Towards a Universal RAM Machine Resistant to Isolated Bursts of Faults

Ilir Çapuni Ervin Dervishaj
Computer Engineering Department
Epoka University
{icapuni, edervishaj10}@epoka.edu.al

***Abstract*—The most natural question of reliable computation, in every computation model and noise model, is whether given a certain level of noise, a machine of that model exists that can perform arbitrarily complex computations under noise of that level. This question has positive answers for circuits, cellular automata, and recently for Turing machines [3], [4].**

Here, we raise the question of the existence of a random access machine that—with some moderate slowdown — can simulate any other random access machine even if the simulator is subjected to constant size bursts of faults separated by a certain minimum number of steps from each other.

We will analyze and spell out the problems and difficulties that need to be addressed in such construction.

***Index Terms*—Random access machines, faults, reliability.**

I. Introduction

The problem of constructing fault-proof machines from components that can fail was first considered by von Neumann in [12], who addressed the problem in the Boolean circuits model. New advances along this path were made in [9], [10]. The question has been considered in uniform models of computation as well. A simple rule for two-dimensional cellular automata that keeps one bit forever even though each cell can fail with some small probability was given in [11]. A 3-dimensional reliably computing cellular automaton using Toom's rule was constructed in [7]. Alas, all simple one-dimensional cellular automata appear to be "ergodic" (forgetting everything about their initial configuration in time independent of the size). The first, complex, nonergodic cellular automaton was constructed in [5], and improved upon in [6]. Surprisingly, even non-ergodic Turing machines exists [3], [4].

Computing reliably with RAM machines is considered in a myriad of papers with various assumptions on the noise. A similar problem but where only the memory is subject to a limited amount of noise is considered in [8]. The cited work initiated an entire line of research approaching the problem from data structures' and algorithmic perspective. A recent result in this line is [2].

Our approach differs from the papers above for that faults can perturb the central unit and for that that the noise comes in bursts that are spaced out of each other. The reason for these assumptions on noise is that — similarly as in [4], [6]— we hope to use this construction as a building block for a hierarchically organized RAM that can withstand faults that occur independently of each other with some small probability.

The first author's dedication to this work was made possible by a research grant RD 01-2013 of Zanus ltd, Ulqin Montenegro

A. *The random access machine model*

We view the random-access machine as an extension of the Turing machine in that that the tape consists of cells able to store an arbitrary integer, and that the head can jump at any cell of the memory $\boldsymbol{M}$ specified by its address.

Register PC is a special register called the ***program counter***. Let $\Gamma = \{PC, R_0, \dots, R_k\}$ be the set of registers of the control unit. Register R_0 will be called the ***accumulator***.

The program is a sequence of instructions from the Table I.

Definition I.1. A ***Random Access Machine*** is a pair (k, Π) where $k > 0$ is the number of registers, and Π is the program (i.e., a finite sequence of instructions). ⌟

The operation of a RAM machine is described below.

1) Initially, the input x is loaded in the first m cells of the memory $\boldsymbol{M}$, where m is the length of x. The registers $R_1, \dots, R_k$, PC and the other memory cells $M[m]$, $M[m+1]$, …are initialized to the value 0.
2) At each step of the execution, the random access machine executes the program line pointed to by the program counter PC. After executing the instruction, the value of PC is incremented by 1, unless the instruction is `jump`, `jzero`, or `jpos` If the instruction is `jzero` or `jpos` and the condition is not satisfied, PC is also incremented by 1.

Symbols are encoded by integers, and we assume that the blank symbol is encoded by 0.

Example I.2. If $\Sigma = \{\#, a, b\}$, then we might assign $\# = 0, a = 1, b = 2$. Then the input *abba* would be represented in the first four cells of the memory as $1, 2, 2, 1$. ⌟

A configuration of a Random Access Machine (k, Π) is a $k + 2$-tuple

$$(PC, R_0, \dots, R_k, \boldsymbol{M}),$$

where $\boldsymbol{M} \in \mathbb{Z}^{\mathbb{Z}}$ is the ***memory configuration***.

TABLE I
INSTRUCTIONS OF A RAM MACHINE

Instruction	Semantics	Description
`load` n	$R_0 \leftarrow n$	Put the value n into R_0
`load` R_k	$R_0 \leftarrow R_k$	Put the value of R_k into R_0
`store` R_k	$R_k \leftarrow R_0$	Put the value of R_0 into R_k
`read` R_k	$R_0 \leftarrow M[R_k]$	Copy the value at memory location R_k into R_0
`write` R_k	$M[R_k] \leftarrow R_0$	Write the value of R_0 at a memory location R_k
`add` n	$R_0 \leftarrow R_0 + n$	Add the value n to R_0
`add` R_k	$R_0 \leftarrow R_0 + R_k$	Add the value R_k to R_0
`mult` R_k	$R_0 \leftarrow R_0 * R_k$	Multiply the value in R_0 with R_k
`jump` n	$PC \leftarrow n$	Set the program counter to n
`jzero` n	if $R_0 = 0$ then $PC \leftarrow n$	If R_0 is zero, then set the PC to n
`jpos` n	if $R_0 > 0$ then $PC \leftarrow n$	If R_0 is positive, then set the PC to n

The work of the machine can be described as a sequence of configurations $C_0, C_1, C_2, \ldots$, where C_t is the configuration at time t.

The program Π tells us how to compute the next configuration from the present one.

