# How a SIMD machine can implement a complex cellular automaton? A case study: von Neumann's 29-state cellular automaton

JACQUELINE SIGNORINI

Département d'Informatique
Université Paris-8
2, Rue de la Liberté 93526 Saint-Denis - France

## Abstract

This study is a part of an effort to simulate the 29-state self-reproducing cellular automaton described by John von Neumann in a manuscript that dates back to 1952. We are interested in the programming of very large SIMD arrays which, as a consequence of scaling them up, incorporate some features of cellular automata. Designing tools for programming them requires an experimental ground: considering that von Neumann's 29-state is the only known very large and complex cellular automaton, its simulation is a necessary first step. Embedded in a two-dimensional cellular array, using 29 states per cell and 5-cell neighborhood, this automaton exhibits the capabilities of universal computation and universal construction.

This paper concentrates on the transition rule that governs the complex behavior of the 29-state automaton. We give a detailed presentation of its transition rule, with illustrative examples to ease its comprehension. We then discuss its implementation on a SIMD machine, using only 13 bits per processing element to encode the rule, each processing element corresponding to a cell. Finally, we present experimental results based upon the simulation of general-purpose components of the automaton: pulser, decoder, periodic pulser on the SIMD machine.

This work was partially supported by the PRC-GRECO de Programmation C.N.R.S.



## 1 Introduction

The 29-state cellular automaton was conceived by John von Neumann as an ideal structure for modeling the biological process of self-reproduction. Von Neumann was interested in the general question [26]: What kind of logical organization is sufficient for an automaton to control itself in such a manner that it reproduces itself? He first formulated the question in terms of a kinematic automaton system and later reformulated and solved it in terms of a cellular automaton system. He wished to provide a mathematical, logical model of self-reproduction: if self-reproduction is describable as a logical sequence of steps, i.e. as an algorithm, then there is a Turing machine which can perform its own reproduction [8].

Von Neumann [26] defined a two-dimensional cellular automaton with 29 states per cell and 5-cell neighborhood (the current cell and its 4 orthogonal neighbors). He was able to show that this 29-state cellular model was both computation and construction universal by embedding a universal computer-constructor automaton in it. A universal computer-constructor automaton has the following two capabilities: first, it can perform any computation (simulate any Turing machine) and second, given the description of any *quiescent* automaton, it can construct that automaton in an empty region of the cellular space. Thus, self-reproduction follows as a special case of universal construction where the automaton described on the tape is the universal constructor itself.

A.W. Burks [2,26], J.W. Thatcher [22], C. Lee [9] contributed to greatly clarify the design of von Neumann's 29-state automaton. J.G. Kemeny [7] estimated that more than 200 000 cells were needed to simulate it. In 1968, E.F. Codd [3] proposed an eight-state cellular model which could support a self-reproducing universal computer and constructor. However, neither von Neumann's automaton nor Codd's automaton have actually been simulated.

Since their introduction by von Neumann, cellular automata have been used for image processing [14,16,19] and visual pattern recognition [13,17]. Recently, cellular automata have been extensively used as representative models to analyse the behavior of complex and dynamical systems in mathematics [27,28], in physics [4,6,10,23,24,25], in biology [20,21]. Three factors have resulted in the revival of interest in cellular systems. First, the development of powerful computers have made possible the real-time simulation of cellular automata in a serial or parallel mode of operation. Second, cellular automata have yielded novel insights into classical problems as the ergodic problem in statistical mechanics. They have enough expressive power to represent a variety of discrete systems with local interactions. Third, with the advent of VLSI technology, it has become feasible to construct very large one- and two-dimensional arrays of interconnected processing elements. In one dimension, each processing element is connected to its two immediate neighbors; in two dimensions, a processing element can be connected to four, six or eight neighbors.

As a consequence of this current trend, the simulation of von Neumann's 29-state cellular automaton is becoming an interesting issue to consider. With the coming generation of large-scale single instruction multiple data (SIMD) arrays – supporting massive parallelism and local connections – this simulation is now feasible. The obvious question which then arises is: Why undertake this simulation now? There are two important reasons for this, both concerned with the programming of large-scale SIMD arrays.

First, in the process of designing this automaton on a large scale SIMD array, new tools will be requested for construction, placement and exploration of complex configurations – called organs (see sect. 4) – on the cellular array, as well as for routing transportation and communication lines between them. It is likely that these design tools will, in turn, be adequated for programming existing or near future very large SIMD arrays. Explicit attempts [18] to program the automaton on a SIMD array – the Gapp [15] – have shown that design tools have to be incorporated into hierarchical organization similar to CAD software tools for the design of complex digital circuits.

Second, new programming methods will have to be explored for dealing with geometric and kinematic problems when moving or manipulating configurations spread in space [5]. Imagine cellular clusters behaving as welders or cutters of elements, sensors or positioning arms for object manipulation, material carriers or memorization loops. As computations on large SIMD arrays are largely based on geometry of planar data movements and kinematics of wavefronts, these new techniques of computation may ultimately be appropriated for large and massively parallel machines, the 29-state automaton providing an experimental ground for designing and developing them.

Another question may be raised: Why using such a complex automaton for designing tools dedicated to massively parallel computers? Most of the applications currently performed on SIMD arrays, small or large, rely on the automatic generation of parallel versions of ordinary programs. However, there is a growing recognition of the importance of exploiting massive parallelism to develop more complex and original applications for which new software tools are necessary. Von Neumann's 29-state cellular automaton, exhibiting an impressive amount of structured complexity, may be considered as such an application.

In this paper, attention has therefore been focussed on the transition rule that governs the overall behavior of the 29-state cellular automaton. The transition rule of a cellular automaton is the set of local rules specifying which state a cell will enter (during the next time step) as a function of its current state and the state of the cell's neighbor:

$$a_{i,j}^{t+1} = f(a_{i,j}^t,\ a_{i-1,j}^t, a_{i,j-1}^t,\ a_{i+1,j}^t,\ a_{i,j+1}^t)$$

With 29 states per cell and 5-cell neighborhood, there are $29^5$ (i.e. about 20 millions) possible local rules: they represent the sole and complete transition rule that governs the automaton. The updating of cells is synchronous and uniform: all the cells update their state at the same time and according to the same transition rule. Note that the range of the transition rule is 29 elements while its domain has $29^5$ elements. Hence, there are $29^{(29^5)}$ possible transition rules or modes of the class under consideration [26].

