let Q be a set of $8m$ colors. Assume that there is an input array of size $n \times 4m$, holding n groups of $4m$ objects each, and each group of objects is randomly assigned a color from Q. Then the *chromatic load-balancing* problem is to choose any color and distribute all objects of that color into an $n \times m$ array, holding n groups of m objects each. (The groups in the output need not respect the input grouping.)

We will prove a lower bound for the CLB problem and then use this lower bound to establish lower bounds for Load Balancing, LAC (Linear Approximate Compaction) and Padded Sort.

6.1 Chromatic Load Balancing Lower Bound Proof

Without loss of generality, we will assume the objects are tagged with their original group (row) number and their original rank (column) (1 to $4m$) within that group.

Enhanced Chromatic Load Balancing (ECLB) The *enhanced chromatic load-balancing* problem is the same as the CLB problem with the added requirement that for each cell in the input array, there must be a pointer to the destination row (group number) of the object in the output array.

Claim 6.1 *Given a solution to the CLB problem, one can construct a solution to the ECLB problem on a* GSM *in m additional steps.*

Proof: Assign one processor per destination row (of the CLB solution) to step through the at most m objects assigned to that group. For each object with tag (group,rank), have the processor write that destination in the input array at location (group,rank). □

Lemma 6.1 *Any deterministic algorithm which solves the ECLB problem with probability at least $\frac{1}{4}$ on a* GSM *requires $\frac{(1/8)\log\log n - \log 8m\gamma}{2\log(\mu+1)}$ big-steps.*

Proof: Consider a deterministic algorithm that allegedly solves the ECLB problem with probability at least $\frac{1}{4}$ in $t < T$ big-steps. From Corollary 5.1, assuming $\nu = 8m\gamma$, with probability at least $1 - n^{-1}$, for any processor or cell v, $\deg(\text{States}(v,t,f_t)) \leq d_t \leq (\log n)^{1/8}$; $|\text{Know}(v,t,f_t)| \leq k_t \leq 2^{\sqrt{\log n}}$; for any unset input i $|\text{AffCell}(i,t,f_t)| \leq k_t \leq 2^{\sqrt{\log n}}$; and the number of inputs bits set by RANDOMSET is at most $Tn^{\frac{2}{3}}$ For now, we assume all these conditions hold.

We call RANDOMSET to set the input bits associated with any color that has at least one input bit set. Notice that at most $Tn^{\frac{2}{3}}\nu \leq \frac{n}{4}$ colors will be fixed. For the remainder of the analysis, we will discuss colors instead of bits.

Let $f_{(q)} \leq f_t$ be the input map where $f_{(q)}(i) = q$ whenever $f_t(i) =$ '$*$', i.e., $f_{(q)}$ designates the color q for each group with an unset color. (The fact that $f_{(q)}$ is not a relevant input map does not matter in the argument that follows.) Consider an input cell c whose group color 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 the color q to all inputs in $\text{Cert}(c,t,f_{(q)})$. 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 $(3n/4)(4m) = 3nm$ and a range of size at most n. By a simple counting argument, we can find $\frac{n}{2}$ disjoint sets of $m+1$ input cells, each of which point to the same destination group.

Let S be one of the disjoint sets of input cells that we have just found. Each of the $m+1$ cells $c \in S$ has $\text{Cert}(c,t,f_{(q)}) \leq (d_t)^4 \leq \sqrt{\log n}$, and thus at most $2(m+1)\sqrt{\log n}$ inputs affect the contents of these cells, assuming those inputs are all set to q. Each of these inputs i has $\text{AffCell}(i,t,f_t) \leq k_t \leq 2^{\sqrt{\log n}}$, so at most $(m+1)(2\sqrt{\log n})2^{\sqrt{\log n}}$ cells are affected by the same inputs that affect the cells in S. Thus from the $\frac{n}{2}$ disjoint sets of $m+1$ input cells which are mapped by F to the same output cell, we can find a subset of $B = n^{2/3}$ sets whose cell contents are completely independent. Number these sets from 1 to B.

Claim 6.2 *With high probability, at least one of the B sets uses the same pointers as F.*

Proof: We can see that the probability of all cells in one of these sets using the pointers from F is at least the probability that $f(i) = q$ for all $i \in \text{Cert}(c,t,f_{(q)})$ for each c in the set. 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 $(8m)^{-2\sqrt{\log n}}$. (Note that the number of inputs we are conditioning on is at most $(B-1)2\sqrt{\log n} + \frac{1}{16}n^{\frac{2}{3}} \log\log n \leq \frac{n}{2\log\log n}$.)

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 set of input cells will be mapped to the same output cell. □

By Claim 6.2, with very high probability the mapping provided by the algorithm at this point will not be a valid solution to ECLB, and this is true for any choice of q. This is assuming that the conditions implied by Corollary 5.1 hold, and that the number of items is at most $\frac{n}{m}$. But these conditions hold with probability at least $1-2n^{-1}$. Thus with high probability, the mapping provided by the algorithm at this point will not be a valid solution to ECLB. This proves the lemma. □

Lemma 6.2 *A deterministic algorithm which solves the CLB problem with probability at least* $\frac{1}{4}$ *on a* GSM *requires* $\Omega(\mu(\frac{(1/8)\log\log n - \log 8m\gamma}{2\log(\mu+1)} - m)$ *time.*

Proof: Given a deterministic algorithm that solves the CLB problem with probability at least $\frac{1}{4}$ in $\frac{(1/8)\log\log n - \log 8m\gamma}{2\log(\mu+1)} - m$ big-steps, From Claim 6.1 we could solve the ECLB problem with probability at least $\frac{1}{4}$ in $\frac{(1/8)\log\log n - \log 8m\gamma}{2\log(\mu+1)}$ big-steps. This is impossible by Lemma 6.1. The lemma follows from the fact that each big-step takes μ time. □

6.2 Applications of the Chromatic Load Balancing Lower Bound

We apply the lower bound obtained in the previous section to the following problems.

Load Balancing Given h objects distributed among n processors, redistribute the objects so that each processor gets $O(1+h/n)$ objects.

Padded Sort (or **Padded** $U[0,1]$ **Sort**) Given n values taken from a uniform distribution over the unit interval $[0,1]$, arrange them in sorted order in an array of size $n+o(n)$, with the value NULL in all unfilled locations.

