Parallel Implementations of Cellular Automata Algorithms on the AGILA High Performance Computing System

Rafael P. Saldaña, Winfer C. Tabares, and William Emmanuel S. Yu
Mathematics Department, Ateneo de Manila University
Quezon City 1101, PHILIPPINES
E-mail: RAF@admu.edu.ph
Tel./Fax.: (+63 2) 426-6125

Abstract: We implemented four basic cellular automata (CA) algorithms on a Beowulf cluster with 8 processors. The CA algorithms are, namely, (1) Game of Life, (2) Greenburg-Hasting, (3) Cyclic Space, and (4) Hodgepodge Machine.
Keywords: cellular automata, Beowulf cluster, AGILA HPCS, parallel computing, Game of Life, Greenburg-Hasting, Cyclic Space, Hodgepodge Machine

Introduction

Computational simulation has become a third approach - along with theory and laboratory simulation - to studying and solving scientific problems. Based on the use of high-performance computers to model and simulate complex systems, computational simulation makes use of a virtual laboratory in which researchers can build a model for a given problem and run it under varying conditions.

Nowadays, high-performance parallel computers are giving researchers the ability to implement inherently parallel techniques such as cellular automata, neural networks, and genetic algorithms – significant mathematical models for describing complex physical phenomena [11].

Cellular automata are discrete space-time models that can be used to model many complex systems in the universe. Cellular automata have been used to model biological systems form the levels of cell activity to the levels of clusters of cells and populations of organisms. In chemistry, they have been used to model the kinetics of molecular systems and crystal growth. In physics, they have been used to study the dynamical systems as diverse as the interaction of particles and the clustering of galaxies. In computer science, cellular automata have been used to model parallel processing and the von Neumann (self-reproducing) machines.

One of the advantages of cellular automata models is that they provide a simplified description of physical systems, in which only some essential features of the real world are taken into account. Numerically, cellular automata are interesting because they are simple and exact, .e., they do not involve rounding nor truncation of floating point numbers. Another advantage of cellular automata is that several investigations have shown that cellular automata are a good alternative to differential equations.

Although the concept of cellular automata has been introduced as early as 1948 by John von Neumann, it is only recently that CA has received increased interest in the high-performance computing community.

In this paper we discuss some basic concepts of cellular automata. We also discussed the implementation of four basic CA algorithms on the AGILA High Performance Computing System, and 8-node Beowulf cluster located at the High Performance Computing and Networks Laboratory of the Ateneo de Manila University.

Cellular Automata

Basic Components

Informally, a cellular automaton (CA) is a dynamic system of discrete lattice sites whose behavior is completely specified in terms of a local relation. Starting with various initial values, these sites evolve in discrete time steps as each site assumes a new value based on the values of some neighborhood of sites and a finite number of previous time steps [7].

It can be thought of as a *stylized universe*:

- Space is represented by a uniform lattice.
- Each cell contains some data and is connected to other cells in some particular pattern.
- Time advances in discrete steps.
- The state of each cell is governed by a set of laws that are applied at each time step.

A **cell** is the basic component of CA. In most CA models, a cell takes the shape of a square although triangles and hexagons are also used as cells in some CA models. To simplify the discussion, we will refer to a square cell whenever the term *cell* is used. Each cell can take any of the available states. The simplest CA model uses binary states where each cell

takes the value of either 1 or 0.

A **lattice** is an arrangement of cells. Technically, a CA lattice can take an infinite number of cells but most applications requires making the number of cells finite. In a one-dimensional lattice, the cells are arranged in a line or a $1 \times n$ matrix. Similarly, cells are arranged in an $m \times n$ matrix in a two-dimensional lattice. A lattice starts at an initial configuration where each cell in the lattice is given an initial state. At every timestep, the lattice evolves to a new configuration which depends on a set of rules that govern the state of each cell in the succeding timestep.