We discuss the implementation of this transition rule on the Gapp. The Gapp presents interesting features: a) a small local RAM (128 bits) per processor of which 13 bits are used for implementing the rule: 5 bits to encode the 29 states, the remaining bits encoding directions, groups of states and excitation marks, b) Gapp devices are fully cascadeable enabling to build up large arrays of processors in $6 \times 12$ increments (a Gapp device has 72 processing cells), c) connections between processors are orthogonal as in the automaton.

To completely specify the transition rule in tabular form would require about 20 million entries. As an alternative, we propose to compute the transition rule during each updating step: thus, 4970 micro-instructions are executed per time unit, in lock-step synchronization.

This paper is organized as follows: Section 2 defines and introduces the 29-state cellular automaton and describes in detail its transition rule. Section 3 discuss the implementation of the transition rule on the Gapp. We expose the method used for encoding states, groups of states and directions. Section 4 gives several examples of implemented organs. Finally, the conclusion is presented in section 5.

## 2 States and Transition Rule

The basic structure of the 29-state cellular automaton is a (possibly infinite in extent) rectangular grid, each cell of which is occupied by the same 29-state automaton. Computation and movement of data on the cellular grid are determined by the local changes of – unexcited or excited – states in cells. Note that a state in von Neumann's automaton, is both a value and a direction. Excited states induce a *directed* propagation of excitation also called a *signal*. Cells change state at discrete times according to a transition rule which determines the next state of a cell as a function of its current state and its four nondiagonal neighboring cells. Logical and arithmetic operations are performed by specific combination of states on the cellular space.

### 2.1 The 29 states

The 29 states can be conveniently grouped into 5 categories.

**First category:**
There are eight ordinary transmission states denoted by $T_{u,v}$ where $u \in [n, s, e, w]$ specifies the direction of the transmission (north, south, east, west), and $v \in [0, 1]$ designates the unexcited or excited ordinary group. Ordinary transmission states act either as direct delay lines or as disjunction elements (OR logical gate), with unit delay. A cell, in an ordinary transmission state, may receive excitation from three directions and transmits excitation to the neighbor in the $u$-direction. The symbols $\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$ represent unexcited ordinary transmission states. One point (.) over the arrow indicates excitation.

**Second category:**
There are eight special transmission states encoded by $T'_{u,v}$, for $u \in [n, s, e, w]$ and $v \in [0, 1]$. They are similar in operation to ordinary transmission states – acting as directed delay lines and as disjunction elements – except that they convert confluent states to U. The symbols $\Uparrow$, $\Downarrow$, $\Leftarrow$, $\Rightarrow$ represent unexcited special states.

**Third category:**
There are four confluent states $C_{u,v}$ where $u$ and $v$ are either 0 or 1: $u$ specifying the current state, $v$ the next output state. There is one unexcited confluent state: $C_{00}$ and three excited confluent states: $C_{01}$, $C_{10}$ and $C_{11}$.
The confluent state has four characteristics:

1. It transmits excitation after 2 units of delay. This feature may be used to delay the propagation of a signal: $n$ $C$ states correspond to $n \times 2$ units of delay.
2. It receives excitation from ordinary transmission states and transmits excitation to ordinary and special transmission states.
3. It acts as a fan out (or T junction) directing excitation to the three immediately neighboring cells not pointing to it.
4. It acts as a conjunction element (AND logical gate): all of the ordinary transmission states pointing to it have to be excited.

**Fourth category:**
This category has a single element: the quiescent state, called U for *u*nexcitable state (or blank state). Cells in the quiescent state have to be excited with more than one signal directed to them.

**Fifth category:**
The eight sensitized states, called $S_\Sigma$ where $\Sigma \in [\theta, 0, 1, 00, 01, 10, 11, 000]$, do not propagate signals. They are intermediary states converting a quiescent state into one of the 9 unexcited states: $T_{u,0}$, $T'_{u,0}$ and $C_{00}$.
Table 1 summarizes the above description. It shows the seven – unexcited and excited – groups of states.

Table 1: The seven unexcited and excited groups of states.

| Groups | The 29 states | Symbols |
|---|---|---|
| 1 | 4 unexcited ordinary transmission states $T_{u,0}$ | $\rightarrow \uparrow \leftarrow \downarrow$ |
| 2 | 4 excited ordinary transmission states $T_{u,1}$ | $\dot{\rightarrow} \dot{\uparrow} \dot{\leftarrow} \dot{\downarrow}$ |
| 3 | 4 unexcited special transmission states $T'_{u,0}$ | $\Rightarrow \Uparrow \Leftarrow \Downarrow$ |
| 4 | 4 excited special transmission states $T'_{u,1}$ | $\dot{\Rightarrow} \dot{\Uparrow} \dot{\Leftarrow} \dot{\Downarrow}$ |
| 5 | 4 confluent states | $C_{00}$ $C_{01}$ $C_{10}$ $C_{11}$ |
| 6 | 1 quiescent state | U |
| 7 | 8 sensitized states | $S_\theta$ $S_0$ $S_1$ $S_{00}$ $S_{01}$ $S_{10}$ $S_{11}$ $S_{000}$ |

## 2.2 The transition rule

The transition rule is the set of local rules that governs the overall behavior of the automaton. The rules themselves fall into three classes:

1. rules for signal transmission
2. rules for construction on the quiescent space
3. rules for killing states and converting them to U

**First class:** • An ordinary (special) transmission state is excited if there exists an excited ordinary (special) transmission state (Fig.$1_a$) or an excited confluent state (Fig.$1_b$) directed to it. Otherwise, an ordinary (special) transmission state remains unchanged.