Linear Approximate Compaction (**LAC** or h-**LAC**) 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 $O(h)$.

Theorem 6.1 *Solving the Load Balancing problem, the LAC problem, or the Padded Sort problem with probability at least* $\frac{1}{2}$ *requires* $\mu(\frac{(1/8)\log\log n - \log\gamma}{2\log\mu} - O(m))$ *time on a Randomized* GSM.

Proof: Load balancing: Assume there is an algorithm that solves Load Balancing with probability at least $\frac{1}{2}$ on the QSM in expected time t. Then by Yao's Theorem, for any input distribution, there is a deterministic algorithm which solves Load Balancing over that distribution with the same probability in time t. Consider the Chromatic Load-Balancing problem (with $m = \log\log\log\log n$) and choose one of the $8m$ colors. Let $\mathcal{D}$ be the input distribution of the objects of that color, and let A be the algorithm given by Yao's theorem for distribution $\mathcal{D}$. We can then solve the Chromatic Load-Balancing problem using the following procedure. First run A for the objects of the chosen color, with each processor taking the objects of a single row. Without loss of generality, for some constant C assume A assigns at most $C(1+h/n)$ objects to each processor, when h objects are

given as input. Also assume n is large enough so that $2C < m$. If at most m objects are assigned to each processor then one can easily assign each processor's objects to a destination group.

On average, there will be $4nm/8m$ objects of the chosen color, and with very high probability, there will be at most n objects of that color. If there are at most n objects of that color, at most $2C$ of them will be assigned to any one processor by A. Thus the Chromatic Load-Balancing problem can be solved with probability at least $\frac{1}{4}$ in the same asymptotic time as A. By Lemma 6.1, $t = \mu(\frac{(1/8)\log\log n - \log 8m\gamma}{2\log\mu} - O(m))$.

LAC: Assume there is an algorithm that solves LAC with probability at least $\frac{1}{2}$ on the QSM in time t. Then by Yao's Theorem, for any input distribution, there is a deterministic algorithm which solves Compaction over that distribution in expected time t. Consider the Chromatic Load-Balancing problem (with $m = \log\log\log\log n$) and choose one of the $8m$ colors. Consider an item to be a group of objects of that color, and let $\mathcal{D}$ be the input distribution of the items. Let A be the algorithm given by Yao's theorem for distribution $\mathcal{D}$. We can then solve the Chromatic Load-Balancing problem using the following procedure. First run A with $h = n/4m$. Without loss of generality, for some constant C assume A inserts the items into an array of size Ch, when h is the parameter given in the definition of Compaction and the input consists of at most h items. Also assume n is large enough so that $C < m$. If A succeeds, then one can easily assign each item to 4 destination groups (i.e. m objects to each destination group), and this solves the Chromatic Load-Balancing problem.

On average, there will be $n/8m$ items, and with very high probability, there will be at most $n/4m$ items. If there are at most $n/4m$ items, then A will succeed. Thus the Chromatic Load-Balancing problem can be solved with probability at least $\frac{1}{4}$ in the same asymptotic time as A. By Lemma 6.1, $t = \mu(\frac{(1/8)\log\log n - \log 8m\gamma}{2\log\mu} - O(m))$.

Padded Sort: (We actually prove that this lower bound holds for sorting into any array of size linear in n, not just $n + o(n)$.) We can reduce the Chromatic Load-Balancing problem (with $m = \log\log\log\log n$) to the Padded-Sort problem with no asymptotic increase in running time as follows. Assign the colors individual integers from 0 to $8m - 1$. For each group with color i, uniformly choose a random real number from the range $(i/8m, (i+1)/8m]$. Thus each group will be assigned a number from $(0, 1]$ and these will be uniformly distributed. Now assume we have a Padded-Sort algorithm which will place these numbers in sorted order into an array A of size kn for some constant k. Run this Padded-Sort algorithm. If a group was placed at location i in array A, place each of the $4m$ objects in the group into an array B of size $4mkn$ at locations $4mi$ to $4m(i+1) - 1$. Then each processor j is assigned the tasks at locations $\ell n + j$, for $0 \leq \ell < 4mk$. If the Padded Sort is successful, there is a color whose objects are mapped to at most $3kn/8$ consecutive positions in B. For this color, each processor is assigned at most $3k/8 < m$ objects, and this solves the Chromatic Load-Balancing problem. By Lemma 6.1, Padded Sort requires $\mu(\frac{(1/8)\log\log n - \log 8m\gamma}{2\log\mu} - O(m))$ time. □

Corollary 6.1 *Solving the Load Balancing problem, the LAC problem, or the Padded Sort problem with probability at least $\frac{1}{2}$ requires $\Omega((g\log\log n)/\log g)$ time on a Randomized QSM, $\Omega(g\log\log n)$ time on a Randomized s-QSM, and $\Omega((L\log\log n)/\log(L/g))$ time on a Randomized BSP if $p = \Omega(n/(\log n)^{1/8-\epsilon})$ for some constant $\epsilon > 0$.*

Corollary 6.2 *Let $n/p \geq \lambda$. Solving the Load Balancing problem, the LAC problem, or the Padded Sort problem with probability at least $\frac{1}{2}$ requires $\frac{(1/8)\log\log n - \log\gamma}{2\log(\mu n/\lambda p)} - O(m)$ rounds on a Randomized GSM.*

Theorem 6.2 *Let $n > p$. Solving the Load Balancing problem, the LAC problem, or the Padded-Sort problem with probability at least $\frac{1}{2}$ requires $\Omega((\log^* n - \log^*(n/p)) + \frac{\log\log n}{\log(gn/p)})$ rounds on a Randomized* QSM*, $\Omega(\frac{\log\log n}{\log(n/p)})$ rounds on a Randomized s-*QSM*, and and $\Omega((\log\log n)/\log(\max\{L/g, n/p\}))$ rounds on a Randomized* BSP *if $p = \Omega(n/(\log n)^{1/8-\epsilon})$ for some constant $\epsilon > 0$.*

Proof Sketch: The first part of the lower bound for the QSM follows from the lower bound of [15] by assuming that a processor can read or write to n/p cells in any given state in a single step. (Note that the contention does not affect that lower bound, since it is for the CRCW PRAM, which allows any amount of contention.) The other parts of the lower bounds follow from Corollary 6.2. □

6.3 Another Lower Bound for Compaction on the GSM

Here we relax the definition of a round on the GSM. Given a time bound h we shall denote by GSM(h) a GSM in which a round will be a phase that takes $O(\mu h/\lambda)$ time, instead of a round being a phase which takes $O(\mu n/\lambda p)$ time. This definition of round will hold for GSM(h) regardless of the number of processors p being used. Note that in a round, a single processor can perform at most $O(\alpha h/\lambda)$ reads and writes, and a cell can be read from or written to by at most $O(\beta h/\lambda)$ processors. (The results below implicitly assume the constant in the $O()$ notation is 1, but modifying them for any constant is straightforward.)

