Limitations of the QRQW and EREW PRAM models*

Mirosław Kutyłowski,
Heinz Nixdorf Institute and Dept. of Mathematics & Computer Science
University of Paderborn, D-33095 Paderborn, Germany,
mirekk@uni-paderborn.de, fax: +49 5251 606482

Krzysztof Loryś
Institute of Computer Science, University of Wrocław,
Przesmyckiego 20, PL-51-151 Wrocław, Poland
lorys@tcs.uni.wroc.pl

Abstract. We consider parallel random access machines (PRAMs) with restricted access to the shared memory resulting from handling congestion of memory requests. We study the (SIMD) QRQW PRAM model where multiple requests are queued and serviced one at a time. We also consider exclusive read exclusive write (EREW) PRAM and its modification obtained by adding a single bus.

For the QRQW PRAMs we investigate the case when the machine can measure the duration of a single step. For such a (powerful) QRQW PRAM PARITY of n bits (PARITY$_n$) requires $\Omega(\log n)$ time while OR of n bits can be computed deterministically in a constant time. On randomized QRQW PRAM the function PARITY$_n$ is still difficult. We prove a lower time bound $\Omega(\sqrt{\log n/\log\log n})$ for algorithms that succeed with probability $\frac{1}{2}+\epsilon$ ($\epsilon > 0$). These bounds show limitations of the QRQW PRAM model when a problem requires intensive communication.

The 2-compaction problem is known to be hard for EREW PRAM. The same time bound $\Omega(\sqrt{\log n})$ for this problem has been proved for both deterministic and randomized EREW PRAM. We show that this is not a coincidence since the time complexity of this problem is the same for deterministic and randomized case. The technique which we apply is quite general and may be used to obtain similar results for any problem where the number of input configurations is small.

We also show that for a slightly modified EREW PRAM (obtained by adding a single bus used for halting) the time complexity of 2-compaction is $\Theta(\sqrt{\log n})$. This is an evidence that improving time bound $\Omega(\sqrt{\log n})$ for 2-compaction on EREW PRAM requires novel and more sophisticated techniques.

1 Introduction

Parallel Random Access Machines (PRAMs) are widely used for designing parallel algorithms (for a detailed definition of PRAMs see for instance [6]). PRAM consists of a number of processors (each possessing a local memory for local computations) and a number of shared memory cells. The execution is divided into steps, each

* partially supported by KBN grant 8 S503 002 07 and DFG-Sonderforschungsbereich 376 "Massive Parallelität"

step consists of a read phase, when each processor may read an arbitrary cell, a computation phase, when each processor makes some internal computation (often no limits are set on this computation except it uses local data known to the processor), and a write phase, when each processor may decide to try to write a value into an arbitrary shared memory cell.

There is a wide collection of fast and work-efficient parallel PRAM algorithms. However, many of them can be hardly implemented in the real world, because of communication problems. Even if the messages between processors and the shared memory are routed efficiently (say by optical channels), it remains a problem how they are serviced at the memory locations that are claimed by many processors. Processing these read or write requests can significantly decrease time efficiency of the algorithm.

In order to avoid the situation described exclusive-read exclusive-write (EREW) PRAMs have been proposed as a model for algorithm design. In this model it is prohibited to attempt to write or read from the same location of the shared memory at the same time. This solves the congestion problems, but makes designing efficient algorithms significantly more difficult or sometimes even impossible. A good example of the last phenomenon is the 2-compaction problem: *given an array of size n storing at most 2 nonempty elements, move these elements to the first two positions in the array.* It is known that 2-compaction problem requires $\Omega(\sqrt{\log n})$ steps of the EREW PRAM [7]. This bound holds even for the randomized EREW PRAM [11]. On the other hand, 2-compaction requires a constant time if 2 requests addressed to the same memory location at the same time may be serviced.

Gibbons, Matias and Ramachandran introduced the Queue Read Queue Write (QRQW) PRAM in which capabilities of servicing many requests concerning the same memory location has been taken into account ([8, 9]). The motivation is that the real machines may queue the requests and therefore the EREW model is too restrictive. The QRQW model differs from the CRCW PRAMs, because the time needed to serve the queue is taken into consideration when analyzing the runtime of an algorithm:

Definition 1. A (SIMD) QRQW PRAM is a PRAM for which concurrent reads and writes are allowed, but the time spent to execute a parallel step equals $r + w$ where r is the maximum of 1 and the maximal number of processors reading the same memory location during this step, and w is the maximal number of processors trying to write into the same memory location during this step. The execution time of the algorithm is defined as the total time used to execute all steps. For the lower time bounds, we assume that during a concurrent-write operation all messages to be written are combined with the old contents of the cell and stored there. (It might be understood that the cells of QRQW PRAM are in fact memory units.)

A wide collection of efficient and tricky algorithms has been designed for the QRQW PRAMs, including for instance k-compaction and linear approximate compaction [8, 9]. But QRQW PRAMs have also their limitations, for instance the $\Omega(\log m)$ bound for broadcasting a message to m locations holds as for the EREW model [9].

Typical Boolean functions used to test the strength of different communication modes on the PRAM are PARITY_n (sum modulo 2 of n input bits) and OR_n (the OR of n input bits). These functions require $\Omega(\log n)$ on the CREW PRAM, even in the randomized case [5, 11] (the complexity of PARITY_n being exactly the same as in the deterministic case even if the error can be arbitrarily close to $\frac{1}{2}$).

Definition 2. QRQW PRAMs with latency detection is a QRQW PRAM for which every processor can measure the execution time of each write or read phase.

