The Swarm Computer, an Analog Cellular-Swarm Hybrid Architecture

Ryanne Dolan, Guilherme DeSouza, and Dan Caputo
University of Missouri
Vision-Guided and Intelligent Robotics Lab
CS, ECE, and Physics Departments

Abstract—**The "killer apps" of cellular and swarm computing are image processing and optimization, respectively; however, applying these platforms to general-purpose computing remains impractical. Designing systems within the restrictive framework of cellular automata is extremely difficult, though often very efficient and scalable. On the other hand, swarm networks are very powerful but difficult to implement in hardware. Here we introduce a hybrid model, the Swarm Computer, which is both practical to program and efficient to implement. Applications in astrophysics and image processing are considered.**

I. Introduction

Recently, there has been an increasing reliance on nature as a source of inspiration for new algorithms in computer science. Evolution has solved many problems in order to sustain life; it makes sense to study the solutions evolved over billions of years if they work well in nature. Two areas of research in computer science that have spawned out of this biological influence recently are cellular computing (e.g. membrane computing) and swarm computing (e.g. ant colony optimization). The field of cellular computing seeks to apply the simple dynamics and massive parallelism found in arrays of immovable biological cells and their artificial analogues. By contrast, swarm computing applies the collective behavior of groups of moving agents (e.g. ants) to solve a problem.

Research in these areas has produced some very interesting algorithms, especially in the fields of computational intelligence and image processing. However, neither cells nor swarms have proven thus far to be a practical platform for general-purpose computation. In this paper, we introduce a novel model of computation which is a hybrid of cellular and swarm models. The model is universal, scalable, and often efficient, as it inherits the best features of both stationary cells and swarming organisms. The model is theoretically universally applicable to general-purpose computing, but we discuss some applications for which the model is especially well suited.

II. Background

A. Cellular Computing

The first cellular computers were the classic cellular automata pioneered by von Neumann, Wolfram, and Conway over the past several decades [4], [22], [26]. All cellular computers are characterized by a regular lattice of simple cells locally connected in a fixed network. Each cell maintains a state which evolves through time based on a function of the previous state and the previous states in the local neighborhood. Often this function is specified with simple pattern matching rules, fuzzy logic [28], or differential equations [3].

Cellular computing is often applied to image processing [27], [28], [25], [15], [14], [11], [7]. This requires that each cell in a rectangular grid be associated with a pixel in an image. Typically, the cell is initialized with color data from an input image. After the grid evolves to a steady state, the resulting output state represents a processed image.

Some cellular automata are universal in the Turing sense, e.g. Conway's Game of Life [23] and the Celluran Neural Network Universal Machine [19], [6], [5]; however, programming such automata to perform general-purpose computing tasks remains exceedingly complex. Image processing is a common application of cellular computation due to the fact that many image processing tasks are space invariant. Since cellular computing performs the same function at each cell (pixel), it is difficult to implement global operations, though not impossible. Cellular automata must propagate information through the network over time for this effect. Wave computing is one sub-field of cellular computing which relies on controlled information propagation in cellular networks [18].

B. Swarm Computing

Swarms as seen in nature have evoked the curiosity of many scientists throughout history and have recently inspired several important algorithms in the fields of computational intelligence and artificial intelligence. Swarm networks use a paradigm similar to that of cellular automata: a network of agents communicate locally and evolve due to some function. The difference is that swarm networks allow for dynamic neighborhoods, wherein agents are able to move in space. Swarm agents maintain a state but also a position. Rather than finite, static neighborhoods, swarm networks involve pairwise distance calculations to determine which agents are close enough to communicate.

The most famous algorithm in computer science to be inspired by swarm behavior in nature is the Particle Swarm Optimizer (PSO) [13]. Interestingly, this algorithm is not swarm computing but a complex cellular automata due to the fact that the particles are arranged in a grid with static neighborhoods; however, the algorithm still stands as a typical application of swarm-like behavior in engineered systems. It

