Research Article

Reaction Diffusion and Chemotaxis for Decentralized Gathering on FPGAs

**Bernard Girau,¹ César Torres-Huitzil,² Nikolaos Vlassopoulos,³
and José Hugo Barrón-Zambrano²**

¹LORIA/University Nancy 1, Cortex Group, Campus Scientifique, Vandoeuvre-les-Nancy, France

²Cinvestav Tamaulipas, Information Technology Laboratory, Victoria, Mexico

³INRIA Nancy Grand Est, Maia group, Villers-les-Nancy, France

Correspondence should be addressed to Bernard Girau, bernard.girau@loria.fr

Received 15 March 2009; Accepted 16 November 2009

Recommended by Lionel Torres

We consider here the feasibility of gathering multiple computational resources by means of decentralized and simple local rules. We study such decentralized gathering by means of a stochastic model inspired from biology: the aggregation of the *Dictyostelium discoideum* cellular slime mold. The environment transmits information according to a reaction-diffusion mechanism and the agents move by following excitation fronts. Despite its simplicity this model exhibits interesting properties of self-organization and robustness to obstacles. We first describe the FPGA implementation of the environment alone, to perform large scale and rapid simulations of the complex dynamics of this reaction-diffusion model. Then we describe the FPGA implementation of the environment together with the agents, to study the major challenges that must be solved when designing a fast embedded implementation of the decentralized gathering model. We analyze the results according to the different goals of these hardware implementations.

Copyright © 2009 Bernard Girau et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. Introduction

Spatial computing is a large research field where researchers try to propose alternative computing devices that consist of a (huge) amount of computational resources that are spread across some physical structure. This research field includes many different domains, such as biological computation, robot swarms [1] and swarm intelligence [2], amorphous computing [3], and reconfigurable computing. The common constraints are that the communication cost between different computational resources strongly increases with their distance, and that the global functionality emerges from the collective behavior of the resources. In this research field, the main algorithmic question may be summarized as “how to make computing units cooperate to solve a given task?” [4]. Our project (*Amybia* INRIA collaborative research project, led by Nazim Fatès) takes place in this approach, by considering an upstream question, “how to gather or assign enough computing resources to solve a given task?”, in a context where faulty units may appear.

More precisely, we consider here the problem of gathering computing units in a strongly constrained context: (1) units use only local rules, (2) units move on a lattice and need to gather to form a compact cluster, (3) units have no idea of their own position and of the position of the other units, (4) units may only send messages that can be relayed (possibly with errors) by the cells of the lattice, (5) units only perceive the state of the neighboring cells, and (6) the only action units may undertake is to move to these cells or change the state of these cells. The possible applications of this problem to several problems of spatial computing still need to be deepened: we discuss them in Section 7. In this paper, our ambition is only to show that a simple model is able to achieve decentralized gathering, while being suitable for efficient distributed implementations. Our approach is inspired by biology, where such decentralized gathering is observed, so as to derive a model and its implementation.

The cellular slime mold *Dictyostelium discoideum* is a fascinating living organism that has the ability to live as a monocellular organism (amoeba) and to transform

into a multicellular organism when needed. In normal conditions, the amoebae live as single individuals. However, when the environment becomes depleted of food, a gathering phenomenon is triggered and single amoebae aggregate to form a complex organism that moves and reacts with coordinated changes. In the Amybia project, we take inspiration from the first stage of the multicellular organization process, the aggregation stage, which consists in gathering all the amoebae in a compact mass called a mound [5].

In [6, 7], Fatès proposes a simplified model of *Dictyostelium discoideum* that exhibits the main behavioral properties of the aggregation mechanism: reaction-diffusion and chemotaxis. Our Amybia project is built around this model. It uses a cellular automaton to describe the environment, and a multiagent approach to model the amoebae. This paper focuses on the hardware aspects of our project. It is roughly divided into two parts. In the first part, the FPGA is used as an accelerator for simulations of the dynamics of the environment, especially close to phase transitions. In the second part, beyond being an accelerator, the FPGA is also considered as a representative device for massively distributed computation so as to study the main issues that may appear while using reaction-diffusion chemotaxis for decentralized gathering within heterogeneous spatial computing devices.

Section 2 focuses on the biological inspiration of our work, describing the aggregation process that is observed with *Dictyostelium discoideum*. The model of [6, 7] consists of an environmental layer and a particle layer. The environmental layer and its properties are described in Section 3, while its hardware implementation for fast simulations is depicted in Section 4. Then Section 5 summarizes the particle layer model and its properties before Section 6 describes its FPGA implementation that performs decentralized gathering on-chip. Section 7 shortly discusses possible contexts of use for such decentralized gathering. Finally, we derive conclusions about the main obstacles and possible modifications of our approach.

2. Decentralized Gathering and Bioinspiration

Decentralized algorithms to gather robots to form circular or polygonal shapes have been proposed in [8], where all robots “see” the positions of each other. Similar problems with a limited visibility range have been studied in [1, 9]. We refer to the work of [10] for recent developments on the decentralized gathering problem. In this paper, we get rid of any assumption on the visibility range using an environment that transmits messages on arbitrarily long distances. The decentralized gathering problem is also related to the Leader Election problem, but our goal is not to select one unit among many, but to gather randomly located units in a compact location that emerges by consensus. This behavior is part of the complex and unusual life cycle of the cellular slime mold *Dictyostelium discoideum*; it corresponds to the aggregation step of the multiple monocellular organisms that gives birth to a single multicellular organism.

FIGURE 1: Some steps of the life cycle of *Dictyostelium discoideum* (Copyright, M.J. Grimson & R.L. Blanton, Biological Sciences Electron Microscopy Laboratory, Texas Tech University).

2.1. *Dictyostelium discoideum*. Despite the biological inspiration of our work, we do not pretend to provide the reader with accurate and complex biological notions. Some biological terms may even be used with some approximation, which does not penalize our work, since we model a behavior and we do not model biological species. Therefore this section only gives an overview of the specificities of *Dictyostelium discoideum*, focusing on its aggregation properties that inspire the model in [6, 7].

