Computer simulations of cellular automata

J. Phys. A: Math. Gen. 24 (1991) 909–927. Printed in the UK

REVIEW

Computer simulations of cellular automata

Dietrich Stauffer

Institute for Theoretical Physics, Cologne University, D-5000 Köln 41, Germany

Abstract. We review methods for large-scale simulations of cellular automata, in particular with only one computer bit used per site. We summarize recent results for basic classification as well as selected applications like ferromagnetism, flow through porous media and biologically motivated automata.

This review was received in December 1990

1. Introduction

In cellular automata, each site of a large lattice carries one or several spins, with each spin pointing either up or down. The orientation of the spin at time $t+1$ is determined completely by the orientation of its neighbour spins at time t. We ignore in this review all probabilistic cellular automata where the neighbour spins determine only the probability of the centre spin to point up (as in the Ising model). We also take into account only nearest-neighbour interactions; neural networks with an infinite range of interaction were recently reviewed by Kohring with emphasis on large-scale simulations [1]. Most of the time we work with only one such spin per lattice site. Wolfram's book [2] collects many older articles as well as new results in its appendices; Weisbuch's selection of fields [3] is similar to ours. We assume here the more elementary aspects of cellular automata computing to be known; they were reviewed recently elsewhere [4]. We deal with generally programmable computers, not with special purpose computers for cellular automata only [5].

In section 2 we therefore describe methods to increase speed and maximum lattice size by storing information in the single bits of a computer, and by treating these bits in parallel. Section 3 summarizes some results from both basic classification and magnetic as well as hydrodynamic applications, and section 4 reviews some speculations about biological applications.

2. Multispin coding

2.1. One word per site

Simple simulation methods use one computer word per spin; memory then can be saved on byte-oriented computers by using less bytes (two or one) for one spin than are used for a normal integer or real number. In a simple cubic $L*L*L$ lattice it is practical [2] to number the spins from 1 to L^3 instead of labelling them with three indices varying from 1 to L each. The six nearest-neighbours of site i are then the sites

$i-1, i+1, i-L, i+L, i-L^2, i+L^2$; on a triangular lattice, the fifth and sixth index are replaced by $i-L+1$, $i+L-1$. To avoid boundary conditions requiring IF conditions in the innermost loop, we also store the first plane of the lattice again in a buffer plane of sites L^3+1 to L^3+L^2; analogously the last plane of the true lattice is stored in a buffer plane with sites $-L^2+1$ to 0. This method gives automatically helical boundary conditions which are sometimes better, sometimes worse, than periodic boundary conditions. In principle one should try to simulate lattices so large that the boundary effects no longer disturb the properties in the interior; in practice, of course, this aim cannot always be reached.

This algorithm can easily be vectorized since the spins at time t depend, in the traditional definition of cellular automata, only on the spin orientations at the previous time step $t-1$ (parallel or simultaneous updating). If we denote the old spin array and the newly determined spin array by different names, no vector dependency occurs at all, and automatic vectorization should be possible. After one sweep through the whole lattice, another simple loop is needed to replace the old spin array by the new spin array. This latter simple loop can be avoided by suitable index manipulation if we use only one name for the old and the new spin array; but then vectorization may no longer be automatic, and programming errors are more likely.

2.2. One bit per site

Much more efficient in both memory and CPU time requirements, but also more difficult to program, are multispin coding techniques [6] where many different spins are stored in one computer word and are treated in parallel during the simulation. For example, a Cray computer stores 64 bits in one word. If we simulate in one dimension an infection process where each site becomes infected at time $t+1$ if at least one of its two neighbours (LEFT and RIGHT) is infected at time t, we have a simple logical OR relation for the centre spin: `CENTRE = LEFT .OR. RIGHT` in FORTRAN. Here we take the spins as logical or Boolean variables, with TRUE corresponding to infection. Storing 64 different spins in one computer word like CENTRE, LEFT and RIGHT, the above FORTRAN statement deals at once with 64 such logical operations, provided for every spin in the word CENTRE the corresponding left and right lattice neighbours are stored in the corresponding bit positions of the words LEFT and RIGHT. (Usually these words like CENTRE have to be declared as integers.)

Let us take for this example a linear chain of 192 spins, stored in three words of 64 bits each: LEFT, CENTRE, RIGHT. We take LEFT to contain spins 1,4,7,...,192, CENTRE to contain spins 2, 5, 8,...,191, and RIGHT to store spins 3,6,9,...,192. The SHIFT command is supposed to shift a computer word circularly to the left by a given number of bits; thus SHIFT(CENTRE,4) shifts the centre word by 4 bit positions to the left and appends the first (most significant) four bits of the original word to the end of the shifted word. A shift by 63 bits thus correspond to a right shift by one bit. With this operation the infection of the 192 sites proceeds by an innermost loop of three lines

```
NLEFT = CENTRE.OR.SHIFT(RIGHT,63)
NCENTRE = LEFT.OR.RIGHT
NRIGHT = CENTRE.OR.SHIFT(LEFT,1)
```

where NLEFT, NCENTRE and NRIGHT *contain the new spin orientations*. If we have a longer chain of $L = LL*64$ spins, we need $LL = L/64$ words. The updating of the first and the last of these words needs shifts like for LEFT and RIGHT in the above example, whereas the remaining $LL - 2$ words are treated normally, like the **CENTRE** word above. With such methods we automatically get periodic boundary conditions, e.g. the right neighbour of spin 192 is spin 1, and the left neighbour of spin 1 is spin 192 in the above example. In more than one dimension, we give a special treatment to the multispin coding direction where 64 spins are stored in one word, whereas the other directions are treated with helical boundary conditions.

While the above method saves memory by a factor of up to 64, and also a lot of computer time, its disadvantage is that different FORTRAN compilers treat these bit-by-bit handling functions differently. Instead of **LEFT.OR.RIGHT**, an IBM mainframe may require **IOR(LEFT,RIGHT)**, and the shift commands are not always circular shifts. The expected new FORTRAN standard may unify such functions (presumably by requiring **IOR**), but it will be a long time until it is implemented everywhere. Other languages like C may be more standardized in this respect. We now follow the way FORTRAN can be programmed on a Cray vector computer; the interested user should first try out how his compiler deals with shifts and bit-by-bit operations.

For such simple rules in one dimension, speeds of 4.5 updates per nanosecond have been reached on one processor of a Cray-YMP [15]; if all 8 processors of that machine were used simultaneously (e.g. on 8 lattices with different initial conditions), the speed thus would have been 34 updates per nanosecond. In two dimensions, four processors of a Cray-2 and ETA reached 4 and 6 updates per nanosecond, respectively, for the Q2R automata to be discussed below [15, 16]. Thus authors should always quote updates per processor, or give the number of processors used, when giving their computation speeds.

The program of figure 1 for a three-dimensional **OR** updates nearly 1.4 sites per nanosecond on one Cray-YMP processor; it is also fully vectorizable. It starts with various size-dependent parameters and a data line containing the input values as desired by the user. The first loop sets the spin array **N** equal to zero everywhere; note that this double loop has the indices I and J reversed to have the one going over the larger range as the innermost loop, a standard trick to increase efficiency for vector computers.

In the next loop, the random number generator **RANF** sets each bit of the array **N** with probability p; otherwise the bit remains zero. (The integer part of **P+1-RANF** is 1 with probability P and 0 with probability 1-P, if the random numbers **RANF** are distributed homogeneously between 0 and 1.) To avoid vector dependencies the bit loop is the outermost loop.