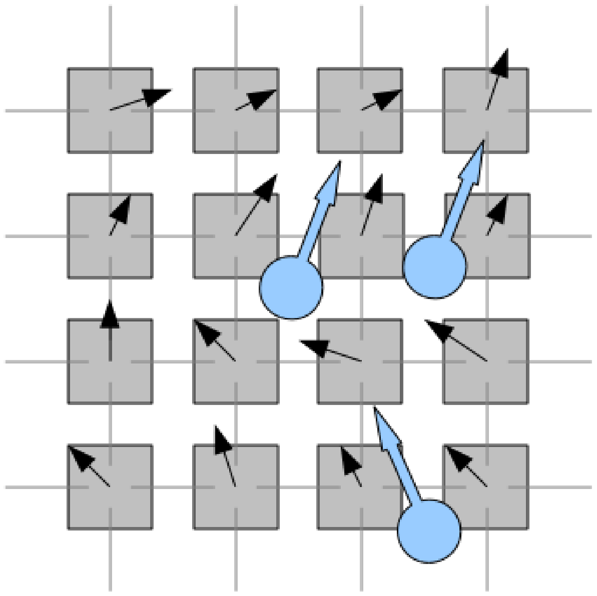

Figure 1. agents traveling in cellular medium, with arrows indicating state vectors

also demonstrates a major efficiency hurdle associated with swarm computation. PSO uses static neighborhoods rather than suffer the computational complexity of pair-wise distance calculations to determine proximity. It is assumed that agents nearby in the static network will end up nearby in state-space as well, so PSO can be said to approximate swarm behavior in this sense.

C. Cellular-Swarm Computing

The previous sections highlight two major problems with both cellular and swarm computing: it is hard to formulate algorithms in terms of spatially invariant rules in cellular automata; on the other hand, it is computationally expensive to allow swarm agents to form dynamic neighborhoods.

To ameliorate these problems, we propose a hybrid model of computation, called cellular-swarm computation, which is both practical to program and implement. We will define cellular-swarm computation as the use of a system with the following characteristics:

- a static network of cells (called the medium) arranged in a regular lattice
- a swarm of agents that are free to move about the medium and interact with the medium locally (see Figure 1)

The medium is exactly a Generalized Cellular Automata [2], and is described with a vector function F (typically a differential equation). Each cell evolves due to F, influenced only by the previous states in its local neighborhood (including itself).

The agents do not form a swarm network in the usual sense, since they are not allowed to communicate between themselves directly. Thus, there is no neighborhood to find

	input	output
F	neighborhood states	cell state
G	agent state and position	agent position
H	agent state and position	agent state
Y	agent state and position	cell state

Figure 2. cellular-swarm programmable functions

and no pairwise distances to calculate. Instead, the position of an agent is mapped to a single cell in the medium, and the agent may only interact locally with that cell. Specifically, an agent may at each timestep:

- detect the state of the current cell and the states of the cell's immediate neighbors
- modify the state of the current cell (according to an output function Y)
- move to an adjacent cell (according to a position function G)
- modify its own state variable (according to an agent state function H).

The functions F, G, H, and Y fully specify a cellular-swarm computation ("program", see Figure 2). The dimensions of the medium and the number of agents are considered arbitrary and may be implementation specific. Given the initial state of all cells in the medium, a swarm computation continues until a steady-state output emerges.

The cellular-swarm model of computation is universal in the Turing sense, as it inherits universality from the generalized cellular automata; however, it is more practical in many ways, as the following sections illustrate.

D. Related Topics

Other recent papers have proposed cellular-swarm hybrid architectures with good results. In particular, [10], [21] both use cellular automata to augment the particle swarm optimizer, making it more robust to local optima and dynamic environments. Additionally, [12] shows that swarm networks can be used for general-purpose computing. The current paper is the first to propose a hybrid cellular-swarm computer for general-purpose computing.

III. The Swarm Computer

Here we describe one possible hardware platform which implements cellular-swarm computation. The "Swarm Computer" is an analog architecture capable of universal computation within the cellular-swarm paradigm (see Figure 3). The computer comprises the following components:

- a grid of identical circuits implementing the cell update function F
- an array of identical circuits implementing the agent position function G
- an array of identical circuits implementing the agent state function H
- an array of identical circuits implementing the agent output function Y