2.1.1. Life Cycle. *Dictyostelium discoideum* amoebae grow as independent cells in natural environments such as moist, decaying wood. In normal conditions, they behave as monocellular organisms, but they are able to interact when a coordinated reaction to extreme conditions is required. Extreme conditions may correspond to a food-depleted environment that might result in starvation for the population of amoebae. By means of their interactions, single cells do not only join to perform a collective reaction, they join to generate a multicellular organism (containing thousands of cells) that is able to better react to extreme conditions than the population of individual cells.

As illustrated by Figure 1, after having grouped together, the population becomes able of cell differentiation, which results in several steps of a life cycle that adapts to the environmental conditions. The mound of cells that results from aggregation is then transformed into some elongated migrating slug, and then into a fruiting body. We are only interested in the process by which amoebae group together, since it fulfills the different constraints for the decentralized gathering of computing units we study.

2.1.2. Aggregation. In vitro experiments show that the aggregation phenomenon of *Dictyostelium* is triggered by one or several amoebae that attract other amoebae that are located in their vicinity to form groups. The first groups merge until only a few clusters remain; these will attract other amoebae to them to form a cluster where cellular differentiations occur to lead to the multicellular organism.

Attraction is led by the transmission of waves of chemical messengers, which follow typical evolving reaction-diffusion

FIGURE 2: Cycle of states for a cell of the environmental layer.

patterns. The chemical messengers are internally produced by the amoebae. When an amoeba detects a high increase in the external concentration of the messengers, it follows the concentration gradient (this phenomenon is called chemotaxis) and it releases its own internal messengers. Then it becomes insensitive to the messengers during a given refractory period, and in the meanwhile, the released messengers diffuse and attract other sensitive amoebae.

2.2. Previous Models. Several models have been proposed to study the dynamics of Dictyostelium (see a review in [11]). Many of them are based on partial differential equations [12, 13]. Some studies aim at being very close to the biological inspiration, comparing simulation outputs with observations of the aggregation of Dictyostelium [14], or modeling the receptors of the chemical messengers [15]. Most studies use continuous or hybrid models; to our knowledge, the model in [6, 7] that founds our Amybia project is the first fully discrete model that captures Dictyostelium's behavior. By fully discrete, we mean that time and space are discrete and the state of the amoebae is described in qualitative terms rather than quantitative (integers or decimal values). This discretization is useful when digital hardware implementations are expected. The reaction-diffusion mechanism alone is well understood, with explicit links between the discrete and continuous models (e.g., [16]). This mechanism shows problem-solving abilities [17]. In our project, we use the model of Fatès [6, 7] that adds virtual chemotaxis as a new feature to study and use. Two layers compose it: the environmental layer is a cellular automaton that models a reaction-diffusion process while the particle layer describes the moves of virtual amoebae.

3. The Environmental Layer

As explained before, attraction of amoebae is led by the transmission of waves of chemical messengers in the environment. In this section, we only consider this reaction-diffusion process. The study of the qualitative behavior of the environmental layer is an important part of the Amybia project. We aim at characterizing this behavior in terms of complex system dynamics, and we study its robustness to noise and obstacles. Therefore we assume here that waves of excitation are initiated at randomly chosen positions, and then we observe how these waves behave in the long term.

Next subsection defines the discrete model of this environmental layer [6, 7], that mostly depends on one parameter called the transmission rate. Then Section 3.2 summarizes the main results about the dynamics of this

reaction-diffusion process, where phase transitions depend on this transmission rate. The goal of the implementation described in the next section is to perform large scale simulations of those phase transitions, and to extend these results to various topologies and perturbations.

3.1. Discrete Model. Space is modelled by a regular lattice $\mathcal{L} = \{1, \dots, X\} \times \{1, \dots, Y\}$ in which each cell $c = (c_x, c_y) \in \mathcal{L}$ is associated to a state. The set of possible states for each cell is $\{0, \dots, M\}$, the state of cell c at time t is denoted by σ_c^t . State 0 is the *neutral* state, state M is the *excited* state.

A “source” cell is an initially excited cell. Any cell may evolve from the neutral state to the excited state if at least one of its neighbors is excited (rule (R1)). To model the uncertainty on this transition, we consider that it happens with a given probability p_T , called the *transmission rate*. States 1 to $M-1$ are the *refractory* states. A cell in a refractory state evolves in an autonomous way by decrementing its state by 1 (rule (R2)) until it reaches the neutral state. A neutral cell surrounded by neutral cells stays neutral (rule (R3)). Figure 2 illustrates the different possible states of the cell.

To express these rules without ambiguity, for a cell $c \in \mathcal{L}$, let us denote by N_c the neighborhood of this cell. Let E_c^t be the set of excited cells in the neighborhood of c at time t : $E_c^t = \{c' \in N_c \mid \sigma_{c'}^t = M\}$. We also denote by $|S|$ the cardinal of a set S .

With these notations, for a time $t \in \mathbb{N}$ and a cell $c \in \mathcal{L}$, let $\mathcal{B}(p)$ be a Bernoulli random variable that equals 1 with probability p and equals 0 with probability $1 - p$. The local rule governing the evolution of the environment is

$$\sigma_c^{t+1} = M \quad \text{if } \sigma_c^t = 0, |E_c^t| > 0, \mathcal{B}(p_T) = 1, \quad (\text{R1})$$

$$\sigma_c^{t+1} = \sigma_c^t - 1 \quad \text{if } \sigma_c^t \in \{1, \dots, M\}, \quad (\text{R2})$$

$$\sigma_c^{t+1} = 0 \quad \text{otherwise.} \quad (\text{R3})$$

A set of adjacent cells that are all in the excited state M is called an *excitation front*. In Section 5, we explain how the excitation fronts guide the amoebae that move on the lattice (chemotaxis).

3.2. Properties. The main properties of this model are presented in [6, 7]. Since this paper focuses on the hardware implementation issues, we only summarize the main results below.

The dynamics of the environment depends on two parameters: the excitation level M and the transmission rate p_T . The study of [6, 7] shows that different qualitative behaviors may be observed: the static regime, the non-coherent regime, and the extinction regime.

FIGURE 3: Two aspects of the noncoherent regime (a): $p_T = 0.95$, (b): $p_T = 0.25$).

3.2.1. Static Regime. This regime is obtained in the case of systematic transmission of waves ($p_T = 1$): the excitation fronts collide systematically and they annihilate themselves. This phenomenon is well known for reaction-diffusion processes.

