Simulation of the Dynamic Behavior of One-Dimensional Cellular Automata Using Reconfigurable Computing

Wagner R. Weinert, César Benitez, Heitor S. Lopes*, and Carlos R. Erig Lima

Bioinformatics Laboratory, Federal University of Technology Paraná (UTFPR),
Av. 7 de setembro, 3165 80230-901, Curitiba (PR), Brazil
hslopes@pesquisador.cnpq.br, erig@utfpr.edu.br

Abstract. This paper presents the implementation of an environment for the evolution of one-dimensional cellular automata using a reconfigurable logic device. This configware is aimed at evaluating the dynamic behavior of automata rules, generated by a computer system. The performance of the configware system was compared with an equivalent software-based approach running in a desktop computer. Results strongly suggest that such implementation is useful for research purposes and that the reconfigurable logic approach is fast and efficient.

1 Introduction

Cellular automata (CA) are discrete distributed systems formed by simple and identical elements. The dynamic behavior of a CA is represented by its evolution along time and this evolution depends on a transition rule [8]. Finding a transition rule capable of modelling a given behavior is a rather difficult task, since the search space promptly becomes computationally intractable. Many works in the literature proposed the use of evolutionary computation techniques for the task of finding suitable transition rule that leads a CA to display a desired behavior (see, for instance, [3,5,6]). Usually, the basic approach in these cited works is to induce rules. Rules are evaluated according to a fitness function, regarding its utility to simulate the desired behavior. Iteratively, the best performed rules are selected and modified by using genetic operators. The evolutionary process finishes after n generations when a given rule achieves the a satisfactory behavior. Frequently, the computation of the fitness function is the most time-consuming task in an evolutionary computation algorithm. Recently, we have witnessed a pronounced growth of the hardware and software technologies for embedded systems, with many technological options arising every year. In particular, applications based on CA can found in recent works [1,2,7]. In our approach to use evolutionary computation for inducing transition rules for cellular automata, the fitness function is computed by simulating the dynamic behavior of the automaton. This is accomplished evolving many automata using a single

* This work was partially supported by the Brazilian National Research Council – CNPq, under research grant no. 305720/2004-0 to H.S. Lopes.

© Springer-Verlag Berlin Heidelberg 2007

transition rule. Such automata are randomly generated and evolved during a fixed number of time steps. The computation of the fitness function is based on the comparison of the obtained behavior and the desired one. The discovery of transition rules for CAs requests the computation of a fitness function that is computationally-intensive. To reduce this computational cost, we propose a methodology for evolving CAs using reconfigurable computing. The system was developed in VHDL (VHSIC Hardware Description Language) and implemented in a FPGA (Field Programmable Gate Array) device.

The case-study addressed in this work is the classic problem of a CA to perform a density classification task [1,3,5,6]. A typical evolutionary computation algorithm for finding a transition rule for such CA will need to try some 1500 different rules until finding a good one. For each rule, its appropriateness is evaluated by applying it to a randomly generated automaton and evolving it for a given number of time steps. Due to the stochastic nature of the algorithm and the randomness of the initial condition of the automaton, around 10,000 different automata are evolved for each rule, and results are statistically evaluated.

In this work it is not aimed to propose any evolutionary computation technique for inducing CAs, but evaluating the usefulness of reconfigurable computing as a hardware accelerator for evolving CAs, in comparison with a software-based approach.

2 Cellular Automata and the Density Classification Task

A Cellular Automata is defined by its cellular space and by its transition rule. The cellular space is represented by a lattice of N cells connected according a boundary condition in a d-dimensional space. The transition rule gives the next state for the cell, considering the configuration of its current neighborhood. At each time step, all cells in the lattice update their current state, according to the transition rule (representing the dynamic nature of the system)[8]. A formal definition of CA is given by [4]:

$$\begin{cases} \Sigma: & \text{set of possible states for each cell;} \\ k: & \text{cardinality of } \Sigma; \\ i: & \text{index of each cell;} \\ S_i^t: & \text{state of a given cell } i \text{ at time } t\ (S_i^t \in \Sigma); \\ \eta_i^t: & \text{neighborhood of cell } i; \\ \Phi(\eta_i): & \text{transition rule that leads cell } i \text{ to the next state } (S_i^{t+1}) \text{ as function of } \eta_i^t. \end{cases} \tag{1}$$

Two parameters are necessary to deal with the neighborhood concept: (m) that represents the size of the neighborhood, and (r) representing its radius. Usually, parameter m is given as function of r, in the form: $m = 2r + 1$. Since a CA is represented by a linear structure, the leftmost cell (i_0) and the rightmost cell (i_{n-1}) of a one-dimensional automaton do not have left and right neighbors, respectively. Therefore, a bounding condition is necessary, such that the leftmost cell is connected with the rightmost cell, and then the transition rule $\Phi(\eta_i)$ can be applied to the whole lattice. Considering that the number of cells of a given neighborhood is $2r+1$, the number of different neighborhood that can be generated using a given rule is $k^{(2r+1)}$. Also, the number of possible transition rules that can be generated for a CA is $k^{k^{(2r+1)}}$.

