Efficient Parallel Computing with Memory Faults

Leszek Gąsieniec[1] and Piotr Indyk[2]

[1] Max-Planck Institut für Informatik, Im Stadtwald, Saarbrücken D-66123 Germany. leszek@mpi-sb.mpg.de
[2] Computer Science Department, Stanford University. indyk@cs.stanford.edu

Abstract. In this paper we show two results on PRAM with constant fraction of memory faults. First we show how to *preprocess* (i.e. connect a constant fraction of processors into a binary tree) a faulty EREW PRAM with $n/\log n$ processors and $O(n)$ memory cells in $O(\log n)$ time. The preprocessing is a basic step of simulations from [7,9,17]. Our algorithm, together with the results from [17], gives a first fully work-optimal randomized simulations of EREW on EREW with faults with logarithmic overhead. In the second part of this paper, we consider the CRCW PRAM with memory faults. We show that (after $O(\log n)$-time preprocessing) any algorithm for $O(n)$-processor PRAM can be simulated with optimal work in $O(\log^* n)$ time on CRCW with memory faults. The simulation improves the result of [7]. All simulations assume static faults, i.e. that the errors are determined before the computation starts and that no new errors occur during the computation.

1 Introduction

The increasing complexity of multiprocessor computers makes the machines prone to hardware failures. This necessitates the design of algorithms that are resilient to faults in the computation environment. The most natural solution is to deliver a general simulation mechanism of an operational machine on its faulty counterpart. In recent years a lot of research was done to develop such simulations on various computational models (PRAM, DMM, asynchronous machines) resilient to processor [20,23,22,11,19,?] or memory [2,7,9,17,5,8] faults.

In this paper we show two results on PRAM with constant fraction of memory faults. First we show how to *preprocess* (i.e. connect a constant fraction of processors into a binary tree) a faulty Exclusive-Read Exclusive-Write (EREW) PRAM with $n/\log n$ processors and $O(n)$ memory cells in $O(\log n)$ time. The preprocessing is a basic step of simulations from [7,9,17]. Our algorithm, together with the results from [17], gives a fully work-optimal randomized simulations of EREW on EREW with faults (i.e. the simulation which enable to execute any n-processor T-step EREW algorithm on $n/\log n$- processor EREW with faults in time $T\log n$). No such simulation was known before, even for a Concurrent-Read Concurrent-Write (CRCW) PRAM. We present both randomized and deterministic algorithms for preprocessing. The latter one is of special interest, as it is based on the existence of family of graphs with the interesting property that the number of vertices in subgraphs with small diameter can be approximated by the number of paths. We prove the existence of such graphs using the probabilistic method. Though this gives only an existential result (and therefore implies that the algorithm is non-uniform), it also gives a fast randomized off-line algorithm for constructing of such graphs. The advantage of the deterministic solution is that together with the result of [9] it gives the first deterministic simulation of EREW on EREW with faults with logarithmic overhead, regardless of the running time of the simulated algorithm.

In the second part of this paper, we consider the CRCW PRAM with memory faults. We show that (after $O(\log n)$-time preprocessing) any algorithm for $O(n)$-processor PRAM can be simulated with optimal work in $O(\log^* n)$ time on CRCW with memory faults. This fact is quite surprising, as it means that the memory faults are negligible as far as computation on CRCW is concerned. The simulation improves the result of [7].

All simulations assume static faults, i.e. that the errors are determined before the computation starts and that no new errors occur during the computation. For other assumptions about the computation model, see subsection 1.2.

The rest of this paper is organized as follows. In section 2 we define basic notation and introduce probabilistic techniques used later in the analysis of presented simulations. In section 3 we present the $O(\log n)$-time preprocessing for the simulation of computations on faulty EREW PRAM. In section 4 we give an optimal $O(\log^* n)$-time simulation for CRCW PRAM. Section 5 contains final remarks.

1.1 Related work

The study on PRAMs with faulty processors was initiated by Kanellakis and Shvartsman [20] (for the deterministic distribution of faults) and Kedem, Palem and Spirakis [23] (for probabilistic faults). They considered *fail-stop dynamic* faults, i.e. faults which may happen at any time during the computation and which causes the processor to stop performing its task. For the same model of faults Kedem *et al.* [22] designed robust simulations which have constant expected slowdown, under some assumptions on the faults probability. Diks and Pelc [11] developed reliable and efficient algorithms for the EREW PRAM, also under the probabilistic processor-fault model. Kanellakis, Michailidis and Shvartsman [19] developed optimal simulation on EREW PRAM when all processor faults are initial (i.e static in our notation). Buss *et. al* [4] developed deterministic robust PRAM emulations under the model of restartable failures.