3.2.2. Noncoherent Regime. This regime may be observed in the case of nonperfect transmission conditions ($p_T < 1$); the reaction-diffusion waves are independent from the position of the source cells and no organization can occur. Figure 3 illustrates the influence of p_T on the transmission of waves, in a 50×50 environment with $M = 4$ and a Moore neighborhood. Black pixels denote neutral cells, and red pixels stand for excited cells, while shaded colors are used for refractory states. Reaction-diffusion waves remain visible with $p_T = 0.95$; whereas they appear unorganized with $p_T = 0.25$.

3.2.3. Extinction Regime. This regime is attained when the transmission rate is less than a critical value ($p_T < 0.23$ for $M = 3$). In that case, waves spontaneously disappear.

Following well-known studies in statistical physics, the first experiments depicted in [7, 18] indicate that the universality class of the phase transition from the non-coherent regime to the extinction regime might be *directed percolation* [19, 20]. The robustness properties of the model strongly depend on the universality class of its phase transitions. These experiments need to be extended to larger environments, but software simulations are very time-consuming. Therefore we have developed a block-synchronous hardware implementation to handle large-scale simulations.

4. Fast FPGA Simulation of Phase Transitions

The hardware part of the Amybia project is motivated by two main goals. The first one is to develop fast implementations to explore complex dynamics in large-scale environments. The corresponding implementation work is described in this section. The second goal is to perform a preliminary study of the ability of our model to provide an efficient decentralized gathering process for a large amount of distributed computing units. The corresponding implementation is the subject of Section 6.

4.1. Block-Synchronous Implementation. The behavioral description of each environment cell may reduce to a very simple state machine that could be implemented with very few hardware resources. Nevertheless, the most area-greedy computation in the environment layer is not the state transition, but the generation of the Bernoulli law with probability p_T for each cell. As a consequence, a fully parallel implementation of the environmental layer would not be able to implement environment sizes that are out of reach for software simulations (a few thousands of cells at most, see below for the implementation area of the random generators). Therefore, we have chosen to use a block-synchronous (or block-parallel) implementation based on the embedded B-RAM memories (Block RAM) of the FPGA, as in [21]. The environment is partitioned into several blocks, with each block of cells being handled in a fully parallel way by the FPGA while the different blocks are sequentially handled. Let $X \times Y$ be the total number of cells in the environment layer. Let $n \times m$ be the number of cells that may be simultaneously handled on the FPGA. The environment is partitioned into $\mathcal{B} = (X/n) \times (Y/m)$ blocks of $n \times m$ cells.

Let us consider a cell in the environment. It is located at relative position (x, y) in block (α, β) , so that its coordinates in the whole environment are $(\alpha n + x, \beta m + y)$. We store its state in a local B-RAM memory, with an address that corresponds to the block number. The local position of the B-RAM memory is sufficient to stand for the (x, y) coordinates, so that they do not appear in the address. The computation of the block is performed by using the same block-dependent address for all local B-RAM memories, thus handling all cells in this block. Then the computations are performed for the next blocks by increasing the common address used for all B-RAM memories. It should be pointed out that the choice of B-RAMs to store cell states in this implementation is only related to the need to store many states at the same location in the FPGA; whereas the cell states will be stored in simple elementary flip-flops when considering the fully parallel implementation of the model with amoebae in Section 6.1.

Figure 4 illustrates the decomposition of the environment into blocks and the block-synchronous scheduling of the computation. The different blocks are shown, each one containing an outlined cell at relative position (x, y) : the

FIGURE 4: Block partitioning of the environmental layer.

states of all these outlined cells are stored in the same B-RAM memory of the FPGA. All cells are handled simultaneously in a given block, and the red arrows denote the cyclic block-scheduling of the computation.

4.2. General Architecture. Figure 5 schematizes the general architecture of our implementation of the environmental layer. Since the environment is split into several blocks, this architecture mostly consists of a grid of $n \times m$ identical cell modules (gathered as groups of 4 or 6 cells using the same B-RAM memory to handle on-chip data storage and access) surrounded by border modules. An additional control module computes the memory addresses that are used by all modules (block-scheduling) and computes the number of excited cells in the environment. Figure 5 only illustrates a simple 4-neighborhood, but our implementation handles the 8-neighborhood. The role of each component is as follows.

4.2.1. Cell Module. Each cell module updates the state of its corresponding cell within the currently handled block. More precisely, we use a bufferized storage of the states of all cells so as to synchronize the computations of all blocks: a most significant bit δ or $1 - \delta$ is added to the addresses that are sent to the dual-port B-RAM memories; the current states are read with δ , whereas updated states are stored with $1 - \delta$. When the current iteration of the equations of rules (R1), (R2), and (R3) has been performed for all blocks, buffers are exchanged by means of $\delta \leftarrow 1 - \delta$.

Depending on the value of M , the cell modules are split in groups of 2×2 cells ($M < 16$) or 2×3 cells ($M < 8$) that share common storage resources: a single 18 Kbit dual-port B-RAM memory stores the states of 4 or 6 cells for all blocks of cells, using 18-bit words.

4.2.2. Border Module. The border modules are simpler than the cell modules. They only store one bit for each one of the immediate neighbors of the most outer cells within each block; this bit stands for the cell being excited or not. The only difficulty is to handle the addressing scheme so that the information stored within each of the 4 possible borders is updated when the block that contains the corresponding cells is being handled. This update requires long-range connections from the cell modules on each side of the block to the opposite border modules. Moreover, when the borders lie outside the whole environmental layer, the border modules simply generate the constant value 0 (not excited).

4.2.3. Control Module. The control module uses a 10-bit counter to perform block-scheduling, and a 16-bit counter to handle iterations on the environment. Moreover, our goal is to study the phase transition between the non-coherent regime and the extinction regime. Therefore, the control module computes the number of excited cells within each row of cells, it adds these numbers for all rows, and then it accumulates the results for all blocks. Nevertheless, it is sufficient to detect if the number of excited cells tends to zero, so that all numbers are computed up to 64, which reduces the cost of the adders.

4.3. Implementation of a Cell. A cell module mostly consists of two parts: a random number generator (RNG) to compute the Bernoulli random variable $\mathcal{B}(p_T)$, and the cell state update.

