Parallel Processing Letters, Vol. 17, No. 4 (2007) 349–361


World Scientific
www.worldscientific.com

# PARALLELIZING THREE DIMENSIONAL CELLULAR AUTOMATA WITH OpenMP

SANTIAGO GARCIA CARBAJAL
*Computer Science Department, University of Oviedo, Campus de Viesques, Despacho 1.b.15. Gijon, ASTURIAS, Spain*

Received July 2007
Revised October 2007
Communicated by Colin Stirling

ABSTRACT

This paper describes our research on using Genetic Programming to obtain transition rules for Cellular Automata, which are one type of massively parallel computing system. Our purpose is to determine the existence of a limit of chaos for three dimensional Cellular Automata, empirically demonstrated for the two dimensional case. To do so, we must study statistical properties of 3D Cellular Automata over long simulation periods. When dealing with big three dimensional meshes, applying the transition rule to the whole structure can become a extremely slow task. In this work we decompose the Automata into pieces and use OpenMp to parallelize the process. Results show that using a decomposition procedure, and distributing the mesh between a set of processors, 3D Cellular Automata can be studied without having long execution times.

*Keywords*: Genetic Programming, Cellular Automata Parallelization, OpenMp.

## 1. Introduction. Cellular Automata

Cellular automata were originally conceived by Ulam and von Neumann in the 1940s to provide a formal framework for investigating the behavior of complex, extended systems [11], [9]. They are dynamical structures in which space, time, and the states of the system are discrete. Each cell in a regular lattice changes its state with time according to a rule which is local and deterministic. All cells in the lattice obey the same rule, and their state is determined by the previous states of a surrounding neighborhood of cells [12], [10].

The infinite or finite cellular array is n-dimensional, where n=1,2,3 is used in practice. The identical rule contained in each cell is essentially a finite state machine, usually specified in the form of a rule table (also known as the transition function), with an entry for every possible neighborhood configuration of states. The neighborhood of a cell consists of the adjacent cells. For one-dimensional CAs, a cell is connected to r local neighbors on either side, where r is a parameter referred to as the radius. Thus, each cell has 2r+1 neighbors, including itself. The value $a_i^t$

of a site at position i changes according to $\oint$.

$$a_i^{(t+1)} = \oint (a_{i-r}^{(t)}, a_{i-r+1}^{(t)}, .., a_{i+r}^{(t)}) \tag{1}$$

The local rule $\oint$ has a range of $r$ sites. Its form determines the behavior of the cellular automaton.

For two-dimensional CAs, two types of cellular neighborhoods are usually considered:

- von Neumann neighborhood: The von Neumann neighborhood of range $r$ is defined by
$$N^V(x_0, y_0) = \{(x, y) : |x - x_0| + |y - y_0| \leq r\} \tag{2}$$
See figure 1 (left).

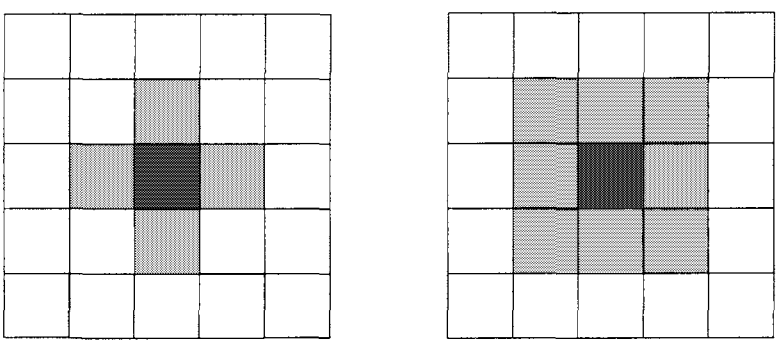

Fig. 1. Von Neumann Neighborhood (left). Moore Neighborhood (right).

- Moore neighborhood: The Moore neighborhood of range $r$ is defined by
$$N^M(x_0, y_0) = \{(x, y) : |x - x_0| \leq r, |y - y_0| \leq r\} \tag{3}$$
See figure 1 (right).

