Increasing Fault-Tolerance in Cellular Automata-Based Systems

Luděk Žaloudek and Lukáš Sekanina

Faculty of Information Technology, Brno University of Technology,
Božetěchova 2, 612 66 Brno, Czech Republic
{izaloude, sekanina}@fit.vutbr.cz

Abstract. In the light of emergence of cellular computing, new cellular computing systems based on yet-unknown methods of fabrication need to address the problem of fault tolerance in a way which is not tightly connected to used technology. This may not be possible with existing elaborate fault-tolerant cellular systems so we strive to reach simple solutions. This paper presents a possible solution for increasing fault-tolerance in cellular automata in a form of static module redundancy. Further, a set of experiments evaluating this solution is described, using triple and quintuple module redundancy in the automata with the presence of defects. The results show that the concept works for low intensity of defects for most of our selected benchmarks, however the ability to cope with errors can not be intuitively deduced as indicated on the example of the majority problem.

Keywords: cellular automata, fault tolerance, static module redundance, TMR, cellular computing, rule 30, Game of Life, Byl's Loop

1 Introduction

With today's unending need for faster and ubiquitous computation, the von Neumann architecture is being pushed to its limits. As new problems emerge [2], more and more computation is performed in parallel and even new computing paradigms emerge. One of these new paradigms is what Sipper calls cellular computing [14].

Cellular computing is based on the model of cellular automaton (CA) and its three key features: simplicity, locality of interactions and massive (or vast) parallelism. The CA model is flexible and capable of performing extensive range of computational tasks (overview of most notable may be found in [14]). Also, the CA model keeps rising in importance in several other areas including simulation of biological, chemical or social phenomena.

Since cellular computing is based on a model rather than a physical piece of hardware, it is not limited in the technology, by which it could be fabricated or synthesized. As a relatively new paradigm and possible future solution to (some of) our computational needs, it could be implemented not only in silicon (where there are existing examples of cellular-based hardware [5, 11]).

However novel approaches to digital hardware fabrication like chemical synthesis or inkjet printing are not yet refined and possibly prone to defects and transient errors more than the most common CMOS technology. In order to address this problem, methods of fault-tolerance need to be introduced. Existing cellular systems [5, 11] possess fault-tolerance mechanisms which arguably significantly complicate the underlying hardware. Of the simpler methods of fault-tolerance the most prominent are module redundancy and error correcting codes. The goal of this paper is to demonstrate the feasibility of module redundancy on several distinct examples of CA tasks.

Section 2 introduces the cellular automaton model together with its notable modifications, properties and problems. One of the problems, specifically fault-tolerance in CA, is discussed in the remainder of Section 2 and current methods of fault-tolerance in CA are outlined. In Section 3 our solution is proposed. Section 4 describes sample CA problems which are then used in an experimental setup. Results of aforementioned experiments are provided in Section 5. Section 6 concludes the paper.

2 Cellular Automata

2.1 General Description of Cellular Automata

A cellular automaton is a d–dimensional grid of cells, where each cell is a finite automaton. The cells operate according to their *local transition functions*. In the most common scenario, the cells work synchronously - a new state of every cell is calculated from its previous state and the previous states of the cell's 'neighbors' at each time step. If all automata of the grid are identical (i.e., the automata operate according to the same local transition function - *rule*) the automaton is called *uniform*, otherwise it is *non-uniform*. By *configuration* of the cellular automaton we mean the states of all the cells at a given moment. The global behavior is captured in the *global transition function*, which defines a transformation from one configuration to the next configuration of the cellular automaton, in case of synchronous automaton the transition between steps. The sequence of configurations, determined by the global transition function, represents the *computation* of the cellular automaton.

Theoretically, the CA model assumes an infinite number of cells. However, in the case of practical applications the number of cells is finite. In that case, it is necessary to define the *boundary conditions*, i.e. the setting of the boundary cells' neighborhood. One of the states is also usually used as *quiescent* or inactive state. A convention is that when a quiescent cell has an entirely quiescent neighborhood, it will remain quiescent at the next time step.