Now follows the main loop for times between 1 and **MAX**; our time unit is one sweep through the lattice. The five loops within this time loop have the following meaning. The first loop counts as **MAG** the number of computer words **N** which are not yet completely infected. The second loop fills two buffer planes to ensure periodic boundary conditions in a simple way: the top buffer plane contains the information stored in the lowermost physical plane, and the bottom buffer plane repeats the information in the uppermost real plane. The third loop deals similarly with two buffers in the direction of multispin coding; here the same shift operations occur which were needed in the above example for **LEFT** and **RIGHT**. The fourth loop is the core of the program and deals with the logical **OR** over the six neighbours; that information is stored in the second array **M**; the six neighbours have the indices $\mathrm{I} \pm 1$ in the multispin direction, and $\mathrm{J} \pm 1$ and $\mathrm{J} \pm \mathrm{L}$ for the four other directions. In the fifth and last loop, after the calculation of all **M** words, the original array **N** is updated by the contents of the newly calculated **M**.

```
      PARAMETER(LL= 7,LL1=LL+1,NBIT=64,L=LL*NBIT,
     1           LS=L*L,LP=LS+L,LM1=L-1)
      DIMENSION N(-LM1:LP,0:LL1),M(LS,LL)
      DATA P,MAX,ISEED /0.001,100,123456789/
      PRINT *, L,P,MAX,ISEED
      CALL RANSET(2*ISEED-1)
      PP1=P+1.0
      DO 1 I=1,LL
      DO 1 J=1,LS
1     N(J,I)=0
      DO 2 NB=1,NBIT
      DO 2 I=1,LL
      DO 2 J=1,LS
2       N(J,I)=OR(SHIFT(N(J,I),1),INT(PP1-RANF()))
      DO 3 ITIME=1,MAX
        MAG=0
        DO 4 I=1,LL
        DO 4 J=1,LS
4        IF(NOT(N(J,I)).NE.0) MAG=MAG+1
        IF(MAG.EQ.0) STOP
        PRINT *, ITIME,MAG
        DO 5 I=1,LL
          DO 5 J=1,L
            N(J+LS,I)=N(J,I)
5           N(J-L ,I)=N(J-L+LS,I)
        DO 6 J=1,LS
          N(J, 0 )= SHIFT(N(J,LL),NBIT-1)
6         N(J,LL1)= SHIFT(N(J,1),  1)
        DO 7 I=1,LL
          DO 7 J=1,LS
7           M(J,I)= N(J,I-1).OR.N(J,I+1).OR.N(J-1,I).OR.N(J+1,I)
     1              .OR.N(J-L,I).OR.N(J+L,I)
        DO 8 I=1,LL
          DO 8 J=1,LS
8         N(J,I)=M(J,I)
3     CONTINUE
      END
```

Figure 1. Infection program (logical OR) for cubic lattice.

A disadvantage of the whole multispin coding approach is that very small lattices (smaller than L = 64 in our case) cannot be simulated. For large lattices, on the other hand, memory requirements could be reduced even further by storing only suitable planes of the auxiliary array **M**, and not the whole lattice. Often one may also relax the requirement of fully parallel updating and thus have **N** depend directly on **N**.

By simulating this trivial infection process the user may observe how long it takes until all sites are infected. Since at every time step, each infected site infects all its neighbour, this maximum time is given by the biggest 'hole' of uninfected sites in the random initialization and increases logarithmically with lattice size [7]. For L = 896 (one sample, one minute Cray processor time) I found that 31 sweeps are needed to infect the whole lattice if initially a random fraction of 0.1 per cent of all sites are infected.

More complicated rules than this logical OR may need more complicated combinations of logical operations. For example [8], if the centre spin is up if and only if at least m of its six neighbours are up, then one has to go through all possible combinations of up (or down) spins, and ends up with about twenty **AND**, OR and **NOT**. In principle, this computational effort increases exponentially with the number of neighbours to be taken into account. (Computer time may increase somewhat less since, in the above simple example, memory access may be a bottleneck because only few operations are made with each word.)

2.3. Many rules in one program

On a lattice with K neighbour spins, of which each can be either up or down, we can find $C = 2^K$ neighbour configurations. For each of these C configurations, the rule may require the centre spin to be up or to be down at the next time step; thus we have in total $R = 2^C$ possible rules. On the simple cubic lattice with $K = 6$ neighbours we thus have $R = 2^{64}$, or more than 10^{19}, different types of cellular automata. I was unable to write a separate computer program for each of them. Even on the square lattice, $K = 4$, where many of the $R = 2^{16} = 65536$ differerent rules follow from each other by rotations or reflections of the lattice, or by spin inversion, it is hardly possible to write a separate algorithm for each of the 4856 different groups of rules. Thus a unified, though slower, algorithm is needed to treat numerous different rules by one program: the algorithm of da Silva and Herrmann [9]. Again, each bit corresponds to a different site and neighbouring sites are stored in different computer words, just as above.

We explain this algorithm for the square lattice and for a completely random mixture of automata rules (the Kauffman model, see last chapter). Thus at the beginning each site selects which of the 65536 rules on the square lattice with four neighbours it wants to obey, and then it sticks to this rule. Therefore one program has to accomodate all sites and all their rules. With the help of eight vertical and horizontal variables `N1H`, `N1V`, `N2H`, `N2V`, `N2H`, `N2V`, `N3H`, `N3V`, `N4H`, `N4V` we find out which of the sixteen possible neighbour configurations surrounds a given site. For example, `N1V` is true if and only if both vertical neighbours are up, and `N3H` is true if the left neighbour is up and the right neighbour is down. The logical `AND` of `N1V` and `N3H` thus is true if and only if the right neighbour is down and all other three neighbours are up. Of the sixteen logical `AND`s that are formed by combining one of the four horizontal with one of the four vertical variables, exactly one is true and the fifteen others are false. The true combination indicates which neighbour configuration is realized. These statements make up the first third of the loop beginning with `DO 11 I=1,L` in our figure 2.

In the second part of that loop we calculate the logical `AND` of each of these sixteen combinations with the corresponding rule `NR` of that site; again of these sixteen `AND`s, only one is true and all others are false. The logical or arithmetic sum (`OR` or +) of these sixteen `AND`s thus gives the new value `M`. In the following loop `14` the old array `N` is updated by the newly calculated values `M`, just as in the simple infection program. (We are allowed to use arithmetic instead of logical sums here since only one of the sixteen terms to be summed over is one and all others are zero. Normally in one-bit-per-site multispin coding, neither additions nor multiplications are allowed since they mix different bits.)

Also the other parts of the algorithm are similar to the infection program of figure 1. The loop starting with `DO 1 K=1,16` is new: here we select randomly for each site separately the rule it wants to follow. Thus we go through all sixteen possible neighbour configurations, and with probability `P` we select the rule that for this configuration the centre spin should point up; otherwise it points down. We thus have a random but deterministic mixture of all possible cellular automata. The last loop `16` determines how many spins point up, using the function `POPCNT` which counts the number of up bits in a word. More interesting applications are discussed in our last chapter. The speed of this program, on one Cray-YMP processor (`L = 1280`), was 235 updates per microsecond.