The latency detection seems to be a natural mechanism that should be considered as a hardware option, and could be useful even if the clock is not very precise. For instance OR_n can be computed in $O(1)$ time on QRQW PRAMs with latency detection (Proposition 4). On the other hand we show that PARITY_n requires $\Omega(\log n)$ steps on such a machine (Theorem 5). The lower bound that has been previously known for PARITY_n was $\Omega(\log n/\log\log n)$ for the CRQW model with no latency detection. It follows from a general bound $\Omega(\frac{\log \deg f}{\log\log \deg f})$, where $\deg(f)$ is the degree of f, derived from the bound on the Few-Write CRCW (see [9, 5]).

We cannot expect that for PARITY_n the situation is much better when we add randomization; for QRQW PRAM with polynomially many resources it follows from a generalization of the theorem of Beame and Håstad on CRCW [2]. For randomized QRQW PRAMs with arbitrary resources, latency detection and with success probability at least $\frac{1}{2}+\epsilon$ ($\epsilon > 0$) we prove the time bound $\Omega(\sqrt{\log n/\log\log n})$.

In the second part of the paper we turn our attention to complexity of 2-compaction problem on EREW PRAMs. As already mentioned, there is a bound of $\Omega(\sqrt{\log n})$ for the runtime on deterministic and randomized EREW PRAMs. We show that it is no coincidence that these bounds are equal, since the runtime of the best randomized algorithm for 2-compaction is the same (up to a constant) as the deterministic runtime on the EREW PRAM (Corollary 11). This follows from a general observation that if the set of all possible input string to a problem consists of m elements, then with extra $O(\log\log m)$ steps it is possible to derandomize any PRAM algorithm solving the problem (Theorem 10). Hence very complicated analysis that was one of the main contributions of [11] is no longer necessary.

It seems that time complexity of 2-compaction on EREW is higher than $\sqrt{\log n}$. However, we found out why new type of arguments must be designed to prove this conjecture:

Definition 3. EREW^+ PRAM is an EREW PRAM for which any processor may halt the computation at any time (many of them may do it simultaneously). After halting the correct output should be stored at fixed locations.

Obviously the EREW^+ PRAM may compute OR_n in $O(1)$ time. On the other hand, the proof from [7] establishing the bound $\Omega(\sqrt{\log n})$ for 2-compaction may be applied to the EREW^+ PRAM after slight changes. However, we show how to compute 2-compaction within this time on the EREW^+ PRAM (Proposition 12).

In order to shed some light light on EREW^+ PRAM model we state that time complexity of PARITY_n is $\Theta(\log n)$ in deterministic case and $\Omega(\sqrt{\log n})$ in randomized case (Theorem 14).

2 Complexity of PARITY function on QRQW PRAMs

2.1 Deterministic QRQW PRAMs with latency detection

Proposition 4. OR_n *can be computed in a constant time on QRQW with latency detection.*

Proof. We use $2n$ processors. Processor P_i, for $i \leq n$, reads the ith input bit and if it is a one, P_i writes it to cell C_i during the write phase. At this moment, for every $i \leq n$ processor P_{n+i} writes to C_i, too. Obviously, the write phase takes 1 time unit if and only if globally no conflict occurs. No write conflict may cost more than 2 time units, since for a cell C_i only processors P_i and P_{n+i} may try to write there. By measuring duration of the write phase every processor may determine the value of OR_n. □

Proposition 4 shows that measuring latency is a powerful tool. However, its power is limited to disseminating only a small portion of information from a single source (even if the source is unknown) of the form "there was a write conflict of this size". For some functions it does not help at all, as shown by the following theorem:

Theorem 5. PARITY_n *has time complexity* $\Theta(\log n)$ *on CRQW PRAM with latency detection.*

(CRQW PRAM is a PRAM for which concurrent reads are executed in one time unit, but concurrent writes are executed as for QRQW PRAM.) Time complexity of PARITY_n on a weaker CREW PRAM is $\Theta(\log n)$ [4, 5] so admitting queues brings no runtime improvements for PARITY_n. For the proof of Theorem 5 we have to recall some techniques for describing processor and cell states of a PRAM machine.

2.1.1 Technical preliminaries

Boolean functions [5]. Each Boolean formula f has a characteristic function that can be represented by a polynomial P_f with integer coefficients. Then obviously

$$P_{\neg f} = 1 - P_f, \qquad P_{f \wedge g} = P_f \cdot P_g, \qquad P_{f \vee g} = P_f + P_g - P_f \cdot P_g.$$

Note that in particular for $P_{f_1 \vee \ldots \vee f_k}$ we get the following inclusion-exclusion formula:

$$P_{f_1} + \ldots + P_{f_k} - \sum_{\substack{i,j \leq k \\ i \neq j}} P_{f_i} \cdot P_{f_j} + \sum_{\substack{i,j,l \leq k \\ i \neq j \neq l}} P_{f_i} \cdot P_{f_j} \cdot P_{f_l} + \ldots + (-1)^{k+1} \cdot P_{f_1} \cdot \ldots \cdot P_{f_k}. \quad (1)$$

It is known that for every Boolean function f the polynomial P_f is unique provided that we reduce every x^i to x for $i \geq 1$ (simply x and x^i have the same values for arguments $0, 1$). The degree of a Boolean function f, $deg(f)$, is defined as the degree of polynomial P_f. The fundamental properties of degree are :

$$\begin{aligned}
\deg(\neg f) &= \deg(f), \\
\deg(f_1 \wedge \cdots \wedge f_k) &\leq \deg(f_1) + \cdots + \deg(f_k), && (2)\\
\deg(f_1 \vee \cdots \vee f_k) &\leq \deg(f_1) + \cdots + \deg(f_k), && (3)\\
\text{and } \deg(f_1 \vee \cdots \vee f_k) &= \max(\deg(f_1), \ldots, \deg(f_k)) \\
&\text{if} \quad f_1, \ldots, f_k \text{ are mutually exclusive.} && (4)
\end{aligned}$$

Representing processor and memory states. With each possible state of a processor (or a cell) at a given moment of the computation we associate a Boolean function on the input bits that describes for which inputs this state is reached.