The simplest one-dimensional uniform cellular automaton, with only two states and nearest neighbors neighborhood $N = \{-1, 0, 1\}$ (only left and right neighbor cells together with the cell itself are relevant for the local transition function[1]), can exhibit very complex behavior [18]. Each such CA is uniquely

[1] We also denote this parameter as neighborhood radius r and in the nearest neighborhood case we would write that $r = 1$.

defined by a mapping $0, 1^N \rightarrow \{0, 1\}$. Hence there are 28 such cellular automata, each of which is uniquely specified by the (transition) rule i ($0 \leq i < 256$). The transition functions of such automata then can be called as *rule 0*, *rule 1*, *rule 2* etc.

With two-dimensional cellular automata, the neighborhood is usually comprised of five or nine cells (see (Fig. 1). The central cell is normally included into its own neighborhood. In the 5-neighbor (von Neumann) model, the individual transition rules would adopt the form $CTRBL \rightarrow C'$ where $CTRBL$ specifies the states of the Center, Top, Right, Bottom, and Left positions of the neighborhood's present state and C' represents the next state of the center cell.

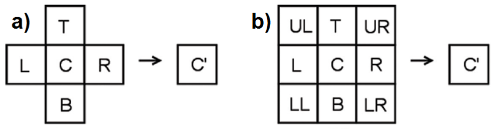

Fig. 1. a) 5-cell von Neumann's neighborhood, **b)** 9-cell Moore's neighborhood

The applications of cellular automata include many scientific areas, especially the modeling of complex (biological, chemical, computational, etc.) phenomena. Because the number of cells is typically very large, the cellular automaton can model a massively parallel computational system where some useful computation can *emerge* only on the base of local interactions. The properties of cellular automata have been investigated by means of analytic as well as experimental methods [8, 15–18].

In order to obtain specific behaviors, the concept of cellular automata has been modified in several ways. Examples include *non–uniform* CA [15], *non–local* CA [10], *asynchronous* CA [3] or CA *with global control* [13].

2.2 Notable Inherent Properties and Problems

The cellular automaton model also exhibits several notable inherent properties that set it apart from conventional computational models: *massive parallelism*, *locality* of cellular interactions and *simplicity* of basic components (cells) [15, 14]. This was mentioned in Section 1, however the terms need to be clarified. Note these properties are especially important for our case where every cell would be physically implemented (no simulation considered).

Massive parallelism is something rarely seen even in today's supercomputers. When talking about it in CA, we mean vast numbers of cells on much larger scale than the numbers of processing elements usually found in conventional parallel systems. The cell figures in CA are often expressed by exponential notation 10^X.

Locality of interactions means total absence of global control. Transitions between states are based on nearest neighborhood. Best known examples of such neighborhood are von Neumann's and Moore's neighborhood for 2D CA.

Simplicity refers to cells as basic computational elements. Instead of depending on complex components similar to modern processors (e.g. Pentium), a CA cell should be easily constructed as a simple electronic circuit or using chemical synthesis.

These properties put together are key to *emergent behavior* mentioned in Subsection 2.1. However, these features also require certain prerequisites, which we usually assume when modeling cellular automata in software: perfect clock synchronization (for synchronous CA) and error-free implementation platform. The two prerequisites turn into problems when we attempt to implement CA physically.

In particular, perfect clock synchronization may be unachievable when considering truly massive cellular arrays due to different lengths of clock wiring in different parts of the cellular array and the spatial requirement for such wiring. Moreover, since cellular automata are one of possible future solutions to problems of conventional computational architectures [14], we cannot assume that the clocking problem will not encounter further complications when migrating cellular architectures to different materials and/or fabrication methods (with respect to current CMOS technology). One of possible solutions and probably the most feasible is the employment of asynchronous CA mentioned in Subsection 2.1. With recent achievements in the field of delay-insensitive circuits, techniques employing asynchronous CA were demonstrated [12]. It is arguable if such functionality may be duplicated in different technology such as chemical synthesis. In addition, severe modification of local transition function (rules) with respect to synchronous version of implemented applications is required when applying the asynchronous CA paradigm.

The second problem is virtual impossibility of attaining error-free implementation platform for cellular-based computing machines. Not only do we encounter a variety of transient errors (caused by cross talk, noise etc.) and transient faults (e.g. temporary hardware element failures due to ambient radiation), with decreasing component sizes and the dawn of nanotechnology in mind, we must also deal with permanent defects originating in fabrication or arising during the lifetime of the hardware. Such defects may not be detected (or even detectable) after the fabrication and the CA model should be prepared to deal with them independently of hardware.