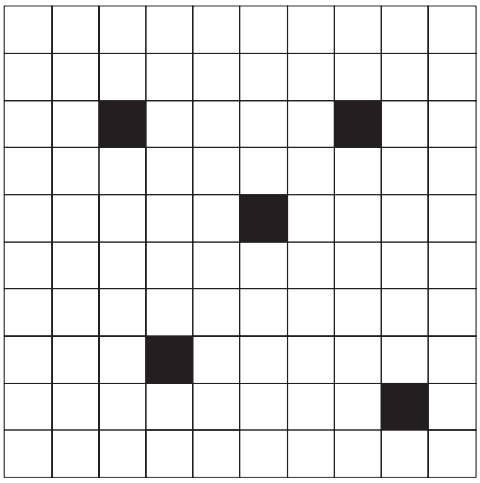

Figure 1: A 10×10 lattice with each cell having a state of either black or white

The general CA rule is that the state of each cell in the next timestep depends on the state of the cell and some other cells in the lattice. The collection of all these cells is called the **neighborhood** of the cell. The type of neighborhood depends on the CA model. The classical neighborhoods for two-dimensional CA are Von Neumann Neighborhood, Moore Neighborhood and the Extended Moore Neighborhood.

Formal Definitions

We now give two formal definitions of a cellular automata - a mathematical definition and a computational definition.

Definition 1 Mathematical Definition

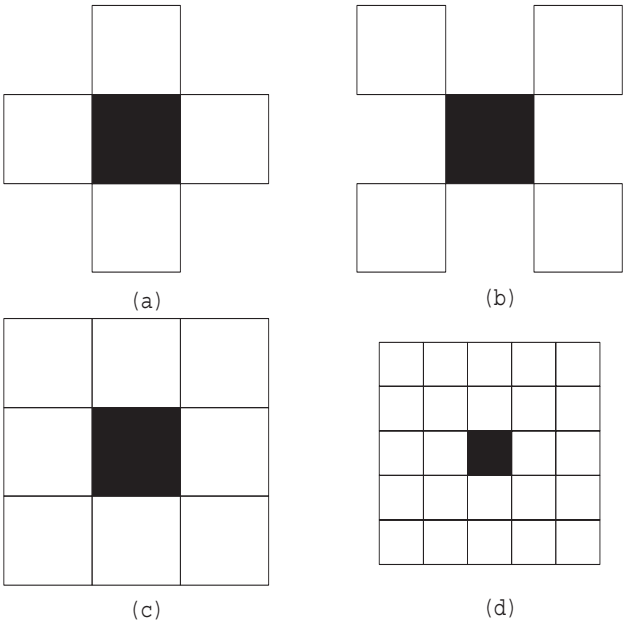

Figure 2: (a)-(b) Von Neumann Neighborhood and its variant (c) Moore Neighborhood (d) Extended Moore Neighborhood

A cellular automaton α is a 4-tuplet (L, S, N, Ψ), where

- $L = \{c(i,j)|i,j \in \aleph, 0 \leq m, 0 \leq n\}$ *is an $m \times n$ lattice with $c(i,j)$ being the cell in the i^{th} row and j^{th} column*
- $S = \{s_1, s_2, \ldots, s_r\}$ *be the set of possible states of a cell in the lattice. The state of a cell $c(i,j)$ at time t is denoted by $s^t(i,j)$.*
- $N(i,j)$ *be the neighborhood of cell $c(i,j)$ at time t. The neighborhood of $c(i,j)$ is composed its state $s^t(i,j)$ and the states of the cells to which it is connected.*
- $\Psi(N(i,j))$ *be the CA-rule which gives the new state $s^{t+1}(i,j)$ of cell $c(i,j)$ at time $t+1$.*

Note that this definition only considers one- and two-dimensional CA. It can be extended to higher dimensions. With this CA definition, we can define the classical neighborhoods as follows:

- **Von Neumann:** $N_{VM}(i,j) = \{s^t(k,l) \in L||k-i| + |l-j| \leq 1\}$
- **Moore:** $N_M(i,j) = \{s^t(k,l) \in L||k-i| \leq 1 \text{ and } |l-j| \leq 1\}$

- **Extended Moore:** $N_{EM}(i,j) = \{s^t(k,l) \in L || k-i| \leq 2 \text{ and } |l-j| \leq 2\}$

Definition 2 Computational Definition

A Cellular Automaton is a computer program in which the following are performed in the following order [7, 8]:

- *A matrix is constructed with a finite set of specific values (integer, real, symbol or list)*
- *A function, or a set of functions, is defined. The function(s) can be used to change the value of a matrix element, based on the values of the element and nearby elements.*
- *The function is applied (repeatedly) to the matrix, each time changing the values of all matrix elements simultaneously.*

Wolfram's One-Dimensional CA

Stephen Wolfram studied extensively on one-dimensional CA. His work involved a finite $1 \times n$ lattice with periodic boundary conditions. This means that the first lattice site is connected with the last lattice site to form a circle. The CA has a binary state with each cell taking the value of either 0 or 1. The neighborhood of a cell consists of the cell itself, the cell to its immediate left and the cell to its immediate right. There are eight different neighborhoods; namely, 111,110,101,100,011,010, 001 and 000. To determine the state of the cell at the next time step, the update procedure uses a *lookup table* that gives the rule. Since there are eight neighborhoods and each neighborhood can result in two possible state change, there will be $2^8 = 256$ different state change rules. Each rule is given a number called the *Wolfram number*. This is obtained by getting the value of each of the eight neighborhoods and combining them to form a binary number which is then converted to its decimal equivalent. For example, the table below gives "rule 86" ($01010110_2 = 84_{10}$).

111	110	101	100	011	010	001	000
0	1	0	1	0	1	1	0

The following figure represents the application of "rule 86" on a 13-cell CA.

In 1984, Stephen Wolfram presented the results of his explorations in the possible configurations that a CA model could evolve to. He made a qualitative classification of a CA behavior based on the pattern that they generated. The four classes of behavior as described by Wolfram are [3, 10]:

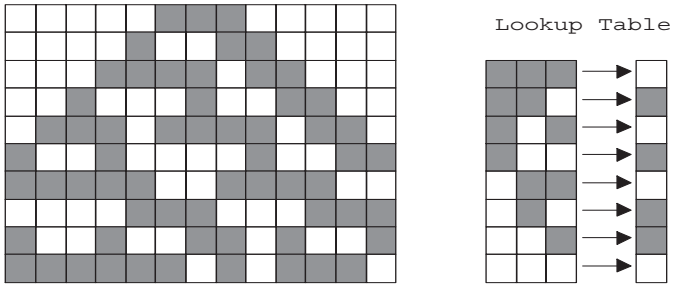

Figure 3: Application of Rule 86

- *Class 1*: This type of automata evolve to a fixed configuration after a finite period of time steps. In a CA with binary states, this will result in a configuration with all 1's (or 0's).
- *Class 2*: This type of automata evolve to a configuration whose patterns are either periodic cycle of configurations or fixed configurations.
- *Class 3*: CA's of this type evolve to chaotic aperiodic patterns. The patterns created are not necessarily random. They are often self-organizing patterns that can be seen in some fractal patterns.
- *Class 4*: CA's of this type evolve to complex structures that are often long-lived. In certain cases, the complexity of this class suggest that they are capable of universal computation.

Cellular Automata Models

Game of Life

In the late 1960's a British mathematician, John Conway, explored on a possible system capable of universal computation. After several years of research and experimentation, he created a system that is now currently known as the "Game of Life". The system is composed of a rectangular grid where each cell can have a value of either 0 or 1. A cell with a value of 1 is said to be *"alive"*. A cell with a value of 0 is said to be *"dead"*. The neighborhood used is the Moore neighborhood. The system is updated based on the following transition rules known as the **life and death** rules:

- **Survival Rule:** A cell that is alive remains alive if it has exactly two or three neighbor cells that are alive.

- **Birth Rule:** A cell that is dead becomes alive if it has exactly three neighbor cells that are alive.
- **Death Rule:** All other cells remain dead or die.