It is also convenient to assume that the processors and cells forget no information: for instance if a processor P in a state described by a function f reads a cell being in a state described by function g, then the state of P after reading is described by $f \wedge g$. So, by (2), the degrees of Boolean functions describing processor states at most double at each reading phase. The growth of degrees while writing may be bounded similarly provided that latency detection brings no information. Initially the processors and cells are described by functions of degree at most 1. On the other hand, PARITY_n has degree n. Therefore the time required to compute PARITY_n may be bounded by the time required to achieve degree n by functions describing states of the cells.

The idea of the proof for the CRQW PRAM is to restrict the possible input strings in a way that every step takes a fixed amount of time but nevertheless the restriction is described by a Boolean function of a small degree. The growth of degrees of functions describing processor and cell states can be controlled as in the case of the CREW PRAM. Finally, after the last step of the computation for such restricted inputs the output cell is described by a function h of bounded degree. A contradiction is reached if the degree of h is smaller than n:

Fact 6 [5] *If h is not a constant function and $h(x) \leq \mathrm{PARITY}_n(x)$ for every x, then $\deg(h) = n$. Also $\deg(h) = n$, if $h(x) \geq \mathrm{PARITY}_n(x)$ for every x.*

2.1.2 Proof of Theorem 5.

Let PARITY_n be computed in time T by a CRQW PRAM M. We define inductively sets $A_0 = \{0,1\}^n \supseteq A_1 \supseteq \ldots \supseteq A_l$, such that

(i) Let t_i be the size of the longest queue that may be formed during write phase of step i for the inputs in A_i. Then there are a cell Z_i and processors $P_{i,1}, \ldots, P_{i,t_i}$ so that they all write to Z_i at step i for all inputs in A_{i+1}.

(ii) Degree of f_{A_i}, the characteristic function of A_i, is bounded by $b_i = \prod_{j=1}^{i}(4t_j+2)$.

(iii) For the inputs in A_i the states of memory cells and processors after step i are described by polynomials of degree at most b_i.

(iv) We stop the construction if M halts for any input in A_l.

Let us assume that we have determined the sets $A_0, \ldots, A_{i-1}$. In order to define A_i we consider step i executed for the inputs in A_{i-1}. Let t_i be the maximal duration of this write step for the inputs in A_{i-1}. So there are processors $P_{i_1}, \ldots, P_{i_{t_i}}$ that may write simultaneously into a cell Z_i during the write phase of step i. Let $f_j \wedge f_{A_{i-1}}$ be the characteristic function of the state of P_{i_j} causing P_{i_j} to write into Z_i. Let A_i be the set of inputs from A_{i-1} for which the function $f_{A_i} = f_1 \wedge \cdots \wedge f_{t_i} \wedge f_{A_{i-1}}$ has value 1. By property (iii), after read phase the state of each processor is described by a polynomial of degree at most $2 \cdot b_{i-1}$, so by (2), $\deg(f_{A_i}) \leq (2t_i + 1) \cdot b_{i-1}$. Note that for inputs from A_i the write phase takes **always** t_i time units. So latency detection provides no additional information.

Now let us describe the state of an arbitrary cell C after the write phase of step i. Consider the situation when C contains a symbol a after writing. Then a is a combination of b, the previous contents of C, and the messages sent to C by processors. Let g be the characteristic function describing inputs for which C stores b immediately before the write phase. Let $P_{j_1}, \ldots, P_{j_k}$ be the list of all processors that for some inputs from A_{i-1} may write into C at this step, and let $S = \{j_1, \ldots, j_k\}$.

For $T \subseteq S$ let $\phi_T = \left(\bigwedge_{j \in T} h_j\right) \wedge \left(\bigwedge_{j \in S \setminus T} \neg h_j\right)$, where for $l \leq k$ function h_l describes the set of inputs for which P_{j_l} writes into C at this step. Then for input $x \in A_i$ cell C stores a after the write phase of step i if and only if the following function has value 1:

$$s(x) = g(x) \wedge \Big(\bigvee_{T \subseteq S} \phi_T(x) \Big) \wedge f_{A_i},$$

where the disjunction is taken over all $T = \{j_{i_1}, \dots, j_{i_T}\}$ such that the messages sent by $P_{j_{i_1}}, \dots, P_{j_{i_T}}$ combined with b give a.

We show that $\deg(s) \leq (4t_j + 2) \cdot b_{i-1}$. Since $\phi_{T'}$ and $\phi_{T''}$ are mutually exclusive for $T' \neq T''$, by (4) it suffices to show that $\deg(g \wedge \phi_T \wedge f_{A_i}) \leq (4t_j + 2) \cdot b_{i-1}$ for every ϕ_T appearing in the formula defining s. To show that each such ϕ_T has a proper degree we apply formula (1). Long products occur in this formula, but it is impossible to satisfy more than t_i polynomials P_{h_s} simultaneously. Indeed, it follows from the assumption that t_i is the size of the biggest queue at step i for the inputs in A_i. Therefore we may remove all products of more than t_i polynomials P_{h_i}. It follows that $\deg(g \wedge \phi_T \wedge f_{A_i}) \leq \deg(g) + t_i \cdot 2b_{i-1} + \deg(f_{A_i}) \leq (4t_i + 2) \cdot b_{i-1}$.