Theorem 6.3 *Solving $((\mu h/\lambda)+1)$-LAC with a destination array of size d on a* GSM(h) *requires $\Omega(\sqrt{\log(n/(d\gamma))/\log(\mu h/\lambda)})$ rounds.*

Proof: This proof is similar to the proof of Theorem 3.2. Consider a GSM(h) that solves the $((\mu h/\lambda)+1)$-LAC problem over the input set $I = \{x_0, \ldots, x_{n-1}\}$. We prove by induction on t, that at the end of round t, there is a set of inputs $V_t \subseteq I$ such that **(1)** $|V_t| \geq \frac{n/\gamma}{(4\mu h/\lambda)^{2t(t+1)}}$, **(2)** each processor and cell depends on at most one variable from V_t, **(3)** each variable in V_t affects at most $(4\mu h/\lambda)^{2t}$ processors and at most $(4\mu h/\lambda)^{2t}$ cells. These conditions hold for $V_0 = \{x_0, x_\gamma, x_{2\gamma}, \ldots, x_{n-\gamma}\}$.

Assume that these properties hold for step t. Let $G_t = (V_t, E)$ be the directed graph with $(x_i, x_j) \in E$ if, at the beginning of step $t+1$, there exists a processor P that depends on x_i and a cell C that depends on x_j such that P reads from C during step $t+1$ for some value of x_i. Note that $2(\beta h/\lambda)+1$ processors cannot all read from the same cell during step $t+1$. Otherwise we could set the values of $(\beta h/\lambda)+1$ variables so that more than $\beta h/\lambda$ read from a particular cell. The indegree of any node $x_j \in V_t$ is at most $2(\beta h/\lambda)(4\mu h/\lambda)^{2t}$. The outdegree of any node $x_i \in V_t$ is at most $2(\alpha h/\lambda)(4\mu h/\lambda)^{4t}$.

A graph $G = (V, E)$ of degree at most k has an independent set of cardinality at least $|V|/(k+1)$. Since G_t has degree at most $[2(\beta h/\lambda) + 2(\alpha h/\lambda)](4\mu h/\lambda)^{2t} < (4\mu h/\lambda)^{2t+1}$, it has an independent set $V'_{t+1} \subseteq V_t$ such that $|V'_{t+1}| \geq |V_t|/(4\mu h/\lambda)^{2t+1}$.

After the read phase, each processor depends on at most one variable in V'_{t+1}. Also each variable in V'_{t+1} affects at most $(4\mu h/\lambda)^{2t} + 2(\alpha h/\lambda)(4\mu h/\lambda)^{2t} \leq (4\mu h/\lambda)^{2t+1}$ processors.

Now consider the write phase of step $t+1$. Let $G'_{t+1} = (V'_{t+1}, E')$ be the directed graph with $(x_i, x_j) \in E$ if, before the write phase of step $t+1$, there exists a processor P that depends on x_i and a cell C that depends on x_j such that P writes to C during step $t+1$ for some value of x_i. Note that $2(\beta h/\lambda)+1$ processors cannot all write to the same cell during step $t+1$. Otherwise we could set the values of $(\beta h/\lambda)+1$ variables so that more than $\beta h/\lambda$ write to a particular cell. The

indegree of any node $x_j \in V_t$ is at most $2(\beta h/\lambda)(4\mu h/\lambda)^{2t+1}$. The outdegree of any node $x_i \in V_t$ is at most $2(\alpha h/\lambda)(4\mu h/\lambda)^{2t+1}$.

Since G'_{t+1} has degree at most $[2(\beta h/\lambda) + 2(\alpha h/\lambda)](4\mu h/\lambda)^{2t+1} < (4\mu h/\lambda)^{2t+2}$, it has an independent set $V_{t+1} \subseteq V'_{t+1}$ such that $|V_{t+1}| \geq |V'_{t+1}|/(4\mu h/\lambda)^{2(t+1)}$.

Obviously, each processor depends on at most one variable in V_{t+1}. Also each variable in V_{t+1} affects at most $(4\mu h/\lambda)^{2t+1} + 2(\alpha h/\lambda)(4\mu h/\lambda)^{2t+1} \leq (4\mu h/\lambda)^{2(t+1)}$ cells.

$$|V_{t+1}| \geq \frac{|V_t|}{(4\mu h/\lambda)^{2(t+1)}} \geq \frac{(n/\gamma)/(4\mu h/\lambda)^{t(t+1)}}{(4\mu h/\lambda)^{2(t+1)}} = \frac{n/\gamma}{(4\mu h/\lambda)^{(t+1)(t+2)}}.$$

Since $|V_t| \geq (n/\gamma)/(4\mu h/\lambda)^{t(t+1)}$, it follows that for any $x > 0$, $V_t \geq x$, as long as $t \leq \sqrt{\log(n/x\gamma)/(\log(4\mu h/\lambda))} - 1$. When $x = d + 1$, for any output that the algorithm produces in the output cells (at most d of them), we can set the marked elements (bits) in V_t such that the algorithm errs. Thus, any algorithm solving $((\mu h/\lambda) + 1)$-LAC requires more than $\sqrt{\log(n/((d+1)\gamma))/\log(4\mu h/\lambda)} - 1 = \Omega(\sqrt{\log(n/(d\gamma))/\log(\mu h/\lambda)})$ rounds. □

Corollary 6.3 *Solving $((gn/p) + 1)$-LAC on a* QSM *requires $\Omega(\sqrt{\log n/\log(gn/p)})$ rounds. Solving $((n/p) + 1)$-LAC on an s-*QSM *requires $\Omega(\sqrt{\log n/\log(n/p)})$ rounds, and on a* BSP *requires $\Omega(\sqrt{\log p/\log(n/p)})$ rounds.*

We now prove a lower bound on the time needed by deterministic algorithms to solve LAC.

