An FPGA Design for the Stochastic Greenberg-Hastings Cellular Automata

Nikolaos Vlassopoulos
INRIA Nancy-Grand Est,
LORIA MAIA Team,
Nikolaos.Vlassopoulos@loria.fr

Nazim Fatès
INRIA Nancy-Grand Est,
LORIA MAIA Team,
Nazim.Fates@loria.fr

Hugues Berry
INRIA Rhône-Alpes, Combining Team,
Université de Lyon, LIRIS
Hugues.Berry@inria.fr

Bernard Girau
Université Henry Poincaré Nancy 1,
LORIA Cortex Team,
Bernard.Girau@loria.fr

ABSTRACT

The stochastic Greenberg-Hastings cellular automaton is a model that mimics the propagation of reaction-diffusion waves in active media. Notably, this model undergoes a phase transition when the probability of excitation of a cell varies. We developed a specific FPGA design to study the critical behavior of this model. Using dedicated architectural optimizations, we obtain a significant speed-up with respect to software simulation for lattice sizes of 512×512*. We exploited this speed-up to obtain improved estimations of the critical threshold.Our results indicate the existence of an asymptotic value of this threshold when the number of cell states increases.*

KEYWORDS: Cellular Automata Models and Algorithms, Fine-Grained Parallel Architectures and FPGA, Efficient Architectures and Implementations, Local Neighborhood and Topology Awareness

1. INTRODUCTION

Reaction-diffusion systems are a class of dynamical systems that have been used to model a large class of natural phenomena. In 1952, Turing proposed to couple reactions and diffusion of some imaginary chemical components, the *morphogens*, to explain how structured patterns could appear in initially symmetric organisms [1]. Since then reaction-diffusion systems were used to describe natural phenomena in various fields, such as mollusk shell pigmentation [2], heart modelling [3], etc. Beyond the modeling of natural phenomena, reaction-diffusion media have also been suggested as possible means for building new types of computing devices [4].

Here, we focus on a discrete version of reaction-diffusion systems. The model we study is a stochastic variation of the Greenberg-Hastings cellular automaton [5] (GHCA). In spite of its apparent simplicity, this model displays complex behaviors. In particular, when the transmission probability varies, the GHCA undergoes a critical phase transition from an active (alive) into an absorbing (dead) phase [6, 7]. This critical behavior is surprisingly robust and is preserved even when large amounts of failures or defects are added to the system [8]. Such properties are especially suitable for technology-oriented applications. For instance, we have recently proposed a bio-inspired system based on GHCA, to achieve decentralized and robust gathering of mobile agents scattered on a surface [7].

Several aspects of the GHCA and its critical behavior remain poorly understood. For instance, when the number of states (see below) is not too large, the critical behavior is known to be in the universality class of directed percolation [7]. At the limit when this number tends to infinity, the GHCA reduces to a SIR epidemic model with perfect immunization [9], which is in a different universality class (namely dynamic percolation), e.g. presents a different critical behavior [10]. Hence the behavior of the GHCA with large (but finite) number of states is unclear, albeit our preliminary results suggested the existence of an asymptotic value for the critical threshold [8].

To test the behavior of the GHCA with a large number of states is a computational challenge because one needs to average over large numbers of simulations to reach sufficient statistics for increasing simulation times. This imposes a limit on software-based simulations, since in sequential machines, the inherent parallelism of CA has to be emulated. Typically, this is achieved by calculating the time evolution of each cell separately and using double buffers to simulate the parallel nature of CA, thus leading to a considerable slowing down of the simulations. The primary objective of the present study is to implement the GHCA model on an FPGA device so as to leverage the natural parallelism of FPGAs, thus to reach significant speed-ups with respect to software simulation. This gain is then utilized to obtain improved simulations of the behavior of the GHCA with an increasing number of states.

The article is organized as follows. We first present the GHCA model and our motivation for its FPGA implementation in Sec. 2. Then, Section 3 describes the more technical aspects of the FPGA architecture that was designed. The results of the implementation are given in Sec. 4 while the simulation results are shown and analyzed in Sec. 5. Finally we present our conclusions in Sec. 6.

2. MODEL

2.1. Model Description

A cellular automaton (CA) is a discrete, spatially extended dynamical system. The CA we study is composed of a two-dimensional array of cells, $\mathcal{L} \subset \mathbb{Z}^2$. Each cell can be in one of the states in $\{0, \ldots, M\}$, where the state 0 is called the *neutral* state, state M is called the *excited* state and the other states are called the *refractory* states. The state of a cell $c \in \mathcal{L}$ at time t is denoted by σ_c^t. At each time step, the state of a cell evolves according to is own current state and the state of its neighboring cells, $\mathcal{N}(c)$. The rules that define the evolution of the stochastic Greenberg-Hastings CA are:

- A cell in the neutral state becomes excited, with probability p_T if at least one of its neighbors is excited.
- A cell in a refractory state, $\{1, \ldots, M-1\}$, will evolve independently by decreasing its state until it reaches the neutral state.
- A neutral cell, whose neighbors are all neutral or refractory remains in the neutral state.