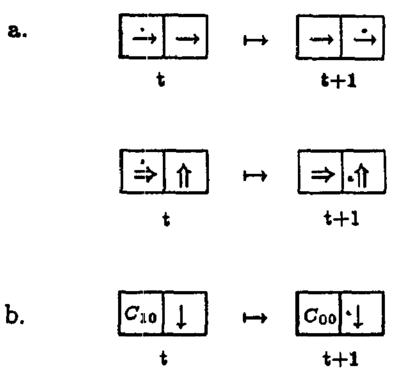



Figure 1: Signal Transmission Rule – Example 1.

• The signal transmission rules for confluent states are best illustrated by Lee's diagram [9] in Fig.2. One 1 over the arrow means excited, 0 unexcited.
A cell in confluent state, at time t, goes into one of the three excited confluent states $C_{01}$, $C_{10}$, $C_{11}$ if the two following conditions are satisfied: i) all the neighbors pointing to it are ordinary transmission states, ii) all the neighbors are in the excited state at time t (Fig.3).

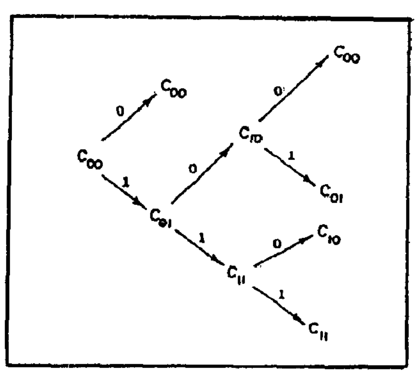
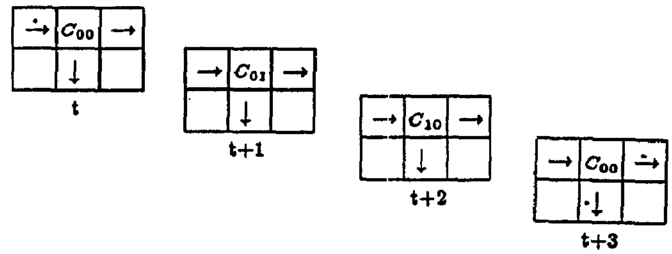
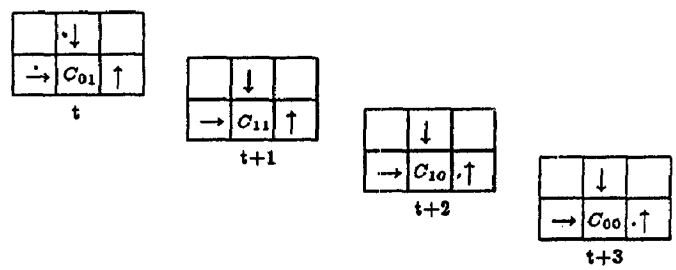



Figure 2: The Confluent Tree.

a. illustration of $C_{10}$ meaning excitation *at the current time step* and $C_{01}$ meaning excitation *at the next time step*




b. illustration of $C_{11}$ meaning excitation *for two time steps* in succession




c. illustration of unexcitation

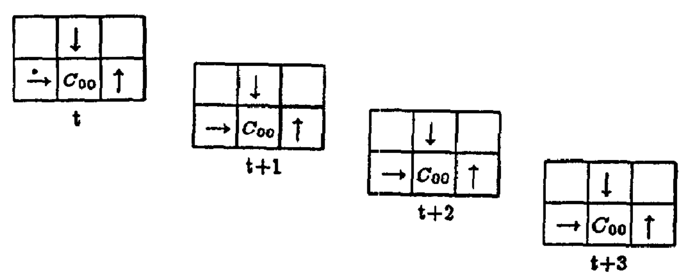



Figure 3: Signal Transmission Rule – Example 2.

**Second class:** The construction process (von Neumann called it the *direct process*) transforms the quiescent state U into any of the nine unexcited states: $\rightarrow$, $\uparrow$, $\downarrow$, $\leftarrow$, $\Rightarrow$, $\Uparrow$, $\Downarrow$, $\Leftarrow$ and $C_{00}$ using the eight *sensitive* states $S_\theta$, $S_0$, $S_1$, $S_{00}$, $S_{01}$, $S_{10}$, $S_{11}$, $S_{000}$ as intermediaries (Fig.4).

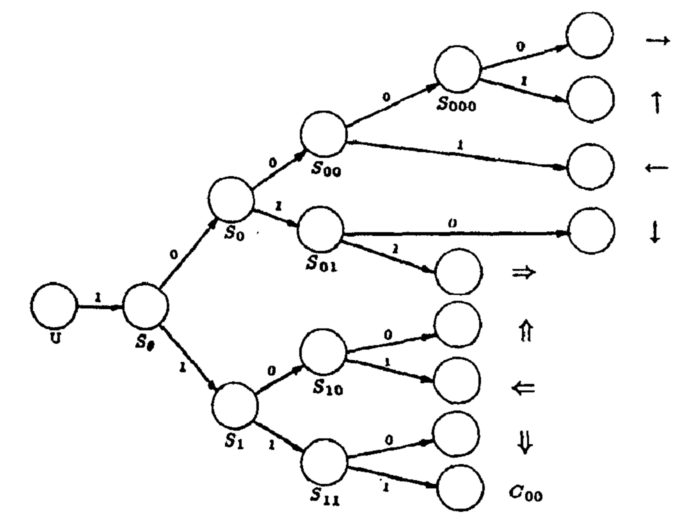



Figure 4: The Sensitized Tree.

It shows the possible sensitized states where 0 means *passive* (unexcited) and 1 *active* (excited).

The following diagrams illustrate the construction process (Fig.5).

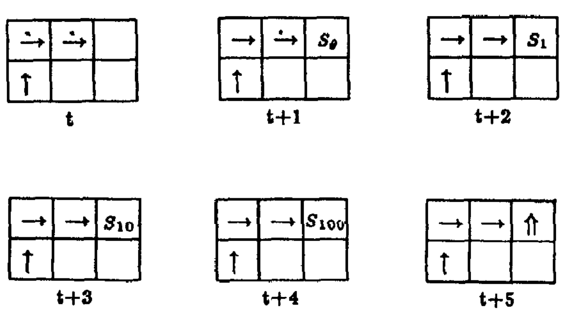
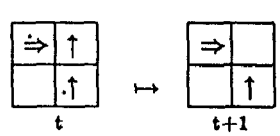



Figure 5: The Construction Process.