Lemma 6.3 *Solving LAC on a* GSM *requires $\Omega(\mu\sqrt{\log(n/\gamma)/(\log\log(n/\gamma) + \log\mu)})\}$ time.*

Proof: Let $r = n/\gamma$ and let $\tau = \sqrt{\log r/(\log\log r + \log\mu)}$. For $h = \lambda\tau$ we show that solving $((\mu h/\lambda) + 1)$-LAC requires $\Omega(\mu\tau)$ time.

Consider any algorithm that solves $((\mu h/\lambda) + 1)$-compaction. If the number of read-writes performed by any processor in a phase is more than $\alpha\tau$ or the maximum contention at any memory location in a phase is more than $\beta\tau$, then the time bound will exceed $\mu\tau$, i.e., exceed $\mu h/\lambda$. Otherwise, each phase takes no more than $\mu h/\lambda$ time, hence by Theorem 6.3 the algorithm must execute $\Omega(\sqrt{\log r/\log(\mu h/\lambda)})$ rounds, i.e., $\Omega(\sqrt{\log r/(\log\log r + \log\mu)})$ rounds. Since each round takes at least μ time, the desired lower bound follows. □

Corollary 6.4 *Solving LAC with a deterministic algorithm requires $\Omega(g\sqrt{\frac{\log n}{\log\log n + \log g}})$ time on a* QSM*, $\Omega(g\sqrt{\frac{\log n}{\log\log n}})$ time on an s-*QSM*, and $\Omega(L\sqrt{\frac{\log p}{\log\log p + \log(L/g)}})$ time on a p-processor* BSP*.*

We now prove lower bounds on the number of rounds needed by randomized algorithms.

Corollary 6.5 *If $(\mu h/\lambda) = 2^{o(\log^{1/3} n/d\gamma)}$, solving $((\mu h/\lambda) + 1)$-LAC with a destination array of size d with probability greater than $\frac{1}{2}(1 + \epsilon)$ on a Randomized* GSM(h) *requires $\Omega(\sqrt{\log(n/d\gamma)/\log(\mu h/\lambda)})$ rounds, for any constant $\epsilon > 0$.*

Proof: We note that Theorem 10 from [14] also applies to rounds on the GSM. Assume there is a randomized algorithm for $((\mu h/\lambda) + 1)$-LAC that runs in time T. Since there are $n^{(\mu h/\lambda)+1}$ possible input configurations, there is a deterministic algorithm that runs in time $T + \log\log n^{(\mu h/\lambda)+1} = T + \log((\mu h/\lambda) + 1) + \log\log n$. Then as long as $\log(\mu h/\lambda) = o(\sqrt{\log(n/d\gamma)/\log h})$, or $(\mu h/\lambda) = 2^{o(\log^{1/3} n/d\gamma)}$, the asymptotic lower bound for deterministic algorithms holds for randomized algorithms as well. □

Corollary 6.6 *If $(n/p) = 2^{o(\log^{1/3} n/d\gamma)}$, and ϵ is any positive constant, solving $((gn/p)+1)$-LAC with probability greater than $\frac{1}{2}(1+\epsilon)$ on a Randomized* QSM *requires $\Omega(\sqrt{\log n/\log(gn/p)})$ rounds; solving $((n/p)+1)$-LAC with probability greater than $\frac{1}{2}(1+\epsilon)$ on a Randomized s-*QSM *requires $\Omega(\sqrt{\log n/\log(n/p)})$ rounds, and on a Randomized* BSP *requires $\Omega(\sqrt{\log p/\log(n/p)})$ rounds.*

7 The OR Lower Bound

In Section 4 we give a description of the standard Random Adversary Technique. Here we show how to modify the technique, which will allow us to prove a lower bound for OR.

7.1 Modified Random Adversary

First we need some 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 '$*$'.

Instead of having the Random Adversary proceed through the given deterministic algorithm phase by phase, fixing some of the inputs, the Random Adversary will be proceeding phase by phase, restricting the set of possible input maps (but not fixing any inputs). Only at the last phase will the Random Adversary be randomly fixing inputs. Formally, we let $\mathcal{F}$ be a set of input maps and we say $\mathcal{F}'$ refines $\mathcal{F}$ if $\mathcal{F}' \subseteq \mathcal{F}$.

We will use a function RANDOMFIX which will be called with a set of input maps and which returns an input map according to the distribution $\mathcal{D}$ restricted to those input maps.

We will also have a RANDOMRESTRICT procedure which is called with a set of input maps $\mathcal{F}$ and a subset $\mathcal{F}' \subseteq \mathcal{F}$. According to the distribution $\mathcal{D}$, it returns either $\mathcal{F} \setminus \mathcal{F}'$ or $\mathcal{F}'$

Given an algorithm A, we construct a REFINE procedure that will tell the Random Adversary how to restrict the input maps at each step. Formally, REFINE$(t, \mathcal{F})$ takes a number of big-steps t and a set of input maps $\mathcal{F}$ and returns a triple $(\mathcal{F}', x, d)$ consisting of a new set of input maps $\mathcal{F}'$ that refines $\mathcal{F}$, a lower bound x on the number of big-steps required for the next phase, and a boolean variable d indicating TRUE if the input map is fully defined. The definition of t-goodness will refer to a set of input maps, rather than partial input maps. Then for some target probability Z, we need to prove the following.

Lemma 7.1 *With probability Z, for every step ($0 \leq t \leq T$), $\mathcal{F}_t$ (the restricted set of input maps at step t) is t-good.*

7.2 GSM Definitions

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

7.3 The Lower Bound

We will prove a general lower bound on the amount of information which can be transferred between processors given a random input of a special form. 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 $Q = \{0, 1\}$. Let $d_0 = \log_{\mu+1}^{(\frac{3}{4}\log^*_{\mu+1}(n/\gamma))}(n/\gamma)$, and let $d_{i+1} = (\mu+1)^{(\mu+1)^{d_i}}$ for $i \geq 1$. Let $\mathcal{H}_i$ be the distribution of input maps in which each set of γ inputs associated with a cell is set to 1 with probability $1/d_i$. The probability distribution $\mathcal{D}$ used in our lower bound is as follows. The input map consisting of all zeros is chosen with probability $\frac{1}{2}$. For $i \in \{0, \frac{1}{4}\log^*_{\mu+1}(n/\gamma)\}$, with probability $2/\log^*_{\mu+1}(n/\gamma)$, the distribution $\mathcal{H}_i$ is used.