Here p_T is called the *transmission rate*; it is the main parameter of our study. More formally, let $\mathcal{B}(p_T)$ be a Bernoulli random variable that takes the value 1 with probability p_T and the value 0 with probability $1 - p_T$. Further, let E_c^t denote the set of excited cells in the neighborhood of c at time t, $E_c^t = \{c' \in \mathcal{N}_c : \sigma_{c'}^t = M\}$. Then

$$\sigma_c^{t+1} = \begin{cases} M, & \sigma_c^t = 0, |E_c^t| > 0 \text{ and } \mathcal{B}(p_T) = 1 \\ \sigma_c^t - 1, & \sigma_c^t \in \{1, \ldots, M-1\} \\ 0, & \text{otherwise} \end{cases}$$

For our experiments we use two different types of neighborhood, the *4-connected neighborhood*, $\mathcal{N}_4$, defined as

$$\mathcal{N}_4(c) = \{c' \in \mathcal{L} : |c_x - c'_x| + |c_y - c'_y| = 1\}$$

and the *8-connected* neighborhood, $\mathcal{N}_8$, defined as

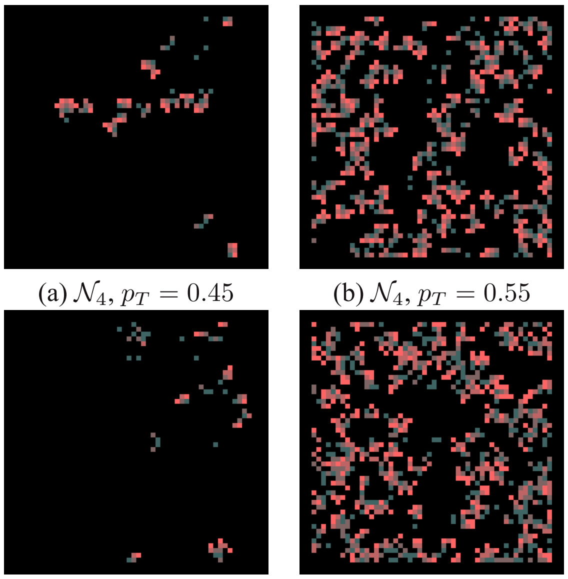
$$\mathcal{N}_8(c) = \{c' \in \mathcal{L} : \max(|c_x - c'_x|, |c_y - c'_y|) = 1\}$$

Finally, for the simulation environment, we used square lattices with dimension L and toric boundary conditions so that the indices of cells, (c_x, c_y), are taken in $\mathbb{Z}/L\mathbb{Z}$.

2.2. Analysis of Phase Transitions

As mentioned in the introduction, the stochastic reaction-diffusion CA exhibits a phase transition as p_T varies. The origin of the transition is a change in the probability that the process will be active, i.e. that the number of excited cells will be greater than zero, as $t \to \infty$ for infinite size systems. The value of p_T where the phase transition occurs is called the *critical threshold*, p_{T_C}. For values of p_T lower than p_{T_C}, the probability that the reaction-diffusion process will be active as $t \to \infty$ is zero, while for values greater than p_{T_C}, this probability becomes non-zero. Figure 1 shows two examples of the phase transition for $\mathcal{N}_4$ and $\mathcal{N}_8$.

The most straightforward method to determine p_T is to study the evolution of the density of excited cells with respect to time (see for instance [11, 7]). In the current work we focus on another important time-dependent scaling property of directed percolation, the probability that a cluster grown from a single seed will be active after t time steps [12]. For the GHCA, this amounts to studying the probability that a reaction-diffusion process starting from a single cell in the excited state will still be active after t time steps. We expect, and seek to verify that this probability scales as $t^{-\delta}$ in the critical region [12]. We denote the probability that a reaction-diffusion process survives after time t by $P_s(t)$.
To evaluate $P_s(t)$ we used the following process: let D denote the random variable that expresses the time that a process reaches extinction (time of "death"). Further, let n_t be the number of experiments, such that the reaction-diffusion process reaches extinction at time t, with M and p_T fixed. Then, the probability that a process will "die" at time τ can be expressed as $P(D = \tau)$ and can be estimated using $\frac{n_\tau}{n_{\text{tot}}}$, where n_{tot} is the total number of experiments carried out for


(a) $\mathcal{N}_4, p_T = 0.45$ (b) $\mathcal{N}_4, p_T = 0.55$

(c) $\mathcal{N}_8, p_T = 0.20$ (d) $\mathcal{N}_8, p_T = 0.30$

Figure 1. Examples of the Stochastic Reaction-Diffusion Process Obtained for $M = 4$. Cells in the neutral state are shown in black and cells in the excited state in bright red. (a) and (b) are examples of sub- and super-critical p_T for the 4-connected topology. (c) and (d) correspond to sub- and super-critical values of p_T for the 8-connected topology. In (a) and (c) the process will quickly fall into the absorbing phase (all cells in neutral state). The snapshots were taken with FiatLux [13].

the specific M, p_T and L values. The cummulative distribution

$$P_d(t) = P(D < \tau) = \sum_{\tau=1}^{t-1} P(D = \tau)$$

describes the probability that the process will become extinct in the time interval $[1, t)$. From this, we can calculate the probability that the process will survive up to time t as $P_s(t) = 1 - P_d(t)$. The above processing was performed on the sampled data in an off-line fashion. Finally, an individual experiment is defined by the values of five parameters that characterize the environment and the cell transition function, namely M, p_T, $t_{\max}$, $\mathcal{N}_c$ and L, where $t_{\max}$ is the maximum number of steps an experiment is allowed to run.