The dynamic behavior of an one-dimensional CA generated by the application of a transition rule over the automaton for n time steps is usually illustrated by a spatiotemporal diagram. In such diagram, the configuration of states in a lattice is plot as a function of time.

The number of possible rules for a given automaton is $k^{k^{(2r+1)}}$. The larger k and r, the larger the set of rules applicable to a given CA. In most cases it is not computationally feasible to evaluate the whole set of rules so as to find a given one that explains the dynamic behavior of a system.

The density classification task, also known as majority problem, is a classical problem in CA theory [8]. The objective is to find a transition rule such that, when applied for M time steps to a random initial configuration, will lead all cells either to state 0 or state 1, depending on the density of the initial configuration. The parameter ρ is defined as a threshold for the density classification task [4], in such a way that ρ_0 represents the density of cells in state 1 in the initial configuration, and ρ_c is the same density in a given configuration (for $\forall t > 0$).

The density classification task can be modelled in several ways. Here we use the approach proposed by [3]. In this model, all cells can have binary states ($k=2$), the lattice is composed by an odd number of cells ($N=149$), and the neighborhood radius of $r=3$. The number of possible transition rules for this CA is 2^{128}.

3 Methodology

To implement the system, we used the Altera Quartus II development system, version 5.1, and a Cyclone EP1C6Q240C8 FPGA device. The software running in the desktop PC was developed in C++ language.

The software running in the desktop PC generates a transition rule (as part of other evolutionary strategy for inducing rules), encodes and sends it to the FPGA through a parallel interface. Inside the FPGA, several hardware blocks (see Figure 1 perform the evaluation of the transition rule, and send back both, the result of such evaluation and the processing time.

Using a serial to parallel converter, a 128-bits long transition rule is assembled (from the parallel interface block) and stored in the register block.

The command decoder block decodes the received commands and executes control actions in the other FPGA blocks: the pseudo-random number generator block, the CA evolver block, the chronometer block and to the accuracy calculator block. In the current version, five different commands can be decoded and executed:

– Internal reset.
– Restart MLS pseudo-random number generator.
– Clear current transition rule in register.

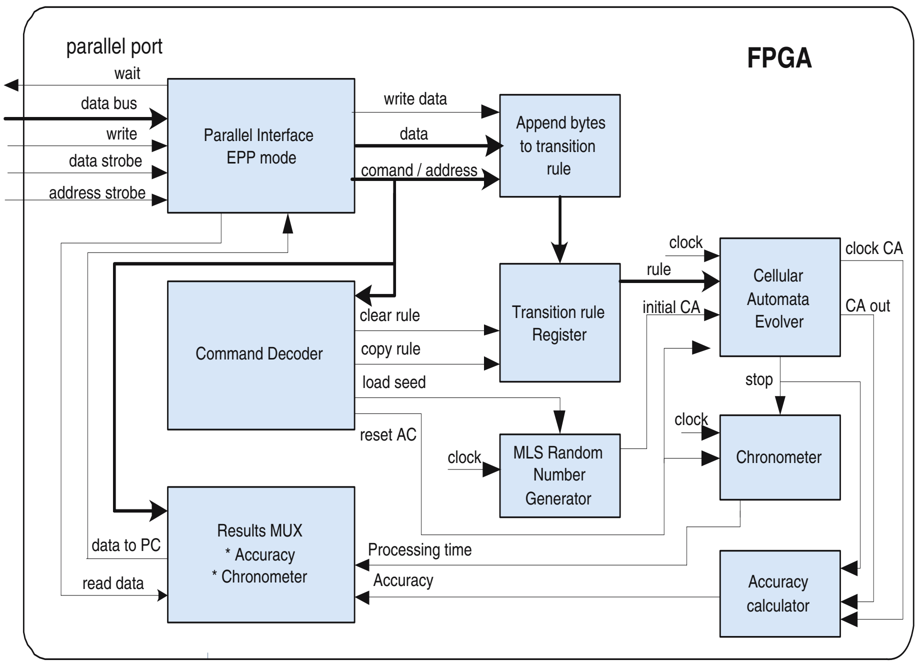

Fig. 1. Block diagram of the proposed system for evolving CAs

– Latch transition rule.
– Start CA evolution.

The CA evolver block receives a previously registered transition rule and an initial configuration for the automaton, generated by the pseudo-random number generator, and evolves the CA for a given number of iterations (in our case, 200). Inside this block there are two Finite State Machines (FSM). The first one controls the number of iterations of the CA. The second FSM controls the execution of 10,000 runs, with a random initial configuration.