We put $b_i = (4t_i + 2) \cdot b_{i-1}$. Then, as we have just seen, (iii) holds for memory cells. In the same way we can prove claim (iii) for the states of processors.

Since T is the runtime of the algorithm, we have $T \geq t_1 + \dots + t_l$. On the other hand, since the machine stops after step l, the output cell must store the correct values of PARITY_n. Hence by Fact 6 the degree of a Boolean function describing the output cell at this moment for inputs in A_l must be n. Hence

$$n \leq b_l = \prod_{j=1}^{l} (4t_j + 2) \leq \prod_{j=1}^{l} 6^{t_j} = 6^{t_1 + \dots + t_l} \leq 6^T.$$

It follows that $T = \Omega(\log n)$. □

A more careful examination shows that CRQW PRAMs cannot compute PARITY_n even by one step faster than CREW PRAMs.

2.2 Time complexity of PARITY on randomized QRQW PRAMs

We say that a randomized algorithm A succeeds in time T with probability p if for every input x the probability that A halts within T steps **and** gives the correct answer is at least p.

Theorem 7. *Suppose that a randomized QRQW PRAM M with latency detection computes PARITY_n in time T with success probability at least $0.5 + \epsilon$ ($\epsilon > 0$). Then $T = \Omega(\sqrt{\log n / \log\log n})$.*

Proof. The first step is to reduce the problem to a deterministic QRQW PRAM by the following (simple) version of Yao's Theorem [12]:

Lemma 8. *Let a randomized QRQW PRAM N compute a function f and succeed with probability p in time T. Then there is a deterministic QRQW PRAM N' that outputs the value of f within T steps for at least a fraction of p of all input sequences.*

By Lemma 8 there is a deterministic QRQW PRAM A that in T steps yields the value of PARITY_n for at least a fraction of $0.5 + \epsilon$ of all inputs. Our analysis of algorithm A is a kind of *independent set technique.* We first sketch the general idea of the construction and then go into details.

Outline of the independent set technique.

To adhere to the standard conventions let every input position be called an (input) variable. The actual input bits will be called values of the input variables.

At each step of the computation we fix and reveal the values of certain input variables to all processors. With this additional knowledge the machine cannot work slower or provide more false answers. The idea is that through revealing the variables we achieve that the configuration of the machine and the next step can be easily described. Which variables should be revealed and which values should they have is a major concern. If V_t denotes the set of variables that remain unrevealed immediately after step t, then we require that the following properties hold at this time:

1. each processor and cell knows at most one unrevealed variable, that is, either the state of the processor (the cell) is the same for all settings of the variables in V_t, or it depends only on the value of exactly one variable in V_t,
2. for each variable $x_i \in V_t$ there is a bounded number of processors and memory cells that know x_i (which is about $(\log n)^t$), the locations of these processors and cells are the same for all settings of unrevealed variables,
3. a minimal size of V_t is, roughly speaking, $n/(\log n)^{t^2}$,
4. for all settings of the unrevealed variables in V_t, the duration of each of the steps $1, \ldots, t$ is fixed and follows from the values of the revealed variables; thereby revealing duration of these steps brings no new information,
5. the fraction of settings of the unrevealed variables for which the algorithm yields correct answers is at least $\frac{1}{2} + \epsilon - \frac{t \cdot \epsilon}{\sqrt{\log n}}$.

Termination conditions.

The process of analyzing the steps might be interrupted at step t if we notice that for at least half of the settings of the variables from V_{t-1} the computation time exceeds $\sqrt{\log n}$ at step t. Then we have a contradiction. This happens if there is a memory cell C that may be read (written) by $\alpha \geq \log n$ processors. The queue in this case is formed by, say β, processors that always read C and $\alpha - \beta$ processors that may read C depending on the values of the unrevealed variables that they know. The process of reading by the later group can be described as a Bernoulli trial, where each independent random variable X_i corresponds to an unrevealed variable x_i and the processors that know x_i. The value of X_i is the number of processors depending on x_i that read C for a given value of x_i. It is elementary to show that with probability at least 0.5 the number of processors of the later group reading C is at least $0.5 \cdot (\alpha - \beta)$. This makes the total length of the queue at least $0.5 \cdot \alpha \geq 0.5 \cdot \log n$ with probability at least 0.5. For such inputs the algorithm fails, a contradiction.

If we have got a set V_t with $|V_t| > 1$, then the machine cannot halt immediately after step t. Indeed, no matter which variable from V_t is known to the output cell, for exactly 50% settings of the remaining unrevealed variables the correct output is 1 and for the rest it is 0. Since for majority of settings of the variables in $|V_t|$ the algorithm succeeds, the machine does not halt immediately after step t, and $t < T$.

Analysis of a read phase.

We assume, as an inductive hypothesis, that the conditions claimed hold after step t. We analyze the read phase of step $t+1$ and show how to reveal the variables.