To identify the critical threshold, p_{T_C} where the phase transition between survival and extinction of the process occurs, we need to run experiments for a large number of time-steps, close to the critical threshold, p_{T_C}. The number of experiments for each value of p_T should be such that the resulting measurements exhibit as less noise as possible. Each iteration of an individual experiment consists of simulating the CA and measuring the number of time steps required until the reaction-diffusion process becomes extinct. Each individual experiment was repeated from at least 10^4 times up to 10^5 times, with the actual number depending on the quality of the results. To evaluate the quality, after each $t_{\max}$ iterations, we processed the results and visually inspected the figures in log-log scale, so as to ensure that the lines did not exhibit too much noise (which could cause the lines to cross) and repeated the experiment for another 10^4 iterations, if necessary.

To further justify our choice for using FPGAs for the simulation process, it is worth mentioning that the time needed for the experiments amounted to a couple of weeks, on a single FPGA, despite a great speed-up factor (see § 4.3). Time is the main limiting factor for the precision in which we determine p_{T_C}.

3. SIMULATION ENVIRONMENT AND ARCHITECTURE

For conducting the experiments, we have set up an FPGA-based environment for simulating the reaction-diffusion process. The main motivation for using FPGAs was that their regular, two-dimensional structure provides an ideal architecture for mapping the structure of a cellular automaton. The architectural approach we use is similar to the one described in [14], [15], which is based on dividing the simulation environment into groups of cells and calculating each group in parallel. The groups of cells that form the entire environment are then processed sequentially. The main difference with the approach followed there is that we use a technique that reduces the required memory but increases the simulation time.

3.1. Objectives

Our objective was to implement a tool that helps the generation of FPGA designs for simulating cellular automata. The generated designs are configurable up to a certain degree and depend on an external host for initializing their parameters and running the simulations. This is an improvement with respect to our previous work [14], since it allows us to run a wider range of experiments on each FPGA design. The current version of the tool supports measuring the number of active cells in the environment. The maximum number of time steps, $t_{\max}$ is a run-time configuration option and each experiment terminates either after $t_{\max}$ time steps or when a condition on the density is met (*terminating condition*), in our case, when there are no more active cells in the environment.

The generated designs use a simple memory-like interface that maps the runtime and design time parameters to memory locations and allows the host software to query the static parameters and modify the current state of the simulated CA. In our designs, the memory interface allows the software to read and write the state of each cell, configure the excited cell state and modify the probability of transmission (p_T), as well as read parameters that were used to generate the design.

3.2. Architecture

The overall architecture of the FPGA is depicted in Fig. 2. The architecture is based on dividing the environment space into groups of cells, so that each group is processed in parallel, while the different groups are processed sequentially. The simulation environment depends on an external host that initializes each simulation and reads back the simulation results. The initialisation process typically consists of

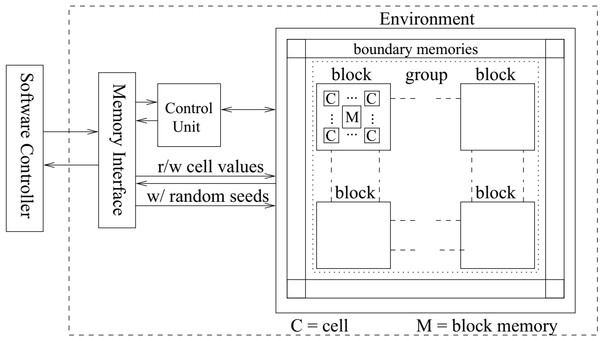


Figure 2. Outline of the Architecture of the Simulation Environment

the following steps: (1) initialise the random number generators with random seeds generated by the host, (2) clear the CA array, (3) set the p_T value and (4) set the initial state of the array.

3.2.1. Structure of the Environment and Definitions

The structure and organization of the environment follows the *block-synchronous* method, as described in [14], [16] and [15]. The main characteristic of this approach is that it provides a solution to the problem of *environment scaling*, which is caused by the resource utilization of the cells. Environment scaling refers to the limits imposed on the number of cells that can be fit in an FPGA, as the environment size increases, because of the limited FPGA resources. In our case, it is important to note that the resources are not so much consumed by the computation of the transition function, but much more by the generation of the random events. Finally, the block-synchronous approach exploits the regular structure of FPGAs and provides a design methodology that maps the regular two-dimensional structure of a CA to an FPGA.

The approach consists in organizing a set of cells around a memory, so that they form a block. The notion of a *block* is that of a self-contained partition, i.e., a partition that holds the necessary information and processing units in order to simulate a subset of the environment. Blocks are further organized in a *group* so that a group is equivalent to a top-level partition of the environment that can be processed in a completely autonomous and parallel fashion. Since, in FPGAs, the memory resources are regularly distributed among the configurable logic slices, the block-synchronous method tries to organize the CA so as to be roughly topologically equivalent to the device, i.e. by arranging a block of cells around a memory. For an optimized partitioning of the CA with respect to FPGA resources, see [15].

Let us define the following: let G_x and G_y be the number of groups in the two spatial dimensions and B_x, B_y the number of blocks in the two dimensions of the group. Further, let (g_x, g_y) be the coordinates of a group in the environment and (b_x, b_y) the coordinates of a block within the group. Finally, we will use s_b to denote the number of bits used to store the state of each cell and C_x and C_y to denote the number of cells in the two spatial dimensions of each group.