Additional speed-ups to 285 sites per microsecond are possible if all sites obey the same rule, but one program has to go through thousands of different cellular automata rules in order to classify them. Then we perform the arithmetic sum in the last part of loop `11` (figure 2) only over those neighbour configurations where the rule gives spin up. Thus loop `11` is divided into two parts; in the first part we calculate and store `N1V`,

```
      PARAMETER (LL= 5,L=64*LL,LP1=L+1,LL1=LL+1)
      DIMENSION NR(0:LP1,LL,16),N(0:LP1,0:LL1),M(L,LL)
      DATA P,MAX,ISEED/0.27,200,1/
      PRINT *, P,MAX,L,ISEED
      CALL RANSET(1+ISEED*2)
      PP1=1.0+P
      DO 1 J=1,LL
        DO 2 I=0,LP1
2         N(I,J)=0
        DO 3 II=1,64
          DO 3 I=0,LP1
3           N( I, J) =SHIFT(OR(N(I, J) , IFIX(1.5-RANF())),1)
        DO 1 K=1,16
          DO 5 I=1,L
5           NR(I,J,K)=0
        DO 1 II=1,64
          DO 1 I=1,L
1           NR(I,J,K)=SHIFT(OR(NR(I,J,K),IFIX(PP1-RANF())),1)
C     END OF INITIALIZATION, START OF TIME DEVELOPMENT
      DO 7 ITIME=1,MAX
        DO 8 I=1,L
8         N(I, 0 ) =SHIFT(N(I,LL),63)
        DO 9 I=1,L
9         N(I,LL1)=SHIFT(N(I, 1), 1)
        DO 10 J=1,LL
          DO 11 I=1,L
            N1H=AND(    N(I,J+1) ,    N(I,J-1))
            N2H=AND(    N(I,J+1) ,NOT(N(I,J-1)))
            N3H=AND(NOT(N(I,J+1)),    N(I,J-1))
            N4H=AND(NOT(N(I,J+1)),NOT(N(I,J-1)))
            N1V=AND(    N(I+1,J) ,    N(I-1,J))
            N2V=AND(    N(I+1,J) ,NOT(N(I-1,J)))
            N3V=AND(NOT(N(I+1,J)),    N(I-1,J))
            N4V=AND(NOT(N(I+1,J)),NOT(N(I-1,J)))
11        M(I,J)=AND(NR(I,J, 1), AND(N1H,N1V))
     1          +AND(NR(I,J, 2), AND(N1H,N2V))
     1          +AND(NR(I,J, 3), AND(N1H,N3V))
     1          +AND(NR(I,J, 4), AND(N1H,N4V))
     1          +AND(NR(I,J, 5), AND(N2H,N1V))
     1          +AND(NR(I,J, 6), AND(N2H,N2V))
     1          +AND(NR(I,J, 7), AND(N2H,N3V))
     1          +AND(NR(I,J, 8), AND(N2H,N4V))
     1          +AND(NR(I,J, 9), AND(N3H,N1V))
     1          +AND(NR(I,J,10), AND(N3H,N2V))
     1          +AND(NR(I,J,11), AND(N3H,N3V))
     1          +AND(NR(I,J,12), AND(N3H,N4V))
     1          +AND(NR(I,J,13), AND(N4H,N1V))
     1          +AND(NR(I,J,14), AND(N4H,N2V))
     1          +AND(NR(I,J,15), AND(N4H,N3V))
     1          +AND(NR(I,J,16), AND(N4H,N4V))
10      CONTINUE
        DO 14 J=1,LL
          DO 14 I=1,L
14          N(I,J)=M(I,J)
        DO 15 J=1,LL
          N( 0, J)=N(L,J)
15        N(LP1,J)=N(1,J)
7     CONTINUE
      MAG=0
      DO 16 J=1,LL
        DO 16 I=1,L
16        MAG=MAG+POPCNT(N(I,J))
      PRINT *,MAG
      END
```

Figure 2. Kauffman program (disordered mixture of all cellular automata) for square lattice. (H A Wischmann speeded it up further to 260 updates per microsecond and on a Cray processor by defining `N1H`, `N2H`, `N3H` and `N4H` not through the opposite neighbours `(i,j+1)` and `(i,j-1)` but through the diagonal neighbours `(i+1,j)` and `(i,j+1)` of site `(i,j)`. These four variables are stored and reused for site `(i+1,j+1)`. The vertical variables like `N1V` are no longer needed.)

`N1H`, ..., `N4H` for each site; in the second part an outer loop over the sixteen neighbour configurations contains an innermost loop, executed only if the rule gives an up spin, running over all sites. The corresponding program has been published elsewhere [10], and even a whole book has been devoted to such multispin coding methods [11].

2.4. Hydrodynamic cellular automata

Thus far each site carried only one spin; now we deal with six to eight bits needed per site in connection with hydrodynamic cellular automata [12], also called lattice gases in the literature. A complete description of a classical fluid would let each molecule move under the influence of the forces the other molecule exert on it: molecular dynamics with Newton's law. Such simulations consume lots of hours and megabytes, but have been applied successfully to simple hydrodynamic problems [13]. Orders of magnitude faster, and also less memory consuming, is the simplified algorithm on the basis of complicated cellular automata. Molecules are allowed to travel with unit speed along the nearest-neighbour bonds of the triangular lattice; they scatter at integer times on the lattice site if another molecule is present there at the same time. These scattering events preserve momentum and also (for some algorithms) energy and angular momentum. Various programming methods have been published in detail [14]; we follow that of Kohring here, which is one of those storing 64 different sites in one Cray word and using only logical operations.

The six directions leading to a given site are represented by six different bits stored in six different computer words `X1`, `X2`, `X3`, `X4`, `X5` and `X6` (clockwise or counterclockwise). If the corresponding bit is one, an atom flies with unit velocity towards the site on this bond; for a zero bit, there is no particle coming at this moment from this direction. Thus the density can vary. Collisions of two, three and four particles are taken into account; collisions resulting only in an exchange of particles (like head-on collisions of two atoms leading to reflection by 180 degrees) are ignored since we do not keep track of the identity of the particles. (To study dispersion, this simplification has to be avoided.) Figure 3 shows some of these collisions. In order to determine if in a two-body collision the direction of the outgoing atoms is rotated positively or negatively compared with the incoming direction, a seventh bit is set initially at random to represent positive or negative angular momentum. This angular momentum determines how the direction of the outgoing particles is rotated compared to the incoming direction; after such a collision the angular momentum bit is reversed.

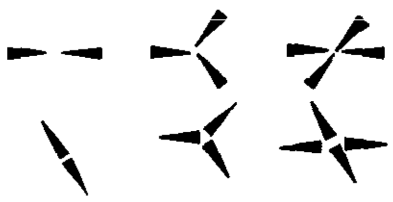

Figure 3. Collisions in hydrodynamical cellular automata. The top velocities are rotated into the bottom velocities.

Two- and four-body collisions are impossible if at least one pair of opposite directions has different status (e.g. if a particle is coming from the left but no particle from the right). Different bits are found by the `XOR` function representing exclusive-or. Three-body collisions require directions 1, 3, 5 to have the same status and also directions 2, 4, 6 to have the same status; again a positive `XOR` test destroys the possibility for a three-body collision. Thus with purely logical bit-by-bit operations we find a computer word `COL` telling us if a collision occurs. Again, `COL` stores 64 different sites, and neighbouring sites are stored in different words, just as in our earlier examples.

With the help of the collision word `COL`, the angular momentum bit `ANG`, and the incoming particle directions X1, ..., X6 we now can determine the outgoing directions Y1, Y6. For example, after a collision we have Y3 = X2 or = X4, depending on the angular momentum bit, whereas Y3 = X3 if no collision happened. The expression

```
Y3 = (X2.AND.COL.AND.ANG)+(X4.AND.COL.AND..NOT.ANG)+(X4.AND..NOT.COL)
```

takes into account these mutually exclusive collision possibilities. Having thus calculated the outgoing velocities Y1, ..., Y6, the program then transfers the Y information about outgoing atoms into the new X variables for incoming atoms at the corresponding neighbour site; also the angular momentum bit must be reversed wherever a collision occured. One NEC-SX3 processor updated about 500 sites per microsecond with Kohring's FORTRAN program [14], whereas for all 65 536 processors together of a Connection Machine the speed was about twice as high (assembler programming, Bhogosian [14]).

