*Complex Systems* **1** (1987) 585–597

# RAP1, a Cellular Automaton Machine for Fluid Dynamics

**Andre Clouqueur**
**Dominique d'Humières**
*CNRS, Laboratoire de Physique de l'Ecole Normale Supérieure, 24 rue Lhomond, 75231 Paris Cedex 05, France*

**Abstract.** RAP1 is a special purpose computer built to study lattice gas models. It allows the simulation of any model using less than 16 bits per node, and interactions restricted to first and second nearest neighbors on a $256 \times 512$ square lattice. The time evolution of the automaton is displayed in real time on a color monitor at a speed of 50 frames per second.

## 1. Introduction

The concept of cellular automata was introduced in the early fifties by von Neumann and Ulam [1] to study the behavior and the organization of complex systems. A cellular automaton (CA) is a set of *identical* processors located on a *regular lattice* and with *limited connections* with their neighbors. For each time step, the CA is described by the values of the states of all the processors. At time $t+1$, all the processors compute in parallel their new state as a given function of their state and those of the connected processors at time $t$. Wolfram [2] has shown that very simple one-dimensional CA with one-bit internal states may give extremely complicated behaviors, as soon as each cell is connected to its first- and second-nearest neighbors and its time evolution is given by a Boolean function chosen within the suitable set of Boolean functions of five Boolean variables. The CAM machines built at MIT by T. Toffoli [3] played a prominent part in the interest for CA during the last five years. These machines demonstrated that cheap, but very powerful, special-purpose computers can be built to study a wide class of CA. They have also shown the impact of direct visualization on the study of very complex phenomena.

During the same time, several attempts were done in the physicist community to find simple ways to describe and study the motion of a collection of interacting particles [4,5]. In the simplest model [5], the particles are constrained to move on a square lattice from a node to one of its nearest neighbors and to interact on the nodes only. However, this model was too simple to give realistic flows. Two years ago, Frisch, Hasslacher, and

Pomeau [6] found that the same kind of model on a triangular lattice leads to a more accurate approximation of a real gas. Using standard methods of statistical mechanics, they were able to show that the time evolution of this gas is described by the Navier-Stokes equation like most of the real fluids. Since, lattice gas models have received a lot of interest. This class of models, which first lead to Navier-Stokes equation are now applied to a wide range of systems from thermal effects to combustion phenomena [7]. Thus, lattice gas automata not only support the conjecture that cellular automata are able to simulate partial differential equations, but bring standard methods of physics to build and study a wide class of cellular automata. In this class, the $i^{\text{th}}$ bit of the processor states is viewed as a particle which can jump at each time step from one node of the lattice to its neighbor in the direction $\mathbf{c}_i$. Thus, the time evolution is split in two substeps. During the first one, called the "collision step", each processor computes its new state as a function of its state at time $t$; during the second, the "propagation step", the bit $i$ is moved from each processor to its neighbor in the direction $\mathbf{c}_i$.

While the CAM machine is very well suited for the study of two-dimensional cellular automata deriving from the original study of Von Neumann and Ulam, in which the neighborhood relations are essential, its use to simulate the relative motion of bits of information needed for lattice gas models requires quite a subtle trick known as the Margolus neighborhood [3,8,9]: the nodes are no longer equivalent but are packed in two-by-two cells which become the new node of the automaton, thus decreasing by a factor of four the effective size of the lattice. The specific algorithm of the two-dimensional lattice gas models leads to a more practical architecture: the original two-dimensional lattice, made of $M \times N$ cells with $b$ bits per cells, is also viewed as a three-dimensional lattice, made of $b$ one-bit $M \times N$ planes. During the collision step, the two-dimensional structure is used, each cell computing its new state; during the propagation step, the $i^{\text{th}}$ plane is moved as a whole in the direction $\mathbf{c}_i$ with respect to a reference plane.