3.2.2. Cell Architecture

The main computational unit of the simulation environment is the *cell*. For each time iteration, each cell of the environment must update its state according to the rules of Sec. 2. An outline of the cell architecture is depicted in Fig. 3. It consists of two main parts, one for generating the Bernoulli random event and one for generating the output state. In the

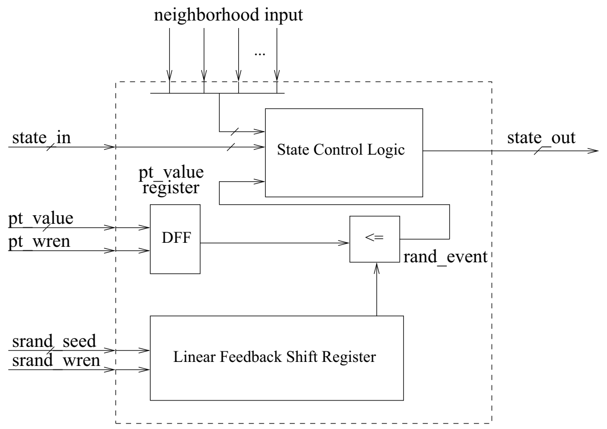


Figure 3. Outline of the Cell Architecture.

reaction-diffusion process, the information communicated from and to the neighbors can be reduced to a single bit, since we are interested in knowing only if a neighbor is excited or not, and the state computation is trivial. The main computational part of each cell is the random number generation, which consumes the greatest part of the resources when scaling the architecture to higher parallelism.

3.2.3. Random Number Generation

The Bernoulli random event generator for the transmission probability was implemented using a Linear Feedback Shift Register (LFSR) and a comparator. Ideally, the random event generator should use as many LFSRs as the number of bits required to achieve the desired precision for the value of p_T, extracting one bit from each LFSR to generate the random word. Alternatively, we could use the same LFSR and extract the number of required bits, one at each clock step. Unfortunately, the first approach increases the FPGA resources occupied by each cell and reduces the number of cells that can be implemented in parallel, while the later introduces large delays in the computation. As a compromise, we extract the required bits from the same LFSR by reading the bit values from different positions of the shift register (Figure 4). To avoid immediate spatial dependencies, the bits were extracted from non-regular intervals. The required

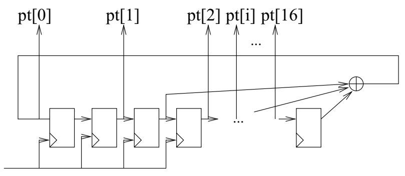


Figure 4. Generation of the Random Word used in the Random Events.

precision for p_T was of the order of 10^{-4}, a value which can be achieved by using 17 bits for the pseudorandom word. To ensure that the period of the random numbers will not introduce problems in the simulation, the LFSR length has been chosen to be 168 bits, following the design guidelines described by George and Alfke [17].

3.2.4. Block Architecture

The overall architecture of a block is depicted in Fig. 5. Each block is built around an FPGA Block RAM module that stores the state of the cells that form the block. Our approach uses a single buffer implementation, that allows for larger environment sizes at the cost of doubling the time required to complete a time iteration. The architecture is similar to the one described by Girau et al. [15], the main difference being the extra control that is required to allow the host program to read and write the value of a cell.

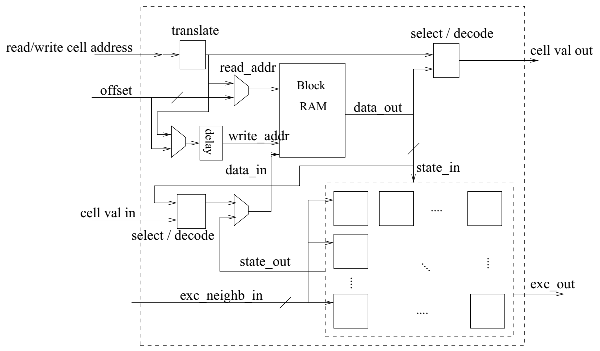


Figure 5. Outline of the Block Architecture.

3.2.5. Group and Environment Architecture

A group of blocks consists of $B_x B_y$ blocks and a set of memories that store the state information of the boundary cells of the group. This information is communicated back to the boundary cells, as the group iterates through the environment as it is displayed in Fig. 6. The number of boundary memory cells and how they are connected to a group depends on the topology of the CA. For the 4-connected topology we need 2 memories of capacity $C_x G_x G_y$ bits and 2 memories of capacity $C_y G_x G_y$ bits. In the 8-connected topology we need 4 additional 1-bit memories with depth $G_x G_y$ that store the state of the diagonal corner cells.

To describe the single-buffering technique, note that the boundary memories act as an information buffer, capable of storing the information that is communicated among groups from time $t-1$ to time t. The state of the boundary cells at time t depends on the information generated by the transition of the neighboring cells during the previous time step. If that information is available during time t for all boundary cells of a group, then we can calculate the transition of a group independently of its environment. The technique consists in breaking up the update process in two distinct steps. The first one simulates a run of the group for the entire environment in order to generate the information at time t and stores the information in the boundaries. Then in the next step, the boundary information from $t-1$ is used to calculate the new states and the results are stored in the cell memories. When all groups have been processed this way, the next state transition may begin (with its two steps). Note that, although we can avoid using double buffering in the main CA array and in the block memories, we still need to use double buffering in the boundary memories, so that we are able to store the states computed in step 1 for time t, while storing the states of time $t-1$ required for step 2.