Mathematically, the Game of Life CA can be defined as follows:

Definition 3 .
The Game of Life CA is composed of an $m \times n$ lattice, L, a state set $S = \{0, 1\}$ and a Moore neighborhood $N_M(i, j)$. Let

$$\delta = \sum_{r=-1}^{1} \sum_{p=-1}^{1} s^t(i + p, j + r)$$

The transition function is defined by

$$s^{t+1}(i,j) = \Psi(N_M(i,j)) = \begin{cases} 1 & , \quad if \quad s^t(i,j) = 1 \ and \ \delta = 3 \ or \ 4 \\ 1 & , \quad if \quad s^t(i,j) = 0 \ and \ \delta = 3 \\ 0 & , \quad otherwise \end{cases}$$

Let us look at an example. We start with an 8×8 lattice with some initial state.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	2	3	2	1	0	0
0	1	3	4	3	1	0	0
0	1	3	4	3	1	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 4: Starting state of a Life CA

In the figure, a shaded cell represents a living cell. All others are dead. The number inside the cell represents the number of living neighbors of that cell. With the initial state of the system and applying the transition rules of the Life CA, we have the corresponding states for the 2nd until the 5th generation (Fig 2.4).

0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0
0	1	3	4	3	1	0	0
0	2	5	7	5	2	0	0
0	2	4	6	4	2	0	0
0	1	2	3	2	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

2nd Generation

0	0	0	0	0	0	0	0
0	1	2	3	2	1	0	0
0	1	2	3	2	1	0	0
0	2	3	5	3	2	0	0
0	1	2	3	2	1	0	0
0	1	2	3	2	1	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0

3rd Generation

0	0	1	1	1	0	0	0
0	0	2	2	2	0	0	0
0	1	3	4	3	1	0	0
0	1	3	4	3	1	0	0
0	1	3	4	3	1	0	0
0	0	2	2	2	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0

4th Generation

0	0	0	0	0	0	0	0
0	1	2	3	2	1	0	0
0	2	3	5	3	2	0	0
0	3	5	8	5	3	0	0
0	2	3	5	3	2	0	0
0	1	2	3	2	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

5th Generation

Figure 5: 2nd to 5th generation of the given Life CA

From any given initial state, a Life CA can end in three possible states [4]:

- All cells becoming dead.

- The system becoming a steady state and becomes fixed thereafafter.
- The system entering an oscillating phase which repeats itself for some period.

Excitable Cellular Automata

Another common CA algorithm is the **excitable cellular automata**. This CA is usually used to study excitable media. Excitable media are extended non-equilibrium systems having uniform states that are linearly stable but susceptible to finite perturbations [8]. Excitable media are used to model physical and chemical systems such as oscillating chemical reactions (e.g. Belousov-Zhabotinski reaction),transmissions in neural networks, wave dynamics, slime mold self-organization and others [8, 12].

These systems are composed of a lattice of cells with three basic states: excited, recovering and relax. The recovering state can have several defined stages. A cell can either remain in its current state or move to next defined state depending on the state of its neighbor cells.

There are two types of excitable CA algorithms: the **Greenburg-Hasting**,**Cyclic Space** and **Hodgepdge Machine** algorithm. In the Greenburg-Hasting algorithm, the CA sytems is updated using the following rules:

- A cell in rested state becomes excited if a certain number of neighbor cells is excited.
- A cell in recovery state moves to the next recovery state.
- All other cells remains at its current state.

In the Cyclic Space algorithm, the update rules are as follows:

- A cell in state s moves to state $s + 1$ if a certain number of neighbor cells is in state $s + 1$
- All other cells remains at its current state.

The Hodgepodge Machine model makes use of an $s \times s$ square lattice with periodic boundary conditions. The lattice sites (or cells) have values ranging from 0 to $\kappa - 1$. A cell is said to be healthy if it has a value 0. It is said to be ill if the cell value is $\kappa - 1$. All other cells (those that have non-zero values less than the maximum value) are said to be infected. Its update rules are as follows:

- If a cell is ill then it becomes healthy.

- If a cell is healthy, it becomes infected and its value changes to an integer value determined by the formula

$$\text{Min}\left[\kappa - 1, \left\lfloor \frac{Infected}{k_1} \right\rfloor + \left\lfloor \frac{Ill}{k_2} \right\rfloor\right]$$

where k_1 and k_2 are infection constants. The Min[ ] function is used to ensure that the value will not exceed $\kappa - 1$. $Infected$ refers to the number of infected nearest neighbor cells and Ill refers to the number of ill nearest neighbor cells.