Unlike solving problems with synchronization, attaining fault-tolerance is the subject of this paper, which is discussed in more detail in Subsection 2.3.

2.3 Fault-Tolerance in Cellular Automata

In recent years several methods of fault-tolerance in CA have been implemented or proposed. Those which are employed in existing cellular-based hardware [5, 11] added new layers of functionality and also complexity. Cells are formed into supercells, routing mechanisms circumvent defective elements and positional information and whole sets of functionality are passed to each cell/element. Although such systems possess fault-tolerant functionality, the overhead cost is extremely high in terms of hardware complexity.

Other, much simpler mechanisms were introduced to the CA model, which were inspired by more conventional digital circuit solutions [6, 12]. Most prominent of such solutions are module redundancy and error detecting and correcting codes.

When utilizing ECC in CA, the fault-tolerance deals with errors that occur in the cell memory, i.e. its state information or its transition function look-up table, if that is implemented in such way. Example of cellular automaton with error detection and correction is [12].

3 Proposed Approach

Generally, static module redundancy which is our main concern in this paper is based on backing up the subsystem's function by multiplying the modules which compute it. The results of each module are then compared in a voting element which produces the final result by majority voting [9]. Best practice is to use odd number of modules to insure no ties occur. The most commonly used form of static modular redundancy in digital circuits is TMR (triple). Other possible approaches to module redundancy include dynamic redundancy and/or reconfiguration and combinations of previously mentioned. Of course, such methods increase the complexity of the implementation due to additional multiplexing, voting and/or reconfiguring elements. This paper is further limited to static NMR so we do not complicate the fabrication process of cellular systems which should be cheap and easily produced in vast numbers.

An example of a general analysis of reliability of TMR in quantum-dot CA may be found in [6]. However, we believe that distinct types of CA tasks may behave differently depending on their complexity and number of active cells in presence of errors. The approach to such discovery is presented in following paragraphs.

Since we work with CA as a possible model for future computing systems, we cannot predict which technologies or materials will be used in actual cellular system fabrication. Because of that, transient errors are not considered in our analysis. The reason is apparent possibility of difference in transient error sensitivity in various technologies which could be used in cellular system's fabrication (e.g. chemical synthesis, DNA-based scaffolding, ink-printed circuits etc.).

We also limit our analysis to small standalone defects. Such defects could occur during production or during the lifetime of the application and may not be easily detectable like common large-scale burst defects which destroy hundreds of elements in tightly localized patterns. Since we assume such environment and implementation platform, we can assume that usage of TMR or 5-module redundancy (5MR) should be sufficient.

The design of each cell is straightforward and was roughly outlined at the beginning of this section. Each cell contains three (TMR) or five (5MR) modules, which have common inputs from the cell's neighborhood. Each module computes the local transition function F of the CA and all module's results are provided to the majority voting element. The voting element then passes the dominating result as the next state of the cell. In theory, TMR can mask one error and 5MR can mask two errors without system failure. Graphical representation of one CA cell with TMR is depicted in Fig. 2.

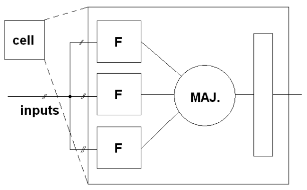

Fig. 2. Inside a CA cell with TMR. Triple modules take *inputs* from neighborhood and compute the local transition function *F*. Results are then voted on by *MAJ.* element which passes the prevailing result to register as the cell's new state.

4 The Experiments

This section describes how the experiments were conducted. First the benchmark tasks are outlined, later the experiment's conditions are recounted.

4.1 The Benchmark Problems

In order to evaluate the modified CA's behavior in the presence of defects, four common CA computational tasks were selected.

The first task is in a 1D CA and it is commonly known as *rule 30*. The designation originates from Wolfram's classification scheme [18] and denotes the number which comes from the new states' values when the transition rule set is sorted by neighborhood values in descending order as shown on Fig. 3. Note that in this case, the CA uses only the simplest 3-cell (r = 1) neighborhood. The rule produces chaotic aperiodic behavior and may be used for e.g. random number generation [18].