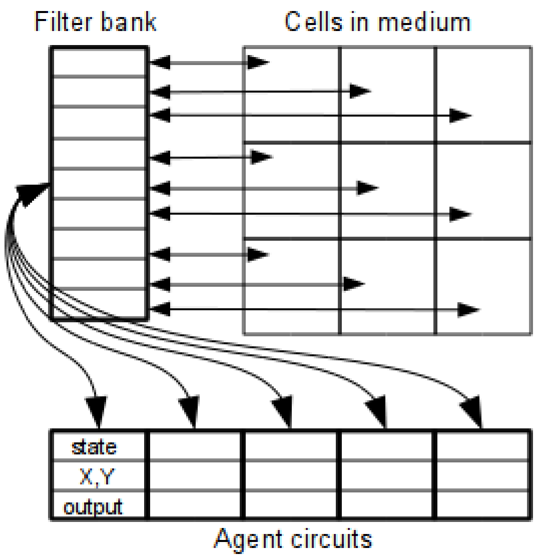

Figure 3. swarm computer architecture

- an array of filters and gating devices, which select one cell per agent

The agent position circuits expose two signals (in the form of voltages across a pair of terminals) corresponding to the horizontal and vertical position of an agent. Likewise, the agent output circuits expose a single output signal per agent.

The filter bank requires more explanation. For each pair of agent position and output circuits, the corresponding filter selects one cell circuit from the grid based on the horizontal and vertical position signals. A gating mechanism sinks the output associated circuit to the target output cell, which in turn feeds back to the corresponding agent position and state circuits. The filter bank is designed to select only one cell per agent and to prevent an agent from interacting with nonlocal cells.

Each cell in the grid has an input and output signal in the form of a light sensor and lamp, for example. The state of each cell can be initialized prior to computation by exposing the grid to an image. After steady-state, the lamps at each cell will indicate via their intensity the final state of the corresponding cell.

By fabricating this sort of "vision chip" with custom circuitry for functions F, G, H, and Y, any image processing algorithm can be realized. In the case of spatially invariant algorithms, the agent circuits (G, H, and Y) can be ignored or grounded, since only the function F is needed. For spatially variant algorithms, the function F can be designed to cause information to propagate through the network of cells, or else functions G, H, and Y can be designed to transport packets of information along a trajectory through the network.

A. Runtime Complexity and Scalability Considerations

Unlike traditional processors, the swarm computer does not perform one instruction at a time. Instead, there is only one "program" that continuously applies the same functions to the entire state vector. For this reason, the runtime complexity of algorithms implemented on the swarm computer is hard to define. However, we can compare the runtimes of existing cellular and swarm algorithms with their corresponding implementations on the swarm computer, clearly illustrating improvements in many cases.

First, we might consider the simulation of Cellular Neural Networks (CNNs) on the swarm computer. A CNN is a special case of the generalized cellular automata which forms the communication medium of the swarm computer [3]. Therefore, it is not surprising that CNNs are easily implemented on the swarm computer. In particular, the Chua circuit is used for each cell update circuit, and all agent circuits are ignored (since no agents are required for this simulation).

Now it is easy to see the speed improvement of implementing CNNs on the swarm computer compared to traditional computers. A software simulation of CNNs would involve an $O(KMT)$ runtime, where K is the number of neighbors to each cell, M is the number of cells, and T is the settling time. Typically, the neighborhood is small (9 neighbors) and the settling time is short, so we may write this as $O(M)$, or linear with the size of the network.

By comparison, the swarm computer performs all M cell updates in parallel. Ignoring settling time again, the same simulation takes constant time $O(1)$. Not surprisingly, the M cells of the swarm computer offer a speedup of a factor of M for CNN simulation and some similar massively parallel operations. This improvement is comparable to GPU-based CNN simulation [7].

Next, we consider the simulation of any swarm algorithm on the swarm computer. All swarm algorithms involve pair-wise distance calculations among agents of the system in order to find which agents are near enough to communicate; therefore, a naive algorithm would have $O(N^2T)$ complexity, where N is the number of agents and T is the settling time. Search trees such as KD-Trees or R-Trees or approximate methods like Approximate Nearest Neighbors can speed up query time to $O(lgN)$, reducing the overall complexity of swarm algorithms to $O(KNlgN)$ where K is the number of neighbors queried (K-nearest neighbors) if we ignore settling time [9], [20], [1].