When considering a finite-sized grid, spatially periodic boundary conditions are frequently applied, resulting in a circular grid for the one-dimensional case, and a toroidal one for the two-dimensional case. The term configuration refers to an assignment of states to cells in the grid.

For this work, we will be using a 3D extension of Moore Neighborhood, and non periodic boundary conditions. The radius value is 1, which results on a neighborhood of size 27 (26 plus the central cell) for each position in the grid. Finally, the size of our automata will be [100]x[100]x[100]. The natural extension of Moore Neighborhood (see figure 2, right) can be defined as

$$N^M(x_0, y_0, z_0) = \{(x, y, z) : |x - x_0| \leq r, |y - y_0| \leq r, |z - z_0| \leq r\} \tag{4}$$

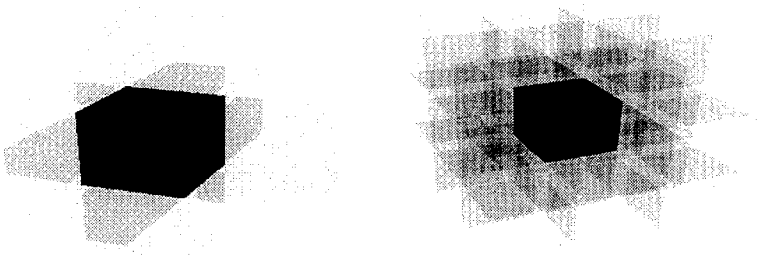

Fig. 2. von Neumann (left) and Moore (right) neighborhoods: 3D extension.

### *1.1. Generation of transition rules for Cellular Automata with Genetic Programming*

#### 1.1.1. Langton parameter

According to Stephen Wolfram [12], the patterns generated in the evolution of Cellular Automata from disordered initial states can be grouped into four general classes:

(i) Evolves to homogeneous state (Class I).
(ii) Evolves to simple separated periodic structures (Class II).
(iii) Yields chaotic aperiodic patterns (Class III).
(iv) Yields complex pattern of localized structures (Class IV).

In class I and II, and almost in all class III automata, information cannot be transmitted between cells. Class I and II present too much inter-cell dependence (two much order), and class III presents too little inter-cell dependence (too much disorder).

In [6] Christopher Langton, proposes an algorithm to construct Cellular Automata with "interesting" behavior. In short, Cellular Automata are interesting when their global behavior is more than the sum of the behaviors of their individual parts. The cells in interesting C.A. must interact cooperatively in some way in order to support global dynamics of the system. To do so, they must communicate information between themselves in a meaningful manner.

Langton proves that the four different classes of Cellular Automata observed by Stephen Wolfram are grouped as a function of what is known as **Langton Parameter**, $\lambda$. If we let $P(\Re)$ be the percentage of transitions to state $\Re$ in the rule table of any Cellular Automata, then we can define the parameter lambda as:

$$\lambda = 1.0 - P(\Re) \tag{5}$$

In a Cellular Automaton with two possible states, $\lambda$ is bounded above by 0.5, as it is the probability of each cell of being activated at any time. Figure 3 shows

the situation of the different classes of C.A. with respect to the value of $\lambda$, and a figuration of the existent "order" in the grid, decreasing as $\lambda$ tends to 0.5. There is some empiric evidences of the existence of a "Limit of Chaos" value for Cellular Automata somewhere in $[0.3 - \epsilon : 0.3 + \epsilon]$.

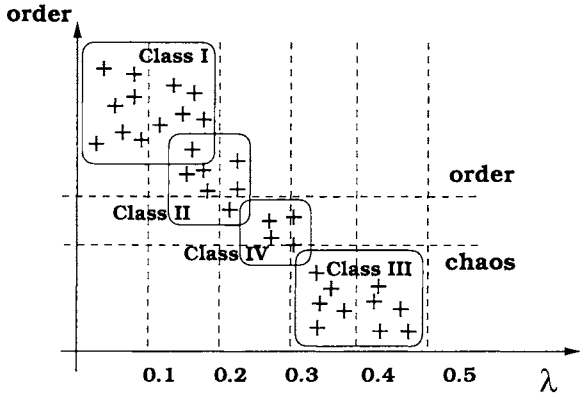


Fig. 3. Cellular Automata Classification according to Langton Parameter.