4.3.1. Generating the Bernoulli Law. In the software implementation developed by Fatès, the same RNG is used for all cells, thanks to the assumed independence of the successively generated numbers. It should be pointed out that generating high-quality random variables to ensure a real

FIGURE 5: General architecture (environment).

independence of successive random numbers still remains a research subject. Nevertheless, this issue of software RNGs does not appear as relevant for our model, where the quality of usual RNGs is sufficient to break the symmetry of wave transmissions (see [6, 7]). But in a parallel hardware implementation, all cell modules must generate their own random variable in parallel. Therefore we have to implement $n \times m$ RNGs. The precision of the random processes is particularly important when studying phase transitions. Moreover, the spatial independence that is required for symmetry breaking implies that the hardware RNGs we use must be sufficiently good (long period, and independent seeds). This induces an important cost in space for the random aspects of the environmental layer. This hardware resource cost is the counterpart of the computation time that is mostly spent in generating random numbers in the software implementation.

Our choices for the implementation of the random processes have been carefully studied. Most digital hardware solutions are based on LFSR or cellular automata (CA) [22]. Another approach takes advantage of large numbers stored in parallel [23]. Since the LUTs of the FPGA logic cells may be efficiently configured as synchronous RAMs standing for shift registers [24], we use LFSR-based RNGs. See Section 8

for an extension of our work to spatially distributed and mutualized CA-based RNGs.

Experiments in [6, 7] show that the transmission rate needs to be taken into account with a rather high precision (more than 16 bits). Taking into account this precision and the need for random bitstreams as aperiodical as possible, we use a 168-bit RNG adapted from [24] (a similar 63-bit RNG will be depicted later in Figure 14), comparing at least 16 of its generated bits to p_T so as to output 0 or 1 as $\mathcal{B}(p_T)$. To ensure spatial independence, all RNGs use different seeds (set during initialization through on-chip registers).

4.3.2. Updating the Cell States. Figure 6 shows the simplified architecture of a cell module. The current state of the cell is read in the local B-RAM memory. It is compared to M so as to send to the neighboring cells a signal that is equal to 1 if the local cell is in the excited state. The current state is decreased (if excited or refractory) by the “state decrease” module, while a large AND gate outputs 1 if the state is neutral and if there is at least one excited neighbor and if the Bernoulli law generator currently outputs 1. A final multiplexer chooses between M and the computed decreased state according to the output of the AND gate. The resulting value is written in

TABLE 1: Synthesis results for a single cell.

Single cell hardware resource utilization Xilinx FPGA XC4VLX160ff1513-12	
Number of Slice Flip Flops	24/135,168
Number of 4 input LUTs	53/135,168
Number of occupied Slices	44/67,584
Max Frequency	143 MHz

TABLE 2: Synthesis results for a block of cells.

32 × 32 block hardware resource utilization (16 × 16 groups of 2 × 2 cells) Xilinx FPGA XC4VLX160ff1513-12	
Number of Slice Flip Flops	25,397/135,168 (18.79%)
Number of 4 input LUTs	54,315/135,168 (40.18%)
Number of occupied Slices	39,728/67,584 (58.78%)
Frequency	100 MHz

the local B-RAM memory. It must be noticed that the storage of the states in the B-RAM memories makes it impossible to implement the cell as a simple finite state machine (unlike the implementation in Section 6).

4.4. Performance

4.4.1. Implementation Results. The prototyping platform is a PCI-based board (DN8000K10PCI) with three virtex-4 family FPGAs. For experimental results, the FPGA implementation of the model is only targeted towards the XC4VLX160ff1513-12 device of this board. This FPGA has a capacity of 135,160 logic cells, and it contains 288 embedded 18 Kbit B-RAM memories. The design was synthesized, placed, and routed with the Xilinx Foundation ISE 9.2i tool suite. According to the reported synthesis results in Table 1, a compact implementation was obtained since a single cell requires 44 slices. It is important to point out that these resources take into account the implementation of the 168-bit RNG adapted from [24] which was efficiently implemented as a LFSR using FPGA shift register LUT primitives.

As summarized in Table 2, a block of 16 × 16 groups of 4 cells only requires around 59% of the total logic resources available in the FPGA device, taking advantage of the optimization of the slices that are partially used by a single cell. The size of the grid, 1024 cells, is limited by the amount of embedded distributed B-RAM memories in the FPGA. For this grid size, 256 B-RAM memories are used since the 4-bit states (we consider here the case $M < 16$) for the 4 adjacent cells of a group in a block are stored in the same memory.

In order to achieve large-scale efficient simulations, larger grid sizes are desirable, corresponding to interesting experimental environments. Therefore, this block implementation is used as the basic computational unit for each part of the partitioned environment. Only 8 additional B-RAM

memories are required to store the excitation states of the $2 \times (32 + 32)$ border cells in the border modules. Therefore, 264 out of the 288 B-RAM memories of the XC4VLX160 are used. Finally, the module that controls the computation scheduling of all blocks and that accumulates the number of excited cells found in each block uses less than 2% of the logic resources, so that the whole architecture uses 60,5% of the FPGA resources.

4.4.2. Fast Large-Scale Simulations. The embedded B-RAM memories are able to store the states of 512 groups of 4 cells (with state buffering). Therefore we implement a total size of 512×1024 cells for the environmental layer. Despite this large size, we still have to face border effects when studying phase transitions in these environments. Therefore, we take advantage of the methodology inspired by [25] so as to study phase transitions only at the limit of the stable state (where all cells are neutral): all cell states are initially set to neutral, except the central cell of the central block, that is initially excited.

We estimate here the simulation speedup of our FPGA implementation with respect to the software simulation tool developed by Fatès. These estimations should be considered with great caution, since the software tool and the hardware implementation are difficult to compare: this software is a not-optimized version written in Java with jdk 1.6; moreover, the hardware and software computations are not fully equivalent (considering the way random numbers are generated). Therefore, we consider that the computed speedup should only be interpreted in terms of order of magnitude. It should be noted that unlike the widely spread idea that Java is slow, recent benchmarks show that Java 1.6 easily compete with C, C#, or C++. Yet, it is not possible to extrapolate this comparison to a software with cache optimization or similar improvements, for which the performance improvements might be great, but highly dependent on the application. For a 512×1024 environmental layer, Java-based software simulations on a microprocessor-based computer, Pentium 4.2 GHz, require 0.5 s per evolution step, resulting in very long experiments (thousands of iterations are required for each run, and thousands of runs are required to reach significant statistical results for each value of p_T). The computation time mostly lies in both the generation of the random values and the cache management, because of the huge number of cells.