In section 2, the similarities and the differences of RAP1 with the raster displays and the CAM machines will be presented. Section 3 will be devoted to the description of the hardware implementation. Preliminary results of hydrodynamical simulations will be given in section 4.

## 2. Video architectures

The RAP1 project started at the beginning of October 1985, with the following constraints:

1. Versatility, to allow the machine to be used to study a large class of lattice gas models of physical interest.

2. Direct display of the automaton evolution, to remove a classical bottleneck of the simulations on general purpose computers, i.e., the

visualization of the results.

3. Fast completion in order to build the prototype before the subject drifts too far from its starting point. This point implied the use of the limited resources of the laboratory for this not scheduled project.

4. Possibility to extend the design to larger lattice sizes.

These constraints implied several technical choices. First, we ruled out the use of true parallel architecture and chose to take benefit of the CAM experience, that is, to use a serial implementation of the algorithm. A consequence of this choice and of the synchronization of the computation process with the visualization restricted the size of the lattice to 256 lines of 512 pixels, a size which can easily be displayed on a low-resolution color monitor. In the following subsection, the evolution from raster displays to the RAP1 architecture will be presented at the conceptual level to stress the basic similarities along with the main differences between raster displays, CAM, and RAP machines.

### 2.1 Raster dispays

The raster displays are the basis of both the CAM and RAP machines. In these displays, the image is stored in a memory which we will refer to as screen memory. This memory is serially read row by row, synchronously with the sweep of the horizontal lines of the screen; the content of each memory location gives the intensity of the corresponding dot location on the screen as shown figure 1a. The time is divided into frame corresponding to the display of a full screen. The beginning of each frame is marked by a VSYNC signal. Each frame is in turn divided into lines corresponding to the display of one horizontal line. The beginning of each line is marked by an HSYNC signal. A VBLANK signal selects the lines during which the screen memory rows are actually displayed, and a HBLANK signal sets the active part of the lines. The VBLANK and HBLANK signals reset row and dot counters respectively; these counters are then incremented by the HSYNC and dot clocks respectively to give the address of the pixel to be read in the screen memory. The value of the pixel is then sent to a color look-up table which feeds digital-to-analog converters (DACs) to control the intensity of the red, green, and blue inputs of the monitor. Today, the details of the hardware implementation of this basic scheme are handled by graphic display processors (GDPs), which provide most of the needed signals.

For a typical low-resolution European monitor, such as the one used for RAP1, the frame frequency is 50 Hz or 20 ms per frame, and each line is 64-$\mu$s long. For a $256 \times 512$ resolution, the dot clock frequency is 14 MHz or 71.4 ns per pixel, the HBLANK signal is 36.57-$\mu$s long and occurs 17.71 $\mu$s after the HSYNC signal, and the VBLANK signal is active from line 38 to 293.