This research is part of a work where we are trying to determine the existence of a "Limit of Chaos" in 3D Cellular Automata. For such a goal, we can approximately calculate the value for Langton Parameter, by simply generating a high number of test cases, and counting the number of positive and negative evaluations of the rule that defines the automaton. Consequently, we can use the value of $\lambda$ as Fitness Function in a standard Genetic Programming approach [5] ($|P(a) - \lambda|$, actually. Explained in section 1.2).

Once we have obtained Cellular Automata with any desired $\lambda$, we need to make long simulations of the behavior of the automata, in order to calculate statistical properties for each different $\lambda$. Our study is based in the relation between Langton Parameter, Entropy and Mutual Information [8]. Here arises the need for a fast implementation of transition rules.

### *1.2. Genetic Programming*

Genetic programming (GP) is a generic term used to mean an evolutionary computation system which is used to evolve programs. Early forms of GP can be traced back to Friedberg ([3]) and Cramer ([2]). The first GP system to bear the name 'Genetic Programming' was devised by John R. Koza ([5]), and forms the basis of conventional GP systems.

Koza's genetic programming represents programs by their parse trees. A parse tree is a tree-structure which captures the executional ordering of the functional components within a program: such that a program output appears at the root node; functions are internal tree nodes; a function's arguments are given by its child nodes; and terminal arguments are found at leaf nodes. A parse tree is a

particularly natural structure for representing programs in LISP, the first used language for genetic programming. This is one reason why the parse tree was chosen as a representation for genetic programming. A problem, in GP, is specified by a fitness function, a function set, and a terminal set. The function and terminal sets determine from which components a program may be constructed; and the fitness function measures how close a particular program's outputs are to the problem's required outputs. The initial population is filled with programs constructed randomly from components in the function and terminal sets.

1.2.1. Obtaining any desired $\lambda$ with genetic programming

In order to obtain populations of Cellular Automata with any desired value for Langton's Parameter we use our own standard Genetic Programming System (the one explained in [4]). Parameters are listed in table 1. After setting the initial configuration of the GP system, we obtain transition rules whose probability of activation for any cell ($\lambda$) is close to the desired one, in most cases. These large populations of parse trees are the material we will be using in the future to study the relationship between $\lambda$ and chaos in 3D Cellular Automata.

In [7], Christopher Langton developed the concept of "Edge of Chaos" while trying to characterize the space of Cellular Automata. He found that raising the value of parameter $\lambda$ (the proportion of transitions to an active state), one could obtain all types of Cellular Automata. From frozen at a fixed point, to chaotic ones. This is very analogous to Stephen Wolfram C.A. Classification ([12], [13]). Additionally, it is well known that a system capable of computation must be able of:

1. Information Storage, and
2. Information Transmission

Cellular Automata capable of universal computation are said to be in Class IV. Classes seem to be passed in order I, II, IV, III when rising $\lambda$. The main problem is: *"to determine whether a rule table is capable or universal computation or not"*. We will use a Genetic Programming approach to obtain transition rules in the form of syntactic trees, whose value of Langton Parameter, $\lambda$, will be determined by the evolutionary process. To do this, as in any other GP procedure, we must define:

1. The terminal set, $T$.
2. The function set, $F$.
3. The goal, and a fitness function capable of evaluate the performance of any valid individual.
4. The set of parameters of the algorithm.
5. The method for designating a solution and the criterion for terminating a run.