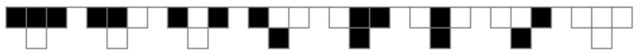

Fig. 3. The sequence of individual transition rules in descending order can form a number from the resulting states. In this case we can transform binary 00011110 into decimal 30 which explains why this transition function is called *rule 30*.

Fig. 4 a) shows several steps in a 1D CA with *rule 30*, where the initial configuration is *seeded* with one active state in the middle.

Another common 1D CA task, which is often used as a benchmark [1, 15], is the problem of *majority* (or alternatively known as *density*). The goal of density computation is a classification of the initial CA configuration by determining whether there is a majority of 0s or 1s. Majority could be easily solved from a global view, however a CA works only with local interactions. In this case, CA uses 7-cell (r = 3) neighborhood and the result should be known in number of steps equal to double the width of the CA. The final result is identified by the final configuration of the CA, where the cellular space is completely filled with the dominant state (Fig. 4 b).

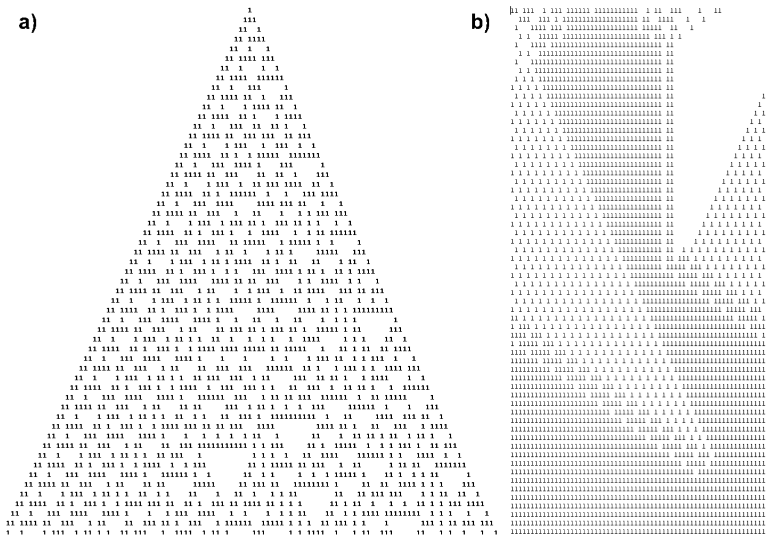

Fig. 4. a) Evolution of rule 30, **b)** successful computation of majority (rule taken from [1]). Note that white spaces represent the cells in state 0.

It is not possible to solve the majority problem for all possible configurations but researchers strive to find the best possible solution e.g. by evolutionary approaches. Their success is measured in the percentage of successful runs on a set of randomly generated initial configurations [15]. For our experiments we selected one of the better known solutions which is capable of correct classification of 82.326% configurations [1]. The rule can be encoded in hexadecimal number by the sequence FFAAFFAAFFAAFFAAA0AA00A0A0AA00A0.

Third task that was selected for our experiments is commonly known as Conway's *Game of Life* [7]. It is well known artificial life simulation with a 2-state 2D CA with von Neumann neighborhood (5 cells - center and 4 immediate neighbors). Rules for Life can be briefly described as following:

1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.

2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overcrowding.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

With these simple rules the behavior can be surprisingly complex and several notable types of structures within the cellular space emerge, including *oscillators* and *spaceships*. One of such spaceships is called a glider: a structure capable of translating trough cellular space by nontrivial (as in multi-step) transformations. The glider is used in the experiments described in this paper and its translation is depicted in Fig. 5.

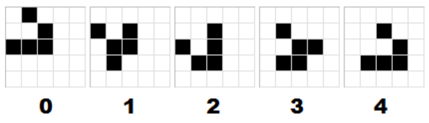

Fig. 5. The translation of the *glider* spaceship in the Game of Life in 4 steps.