3. Simulation results

3.1. General classification

Wolfram [2] made the first systematic classification of one-dimensional cellular automata, and that classification can be regarded as the starting point of the scientific investigation of cellular automata in general, as opposed to studies of specific examples. Thus we start here with a short report on how to classify automata in two and three dimensions [17].

Wolfram distinguishes four classes according to the asymptopic behaviour after very long times: in class 1 the spins end up in a fixed point and are all parallel, limit cycles with short periods are observed in class 2, the chaotic class 3 has infinite periods, and propagating structures are seen in class 4. To make such a classification programmable in an automatic computer search for thousands of different rules, we need precise and mutually exclusive criteria replacing Wolfram's general concepts. We divide the Wolfram classes 1 and 2 into subclasses. In this way five groups can be easily identified, with the sixth group containing all rules with non-identified asymptotic behaviour.

Group 0: fixed point with all spins down
Group 1: fixed point with all spins up
Group 2: fixed point with some spins up, some spins down
Group 3: oscillations of period two between all spins up and all spins down
Group 4: oscillations of period two with some spins up and some spins down
Group 5: all other behaviour.

Initially we set randomly half of the spins up and half down, and use large lattices the length of which is not exactly a power of two (since powers of two lead to special behaviour).

Results are available for chains (two neighbours), honeycomb (three neighbours), square (four neighbours), triangular (six neighbours), and simple cubic (six neighbours) lattices [17]. The larger the number of neighbours, the larger the fraction of rules belonging to the 'unidentified' class 5. For six neighbours, only one per cent of all rules fit into classes 0 to 4; of course, one per cent of 2^{64} is still a huge number of rules. Of the 16 one-dimensional rules, on the other hand, two each belong to classes

0, 1, and 4. On a square lattice, the percentages for classes 0 to 4 are 2.8, 2.8, 0.1, 0.7, and 2.1. (A large class for two to four neighbours with 16 per cent on the square lattice are the rules which lead to translations of the whole configuration; however, that class partially overlaps with oscillations of period two, if the spin configuration is checkerboard-like on a square lattice. Reference [21] speculates that Wolfram's class 4 quite generally might be class 2 with long transient behaviour.)

Quite similar is the situation if we check for stability against small perturbations [18]. In politics it is difficult to find out if one single decision (e.g. the selection of a presidential candidate) later leads to an observed result (e.g. the loss of the election). Computer scientists have it easier to investigate such relations between cause and effect: they simulate the system twice, once with the suspected cause and once without it, and then later compare the two resulting configurations to find the true effect of that cause.

Thus we start from two lattices obeying exactly the same rules and having exactly the same initial configuration; only on one site or along one lattice line, the spin or the rule is changed in one replica compared to the other. Then the simulation proceeds, and we compare site-by-site the resulting configuration to see how this initial 'damage' spreads later. Such damage spreading questions are very traditional in classical mechanics where an energy minimum corresponds to stability (damage dies out due to friction) and an energy maximum to instability (damage increases, often exponentially). Chaotic systems have the damage spreading over the whole lattice; for example, an apple dropping in a Californian orchard later changes the weather all over Europe, and even modifies much later the decay of the solar planetary system if the latter is chaotic. In this sense we call cellular automata stable, unstable, and marginal depending on whether the initially localized damage dies out, spreads over the lattice, or goes neither to zero nor to infinity. (Spreading over the lattice is usually measured by checking if the damage 'cloud', the set of sites different in a comparison of the two replicas, has touched the boundary of the lattice far away from the initial perturbation.)

Simulations [17] show that in chains, honeycomb and square lattices, about 60 per cent of all rules are unstable; for six neighbours that percentage increases to 98.5. Thus the more neighbours one has to deal with, the more chaotic is life.

3.2. Q2R ferromagnetism

Apart from hydrodynamic cellular automata, the most studied single rule [19] presumably is Q2R. A spin is flipped if and only if it has as many up as down neighbours. These cellular automata give the spontaneous magnetization of the Ising model, according to present numerical knowledge. The condition for flipping spins means that the Ising interaction energy is not changed by a spin flip; thus Q2R is a microcanonical Ising model simulation, complementing the traditional canonical (Kawasaki) or grand canonical (Glauber) simulation methods for Ising magnets. In contrast to earlier hopes, it is not a particularly efficient way to find Ising properties, but instead it is useful for teaching and illustrates quantitatively certain basic ergodicity and irreversibility problems of statistical physics.

We can simulate Q2R by going sequentially or randomly through a lattice and flipping a spin if its number of up neighbours equals the number of down neighbours. Then the energy, measured through the number of antiparallel neighbour pairs, is constant. If instead we use simultaneous updating as is customary for cellular automata, we have to be more cautious. Imagine, for example, a very long chain where all spins except those at site 102 and 103 are up; spins 102 and 103 are down. Thus we have two broken bonds (antiparallel neighbours), between 101/102 and between 103/104. If we would update the spins sequentially, i.e. first spin 1, then 2, etc, we would flip 101 and 102 and still end up with two broken bonds: energy conserved. Simultaneous updating, however, means that spins 101, 102, 103, and 104 are all flipped at the same time since they 'do not yet know' that their neighbour is flipped, too. Then we end up with two isolated down spins 101 and 104 and four instead of two broken bonds: energy increased. Therefore in Q2R automata energy conservation has to be restored by dividing the lattice into two sublattices, like even and odd spins, in such a way that spins on one sublattice are neighbours only to spins on the other sublattice. Thus one time step means that we first update one sublattice simultaneously, and then the other sublattice simultaneously. In the above example, after updating the odd spins we have flipped 101 and 103, and then the even spins 100 and 102 are flipped. The whole iteration thus leads to a pair of down spins at sites 100 and 101, surrounded by up spins: two broken bonds as before, and energy is conserved.

The model is not ergodic, i.e. even if we wait infinitely long we do not get through all possible states with the same energy. That number, with roughly half spins up, varies as $2^N/\sqrt{N}$ for a system of N spins. Q2R is a reversible rule, and thus after a certain time the system returns to its starting configuration, after which it repeats again and again the same time development: 'limit cycle'. If the period of that cycle would be of the order 2^N, ergodicity would be ensured. Simulations gave [20], however, a median limit cycle period varying only as $2^{N/4}$. Thus even if we start from a random distribution, corresponding to infinite temperature, we go only through an exponentially small fraction of the available phase space: no ergodicity. Nevertheless the spontaneous magnetization comes out correctly as in the Ising model. Thus ergodicity is not needed to get good results.

(For comparison with Ising results we start from a random configuration with a concentration p of up spins. The thermal energy or number of broken bonds is then proportional to $p(1-p)$. The temperature, on the other hand, can be found from this energy by traditional Ising model calculations. In this way, the Curie temperature of the square lattice corresponds to a concentration p of 7.955 18 per cent.)

Because of the reversible nature of the spin flip rule we see a similarity to Newton's law of motion, where also no time arrow is preferred: flipping Q2R spins or letting billard balls collide is a reversible process. Nevertheless, molecular dynamics simulations of many particles lead to irreversible relaxation into an equilibrium, and simulation of Q2R automata also lets the magnetization relax towards its Ising equilibrium value. How is this contradiction solved: the period after which Q2R returns to its initial configuration increases exponentially with N, though not as 2^N. And for $N = 10^{24}$, the period $2^{N/4}$ is much longer than the age of the universe. However, for *small* systems, particularly for p below 0.5, the return to the initial random distribution is easily observable [20] because of the discrete nature of cellular automata and the lack of any rounding errors which plague molecular dynamics studies. Thus for small systems one can study explicitly that the second law of thermodynamics is violated; it is an approximation valid only for large systems.