**Third class:** The destruction process (von Neumann called it the *reverse process*) transforms both unexcited and excited states into the quiescent state, in one single time step.
An ordinary transmission state or a confluent state is killed to U, if there exists an excited special transmission state directed towards it; killing dominates reception (Fig.$6_{a,b}$).
A special transmission state is killed to U, if there exists an excited ordinary transmission state directed towards it; killing dominates reception (Fig.$6_c$).



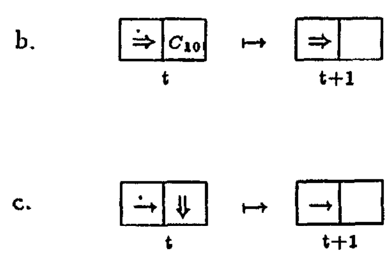




Figure 6: The Destruction Process.

# 3 Implementation

## 3.1 The Geometric Arithmetic Parallel Processor

The chip[1] Gapp has 72 bit-serial parallel processing cells. Each cell contains an ALU and 128 bits of RAM as well as bidirectional communication lines that connect the cell to its neighbors on the north, south, east and west. Fig.7 shows the block diagram of a cell. Interestingly, the processing cells themselves are not particularly fast, taking 2.5$\mu$s to add two 8 bit numbers. Executing 72 such operations simultaneously, though, yields an overall data rate of 28 millions 8 bit additions per second. Assembling an array of chips also eliminates memory bandwidth limitations. For instance, a 48-by-48-cell processor – composing 32 chips – can grab a 48-bit-wide word every 100ns when operating with 10-MHz clock. The array's bandwidth thus equals 480 Mbits/s [15].

[1]no. NCR45CG72 designed by NCR Corp.

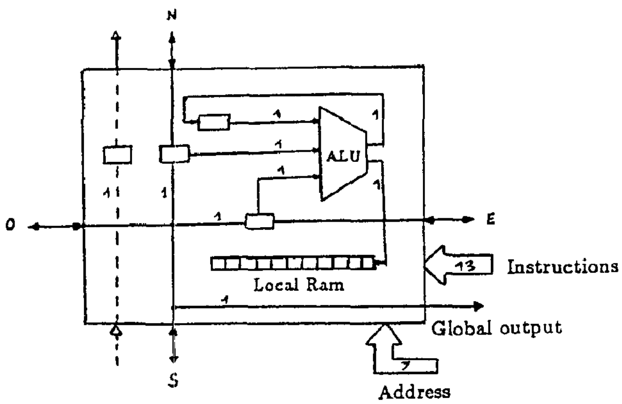



Figure 7: Block Diagram of a Processing Cell.

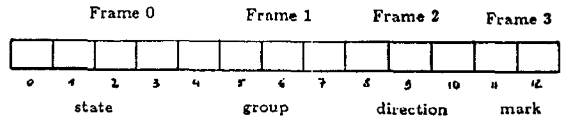



Figure 8: Memory Frames.

## 3.2 The Logical State of a Cell in the 29-state Automaton

The questions we were faced in implementing the transition rule were:

1/ what information is requested in a cell for updating it?

2/ how to structure this information for minimizing storage space?

As illustrated in Table 1, the state of a cell is defined by a symbol (*value* and *direction*) and the *group* to which it belongs (ordinary quiescent, ordinary excited, confluent ...). Moreover, information on neighbors' states has to be provided. Since only excitation is meaningful, a *marker bit* suffices to designate the surrounding as ordinary excited, special excited or confluent excited. We propose to call these marks OX, SX and CX (X means excited).

Thus, four sets of data are requested to completely specify the logical state of a cell using 13 bits of local RAM (see Table 2). Data are encoded by binary values in a corresponding memory frame (Fig.8) as shown in Table 3.

Table 2: Data sets to encode the logical state of a cell.

| Data | Nb. of Bits |
|---|---|
| 29 states | 5 |
| 7 groups | 3 |
| 4 directions | 3[1] |
| 3 marks | 2 |
| total | 13 |

(1) the binary representation starts from 1 to 4.

Table 3: Binary representation of the four data sets.

| Register | Code Values | Description |
|---|---|---|
| State | xxxxxxxx00000 | U(blank) |
| | xxxxxxxx00001 | ↑ |
| | xxxxxxxx00010 | ↓ |
| | xxxxxxxx00011 | ← |
| | xxxxxxxx00100 | → |
| | xxxxxxxx00101 | ↑ |
| | xxxxxxxx00110 | ↓ |
| | xxxxxxxx00111 | ← |
| | xxxxxxxx01000 | → |
| | xxxxxxxx01001 | ⇑ |
| | xxxxxxxx01010 | ⇓ |
| | xxxxxxxx01011 | ⇐ |
| | xxxxxxxx01100 | ⇒ |
| | xxxxxxxx01101 | ⇑ |
| | xxxxxxxx01110 | ⇓ |
| | xxxxxxxx01111 | ⇐ |
| | xxxxxxxx10000 | ⇒ |
| | xxxxxxxx10001 | $C_{00}$ |
| | xxxxxxxx10010 | $C_{01}$ |
| | xxxxxxxx10011 | $C_{10}$ |
| | xxxxxxxx10100 | $C_{11}$ |
| | xxxxxxxx10101 | $S_\theta$ |
| | xxxxxxxx10110 | $S_0$ |
| | xxxxxxxx10111 | $S_1$ |
| | xxxxxxxx11000 | $S_{00}$ |
| | xxxxxxxx11001 | $S_{01}$ |
| | xxxxxxxx11010 | $S_{10}$ |
| | xxxxxxxx11011 | $S_{11}$ |
| | xxxxxxxx11100 | $S_{000}$ |
| Group | xxxxx001xxxxx | U |
| | xxxxx010xxxxx | Ord Q |
| | xxxxx011xxxxx | Ord X |
| | xxxxx100xxxxx | Spe Q |
| | xxxxx101xxxxx | Spe X |
| | xxxxx110xxxxx | Conf |
| | xxxxx111xxxxx | Sens |
| Direction | xx001xxxxxxxx | Up |
| | xx010xxxxxxxx | Down |
| | xx011xxxxxxxx | Left |
| | xx100xxxxxxxx | Right |
| Mark | 01xxxxxxxxxxx | Ord X |
| | 10xxxxxxxxxxx | Spe X |
| | 11xxxxxxxxxxx | Conf X |

### 3.3 The Algorithm

The algorithm comprises two phases.