The problem of designing efficient simulations with shared memory faults was first considered by Afek *et al.* [2]. They gave a general simulation mechanism working in asynchronous enviroment with arbitrary faults and posed a problem of designing of more efficient simulations for synchronous systems. They also discussed the possible simplification of their model, for example by restricting all faults to be static. The first analysis of computational power of PRAM with faulty memory appeared in Chlebus, Gambin and Indyk [7]. They assumed an arbitrary constant fraction $0 < q < 1$ of deterministic memory errors and designed efficient PRAM simulations for both the dynamic and static case. For static errors, they presented an algorithm for the simulation of n-processor CRCW PRAM on $n/\log n$-processor CRCW with $O(\log n)$ step overhead. The simulation is preceeded by a $O(\log^{3/2} n)$-time preprocessing phase. Also, they gave an algorithm for performing the simulation in real time. However, the number of processors of the faulty machine grows to $O(n \log n)$.

The faulty-memory simulations mentioned above were randomized. Chlebus, Gąsieniec and Pelc [9] presented a deterministic simulation of n-processor EREW PRAM on n-processor EREW prone to processor and memory failures, with step overhead $O(\log n)$ and preprocessing time $O(\log^2 n)$. By using randomization the number of processors was reduced by Indyk [17] to $n/\log n$ without changing the step overhead, the preprocessing time however increased to $O(\log^3 n)$. Recently, Chlebus, Gambin and Indyk [8] and Berenbrink, Meyer auf der Heide and Stemann [5] obtained several simulations of EREW PRAM on Distributed Memory Machine [27] for static and dynamic models of faults.

1.2 Our model

In this paper we make the following assumptions about the simulation enviroment (similar assumptions were made in [7,9,17]) :

• a constant fraction $0 < q < 1$ of memory cells may fail,
• the faults are static, i.e. the errors are determined before the computation starts and no new errors occur during the computation,
• there exists a mechanism for detection of faulty memory cells during the read/write operations,
• the memory word contains $\Theta(\log n)$ bits.

2 Preliminaries

We use the following notation through the rest of the paper. Let $\Pi = \{\mathcal{P}_1, .., \mathcal{P}_n\}$ be the set of processors of the simulated machine and $\lambda = \log n$. We also use emphasized letters $P_1, P_2, P_3...$ to denote processors of the simulating machine.

Definition: Bounded-differences inequality (BDI)
Let $X_1, \ldots, X_r$ be a sequence of independent random variables, where the values of X_k are in a finite set A_k, and a random variable Y is defined as $Y = f(X_1, \ldots, X_r)$, for a function f. If $|f(x) - f(x')| \leq c_k$, for every $1 \leq k \leq r$ and any vectors x, x' which differ only at the kth coordinate, then, for any $t > 0$, the following *bounded-differences inequality* (BDI) holds:

$$\Pr[|Y - E(Y)| \geq t] \leq 2\exp(-2t^2/\sum c_k^2)$$

(see [26]).

Definition: Chernoff bound
Suppose that all the random variables X_k are Bernoulli trials with $\Pr[X_k = 1] = p$ and $\Pr[X_k = 0] = 1 - p$. Let $Y = \sum_{k=1}^{r} X_k$. Then $E(Y) = r \cdot p = \mu$, and the following *Chernoff bound* [16] holds, for $0 < \varepsilon < 1$:

$$\Pr[|Y - \mu| \geq \varepsilon \cdot \mu] < 2\exp(-\varepsilon^2 \mu/3).$$

Definition: A property Φ depending on a natural number n is said to hold *with a high probability* (abbreviated to w.h.p.) if, for some constant $\delta > 0$, Φ holds with the probability at least $1 - n^{-\delta}$, for sufficiently large n. Whenever this phrase is used, it is understood that δ can be made arbitrarily large by manipulating other constants.

Definition: Let $G = (V, E)$ be any graph. We say that G is an (α, β)*-expander*, if for any $S \subset V$ such that $|S| \leq \alpha|V|$ the set $\Gamma(S)$ of neighbors of S in G satisfies $|\Gamma(S) - S| \geq \beta|S|$.