Genetic Programming [5] is directly applicable to the generation of transition rules for Cellular Automata, as the evaluation of a GP tree formed by logical operations in the internal nodes, with variables representing the neighbors of a cell in the

terminal ones, can be used as a way to obtain each cell's next state as a function of the neighbors' previous ones. See figure 4.

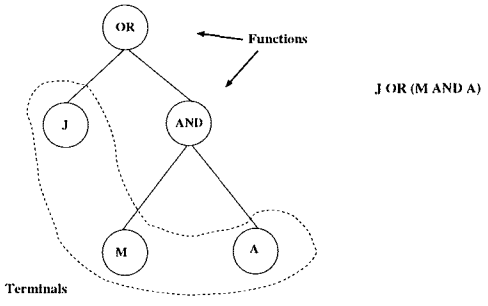


Figure 4: Syntactic tree and meaning of a transition rule.

### *1.3. Terminal set*

The first step when preparing to use Genetic Programming is to identify the set of terminals. The terminal symbols are the inputs to the as-yet-undiscovered computer program. When using 3D extension of Moore Neighborhood, (see figure 2, right), we have a neighborhood of 27 cells. We name each one of the cells in the neighborhood with a capital letter (from A to Z), being **x** the central cell. Each one of these letters can be a terminal node in the trees that we will search for with our Genetic Programming system, as can be seen in figure 4.

### *1.4. Function Set*

After identifying the terminal set, the second step is to define the set of functions that will be used to generate the mathematical expression that attempts to solve the problem. Each of the functions in the function set should be able to accept, as its arguments, any value and data type that may possibly be returned by any function in the function set and any value and data type that may possibly be assumed by any terminal in the terminal set. That is, the function set and terminal set selected should have the closure property so that any possible composition of functions and terminals produces a valid executable computer program. If we want to obtain Cellular Automata with two possible states (true, false) presenting behaviors depending on the state of any of the cells in a 3D Moore Neighborhood, we need to have logical functions in the internal nodes of the trees. So, we use the following function set:

- AND: a node in the tree is evaluated to true if its two children nodes are evaluated to true.
- OR: a node in the tree is evaluated to true if any of its two children nodes is evaluated to true.
- XOR: a node in the tree is evaluated to true if its two children nodes are evaluated to different values.

- NOT: a node in the tree, with only one children node, is evaluated to true if the latter is evaluated to false.

Terminal symbols and functions will be combined randomly in the construction of the initial population. See figure 4. The tree in the figure represents the boolean function J OR (M AND A). For optimization reasons, the transition rules are stored as Lisp-like ones, with polish inverse notation. The expression in figure 4 is stored this way:

$$OR(J, AND(M, A)). \tag{6}$$

### *1.5. Fitness Function*

In order to let the genetic process end up with a solution to any problem, it is necessary to define the Fitness Function. In the situation we are dealing with, the process, for each logical expression, $S$, is this:

1. Let K be the desired value for Langton parameter, $\lambda$.
2. One hundred thousands different fitness cases are generated (it could be a greater number, as this is just a parameter of the GP system). Each one of these strings consists on 27 0's or 1's randomly generated. Let $F_c$ be the number of cases.
3. $S$ is evaluated on each fitness case. Each one of these evaluations will lead to a transition to true (1) or false (0). Let $K_1$ be the number of evaluations of $S$ that give true as result, and $K_2$ the number of negative evaluations.
4. We define the Fitness Function of a logical expression, $S$ , $F(S)$, as:

$$F(S) = |K - (K_1/F_c)|. \tag{7}$$

This is exactly the meaning of Langton Parameter. This way, we have an estimation of the value of the parameter, that is used by the genetic system to evolve the complete population towards this value. See figure 5.