Since the swarm computer has N agents, we might expect the swarm computer to improve simulation time by a factor of N to $O(KlgN)$; however, this does not consider the fact that agents cannot directly communicate as required by the swarm algorithm. Instead, agents must communicate via waves propagated through the medium, which never happens instantaneously. Luckily, we can measure the speed of propagation of such waves, and with constant "speed of information" the time is proportional only to the size of the medium. Thus, we have a total runtime complexity of$O(M^{1/d})$ where d is the dimensions of the medium. Notice that we have removed

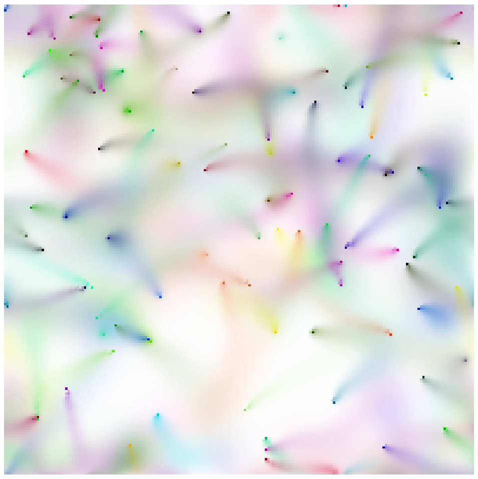

Figure 4. GPU simulation of agents leaving trails in large cellular medium

the dependence on the number of agents (and the number of neighbors) entirely from the computational complexity; indeed, the swarm computer can simulate any number of agents without affecting runtime. So long as $M^{1/d}$ is less than N (the dimensions of the network are small compared to the number of agents), the swarm computer is much more efficient than even the fastest swarm algorithms.

Of course, this discussion so far ignores the settling time of cellular-swarm computation compared to swarm or cellular computation methods. It is very possible that the runtime of a swarm simulation on the swarm computer would take longer than a traditional implementation if the settling time is much larger, e.g. due to the more complicated dynamics required by the wave-based inter-agent communication. However, this settling time is generally constant or nearly constant for a given algorithm, so it does not factor into runtime complexity analysis. Furthermore, it is always possible to change the number of agents in a particular swarm computer-based implementation without affecting the runtime complexity, since the number of agents never figures into the complexity. The settling time may be affected by the number of agents, however; in general, adding more agents will reduce settling time and shorten the runtime of any algorithm. For this reason, we can consider the swarm computer to be an extremely scalable platform for massively parallel computation. If a given "program" is slow to compute, it should always be possible to increase the number of agents and reduce settling time accordingly.

B. Simulation on GPUs

In addition to being relatively simple to fabricate in hardware, it is straightforward to simulate the swarm computer architecture using existing processors. Here we discuss software considerations.

Modern general-purpose graphics processing units (GPGPUs) offer a powerful SIMD computing platform which are relatively inexpensive and readily available to consumers. GPUs of this sort are implemented as a grid of multiprocessors with local memory, shared (global) memory, and a hardware scheduler to coordinate work among the processors. Each multiprocessor (often there are 16 or more) can be programmed with a separate kernel function, which is applied to a region of the shared memory. Each multiprocessor is typically capable of performing multiple instances of the kernel concurrently (stream processing). With multiple multiprocessors and multiple "cores" within each multiprocessor, thousands of simultaneous kernel computations are possible with incredible speed.

Exploiting this parallelism to simulate the swarm computer is straightforward. Simulating the medium involves Newton's method or similar numerical integration techniques to approximate the state evolution of the analog system. Both the medium (cells) and agents can be implemented in this way on the GPU.

Other research has already pioneered the application of GPUs to the simulation of cellular media [7], [8], [11]. A simplified approach is mentioned here. Each multiprocessor is responsible for a region of cells in the medium and is programmed to continuously update per-cell state vectors in shared memory.

To simulate the agents, one multiprocessor is reserved to implement the G, H, and Y functions. Agent state and position is stored in local memory (only accessible to the one multiprocessor) to prevent unnecessary reads and writes to the shared memory. Within the agent kernel function, each agent's position is hashed to an index into shared memory. In this way, each agent can query the state of the current cell in constant time.

Several optimizations to this approach are possible; however, even this simplistic implementation is capable of updating millions of cells a second on consumer hardware [7].