If $T \leq \frac{1}{4}\log^*_{\mu+1}(n/\gamma)$ then the following is easily proved.

Fact 7.1 $d_T \leq \log\log(n/\gamma)$.

A set of input maps $\mathcal{F}$ is called *t-good* if the following conditions are satisfied.
(1) For each processor or cell v, $|\text{Know}(v, t, \mathcal{F})| \leq d_t$,
(2) For each input i, $|\text{AffProc}(i, t, \mathcal{F})| \leq d_t$ and $|\text{AffCell}(i, t, \mathcal{F})| \leq d_t$.

For each processor or cell v, $|\text{Know}(v, 0, \mathcal{F}_*)| \leq 1$, and for each input i, $|\text{AffProc}(i, 0, \mathcal{F}_*)| \leq 1$ and $|\text{AffCell}(i, 0, \mathcal{F}_*)| \leq 1$, so the set of input maps $\mathcal{F}_*$ is 0-good.

We now describe algorithm REFINE which is called with a time t and a set of input maps $\mathcal{F}$, and which returns a triple $(\mathcal{F}', x, d)$ consisting of a partial input map $\mathcal{F}'$ which is a random refinement of $\mathcal{F}$, a lower bound x on the time of the step taken, and a boolean variable d indicating TRUE if the input map is fully defined. 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 (13), we test to see if the expected time of this step will be large because of a processor accessing many cells, or a cell being accessed by many processors. If this is the case, then when we fix all inputs, the expected time of the algorithm is high. Next, in lines (15) through (19), we test whether the input has many ones. If so, we "give up". If not, then we continue. In summary, we force the algorithm to restrict the information exchange (size of reads and writes) at early steps because of the possibility of large numbers of ones. Gradually however, the algorithm can ascertain that the number of ones is smaller, and thus increase the size of its reads and writes (while maintaining small probabilities of congestion).

We define $\text{MaxCell}(t, \mathcal{G})$ as the cell with the maximum possible contention at big-step t for any input map in $\mathcal{G}$. We define $\text{MaxRWC}(c, t, \mathcal{G})$ as the maximum possible contention at c at big-step t for any input map in $\mathcal{G}$. We define $\text{MaxProc}(t, \mathcal{G})$ as the processor with the maximum possible number of reads or writes at big-step t for any input map in $\mathcal{G}$. We define $\text{MaxRWP}(p, t, \mathcal{G})$ as the maximum possible number of reads or writes p makes at big-step t for any input map in $\mathcal{G}$. Let $\text{ACCESS}(c, t, \mathcal{G})$ be the set of processors that read from or write to cell c at big-step t for any input map in $\mathcal{G}$.

Function REFINE$(t, \mathcal{F})$
(1) Let Done = FALSE
(2) Let $p = \text{MaxProc}(t, \mathcal{F})$
(3) If $\text{MaxRWP}(p, t, \mathcal{F}) \geq \alpha d_t^{d_t+2} \log^*_{\mu+1}(n/\gamma)$
(4) Let $\mathcal{F}' = \{\text{RANDOMFIX}(\mathcal{F}, I)\}$
(5) Let $p = \text{MaxProc}(t, \mathcal{F}')$
(6) Let MaxContention $= \lceil \text{MaxRWP}(p, t, \mathcal{F}')/\alpha \rceil$

(7) Let Done = TRUE
(8) Else
(9) If $\max_c \text{ACCESS}(c, t, \mathcal{F}) \geq \beta d_t^{d_t+2} \log^*_{\mu+1}(n/\gamma)$
(10) Let $\mathcal{F}' = \{\text{RANDOMFIX}(\mathcal{F}, I)\}$
(11) Let $c = \text{MaxCell}(t, \mathcal{F}')$
(12) Let MaxContention = $\lceil \text{MaxRWC}(c, t, \mathcal{F}')/\beta \rceil$
(13) Let Done = TRUE
(14) Else
(15) Let $\mathcal{F}' = \text{RANDOMRESTRICT}(\mathcal{F}, \mathcal{H}_t)$
(16) If $\mathcal{F}' = \mathcal{H}_t$
(17) Let $\mathcal{F}' = \{\text{RANDOMFIX}(\mathcal{F}', I)\}$
(18) Let Done = TRUE
(19) Let MaxContention = 1
(20) Return $(\mathcal{F}', \text{MaxContention}, \text{Done})$

Lemma 7.2 *If $\mathcal{F}$ is t-good and* REFINE$(t, \mathcal{F})$ *returns $(\mathcal{F}', 1, FALSE)$, then (1) For each processor or cell v, $|Know(v, t+1, \mathcal{F}')| \leq d_{t+1}$, and (2) For each input i, $|AffProc(i, t+1, \mathcal{F}')| \leq d_{t+1}$ and $|AffCell(i, t+1, \mathcal{F}')| \leq d_{t+1}$.*

Proof: The inputs that affect processor v are the d_t that originally affect it plus the d_t that affect each of the $2^{d_t} \alpha d_t^{d_t+2} \log^*_{\mu+1}(n/\gamma)$ possible cells it reads. Thus $|\text{Know}(v, t+1, \mathcal{F}')| \leq d_t + 2^{d_t} \alpha d_t^{d_t+3} \log^*_{\mu+1}(n/\gamma) \leq d_{t+1}$.

An input affects all the processors it originally affects, plus all the processors that possibly read from the cells that it affects. Thus $|\text{AffProc}(v, t+1, \mathcal{F}')| \leq d_t + \beta d_t^{d_t+2} (\log^*_{\mu+1}(n/\gamma)) d_t \leq d_{t+1}$.