With the above FPGA implementation, each iteration lasts 512 clock cycles (number of blocks), so that the observed speedup is $\Omega(10^5)$. Beyond this order of magnitude that might be reduced if an optimized software was designed, the important result is that experiments that are obviously not within our grasp with a software approach may be easily performed on the FPGA (some tens of seconds being sufficient to have a valuable statistical estimate of the behavior of the system for a given set of parameters).

Finally, we mention the fact that many experiments handle values of M lower than 7, and the most up-to-date FPGAs (XC6VLX160) contain up to 720 embedded 36 Kbit B-RAM memories (each B-RAM being able to store the states

FIGURE 6: Implementation of a cell of the environmental layer.

of a group of 3×4 cells). Therefore our implementation might scale up to more than 4 300 000 cells (the logic resources utilization rate remaining markedly below 100%).

5. The Particle Layer

When restricted to the environmental layer, the model of [6, 7] only takes advantage of a reaction-diffusion mechanism. We are mostly interested in the decentralized gathering that occurs when amoebae are subject to a chemotaxis process. We now focus on these amoebae that are modeled by agents.

5.1. Discrete Model. The amoebae are supposed to be all identical, and in constant number as no birth or death process is considered. Several amoebae may be located at the same cell. We arbitrarily allow only one amoeba to move from a nonempty cell at each time step. We do not limit the number of amoebae that can simultaneously move to a given cell, but we arbitrarily choose to allow an amoeba to go on a neighboring cell only if this cell contains less than two amoebae [5–7]. Let us define a cell that contains no amoeba as an *empty* cell, and a cell that contains strictly less than two amoebae as a *free* cell. The movement rules state that, at each time step, for each non-empty cell, one single amoeba may

- (i) move to an adjacent free cell (rule (R4),
- (ii) move to an adjacent excited free cell (rule (R5),
- (iii) stay on the same cell (rule (R6).

To apply rule (R4) (noise rule), we consider that each non-empty cell may send an amoeba to one of its neighbors with probability p_A , called the *agitation rate*. This neighbor is randomly selected among all neighbors that are free. Similarly, to apply rule (R5) (chemotaxis rule), amoebae move to a cell that is randomly selected among the excited free cells of the neighborhood. Rules (R4) and (R5) are made mutually exclusive. Formally, for $t \in \mathbb{N}$ and $c \in \mathcal{L}$, let $\tilde{N}_c^t$,

respectively, $\tilde{E}_c^t$, be the set of *free* cells, respectively *excited free* cells, in the neighborhood of c . For a finite set S , we denote by $\mathcal{R}(S)$ the operation of selecting one element in S with uniform probability, with the convention $\mathcal{R}(\emptyset) = \emptyset$. $\mathcal{R}$ randomly selects a neighbor for moving. We use a Bernoulli function to impose noise on the moves of an amoeba with probability p_A . To represent the move of one amoeba from a *non-empty* cell c to another cell Δ_c^t , with the convention $\Delta_c^t = \emptyset$ if no move occurs, we have

$$\text{if } \mathcal{B}(p_A) = 1, \quad \text{then } \Delta_c^t = \mathcal{R}(\tilde{N}_c^t), \quad (\text{R4})$$

$$\text{else if } \sigma_c^t = 0, \quad \text{then } \Delta_c^t = \mathcal{R}(\tilde{E}_c^t), \quad (\text{R5})$$

$$\text{else } \Delta_c^t = \emptyset. \quad (\text{R6})$$

5.2. Coupling of Environment and Particles. Amoebae act on the environment by emitting excitations that propagate to neighboring cells. We do not take into account the number of amoebae contained in each cell; a non-empty neutral cell may become excited with probability p_E called the *emission rate*. Since this rule may interfere with rule (R1), we combine both rules into rule (R1'):

$$\begin{aligned} \sigma_c^{t+1} = M \quad & \text{if } \mathcal{B}(p_T) = 1, \sigma_c^t = 0, \\ & (|E_c^t| > 0 \text{ or } (c \text{ non-empty and } \mathcal{B}(p_E) = 1)). \end{aligned} \quad (\text{R1}')$$

5.3. Properties. Similar regimes may be observed as in Section 3.2: the non-coherent regime ($p_T < 1$), the extinction regime (p_T less than a critical value that depends on M), and the static regime, that is obtained in the case of systematic transmission of waves ($p_T = 1$) and if amoebae constantly initiate wave fronts ($p_E = 1$). In the static regime, excitation fronts collide systematically, so that amoebae are

not attracted by each other (no move can occur, since no information may be exchanged between different amoebae).

The most promising behavior, the *self-organizing regime*, is observed when the transmission is perfect and when the emission rate is less than 1 (typically $p_E = 0.1$) and for various values of agitation rate. In this regime, a gathering phenomenon shows a progressive merging of the amoebae from small clusters to large clusters, after a few tens to a few thousands of iterations (depending on the environment). The complexity of this hierarchical dynamics results from successive emerging behaviors: formation of waves, formation of first groups, extension and shrinking of the regions according to their respective size, and captures of small clusters by a few clusters. Among interesting properties observed in the system, Fatès has shown that gathering could also occur in the presence of obstacles as the virtual amoebae could take advantage of narrow corridors to find their way to an attracting cluster [6, 7].

Figure 7 illustrates the resulting aggregation and the propagation of waves (simulated by the software implementation developed by Fatès) in a “perfect” environment. Figure 8 illustrates the same phenomenon in an environment with both obstacles and noise. Purple pixels are the amoebae, green pixels are obstacles, excited and refractory cells are drawn with shaded orange colors, and neutral cells are white. The behavior of the model satisfactorily reproduces the aggregation properties of *Dictyostelium discoideum*, while fulfilling all required constraints. Moreover, the decentralized gathering appears as robust to noise and irregular topologies. For further details and illustrations about the model dynamics and its self-organizing regime, see [6, 7].

6. Hardware Implementation of the Model

Following the study of the dynamical behavior of the model in [6, 7], we also set $p_T = 1$ since aggregation only occurs with perfect transmission. From now on, we arbitrarily use the 8-neighborhood, and we set the excitation level to $M = 3$.