3 Fast preprocessing for EREW with static faults

In this section we present a $O(\log n)$-time algorithm for preprocessing of faulty EREW with $O(n/\log n)$ processors. For the sake of presentation we give first a (non-optimal) algorithm using $O(n)$ processors and then describe the neccessary modification to achive the work-optimality. Following [9] we assume that the number of memory cells in the simulated machine is also $\Theta(n)$ (this assumption may be relaxed by increasing the time of the preprocessing using techniques from [17], we do not go into details here). The non-optimal simulation proceeds as follows. Every processor in the simulating machine is associated with a constant number of consecutive memory cells. We say that a processor is *active* if one of its associated memory cells is fault-free. It is not difficult to show that a fraction $0 < (1 - q) < 1$ of all processors are active.
The main goal of the preprocessing is to create a binary tree of all active processors. In fact, it is sufficient just to create a linked list of such processors, as the binary tree can be constructed using the simulation algorithm of [9]. As in that paper, during the preprocessing we might also need to decode the input data for the simulated algorithm, which was assumed to be stored in an encoded form in the faulty memory. In [9,17] the Information Dispersal Algorithm of Rabin [28] was used. This algorithm works in time $O(\log^2 n)$, so we cannot use it here. Moreover no codes with $O(\log n)$ time and linear work decoding scheme are known (the best algorithm by Sipser and Spielman [31] is not optimal). To overcome this difficulty, we may assume that the input data is provided in the consecutive operational memory cells, hence eliminating the necessity of decoding procedure. Alternatively one may introduce a special hardware mechanism which provides the input data. In the following we skip this issue, and concentrate on the list-creation procedure.
Recall that in PRAM processors communicate by shared memory. Thus by communication between processors $\mathcal{P}_i$ and $\mathcal{P}_j$ we mean that $\mathcal{P}_i$ modifies some values of memory cells associated with P_j and vice versa. Since it is not known in advance which processors in the simulating machine are active (and which are not) we restrict, for our purpose, communication abilities between processors of simulating machine. We embed the set of processors of the simulating machine onto nodes of a constant degree graph from the family defined in [1] (based on Ramanujan graphs from [24]). The family has the following fault-tolerant property: for any $0 < \varepsilon < 1$ there exist constants c, d and α such that for any n the family contains a graph with $n' \leq dn$ vertices such that if we remove $\varepsilon n'$ vertices from G, then the remaining graph contains a subgraph G' with at least cn' vertices, which has an $(\alpha, 2)$-expansion property. In fact, we modify this graph slightly in order to have a better expansion property (we will give the details later). The (fault-tolerant) property of G' guarantees that there is a vertex having $\Omega(n)$ vertices in its neighborhood of radius $O(\lambda)$. In the sequel it is assumed that $\varepsilon = (1 - q)$.
The preprocessing is divided into two parts. During the first part every processor P checks if its close neighborhood (the set of nodes in G which are of distance $O(\lambda)$ from P) contains a large number (at least $s_1 n$, for some constant $s_1 > 0$) of active processors. There are two methods for doing this. One (which we show below) is randomized and the other one (described later) is deterministic. After this stage processors with large number of active neighbors become *local leaders*. During the second part of the preprocessing local leaders take part in global duels and the global leader is selected. Then the tree-structure of all active processors is created with the root labeled by the global leader. Finally the tree is transformed into a list of active processors by using the Euler tour technique [14].
We start with a more detailed description of the first part of preprocessing. We assume that every processor (in the simulating machine) has its own bit mask of size $c_2\lambda$, for some constant c_2. Originally

all positions in the bitmask of every processor are set to 0.

PART 1 Selection of local leaders

- each processor P becomes marked with probability $\frac{c_1\lambda}{n}$, for some constant c_1 specified later,
- each marked processor chooses a random (integer) value ξ from the range $1 \ldots c_2\lambda$ and sets the ξ's position in its bitmask to 1, all other positions in the mask being left unchanged,
- the processors communicate with their neighbors (in G) during $r = d\lambda$ round (each round consists of a constant number of matchings (connections in pairs) which cover all edges of G). During each round the processors exchange information about their bitmasks. At the end of the round, each processor computes a bitwise OR of all the masks collected from its neighbors. The result becomes a new bitmask of the processor.

For every processor P_i of the simulating machine, let N_i be the neighborhood of P_i of radius r in the graph G after deletion of vertices and let $n_i = |N_i|$. Moreover let M_i be the set of marked processors in neighborhood N_i and $m_i = |M_i|$. Denote by B_i the bit mask held by the ith processor and let b_i be the number of bits in B_i set to 1.

Lemma 1. (Sampling lemma)
There exist constants s_1, s_2 and t such that w.h.p. for each i the following conditions hold:
1. if $n_i \geq s_1 n$ then $b_i \geq t\lambda$,
2. if $n_i < s_2 n$ then $b_i < t\lambda$.

Proof. Follows from Chernoff bound, details are omitted.

All processors P_i for which $n_i \geq s_1 n$ become local leaders. All of them have a large neighborhood w.h.p. At the beginning of the second part of preprocessing every local leader keeps its index as an exchange value and the rest of processors keep value 0.

PART 2 Construction of a list

- the processors communicate with their neighbors during $2r+1$ rounds. During every round processors exchange their values and both of them keep the larger one.
- the processor who won all duels (i.e. which keeps its initial value) becomes a global leader (below we show that there is only one such processor w.h.p.),
- a BFS tree of depth r with a root at the global leader is created (in $O(r)$ rounds),
- the BFS tree is transformed into a list using the Euler tour technique [14].