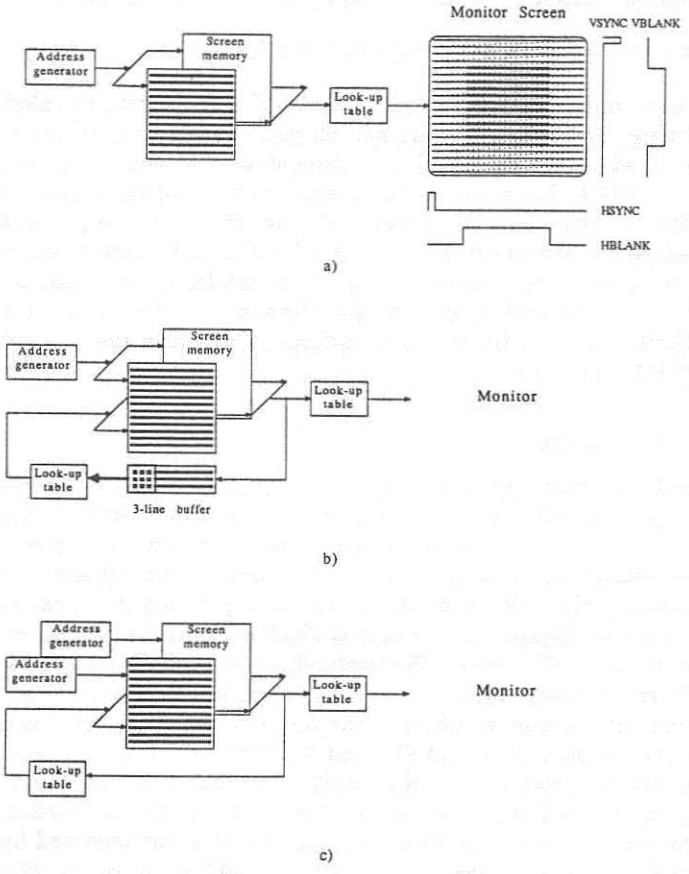


Figure 1: (a) Schematic of a raster display. (b) Schematic of the CAM machine. (c) Schematic of the RAP1.

### 2.2 The CAM machines

The first version of the CAM machine was completed in the early eighties at MIT and was one of the very first special-purpose computers designed to study physical problems. The description of one of its descendants, CAM-5, is given in reference 8, and the CAM-6 version is now commercially available along with a user manual and an extensive amount of software [9]. The basic idea of all these machines was to simulate two-dimensional cellular automata on a square lattice with the so-called von Neumann neighborhood: connections of each node to its first and second neighbors on a square lattice. The CAM machines were built around a raster display architecture. The simplest possible version is schematized in figure 1b. The screen memory is sequentially read row by row and sent to the display and to a three-line buffer which stores the data needed for a node and its eight neighbors. This buffer feeds a "computation" look-up table with nine inputs. The output of this table is then written back to the memory. In principle, this scheme can be used for multi-bit states; however, the size of the look-up table grows exponentially with the number of bits. Even for two-bit states, it is necessary to restrict either the number of bits interacting together or the number of available rules for the automaton, using some combinations of smaller tables. For example, the CAM-6 machine has four bits per state organized as two pairs of two bits, one of them interacting with its associated von Neumann neighborhood and the other alone. Up to ten bits per state can be obtained using the Margolus neighborhood [8], which has been used to study correlation functions of different lattice gas models [10].

### 2.3 The RAP1 machine

The RAP1 machine uses the same kind of basic architecture as the CAM machines use, but the neighborhood interactions are replaced by plane displacements (as in the lattice gas simulations [11] where time evolution of the automaton is split in two steps per frame). During the first step, the collision step, the value of the displayed pixel is sent to a computation look-up table, its input being written back to the screen memory as in the CAM machines; however, in the RAP, only the value of the pixel itself is used instead of the value of the pixel and of its neighbors. During the second step, the propagation step, the automon is viewed as made of a collection of one-bit two-dimensional planes, one for each bit of the internal state of the nodes. The displacement of bits from one node to one of its neighbors is done once by the corresponding translation of the whole plane, using an address generator per plane rather than only one for all the screen memory as is used for raster displays and CAM machines.

As for the CAM machines, the neighborhood relations can be implemented on the RAP machine. In this case, the information must be duplicated at each step as many times as there are relevant neighbors, then all these replica are moved in the corresponding direction. This process wastes

part of the screen memory, a penalty which can be somewhat decreased when several nodes are packed in the same 16-bit pixel. For example, four cells of the Conway's game of life [12] can be packed in one pixel, saving a factor of two for the overhead of the duplication.

## 3. RAP1 scheme

### 3.1 Collision step

In this section, we assume that the machine has only to compute its state at time $t+1$ as a function of its state at time $t$ with no need for information coming from the neighborhood. This computation corresponds exactly to the collision step of the lattice gas automata. In this mode, the information stored in the screen memory are serially read, sent to the computation look-up table, and written back to the screen memory during the display window.