To quantify the memory usage reduction that we can achieve with this approach, the total memory required for storing the state of the environment amounts to $2G_xC_xG_yC_ys_b$ bits, if we use double buffering. In addition, the number of bits in the boundary memories amounts to $2(G_xG_y)(2C_x + 2C_y)$, assuming 4-connected topology. Therefore, the memory reduction percentage is $\frac{C_xC_ys_b}{2(2C_x+2C_y+C_xC_ys_b)}$. As we can see the percentage does not depend on the environment size in terms of groups, but on the group dimensions. In our case where we used group size of 16×16 and $s_b = 4$, we achieved a memory reduction of about 47%.

3.2.5. Environment Control Module

The environment control module is the functional unit of the architecture that instruments the function of the remaining modules. It is responsible for generating the memory addresses and control signals for the group and the blocks and for performing a series of operations, as for example simulating a single time step, evaluating the termination conditions and so on. The global control for implementing a single time iteration consists of two distinct steps. At the first step, it performs a "simulation run" during which the boundary information are read from one of the memory banks and stored in the second one. At the second step, the boundary memories are switched, so that each group reads the updated information. The second step is the simulation step where the block memories are enabled and the next state of the CA is computed. After the second step, the boundary memories are switched, so that each group reads the updated information in the next update process. The total number of clock cycles for these steps is $2G_xG_y$.

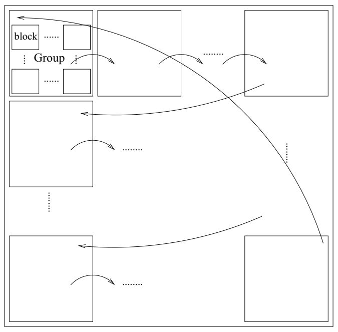


Figure 6. Environment Partition into Groups and Scheduling of Group Evaluation

3.2.6. Communication with the external host

As we mentioned, the FPGA device uses a memory interface to communicate with an external host. This results in the FPGA being actually a memory mapped peripheral, so that the configuration parameters and CA related values are visible as memory addresses. The advantage of this approach is that it allows for reusability of a design, since the "hosting environment" needs only to implement a memory interface. In our case, where the prototyping board that we used for our experiments was based on a PCI interface, communicating with the FPGA design was accomplished by directly writting to memory locations. The only overhead was some "glue logic" that translated the relative offsets of the PCI memory base to the one used by the FPGA.

Table 1. Synthesis Results for the 512×512 Environment

	Slices	RAMB16
Cell	34 ($< 1\%$)	0
Block	234 ($< 1\%$)	1
Group	12743($\approx 25\%$)	64
Environment	13143 ($\approx 26\%$)	136

4. IMPLEMENTATION RESULTS

4.1. Synthesis Results

The device we used was a Xilinx Virtex 4 (XC4VLX100-10FF1513) with a total of 49152 slices and 240 block rams. Table 1 shows the device utilization in terms of slices and block rams for the different modules in hierarhical order. We should note that the number of block rams occupied by the group and environment modules would be doubled if we used a double buffering scheme.

4.2. Experimental Setup

The experiments were carried out using an FPGA Prototyping board equiped with a XC4VLX100-10FF1513 Xilinx device. For our simulations we used mainly two generated FPGA designs, with environment sizes consisting of 256×256 and 512×512 cells, respectively, and having a configurable state M and toric boundaries. In both cases, the block size was 2×2 cells while the group consisted of 8×8 blocks, thus achieving a total parallelism of 256 cell calculations per cycle. Since simulation of each time step consists of two distinct iterations, and therefore, each block has to be evaluated twice in order to obtain the new state, we have an *effective parallelism* of 128 cells per cycle. Finally, the choice between 256×256 or 512×512 environment sizes was made after getting and evaluating the first results of each experiment. Normally, for small values of M we would start with the 256×256 environment size and switch to a larger one when finite size effects became apparent.

4.3. Speed up and Device Metrics

In order to measure the obtained speed-up we used as a benchmark machine an Intel(R) Core(TM)2 Quad CPU, Q9550 running at 2.83GHz. We ran a series of experiments on a 256x256 cells environment using the FiatLux CA simulator ([13]), so that each experiment consisted of 10^4 time steps. To minimize the CPU time required for each time step, we disabled the display of the CA state. From these experiments, we calculated the mean time required for a software implementation to complete 10^4 time steps. The expected simulation time for an array of the same dimensions, consisting of 4x4 block groups and 4x4 cell blocks can be calculated as follows. The effective parallelism of the above design is 128 cells per cycle, meaning that a time iteration of the environment completes in 512 clock cycles. Given that the FPGA device operates at 100MHz, each time iteration completes in 5120ns $\approx$ 5.2us. Since each experiment consisted of 10^4 iterations, the actual time required to complete an experiment is $5.2 \cdot 10^4$us = 52ms. Given that the mean time to complete an experiment in the CPU was $86.5 \cdot 10^3$ms, we obtain a speed-up of approximately 1650.

The actual speed-up figure was much lower than that, since the time taken up by the CPU for the initialisation of each experiment was significant compared to the time required for the FPGA to carry out the actual experiment. Although the exact figure is not straighforward to measure we estimate that the time taken by the CPU to initialize each experiment, wait until the experiment finishes, read back the result and store it to a file, amounted to several multiples of the time required for a single experiment. As a very rough estimate we could say that the resulting speed-up was, because of this overhead, close to one third of the calculated one. A way to improve this would be to move parts of the CPU control logic into the FPGA by integrating, for example, a microcontroller unit capable of running a batch of experiments and storing the results in the FPGA, for the host to read them back. We are planning on moving towards this direction in our future implementations.