Lemma 2. *The global leader is unique w.h.p.*

Proof. ¿From Lemma 1 it follows that if a node is elected as a leader, than w.h.p. it has a neighborhood of size at least $s_2 n$. It is a well known fact, that for any $0 < \alpha < 1$ and $0 < \beta < 1$, there exists a bounded degree graph $G'' = (V'', E'')$ such that for any set $S \subset V''$ with $|S| = \alpha n$ we have $|\Gamma(S)| \geq \beta n$ (for example, Gabber and Galil [13] constructed a graph with a property that $|\Gamma(S) - S| \geq d(1 - |S|/n)$ for some constant $d > 0$. Taking a product of their graph increases the number of neighbors by some constant fraction (provided $|S| \leq \beta n$), so a constant number of multiplications suffice to get a neighborhood large enough. We may hence take G'' to be some constant power of the graph from [13]). For our purpose we need a graph with the specified property for $\alpha = s_2$ and $\beta > (1 - s_2)$. It is easy to see that for any two sets $S_1, S_2 \subset V''$ s.t. $|S_i| \geq s_2 n$ for $i = 1, 2$ there has to be at least one edge in E'' leading from S_1 to S_2. Hence, if (besides G) we also use G'' for communication purposes we ensure that every pair of local leaders are within a distance of $2r+1$ of each other, hence the leaders will communicate with each other during the global duel phase and only one of them will survive.

Theorem 3. *There is a randomized algorithm which w.h.p. performs preprocessing of an $n/\log n$-processor EREW PRAM with static memory faults in $O(\log n)$ time.*

Deterministic preprocessing. There exists a method for performing the neighborhood counting stage in a deterministic way. However, this method requires the underlying graph G to posses an additional property (besides the fault-tolerant property as above). To define this property, first let us say that a path in a graph is *good* if it does not contain cycles of length 1 or 2. For the graph G we require that there exists r such that for every vertex v the number of good paths of length r starting from v is approximately equal to the number of vertices within distance r from v. Moreover we require that the number of such vertices should be linear in n.
We don't know of any explicit family of graphs having this property. However, by the probabilistic method we show that such graphs exist.

Definition: The family $\mathcal{G}_{d,n}$ contains all undirected graphs $G = (V, E)$ with n vertices obtained by choosing d permutations $\pi_1 \ldots \pi_d$ of $\{1 \ldots n\}$ and forming E by collecting edges from v to $\pi_i(v)$, for $i = 1 \ldots d$.

Let $S'_{G,i}(v)$ (or S'_i when v and G are fixed) be the set of vertices in G within distance i from v, and let $S_{G,i}(v)$ (or S_i) be the set of vertices from S'_i with a unique shortest paths of length i to v. Then the following lemma holds.

Lemma 4. *For any $0 < \gamma < 1$ there exists a constant c_r and degree d such that if we choose $G = (V, E)$ from $\mathcal{G}_{d,n}$ at random, then with probability tending to 1 for every vertex $v \in V$ the inequality $|S_r(v)| \geq (d-1)^r(1-\gamma)$ holds, for $r = \log_{d-1} n - c_r$.*

Proof. Omitted.

The graph G from the Lemma 4 has the desired approximation property. Let v be any vertex of G. For any $r > 1$ let $H_r(v)$ (or H for short) be the subgraph of G induced by $S'_r(v)$. Also, let $H'_r(v)$ (or H') denote any connected subgraph of H containing v. Let Δ denote the difference of orders of H and H'. Finally, let $P_{G,i}(v)$ be the set of path of length i in G starting from v.

Definition: The graph G is an (α, r)-approximator if for any vertex v and any $H'_r(v)$ as above

$$|S_r^{H'}(v)| \geq \alpha |P_r^{H'}(v)| - (1-\alpha)\Delta.$$

Lemma 5. (approximation lemma)
For any $0 < \alpha < 1$ there exists constants $0 < c < 1$, $d > 1$ and $n_0 \geq 1$, s.t. for any $n > n_0$ there exists a graph G of degree d and order n having (α, r)-approximation property for $r = log_{d-1} n - c$.

Proof. Take the graph from lemma 4 and consider any vertex v. Clearly $|P_r^H(v)| \leq \sum_{i=1}^r d(d-1)^{i-1} \leq (1 + \frac{2}{d-2})(d-1)^r$, so $|P_r^{H'}(v)| \leq |P_r^H(v)| - \Delta = (1 + \frac{2}{d-2})(d-1)^r - \Delta$. On the other hand (by Lemma 4) $|S_r^H(v)| \geq (1-\gamma)(d-1)^r$, hence $|S_r^{H'}(v)| \geq |S_r^H(v)| - \Delta \geq (1-\gamma)(d-1)^r - \Delta$. It follows that

$$|S_r^{H'}| \geq \alpha |P_r^{H'}| - (1-\alpha)\Delta$$

for $\alpha = \frac{1-\gamma}{1+2/(d-2)}$. As γ may be arbitrarily close to 1 and d arbitrarily large, the lemma follows.