For that, we use video random access memory (VRAM), which are the combination of a standard 65536 bits random access memory (64k RAM)[1] and a shift register 256 bits long. This shift register can be loaded by the content of an arbitrary row or its content can be written back to an arbitrary row of the memory; both operations use only one memory access. Except for these two operations, the use of the RAM array and the shift register are completely decoupled. The content of the shift register can be serially clocked out, while an external signal is clocked in at the same time. The shift registers of the VRAM are designed to be easily cascaded, and the screen memory of the RAP1 machine is made of 16 planes $256 \times 512$, each plane being made of two VRAM TMS 4161. The RAMs corresponding to the left and right parts of the screen will be said to be in zone 0 and 1 respectively.

This VRAM allows a very simple architecture. The row counter sets the address of the row to be displayed, which is down loaded into the shift registers 3.43 $\mu$s before the HBLANK signal becomes active. The shift registers are then clocked out 512 times at a 14-MHz rate and the 16-bit output word is used as an address for 16 static random access memories (64k SRAM) to produce a new 16-bit word which is written back to the shift registers. During the same time, the output of the shift register is also sent to an other look-up table made of three $4k \times 4$ SRAMs to generate information for the color display. After the visualization window, the shift register contains in place the new line after processing and is written back to the corresponding row of the screen memory.

### 3.2 Horizontal shifts

The horizontal shifts needed to simulate the horizontal motion of planes are obtained through a slight modification of the previous scheme. If the

[1] The RAM is organized as 256 rows of 256 bits each.

shift register is clocked 511 times and written back to the memory, the net result of this operation is a global shift of the line toward the right of the screen. To define periodic boundary conditions in the horizontal direction, the new state of the point located on the left edge of the row must be computed and the result written directly into the row of the left memory (zone 0) at location 00. For that, the output of the look-up table must be stored in an additional register inserted in the computation loop after the look-up table. In a similar way, if the shift register is clocked 513 times and written back to the memory, the result is a shift of the line toward the left of the screen. After 513 clock pulses, the leftmost point is shifted out of the shift register. To avoid a new computation of its value, another register must be inserted in the computation loop just before the look-up table. For periodic boundary conditions, the content of this register is directly written into the row of the right memory (zone 1) at location $FF$.[2] Thus, two registers must be inserted in the computation loop introducing two additional time delays in the loop. These delays must be compensated for by two additional clock pulses requiring 513 pulses for right shifts, 514 pulses for in place computations, and 515 ones for left shifts.

In fact, the TMS 4161 memories require an even number of clock pulses to correctly write back the content of the shift register, and the previous scheme must be slightly corrected by insertion in the computation loop of a third register and a multiplexer allowing this new register to be bypassed. When this third register is bypassed and the shift registers are clocked 514 times, there is no horizontal shift. If the third register is inserted in the loop, 514 or 516 clock pulses give right or left shifts respectively. The final loop is represented in figure 2 and the processing of any line can be summarized as follows:

1. Download into the shift registers the row memories at address given by the row counter.
2. Repeat steps 3 to 5 514 times.
3. Clock the shift registers and the three additional registers.
4. Compute the new state through the look-up table.
5. Feedback the shift register.
6. Write back the shift registers to the row memories for the planes with right shift.
7. Fill directly location 00 of the row memories of zone 0 for the planes with right shift.
8. Write back the shift registers to the row memories for the planes without horizontal shift.

[2]Hexadecimal notation.

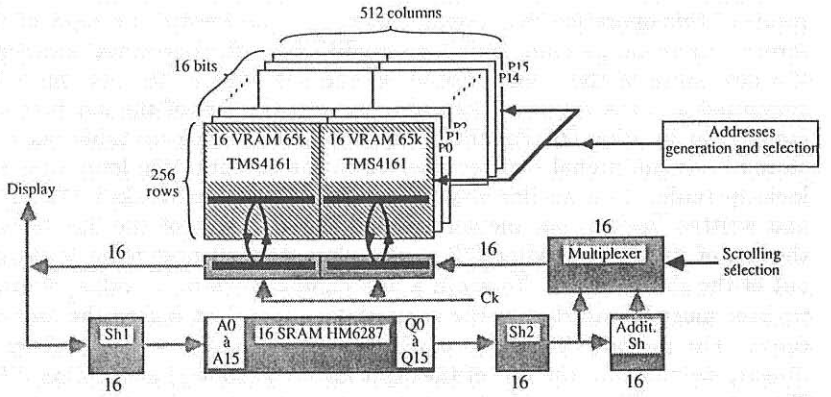


Figure 2: Detail of the computation loop of RAP1.