6.1. Cell and Amoebae Implementation. For implementation purposes, we define a *node* as a cell together with the amoebae it contains. The moves of amoebae may be simply described as the evolution of the “population” of each node as a part of its internal state. Figure 9 shows the I/O of the node module.

6.1.1. State of a Cell. Considering the environmental layer only, the state of each cell belongs to $\{0, 1, 2, 3\}$, so that we code it with two bits (s_1, s_0). This state evolves according to rules (R1'), (R2), and (R3). These rules may be expressed as the state machine depicted in Figure 10. Signal p_E stands for $\mathcal{B}(p_E)$. Input `not_empty` codes for the presence of at least one amoeba in the node (it is an internal signal generated by the module that codes the population of the node).

6.1.2. Population of a Node. Considering the amoebae, they are coded as the number of amoebae that are located in the cell that corresponds to the local node. Amoebae may move

FIGURE 7: Aggregation of amoebae (purple) and propagation of waves (shaded orange) in a “perfect” environment (steps 0, 1, 2, 4, 10, 15, 20, and 40).

towards free cells only. Free cells contain at most one amoeba. Since up to 8 amoebae may simultaneously move towards a free cell, each node contains at most 9 amoebae. Instead of coding the population size (using 4 bits and counting at each time the number of arriving amoebae), we use 9 flip-flops: though less compact in terms of number representation, this solution does not require coding and counting resources, so that it uses significantly less logic cells. The first flip-flop stores “1” if there is at least one amoeba. Then the 8 other flip-flops directly receive arriving amoebae. Each time an amoeba leaves the node, one of the flip-flops storing “1” is reset to “0” (the reset command is transmitted among flip-flops until finding a “1”). Similarly, if amoeba arrivals occur when the cell is empty, then the first flip-flop is set to “1” and one of the other flip-flops is reset to “0”. Figure 11 depicts the resulting architecture to store the node population. The node indicates whether it is free or not with signal `free_out`.

6.1.3. Amoeba Moves. Figure 12 shows how the moves of the amoebae are implemented (rules (R4), (R5), (R6)). Signal p_A stands for $\mathcal{B}(p_A)$. It controls 8 multiplexers (one for each neighbor) that indicate whether the corresponding neighbor is free or excited and free. Moreover signal `neutral` is used in the second case (R5). It is internal and it codes for $\sigma_c^t = 0$,

FIGURE 8: Aggregation of amoebae (purple) with obstacles (green) and noise (steps 0, 1, 2, 4, 10, 20, 40, and 60).

that is, $(s1, s0) = (0, 0)$. In the same way, if signal `not_empty` is “0” then all choices are set to “0” because no amoeba may move if the cell is empty. Then the `Select` module randomly selects only one choice among possibly several. Finally an OR gate determines if an amoeba will move while the bus `am_towards` indicates where it will move.

The random selection of a signal set to “1” among possibly several is complex. In our implementation, we use a cyclic priority module, where the main priority is given to a signal that is randomly specified by three bits provided by a linear feedback shift register (LFSR), as shown in Figure 13. This implementation suffers the following drawbacks: (1) it is not uniformly random, and (2) though it is fair, it introduces some systematic bias in the selection of close signals because of the cyclic priority. Nevertheless, first experiments indicate that these drawbacks do not modify the overall behavior of the model.

6.2. Random Processes. The definition of the model includes several random aspects: $\mathcal{B}(p_T)$, $\mathcal{B}(p_A)$, $\mathcal{B}(p_E)$, $\mathcal{R}(\tilde{N}_c^t)$, $\mathcal{R}(\tilde{E}_c^t)$. The software implementation uses the same RNG for all nodes and for all Bernoulli laws (see Section 4.3 for a discussion about the required RNG quality). But in this parallel hardware implementation, all streams of stochastic

bits must be generated by separate modules, in each node. Since we consider the case, where $p_T = 1$, and since our random selection module (used for both $\mathcal{R}(\tilde{N}_c^t)$ and $\mathcal{R}(\tilde{E}_c^t)$) just needs a single LFSR, we finally have to implement $3 \times X \times Y$ RNGs. As for the environmental layer, the cost in space for all random aspects of the model is huge.

We choose again to adapt the LFSR-based RNGs of [24]. Experiments in [6, 7] show that both emission and agitation rates do not need a high precision. Therefore we use two 63-bit RNGs adapted from [24], comparing only 8 of their generated bits to p_E and p_A (coded on 8 bits). All RNGs use different seeds (set serially during initialization). Figure 14 depicts one of the used RNGs (to choose other irregularly extracted bits, one has just to use other arrangements of SRAMs and flip-flops, and pick the 8 signals at different places).

The selection module uses a 3-bit random counter to define the main priority choice. Since all 3 bits must be simultaneously accessed, 3 flip-flops are required. Instead of only using 3 bits for the random counter (resulting in an 8 cycle periodicity), we use here an adapted version of the 15-bit random counter of [24] that only needs 3 logic cells to strongly increase the periodicity without requiring more resources.

6.3. General Architecture. Figure 15 describes the general architecture of our implementation. It consists of a grid of 40×30 identical nodes. Border nodes receive constant inputs from their nonexistent neighbors (`exc_in`, `free_in`, and `cell_in` are set to “0” for these nodes).

6.3.1. Initialization. In this implementation, the user defines the desired average number of amoebae. Then the induced ratio (number of amoebae/number of cells) is sent at run time to all nodes, that use it in combination with their 63-bit RNG (threshold compared with the 8 extracted bits), so as to decide whether they initially contain an amoeba or not. This initialization scheme avoids the resource consumption of the large demultiplexer that is required when an external memory defines the exact initial positions of the desired amoebae (this second version has been synthesized but not validated onboard).

6.3.2. Output. In the current version (validated on board) the states of all nodes are sequentially sent as an output to the host PC though the Master bus. This large output is useful for debug, but it requires a significant amount of resources, and it takes time. In the final version (not yet validated onboard), we take advantage of the quantitative criterion BBR (bounding box ratio) that is used in the experimental study of [6, 7] for the evaluation of the aggregation: minimal relative size of an array of nodes that contains all amoebae. Therefore, we implement an OR gate for each row and for each column of nodes, and we compute on-chip the resulting BBR, that is sent to the host PC in real time (i.e., during each clock cycle, the BBR is computed while all node states are updated).