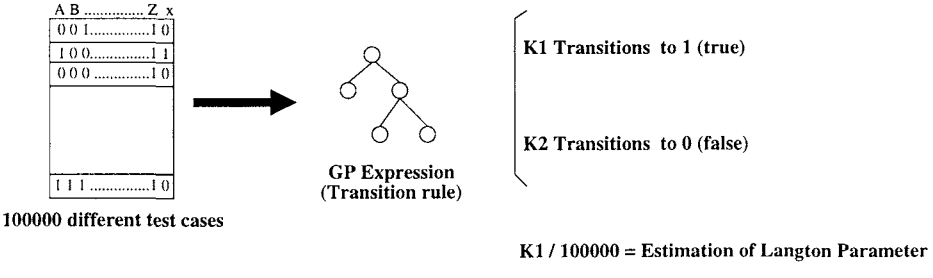


Figure 5: Estimation of $\lambda$ for fitness function calculation.

### *1.6. Parameter Set, and Termination Criterion*

Our goal is to obtain populations with any desired value for Langton Parameter, in order to measure Entropy and Mutual Information at each value of $\lambda$. Because of this, our Termination Criterion is not to have one individual in the population that satisfies $\lambda = K$, but have a complete population with all the individuals as adapted as possible. Because of that, our evolutionary process will stop after a huge number of generations, G, empirically determined, is reached. Population size is M=1000 individuals, and we use a depth limit of 10 levels for the randomly generated individuals included in the initial population. See table 1 for complete set of parameters.

After deciding a way to represent transition rules, our Genetic Programming strategy to obtain expressions with any desired value of Langton Parameter consist of:

1. Generate a random initial population of expressions.
2. Calculate the fitness of each expression (that is, how close is each one of them to the desired value of $\lambda$.
3. Apply genetic operators (reproduction, crossover and mutation) to evolve population.
4. Stop when Termination criterion is satisfied. In this case, when maximum number of generations is reached.

When the genetic process is finished, we have populations with a known approximated value for $\lambda$. Each individual is represented by a boolean function, or tree, whose evolution, and statistical properties can be studied. An example of this boolean expressions is shown below:

| and(not(M),not(and(and(not(B),W),and(K,H)))) |
|---|

## 2. Parallelization

### *2.1. Experimental Platform*

The Sunfire E15K of Edinburgh Parallel Computing Center (EPCC) was used as the experimental platform. It consist of a four CPU front end for developing and compilation, and a backend of forty eight CPUs for execution of jobs. The two domains are running Solaris 9 and have the latest Sun Forte Development package and HPC ClusterTools installed. The Message Passing and Shared Memory programming paradigms are supported.

### *2.2. Parallelization Strategy*

Cellular Automata are inherently parallel, as they evolve in a discrete way, following the same transition rule. Parallelizing a 3D Cellular Automata is so direct as dividing the grid between the number of available processors, and collect the final result when all of them have finished. See figure 6. Process and Processor

are synonyms in this context. If an MPI approach is chosen, each processor will apply the transition rule on a section of the whole Cellular Automaton. If the system is developed using POSIX threads, or OpenMP, each process will do this work, and the main process will break, deliver, and collect the new state of the grid.

In an OpenMp program, the programmer specifies, roughly speaking, what pieces of the code can be run in parallel. The number of processes executing these parallel constructs need not be hardwired. Therefore, adjusting the number of processors at run time can be done transparently. Furthermore, OpenMp's execution model, consisting of a succession of sequential code an parallel constructs, seems to be appropriate for our 3D Cellular Automata schema.

Our parallelization strategy is based on the *OpenMP parallel for construct* [1], whose pseudo code is shown below:

```
– code to be executed only by the master –
#pragma omp parallel for
for (i=0; i<MAX; i++)
{
– the iterations of this loop are divided among all processes –
}
– code to be executed only by the master –
```

What we do is (see figure 6) to break the Cellular Automaton into so many pieces as the number of threads (i.e. processors) we are using. Each one of these pieces is passed to one thread, and after applying the transition rule, the whole automaton is gathered together again. This approach is very sensible to the complexity of the transition rule, but permits us reduce the execution time by a factor of five using ten processors, as we explain in section 3.