9. Clock twice the shift registers and the three additional registers, doing steps 3 to 5.

10. Write back the shift registers to the row memories for the planes with left shift.

11. Fill directly location $FF$ of the row memories of zone 1 for the planes with left shift.

Steps 3 to 5 are done at the same time, using the pipelined architecture of the loop, at a 14-MHz rate, and steps 1 and 6 to 11 are done at a much slower rate during the blanking windows.

Since the underlying structure of memories is a square lattice, a trick is needed to map hexagonal lattices on them. The simplest way consists to use the vertical and horizontal directions along with one of the 45° directions, leading to very severe geometric distortions: a rectangular area in the ordinary plane becoming a diamond. A better solution is given in reference 11 in which the directions of the oblique links depend on the parity of the lines. This technique is implemented in RAP1 with two different horizontal shift masks: one for the even lines and one for the odd ones, selected by the least significant bit of the row counter.[3]

### 3.3 Vertical scrolling

While the horizontal shifts correspond to physical displacements inside the screen memory, the vertical scrollings correspond to virtual displacements.

[3]For future use, a similar posibility is provided for the vertical scrolling, but depending now on the frame parity.

For that, a frame counter is needed and the address generator is split in three pieces:

1. one for the planes without scrolling with the same address for the displayed line and the downloaded memory row,
2. one for the plane with an upward scrolling, the address of the memory row being now the address of the displayed line plus the content of the frame counter,[4] and
3. one for the plane with a downward scrolling, the address of the memory row being now the address of the displayed line minus the content of the frame counter.

All these operations are done modulo 256 and naturally implement periodic vertical boundary conditions.

In principle, every plane should have its own generator and a selection mechanism to select the model dependent scrolling direction. In fact, to reduce the number of components, the address selection is done by a multiplexing technique and only one address generator. For that, steps 1 and 6 through 11 of the previous section are divided into three substeps, one for each of the three kinds of addresses. A four-bit pattern is associated with these substeps. The direction of the plane motion is then fixed by a four-bit mask and a selector on each plane prevents the row address select (RAS) and the column address select (CAS) signals to be sent to the screen memories except when a match occurs between the substep pattern and the plane mask.

### 3.4 Hardware realization

The RAP1 machine is made of ten printed boards, with printed circuits for the regular connections like the address lines of the memory and wire wrapping for random connections:

1. Two $5V - 6A$ power supply boards with serial regulators.
2. One interface board connects the RAP1 to an IBM PC-compatible microcomputer through two 16-bit parallel lines with handshake. One set of lines is used to send to the RAP1 a 16-bit command word giving the next operating mode:
   (a) Write the plane direction patterns in one of the four-plane boards;
   (b) set the memory address of the following input-output operation (this step is needed once for consecutive memory addresses, any input-output operation incrementing an I/O address counter);
   (c) write a word in the computation look-up table;
   (d) write a word in the color look-up table;

[4] The address of the top and bottom lines on the screen are respectively 0 and $FF$.

(e) write a word in the screen memory;

(f) read a word from the screen memory;

(g) read the line counter;

(h) read the frame counter (two words);

(i) start the computation process;

(j) stop the computation process.

3. The second set of lines is used to exchange 16-bit words between the PC and the RAP1 during the input-output operations.

4. One board provides all the timing and the address signals. The heart of this board is a graphic display processor (GDP) EF9365 which provides all the synchronization signals along with the line address counter for the display.