The inputs that affect a cell v are the at most d_t that originally affect it plus the d_t that affect each of the at most $\beta d_t^{d_t+2} \log^*_{\mu+1}(n/\gamma)$ processors that possibly write to it. Thus $|\text{Know}(v, t+1, \mathcal{F}')| \leq d_t + \beta d_t^{d_t+2} (\log^*_{\mu+1}(n/\gamma)) d_t \leq d_{t+1}$.

An input affects all the cells it originally affects, plus the $2^{d_t} \beta d_t^{d_t+2} \log^*_{\mu+1}(n/\gamma)$ cells possibly written to by each processor that it affects. Thus $|\text{AffCell}(v, t+1, \mathcal{F}')| \leq d_t + 2^{d_t} \beta d_t^{d_t+2} \log^*_{\mu+1}(n/\gamma) d_t \leq d_{t+1}$. □

Lemma 7.3 *If f is t-good and REFINE$(t, \mathcal{F})$ returns $(\mathcal{F}', 1, FALSE)$, then $\mathcal{F}'$ is $(t+1)$-good.*

Proof: Follows from Lemma 7.2, and the definition of t-good. □

Lemma 7.4 *Say REFINE$(t, \mathcal{F})$ is called t' times with input parameters $(0, \mathcal{F}_*)$ through $(t'-1, \mathcal{F}_{t'})$. Then the probability that line (17) executes is at most $2t' / \log^*_{\mu+1}(n/\gamma)$.*

Proof: The probability of an input map from $\mathcal{H}_t$ being chosen is $2/\log^*_{\mu+1}(n/\gamma)$. □

Lemma 7.5 *If $\mathcal{F}$ is t-good and line (4) or (10) of* REFINE$(t, \mathcal{F})$ *is executed, then $(\mathcal{F}', x, TRUE)$ is returned with the expected value of $x \geq \log^*_{\mu+1}(n/\gamma)$.*

Proof: Assuming $\mathcal{H}_t$ is chosen, for line (4) the probability of setting the processor's inputs such that it writes to the $\beta d_t^{d_t+2} \log^*_{\mu+1}(n/\gamma)$ cells is at least $d_t^{-d_t}$. Overall, for line (4), the expected number of cells written to is expected to be at least $\log^*_{\mu+1}(n/\gamma)$.

Note: for any processor P, at most d_t^2 processors are affected by inputs which affect P. Say a set S of processors are independent if the sets of inputs affecting each processor in S are disjoint.

Now for line (10) consider the $\beta d_t^{d_t+2}(\log^*_{\mu+1}(n/\gamma))/d_t^2$ independent processors that write to the cell. Assuming $\mathcal{H}_t$ is chosen, on average at least $\beta d_t^{d_t+2}(\log^*_{\mu+1}(n/\gamma))/d_t^2 d_t^{d_t} = \beta \log^*_{\mu+1}(n/\gamma)$ write to the cell. Overall, for line (10), the expected number of processors that write to the cell is expected to be at least $\beta \log^*_{\mu+1}(n/\gamma)$. □

Theorem 7.1 *For any constant $\epsilon > 0$ solving OR with probability greater than $\frac{1}{2}(1+\epsilon)$ requires $\Omega(\mu(\log^*(n/\gamma) - \log^* \mu))$ expected time on a Randomized* GSM.

Proof: By Theorem 2.1, we simply need to show that any deterministic algorithm solving OR with the desired probability over the given distribution requires $\Omega(\mu(\log^*(n/\gamma) - \log^* \mu))$ expected time. This follows easily once we show that solving OR with the desired probability requires $\Omega(\log^*_{\mu+1}(n/\gamma))$ big-steps. (We use the fact that $\log^* n \leq \log^*_{z+1} n + \log^* z + 2$, for $z \geq 1$ [15].)

Let $T = \frac{\epsilon}{20} \log^*_{\mu+1}(n/\gamma)$. From Lemma 7.5, If line (4) or (10) is ever executed, then the expected time is $\Omega(\mu \log^*_{\mu+1}(n/\gamma))$. Now assume neither line (4) nor (10) are executed. WLOG we can assume that if line (17) was executed, or if any input affecting the output cell is 1, then the output cell contains 1. Otherwise, all inputs affecting the output cell are 0, and since the algorithm is deterministic, the output cell is fixed. If it is fixed to 1, then the output cell is always 1, and the algorithm succeeds with probability at most $\frac{1}{2}$. Otherwise, we bound the probability of success as follows. Let A be the event that the algorithm is successful in computing OR at time T. Let B_1 be the event that the input is all zeros. Let B_2 be the event that the input is chosen from one of the distributions $\{\mathcal{H}_0, \ldots, \mathcal{H}_T\}$. Let B_3 be the event that the input is chosen from one of the distributions $\{\mathcal{H}_{T+1}, \ldots, \mathcal{H}_{\log^*_{\mu+1}(n/\gamma)/4-1}\}$. Then

$$\Pr(A) = \Pr(A|B_1)\Pr(B_1) + \Pr(A|B_2)\Pr(B_2) + \Pr(A|B_3)\Pr(B_3).$$

Note $\Pr(B_1) \leq \frac{1}{2}$ and $\Pr(B_2) \leq \frac{\epsilon}{10} + \frac{1}{\log^*_{\mu+1}(n/\gamma)}$. Also, by Lemma 7.3, the number of inputs that affect the output cell is d_T, and conditioned on B_3, the probability of any of those inputs being one is at most ed_T/d_{T+1}. Thus $\Pr(A|B_3) \leq \frac{ed_T}{d_{T+1}} \leq \frac{1}{\log^*_{\mu+1}(n/\gamma)}$. Then

$$\Pr(A) \leq \frac{1}{2} + \frac{\epsilon}{10} + \frac{2}{\log^*_{\mu+1}(n/\gamma)} \leq \frac{1}{2}(1+\epsilon).$$

□

Corollary 7.1 *For any constant $\epsilon > 0$, solving OR with probability greater than $\frac{1}{2}(1+\epsilon)$ requires $\Omega(g(\log^* n - \log^* g))$ expected time on a Randomized* QSM, *$\Omega(g \log^* n)$ expected time on a Randomized s-*QSM, *and $\Omega(L(\log^*(\min\{n,p\}) - \log^*(L/g)))$ expected time on a Randomized* BSP.

For deterministic algorithms we can obtain a stronger lower bound.