In other aspects Q2R is a good teaching tool: its simulation requires little knowledge of traditional physics like probabilities proportional to $\exp(-E/kT)$. It can be studied on powerful pocket calculators and gives then a spontaneous magnetization for low enough p.

Problems occur only if one desires great accuracy; then one realizes that relaxation to equilibrium is very slow for p below the Curie point concentration (8 per cent for square lattice). We can quantify this non-critical slowing down by starting from a random configuration of concentration p and by measuring the time τ after which the actual magnetization reaches the midpoint between the initial magnetization $2p-1$ and the final Ising magnetization. This time does not diverge for ordinary critical slowing down. For Q2R on the square lattice, our data in figure 4 show for decreasing p first a maximum of τ near the Curie point of $p = 8$ per cent, followed by a minimum of $\tau \sim 10^3$ near 7.6 per cent, and then for decreasing p the relaxation time τ increases rapidly to reach 4 to 5 million at 5 per cent. (This interval in concentrations corresponds to temperatures a few per cent below the Curie temperature.) A Vogel–Fulcher law like for glasses would predict

$$\tau \propto \mathrm{e}^{\mathrm{const}/(p-p_0)}$$

and indeed figure 4 at first suggests such a behaviour with p_0 near 3 per cent. Closer inspection, however, indicates a possibly s-shaped curve, instead of a straight line, for the data plotted in figure 4, and thus this apparent dynamical transition at 3 per cent may be shifted downwards to lower concentrations, perhaps to $p = 0$.

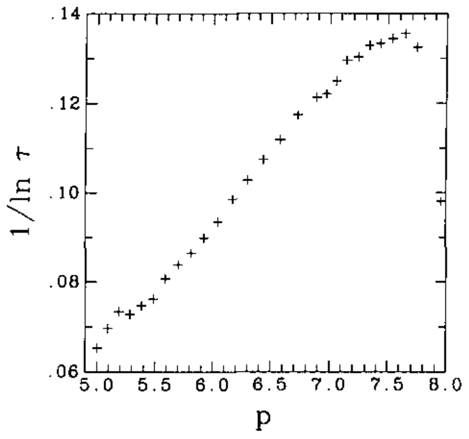

Figure 4. Relaxation time τ against initial concentration p for Q2R on the square lattice. We plot $1/\ln(\tau)$ against p.

Such effective phase transitions, which shift slightly towards $p = 0$ (or $p = 1$) if the computing effort is increased exponentially, are also known from bootstrap percolation [8], where a threshold may go to zero with some negative power of the $\log(N)$ only. Similar to the above-mentioned problem with the second law of thermodynamics, it may then physically be more relevant to study the *effective* transition at a finite p_0 than the mathematically exact limit for infinite times or systems. Obviously, times and sizes larger than those of the universe correspond to sound mathematics but not to good physics: for a physicist, the logarithm of infinity is below 100.

Other criteria have also given effective transitions near a few per cent initial concentration of up spins; some of these transitions seem to vanish logarithmically while others do not [22]. No clear picture has yet emerged. From the practical point of view it is remarkable that the slow relaxation into equilibrium at low p can be avoided if one starts with a normal Ising model simulation (Glauber–Metropolis) and lets Q2R run only after equilibration [23].

From the technical point of view, Q2R was one of the first cases where the one-bit-per-site technique was applied (Herrmann [6]). If the four computer word `I,J,K,M` contain the four neighbours of a site (with 64 sites in each of the 64 bits), then

```
((I.XOR.J).AND.(K.XOR.M)) .OR. ((I.XOR.K).AND.(J.XOR.M))
```

is the condition to flip the centre spin. A fully vectorized program reaches 1.7 updates per nanosecond on one Cray-YMP processor, with the program published elsewhere [24] as a simplification of Herrmann's program.

3.3. Hydrodynamic cellular automata

The hydrodynamical methods described at the end of the first section have been applied to a variety of two-dimensional flow problems, and generalized to more complicated models and to three dimensions. On the fundamental side, the viscosity of two-dimensional fluids was shown numerically [25] to diverge logarithmically with system size, as predicted theoretically long before. In this sense the cellular automata approach is superior to the application of Navier–Stokes equations since there one *assumes* the existence of a finite viscosity. On the other hand, very large velocities (large Reynolds numbers for turbulence) cannot be treated correctly by the automata method.

For low velocities, the flow through porous media seems to be a good field of application [26] where traditional methods work less well. If the open pore space is no longer geometrically connected, then no flow is possible at all; however, already far away from this percolation threshold, the tortuosity of connected paths of pores is so high that the hydrodynamic permeability is strongly reduced compared to that in free flow. (The permeability basically is the ratio of hydrodynamic current to pressure gradient according to Darcy's law.) For specific well defined geometries, good agreement of laboratory and computer experiments was obtained.

In three dimensions, a simple cubic lattice does not work well, and instead normally a projection of a four-dimensional face-centred lattice is used [27]. Nevertheless, speeds of 30 to 40 updates per microsecond on one Cray processor were obtained. However, realistic applications to the flow of oil through a porous medium are quite difficult. If we require the sand grain radius to be at least hundred times bigger than the molecular size (lattice constant) and at least ten times smaller than the channel width, then we need lattices at least of size 1000^3 and thus computer times three orders of magnitude above present typical simulations. Future 'teraflop' machines would have here a nice field of practical applications, since they cost about as much as a single offshore oil drilling attempt.

The above simulations worked with one type of molecule. Two-fluid models have also been studied [28], to simulate oil-water mixtures or spinodal decomposition. We refer to a recent conference for more details and applications [29].

4. Biologically motivated cellular automata

4.1. Kauffman model

The Kauffman model was invented [30] to simulate the interaction of biological genes. Humans have of the order of $N = 10^5$ different genes; these genes are the same in each different cell of our body. However, genes can be turned on (spin up) or off (spin down) according to interactions between them. In total this leads to 2^N possible different genetic setups. In reality, we have only of the order of 1000 different cell types, and a good model should explain why only one thousand of the huge number of possibilities are realized.

Living beings differ from Intel 860 chips in that they are not mass-produced in automated factories. Thus the interactions between different genes should be much more random than between computer parts. Kauffman [30] thus assumes that each gene selects randomly which of the many possible rules for cellular automata it wants to obey. If each gene is influenced by $K = 4$ neighbours, then we have $2^{16} = 65536$ possible rules, and thus most of the $N = 10^5$ genes can follow a different rule each. Once selected randomly, these rules are fixed forever, and thus the system ends up in limit cycles. Kauffman identifies the different limit cycles a system can have with the different cell types your body or mine has (brain cells, fat cells, ...) and finds their number to increase only as $\sqrt{N}$ for N interacting genes, as desired.

Mutations are random changes of a gene or of a rule by which the gene is influenced. A mutation that changes the whole genetic setup and e.g. changes a computational scientist into an exact mathematician is called a catastrophic mutation; biologically useful mutations have much less drastic effects. This question of the influence of mutations is just our 'damage spreading' of the general classification; in fact, Kauffman seems to have introduced that question into biology long before physicists have studied it. (Statistical physics normally deals with averaged quantities like the density, not with particular configurations as needed for damage spreading studies.)