Final benchmark task used in our experiments is the self-replication of the so-called *Byl's Loop* [4]. Byl's Loop uses von Neumann neighborhood and apart from the other mentioned benchmarks, it uses 6 states. The loop is capable of replicating itself in 26 steps from its initial configuration and continues to do so until it runs of cellular space. Fig. 6 a) shows the replication of Byl's Loop in 5-step jumps and Fig. 6 b) shows the cellular space after 11 generations of loop replication. The replication mechanism is very carefully connected to the shape of the loop and even minuscule alterations in its configuration cause failure of the replication process.

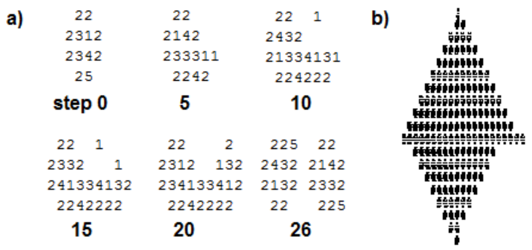

Fig. 6. a) Single Byl's Loop's replication, **b)** cellular space after 11 generation of replications

From the description of the benchmarks, it is clear that tasks with increasing complexity were selected in order to provide the experiments with a breadth of different problems representing wide area of cellular automata applications.

4.2 The Experimental Setup

Each experiment used CA with cyclic neighborhood. That means that each edge cell is connected to the cell on the opposite edge. Size of the CA was always the same: 225 cells, i.e. 15x15 for 2D and 1x225 for 1D. The reason was to provide similar conditions for all experiments.

All initial configurations were carefully set so the maximum possible area of the CA could be used during the 100 steps each CA performed. In the case of rule 30, the initial configuration was seeded by two cells in state 1 on positions 64 and 147 which created the image of two triangles connecting together at the bottom. For the Game of Life computation, three gliders were set in column above each other from the upper left corner of the CA (each glider in the position shown in the 0 frame of Fig. 5) separated by space of two quiescent cells, so the gliders would travel trough the whole CA during the 100-step run. The Byl's Loop initial configuration (see Fig. 6) was set with one loop in the upper left corner of the cellular space so there would be 8 replicated loops after the 100-step run, filling roughly 8/9 of the available cellular space.

The defects in each run were generated on the level of redundant modules. Note that three levels of fault-tolerance were simulated: (i) no redundance (single module per cell), (ii) TMR and (iii) 5MR. Each module could be damaged with the probability ranging from 0,5% to 15% with uniform distribution. Damaged module manifests itself as dead - it always generates state 0. Each case was run 500 times.

A correct run occurs when the CA's configuration after defined number of steps is identical to the configuration of the same CA run without errors for the defined number of steps. Other definitions of correctness are not considered.

5 Results

The tables below are grouped by task type and in order of ascending redundancy of the simulated CA. The tables show the percentage of correct runs with different probabilities of defects.

Special case not shown in the tables is the majority task, which was not so extensively tested as the other benchmarks. The reason was that the used rule shifts the information about density trough the CA (as seen in Fig. 4 b)). The parameters for the few runs were also different - the CA was only 64 cells long as in Fig. 4. If there is a failing "dead" cell in a CA running the majority task, 0 states are shifted from its position and the result turns in favor of 0. There was no point in testing the majority since most of the results ended in failure. On the other hand, the evolution of rule 30 may be distorted by defective cells but it still retains at least part of its shape because the failing cells do not distribute their 0 states so aggressively.

Many of the tables show results of 99% or 98%. Because each experiment was run 500 times, some freak probability cases always caused few of the runs to have several defective adjacent modules. Such occurrences may not be likely, but still possible.

Table 1. Selected results for all benchmarks without fault tolerance. Each single number was obtained from 500 runs.

No FT	**Rate of correct runs up to 30, 50, 80 and 100 steps [%]**											
Prob. of defects [%]	**30**	**50**	**80**	**100**	**30**	**50**	**80**	**100**	**30**	**50**	**80**	**100**
	Rule 30				**Game of Life**				**Byl's Loop**			
0.5	55	42	33	33	56	42	38	38	83	73	56	48
0.7	42	28	22	22	49	43	31	31	75	62	45	34
1	31	16	11	11	32	15	12	12	67	49	30	23
1.5	16	7	4	4	19	7	6	6	53	34	20	12
2	6	1	0	0	11	3	3	3	50	26	12	8
2.5	0	0	0	0	3	0	0	0	38	16	6	2
3	0	0	0	0	0	0	0	0	30	10	3	0

Table 2. Selected results for all benchmarks with TMR. Each single number was obtained from 500 runs.