We also have to make sure that after deleting a fraction of G's vertices it still contains a subgraph G' having an expansion property. However, it is easy to show this fact by using the following two lemmas.

Lemma 6. *Any graph G on n vertices with average degree d contains an induced subgraph H such that for every vertex-set S of H the number of edges incident to vertices in S is at least $d|S|/2$.*

Proof. See [1].

Lemma 7. *For any $0 < \beta < 1$ and degree d there exists $0 < \alpha < 1$ such that for any n a graph G randomly chosen from $\mathcal{G}_{d,n}$ has $(\alpha, \beta d)$-expansion property w.h.p.*

Proof. Folklore, see [31] for even tighter analysis of expansion of random graphs.

Corollary 8. *For any $0 < \varepsilon < 1$ there exists constants c, d and α such that for any n, if we choose G from $\mathcal{G}_{d,n}$ at random, then w.h.p. if we remove εn vertices from G, then the remaining graph contains a subgraph G' with at least cn vertices, which has an $(\alpha, (1-\varepsilon)d/4)$-expansion property.*

Proof. The average degree of G after removing the vertices is $(1-\varepsilon)d$, hence (from Lemma 6) G contains a subgraph $G' = (V', E')$ such that for every $S \in V'$ there are at least $(1-\varepsilon)d|S|/2$ edges incident to S. On the other hand (from Lemma 7), we know that w.h.p. the original graph G had (α, β)-expansion property and we may assume that $\beta = 1 - (1-\varepsilon)/4$. It follows that G has $(\alpha, (1-\varepsilon)d/4)$-expansion property, because otherwise there have to be a set $|S| \leq \alpha n$ such $|\Gamma(S) - S| < (1-\varepsilon)d|S|/4$. However, it implies that in the original graph we have $|\Gamma(S) - S| < (1-\varepsilon)d|S|/4 + (1-(1-\varepsilon)/2)d|S|$ (as we had at most $(1-(1-\varepsilon)/2)d|S|$ edges incident to S more in G than G'), hence $|\Gamma(S) - S| < (1-(1-\varepsilon)/4)d|S| = \beta d|S|$, which is a contradiction.

Let s_1 and n_i be defined as previously, and let b_i denote the number of good paths of length r with one endpoint in v_i in the graph after the deletion of vertices. The value of b_i is easy to compute as follows. Let $P_r^v(u)$ be the set of good paths of length r starting with edge $u - v$. It is easy to see, that the following recurrenct formula holds :

$$|P_r^v(u)| = \sum_{w \neq u} |P_{r-1}^w(v)|$$

As this formula can be computed in a constant time in a constant degree graph, we can compute $|P_r^v(u)|$ in time $O(r)$. Finally we compute $b_i = \sum_v |P_v^r(u_i)|$.
To estimate n_i we can now use the following

Corollary 9. *For the graphs satisfying the above properties, there exist constants s_1, s_2 and t such that for each i the following conditions hold :*
1. if $n_i \geq s_1 n$ then $b_i \geq tn$,
2. if $n_i < s_2 n$ then $b_i < tn$.

Proof. Take s_1, s_2 and t such that $\frac{1}{\alpha}(s_2 + (1-\alpha)) \leq t \leq s_1$ (clearly such a choice exists for α sufficiently close to 1). Then if $b_i < tn$, then clearly $n_i < tn \leq s_1 n$, as $n_i \leq b_i$. On the other hand, if $n_i < s_2 n$, then (from lemma 5) $b_i < \frac{1}{\alpha}(n_i + (1-\alpha)n)$, as clearly the number of deleted vertices is at most n. As $n_i < s_2 n$, it follows that $b_i < \frac{1}{\alpha}((s_2 + (1-\alpha))n < tn$.

Theorem 10. *There exists a deterministic algorithm which performs preprocessing of an $n/\log n$-processor EREW PRAM with static memory faults in $O(\log n)$ time.*

Optimal algorithms. To make our algorithms optimal, it is sufficient to split the memory into $O(n/\log n)$ blocks of logarithmic size and associate each of $n/\log n$ processors with each block. We now call the processor active, if its block is *good*, i.e. contains some fixed fraction of operational memory cells. By an easy counting argument one can see that at least a constant fraction of the processors are active. Each processor can easily detect its status in time $O(\log n)$, by scanning the cells of its block. The remaining part of the above algorithms can be easily modifed to capture the new situation.

4 Optimal $O(\log^* n)$-time simulation for CRCW

In this section we present a simulation of n-processor CRCW on $n/\log^* n$-processor CRCW with $O(\log^* n)$ step overhead. For the sake of presentation, we will present first the simulation of *EREW* using n processors working in time $O((\log^* n)^2)$. Then we sketch how to achive optimal simulation by resorting to techniques of Bast and Hagerup [3]. Finally, we show how to use the algorithms to simulate CRCW PRAM.