The accuracy calculator block verifies if the final configuration is the expected one, considering the density of the initial configuration. If positive, the score of the rule under evaluation is incremented. At the end of all runs, this block retains the total number of hits, which, divided by the number of runs and taken as a percentage, is the accuracy rate.

To generate random initial configurations, a Maximum Length Sequence (MLS) pseudo-random number generator is used [1]. MLS is an n-stage linear shift-register that can generate binary periodical sequences of maximal period length of $L = 2^n - 1$ These sequences are referred to as maximal-length sequences (MLS), and n is said to be the degree of the sequences. In our work, we used $n = 16$, thus generating 16-bits sequences. The random seed is loaded into the

[1] Available in *http://www.ph.ed.ac.uk/~jonathan/thesis/node83.html*

shift-register by a command. Ten parallel shift-registers were implemented, and out from these 160 bits, 149 are used for the initial configuration of the CA.

The chronometer block counts and register the total elapsed time for generate, evolve and evaluate the behavior of 10,000 CAs. The multiplex block selects output data between the computed accuracy rate and the processing time, to be sent out to the desktop computer.

4 Results

The evaluation of the proposed system consists of five experiments. For each experiment, 10,000 one-dimensional CAs were randomly generated and evolved for 200 iterations. For these experiments, we used a single rule, proposed by Juillé et al. [3], named “Coevolution(1)”. To date, this rule is supposed to be the best known rule for the density classification task. Results take into account the average accuracy (that evaluates how good the rule performs) and the average processing time (that represents the computational cost). In order to compare such results with other implementation, a similar software was developed in C++ programming language, compiled without any optimization flag and run in a desktop computer under Microsoft Windows XP operational system. Exactly the same rule and parameters were used for both systems, and Table 1 shows the results obtained. The FPGA was run at 33,33 Mhz and the real clock of the desktop PC was 1800 MHz. The comparison between different hardware platforms such as a desktop PC and FPGA-based system is not completely fair, but necessary to emphasize the dramatic performance difference.

Table 1. Comparison PC versus hardware-based approach

Device	Accuracy	Processing Time
FPGA CYCLONE EP1C6Q24068	82.28	58.660ms
PC AMD Athlon XP 2400+, 512MB	79.33	3h:9m:26.6s

5 Conclusions and Future Work

In this work we described a reconfigurable computing system for evolving cellular automata. This system is to be used in conjunction with a software application, running in a desktop PC, to study the behavior of transition rules. The reconfigurable hardware is able to communicate with any software application, since a standard parallel interface was implemented.

For this work, the average accuracy of the system has small importance, since the main focus is the processing speed. The difference of 2.95% between the accuracy rates of the hardware-based system and the software-based system is due to different random number generators, responsible for the generation of the initial configuration of the CAs.

The processing time needed to evolve the CAs in the PC is extremely high, when compared with the time needed by the hardware approach. This is due to the fact that, in the PC, processing is sequential, and in the FPGA parallelization was largely explored. Even considering that the FPGA was running at 33,33 MHz and the PC was 54 times faster, the processing time of the later was more than 5 orders of magnitude higher than the former. This makes evident the great advantages of reconfigurable computing, strongly suggesting its adequacy for this kind of task.

Future work will focus on the implementation of some evolutionary computation technique to search transition rules for one-dimensional CA, using the reconfigurable computing module as an external accelerator. Also, other levels of parallelization will be sought, thus improving even more the efficiency of the system.

References

1. Corsonello, P., Spezzano, G., Staino, G., et al.: Efficient implementation of cellular algorithms on reconfigurable hardware. In: *Proc. of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing* (2002) 211–218.
2. Halbach, M., Holffmann, R.: Implementing cellular automata in FPGA logic. In: P*roceedings of the 18th International Parallel and Distributed Processing Symposium* (2004) 258–262.
3. Juillé, H., Pollack, J.B.: Coevolving the ideal trainer: application to the discovery of cellular automata rules. In: Koza, J.R et al. (eds.), *Genetic Programming 1996: Proc. 3rd Annual Conference*, (1998) 519–527.
4. Mitchell, M.: Computation in cellular automata: a selected review. In: Gramms, T., ed., *Nonstandard Computation.* VCH Verlagsgesellschaft, Weinheim (1996).
5. Morales, F.J., Crutchfield, J.P., Mitchell, M.: Evolving two-dimensional cellular automata to perform density classification: a report on work in progress. *Parallel Computing* **27** (2001) 571–585.
6. Oliveira, G.M.B., Bortot, J.C., Oliveira, P.P.B.: Multiobjective evolutionary search for one-dimensional cellular automata in the density classification task. In: *Proc. 8th Int. Conf. on Artificial Life*, (2002) 202–206.
7. Shackleford, B., Tanaka, M., Carter, R.J., et al.: FPGA implementation of neighborhood-of-four cellular automata random number generators. In: *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, (2002) 106–112.
8. Wolfram, S.: *Cellular Automata and Complexity.* Westview Press, Boulder (1994).