Most of the early work on the Kauffman model dealt with an infinite range of interaction, as seems relevant biologically [31]; then each gene selects randomly which K out of the $N - 1$ other genes will influence its behaviour. Some sort of mean-field theory [32] becomes exact for infinite range interactions in infinitely large systems: for large K the system is unstable against damage spreading (catastrophic mutations), for small K it is stable, and $K = 2$ is the border case.

More relevant to cellular automata are simulations of the Kauffman model on a lattice with nearest neighbour interactions. Then on a square lattice each site only selects which of the 65536 possible rules it wants to obey; the neighbours are already fixed by the lattice structure. In one dimension and also on the honeycomb lattice, the system is always stable against mutations whereas for the square, triangular, cubic and four-dimensional hypercubic lattice, catastrophic mutations are possible where the damage spreads over the whole lattice [33]. A program for the square lattice was described in subsection 2.3.

The probability for damage to spread can be reduced to zero by introducing a new parameter p, which here is no longer the initial concentration of up spins but the probability of selecting a rule with the result: spin up. More precisely, during the initialization we select a new site to get its spin orientation and its rule. First by calling one random number we determine if the spin is up or down. Then we go through all possible neighbour configurations, 16 for the square lattice ($K = 4$ neighbours), draw a new random number between 0 and 1 for each, and select for this neighbour configuration the result spin up if the random number is smaller than p; otherwise the result shall be down. Having dealt in this way with all (16) neighbour configurations, we have finished the determination of the rule for this one site. Then we jump to another site.

Obviously, p and $1 - p$ are statistically equivalent, and we ignore $p > 1/2$. For $p = 0$ after one iteration all spins are down and remain so forever. Thus $p = 0$ makes the system stable against damage spreading, even if at $p = 1/2$ (the unbiased case we started with) damage could spread. At some threshold between 0 and 1/2 therefore a transition to unstable behaviour can occur. Life in this sense requires a p below the threshold p_c to unstable behaviour.

Numerical estimates for p_c in the square lattice were plagued by strong finite-size effects and changed over three years [34] from 0.26 to 0.31; for size $6976 * 6976$ the effective threshold was near 0.305. The estimates $p_c = 0.16$, 0.12, and 0.08 for the triangular, simple cubic and hypercubic lattices [33] may still contain systematic errors since there the lattices sizes were smaller than in the square lattice.

The threshold p_c to unstable behaviour seems to be a second-order phase transition with critical phenomena: the probability that from a single damaged site in the lattice centre the damage spreads to the lattice boundary is zero below the threshold, non-zero above it, and goes to zero continuously if the threshold is approached from above. Thus right at the threshold, fractal behaviour (finite size scaling) sets in. The number of damaged sites at the moment the damage touches the boundary varies as L^D in a lattice of linear dimension L; the time the damage needs to reach the boundary varies as $L^{D'}$. On the square lattice, D is about 1.9, roughly compatible with the percolation fractal dimension [34], whereas D' is about 1.3. For other lattices we refer to [33]. Again, these other estimates may be less reliable since the square lattice estimates for D increased from 1.5 to 1.9 with the availability of better computers and better algorithm, a nice example of the need for supercomputing.

Finally we mention that the Kauffman model is related to neural networks and cellular automata of a more complicated kind [35].

4.2. Immunology

No real supercomputing has been needed so far for immunological speculations since there no phase transitions with critical phenomena have been found. The biological relevance of the cellular automata approximation is still controversial in this field where traditional descriptions mostly rely on first-order nonlinear differential equations as known from population dynamics. Nevertheless the application of statistical physics (percolation) ideas to immunology has a tradition of nearly 40 years [36].

If we get influenza in one winter it is unlikely we get the same sickness again in the same winter: we have become immune against this disease. Smallpox vaccination is one of the major successes of immunology. Many cell types and molecules play an important role in the immune response. Antibodies (`A`) come from bone-marrow derived lymphocytes (`B`) and neutralize the virus or other antigen (`V`), helper cells `H` and suppressor cells `S` regulate the immune response in a positive or negative way, killer cells `K` may attack the own body, interleukin molecules `I` mediate the connection between `H` and `S`, the AIDS virus `D` destroys the immune system, etc. Antibodies roughly correspond to one antigen like lock and key, and one may find a second type of antibodies which regard the first one as antigens, and so on in Jerne's network of lock-and-key relations. The review collection *Theoretical Immunology* [37] gives an overview over the different aspects and approaches of this field.

The KUT model [38] presumably was the first cellular automata description of the immune response; a larger variety of models was investigated only after a publication in, of course, *J. Phys. A: Math. Gen.* [39]. In all these cellular automata approximations, the concentrations of various cell types can be either high (spin up, true, 1) or low (spin down, false, 0). The interactions between the various cell types can be defined by Boolean relations, like `S = S .OR. H`, meaning that at the next time step the concentration of suppressor cells is high if and only if before there were lots of suppressor cells or lots of helper cells present. Sometimes an analogy is helpful which identifies the logical **AND** with a multiplication ($*$) and the logical **OR** with an addition (+). The **NOT** function (negation) is symbolized by a bar on top of the symbol. Then `H .OR. (V .AND..NOT. S)` is written as $\mathtt{H} + \mathtt{V} * \overline{\mathtt{S}}$, using that multiplication has precedence over addition.

With this arithmetic notation of logical relations, the KUT model for normal immune response [38] with some simplifications [40] reads for antibodies **A**, suppressors S, helpers H, B cells B, and virus V:

$$\mathtt{A} = \mathtt{V} * \mathtt{B} * \mathtt{H} \qquad \mathtt{S} = \mathtt{H} + \mathtt{S} \qquad \mathtt{H} = \mathtt{H} + \mathtt{V} * \overline{\mathtt{S}} \qquad \mathtt{B} = \mathtt{H} * (\mathtt{V} + \mathtt{B}) \qquad \mathtt{V} = \mathtt{V} * \overline{\mathtt{A}}.$$

The initial condition is defined by the five spins A, S, H, B, V; thus $2^5 = 32$ different initial conditions are possible and are best dealt with by a very simple computer program. Each time step corresponds to 10^2 hours in reality; thus in contrast to fluids and magnets, here computers really beat nature in speed.

As a result we find five fixed points for the variables (ASHBV): (00000), (01100), (01110), (01000) and (01001). The first case is the naive state where no immune reaction has ever happened. The second and third fixed point correspond to the vaccinated or immune state; adding a virus V to these states will quickly lead back to the immune state (01110) where the virus is again destroyed. This is not the case for the fourth fixed point (01000) where an addition of virus leads immediately to the fifth fixed point (01001). Thus for these last two fixed points the body is not protected against infection: the virus or other antigen stays on.

In autoimmune diseases like multiple sclerosis or diabetes mellitus, killer (effector) cells K of the immune system attack the own body as if that would be an antigen. For the simplest three-cell model involving K, H, and S only, Chowdhury [40] let a computer search through all $3^9 = 19683$ possible models with positive, negative or missing interactions (including self-interactions) between the three cell types. Applying further constraints, like that helper cells should not suppress, the computer gave just one model:

$$\mathtt{S} = \mathtt{H} \qquad \mathtt{H} = \mathtt{H} + \mathtt{K} \qquad \mathtt{K} = \mathtt{K} + \mathtt{H} + \mathtt{K} * \mathtt{H} * \mathtt{S}$$

which gave the desired fixed points (SHK) = (000) (healthy), (110) (immune), and (111) (sick). Also five-cell models for autoimmunity were invented [39–41], which may be more realistic biologically.