A processor that does not know any unrevealed variable reads always the same cell or no cell at all. A processor P that knows a variable $x_l \in V_t$ may read one of at most two cells, one for each possible value. Both these cells are called to be *referred* by P at this step. If any cell is referred by at least $\log n$ processors, then we find a contradiction, as described above. So we may assume that every cell is referred by less than $\log n$ processors.

Now we choose the variables to be revealed at the read phase of step $t+1$. Let k_t denote the bound on the number of processors (cells) knowing a single unrevealed variable immediately after step t. We consider a graph $G_{t+1} = (V_t, E)$ with $(x_i, x_j) \in E$, if during the read phase of step $t+1$, there exists a processor P that knows x_i and a memory cell C that depends on x_j and P refers C at step $t+1$, or there are processors P_1, P_2 that know, respectively, x_i and x_j and refer the same memory cell. We estimate the degree of the graph G_{t+1}. Let $x_l \in V_t$ be an arbitrary node of this graph.

- Let x_l be known to a memory cell. There are less than $\log n$ processors referring this cell. Hence together there are no more than $k_t \cdot (\log n - 1)$ edges with the endpoint x_l constructed because reading a cell knowing x_l.
- If a processor P knows x_l and refers C and C', then for this reason at most $2 \cdot (\log n - 2)$ edges are constructed. Indeed, C (C') is referred by less than $\log n$ processors, one of them being P itself. Since x_l is initially known to up to k_t processors, there are together at most $2 \cdot (\log n - 2) \cdot k_t$ edges of this kind with endpoint x_l.

We conclude that the degree of graph G_{t+1} is less than than $u_t = (3\log n - 5) \cdot k_t + 1$. For such a graph we can choose an independent set of cardinality $|V_t|/(\text{degree}(G_{t+1}) + 1) \geq |V_t|/u_t$ [3, page 284]. We fix such an independent set V_t' and reveal the values of all variables that are not in V_t'. The values of the revealed variables are chosen so that the number of correct answers of the algorithm for the resulting subproblem is maximal over all settings of variables in $V_t \setminus V_t'$. This ensures that after revealing the fraction of errors is not larger than before revealing.

By the above construction the properties (1) and (3) obviously hold after the read phase and revealing. Now let us check how many processors may know the value of an unrevealed variable. A processor may learn a variable $x_l \in V_t$ by reading a cell that knows x_l. There are at most k_t memory cells storing x_l, each cell may be read by less than $\log n$ processors. Hence the number of new processors that may learn x_l is less than $k_t \cdot (\log n - 1)$. Together with k_t processors that has known x_l before, it makes at most $k'_{t+1} = k_t \cdot \log n$ processors knowing x_l at the end of the read phase of step $t+1$. Obviously, the number of cells knowing x_l remains unchanged.

It is still not guaranteed that the length of the longest queue at the read phase is always the same. We only know that for every cell C there is at most one unrevealed variable x_l such that a processor knowing x_l refers C. Therefore we have to reveal some additional variables to fix the length of the longest queue.

Revealing latency.

Let α be the longest queue that may occur during the read phase of step $t+1$ for the chosen values of the revealed variables. If for some cell C such a queue is formed by the processors that always write into C, then we are done – the queue size is always α and it does not leak any information on the unrevealed variables. So the case we have to take care of is when for every cell C where a queue of size α may be formed, the actual size of the queue depends on an unrevealed variable. Let $Z_1, \ldots, Z_m$ be these cells and $d_1, \ldots, d_m$ be the values of unrevealed variables $x_{s_1}, \ldots, x_{s_m}$ such that for $x_{s_j} = d_j$ a queue of length α is formed at Z_j.

Now we consider two cases. First assume that $m > (\frac{c}{\epsilon})^2 \cdot \log n$. Every string $u \in \{0,1\}^m$ can be considered as a node of the m-dimensional hypercube. For a node u we consider the inputs with $x_{s_j} := d_j$, if $u_j = 1$, and $x_{s_j} := \neg d_j$ if $u_j = 0$, for $j \leq m$, and the fixed values of the revealed variables. Then we define $l(u)$ as the fraction of these inputs on which the algorithm succeeds. Let b'_t be the fraction of settings of the unrevealed variables of V'_t, for which the algorithm gives a correct answer. Then obviously b'_t is the average value of all labels $l(u)$. At this moment we need the following combinatorial lemma, proof of which can be found in [10] (see also the appendix):

Lemma 9. *Suppose that every node u of a k-dimensional hypercube H has a label $l(u) \in [0,1]$. Let $\sum_{u \in H} l(u) = 2^k \cdot (0.5 + \epsilon)$ for $\epsilon > 0$. Then there is an $i \leq k$ such that for $F_i = \{(u_1, \ldots, u_k) \in H : u_i = 1\}$ the following inequality holds:*

$$\sum_{u \in F_i} l(u) \;\geq\; 2^{k-1} \cdot (0.5 + \epsilon - \frac{c}{\sqrt{k}}) \tag{5}$$

for some constant c.

Since $b'_t > 0.5$ we may apply Lemma 9 and find a number $j \leq m$ such that after fixing $x_{s_j} = d_j$ the algorithm yields the correct answer for a fraction of at least $b'_t - c/\sqrt{\log m} \geq b'_t - \epsilon/\sqrt{\log n}$. We reveal the variable x_{s_j} and set it to d_j. This value of x_{s_j} guarantees also that the maximal queue has length α for all values of the remaining unrevealed variables.