FIGURE 9: Node module.

FIGURE 10: Cell state machine.

TABLE 3: Synthesis results for a 40×30 grid.

40 × 30 grid hardware resource utilization Xilinx FPGA XC4VLX160ff1513-12	
Number of Slice Flip Flops	31,642/135,168 (23.4%)
Number of 4 input LUTs	113,458/135,168 (83.94%)
Number of occupied Slices	61,727/67,584 (91.33%)
Frequency	130 MHz

6.4. Implementation Results

6.4.1. Resource Consumption. The prototyping platform is the same as in Section 4.4. Each node module requires 57 slices (21 for the different RNGs). Table 3 gives the synthesis results for an environment of 40×30 cells, taking advantage of resource optimization. Among the 61,727 used slices, only 794 ones implement the control and I/O handling (though we output all node states in the current version), so that the whole architecture is implemented on 91 of the FPGA resources.

6.4.2. Speedup. Software implementations on a micro-processor-based computer, Pentium 4.2 GHz, require $170 \mu\text{s}$ per evolution step for a 40×30 grid. As for the simulation of the environment alone, the main bottleneck for the software computation time lies in the random number generation (no cache management issue here). The maximum clock frequency of the proposed hardware is 130 MHz. Thus, the implementation on the Virtex-4 provides a speed factor up to $22000\times$.

Again, it must be pointed out that the used software is not optimized and has been written in Java, and that it does not perform exactly the same computations as the hardware architecture (random number generation, handling of priorities among neighbouring cells). Therefore, we consider that these results only indicate a $\Omega(10^4)$ order of magnitude for the speedup.

6.4.3. Analysis. Depending on the parameter values, the size of the environment and the obstacles, aggregation occurs in the experiments in [6, 7] after up to 20,000 iterations. Therefore the great speedup we obtain becomes particularly interesting if we are able to implement much larger grids.

Such improvements strongly depend on the analysis of the limits of the implementation depicted in this work (which was the main goal of the hardware design of the whole model with amoebae, as explained before). This analysis highlights three major sources of area consumption: coding and handling of populations of amoebae (28%), priority handling (23%), and above all random number generators (37%). Moreover, the implementation of the environment alone shows the great improvements that may be obtained thanks to a block-synchronous approach. But the described implementation would require that we store 11 bits per node (population + state) in the B-RAMs, and most of all, the exchanges of amoebae between nodes at the border could not be performed with sequentially handled blocks (since this handling results from a bidirectional information exchange through the `cell_in` and `am_towards` signals). This is why the current description of the model does not easily fit a block-synchronous version with amoebae.

FIGURE 11: Module to code the population of amoebae.

FIGURE 12: Implementation of the particle layer.

FIGURE 13: Selection module.

FIGURE 14: 63-bit random number generator to generate Bernoulli laws.

FIGURE 15: General architecture.

All these issues have led us to explore the definition of a new model for this decentralized gathering process. This new approach is fully based on cellular automata, including the RNGs. Though many theoretical and hardware aspects still need to be studied, it appears to be able to reduce the implementation area drastically: populations are directly handled through the cell state, resulting in a more likely block-synchronous implementation (though a fully parallel implementation corresponds more to the idea of decentralized gathering we explore), and random number resources are spatially mutualized. This is the main current research subject within the Amybia project.

7. Towards Decentralized Gathering of Computational Resources

The context of this work is the definition of innovative schemes of decentralized and massively distributed computing. Recent trends of integrated circuit design investigate various types of alternative computing devices based on multiple generic computing units, possibly distributed in an unknown and irregular way [26]. As stated in the introduction, our work aims at answering one of the problems raised by such new computing paradigms: how to gather enough computing resources to solve a given task. Though this paper describes an upstream work that does not yet pretend to define precisely how the gathering process will be applied to a real system, we may exhibit two possible contexts of use for such decentralized gathering.

7.1. Robot Swarms. Considering a swarm of simple robots that evolve in an environment with very restricted communication possibilities (due to obstacles for example), one may consider a task that alternates exploration and cooperation steps. Exploration is performed by robots that behave as autonomous agents, while cooperation is required when a “target” has been found. Robots that find targets try to attract other robots through decentralized gathering, until a sufficient number of gathered agents are able to perform the task associated to the target. Then robots start again their individual exploration.

As a first experimental setup, we have already implemented our decentralized gathering algorithm with Alice micro-robots (see a demo on <http://www.loria.fr/~fates/Amybia/project.html>). This application shows the great robustness of our algorithm, since these old robots have only two sensors to detect the light-simulated waves of the environment, and their motions are heterogeneous and almost unpredictable, due to the faulty control of their wheels. In such a context, the study of the properties of our decentralized gathering algorithm is essential, and it may take advantage of rapid simulations on FPGA; whereas an embedded implementation of the whole algorithm has no meaning, since each robot is an agent.

7.2. Task Assignment. Decentralized gathering may also be useful to handle task assignment in a massively distributed and heterogeneous computing device. In such a context,

“moving” agents might correspond to transmitting the task assignments between units when using computational resources with fixed locations. In such devices, communication costs depend on the distance between the units, so that the communicating threads should be assigned to neighboring resources if possible. In a multi-task context, when a thread gives birth to other threads, they may be assigned to available computational resources that are not located in the neighborhood. When some resources become idle after having completed some thread, a reassignment process could be useful to gather the resources that handle the threads associated to the same task. A permanent decentralized gathering process might be useful for that if the resources are irregularly distributed and possibly faulty, provided that its cost is negligible with respect to the threads. Other constraints must be studied, such as the cost of context transfer between computational units, or the extension of decentralized gathering to multiple sets of agents to handle multiple tasks. Our preliminary implementation work does not conclude yet about the feasibility of a decentralized gathering process with a negligible cost.

8. Conclusion and Future Work

In this paper, a bioinspired model to solve the decentralized gathering problem is shortly described. It is based on the aggregation properties of the cellular slime mold *Dictyostelium discoideum* that may live as a monocellular organism, and that is able to behave as a multicellular organism when needed. We model the environment and the individual amoebae by means of cellular automata and reactive agents (simple computational abilities and no memory).

We have designed a hardware parallel implementation of the environment alone, that helps us perform rapid large-scale simulations to study the properties of our model, such as its robustness to noise and obstacles. The implementation results are highly satisfactory in terms of computation speed and environment size. This implementation is currently used so as to perform rapid simulations of phase transitions within a close-to-the-stable-state experimental framework.