5. One board contains the 16 HM6287 SRAMs for the computation look-up table and two of the 16-bit registers in the computation loop.

6. Four boards with four planes per board are used for the screen memory. Each board contains eight TMS 4161 VRAMs, the displacement masks for the four planes, the RAS and CAS selectors, and the third 16-bit register and the multiplexer in the computation loop.

7. Half of the last board is used for the display interface, while the second half is available for future use, like post-processing of the data. This color look-up table is made of three 4k $\times$ 4 HM6168 SRAMs and the 16 bits to 12 bits selection is done the following way:

   (a) the 12 least significant bits for the blue look-up table,

   (b) the 12 middle bits for the green look-up table,

   (c) the 12 most significant bits for the red look-up table.

8. This configuration restricts the color coding somewhat, but allows substantial space saving. In most of the studied cases, a suitable choice of the plane meaning allows correct color codings.

## 4. Lattice gas simulations with RAP1

The debugging of RAP1 was finished at the end of the first quarter of 1986, six months after we started the project. The next six months were used to write enough software on the host PC to allow friendly use by non-expert users. Most of the efforts were concentrated on the hexagonal lattice gas models, especially on model III of reference 11, which allow reasonable Reynolds numbers to be obtained for moderate lattice sizes. This model was implemented using two seven-bit collision tables as in reference 11 computed once. Then, a spreadsheet allows quick definitions of different look-up tables using the remaining nine bits to define obstacles, sink or

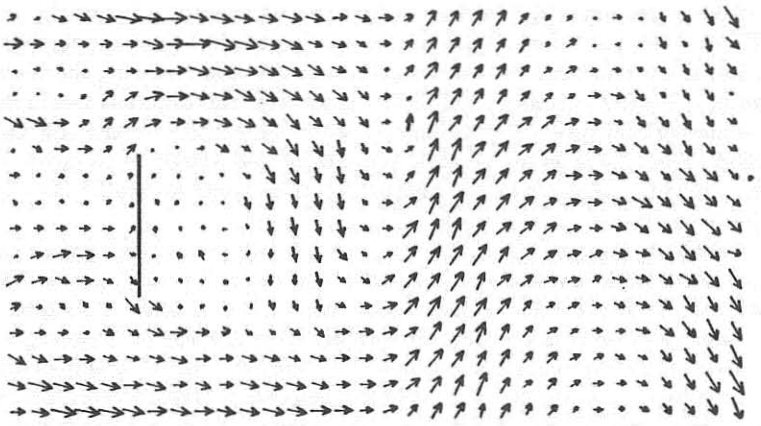

Figure 3: Von Karman street behind a flat plate obtained with RAP1. Local averages over $16 \times 16$ nodes are performed in the host IBM-PC computer from data retrieved from RAP1.

sources of particles, and so on. At present, we are using a configuration with seven bits for the particles: one to choose between the two possible collision tables, one for obstacles corresponding to "bounce-back" conditions, one for "absolute" sinks which destroy all the incoming particles, and the remaining for directed sources, one for each of the six directions of the triangular lattice. The existing software allows a very easy definition of flows around arbitrary obstacles with different "wind tunnel" conditions. The final geometry can be saved in some kind of library.

Despite the limited size of the RAP1 memory, several interesting hydrodynamics instabilities were observed: Kelvin-Helmholtz and von Karman instabilities. Figure 3 gives an example of the results obtained on RAP1 for a Reynolds number of about 100 and a Mach number of 0.5 around a flat plate, showing the shedding of vortices in the wake. This figure was obtained forty times faster than the corresponding flow of figure 1 of reference 13 obtained on a FPS164 for model II, clearly demonstrating the potentiality of this class of special-purpose computers for lattice gas simulations.