#### 3.3.1 First phase: Marking cells

Let us assume a finite area in the cellular space whose cells are initially in some state other than U. All the excited cells *mark* the neighbor cells to which they are pointing. Since the fan out of a confluent cell is equal to the number of neighbors not directed to it, up to three neighbors may be marked by an excited confluent cell. Depending on its group, a marked cell is excited or killed at the next time step. An unmarked cell becomes quiescent or remains quiescent. Fig.9 illustrates the marking phase.

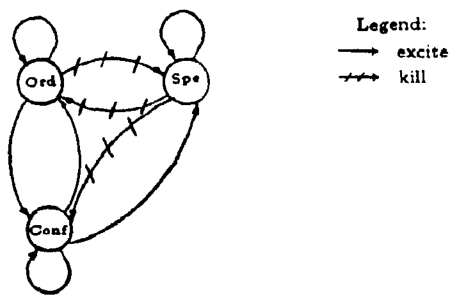



Figure 9: Marking Phase Graph.

Inside their own group, cells propagate excitation-mark; outside their group, cells propagate kill-mark, except between ordinary and confluent cells.

Table 4 shows how groups and marks can be combined to induce excitation or killing. For example, a special – excited or unexcited – transmission cell is killed with an OX-mark but excite with a CX-mark.

Table 4: Combinations of marks and groups.

| Kill | | Excite | |
|---|---|---|---|
| Group | Mark | Group | Mark |
| Special | OX | Ordinary | CX |
| Ordinary | SX | Ordinary | OX |
| Confluent | SX | Special | SX |
| | | Special | CX |
| | | Confluent | OX |
| | | State U | OX |
| | | State U | SX |
| | | Sensitized | OX |
| | | Sensitized | SX |

Consider the configuration in Fig.$10_a$:

a.

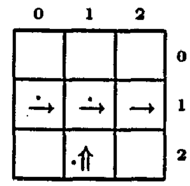



Figure $10_a$: Step 1.

Cell [1,1] is excited and receives two marks: OX and SX. Knowing that killing dominates reception, cell [1,1] will be killed at the next time step. However, killing does not dominate emission: thus, excitation in cell [1,1] will be propagated into cell [1,2] (Fig.$10_b$).

b.

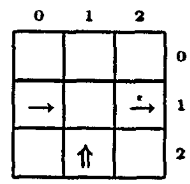



Figure $10_b$: Step 2.

#### 3.3.2 Second phase: Updating cells

For the execution of the transition rule, seven sets of SIMD micro-instructions are executed in succession, each one corresponding to a group listed in Table 1. Four cases are considered:

- Case 1:
  Killing dominates (Table 4).
  Erase: group, state, direction, mark
  New state: state U
  New group: state U
- Case 2:
  Excitation dominates (Table 4).
  Erase: group, state, mark
  New state: excited state (11 possible states)
  New group: excited group (3 possible groups)
- Case 3:
  No mark in excited state.
  Erase: group, state
  New state: unexcited state
  For ordinary and special:
  New group: quiescent group (8 possible groups)
  For confluent:
  New group: quiescent / excited group (Fig.2)
- Case 4:
  No mark in quiescent state.
  For sensitized:
  Erase: state
  New state: quiescent state (4 possible states) (Fig.4)

We conclude this section by the block diagram of a cell, based on the algorithm we have discussed (Fig.11). Four registers are sufficient to specify the internal structure of the cell. The mark register is connected to the four direct neighbors. It is written during a first clock cycle, then read during a second clock cycle, to be compared with the contents of the group register. Depending on this test the others registers may be updated.

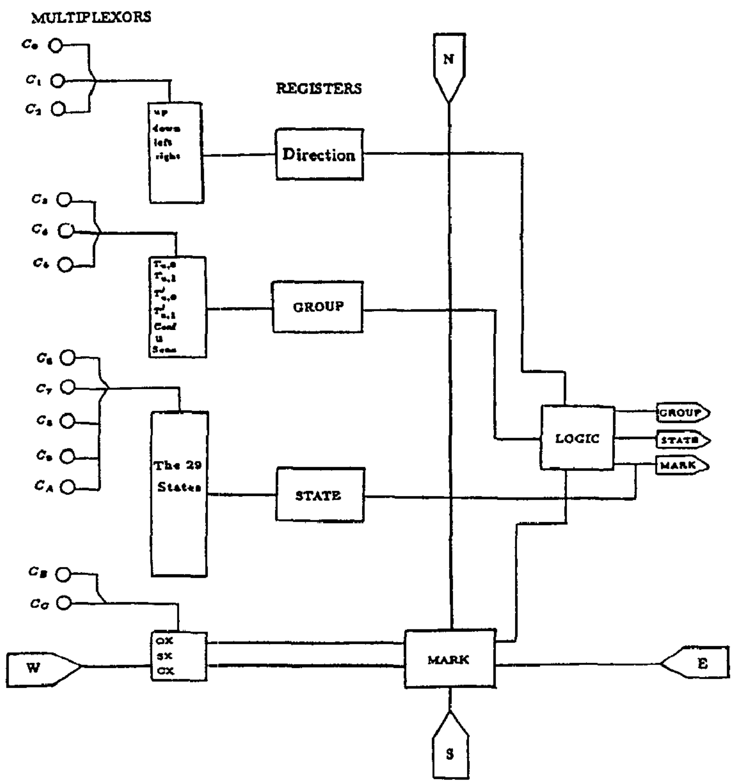



Figure 11: Block Diagram of a Cell on the Automaton.