Several AIDS models were studied [42,43] where the HIV virus, abbreviated as D, destroys the helper cells (or T4 cells) of the immune system. The last model [43] uses interleukin molecules I as messengers from helper cells H to cytotoxic cells S:

$$\mathtt{S} = \mathtt{I} \qquad \mathtt{H} = \mathtt{I} + \overline{\mathtt{D}} \qquad \mathtt{I} = \mathtt{H} \qquad \mathtt{D} = \mathtt{H} + \overline{\mathtt{S}}.$$

This dynamics gives for (SHID) the fixed points (0001) and (1111) as well as an oscillation of period two between (1101) and (0011). In the first fixed point, the virus D has completely destroyed the immune system, and the body will succumb to some normally not deadly disease like pneumonia. For the other states, the virus never vanishes but the immune system is still fully or partially present; these states could correspond to the many years of latency (correponding to about 10^3 iterations) after an infection with the AIDS virus before the full sickness breaks out.

However, the model does not allow for these metastable states to decay after thousand iterations to the true fixed point (0001). To allow such decay, the interleukin production I was allowed to fail with a low probability p, and then after strongly fluctuating times of the order of 1/p the final fixed point (0001) was always reached. Similar results were obtained when these logical equations were replaced by differential equations. (The decay of the metastable state through some *assumed* low probability may look somewhat *ad hoc*. There are cellular automata like Q2R which by themselves give very long relaxation times; see above. Unfortunately in none of the deterministic AIDS known to me were such long relaxation times found, not even on a lattice.)

Cancer cells evade the immune system, perhaps by omitting or changing the cell surface antigens which are recognized by the killer cells K of the immune system. Thus a tumour cell T is assumed [44] to be in one of three states: with the normal surface antigen (T1) recognized by the killer cells, with a changed surface antigen (T2), or without any such surface property (T0). The model repeats for the antibodies, suppressors, helpers, and B cells the equations of the KUT model above, and adds for the killer and the tumour cells:

$$\mathtt{K} = \mathtt{T1} * \mathtt{H} * \overline{\mathtt{S}} \qquad \mathtt{T0} = \mathtt{T0} * \overline{\mathtt{T1} + \mathtt{T2}}$$
$$\mathtt{T1} = \mathtt{T1} * \overline{\mathtt{A} + \mathtt{H} + \mathtt{B} + \mathtt{K} + \mathtt{T0} + \mathtt{T2}} \qquad \mathtt{T2} = (\mathtt{T2} + \mathtt{A} + \mathtt{H} + \mathtt{B}) * \overline{\mathtt{T0} + \mathtt{T1}}.$$

A simple program going through all 256 initial states of this eight-cell model is given in figure 5; the reader can easily change the interactions or even reduce the number of different cell types, now eight. This simulation gives eleven fixed points of which two have no tumour cells whereas in the nine other fixed points the tumour cells survive and thus proliferate.

In a similar spirit, a unified model [45] tries to combine normal immune response, autoimmune diseases, and AIDS. We start again from the usual KUT model with A, S, H, B and V, where V now corresponds e.g. to pneumonia. To this model are added killer cells K attacking the 'own' body O, and AIDS virus D. The dynamics is defined through

$$\mathtt{A} = \mathtt{V} * \mathtt{B} * \mathtt{H} \qquad \mathtt{S} = \mathtt{H} + \mathtt{S} \qquad \mathtt{H} = [(\mathtt{V} + \mathtt{O}) * \overline{\mathtt{S}} + \mathtt{H}] * \overline{\mathtt{D}}$$
$$\mathtt{B} = \mathtt{H} * (\mathtt{V} + \mathtt{B}) \qquad \mathtt{V} = \mathtt{V} * \overline{\mathtt{A}} \qquad \mathtt{K} = \mathtt{O} * \mathtt{H} * \overline{\mathtt{S}}.$$

Without AIDS and autoimmune problems, when D and O are absent (false), we recover the standard KUT model; presence of O may cause attacks from the killer cells, and the AIDS virus D immeditely destroys the helper cells.

All these models are mean-field approximations in the sense that one concentration for one cell type is valid for the whole body. No geometry was involved. The other extreme is lattice models [7,40–5] with nearest-neighbour interactions, where at each lattice site we repeat the above model, and let neighbouring lattice sites interact with logical OR. Presumably the mean-field approach is good if the immune response is much slower than the spread of a cell type through the system, whereas in the opposite limit the lattice model is better. The truth may be in between, though closer to mean field.

For such lattice models, one Cray-YMP processor typically updates slightly more than one cell per nanosecond, and lattices with more than hundred million sites were studied, using one-bit-per-cell methods similar to the other cellular automata. We just have to replace * by .AND. and + by .OR. when we translate the above immunity equations into FORTRAN. The program for cancer then looks similar to figure 5. These

```
C     8-CELL GENERAL IMMUNOLOGY PROGRAM
C     CELL TYPES: 1=ANTIBODY,2=SUPPRESSOR,3=HELPER,4=B-CELL,
C     5=K=KILLER, 6=T0, 7=T1, 8=T2  (3 TYPES OF T=TUMOR)
      PARAMETER (NC=8,MAX=80)
      LOGICAL N(NC),M(NC),IA,IS,IH,IB,IK,IT0,IT1,IT2,IDUM,FP
      DO 1 INDEX=0,255
        DO 6 K=1,NC
 6        M(K)=RSHIFT(INDEX,K-1).AND.1
        DO 8 IT=1,MAX
          IA =M(1)
          IS =M(2)
          IH =M(3)
          IB =M(4)
          IK =M(5)
          IT0=M(6)
          IT1=M(7)
          IT2=M(8)
C HERE SPECIFIC MODEL OF NC DIFFERENT CELL TYPES
          N(1) = IT1.AND.IB.AND.IH
          N(2) = IH.OR.IS
          N(3) = (IT1.AND..NOT.IS).OR.IH
          N(4) = (IT1.OR.IB).AND.IH
          N(5) = (IT1.AND.IH.AND..NOT.IS)
          N(6) = IT0.AND..NOT.(IT1.OR.IT2)
            IDUM = IH.OR.IB.OR.IA
          N(7) = IT1.AND..NOT.(IDUM.OR.IT0.OR.IT2.OR.IK)
          N(8) = (IT2.OR.IDUM).AND..NOT.(IT0.OR.IT1)
          FP=.TRUE.
          DO 2 K=1,NC
 2          FP=FP.AND.(N(K).EQV.M(K))
          DO 7 K=1,NC
 7          M(K)=N(K)
 8        IF(FP) GOTO 9
        PRINT 100, INDEX
 9      IFIX=0
        DO 4 K=1,NC
 4        IF(M(K)) IFIX=IFIX+2**(K-1)
 1      IF(IT.LE.MAX) PRINT 100, INDEX,IFIX,IT,M
 100  FORMAT(1X,3I8,8L3)
      END
```

Figure 5. Cancer program for immunology (one site only).

lattice sizes already approach the order of magnitude of a real immune system, and still allow a simulation much faster on a supercomputer than in reality. However, the biological significance of such lattice models is not clear at present.

Of course, the biological significance of the whole cellular automata approach to immunity can also be questioned: a true cell concentration does not jump between 0 and 1 only. But that does not yet mean that this Boolean approximation is wrong. For example, the Earth is certainly not a point mass but nevertheless can be approximated quite well by a point if we discuss Kepler's laws. The hydrodynamic cellular automata are also unrealistic but nevertheless useful. And similar to hydrodynamics, it may take many years before the right modification of the cellular automata approach is found which gives a good model for immunity.

Acknowledgments

We thank H J Herrmann for suggesting this review, which was necessarily biased and incomplete.