Focusing on the whole model (environment and amoebae), we have designed a fully parallel hardware implementation so as to study its ability to provide a massively distributed computational model for decentralized gathering. Despite a great speedup factor, our implementation work points out two main limitations. In terms of embeddability, the area cost of the stochastic aspects of the model is important. Therefore, our theoretical study should evaluate the robustness of our model to low-quality random streams that may also be spatially correlated. In terms of usefulness for large-scale efficient simulations, the grid size we are able to handle does not correspond to interesting experimental environments, and the corresponding software computation time does not justify the use of fast FPGA-based simulations. To significantly increase the grid sizes handled by the FPGA, we currently explore solutions that are based on a block-synchronous approach and a new description of the model

that is fully based on cellular automata. This CA-based approach does not only intend to insert the behaviour of the agents within the state of each cell, but it also applies to the generation of random numbers. We currently consider the definition and design of spatially mutualized CA-based RNGs, that ensure both low-area implementations and a satisfactory spatial independence.

Acknowledgments

The authors wish to thank the other members of the *Amybia* INRIA collaborative research project (<http://www.loria.fr/~fates/Amybia/project.html>), Nazim Fatès and Hugues Berry, for their useful help and comments.

References

- [1] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer, "Gathering of asynchronous robots with limited visibility," *Theoretical Computer Science*, vol. 337, no. 1–3, pp. 147–168, 2005.
- [2] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, Oxford, UK, 1999.
- [3] H. Abelson, D. Allen, D. Coore, et al., "Amorphous computing," *Communications of the ACM*, vol. 43, no. 5, pp. 74–82, 2000.
- [4] J. Lawson and D. Wolpert, "Adaptive programming of unconventional nano-architectures," *Journal of Computational and Theoretical Nanoscience*, vol. 3, pp. 272–279, 2006.
- [5] T. Bretschneider, B. Vasiev, and C. J. Weijer, "A model for cell movement during *Dictyostelium* mound formation," *Journal of Theoretical Biology*, vol. 189, no. 1, pp. 41–51, 1997.
- [6] N. Fatès, "Gathering agents on a lattice by coupling reaction-diffusion and chemotaxis," Tech. Rep., INRIA Nancy Grand-Est, 2008, <http://hal.inria.fr/inria-00132266/>.
- [7] N. Fatès, "Solving the decentralised gathering problem with a reaction-diffusion-chemotaxis scheme - Social amoebae as a source of inspiration," to appear in *Swarm Intelligence*, 2010 <http://hal.inria.fr/inria-00132266/>.
- [8] K. Sugihara and I. Suzuki, "Distributed motion coordination of multiple mobile robots," in *Proceedings of the 5th IEEE International Symposium on Intelligent Control*, vol. 1, pp. 138–143, Philadelphia, Pa, USA, September 1990.
- [9] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita, "Distributed memoryless point convergence algorithm for mobilerobots with limited visibility," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 5, pp. 818–828, 1999.
- [10] G. Prencipe, "Impossibility of gathering by a set of autonomousmobile robots," *Theoretical Computer Science*, vol. 384, no. 2–3, pp. 222–231, 2007.
- [11] S. Nagano, "Modeling the model organism *Dictyostelium discoideum*," *Development Growth and Differentiation*, vol. 42, no. 6, pp. 541–550, 2000.
- [12] A. Marée, *From pattern formation to morphogenesis: multicellular coordination in Dictyostelium discoideum*, Ph.D. thesis, Utrecht University, 2000.
- [13] B. Vasiev, F. Siegert, and C. J. Weller II, "A hydrodynamic model for *Dictyostelium discoideum* mound formation," *Journal of Theoretical Biology*, vol. 184, no. 4, pp. 441–450, 1997.
- [14] S. Mackay, "Computer simulaion of aggregation in *Dictyostelium discoideum*," *Journal of Cell Science*, vol. 33, pp. 1–16, 1978.
- [15] J.-L. Martiel and A. Goldbeter, "A model based on receptor desensitization for cyclic AMP signaling in *Dictyostelium* cells," *Biophysical Journal*, vol. 52, no. 5, pp. 807–828, 1987.
- [16] J. R. Weimar, "Cellular automata for reaction-diffusion systems," *Parallel Computing*, vol. 23, no. 11, pp. 1699–1715, 1997.
- [17] A. Adamatzky, *Computing in Nonlinear Media and Automata Collectives*, Institute of Physics, Bristol, UK, 2001.
- [18] N. Fatès, "Asynchronism induces second order phase transitions in elementary cellular automata," *Journal of Cellular Automata*, vol. 4, no. 1, pp. 21–38, 2009.
- [19] H. J. Blok and B. Bergersen, "Synchronous versus asynchronous updating in the "game of Life"," *Physical Review E*, vol. 59, no. 4, pp. 3876–3879, 1999.
- [20] H. Hinrichsen, "Nonequilibrium critical phenomena and phase transitions into absorbing states," *Advances in Physics*, vol. 49, no. 7, pp. 815–958, 2000.
- [21] B. Girau, A. Boumaza, B. Scherrer, and C. Torres-Huitzil, "Block-synchronous harmonic control for scalable trajectory planning," in *Robotics Automation and Control*, A. Iazinic, Ed., pp. 85–110, I-Tech Publications, 2008.
- [22] I. Kokolakis, I. Andreadis, and P. Tsalides, "Comparison between cellular automata and linear feedback shift registers bases pseudo-random number generator," *Microprocessors and Microsystems*, vol. 20, no. 10, pp. 643–658, 1997.
- [23] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, 1998.
- [24] M. George and P. Alfke, "Linear feedback shift registers in virtex devices," *Xilinx Application Note XAPP210 (v1.3)*, 2007.
- [25] G. DeYoung, P. B. Monk, and H. G. Othmer, "Pacemakers in aggregation fields of *Dictyostelium discoideum*: does a single cell suffice?" *Journal of Mathematical Biology*, vol. 26, no. 5, pp. 487–517, 1988.
- [26] A. Dehon, "Very large scale spatial computing," in *Proceedings of the 3rd International Conference on Unconventional Models of Computation (UMC '02)*, pp. 27–37, Kobe, Japan, October 2002.