Journal of Statistical Physics, Vol. 60, Nos. 5/6, 1990

Flow through a Porous Membrane Simulated by Cellular Automata and by Finite Elements

U. Brosa,[1] C. Küttner,[2] and U. Werner[2]

Received March 22, 1990

Computational results concerning incompressible viscous flow through two channels connected by a porous membrane are presented. The example is extraordinary for its four different types of boundary conditions that are necessary to make the problem complete. The solution is accomplished by two methods: by cellular automata and by finite elements. The numerical means to satisfy the boundary conditions are given for both methods. Overall agreement is achieved, but significant differences show up in details.

KEY WORDS: Hydrodynamics; cellular automata; finite elements.

1. POSING THE PROBLEM

Viscous flow through a permeable membrane is important in many technical applications. Yet it does not seem to have been thoroughly studied. We approach the problem via simulations of a certain boundary-value problem.

The ground on which the play takes place is staked out in Fig. 1. Let us begin with its lower part. We have two channels divided by a wall which is impenetrable except at its central part, where it is porous. The velocity field **V** thus is subject to *nonslip* boundary conditions

$$\mathbf{V}(\mathbf{r}) = 0 \tag{1}$$

on the solid wall (solid line in Fig. 1) and to

$$V^{\parallel}(\mathbf{r}) = 0 \tag{2}$$

on the membrane (dotted line).

[1] HLRZ c/o KFA, Postfach 1913, D-517 Jülich, Federal Republic of Germany.

[2] Universität Dortmund, FB Chemietechnik, LS Mechanische Verfahrenstechnik, Postfach 500 500, D-46 Dortmund 50, Federal Republic of Germany.

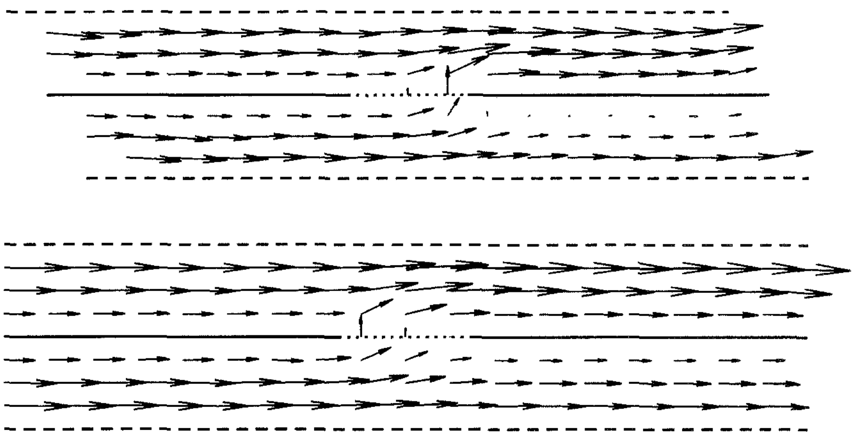
Fig. 1. Flow through a porous membrane computed with finite elements (lower part) and cellular automata (upper part) at Re = 500 and $\Delta v_{\max} = 0.5$. The aspect ratio is $a = 4.5$ and the size of the porous piece $p = 0.75$. The finite-element calculation took place on a rectangular grid of 112 elements with 410 nodes. The triangular grid for the cellular automaton had 3094 × 794 nodes; $v_{\max}$ [cf. Eq. (4)] was 0.2 in lattice units.

Here is an explanation of the somewhat unusual boundary condition (2): The membrane stops the flow parallel to its surface, so that only a perpendicular current may occur. This is based on the assumption that the porous membrane is infinitely thin in an ideal sense. Thus, no flow parallel to the membrane can occur within it. The flow in the zone very close to the membrane must be perpendicular to it.

It is supposed that the two channels are shaped symmetrically, so that the dashed centerlines in Fig. 1 represent the symmetry axes of the respective channels. That way it is implied that the flow is symmetrical, too, an assumption which is justified at least as long as the flow is laminar. We assume that along the centerlines *slip* boundary conditions apply

$$V^{\perp}(\mathbf{r}) = 0 \tag{3}$$

(dashed lines in Fig. 1), reducing the computational labor by a factor of two.