Let $l = O(\lambda)$. As in [7], the memory cells of the simulated PRAM are stored in $O(m/\log n)$ blocks of the memory of the simulated machine, in such a way that one blocks stores $\log n$ consecutive memory cells. To access a memory cell within the block, we need a pointer to this block plus some additional information, both of which can be stored in one memory cell. We store this information in cells $D_1(x), .., D_l(x)$, for some random functions $D_i : [1 \ldots m] \to [1 \ldots M]$, where $M = O(m)$ is the memory of the simulated machine. We assume that the D_i are chosen from the family of functions defined by Siegel [29], and hence have the k-independence property for $k = O(\log^c n)$ for some constant c, i.e. for all i and for all pairs $(x_1, .., x_k \; ; \; y_1, .., y_k)$ of k-sequences

$$\Pr[D_i(x_1) = y_1, D_i(x_2) = y_2, ..., D_i(x_k) = y_k] = \frac{1}{M^k}.$$

Moreover such functions can be stored in an array T of size $O(n^\varepsilon)$, for any constant $\varepsilon > 0$, and for any x the value $D_i(x)$ can be computed in a constant time. Since we cannot use continuous blocks of memory cells due to memory faults, we represent the array T by keeping $O(\log n)$ copies of it at random positions of memory. When the access to the item $T[z]$ is desired, a random copy of $T[z]$ is accessed. The access to $T[z]$ is successful with constant probability, which means that each attempt to compute the value of $D_i(x)$ in constant time is succesful with constant probability. The copies of T are created during the preprocessing.

For purpose of the simulation all processors are divided into groups of size $O(k)$. We assume that each group has an associated memory of size $O(k)$ with a constant time access. The assumption can be satisfied by assigning to each group a good block of polylogarithmic length (i.e. a block of memory cells containing a constant fraction of operational cells). An access to any of such cells can be then performed in a constant time using techniques of [17]. The block assignment is performed during preprocessing.

Consider some specific group G. During each step of simulation, each processor from G (say processor P) has to find a memory cell simulating some memory cell (say x). To this end it tries to find some operational cell $D_i(x)$. The searching process consists of phases. In the first phase each processor from G independently tries to get the information which allows it to read $D_1(x)$. If it was successful, then it checks memory cell $D_1(x)$ if it holds the contents of x (we call the above two steps an *attempt*). Notice that for every processor the probability that its attempt was successful is constant. Since the function D_1 is $O(k)$-independent and as all processors attempt to access different x (as we simulate EREW), by using standard techniques for random variables with limited independence (for example see [30]) we obtain that a constant fraction of processors in G had a successful attempt w.h.p. and finish their search. The unsuccessful processors next have to try the functions D_2, D_3, etc during subsequent phases. Their search can be supported by the processors which already have finished their search. In other words, we need to make an allocation of successful processors to the ones which still have to perform searching. Such an allocation algorithm was proposed by Bast and Hagerup [3]. Their algorithm works in time $O(\log^* n)$ with probability $1 - 2^{-|G|^{\frac{1}{4}}}$. Since the size of G is $O(\log^c n)$, thus for a proper constant c we achieve a high probability.

Assume that after the allocation every searching processor gets $O(h)$ additional helpers. The helpers try to reach yet untried values $D_i(x), D_{i+1}(x), .., D_{i+h-1}$ simultaneously. Now the probability of success is $1 - 2^{-\Theta(h)}$ thus again the fraction of unsuccessful processors is at most $2^{-\Theta(h)}$ w.h.p. After the allocation step each unsuccessful processor gets $2^{\Theta(h)}$ helpers, and so on. After $O(\log^* n)$ phases the fraction of unsuccessful processors becomes $O(1/\log n)$, so the number of helpers becomes $\Omega(\log n)$ and after yet another phase all processors become successful w.h.p. Since there are $O(\log^* n)$ allocations and each costs $O(\log^* n)$ we obtain the running time $O((\log^* n)^2)$.

The above solution can be transformed into the optimal one in time $O(\log^* n)$ using techniques from [3]. They proposed an abstract scheme for optimal $(\log^* n)$-time algorithms. In this scheme, the input consists of a number of *active objects* stored in an array of size n, and the goal is to *deactivate* all objects. To this end, we need to have a constant-time procedure *Deactivate*, which deactivates most objects w.h.p. The objects are permanently partitioned into *groups*, and *Deactivate* operates on group-by- group basis, either deactivating all objects in a group or leaving them all active (a group is called