# 4 Experimental Results

Von Neumann's 29-state automaton is a hierarchical organization of complex configurations composed of general-purpose components, called organs [26]. In this section, we will first consider the simulation of two organs, on the SIMD array: the pulser and the decoder and of one higher level structure: the periodic pulser. We will use von Neumann's notation $\overline{i_1 \cdots i_n}$ to specify the sequence of signals received or emitted by an organ where $i_1 \cdots i_n$ is its characteristic and $n$ its length. Next, we will consider the simulation of a serial adder, implemented on the SIMD machine, using the 29 states of the automaton to perform its operations.

## 4.1 Pulser

A pulser $\mathbf{P}(i_1 \cdots i_n)$ has an input cell, a pulser body and a output cell. It is used to encode a sequence of signals so that a single *excited* signal entering the input cell will produce the sequence $\overline{i_1 \cdots i_n}$ at the output cell. Confluent (as T junction) cells in the pulser body duplicate the input signal and direct it through different paths. As a consequence, n signals will be output corresponding to the number of duplications. At time $t+\Delta$ through $t+\Delta+n$ the sequence of signals is emitted.
The principles involved in the construction of a pulser are quite simple [26]. Determining $k$ and $u$, respectively the number of excited signals (1) and the number of unexcited signals (0) in the characteristic of the pulser, gives the height of the pulser: $u+2$, its width: $2 \times k$ and its delay (time requested for the input signal to be output): $2k+u+2$.
For example, pulser $\mathbf{P}(1010101)$ (Fig.12) has been constructed on the SIMD array. The excited signal in cell[4, 0] enters the pulser body, changing states in cells according to the implemented transition rule and is output from $t+13$ through $t+19$. Since 4970 microinstructions are executed per time unit, 94 430 microinstructions are necessary for emitting the complete sequence 1010101 at time $t+19$.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| | → | → | → | → | → | → | → | 0 |
| | ↑ | | $C_{00}$ | | ↑ | | $C_{00}$ | 1 |
| | ↑ | | ↑ | → | ↑ | → | ↑ | 2 |
| | ↑ | | ↑ | ↑ | ← | ↑ | ← | 3 |
| → | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | 4 |

Figure 12: Pulser $\mathbf{P}(1010101)$.

## 4.2 Decoder

The decoder, denoted $\mathbf{D}(i_1 \cdots i_n)$, is used to detect the existence of particular sequences of input signals. It generates an excited output signal for any input sequence $\overline{j_1 \cdots j_n}$ which is bitwise implied by $\overline{i_1 \cdots i_n}$. The decoding operation can be explained as follows: let $x_1 \cdots x_n$ be the ordered indices of excited signals (1) in the sequence $\overline{i_1 \cdots i_n}$ and let $\overline{j_1 \cdots j_n}$ be the input sequence. The bits $j_{x1} \cdots j_{xn}$ of the sequence are ANDed together bitwise: first, $j_{x1}$ is ANDed with $j_{x2}$ to produce $j_{x1} \circ j_{x2}$, then $j_{x3}$ is ANDed with $j_{x1} \circ j_{x2}$ to produce $j_{x1} \circ j_{x2} \circ j_{x3}$; finally, $j_{xn}$ is ANDed with $j_{x1} \circ j_{x2} \circ j_{x3} \cdots \circ j_{xn-1}$. The value of this final AND determines the output of the decoder: 1 (excited signal), 0 otherwise.

The design of a decoding organ is very similar to that of a pulser [26] except that confluent states are needed along the top row of the decoder to AND bits of the input sequence.
With $k$ representing the number of excited signals in the input sequence and $u$ standing for $n-1$, the height of the decoder is equal to $u+2+1$, its width is equal to $2k$ and its delay to $3k+u+n+1$. Decoder **D**(100101) has been simulated (Fig.13). A single excited signal is output at cell[0,5], at time $t+21$ requiring 104 370 micro-instructions to be executed.

## 4.3 Periodic Pulser

A periodic pulser $\mathbf{PP}(i_1 \cdots i_n)$ is a higher level structure constructed from tow pulsers and a repeater. **PP**(10010001)[2] has been simulated.
It is constructed from a pulser **P**(10010001) which produces the sequence 10010001, a repeater which repeats it periodically, a transducer by means of which the repeater is turned off and a pulser **P(1111111111)** which propagates the signals needed for turning the repeater off. Fig.14 represents the schematic diagram of **PP(10010001)**. Fig.15 shows the implemented periodic pulser.

A start signal entering the input cell $a_+$ of **P(10010001)** goes back through the input line, then enters the pulser body and flows out towards the repeater. Entering the repeater, the sequence of signals starts looping through the cellular space of the repeater. At equal intervals, a complete sequence of signals is output at the output cell of the repeater.
A start signal at the input cell $a_-$ of **P(1111111111)** goes through the input line, enters the pulser body, propagates through the channel and enters the repeater. The first excited signal of the sequence of signals inhibits the repeated process.

The succeding signals transform the output cell of the repeater into an unexcited cell. Repeater and pulsers are stopped.

| 0 | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | 0 |
| | ↑ | | ↑ | | ↑ | 1 |
| | $C_{00}$ | | ↑ | | ↑ | 2 |
| → | ↑ | | ↑ | | ↑ | 3 |
| ↑ | ← | | ↑ | | ↑ | 4 |
| → | ↑ | → | ↑ | | ↑ | 5 |
| ↑ | ← | ↑ | ← | | ↑ | 6 |
| → | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | 7 |

Figure 13: Decoder **D**(100101).

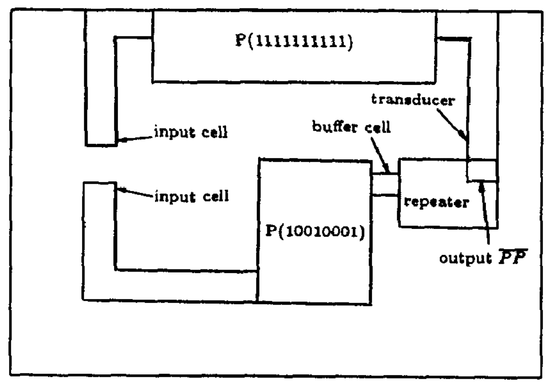



Figure 14: Schematic design of **PP**(10010001).