- If a cell is infected, it becomes more infected and its value is determined by

$$\text{Min}\left[\kappa - 1, g + \left\lfloor \frac{Sum}{Infected} \right\rfloor\right]$$

where g is the rate of infection constant and Sum refers to the sum of neighborhood cell values.

Formally, the Greenburg-Hastings and Cyclic Space algorithms can be defined as follows:

Definition 4 .
The Excitable CA model is composed of an $m \times n$ lattice, L, a state set $S = \{0, 1, \ldots, \kappa - 1\}$, a threshold constant θ and a Von Neumann neighborhood $N_{VN}(i, j)$. A cell in state 0 is said to be in excited state while a cell in $\kappa - 1$ state is in rested state. All other states are the stages in the recovery state. Let

$$\begin{aligned} \delta_{GH} &= \#\{c(k,l) \in N_{VN}(i,j) | s^t(k,l) = 1\} \\ \delta_C &= \#\{c(k,l) \in N_{VN}(i,j) | s^t(k,l) = (s^t(i,j) + 1) \mod \kappa\} \end{aligned}$$

where $\#\Delta$ denotes the cardinality of set Δ.

The transition function of the Greenburg-Hasting model is defined by

$$s^{t+1}(i,j) = \Psi(N_{VN}(i,j)) = \begin{cases} \kappa & , \ \textit{if} \quad \delta_{GH} \geq \theta \\ (s^t(i,j) + 1) \mod \kappa & , \ \textit{if} \quad 1 \leq s^t(i,j) \leq \kappa - 2 \\ s^t(i,j) & , \ \textit{otherwise} \end{cases}$$

The transition function of the Cyclic Space CA is defined by

$$s^{t+1}(i,j) = \Psi(N_{VN}(i,j)) = \begin{cases} (s^t(i,j) + 1) \mod \kappa & , \ \textit{if} \quad \delta_C \geq \theta \\ s^t(i,j) & , \ \textit{otherwise} \end{cases}$$

AGILA HPCS AND SOFTWARE TOOLS

The implementations of the four basic CA algorithms was done on the AGILA High Performance Computing System. LAM-MPI was used as the message passing library. All the codes were written in the C programming language. Visualization was done using MPE.

AGILA Cluster

The cluster computer used for the computation is the AGILA HPCS, a Beowulf cluster housed in the High Performance Computing and Networking Laboratory of the Ateneo de Manila University.

The system was built by the Ateneo High Performance Computing Group using commodity hardware and free or open source software (e.g,. Linux). Message passing libraries (e.g. Parallel Virtual Machine (PVM) and Message Passing Interface (MPI)) are utilized to simplify the task of creating applications on the AGILA HPCS. It has a peak performance of 3.16 GFlops based on the High Performance LINPACK benchmarks.

LAM-MPI

To generate parallel codes used for the cluster, a message passing library was utilized. The Message Passing Interface (MPI) is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementors, and users. The message passing library used in the CA computation is the Local Area Multicomputer (LAM) MPI. It is an MPI programming environment and development system for heterogeneous computers on a network. With LAM, a dedicated cluster or an existing network computing infrastructure can act as one parallel computer solving one problem. LAM features extensive debugging support in the application development cycle and peak performance for production applications.

It also features a full implementation of the MPI communications standard.

MPE

To visualize the simulations, the Message Passing Environment (MPE) was used. The MPE was installed on the AGILA cluster and compiled to utilize LAM-MPI as its default message passing interface.

The MPE extensions provide a number of useful facilities for MPI programmers. These include several profiling libraries to collect information on MPI programs and log files for *post-mortem* visualization and real-time animation. Also included are routines to provide simple X windows system graphics to parallel programs .

The MPE may be used with any implementation of MPI. It enables users to call a set of functions and routines in order to perform the visualization tasks that are needed. This visualization is critical since it enables the researchers to perform the necessary analysis of the model. Using MPE is a better alternative than writing custom built visualization routines or using numerical plain text outputs.