Definition I.3 (Fault)**.** We say that a ***fault*** occurs at time t if configuration C_{t+1} is obtained from C_t by violating the transition function specified by the program. ⌟

B. *What faults can do*

By definition, if a fault occurred at time t, then, at time $t+1$ a random cell and the content of the registers in the control unit are arbitrary. This means that unlike the effects of bursts in a Turing machine — where bursts produce islands of cells of diameter β close to the head, in RAM model, these islands are scattered in a memory in a random way. This is a major difficulty in construction of a fault-tolerant RAM since this means that the information stored in the memory decays.

For example, consider a computation whose space is s. Suppose further that for time proportional to s^2, the computation was carried out in only the first half of the computation space. Then, within this time, the second half of the computation space may be completely ruined by faults that occurred during this time period.

II. The Desired Result

In this section we will spell out a desired result. Before we need the following definition.

Definition II.1 (Codes)**.** Let Σ_1, Σ_2 be two subsets of $\mathbb{Z}$. A ***block code*** is given by a positive integer Q— called the ***block size*** — and a pair of functions

$$\psi_* : \Sigma_2 \to \Sigma_1^Q, \quad \psi^* : \Sigma_1^Q \to \Sigma_2$$

with the property $\psi^*(\psi_*(x)) = x$. ⌟

Now we can spell out the desired statement.

Let RAM machine M_2 start from an input x with a starting configuration ξ_0 and suppose that it halts within T steps, writing the result in the memory location with address 0. Let S be the amount of space that M_2 used during its computation.
Then, the following can be constructed.

1) RAM machine M_1 with k registers that does not have a halting state.
2) Constant Q and a block code (ψ_*, ψ^*) of block size Q.
3) Constants T' depending on T and S, and constant $k' = O(k)$.

Suppose M_1 starts from the initial configuration $\xi'_0 = \xi'_0(x)$, and it operates under noise that consists of isolated bursts of size at most β.
Then, at any time t, $t > T'$, the result of M_2 can be decoded from the memory block $k', \ldots, k' + Q - 1$.

III. Solutions and difficulties

A. *Solving the problem by simulating the fault-tolerant Turing machine*

It is natural to ask if we can achieve the needed fault tolerance by just simulating the fault-tolerant Turing machine from [3] or [4]. The answer is alas negative. The fault-tolerant Turing machines constructed in [3] and [4] have an underlying assumption that the information on the tape does not decay. However, here faults can ruin portions of the memory far from the location of the head of the Turing machine.

B. *Redundancy and memory updates*

Let us consider a different solution. We will encode the control unit of M_2 using some error-correcting code into a fixed constant size portion of the memory of M_1. Using the same code we will encode the memory of M_2 onto the memory of M_1.

Then, similar to the simulations in [3], [4], we will simulate one step of M_2 by many steps of M_1.

Since information "decays" in the memory, we need to find a way to constantly update and check all the parts of the computation space in the memory. Since RAM is a sequential machine, we need to space out the bursts enough such that the machine can "catch up" with the decay of the information in the memory. However, the minimal distance between two consecutive bursts will now depend on the amount of space that M_2 needs during the computation.

C. A hierarchical organization with digests

In the previous section we have seen that in order to preserve the information from decaying in the memory, we need to constantly refresh it by decoding and encoding with an error-correcting code. Doing this for the entire space s of computation may be time consuming.

Using the approach of [1] we may restrict doing this for a part of the memory of size $O(\log s)$ which will be considered more reliable.

Giving a fully fledged construction based on this idea and proving the desired result spelled out in Section II will be a subject of our forthcoming research.

References

[1] Blum M., Evans W., Gemmel P., Kannan S., and Naor M.: Checking the Correctness of Memories. In: *Algorithmica*, 1995, pp. 90-99.

[2] Christiano P, Demaine D. E., and Kishore Sh.: Lossless Fault-Tolerant Data Structures with Additive Overhead. In *Proceedings of the 12th international conference on Algorithms and data structures* (WADS'11), Frank Dehne, John Iacono, and Jörg-Rüdiger Sack (Eds.). Springer-Verlag, Berlin, Heidelberg, 243-254.

[3] Çapuni I., Gács, P. : A Turing machine resisting isolated bursts of faults. In: Chicago Journal of Theoretical Computer Science, vol. 2013.

[4] Ilir Capuni. 2013. A Fault-Tolerant Turing Machine. Ph.D. Dissertation. Boston University, Boston, MA, USA. Advisor(s) Peter Gacs. AAI3536949.

[5] Gács, P.: Reliable computation with cellular automata. *Journal of Computer System Science*, **32**/1, (1986) 15-78.

[6] Gács, P.: Reliable cellular automata with self-organization. *Journal of Statistical Physics* **103**/1-2 (2001), 45-267.

[7] Gács, P., Reif, J.A: Simple three-dimensional real-time reliable cellular array. *J. Comput. Syst. Sci.* **36**/2 (1988) 125-147.

[8] Finocchi, I., Grandoni, F., Italiano, F., G.: Designing Reliable Algorithms in Unreliable Memories. *Computer Science Review* **1**/2 (2007) 77-87.

[9] Pippenger,N.: On networks of noisy gates. In: *Proc. of the 26-th IEEE FOCS Symposium* (1985) 30-38.

[10] Spielman, D.: Highly fault-tolerant parallel computation. In: *Proc. of the 37th IEEE FOCS Symposium* (1996) 154-163.

[11] Toom, A.: Stable and attractive trajectories in multicomponent systems. In *Multicomponent Systems* (R.L. Dobrushin, ed.), Advances in Probability **6**, Dekker, New York, (1980) 549-575.

[12] von Neumann, J.: Probabilistic logics and the synthesis of reliable organisms from unreliable components. In: *Automata Studies* (C. Shannon and McCarthy eds.), Princeton University Press, Princeton, NJ. (1956)