If $m < (\frac{c}{\epsilon})^2 \cdot \log n$, then applying Lemma 9 decreases the fraction of correct answers of the algorithm too much. Therefore we use another strategy. We choose an $u \in H$ such that $l(u)$ is maximal. Then we reveal all variables $x_{s_1}, \ldots, x_{s_m}$ according to the bits of u (that is, for each $j \leq m$ we put $x_{s_j} = d_j$, if $u_j = 1$, and $x_{s_j} = \neg d_j$ otherwise). If at least one coordinate of u has been 1, then we are done: the fraction of correct answers is at least b'_t, while a queue of length α is formed. The bad case is when $u = (0, \ldots, 0)$. Then no queue of size α is formed. On the other hand, the fraction of the correct answers has not decreased. It may seem that we are in the starting situation, but now the maximal length of a queue, say α', is less than α and we can repeat the same construction for α'. In this way we may be forced to make up to α such iterations, each time decreasing the size of the maximal possible queue. At each iteration we reveal $O(\log n)$ variables. Since $\alpha = O(\log n)$, we may totally reveal only $O(\log^2 n)$ variables. The process described terminates either in the case when $m = m(j) > (\frac{c}{\epsilon})^2 \cdot \log n$ at the jth call, or we decrease the size of the maximal queue to 1 (and then no information is leaked through latency detection).

Let V''_t denote the set of unrevealed variables after revealing some variables of V'_t while dealing with latency of the read phase.

Analysis of a write phase.

Revealing variables at the write phase goes as for the read phase. One can easily derive that afterwards each unrevealed variable is known to at most $k_{t+1} = k_t \cdot (2\log n + 1)$ cells, and that we choose an independent set $V_{t+1} \subseteq V_t''$ of at least $|V_t''|/u_t' - O(\log^2 n)$ elements, where $u_t' = k_t \cdot (4\log n + 3) \cdot (\log n - 1) + 1$.

Achieving the time bound.

Since $k_{t+1} = k_t \cdot (2\log n + 1)$, it follows easily that $k_j \le (e\log n)^j$ for some constant e. We also get

$$\begin{aligned}|V_{t+1}| &\ge \left(|V_t|/u_t - O(\log^2 n)\right)/u_t' - O(\log^2 n)\\ &\ge |V_t|/(d\log^3 n \cdot k_t^2) \ge |V_t|/((d\cdot(e\log n))^{2t+3}).\end{aligned}$$

Thereby $|V_t| \ge n/(\log n)^{f\cdot t^2}$ for some constant $f > 0$. It follows that $|V_t| > 1$ for $t \le \sqrt{\log n/(f\cdot\log\log n)}$. For such t the fraction of good answers has been guaranteed to be at least $0.5 + \epsilon - \frac{\epsilon\cdot t}{\sqrt{\log n}} > 0.5$. So the algorithm may not stop after step t and so $T \ge t$. Hence $T = \Omega(\sqrt{\log n/\log\log n})$. □

3 2-compaction problem and EREW$^+$ model

3.1 Complexity of 2-compaction on the EREW PRAM

Theorem 10. *Suppose that A is a randomized PRAM algorithm that solves a decision problem P in time T. Assume further that the number of possible input configurations for P is m and A succeeds with probability at least $\frac{1}{2} + \epsilon$. Then there is a* **deterministic** *PRAM algorithm A' for P (with the same access modes for reading and writing as A) that runs in time $T + O(\log\log m)$.*

First let us apply Theorem 10 to the 2-compaction problem. There are $\binom{n}{2} + n + 1$ possible inputs for the 2-compaction problem on n input variables, so from a randomized EREW algorithm for 2-compaction running in time T we can get a deterministic EREW algorithm running in time $T + O(\log\log n)$. Since any deterministic EREW PRAM solving the 2-compaction problem requires $\Omega(\sqrt{\log n})$ steps [7] we get $T + O(\log\log n) = O(T)$. So we have obtained the following corollary:

Corollary 11. *The time complexity of 2-compaction is the same (up to a constant factor) on deterministic and randomized EREW PRAMs.*

Proof of Theorem 10. First we show that if the error probability of a randomized algorithm is very small, then a deterministic algorithm running in the same time exists.

Claim A. *Let a randomized PRAM B solve a decision problem R in time L and succeed with probability greater than $1 - \frac{1}{m}$, where m is the number of possible input configurations for R. (We assume that probability distribution on such inputs is uniform.) Then there is a deterministic PRAM B' with the same access mode to the shared memory as B that solves R in time L.*

Indeed, by Lemma 8, there is a deterministic PRAM B' (with the same memory access mode as B) that runs in time L and errs on a fraction of less than $\frac{1}{m}$ of all inputs. But there are only m input configurations, so B' may not err at all.

The second step in the proof is to show that given the randomized algorithm A we may construct an algorithm A' that has the runtime $T + O(\log\log m)$ and errs with probability smaller than $\frac{1}{m}$. PRAM A' works as follows. First we make $k = \alpha \log m$ copies of the input sequence (α is an appropriate constant specified later). Even on the EREW PRAM making the copies can be done in time $O(\log\log m)$. Then on each copy we run independently the algorithm A. After T steps we get k answers, some of them correct, and some of them false. At this moment we perform a majority vote. This takes $O(\log k) = O(\log\log m)$ time on EREW PRAM. Let X be the random variable denoting the number of right answers given by the machines executing algorithm A. Then $E[X] \geq (\frac{1}{2} + \epsilon) \cdot k$. Let us recall the following bound on the sum Z of m independent Bernoulli trials with success probability p (see [1]):

$$\Pr[X \leq (1-\beta)mp] \leq e^{-\beta^2 mp/2}.$$

Hence for $\beta = 2\epsilon/(1+2\epsilon)$, we get

$$\Pr[X < \tfrac{1}{2}k] = \Pr[X < (1-\beta) \cdot k \cdot (\tfrac{1}{2} + \epsilon)] \leq e^{-\beta^2 k(0.5+\epsilon)/2} = e^{-k \cdot \epsilon^2/(1+2\epsilon)}.$$