Pseudo-Random Number Generator

To obtain a substantially random initial state for the CA model, an external pseudo-random number generator was used. We used a generator that was developed by the pLab team at the University of Salzburg . The generator came out with a library of pseudo-random number generators called PRNG, which is available for download at http://random.mat.sbg.ac.at/. The Explicit Inversive Congruential Generator (EICG) algorithm in PRNG was used to generate the random numbers for the initial states of the CA. The EICG was selected due to its excellent splitting properties.

ICG methods such as EICG differ strongly from other linear methods in its geometrical structure. According to Marsaglia and Eichenauer-Herrmann the ICG methods are less prone to biases compared to other methods.

Parallel Implementation

The following steps were used to implement the four CA models:

1. Obtain the rules and parameters of the algorithm.
2. Write the pseudo-code for the algorithm in serial.
3. Parallelize codes using a data-parallel paradigm.
4. Code the implementation in C using LAM-MPI libraries.
5. Use MPE for visualization.
6. Capture results using XVidcap and Xwd capturing utilities for X windows.

Parallelization Issues

The CA models (Game of Life, Greenburg-Hasting, Cyclic Space, Hidgepodge Machine) can be parallelized on N number of processes. Multiple processes can run on a single compute node. The approach taken for this problem was to use data-parallelization. This entails dividing the CA matrix into smaller matrices and distributing it across the different running processing elements.

To determine the portion of the CA matrix that belongs to a single process, the algorithm is defined by Equation 1, where m is the width of the matrix that belongs to that particular process, M_w is the width of the entire CA matrix, N is the number of processes spawned for the current job and R is the rank of the current process.

$$m = \frac{M_w}{n + \alpha} \tag{1}$$

The offset from the left most portion of the entire CA matrix is be defined by Equation 2, where o is the offset of the individual matrix from the entire CA matrix.

$$o = R \times \frac{M_w}{N} \tag{2}$$

where $\alpha = 1$ if $R < M_w \mod N$ and $\alpha = 0$ if $R \geq M_w \mod N$

The matrix strip defined by Equation 1 and Equation 2 is the matrix of the process with rank R in a pool of N number of processes. Each process operates only on this assigned matrix strip and will then allocate memory for a matrix M^P and M^T that is of the size m. M^P is to be used as the main processing matrix. M^T is a temporary storage matrix for miscellaneous operations. Both M^P and will contain the data subset defined by Equation 1 and Equation 2 per process. For each timestep, data from M^P is processed and the new values are stored in M^T. The values of M^T are then swapped with the values of M^P after the entire matrix strip is processed. This method prevents overlapping that may take place if only one matrix were used.

Another issue that was considered was that of the boundary information needed to be propagated. This was done by following the procedure below:

1. Transfer the contents of $M^P[1]$ to the process defined by $R - 1$.
2. Receive the contents of $M^P[0]$ from the process defined by $R - 1$.
3. Transfer the contents of $M^P[m]$ to the process defined by $R + 1$.
4. Receive the contents of $M^P[m + 1]$ from the process defined by $R + 1$.

Two boundary vectors on both sides of the current compute node's M^P are propagated to its neighboring compute nodes.

Results

In this section, we present some representative CA simulations on the AGILA HPCS using the following algorithms: Game of Life, Cyclic Space, Hodgepodge Machine and Greenburg-Hastings.

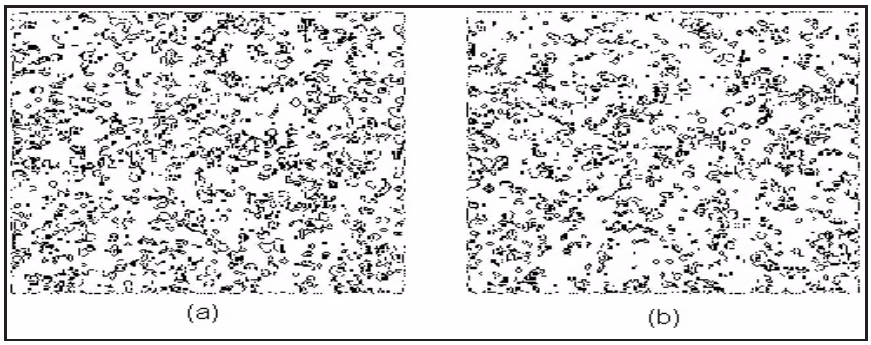

Figure 6: Computer simulations using a 200×200 Game of Life CA model with 10 states and run in 100 iterations. (a) $time = 5$ (b) $time = 50$

Conclusion

This study has shown that cellular automata algorithms such as the Game of Life, Greenburg-Hastings, Cyclic Space and Hodgepodge Machine can be implemented in parallel on the AGILA High Performance Computing System.