Theorem 7.2 *Computing the OR of n bits with a deterministic algorithm requires $\Omega(\mu \cdot (\log(n/\gamma)/(\log\log(n/\gamma) + \log\mu)))$ time on a* GSM.

Proof: Since the input is spread over at least $r = n/\gamma$ cells, the computation takes at least as much time as that needed to compute the OR of r inputs. The proof bounds the degrees of functions describing the states of processors and contents of memory cells at each step. Since the degree of the function representing the OR of r bits is r, the computation cannot terminate with the correct value as long as the degrees of all functions describing the contents of memory cells are less than r.

Let $\tau = \log r / \log\log r$. We assume that in each step, $m_{rw} \leq \alpha\tau$ and $\kappa \leq \beta\tau$, since otherwise $T > \mu \log r / \log\log r$ and we are done. Thus, using an analysis as in [6] for the 'Few Write' PRAM, after l steps the degree of the processor/memory functions is at most $(\mu\tau)^{cl}$, for a suitable constant c. At termination we need $(\mu\tau)^{cl} \geq n$, i.e., $l = \Omega(\log r/(\log\log r + \log\mu))$. □

Corollary 7.2 *Computing the OR of n bits with a deterministic algorithm requires $\Omega(g \log n/(\log\log n + \log g))$ time on a* QSM*, $\Omega(g \log n/\log\log n)$ time on an s-*QSM*, and $\Omega(L \log q/(\log\log q + \log(L/g))$ time on a p-processor* BSP*, where $q = \min\{n, p\}$.*

Theorem 7.3 *Assuming $n/p \geq \lambda$, for any constant $\epsilon > 0$ solving OR with probability greater than $\frac{1}{2}(1+\epsilon)$ requires $\Omega(\frac{\log(n/\gamma)}{\log(\mu n/\lambda p)})$ rounds on a Randomized* GSM *(with $n/p \geq \lambda$).*

Proof: The proof for the lower bound for randomized algorithms follows as in the case of the proof for Theorem 7.2 by noting that each phase must take time $\mu n/\lambda p$, hence, as in the analysis in the proof of Theorem 7.2, after l phases the degrees are bounded by $((\alpha\beta)(n/(p\lambda))^2)^{cl} = (\mu n/\lambda p)^{cl}$, for a suitable constant c. Let $r = n/\gamma$. At termination, $r \leq (\mu n/\lambda p)^{cl}$, hence the number of rounds $l = \Omega(\log r/\log(\mu n/\lambda p))$.

We now apply the Random-Adversary Technique, using the input distribution given for the lower bound on OR in [16] to obtain the same lower bound for randomized algorithms. □

Corollary 7.3 *Assuming $n \geq p$, for any constant $\epsilon > 0$ solving OR with probability greater than $\frac{1}{2}(1+\epsilon)$ requires $\Omega(\frac{\log n}{\log(gn/p)})$ rounds on a Randomized* QSM*, $\Omega(\frac{\log n}{\log(n/p)})$ rounds on a Randomized s-*QSM*, and $\Omega(\frac{\log p}{\log(n/p)})$ rounds on a Randomized* BSP*.*

8 Upper Bounds (sketches)

Parity. On a QSM Parity can be computed in $O(g \log n/\log\log g)$, by emulating the depth 2 unbounded fan-in circuit for parity. This is close to our deterministic lower bound (which has a $\log g$ term in the denominator instead of the $\log\log g$ term in our upper bound). If unit-time concurrent reads are allowed, the resulting algorithm runs in time $O(g \log n/\log g)$, which matches the lower bound (derived for QSM with unit-time concurrent reads) in Theorem 3.1. On the s-QSM the straightforward algorithm gives the tight upper bound of $O(g \log n)$. On a BSP with p processors, $p \leq n$, we can compute parity in $O(L \log n/(\log L/g))$.

On the s-QSM and the BSP we can match the lower bounds on number of rounds needed for randomized algorithms by simple deterministic algorithms. The algorithm has the same upper bound for rounds on the QSM, and this matches the lower bound if $g = O((n/p)^{1-\epsilon})$ or $p = O(n/\log^{\epsilon}(n))$, for some $\epsilon > 0$.

Linear approximate compaction. On the QSM this can be computed in time $O(\sqrt{g \log n} + g \log\log n)$ w.h.p. It is interesting to note that the second term in this expression comes close to matching our time lower bound for QSM. On the s-QSM the same algorithm runs in time $O(g\sqrt{\log n})$. On the BSP this algorithm runs in time $O(\sqrt{Lg \log n/\log(L/g)} + L \log\log n/\log(L/g))$ w.h.p. (provided the number of elements being compacted is $O(n/(\log n 2^{\log n/(n/p)}))$. All of these results are obtained by an adaptation of the QRQW algorithm in [9].

The best algorithm that we know (deterministic or randomized) that computes in rounds is the simple algorithm based on computing prefix sums. This algorithm has the same performance as the algorithms that compute parity in rounds. Note that if we relax the definition of a round in a randomized algorithm to be a computation that terminates in $O(gn/p)$ time *w.h.p.*, then we can design algorithms (by adapting the algorithm in [10]) that beat our lower bounds (which were derived under the restriction that a round must terminate in $O(gn/p)$ time).

OR. On the QSM and the s-QSM the OR can be computed deterministically in $O((g/\log g)\log n)$ time and $O(g \log n)$ time, respectively, with simple algorithms. Both results are a factor of $\log\log n$ away from the lower bound. No better randomized algorithm is known in either model, unless unit-time concurrent reads are allowed, in which case, the OR can be computed with high probability on both models in $O(g \log n/\log\log n)$ time (by an adaptation of a QRQW algorithm given in [9]). This is still much larger than our corresponding lower bounds. On the BSP one can compute the OR in $O(L \log n/\log(L/g))$ [12].

On all three models we can match the lower bound on number of rounds for randomized algorithms by simple deterministic algorithms.

References

[1] M. Adler, P. Gibbons, Y. Matias, V. Ramachandran. Modeling parallel bandwidth: Local vs. global restrictions. *Proc. ACM Symp. on Parallel Algorithms and Architectures*, pp. 94–105, 1997; *Algorithmica*, to appear.

[2] M. Ajtai and M. Ben-Or. A Theorem on Probabilistic Constant Depth Computations. *Proc. 16th ACM Symp. on Theory of Computing*, pp. 471–474, 1984.

