Theoretical Computer Science 583 (2015) 86–97

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Exploiting non-constant safe memory in resilient algorithms and data structures

Lorenzo De Stefani, Francesco Silvestri *

Dipartimento di Ingegneria dell'Informazione, Università di Padova, Via Gradenigo 6/B, I-35131 Padova, Italy

ARTICLE INFO

Article history:
Received 22 May 2013
Received in revised form 29 October 2014
Accepted 2 April 2015
Available online 8 April 2015
Communicated by G.F. Italiano

Keywords:
Resilient algorithm
Resilient data structure
Memory errors
Sorting
Priority queue
Tradeoffs
Fault tolerance

ABSTRACT

We extend the Faulty RAM model by Finocchi and Italiano (2008) by adding a safe memory of arbitrary size S, and we then derive tradeoffs between the performance of resilient algorithmic techniques and the size of the safe memory. Let δ and α denote, respectively, the maximum amount of faults which can happen during the execution of an algorithm and the actual number of occurred faults, with $\alpha \leq \delta$. We propose a resilient algorithm for sorting n entries which requires $O(n \log n + \alpha(\delta/S + \log S))$ time and uses $\Theta(S)$ safe memory words. Our algorithm outperforms previous resilient sorting algorithms which do not exploit the available safe memory and require $O(n \log n + \alpha\delta)$ time. Finally, we exploit our sorting algorithm for deriving a resilient priority queue. Our implementation uses $\Theta(S)$ safe memory words and $\Theta(n)$ faulty memory words for storing n keys, and requires $O(\log n + \delta/S)$ amortized time for each insert and deletemin operation. Our resilient priority queue improves the $O(\log n + \delta)$ amortized time required by the state of the art.

1. Introduction

Memories of modern computational platforms are not completely reliable since a variety of causes, including cosmic radiations and alpha particles [1], may lead to a transient failure of a memory unit and to the loss or corruption of its content. Memory errors are usually silent and hence an application may successfully terminate even if the final output is irreversibly corrupted. This fact has been recognized in many systems, like in Sun Microsystems servers at major customer sites [1] and in Google's server fleets [2]. Eventually, a few works have also shown that memory faults can cause serious security vulnerabilities (see, e.g., [3]).

As hardware solutions, like Error Correcting Codes (ECC), are costly and reduce space and time performance, a number of algorithms and data structures have been proposed that provide (almost) correct solutions even when silent memory errors occur. Algorithmic approaches for dealing with unreliable information have been widely targeted in literature under different settings, and we refer to [4] for a survey. In particular, a number of algorithms and data structures, which are called *resilient*, have been designed in the *Faulty RAM* (*FRAM*) model [5]. In this model, an adaptive adversary can corrupt up to δ memory cells of a large unreliable memory at any time (even simultaneously) during the execution of an algorithm. Resilient algorithmic techniques have been designed for many problems, including sorting [6], selection [7], dynamic pro-

* Corresponding author. Tel.: +39 049 8277954.
E-mail addresses: destefan@dei.unipd.it (L. De Stefani), silvest1@dei.unipd.it (F. Silvestri).

http://dx.doi.org/10.1016/j.tcs.2015.04.003
0304-3975/

gramming [8], dictionaries [9], priority queues [10], matrix multiplication and FFT [11], K-d and suffix trees [12,13]. Resilient algorithms have also been experimentally evaluated [14,11,15,16].

1.1. Our results

Previous results in the FRAM model assume the existence of a safe memory of constant size which cannot be corrupted by the adversary and which is used for storing crucial data such as code and instruction counters. In this paper we follow up the preliminary investigation in [8] studying to which extent the size of the safe memory can affect the performance of resilient algorithms and data structures. We enrich the FRAM model with a safe memory of arbitrary size S and then give evidence that an increased safe memory can be exploited to notably improve the performance of resilient algorithms. In addition to its theoretical interest, the adoption of such a model is supported by recent research on hybrid systems that integrate algorithmic resiliency with the (limited) amount of memory protected by hardware ECC [17]. In this setting, S would denote the memory that is protected by the hardware.