Acknowledgement

This project was funded in part by the Philippine Commission on Higher Education (CHED) Center of Excellence in Science and Mathematics Grant of the Ateneo de Manila University.

References

[1] Marianne Delorme, "An Introduction to Cellular Automata," *Cellular Automata: a Parallel Model, Mathematics and Its Applications*, Kluwer.

[2] Palash Sarkar, "A Brief History of Cellular Automata", *ACM Computing Surveys*, Vol. 32, No. 1, March 2001.

[3] Paul Torrens, "How Cellular Models of Urban Systems Work," Centre for Advanced Spatial Analysis Working Paper Series, University College London, November 2000.

[4] Mirek Wjtowicz. *What is Life and Cellular Automata?,* http://www.mirwoj.opus.chelm.pl/whatis_life.html

[5] R. Gaylord and P. Wellin. *Computer Simulations with Mathematica: Explorations in Physical, Biological and Social Sciences,* New York: TELOS, 1994.

[6] R. Saldaña, F. Muga II, J. Garcia, W. Uy, "Development of a Beowulf-Class High Performance Computing System for Computational Science Applications," Accepted for Publication in Science Diliman, 2001.

[7] Richard J. Gaylord and Kazume Nishidate. *Modeling Nature: Cellular Automata Simulations with Mathematica,* Springer-Verlag, New York, Inc., 1996.

[8] Rafael P. Saldaña and William Emmanuel S. Yu. "Cellular Automata Explorations on a Beowulf Cluster," *Proceedings of the Cellular Automata Symposium (CA 2001)*, Yokohama, Japan, 2001.

[9] Stephen Wolfram. "Universality and Complexity in Cellular Automata," *Physica D*, 10, 1-35.

[10] Alexander Schatten. *Cellular Automata: Digital Worlds,* http://www.ifs.tuwien.ac.at/ aschatt/info/ca/ca.html

[11] D. Talia. "Cellular Processing Tools for High-Performance Simulation," *Computer*, 33, 9, 44-52, September 2000.

[12] R. Fisch, J. Gravner and D. Griffeath. "Threshold-Range Scaling of Excitable Cellular Automata," *Statistics and Computing*, 1991.