TMR	**Rate of correct runs up to 30, 50, 80 and 100 steps [%]**											
Prob. of defects [%]	**30**	**50**	**80**	**100**	**30**	**50**	**80**	**100**	**30**	**50**	**80**	**100**
	Rule 30				**Game of Life**				**Byl's Loop**			
0.5	99	98	98	98	98	97	97	97	99	99	99	97
0.7	98	97	96	96	98	97	97	97	99	99	98	97
1	95	94	93	93	97	95	94	94	98	98	96	95
1.5	91	88	86	86	92	88	88	88	97	96	93	92
2	86	79	76	76	89	83	82	82	94	92	87	85
2.5	78	69	64	64	82	73	72	72	93	88	82	77
3	68	60	54	54	74	63	62	62	87	83	74	69
4	56	41	33	33	61	49	46	46	75	59	42	34
5	55	40	33	33	42	27	23	23	75	59	42	34
8	10	4	2	2	15	5	4	4	48	27	11	6
10	3	0	0	0	3	0	0	0	33	14	3	1
15	0	0	0	0	0	0	0	0	8	1	0	0

Table 3. Selected results for all benchmarks with 5MR. Each single number was obtained from 500 runs.

TMR	**Rate of correct runs up to 30, 50, 80 and 100 steps [%]**											
Prob. of defects [%]	**30**	**50**	**80**	**100**	**30**	**50**	**80**	**100**	**30**	**50**	**80**	**100**
	Rule 30				**Game of Life**				**Byl's Loop**			
0.5	99	99	99	99	99	99	99	99	99	99	99	99
1	99	99	99	99	99	99	99	99	99	99	99	99
1.5	99	99	99	99	99	99	99	99	99	99	99	99
2	98	97	97	97	98	98	98	98	99	99	99	98
2.5	97	96	95	95	98	98	98	98	99	98	98	97
3	96	95	93	93	97	95	95	95	99	99	98	98
4	94	89	87	87	92	88	87	87	96	95	94	92
5	87	81	78	78	89	81	79	79	95	92	87	84
6	79	68	64	64	84	77	76	76	92	87	80	76
7	72	58	51	51	42	73	61	58	88	81	71	66
8	58	47	38	38	59	46	43	43	77	64	49	40
10	33	18	11	11	39	25	22	22	74	57	41	33
12.5	13	3	1	1	16	7	5	5	53	33	16	11
15	3	0	0	0	4	0	0	0	36	15	4	2

6 Conclusion

The results give the indication that static NMR in cellular automata can work with the presence of low defect rate. There was no surprise when 5MR showed substantially better results than TMR. It is arguable which method would suffice if any. Some sources [12] claim that future materials based on nanotechnology could have defect rate up to 10%. However, such claims are at best questionable since we can't analyze future materials and even if some of them already exist, it is quite possible that the fabrication process would improve significantly when or if cellular systems are actually implemented. It is also possible that the defect rate would be even higher.

Another observation which should be considered is the difference between the benchmark task's results. The tables show that some tasks are more sensitive to errors caused by defects than others. Especially Byl's Loop shows lower sensitivity. However, the majority problem indicates that it cannot work with defects at all which was a little surprising. From the intuitive viewpoint, one could assume that few defects could not make such a difference with a configuration that has overwhelming majority of one state. Evidently, some CA tasks work in ways which make them extremely sensitive to permanent errors.

The initial configurations were designed so the runs would use most of the cellular space but those configurations also need to be taken into account, especially with the rule 30 and Byl's Loop which don't use most of the cellular space until late in the computation. If those tasks were configured differently, they could possibly show similar behavior to the majority problem.

In conclusion, it is safe to say that although cellular automata look like uniform arrays of elements, the nature of computations running in them makes analysis of their error susceptibility an assignment dependent on said computations and that such analyses can not be generalized. Moreover the fault-tolerance in cellular systems can be increased by very simple and straightforward manner which NMR undoubtedly is. NMR could also be independent on hardware platform since grouping and connecting several modules in each cell should be much easier task than adding multiple layers of closely connected complex "circuitry".