IV. Applications

A. Image Processing

The swarm computer inherits many applications from the generalized cellular automata and cellular neural network such as image processing, image compression, and optimization [17], [15], [16], [28], [27]. In particular, the medium alone is sufficient for computing space invariant image processing operations and, like the cellular neural network, global operations are possible via the propagation of waves through the medium.

More interestingly, the swarm computer's agents allow information "packets" to travel to different parts of an image, extending lines, following gradients, smoothing contours etc along their trajectories. This makes the cellular-swarm hybrid model a much more attractive platform for high-level image processing, whereas cellular automata have so far been mainly used for low-level processing.

B. Astrophysics

Perhaps less obvious is the application of the swarm computer to N-Body Simulation in astrophysics [24]. To simulate the interaction of forces between stars, planets, etc in space, these simulators require pairwise distance calculations among

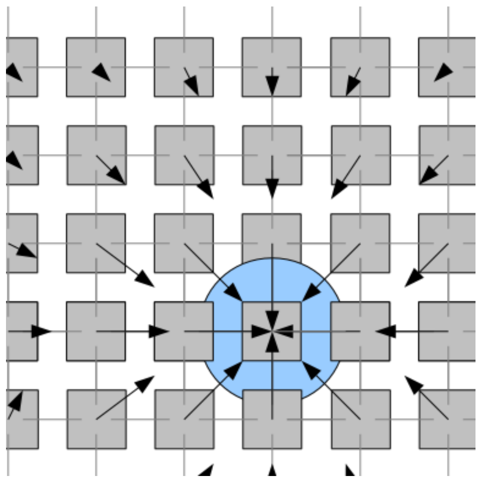

Figure 5. force due to gravity after propagation through the medium

the N "bodies". The problem with these simulations is the explosive computational complexity of computing pairwise distances among millions of bodies in a galaxy, for example.

N-body simulation can be viewed as a form of swarm computation; therefore, the swarm computer should offer an efficient and scalable platform for n-body simulators. It is rather straightforward to implement n-body simulations on the swarm computer, in fact. Just as with other swarm computations, the n-body simulation requires that information be propagated through the cellular medium via waves, thus eliminating the pairwise distance calculations and reducing runtime complexity. In the case of astrophysics simulations, the waves may represent "gravity waves", wherein a flux diffuses through the medium, affecting nearby bodies in proportion to the distance between them. The farther two bodies are apart, the less magnitude of the force of gravity, as expected. Strangely, the increased distance also causes a delay, since the gravity wave must propagate further through the medium. If the waves propagate quickly compared to the speed of the agents, the delay can be negligable. This is related to the "speed of gravity" compared to the speed of planetary motion in physical space.

The dynamics of the cells needed to support this sort of wave propagation can be described as a network of masses connected with springs. This mass-spring lattice allows displacements to spread into basins of attraction (gravity wells). As the agents (bodies) travel through the medium, they follow a straight line in the warped "space-time" medium.

V. Conclusions

We have presented a new hybrid model of computation based on both cellular automata and swarms as found in nature, as well as a corresponding analog hardware platform called the Swarm Computer. The architecture is practical to implement in hardware and easily simulated using modern general purpose GPUs. Application of the swarm computer to image processing, cellular computation, and swarm computation is straightforward and more efficient than corresponding implementations on typical processors or the traditional Turing machine. Moreover, algorithms implemented on the swarm computer scale gracefully when increasing the number of agent circuits, without affecting the runtime negatively.

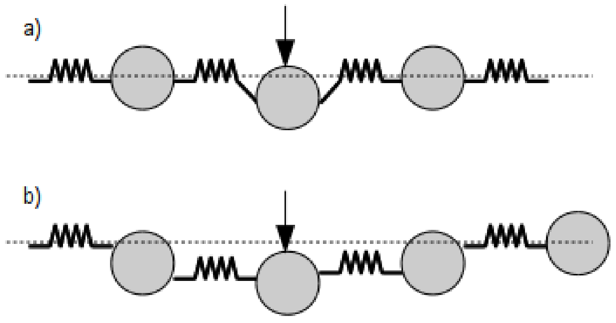

Figure 6. 1-D mass-spring medium showing propagation of "gravity wave" into gravity well
a) initial displacement due to presence of agent
b) gravity well after propagation through the network

References

[1] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):517, 1975.