[3] P. Beame, J. Hastad. Optimal bounds for decision problems on the CRCW PRAM. *JACM*, vol. 36, pages 643–670, 1989.

[4] C. Berge. *Graphs and Hypergraphs.* North-Holland, Amsterdam, 1976.

[5] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. *4th ACM SIGPLAN Symp. on Princ. and Prac. of Para. Prog.*, 1993.

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

[7] F. Fich, M. Kowaluk, M. Kutyłowski, K. Loryś, and P. Ragde. Retrieval of scattered information by EREW, CREW, and CRCW PRAMs. In *Proc. 3rd Scand. Workshop on Alg. Theory*, pages 30–41. Lec. Notes in Comp. Sci., Vol. 621, 1992.

[8] P. B. Gibbons, Y. Matias, and V. Ramachandran. Efficient low-contention parallel algorithms. In *JCSS*, vol. 53, pp. 395–416, 1996 (Special Issue for *SPAA'94*.

[9] 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, *SICOMP*, to appear.

[10] P. B. Gibbons, Y. Matias, and V. Ramachandran. Can a shared-memory model serve as a bridging model for parallel computation? In *ACM Symp. on Parallel Algorithms and Architectures*, pages 72–83, 1997.

[11] M. Goodrich. Communication-efficient parallel sorting. *Proc. STOC*, pages 247–256, 1996.

[12] B. H. H. Juurlink and H. A. G. Wijshoff. Communication primitives for BSP computers. *IPL*, vol. 58, pages 303–310, 1996.

[13] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, chapter 17, pages 869–941. MIT Press/Elsevier, 1990.

[14] M. Kutylowski and K. Lorys. Limitations of the QRQW and EREW PRAM models. In *Proc. Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*, 1996.

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

[16] P. D. MacKenzie. Lower Bounds for Randomized Exclusive-Write PRAMs. In *Proc. 7th ACM Symp. on Para. Alg. and Arch.*, 254–263, 1995,

[17] P. D. MacKenzie. A lower bound for the QRQW PRAM. In *Proc. 7th IEEE Symp. on Para. and Distr. Proc.*, pages 231–237, 1995.

[18] P. D. MacKenzie. An improved lower bound for the QRQW PRAM. In *Workshop on Randomized Parallel Computing, IPPS*, Hawaii, April 1996.

[19] P. D. MacKenzie and V. Ramachandran. Computational Bounds for Fundamental Problems on General-Purpose Parallel Models. *Proc. 1998 ACM Symp. on Parallel Algs. and Architectures*, June-July 1998, to appear.

[20] N. Nisan. CREW PRAMs and decision trees. *SIAM J. Comput.*, 20:999–1007, 1991.

[21] V. Ramachandran. A general purpose shared-memory model for parallel computation. *Proc. IMA Workshop on Parallel Algorithms*, Springer Verlag, to appear.

[22] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 77–82, 1987.

[23] M. Szegedy. *Algebraic methods in lower bounds for computational models with limited communication.* PhD thesis, University of Chicago, 1989.

[24] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

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

Time Lower Bounds for QSM (unlimited no. of procs. unless specified otherwise)		
problem (n = **size of input**)	**Deterministic time l.b.**	**Randomized time l.b.**
Linear approximate compaction	$\Omega(g\sqrt{\frac{\log n}{\log\log n+\log g}})$	$\Omega(\frac{g\log\log n}{\log g})$, $\Omega(g\log^* n)$ with n proc.
OR	$\Omega((\frac{g\log n}{\log\log n+\log g}))$	$\Omega(g(\log^* n-\log^* g))$
Parity and related problems	$\Omega(\frac{g\log n}{\log g})$; $\Theta(\frac{g\log n}{\log g})$ with concur. reads	$\Omega(\frac{g\log n}{\log\log n+\min(\log\log g,\log\log p)})$ with p procs.; $\Omega(\frac{g\log n}{\log\log n})$ if p polynomial in n

Time Lower Bounds for s-QSM (unlimited no. of procs. unless specified otherwise)		
problem (n = **size of input**)	**Deterministic time l.b.**	**Randomized time l.b.**
Linear approximate compaction	$\Omega(g\sqrt{\frac{\log n}{\log\log n}})$	$\Omega(g\log\log n)$
OR	$\Omega(\frac{g\log n}{\log\log n})$	$\Omega(g\log^* n)$
Parity and related problems	$\Theta(g\log n)$	$\Omega(\frac{g\log n}{\log\log n})$

Time Lower Bounds for BSP **with** p **Processors** ($q=\min\{n,p\}$)		
problem (n = **size of input**)	**Deterministic time l.b.**	**Randomized time l.b.**
Linear approximate compaction	$\Omega(L\sqrt{\frac{\log q}{\log\log q+\log(L/g)}})$	$\Omega(L\log\log n/\log(L/g))$ for $p=\Omega(n/(\log n)^{1/8-\epsilon})$
OR	$\Omega(\frac{L\log q}{\log\log q+\log(L/g)})$	$\Omega(L(\log^* q-\log^*(L/g)))$
Parity and related problems	$\Theta(\frac{L\log q}{\log(L/g)})$	$\Omega(L\sqrt{\frac{\log q}{\log\log q+\log(L/g)}})$

Number of Rounds for p**-processor Algorithms** ($p\le n$)			
problem (n = **size of input**)	QSM **lower bound**	s-QSM **lower bound**	BSP **lower bound**
Linear approx. compaction	$\Omega((\log^* n-\log^*(n/p))+\sqrt{\frac{\log n}{\log(gn/p)}})$	$\Omega(\sqrt{\frac{\log n}{\log(n/p)}})$	$\Omega(\sqrt{\frac{\log n}{\log(n/p)}})$
OR	$\Theta(\frac{\log n}{\log(ng/p)})$	$\Theta(\frac{\log n}{\log(n/p)})$	$\Theta(\frac{\log n}{\log(n/p)})$
Parity and related problems	$\Omega(\frac{\log n}{\log(n/p)+\min\{\log g,\ \log\log p\}})$	$\Theta(\frac{\log n}{\log(n/p)})$	$\Theta(\frac{\log n}{\log(n/p)})$

Table 1: A Θ in an entry in any of the tables indicates that the bound is tight.