For future work, we plan to model cellular systems with additional types of errors, possibly in GPU accelerated cellular automata simulator which we successfully implemented [19]. The defects could be modelled differently e.g. by permanent state of 1 or by random state. Other issue is the mathematical reliability model for TMR, which is quite straightforward, however we feel the need to include the CA's task so the model would make sense for the purpose of designing fault-tolerant CA applications. Experiments described in this paper also inspired us to investigate more thoroughly, how the presence of errors would change the behavior of known CA, especially their membership in Wolfram Classes [17].

Acknowledgements This work was partially supported by the grant Natural Computing on Unconventional Platforms GP103/10/1517, the FIT grant FIT-11-S-1 and the research plan Security-Oriented Research in Information Technology, MSM0021630528.

References

1. Andre, D., Bennett, III, F.H., Koza, J.R.: Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In: Proceedings of the First Annual Conference on Genetic Programming. pp. 3–11. GECCO '96, MIT Press, Cambridge, MA, USA (1996)
2. Beckett, P., Jennings, A.: Towards nanocomputer architecture. In: Proceedings of the seventh Asia-Pacific conference on Computer systems architecture. pp. 141–150. CRPIT '02, Australian Computer Society, Inc., Darlinghurst, Australia, Australia (2002)
3. Bersini, H., Detour, V.: Asynchrony induces stability in cellular automata based models. In: Brooks, R.A., Maes, P. (eds.) Artificial Life IV. pp. 382–387. MIT Press, Cambridge, Massachusetts (1994)
4. Byl, J.: Self-reproduction in small cellular automata. Physica D 34(1-2), 295–299 (1989)
5. Durbeck, L., Macias, N.: The cell matrix: An architecture for nanocomputing. Nanotechnology 12(3), 217–230 (2001)
6. Dysart, T., Kogge, P.: System reliabilities when using triple modular redundancy in quantum-dot cellular automata. In: Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS '08. IEEE International Symposium on. pp. 72 –80 (2008)
7. Gardner, M.: The fantastic combinations of john conway's new solitaire game 'life'. Scientific American 223, 120–123 (October 1970)
8. Garzon, M.: Models of massive parallelism: analysis of cellular automata and neural networks. Springer-Verlag, London, UK (1995)
9. Lala, P.K. (ed.): Self-checking and fault-tolerant digital design. Morgan Kaufmann Publishers Inc., San Francisco, CA (2001)
10. Li, W.: Phenomenology of nonlocal cellular automata. Journal of Statistical Physics 68, 829–882 (1992), 10.1007/BF01048877
11. Mange, D., Sipper, M., Stauffer, A., Tempesti, G.: Towards Robust Integrated Circuits: The Embryonics Approach. Proceedings of IEEE 88(4), 516–541 (2000)
12. Peper, F., Lee, J., Abo, F., Isokawa, T., Adachi, S., Matsui, N., Mashiko, S.: Fault-tolerance in nanocomputers: A cellular array approach. Nanotechnology, IEEE Transactions on 3(1), 187 – 201 (2004)
13. Sekanina, L., Komenda, T.: Global control in polymorphic celular automata. Journal of Cellular Automata 0, 1–21 (2010)
14. Sipper, M.: The emergence of cellular computing. Computer 32(7), 18 –26 (Jul 1999)
15. Sipper, M.: Evolution of Parallel Cellular Machines: The Cellular Programming Approach, Lecture Notes in Computer Science, vol. 1194. Springer-Verlag Berlin Heidelberg (1997)
16. Toffoli, T., Margolus, N.: Cellular Automata Machines: A New Environment for Modeling. MIT Press, Cambridge, MA (1987)
17. Wolfram, S.: Cellular Automata and Complexity: Collected Papers. Addison-Wesley, Reading, MA (1994)
18. Wolfram, S.: A New Kind of Science. Wolfram Media, Inc., Champaign, IL (2002)
19. Zaloudek, L., Sekanina, L., Simek, V.: Gpu accelerators for evolvable cellular automata. In: Computation World: Future Computing, Service Computation, Adaptive, Content, Cognitive, Patterns. pp. 533–537. IEEE (2009)