[2] L.O. Chua. *CNN: A paradigm for complexity*. World Scientific Pub Co Inc, 1998.

[3] LO Chua and L. Yang. Cellular neural networks: theory. *Circuits and Systems, IEEE Transactions on*, 35(10):1257–1272, 1988.

[4] J. Conway. The game of life. *Scientific American*, 223:120–123, 1970.

[5] KR Crounse and LO Chua. The CNN Universal Machine is as universal as a Turing Machine. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on [see also Circuits and Systems I: Regular Papers, IEEE Transactions on]*, 43(4):353–355, 1996.

[6] R. Dogaru. *Universality and emergent computation in cellular neural networks*. World Scientific River Edge, NJ, 2003.

[7] Ryanne Dolan and Guilherme DeSouza. Gpu-based simulation of cellular neural networks for image processing. *Neural Networks, IEEE - INNS - ENNS International Joint Conference on*, 0:730–735, 2009.

[8] A. Fernandez, R. San Martin, E. Farguell, and G.E. Pazienza. Cellular neural networks simulation on a parallel graphics processing unit. *Cellular Neural Networks and Their Applications, 2008. CNNA 2008. 11th International Workshop on*, pages 208–212, July 2008.

[9] P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[10] A. Hashemi and M. Meybodi. Cellular Pso: A Pso for Dynamic Environments. *Advances in Computation and Intelligence*, pages 422–433, 2009.

[11] T.Y. Ho, P.M. Lam, and C.S. Leung. Parallelization of cellular neural networks on GPU. *Pattern Recognition*, 2008.

[12] T. Isokawa, F. Peper, M. Mitsui, J.Q. Liu, K. Morita, H. Umeo, N. Kamiura, and N. Matsui. Computing by Swarm Networks. *Cellular Automata*, pages 50–59, 2010.

[13] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks, 1995. Proceedings.*, volume 4, 1995.

[14] P. Kinget and MSJ Steyaert. A programmable analog cellular neural network CMOS chip for highspeed image processing. *Solid-State Circuits, IEEE Journal of*, 30(3):235–243, 1995.

[15] C.C. Lee and JP de Gyvez. Color image processing in a cellular neural-network environment. *Neural Networks, IEEE Transactions on*, 7(5):1086–1098, 1996.

[16] O. Moreira-Tamayor and J.P. de Gyvez. Subband coding and image compression using cnn. *Int. J. Circ. Theor. Appl*, 27:135–151, 1999.

[17] T. Nishio and Y. Nishio. Image processing using periodic pattern formation in cellular neural networks. *Circuit Theory and Design, 2005. Proceedings of the 2005 European Conference on*, 3:III/85–III/88 vol. 3, Aug.-2 Sept. 2005.

[18] T. Roska. Cellular wave computers for brain-like spatial-temporal sensory computing. *IEEE Circuits and Systems Magazine*, 5(2):5–19, 2005.

[19] T. Roska and LO Chua. The CNN universal machine: an analogic array computer. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on]*, 40(3):163–173, 1993.

[20] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R-tree: A dynamic index for multi-dimensional objects. *The VLDB Journal*, pages 507–518, 1987.

[21] Y. Shi, H. Liu, L. Gao, and G. Zhang. Cellular Particle Swarm Optimization. *Information Sciences*, 2010.

[22] J. Von Neumann. Von Neumann's self-reproducing automata. *Essays on Cellular Automata*, pages 4–65.

[23] R.T. Wainwright. Life is universal! In *Proceedings of the 7th conference on Winter simulation-Volume 2*, pages 449–459. ACM New York, NY, USA, 1974.

[24] S. Warren. Astrophysical N-body simulations using hierarchical tree data structures. 1992.

[25] T. Weishaupl and E. Schikuta. Parallelization of cellular neural networks for image processing on cluster architectures. *Parallel Processing Workshops, 2003. Proceedings. 2003 International Conference on*, pages 191–196, 2003.

[26] S. Wolfram. Theory and applications of cellular automata. 1986.

[27] T. Yang. *Cellular Neural Networks and Image Processing*. Nova Science Publishers, 2002.

[28] T. Yang and L.B. Yang. Fuzzy cellular neural network: A new paradigm for image processing. *International Journal of Circuit Theory and Applications*, 25(6):469–481, 1998.