5. DATA ANALYSIS

5.1. Experimental Procedure

Recall that our goal is to identify the critical threshold p_{T_C} for each value of M and each topology. Initially, using inspection on a small number of custom experiments, we determined an interval for the value of p_T, around which the phase transition occurred, with an initial precision of roughly 2×10^{-2}. Dividing this interval into 10 equally spaced subintervals, we measured the probability of survival versus time for up to t_{max} time steps. On the following iterations we further refined the search interval so as to obtain a precision equal to approximately 10^{-4}. We here display only the p_T-time plots that are close to the critical threshold.

5.2. Setup and Results

In this section we describe the procedure that we followed to measure the critical exponent from the experimental data, and we shortly comment on our results. We further outline the open questions that arise, and that will form the basis for our future work. Note that due to the large number of experimental results, we will only present here a subset of the obtained graphs. A larger set of graphs can be found the AMYBIA project website[1].

5.2.1. Identifying the Phase Transition

To locate p_{T_C}, we ploted the measured probability $P_s(t)$ as a function of time for the different values of p_T. Since we expect that near the phase transition, the observed quantity will follow a power-law [12], the plots were drawn in a log-log scale, where the critical case should appear as a straight line. The critical probability is then observed at the point where $P_s(t)$ exhibits a curvature change. This curvature change can be interpreted as a transition from almost certain extinction of the reaction-diffusion process to non-zero probability of survival, so that the values of p_T above the critical value p_{T_C} correspond to survival of the process, while values below the critical lead to extinction. The behavior we described can be observed in Fig. 7, where the curve $f(t) = Kt^{-\delta}$ for the critical exponent value $\delta = 0.451$ ([8]) is shown for reference.

Finally, to determine the maximum number of time steps that each experiment should run, t_{max}, we ran a series of preliminary experiments. As we can see in Fig. 7, the distance of the probability curves from the straight line characterizing the critical threshold, $\Delta p = |p_T - p_{T_C}|$, increases with time. Hence, the closer one gets to the critical threshold, the longer the simulation time must be to discriminate between subcritical and supracritical conditions. This calls for setting the value of t_{max} as large as possible, in order to increase the precision in which we want to determine p_{T_C}. On the other hand, an increase in t_{max} results in an increase of the running time of the experiments. Finally, since in our case we are using toric boundaries, longer run times increase the probability that travelling reaction-diffusion "waves" will cover distances larger than the grid dimensions and, therefore, increases the probability that two or more "waves", travelling in different or even perpendicular directions will collide and probably annihilate. Note that the term reaction-diffusion "wave" is used in a rather relaxed

[1]`http://webloria.loria.fr/~fates/Amybia/readdiff.html`

manner to describe "clusters" of excited cells that persist in time and move randomly in the environment. Taking these factors into account, the value of $t_{\max}$ has been determined empirically to be 10^4 time steps.

5.2.2. 4-connected Topology Results

In the case of 4-connected neighborhood, we ran a series of experiments for values of M ranging from 2 to 15. All of these experiments were carried out using an environment of size 512×512. The number of individual experiments for each one of the last set of iterations ranged from 4×10^4 to 6×10^4, while the total number of experiments that have been run in all iterations was approximately 6×10^6. Figure 7 shows the obtained experimental curves for the case of 4-connected topology, with $M = 14$ and various values for p_T. The curves for $p_T \geq 0.50362$ have a clear tendency to bend upwards at long times, while those for $p_T \leq 0.50330$ tend to bend downwards. The curve for $p_T \geq 0.50346$ is almost linear (at long times) in this log-log plot (though it still bends down around the end of the simulation). As confirmed by good agreement with a power law behavior with exponent -0.451, we take the latter value as an estimate for the critical threshold p_{T_C} at $M = 14$. We used the same method for other values of M and report the corresponding p_{T_C} values in Fig. 8.

First of all, we note that these values are in general agreement with the preliminary results that we reported in [8]. Notwithstanding, the estimates reported in [8] were based on a different technique (namely the decay of the density of excited cells, starting from random initial conditions) and, being obtained through software simulations, displayed much lower statistical quality and concerned a much narrower range of M-values. Furthermore, as we can readily observe, the critical exponent values for consecutive even and odd values of M are equal, at least within the precision of our experiments. This phenomenon has already been noticed in [8], and is further confirmed by the present improved simulations. One last thing to note is that, as we can observe from Fig. 8, the values of p_{T_C} seem to tend to an asymptotic value of around 0.51 at large values of M.

5.2.3. 8-connected Topology Results

For the 8-connected topology, we conducted a series of experiments with the value of M ranging from 2 to 14. For values of M up to 5, the experiments for the 8-connected topology were conducted using a 256×256 grid size, while for the remaining values of M we used 512×512 environment. The total number of experiments that were conducted, during all iterations were approximately 2×10^6. Figure 9 shows the results obtained for $M = 14$. Using the same method as above, we locate the critical threshold at a value around 0.2622, with the same statistical quality as for