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | → | → | → | → | → | → | → | → | → | → | → | → | → | → | → | → | → | → | → | → | ↓ | 0 |
| | → | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | | $C_{00}$ | 1 |
| | ↑ | | | | | | | | | | | | | | | | | | | | | ⇓ | 2 |
| | ↑ | | | | | | | | | | | | | | → | → | → | → | → | → | → | $C_{00}$ | 3 |
| | ↑ | | | | | | | | | | | | | | ↑ | | ↑ | | $C_{00}$ | | ↑ | ↓ | 4 |
| $a_-$ | ↑ | | | | | | | | | | | | | | ↑ | | ↑ | → | ↑ | | $C_{00}$ | ← | 5 |
| | | | | | | | | | | | | | | | ↑ | | ↑ | ↑ | ← | | | | 6 |
| $a_+$ | ↓ | | | | | | | | | | | | | | ↑ | → | ↑ | → | ↑ | | | | 7 |
| | ↓ | | | | | | | | | | | | | | ↑ | ↑ | ← | ↑ | ← | | | | 8 |
| | → | → | → | → | → | → | → | → | → | → | → | → | → | → | $C_{00}$ | → | $C_{00}$ | → | $C_{00}$ | | | | 9 |

Figure 15: Periodic Pulser **PP**(10010001).

[2] section 3.2.2 in [26].

Table 5 summarizes the simulation results. It specifies the number of processors needed to implement each configuring organ and time delays to output a sequence of signals. For example, the pulser **P**(1111111111) in the periodic pulser will propagate a killing sequence into the repeater from time $t+31$ through time $t+40$. Note that a single bit is output from the decoder. Column 2 specifies the number of micro-instructions needed to generate a complete sequence of signals out of the corresponding organ. Composed of two pulsers, the periodic pulser **PP**(10010001) will execute 193 830 micro-instructions in a coding cycle (propagation of 10010001 out of the repeater), and 198 800 micro-instructions in a killing cycle (propagation of 1111111111 into the repeater).

Table 5: Simulation results.

| | Nb. Procs. | $\mu$Instr. | Time |
|---|---|---|---|
| **P**(1010101) | 31 | 94 430 | 13-19 |
| **D**(100101) | 35 | 104 370 | 21 |
| **PP**(10010001) | 102 | P(10010001): 193 830 | 32-39 |
| | | P(1111111111): 198 800 | 31-40 |

## 4.4 Implementation of a serial adder

Some other configurations can be constructed in the 29-state cellular space, using the transition rule to perform computations and movements of data. As an example, Fig.16 presents a serial adder, with a particular coding of states which can be seen as the automaton's assembly language (see Table 6).

In a two-dimensional cellular space, two continuous streams of signals cannot cross without interference. This situation raises the possibility that a signal traveling in one stream can be corrupted by a signal traveling in the other stream. To resolve this problem, a crossing organ was first proposed by von Neumann [26], then by Burks [2], Mukhopadhyay [11], Banks [1] and Nourai [12], allowing the crossing streams to maintain their bidirectional capacity. However, introducing a crossing organ in a configuration increases the spacing and timing between connected parts. In the serial adder, where eight crossing organs are requested, this involves more computational effort than desirable.

As an alternative, we defined four *macro-states* (see Table 7), requiring no additional lag time, since they transmit excitation after two units of time as the confluents states. Only the W (crossing right-up) and the X (crossing right-down) macro-states are used in the serial adder configuration.

In order for the serial adder to function properly, the excited signals in the two entering sequences $\overline{i_1 \cdots i_n}$ and $\overline{j_1 \cdots j_n}$ (represented here by the two arms of the T configuration in Fig.16) must be separated from one another by 21 units of delay. This restriction is necessary to prevent improper collision of signals. To delay

```
                         von Neumann CA Laboratory
+-----------------------------------------------------------------------------+
|             inputs                                                          |
|     McMcMcMcMcMcN     NaMaMaMaMaMaM                                         |
|     d                             d                    aaaaaaaaaaaaad       |
|     MaMaMaMaMaMaM     McMcMcMcMcMcO               aaaaMcccc       d   output|
|                 d     d                           b   b   b       d         |
|                 M     M                           M Mab   b       aaaaaaaad |
|                 D     D                      MaMaMb bcc   M       b       d |
|               aaXaad  d                      b      aab   b       b       d |
|               b d  d  d                      McMcMc Mcc   M       M       d |
|               b d  d  d                           b   b   b       b       d |
|               b MaaXaaXaMaMaMaMaMaMaMaMaMaMaMaMaMaaaWaaaWaaM      McMcMc  d |
|               b d  d  d                               b   b  d          b d |
|               b M  MaaXaaaaaaaaaaaaaaaaaaaaaaaaaaaaaMaaaWaaaaaaaMaMaMb  d |
|               b d  d  d                                   b  M  f       d |
|               b d  dccMaMaMaMaMaMaMaMaMaMaMaMaMaMaMaMaMaMaMb  M   dcccccc |
|             aaWaaaaM                                         aab   aaaaaad |
|             b b    d                                         b b   dcccccd |
|             bcMccccMaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaMaMaMaMaM    aaaaaaa |
|                                                                             |
+-----------------------------------------------------------------------------+
Normal: read (c)onfig, (e)dit_screen, (r)un, (R)un blind, (s)tep, (o)thers
Command:
```

Figure 16: The configuration of a serial adder.