## 3. Results

Experiments were carried out using simulation periods of 100, 1000,10000 and 100000 steps. For each simulation period, we used sets of 1, 2, 5, 10, 20 and 25 processors. During the execution of the tests, the maximum number of threads per user was limited to a maximum of 10, but the experiments were performed anyway for 15, 20, and 25 processors.

Table 2 summarizes the results obtained for each number of processors, including Average execution time and Standard Deviation. Figure 7 shows the execution time, and figure 8 shows speedup (left) and efficiency (right) obtained for each experiment.

The first remarkable aspect about the execution time is that the process scales reasonably well up to ten processors; for larger numbers of multi thread execution, the efficiency decays very fast. In this context, it is important to realize that the function included on each cell that forms the tridimensional Cellular Automata can be a very short one, or a large expression, with long execution time due to its recursive nature.

The second thing that comes up immediately is that the speedup is constant for each number of processors, independently of the length of the simulation period. This is not true for larger Cellular Automata ([150]x[150]x[150]), but we don't need

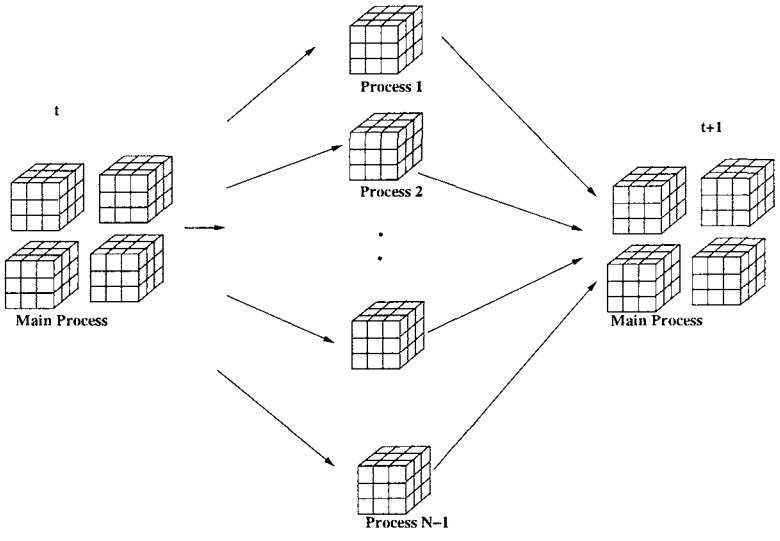


Fig. 6. Parallelization of Cellular Automata evolution.

so many cells for our Entropy calculations.

Table 3 shows a relative measure of the parallel efficiency achieved with each number of processors. To calculate this, we call $S^P$ the pseudo speedup computed with respect to the execution time achieved using only one processor. We will call $E_P$ the corresponding relative efficiency. Again, we can see that we can reduce the execution time by a factor of five in all cases, using ten processors. For higher numbers of processors the relative efficiency goes down under 20%.

Table 1: Induction of Transition rules. Parameter set.

| | |
|---|---|
| Objective | Find Transition Rules with a fixed $\lambda$ |
| Objective | Automata with a fixed Langton Parameter. ($\lambda = K$) |
| Function Set | AND, OR, NOT, XOR |
| Terminal Set | $\{A..Z\} \cup x$ |
| Fitness Cases ($F_c$) | 100000 |
| Fitness Function | $\|K - (K_1/F_c)\|$ |
| Population Size | M=1000 |
| Generations | G=200 |
| Mutation Rate | 0.05 |
| Maximum Initial Depth | 10 |
| Termination Criterion | Maximum number of Generations reached |