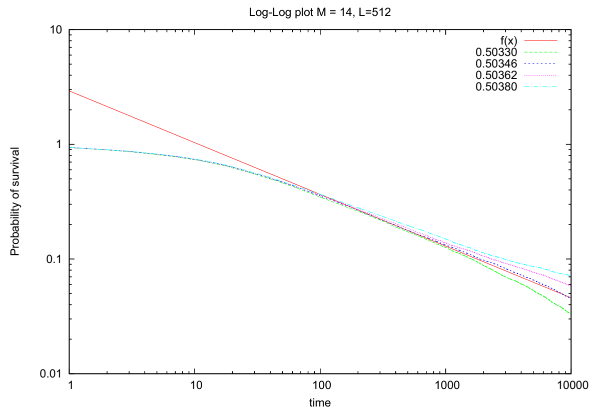


Figure 7. Plot for M=14, 4-connected Topology

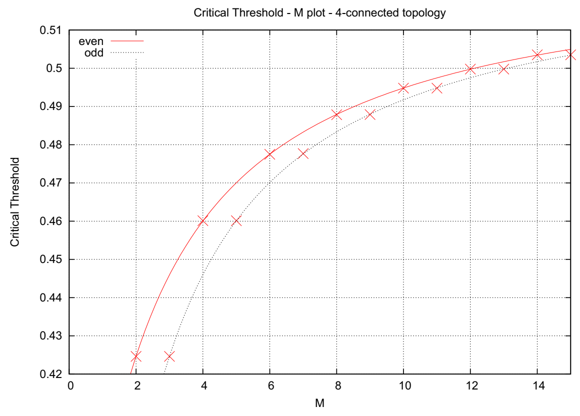


Figure 8. p_{T_C} Dependence on the Number of States M

the 4-connected topology above. The evolution of p_{T_C} with M is shown in Figure 10.
As in the 4-connected case above, the reported data indicates an asymptotic value at large M values situated around 0.25. However, in opposition to the 4-connected case above, consecutive even-odd values of M exhibit different values of p_{T_C}.

5.2.4. Non-toric boundary conditions

Before moving on to the conclusions, we would like to give some insight on the effects of using different boundary conditions. Regarding the hardware complexity, the free boundaries are the simplest ones and can be implemented by using a set of multiplexers for the neighborhood input values. The multiplexers are controlled by a set of comparators that select the value zero when the group position lies on the boundary of the environment. Toric boundaries are sligthly more complex. In this case no multiplexers are required at the neighborhood inputs, but the address genera-

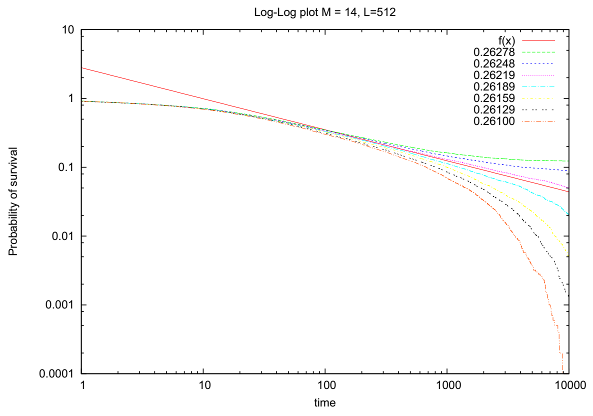


Figure 9. Plot for M=14, 8-connected Topology

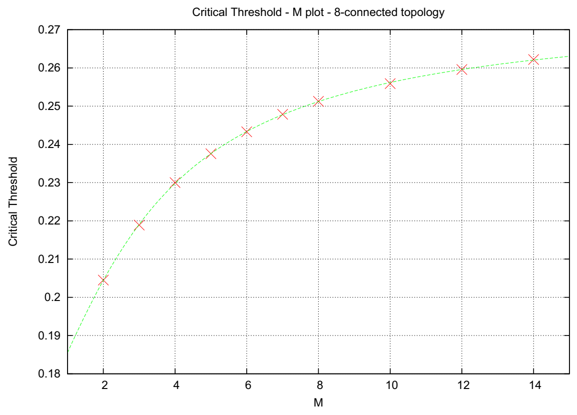


Figure 10. p_{T_C} Dependence on the Number of States M

tion circuit for the boundary memories becomes more complicated. Adiabatic boundaries require the use of "virtual" cells. Their implementation is similar to the toric case.

Regarding the boundary condition effects on the evolution of the reaction-diffusion process, our experiments have shown that they tend to appear more rapidly for the case of free boundaries and manifestate as a second curvature change on a double logarithmic probability-time plot. In the case of toric boundaries, the effects appear for larger values of M, i.e. $M = 6$ on a 256×256 grid. Increasing the grid size generally minimizes these effects, although from our observations, they tend appear more rapidly in the case of free boundaries, as M continues to increase.

6. CONCLUSIONS

The reported experimental results further strengthen the results of [8] and the quality of the statistics in these improved simulations open new and interesting research questions. First, our results confirm the probable existence of an asymptotic value for the critical threshold when the number of states increases. The fact that the values obtained for the 4-connected topology are roughly twice those for the 8-connected topology is easily understood. Indeed, a mean-field approach to the GHCA, validated by several experimental measurements, predicts that the critical threshold decays as the inverse of the average number of neighbors per node in the lattice [8].