If $\alpha > \frac{1+2\epsilon}{\epsilon^2 \cdot \log e}$, then the last term is smaller than $1/m$, as required in order to apply Claim A. □

3.2 Tightness of the bounds and EREW$^+$ PRAMs

Proposition 12. *2-compaction problem can be solved in time $O(\sqrt{\log n})$ on the EREW$^+$ PRAM.*

Proof. The computation consists of at most $O(\sqrt{\log n})$ rounds. After round i:

(i) The input cells are partitioned into blocks $C_1^i, C_2^i, \ldots$ of size $2^{0.5 \cdot i \cdot (i-1)}$.

(ii) With each block C_j^i we associate auxiliary cells $Z_{j,1}^i, \ldots, Z_{j,2^i}^i$. These cells store the position of a nonempty input cell in C_j^i, if there is any.

The computation is interrupted when 2 nonempty input cells are found. We describe round $i+1$. Let $C_j^{i+1} = C_{(j-1)\cdot 2^i+1}^i \cup \cdots \cup C_{j \cdot 2^i}^i$. Then the following steps are performed for each j:

Step 1. For $k < l \leq 2^i$, we check if $Z_{(j-1)\cdot 2^i+k,l}^i$ and $Z_{(j-1)\cdot 2^i+l,k}^i$ both point to nonempty input cells. If it is so, then these cells are moved to the output positions (since there is at most one such pair (k,l), no write conflict arises) and the machine halts.

Step 2. If the machine survived Step 1, then there is at most one nonempty cell in C_j^{i+1}. For each $k, l \leq 2^i$ if $Z_{(j-1)\cdot 2^i+k,l}^i$ contains a pointer to a nonempty cell, then it is copied into $Z_{j,2l-1}^{i+1}, Z_{j,2l}^{i+1}$. Obviously, no write conflict may occur since otherwise the machine would stop at Step 1.

Finally, if all input cells are in a single block and the machine did not halt, then we know that there is at most one nonempty input cell. Its address can be found in the first auxiliary cell. There are at most $O(\sqrt{\log n})$ phases, each consisting of $O(1)$ steps, so the algorithm runs in $O(\sqrt{\log n})$ steps. □

Some slight modifications of the result for EREW PRAM in [7] yield the following result showing that the above algorithm is optimal.

Proposition 13. *2-compaction requires $\Omega(\sqrt{\log n})$ steps on EREW$^+$ PRAM.*

3.3 Complexity of PARITY on randomized EREW$^+$ PRAM

To shed some light on the limitations of the EREW$^+$ PRAM let us mention the following results concerning time complexity of PARITY_n in this model:

Theorem 14. *Computing* PARITY_n *on EREW$^+$ PRAM requires $\Omega(\log n)$ time. On the randomized EREW$^+$ PRAM* PARITY_n *has time complexity $\Omega(\sqrt{\log n})$.*

The above theorem can be proved using techniques that we have developed for the QRQW PRAM.

Acknowledgments. We are deeply grateful to Artur Czumaj and ToMasz Wierzbicki for many fruitful discussions.

References

1. D. Angluin, L.G. Valiant, Fast probabilistic algorithms for Hamiltonian circuits and matchings, *J. Comput. System Sci.* **18** (1979) 155–193.
2. P. Beame, J. Håstad, Optimal bounds for decision problems on the CRCW PRAM, *JACM*, **36(3)** (1989) 643–670.
3. C. Berge: *Graphs and Hypergraphs.* North–Holland, Amsterdam, 1976.
4. S. Cook, C. Dwork, R. Reischuk, Upper and lower time bounds for parallel random access machines without simultaneous writes, *SIAM J. Comput.* **15(1)** (1986) 87–97.
5. M. Dietzfelbinger, M. Kutyłowski, R. Reischuk, Exact lower time bounds for computing Boolean functions on CREW PRAMs, *JCSS* **48(2)** (1994) 231–253.
6. F.E. Fich, The complexity of computation on the Parallel Random Access Machine, in *Synthesis of Parallel Algorithms*, J.H. Reif (ed.) (Morgan Kaufmann, San Mateo, 1993) 843–899.
7. F. Fich, M. Kowaluk, M.Kutyłowski, K. Loryś, P. Ragde, Retrieval of scattered information by EREW, CREW and CRCW PRAMs, in *Algorithm Theory - SWAT'92*, LNCS vol. 621, (Springer, Berlin, 1992), 30-41, a journal version to appear in *Comput. Compl.*.
8. P. Gibbons, Y. Matias, V. Ramachandran, Efficient low-contention parallel algorithms, in *Proc. 6th ACM Symp. on Parallel Algorithms and Architectures*, (ACM Press, New York, 1994) 236–247.
9. P. Gibbons, Y. Matias, V. Ramachandran, The QRQW PRAM: accounting for contention in parallel algorithms, in *Proc. 5th ACM Symp. on Discrete Algorithms*, (ACM Press, New York, 1994) 638–648.
10. M. Kutyłowski, K. Loryś, manuscript, `http://www.uni-paderborn.de/cs/mirekk.html#manuscript`
11. P. D. MacKenzie, Lower bounds for randomized exclusive write PRAMs, in *Proc. 7th ACM Symp. on Parallel Algorithms and Architectures*, (ACM Press, New York, 1995) 254–263.
12. R. Motwani, P. Raghavan: *Randomized Algorithms*, Cambridge University Press, Cambridge 1995.
13. H. Robbins, A remark on Stirling formula, *American Mathematical Monthly* **62** (1955) 26–29.