## 5. Conclusion

When we started this project, we knew that the size of the RAP1 memory would be too small for realistic hydrodynamics simulations, but we thought that the sixteen bits would be sufficient for future applications. At this time, the project for the next stage is to build a second version with several RAP1 modules with slight modifications to allow the exchange of data between them. The horizontal communication is easily done through steps

7 and 10 of section 3.2, if, instead of writing the extra point in the same RAP1 module, the informations are forwarded to the suitable neighboring one. For the vertical communication, a new step is added during the vertical blanking. During this step, the shift registers of the different modules in the same vertical directions are loaded with the content of the row to be scrolled. They are then linked together and clocked 512 times to move their content into the target module, and finally, their contents are written back to the memories.

The rapid progress in lattice gas mixtures [13] made this project completely obsolete, and we are now faced with the difficult problems of building processing structures able to manipulate more than 16 bits. This problem is at present a serious one for the extension of the use of lattice gas to mixtures or three-dimensional spaces [7]. Since the use of 24-bit or more look-up tables is prohibited by technical arguments (cost versus size and speed), new processing elements must be designed allowing enough versatility for a moderate complexity. For two-dimensional lattice gas models, this goal can be partially reached, at least for models with less than 16 bits, if the full word describing a node is split into two parts: one for the particles themselves (16 for example) with a collision step computed with look-up tables and a second one to drive extra memories used as multiplexers? to implement obstacles, sinks, sources, body forces, random choices between the look-up tables of the different modules, and so on. A $1024 \times 1024 \times 24$ machine working at 50 Msites per second with these kind of computation tables is planned for the next stage and a second one four times as big and as fast soon after.

## Acknowledgments

We thank P. Giron, who built the RAP1 machine; J. C. Bernard, S. Bortzmeyer, and V. Beliard, who wrote the software used for the simulations; and P. Lallemand and Y. Pomeau for helpful discussions.

## References

[1] J. von Neumann, *Theory of Self-Reproducing Automata*, (Univ. of Illinois press, 1966).

[2] S. Wolfram, *Rev. Mod. Phys.*, **55** (1984) 96.

[3] T. Toffoli, *Physica*, **10D** (1984) 195.

[4] I. E. Broadwell, *Phys. of Fluids*, **7** (1964) 1243; R. Gatignol, "Théorie Cinétique des Gaz à Répartition Discrète de Vitesse", *Lecture Notes in Physics* (Springer-Verlag, Berlin, 1975).

[5] J. Hardy, and Y. Pomeau, *J. Math. Phys.*, **13** (1972) 1042; J. Hardy, O. de Pazzis and Y. Pomeau, *J. Math. Phys.*, **14** (1973) 1746; J. Hardy, O. de Pazzis and Y. Pomeau, *Phys. Rev.*, **A13** (1976) 1949.

[6] U. Frisch, B. Hasslacher and Y. Pomeau, *Phys. Rev. Lett.*, **56** (1986) 1505.

[7] P. Clavin, P. Lallemand, Y. Pomeau and G. Searby, "Simulation of Front Dynamics in Flows: a New Proposal Based on Lattice-Gas Models", *J. Fluid Mech.* (1986), in press. P. Clavin, D. d'Humières, P. Lallemand and Y. Pomeau, *C. R. Acad. Sci. Paris XII*, **303** (1986) 1169.

[8] N. Margolus, *Physica*, **10D** (1984) 81.

[9] T. Toffoli and N. Margolus, *Cellular Automata Machines: a New Environment for Modeling*, (M.I.T. press, 1986).

[10] N. Margolus, T. Toffoli, and G. Vichniac, *Phys. Rev. Lett.*, **56** (1986) 1694.

[11] D. d'Humières and P. Lallemand, "Numerical Simulations of Hydrodynamics with Lattice Gas Automata in Two Dimensions", *Complex Systems*, **1** (1987) 598.

[12] E. R. Berlekamp, J. H. Conway and R. K. Guy, *Winning Ways for Your Mathematical Plays*, **2** (Academic Press, 1984).

[13] D. d'Humières, Y. Pomeau and P. Lallemand, *C. R. Acad. Sci. Paris XII*, **301** (1985) 1391.