active when its objects are active). The sufficient requirement for the procedure *Deactivate* is that if it is called when the total number of active objects is $O(n/v)$, then the probability that a fixed group remains active is at most 2^{-v}. The probability bound is then used to show (by using the Azuma inequality, which is a variant of BDI) that the fraction of active objects becomes $O(n/2^v)$.
In our situation, a group is formed by taking an unsuccessful processor together with its helpers, and the search for an operational $D_i(x)$ is considered as *Deactivate*. ¿From the previous discussion we know, that by applying *Deactivate* we can reduce the number of active groups as required, hence satisfying the assumptions from [3].
Finally, we show how to remove the assumption that all memory cells accessed by the simulated machine at each step are different. First notice, that in fact we need this assumption to hold only within in each group of processors G (as we want the attempts of memory access to be independent). To satisfy this assumption, it is sufficient to choose one representative of each memory address to access within each group. To this end, we use the following two procedures :

Definition: The *semisorting*problem is: given an array $A[1 \dots k]$ of integers, compute an array $B[1 \dots O(k)]$ containing all elements of A padded with 0s, such that all (say d_x) elements of A with the same value x are contained in consecutive $O(d_x)$ positions of B.

Definition: The *All-Nearest-Ones* (ANO) problem is: given an array $A[1 \dots k]$ of 0s and 1s, find an array $B[1 \dots k]$ such that for any $i = 1 \dots k$ $B[i]$ contains a maximal index $j \leq i$ such that $A[j] = 1$ (if such an index exists).

Lemma 11. [15] *The semisorting problem can be solved with $k/\log^* k$ processors in $O(\log^* k)$ time with probability $1 - 2^{-k^\epsilon}$ for some $\epsilon > 0$.*

Lemma 12. [6] *The ANO problem can be solved with $k/\log^* k$ processors in $O(\log^* k)$ time.*

The representatives selection can be now performed as follows. Let $(x) = x_1 \dots x_k$ be the sequence of memory cells to access. First we semisort (x) obtaining (x'). In this way all address' with the same value are grouped together. Then we solve an ANO on (x'), treating the non-zero elements of (x') as 1s and 0s as 0s. Each non-zero element of (x') can now check if its left neighbor is of the same value. If it is not the case, the element is selected to the set of representatives, otherwise it is not. It is easy to see that exactly one representative for each value is selected.
This proves the following

Theorem 13. *There exists an optimal randomized simulation of operational n-processor CRCW PRAM on $n/\log^* n$-processor CRCW PRAM with static memory faults with step overhead $O(\log^* n)$ w.h.p..*

5 Conclusions

In this paper we have presented two algorithms for simulation of operational PRAMs on faulty ones. The first one performes the preprocessing of a faulty Exclusive-Read Exclusive-Write (EREW) PRAM with $n/\log n$ processors and $O(n)$ memory cells in $O(\log n)$ time. The second one gives a simulation of n-processor CRCW on $n/\log^* n$-processor CRCW with $O(\log^* n)$ step overhead. After some minor changes in preprocessing all our algorithms become tolerant also to deterministic static processor faults using the preprocessing of [19], with the same step overhead.
The techniques used for the first simulation have found additional applications. Using the expander graphs as in our algorithms for EREW preprocessing it is possible to achieve a simulation of n-processor EREW on $n \log n$ processor EREW with faults in time $O(\log \log n)$ [10] and of n-processor CRCW on some variant of $n \log^{1+\epsilon} n$ processor faulty CRCW in constant time (excluding the preprocessing). This will appear in the full version of this paper. The preprocessing procedures may also have some applications for more restricted models of computation, as in fact we use EREW only to simulate a network with a bounded degree.
Designing fast and optimal deterministic simulations for faulty EREW and CRCW PRAM remains the main open problem.

6 Acknowledgments

The authors want to thank Nabil Kahale and Yuan Ma for pointing us to the paper [1] and Krzysztof Diks for helpful discussions.