Appendix

For the referee's convenience we include the proof of Lemma 9 given in [10]. First we check one inequality.

Lemma 15. $\binom{k}{k/2} \cdot 2^{-k} \leq c \cdot \frac{1}{\sqrt{k}}$ *for some constant* c.

Proof. The following Stirling formula approximates $h!$ [13]:

$$\sqrt{2\pi} \cdot h^{h+0.5} \cdot \frac{1}{e^{h-1/(12h+1)}} < h! < \sqrt{2\pi} \cdot h^{h+0.5} \cdot \frac{1}{e^{h-1/12h}}.$$

Therefore

$$\begin{aligned}\binom{n}{n/2} &= \frac{n!}{((n/2)!)^2} \leq \frac{\sqrt{2\pi} \cdot n^{n+0.5}}{e^{n-\frac{1}{12n}}} \cdot \left(\frac{e^{n/2-\frac{1}{6n+1}}}{\sqrt{2\pi} \cdot (n/2)^{n/2+0.5}} \right)^2 \\ &= \frac{1}{\sqrt{2\pi}} \cdot \frac{n^{n+0.5}}{(n/2)^{n+1}} \cdot \frac{e^{n-\frac{2}{6n+1}}}{e^{n-\frac{1}{12n}}} \\ &= \frac{1}{\sqrt{2\pi}} \cdot \frac{1}{\sqrt{n}} \cdot 2^{n+1} \cdot \frac{1}{e^{2/(6n+1)-1/(12n)}} = \frac{1}{\sqrt{2\pi}} \cdot \frac{1}{\sqrt{n}} \cdot 2^{n+1} \cdot e(n)\end{aligned}$$

where $e(n) \approx 1$. Then

$$\binom{n}{n/2}/2^n \leq \sqrt{\frac{2}{\pi}} \cdot e(n) \cdot \frac{1}{\sqrt{n}}.$$

□

Proof of Lemma 9. Consider a function $l' : H \to [0,1]$ such that $\sum_{u \in H} l'(u) = \sum_{u \in H} l(u)$ and the value $\gamma = \sum_{i \leq k} \sum_{u \in F_i} l'(u)$ is minimal. Let $u \in H$ and s be the number of coordinates of u that are ones. Then we say that $\text{weight}(u) = s$. If $\text{weight}(u) = s$, then the node u belongs to s different sets F_i and thereby $l'(u)$ is counted s times in γ.

Claim. *It is impossible that* $\text{weight}(u_1) < \text{weight}(u_2)$, $l'(u_1) < 1$ *and* $l'(u_2) > 0$.

If it would be so, then we could decrease $l'(u_2)$ and increase $l'(u_1)$ by the same amount, and thereby decrease γ. It follows that for a certain d, we have $l'(u) = 1$ if $\text{weight}(u) < d$, and $l'(u) = 0$ if $\text{weight}(u) > d$. For the sum $\gamma = \sum_{i \leq k} \sum_{u \in F_i} l'(u)$ it is immaterial how we assign the values $l'(u)$ for the nodes of the weight d, as long as they are in $[0,1]$ and their sum does not change. So we assume that all nodes of weight d have the same value of l', say p. Since $\gamma \leq \sum_{i \leq k} \sum_{u \in F_i} l(u)$, it follows that there is an $i \leq k$ such that $\sum_{u \in F_i} l(u) \geq \gamma/k$. Therefore it suffices to estimate γ/k. We have

$$2^k \cdot (0.5 + \epsilon) = \sum_{u \in H} l(u) = \sum_{u \in H} l'(u) = \binom{k}{0} + \ldots + \binom{k}{d-1} + p \cdot \binom{k}{d}.$$

Note that $\gamma/k = \sum_{u \in F_i} l'(u)$. Since F_i contains $\binom{k-1}{j}$ nodes of weight $j+1$, we get

$$\begin{aligned}\gamma/k &= \binom{k-1}{0} + \ldots + \binom{k-1}{d-2} + p \cdot \binom{k-1}{d-1} \\ &= \tfrac{1}{2} \cdot \left(2\binom{k-1}{0} + \ldots + 2\binom{k-1}{d-2} + 2p \cdot \binom{k-1}{d-1}\right) \\ &= \tfrac{1}{2} \cdot \left(\left(\binom{k}{0} + \ldots \binom{k}{d-2}\right) + \binom{k-1}{d-2} + 2p \cdot \binom{k-1}{d-1}\right) \\ &= \tfrac{1}{2} \cdot \left(2^k \cdot (0.5+\epsilon) - \binom{k}{d-1} - p \cdot \binom{k}{d} + \binom{k-1}{d-2} + 2p \cdot \binom{k-1}{d-1}\right) \\ &\geq \tfrac{1}{2} \cdot \left(2^k \cdot (0.5+\epsilon) - 2\binom{k}{d}\right) \geq 2^{k-1} \cdot \left(0.5 + \epsilon - \tfrac{c}{\sqrt{k}}\right)\end{aligned}$$

(the last inequality follows from Lemma 15.) □

This article was processed using the LaTeX macro package with LLNCS style