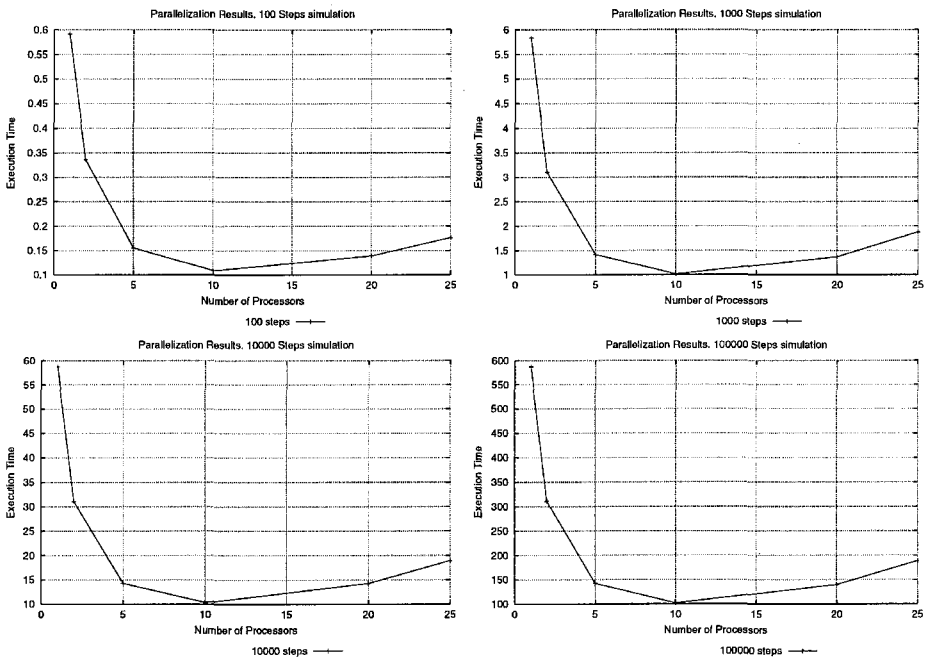


Figure 7: Parallelization Results. Execution Times (mean).

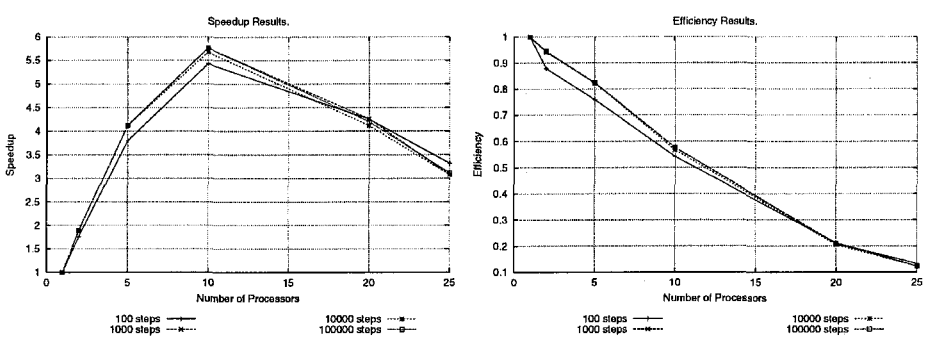


Figure 8: Parallelization Results. Speedup (left). Efficiency (right).

Table 2: Parallelization Results.

| | 100 | | 1000 | | 10000 | | 100000 | |
|---|---|---|---|---|---|---|---|---|
| | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| 1 | 0.591 | 0.0153 | 5.835 | 0.000576 | 58.730 | 0.01577 | 587.3 | 0.110 |
| 2 | 0.336 | 0.0661 | 3.099 | 0.010867 | 31.079 | 0.04798 | 310.878 | 0.407 |
| 5 | 0.155 | 0.0045 | 1.417 | 0.037420 | 14.279 | 0.11885 | 142.404 | 0.640 |
| 10 | 0.108 | 0.0012 | 1.012 | 0.100611 | 10.345 | 0.10635 | 101.929 | 0.649 |
| 20 | 0.139 | 0.0026 | 1.370 | 0.366752 | 14.268 | 0.20432 | 139.903 | 1.832 |
| 25 | 0.178 | 0.0046 | 1.897 | 0.498086 | 19.108 | 0.42188 | 188.698 | 4.645 |

Table 3: Parallelization Results. Efficiency.