References

1. N. Alon, and F.R.K. Chung, Explicit construction of linear sized tolerant networks, *Discrete Mathematics* (1988), vol. 72, pp. 15–19.
2. Y. Afek, D.S. Greenberg, M. Merrit, and G.Taubenfeld, Computing with Faulty Shared Memory, in *Proceedings 11th Ann. Symposium on Principles of Distributed Computing* (1992), pp. 47–58.
3. H. Bast, T. Hagerup, Fast and reliable parallel hashing, in *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures* (1991), pp. 50–61.
4. J.F. Buss, P.C. Kanellakis, P.L. Ragde, A.A. Shvartsman, Parallel algorithms with processor failures and delays. Journal of Algorithms 20 (1996), no.1, p. 45-86.
5. P. Berenbrink, F. Meyer auf der Heide, V. Stemann, Fault-Tolerant Shared Memory Simulations, in *Proceedings of 13th Annual Symposium on Theoretical Aspects of Computer Science* (1996), Springer LNCS 104, pp 181–192.
6. O. Berkman and U. Vishkin, Recursive star-tree parallel data structure, *SIAM J. Comput.* 22 (1993), 221–242.
7. B.S. Chlebus, A. Gambin, and P. Indyk, PRAM computations resilient to memory faults, in *Proceedings of the 2nd Annual European Symposium on Algorithms,* ed. J. van Leeuwen, Utrecht, The Netherlands (1994), Springer LNCS 855, pp. 401–412.
8. B.S. Chlebus, A. Gambin, and P. Indyk, Shared-Memory Simulations on a Faulty DMM, in *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming* (1996).
9. B.S. Chlebus, L. Gąsieniec, and A. Pelc, Fast Deterministic Simulation of Computations on Faulty Parallel Machines, in *Proceedings of the 3rd Annual European Symposium on Algorithms,* ed. J. van Leeuwen, Corfu, Greece (1995), Springer LNCS 979, pp. 89–101.
10. K. Diks, Personal Communication.
11. K. Diks and A. Pelc, Reliable computations on faulty EREW PRAM, *Theoretical Computer Science*, to appear.
12. W. Feller, *An Introduction to Probability Theory and Its Applications*, Wiley, New York (1950).
13. O. Gabber, and Z. Galil, Explicit constructions of linear-sized superconcentrators, *J. Comput. System Sci.*, vol. 22 (1981), pp. 407–420.
14. A. Gibbons, and W. Rytter, *"Efficient Parallel Algorithms"*, Cambridge University Press (1988).
15. T. Hagerup, The Log-star Revolution, Proc. *9th Annual Symposium on Theoretical Aspects of Computer Science* (1992), Springer LNCS, Vol. 577, pp. 259–278.
16. T. Hagerup, and Ch. Rüb, A Guided Tour of Chernoff Bounds, *Inf. Proc. Letters* 33 (1989/90), pp. 305–308.
17. P. Indyk, On Word-Level Parallelism in Fault-Tolerant Computing, in *Proceedings of 13th Annual Symposium on Theoretical Aspects of Computer Science* (1996), Springer LNCS 1046, pp. 193–204.
18. J. JáJá, *"An Introduction to Parallel Algorithms"*, Addison Wesley, Reading, MA (1992).
19. P.C. Kanellakis, D. Michailidis, Shvartsman, A.A., Controlling memory access concurrency in efficient fault-tolerant parallel algorithms, Nordic Journal of Computing 2 (1995), no.2, pp. 146-180.
20. P.C. Kanellakis, and A.A. Shvartsman, Efficient parallel algorithms can be made robust, *Distributed Computing*, 5 (1992), pp. 201–217.
21. R.M. Karp, and V. Ramachandran, Parallel algorithms for shared-memory machines, in *"Handbook of Theoretical Computer Science,"* ed. J. van Leeuwen, Elsevier (1990), vol. A, pp. 869–941.
22. Z. M. Kedem, K. V. Palem, A. Raghunathan, and P. Spirakis, Combining tentative and definite executions for very fast dependable parallel computing, in *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing* (1991), pp. 381–390.
23. Z. M. Kedem, K. V. Palem, and P. Spirakis, Efficient robust parallel computations, in *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing,* (1990), pp. 138–148.
24. A. Lubotzky, R. Philips, and P. Sarnak, Ramanujan graphs, *Combinatorica* (1988), vol. 8, no. 3, pp. 261–77.
25. Ch. Martel, R. Subramanian, and A. Park, In *Proceedings of 31st Annual Symposium on Foundations of Computer Science* (1990), pp. 590–599.

26. Mc Diarmid, On the Method of Bounded Differences, in J. Siemon, ed., *"Surveys in Combinatorics"*, Cambridge University Press (1989), London Math. Soc. Lecture Note Series 141, pp. 148–188.
27. F. Meyer auf der Heide, Hashing strategies for simulating shared memory on distributed memory machines, in *Proceedings of the 1st Heinz Nixdorf Symposium "Parallel Architectures and their Efficient Use,"* ed. F. Meyer auf der Heide, B. Monien, A.L. Rosenberg, Paderborn, Germany (1992), Springer LNCS 678, pp. 20–29.
28. M.O. Rabin, Efficient dispersal of information for security, load balancing, and fault tolerance, *Journal of ACM*, 36 (1989), pp. 335–348.
29. A. Siegel, On Universal Classes of Fast High Performance Hash Functions, Their Time-Space Tradeoff, and Their Applications, In *Proceedings of 30th Annual Symposium on Foundations of Computer Science* (1989), pp. 20–25.
30. J. Schmidt, A. Siegel and A. Srinivasan, Chernoff-Hoeffding Bounds for Applications with Limited Independence, in *Proceedings of the Symposium on Discrete Algorithms* (1993), pp. 331–340.
31. M. Sipser, and D. A. Spielman, Expander Codes, in *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994), pp. 566–576.