The value of the asymptotic limit (around 0.51) is however much more delicate to understand. At the $M \rightarrow \infty$ limit, a cell needs infinite time once activated, to get back to the neutral state $\sigma = 0$. Thus in effect, a cell can only be activated once in a finite time simulation, and once activated it remains insensible to its neighborhood. In this case, the GHCA model reduces to the so-called SIR model with perfect immunization (also called general epidemic process) [9], for which the critical transmission threshold is exactly 0.5 on a 2D square lattice [10]. Using this argument, one would expect an asymptotic value of 0.5 for p_{T_C} in the GHCA with 4-connected topology and not 0.51. The situation is however more complex. Indeed, the SIR is in the universality class of dynamical percolation (DyP), not directed percolation (DP) as for the GHCA. The two models are thus expected to display different critical behavior. For instance the value of the critical exponent δ for DyP is much lower than its value in DP (0.092 versus 0.451) [9]. Hence, when M increases, the determination of p_{T_C} is complicated by the fact that, at some point, the universality class is expected to switch from DP to DyP. Future studies will be needed to understand this crossover at large M values. This issue is actually particularly challenging as the determination of the threshold and the universality class for still larger M values will demand that our experimental platform be extended so as to handle larger environments and higher levels of parallelism.

Another open problem is to explain the even-odd equality of $p_{T_C}(M)$ in the 4-connected case. The reason of this behavior is currently unknown. As our experimental results have shown, this behavior is not observed in the 8-connected case. It is interesting therefore to identify which case, the 4-connected or the 8-connected one, is the regular one. To this end, it would be interesting to investigate other topologies as well, in order to identify if this phenomenon is restricted to the 4-neighbors topology.

From the implementation point of view, future studies will consider methods for expanding the FPGA environments so as to measure more complex quantities, as well as to handle a larger variety of topologies. Finally, we will investigate methods to further increase the simulated environment sizes, as well as to increase the parallelism and decrease simulation times, given the size constraints of FPGA devices.

ACKNOWLEDGEMENTS

This research was funded by an INRIA grant to project AMYBIA.

REFERENCES

[1] A. M. Turing, "The chemical basis of morphogenesis," *Philosophical Transactions of the Royal Society of London*, vol. B 237, pp. 37–72, 1952.

[2] I. Kusch and M. Markus, "Mollusc shell pigmentation: Cellular automaton simulations and evidence for undecidability," *Journal of Theoretical Biology*, vol. 178, no. 3, pp. 333 – 340, 1996.

[3] D. Makowiec, "Cellular automata model of cardiac pacemaker," *Acta Physica Polonica B*, vol. 39, no. 5, pp. 1067–1085, 2008.

[4] A. Adamatzky, B. De Lacy Costello, and T. Asai, *Reaction-Diffusion Computers*. Elsevier, 2005.

[5] J. M. Greenberg, B. D. Hassard, and S. P. Hastings, "Pattern formation and periodic structures in systems modeled by reaction-diffusion equations," *Bulletin of the American Methematical Society*, vol. 84, no. 6, pp. 1296–1327, 1978.

[6] R. Fisch, J. Gravner, and D. Griffeath, "Metastability in the Greenberg-Hastings model," *Ann. Appl. Probab.*, vol. 3, no. 4, pp. 935–967, 1993.

[7] N. Fatès, "Solving the decentralised gathering problem with a reaction-diffusion-chemotaxis scheme - social amoebae as a source of inspiration," *Swarm Intelligence*, 2010. To appear. Preliminary version in "Gathering Agents on a Lattice by Coupling Reaction-Diffusion and Chemotaxis", `http://hal.inria.fr/inria-00132266/`.

[8] N. Fatès and H. Berry, "Robustness of the critical behaviour in a discrete stochastic reaction-diffusion medium," in *Proceedings of IWNC 2009* (F. Peper, H. Umeo, N. Matsui, and T. Isokawa, eds.), vol. 2 of *PICT*, pp. 141–148, Springer, 2010.

[9] S. M. Dammer and H. Hinrichsen, "Spreading with immunization in high dimensions," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2004, no. 07, p. P07011, 2004.

[10] P. Grassberger, "On the critical behavior of the general epidemic process and dynamical percolation," *Mathematical Biosciences*, vol. 63, no. 2, pp. 157 – 172, 1983.

[11] N. Fatès, "Asynchronism induces second order phase transitions in elementary cellular automata," *Journal of Cellular Automata*, vol. 4, no. 1, pp. 21–38, 2009.

[12] H. Hinrichsen, "Nonequilibrium critical phenomena and phase transitions into absorbing states," *Advances in Physics*, vol. 49, pp. 815–958, 2000.

[13] N. Fatès, "FiatLux CA simulator in Java." See `http://webloria.loria.fr/~fates/`.

[14] B. Girau and C. Torres-Huitzel, "Fast implementation of a bio-inspired model for decentralized gathering," in *RECONFIG '08: Proceedings of the 2008 International Conference on Reconfigurable Computing and FPGAs*, (Washington, DC, USA), pp. 229–234, IEEE Computer Society, 2008.

[15] B. Girau, C. Torres-Huitzil, N. Vlassopoulos, and J. H. Baron-Zambrano, "Reaction-diffusion and chemotaxis for decentralized gathering on FPGAs," *International Journal of Reconfigurable Computing*, vol. article in press, 2010.

[16] B. Girau, A. Boumaza, B. Scherrer, and C. Torres Huitzil, "Block-synchronous harmonic control for scalable trajectory planning," in *Robotics, Automation and Control* (A. Lazinica, ed.), pp. 85–110, I-Tech Publications, 2008. ISBN : 978-953-7619-20-6.

[17] M. George and P. Alfke, "Xapp210: Linear feedback shift registers in Virtex devices." `http://www.xilinx.com/support/documentation/application_notes/xapp210.pdf`.