| Processors | 100 Steps. CPU time | $S^P$ | $E_P$ |
|---|---|---|---|
| 1 | 0.591 | 1 | 1 |
| 2 | 0.336 | 1.756 | 0.878 |
| 5 | 0.155 | 3.796 | 0.759 |
| 10 | 0.108 | 5.428 | 0.542 |
| 20 | 0.139 | 4.251 | 0.212 |
| 25 | 0.178 | 3.316 | 0.132 |

| Processors | 1000 Steps CPU time | $S^P$ | $E_P$ |
|---|---|---|---|
| 1 | 5.835 | 1 | 1 |
| 2 | 3.099 | 1.882 | 0.941 |
| 5 | 1.417 | 4.115 | 0.823 |
| 10 | 1.012 | 5.763 | 0.576 |
| 20 | 1.370 | 4.258 | 0.212 |
| 25 | 1.897 | 3.074 | 0.122 |

| Processors | 10000 Steps CPU time | $S^P$ | $E_P$ |
|---|---|---|---|
| 1 | 58.730 | 1 | 1 |
| 2 | 31.079 | 1.889 | 0.944 |
| 5 | 14.279 | 4.112 | 0.822 |
| 10 | 10.345 | 5.676 | 0.567 |
| 20 | 14.268 | 4.116 | 0.205 |
| 25 | 19.108 | 3.073 | 0.122 |

| Processors | 100000 Steps CPU time | $S^P$ | $E_P$ |
|---|---|---|---|
| 1 | 587.3 | 1 | 1 |
| 2 | 310.878 | 1.889 | 0.944 |
| 5 | 142.404 | 4.124 | 0.824 |
| 10 | 101.929 | 5.761 | 0.576 |
| 20 | 139.903 | 4.197 | 0.209 |
| 25 | 188.698 | 3.112 | 0.124 |

## 4. Conclusions

We have employed the *parallel for construct*, one of the basic OpenMP features to decompose, evolve, and gather pieces of 3D Cellular Automata. We are using a radius of 1, that implies a neighborhood of 27 cells. For larger neighborhoods the evaluation of recursive expressions, and the modification of the 3D grid would be extremely slow. The use of or approach will let us study the behavior and statistical properties of 3D Cellular Automata using neighborhood radius greater than 1, as we can reduce the execution times by a factor of five. At the moment, and in the machine used to run these experiments, the optimal number of processor is ten.

## 5. Acknowledgments

This work was carried out under the HPC-EUROPA project (RII3-CT-2003-506079), with the support of the European Community - Research Infrastructure Action - under the FP6 "Structuring the European Research Area" Program.

## 6. References

[1] OpenMP Architecture Review Board. *OpenMP C and C++ Application Program Interface.* OpenMP Architecture Review Board, 2005.

[2] Nichael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In John J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, pages 183–187, Carnegie-Mellon University, Pittsburgh, PA, USA, 24-26 July 1985.

[3] R.M. Friedberg. A learning machine: Part i. *IBM J. Research and Development*, 2(1):2–13, 1958.

[4] S. Garcia and F. Gonzlez. Evolutive introns: A non-costly method of using introns in GP. *Genetic Programming and Evolvable Machines*, 2(2):111–122, June 2001.

[5] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.

[6] C. Langton. pages 1–2, 1988.

[7] C.G. Langton. Computation at the edge of chaos: phase transitions and emergent computation. *Physica D*, 42:12–37, 1990.

[8] Claude E. Shannon and Warren Weaver. The mathematical theory of communication. *The University of Illinois Press*, pages 1–2, 1949.

[9] Mosh Sipper. *Machine nature: The Coming Age of Bio-inspired Computing.* McGraw-Hill, 2002.

[10] T. Toffoli and N. Margolus. *Cellular Automata Machines.* The MIT Press, Cambridge, Massachusetts, 1987.

[11] J. von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, Illinois, Illinois, 1966.

[12] S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

[13] S. Wolfram and N. H. Packard. Two-dimensional cellular automata. *J. Stat. Phys.*, 38:901–946, 1985.