Table 6: Symbols of states in the implemented automaton.

| Groups | The 29 states | Symbols |
|---|---|---|
| 1 | 4 unexcited ordinary transmission states $T_{u,0}$ | *a b c d* |
| 2 | 4 excited ordinary transmission states $T_{u,1}$ | A B C D |
| 3 | 4 unexcited special transmission states $T'_{u,0}$ | *e f g h* |
| 4 | 4 excited special transmission states $T'_{u,1}$ | E F G H |
| 5 | 4 confluent states $C_{00}$ $C_{01}$ $C_{10}$ $C_{11}$ | M N O P |
| 6 | 1 quiescent state | < *space* > |
| 7 | 8 sensitized states | 1 2 3 4 5 6 7 8 |

Table 7: Implemented macro-states.

| Description | Symbols |
|---|---|
| crossing right-up | W |
| crossing right-down | X |
| crossing left-up | Y |
| crossing left-down | Z |

a signal, M confluent states (two units of time) are used in succession, as many times as necessary.

In Fig.16, the two input sequences are 101 and 111, the excited signals being represented by D, N and O. Note that the excited signals in the output sequence are similarly separated from one another by 21 units of time. The output sequence (1100) is transmitted from time $t + 104$ through time $t + 167$, requiring 829 990 micro-instructions, executed by 418 processing cells.

## 5 Conclusion

We have discussed the implementation of the transition rule of von Neumann's 29-state cellular automaton on a fine-grained parallel machine. Our experiments indicate that the Gapp SIMD machine can currently simulate the whole automaton. The method outlined here should generalize to any highly parallel computer with orthogonal and bidirectional connection lines between processors. It provides a simple way to encode the logical state of a cell into a memory frame of a processing element. In our implementation, all of the processors are kept busy most of the time, the transition rule being computed during each time step.

Implementing this complicated large-scale cellular automaton goes through the development of new tools for constructing complex configurations from organs, as those presented in section 4, and for programming them. We think that these tools will ultimately be appropriated to the programming of near future large-scale SIMD arrays.

## References


[1] Banks, E.R., "Universality in Cellular Automata", *IEEE 11th Ann. Symp. Switching and Automata Theory*, Santa Monica, Calif., 1970, pp. 194-215.

[2] Burks, A.W., "Essays on Cellular Automata", *University of Illinois Press*, 1970.

[3] Codd, E.F., "Cellular Automata", *Academic Press*, 1968.

[4] Frisch, U., and al., "Lattice Gas Hydrodynamics in Two and Three Dimensions", *Complex Systems*, No 1, 1987, pp. 649-707.

[5] Greussay, P., "L'Ordinateur Cellulaire", *La Recherche*, No 204, Nov. 1988, pp. 1320-1330.

[6] Hasslacher, B., "Discrete Fluids", *Los Alamos Special Issue*, 1987.

[7] Kemeny, J.G., "Man Viewed as a Machine", *Scientific American*, No 192, Apr. 1955, pp. 58-67.

[8] Langton, C., "Self-reproduction on a Cellular Automaton", *Physica* 10D, 1984, pp. 135-144.

[9] Lee, C., "Synthesis of a Cellular Computer", *Applied Automata Theory*, (ed.) Tou, J.J., Academic Press, 1968, pp. 217-234.

[10] Margolus, N.H., "Quantum Computation", *Annals of the N.Y. Academy of Sciences*, bf. 480, 1980, pp. 487-497.

[11] Mukhopadhyay, A., "Representation of Events in the von Neumann Cellular Model", *Journal of the*

*Association of Computing Machinery*, vol. 15, No 4, Oct. 1968, pp. 693-705.

[12] Nourai, F., Kashef, R.S., "A Universal Four-State Cellular Computer", *IEEE Transactions on Computers*, vol. C-24, No 8, Aug. 1975, pp. 766-776.

[13] Preston, K., Duff, M.J.B., Levialdi, S., Norgren, P.E., Toriwaki, J-L., "Basics of Cellular Logic with Some Applications in Medical Image Processing", *Proc. IEEE 67*, 1979.

[14] Preston, K., Duff, M.J.B., "Modern Cellular Automata Theory and Aplications", *Plenum Press*, 1984.

[15] "Processor Gapp – Geometric Arithmetic Parallel Processor", *NCR Corporation*, Dayton, Ohio, 1984.

[16] Rosenfeld, A., "Parallel Image Processing using Cellular Arrays", *Computer* 16, 1983.

[17] Rosenfeld, A., "Picture Languages", *Academic Press*, 1979.

[18] Signorini, J., "Complex Computing with Cellular Automata", in *Cellular Automata and Modeling of Complex Physical Systems* (Les Houches 1989 Winter Meeting), P. Manneville and G. Vichniac (eds.), Springer-Verlag, 1989. (forthcoming)

[19] Sternberg, S., "Language and Architecture for Parallel Image Processing", *Proc. Conf. in Pattern Recognition and Practice*, (ed.) Gelesman, E.S., and Kanal. L.N., North-Holland, 1980.

[20] Smith, S.A., and al., "Cellular Automata in Cytoskeletal Lattices", *Physica* 10D, 1984, pp. 168-174.

[21] Swindale, N., "A model for the formation of ocular dominance stripes", *Proc. Roy. Soc.*, b208, 1980, pp. 243-253.

[22] Thatcher, J.W., "Universality in the von Neumann Cellular Model", *Essays on Cellular Automata*, (ed.) Burks, A.W., University of Illinois Press, pp. 132-186.

[23] Toffoli, T., Margolus, N.H., "Cellular Automata Machines - a new environment for Modeling", *MIT Press*, 1987.

[24] Toffoli, T., "Cellular Automata Machines as Physics Emulators", *Proc. Impact of Digital Microelectronics and Microprocessors on Particle Physics*, Italy, March 28-30, 1988.

[25] Vichniac, G.Y., "Simulating Physics with Cellular Automata", *Physica* 10D, 1984, pp. 96-115.

[26] von Neumann, J., "Theory of Self-Reproducing Automata", *University of Illinois Press*, (edited and completed by A.W. Burks), 1966.

[27] Wolfram, S., "Statistical Mechanics of Cellular Automata", *Reviews of Modern Physics*, vol. 55, No 3, July 1983, pp. 601-644.

[28] Wolfram, S., "Universality and Complexity in Cellular Automata", *Physica* 10D, 1984, pp. 1-35.