Let δ and α denote respectively the maximum amount of faults which can happen during the execution of an algorithm and the actual number of occurred faults, with $\alpha \leq \delta$. In Section 2, we show that n entries can be resiliently sorted in $O(n \log n + \alpha(\delta/S + \log S))$ time when a safe memory of size $\Theta(S)$ is available in the FRAM. As a consequence, our algorithm runs in optimal $\Theta(n \log n)$ time for $\delta = O\left(\sqrt{nS \log n}\right)$ and $S \leq n/\log n$. When $S = \omega(1)$, our algorithm outperforms previous resilient sorting algorithms, which do not exploit non-constant safe memory and require $O(n \log n + \alpha\delta)$ time [6,7]. Finally, we use the proposed resilient sorting algorithm for deriving a resilient priority queue in Section 3. Our implementation uses $\Theta(S)$ safe memory words and $\Theta(n)$ faulty memory words for storing n keys, and requires $O(\log n + \delta/S)$ amortized time for each insert and deletemin operation. This result improves the state of art for which $O(\log n + \delta)$ amortized time is required for each operation [10].

1.2. Preliminaries

As already mentioned, we use the FRAM model with a safe memory. Specifically, the adopted model features two memories: the *faulty memory* whose size is potentially unbounded, and the *safe memory* of size S. For the sake of simplicity, we allow algorithms to exceed the amount of safe memory by a multiplicative constant factor. The adversary can read the content of the faulty memory and corrupt at any time memory words stored in any position of the faulty memory for up to a total δ times. Note that faults can occur simultaneously and the adversary is allowed to corrupt a value which was already previously altered. The safe memory can be read but not corrupted by the adversary. A similar model was adopted in [8], however in this paper the adversary was not allowed to read the safe memory. We denote with $\alpha \leq \delta$ the actual number of faults injected by the adversary during the execution of the algorithm. Since the performance of our algorithms do not increase as soon as $S > \delta$, we assume through the paper that $S \leq \delta$; this assumption can be easily removed by replacing S with $\min\{S, \delta\}$ in our algorithms.

A variable is *reliably written* if it is replicated $2\delta + 1$ times in the faulty memory and its actual value is determined by majority: clearly, a reliably written variable cannot be corrupted. We say that a value is *faithful* if it has never been corrupted and that a sequence is *faithfully ordered* if all the faithful values in it are correctly ordered. Finally, we assume all faithful input values to be distinct, each value to require a memory word, and that each sequence or buffer to be stored in adjacent memory words.

2. Resilient sorting algorithm

In the resilient sorting problem we are given a set of n keys and the goal is to correctly order all the faithful input keys (corrupted keys can be arbitrarily positioned). We propose *S-Sort*, a resilient sorting algorithm which runs in $O(n \log n + \alpha(\delta/S + \log S))$ time by exploiting $\Theta(S)$ safe memory words. Our approach builds on the resilient sorting algorithm in [6], however major changes are required to fully exploit the safe memory. In particular, the proposed algorithm forces the adversary to inject $\Theta(S)$ faults in order to invalidate part of the computation and to increase the running time by an additive $O(\delta + S \log S)$ term. In contrast, $O(1)$ faults suffice to increase by an additive $O(\delta)$ term the time of previous algorithms [5–7], even when $\omega(1)$ safe memory is available. Our algorithm runs in optimal $\Theta(n \log n)$ time for $\delta = O\left(\sqrt{Sn \log n}\right)$ and $S \leq n/\log n$: this represents a $\Theta\left(\sqrt{S}\right)$ improvement with respect to the state of the art [6], where optimality is reached for $\delta = O\left(\sqrt{n \log n}\right)$.

S-Sort is based on mergesort and uses the resilient algorithm *S-Merge* for merging. The S-Merge algorithm requires $O(n + \alpha(\delta/S + \log S))$ time for merging two faithfully ordered sequences of length n each with $\Theta(S)$ safe memory. S-Merge is structured as follows. An incomplete merge of the two input sequences is initially computed with *S-PurifyingMerge*: this method returns a faithfully ordered sequence Z of length at least $2(n - \alpha)$ that contains a partial merge of the input sequences, and a sequence F with the at most 2α remaining keys that the algorithm has failed to insert into Z. Finally, keys in F are inserted into Z using the *S-BucketSort* algorithm, obtaining the final faithfully ordered sequence of all input values. Procedures S-PurifyingMerge and S-BucketSort are respectively proposed in Sections 2.1 and 2.2, while Section 2.3 describes the resilient algorithms S-Merge and S-Sort.