References

[1] Kohring G A 1990 *Int. J. Mod. Phys.* C **1** 259

[2] Wolfram S 1983 *Rev. Mod. Phys.* **55** 601; 1986 *Theory and Applications of Cellular Automata* ed S Wolfram (Singapore: World Scientific)

H Gutowitz (ed) 1990 *Cellular Automata: Theory and Experiment* (*Physica* **45D**)

[3] Weisbuch G 1989 *Dynamique des Systemes Complexes* (Paris: Editions de CNRS) (Engl. transl. 1991 *Complex System Dynamics* (New York: Addison Wesley))

[4] Stauffer D 1991 *Computers in Phys.* Jan/Feb issue

[5] Toffoli T and Margolus N 1987 *Cellular Automata Machines* (Cambridge, MA: MIT Press)

[6] Friedberg R and Cameron J E 1970 *J. Chem. Phys.* **52** 6049
Hardy J, de Pazzis O and Pomeau Y 1976 *Phys. Rev.* A **13** 1949
Jacobs L and Rebbi C 1981 *J. Comp. Phys.* 41 203
Herrmann H J 1986 *J. Stat. Phys.* **45** 145

[7] Dayan I, Stauffer D and Havlin S 1988 *J. Phys. A: Math. Gen.* **21** 2473

[8] Manna S S, Stauffer D and Heermann D W 1989 *Physica* **162A** 20
Adler J 1991 *Physica* A

[9] da Silva L R and Herrmann H J 1988 *J. Stat. Phys.* **52** 463

[10] Stauffer D 1990 *Computer Simulation Studies in Condensed Matter Physics II* ed D P Landau, K K Mon and H B Schüttler (Berlin: Springer) p 26

[11] de Oliveira P M C 1990 *Statistical Physics and Boolean Networks* (Singapore: World Scientific)

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

[13] Rapaport D C 1987 *Phys. Rev.* A **36** 3288; 1988 *Comp. Phys. Rep.* **9** 1

[14] Hayot F, Mandal M and Suddayappan P 1989 *J. Comp. Phys.* **80** 277
Brosa U and Stauffer D 1989 *J. Stat. Phys.* **57** 399
Gunstensen A 1989 *MIT report* unpublished
Kohring G A 1991 *J. Stat. Phys.* **63** (1)
Bhogosian B M, Taylor IV W and Rothman D H 1989 *Proc. Supercomputing 88* vol II ed J L Martin and S F Lundstrom (New York: IEEE Computer Soc.) p 34

[15] Gerling R W 1990 *Preprint* Mathematisch Naturwissenschaftlicher Unterricht **43** 451

[16] Zabolitzky J L and Herrmann H J 1988 *J. Comp. Phys.* **76** 426
Duke D and Sandee D 1988 *Supertimes* (Newsletter of the Supercomputer Computations Research Institute, Florida State University)

[17] Gerling R W 1990 *Physica* **162A** 196
Burda Z, Jurkiewicz J and Flyvbjerg H 1990 *J. Phys. A: Math. Gen.* **23** 3073
Stauffer D 1990 *J. Phys. A: Math. Gen.* **23** 5933

[18] Derrida B and Pomeau Y 1986 *Europhys. Lett.* **1** 59
Costa U M S 1987 *J. Phys. A: Math. Gen.* **20** L583
Stanley H E, Stauffer D, Kertész J, and Herrmann H J 1987 *Phys. Rev. Lett.* **59** 2326
Derrida B and Weisbuch G 1987 *Europhys. Lett.* **4** 657

[19] Lang W and Stauffer D 1987 *J. Phys. A: Math. Gen.* **20** 5413
Moukarzel C and Parga N 1989 *J. Phys. A: Math. Gen.* **22** 943

[20] Schulte M, Stiefelhagen W and Demme E S 1987 *J. Phys. A: Math. Gen.* **20** L1023

[21] Gallas J A C and Herrmann H J 1990 *Int. J. Mod. Phys.* C **1** 181

[22] Glotzer S C, Stauffer D and Sastry S 1990 *Physica* **164A** 1
Stauffer D 1990 *J. Phys. A: Math. Gen.* **23** 1847

[23] Moukarzel C 1989 *J. Phys. A: Math. Gen.* **22** 4493

[24] Stauffer D 1991 *Fractal and Disordered Systems* ed A Bunde and S Havlin (Berlin: Springer)

[25] Kadanoff L P, McNamara G R and Zanetti G 1989 *Phys. Rev.* A **40** 4527

[26] Balasubramian K, Hayot F and Saam W F 1987 *Phys. Rev.* A **36** 2248
Rothman D H 1988 *Geophys.* **53** 509
Chen S, Diemer K, Doolen G D, Eggert K, Fu C, Gutman S and Travis B 1990 *Preprint*
Brosa U 1990 *J. Physique* **51** 1051
Sahimi M and Stauffer D 1991 *Chem. Eng. Sci.* in press
Kohring G A 1991 *Preprint*

[27] d'Humieres D, Lallemand P and Frisch U 1986 *Europhys. Lett.* **2** 291

[28] Rothman D H and Keller J M 1988 *J. Stat. Phys.* **58** 375

[29] Doolen G (ed) 1990 *Physica* **46D** (*Proc. NATO Advanced Research Workshop on Lattice Gas Methods for PDE's*)

[30] Kauffman S A 1969 *J. Theor. Biol.* **22** 437

[31] Keller U, Thomas B and Pohley H J 1988 *J. Stat. Phys.* **52** 1129

[32] Derrida B and Weisbuch G 1986 *J. Physique* **47** 1297

[33] Stauffer D 1987 *Phil. Mag.* B **56** 901

de Arcangelis L 1987 *J. Phys. A: Math. Gen.* **20** L369
Hansen A 1988 *J. Phys. A: Math. Gen.* **21** 2481
Hansen A and Roux S 1989 *Physica* **160A** 275
[34] Stauffer D 1989 *Physica* **38D** 341
[35] Kürten K E 1988 *J. Phys. A: Math. Gen.* **21** L615; 1990 *Dynamics and Memory in Random and Structured Neural Networks* (invited lecture at British Neural Network Society, April 1990)
[36] Goldberg R J 1952 *J. Am. Chem. Soc.* **74** 571
Perelson A S 1989 *Immunological Rev.* **110** 5
[37] Perelson A S (ed) 1988 *Theoretical Immunology* Parts I and II (New York: Addion Wesley)
[38] Kaufman M, Urbain J and Thomas R 1985 *J. Theor. Biol.* **114** 527
[39] Weisbuch G and Atlan H 1988 *J. Phys. A: Math. Gen.* **21** L189
Kürten K E 1988 *J. Stat. Phys.* **52** 489
[40] Chowdhury D and Stauffer D 1990 *J. Stat. Phys.* **59** 1019
Neumann A U 1989 *Physica* **162A** 1
[41] Cohen I R and Atlan H 1989 *J. Autoimmunity* **2** 613
[42] Pandey R B 1989 *J. Stat. Phys.* **54** 997; 1990 *J. Stat. Phys.* **61** 231; 1990 *J. Phys. A: Math. Gen.* **23** 3421
Kougias C F and Schulte J 1990 *J. Stat. Phys.* **60** 263
Sieburg H B, McCutchan J A, Clay O K, Cabalerro L and Ostlund J J 1990 *Physica* **45D** 208
[43] Pandey R B and Stauffer D 1990 *J. Stat. Phys.* **61** 235
Pandey R B 1991 *Preprint*
[44] Chowdhury D, Sahimi M and Stauffer D 1990 *J. Theor. Biol.* to be published
[45] Chowdhury D, Stauffer D and Choudary 1990 *J. Theor. Biol.* **145** 207