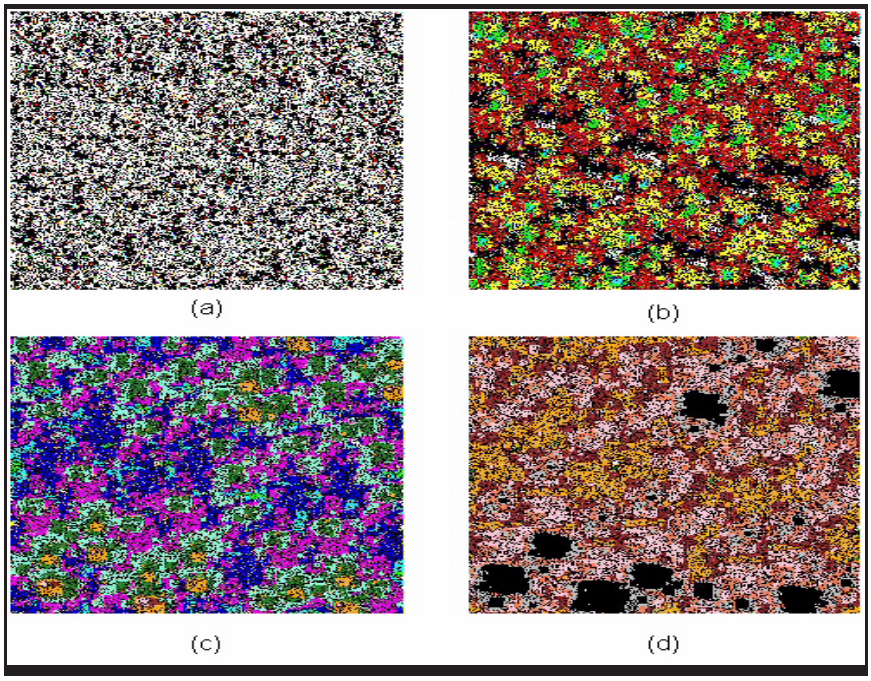

Figure 7: Computer simulations using a 300×300 Cyclic Space Model CA model with 50 states and run in 100 iterations. (a) $time = 5$ (b) $time = 50$ (c) $time = 55$ (d) $time = 60$

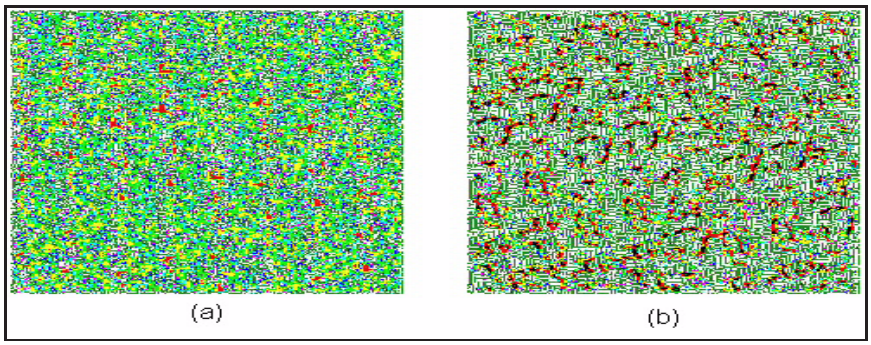

Figure 8: Computer simulations using a 300×300 Hodgepodge Machine CA model with 10 states and run in 100 iterations. (a) $time = 5$ (b) $time = 50$

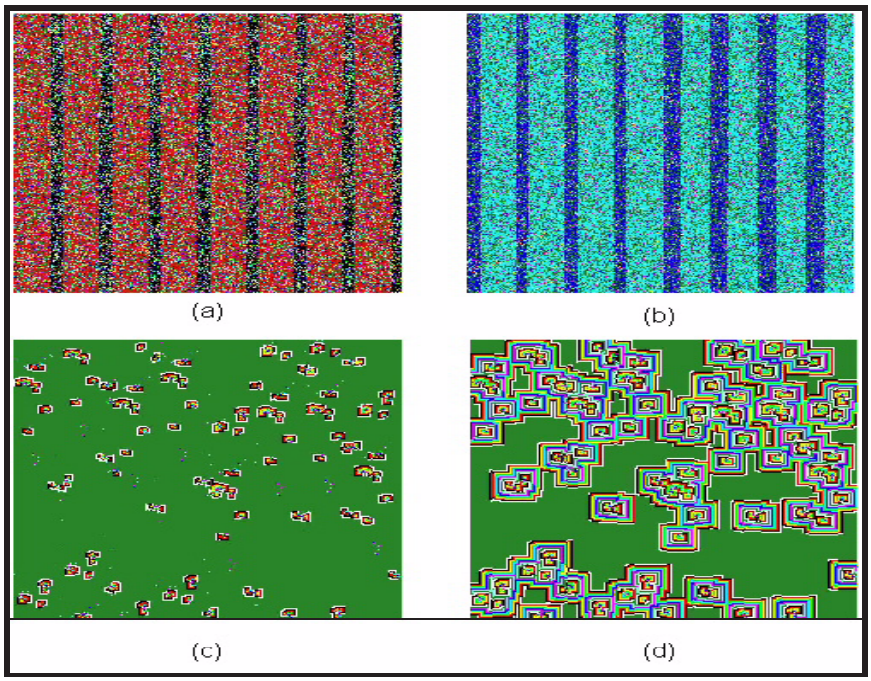

Figure 9: Computer simulations using a 300×300 Greenburg-Hastings CA model with 10 states and run in 100 iterations. (a) $time = 3$ (b) $time = 5$ (c) $time = 50$ (d) $time = 55$