The boundary conditions valid at the open ends left and right in Fig. 1 have to be defined. Let us assume a Cartesian coordinate system (x, y) so that its origin lies in the leftmost point of the central wall and that its x axis is parallel to the walls. The layout as depicted in Fig. 1 extends laterally from $y = -l/2$ to $y = l/2$. Its length is given by $a \cdot l$ (a indicating *aspect ratio*). We stipulate

$$\mathbf{V}(x=0, y) = \begin{cases} \mathbf{e}_x v_{\max} 4y(l-y)/l^2 & \text{for} \quad 0 < y < l/2 \\ \mathbf{e}_x v_{\max} 4y(-y-l)/l^2 & \text{for} \quad -l/2 < y < 0 \end{cases} \tag{4}$$

for the left-hand-side open end—the upstream inlet—and

$$\mathbf{V}(x=al,\ y)=\begin{cases}\mathbf{e}_x(v_{\max}+\Delta v_{\max})\,4y(l-y)/l^2 & \text{for} \quad 0<y<l/2\\ \mathbf{e}_x(v_{\max}-\Delta v_{\max})\,4y(-y-l)/l^2 & \text{for} \quad -l/2<y<0\end{cases} \tag{5}$$

for the right-hand side—the downstream end. As $\mathbf{e}_x$ is to denote the unit vector along the x axis, we have Poiseuille profiles everywhere. This boundary condition is justified by the fact that the inlet and outlet cross sections are far away from the porous part of the channels, so that the flow at inlet and outlet is not influenced by the events near the permeable wall.

The parameters $v_{\max}$ and $\Delta v_{\max}$ are decisive for everything that is to come. $v_{\max}$ determines the Reynolds number

$$\mathrm{Re}=v_{\max}\cdot 2l/\nu \tag{6}$$

with ν as kinematic viscosity. By $\Delta v_{\max}/v_{\max}$ we measure the percentage of liquid passing through the porous membrane.

The dynamical foundation is, of course, the Navier-Stokes equation

$$\partial_t\mathbf{V}=-(\mathbf{V}\nabla)\mathbf{V}-\nabla P/\rho-\nu\nabla\times\nabla\times\mathbf{V} \tag{7}$$

constrained by

$$\nabla\mathbf{V}=0 \tag{8}$$

for Newtonian incompressible fluids with density ρ and pressure P in two dimensions. Since we aim at stationary flows, the time derivative of the velocity field $\partial_t\mathbf{V}$ could be left out. However, the finite-element method presented here uses the time-dependent Navier–Stokes equation, and in the method of cellular automata, a time-dependent technique is applied, too.

For comparisons we use dimensionless units, based on l as unit of length and $v_{\max}$ as unit of velocity. Then the geometry of the device shown in Fig. 1 is fixed by the aspect ratio a and the length p of the porous piece, and the dynamical quantities are characterized by the Reynolds number Re and the throughput $\Delta v_{\max}$.

The lower part of Fig. 1 shows the geometry employed in our calculations with finite elements. The upper part displays the layout for the cellular automata. Here the centerlines are shifted somewhat backward and forward, so that the boundary conditions (4) and (5) apply to skew open ends. This change is motivated by the triangular lattice on which the cellular automata operate. The modification simplifies programming, but does not, we believe, alter the relevant results.

2. SOLUTION BY CELLULAR AUTOMATA

That one can obtain solutions of the Navier–Stokes equation by cellular automata if one takes a triangular lattice has been clear since the publication of ref. 1. During the past few years several papers have appeared which contain worked examples and comparisons with analytically available solutions. We have profitted most from refs. 2 and 3. Various levels of sophistication exist that can increase the effectiveness by a factor of three (see, e.g., ref. 4). However, as will become clear in Section 5, it is of prime importance that the implementation of the cellular automata be as elementary as possible. Hence, all further developments will be based on the extremely simplified algorithms presented in ref. 5. In that paper, only the implementation of the interior dynamics was detailed. Therefore, we must show now how the various boundary conditions introduced in Section 1 can be realized.

The gist of the implementation as utilized in ref. 5 is exhibited in Fig. 2: In every node of the triangular lattice, there may be particles staying (note the 0) or leaving in one of six directions. The presence of such particles is indicated by setting the respectives bits in a byte.

The unit of length is here the next-neighbor distance, and that of time is one complete update of the field.

The update proceeds in two phases, stream and collision. In the colli-

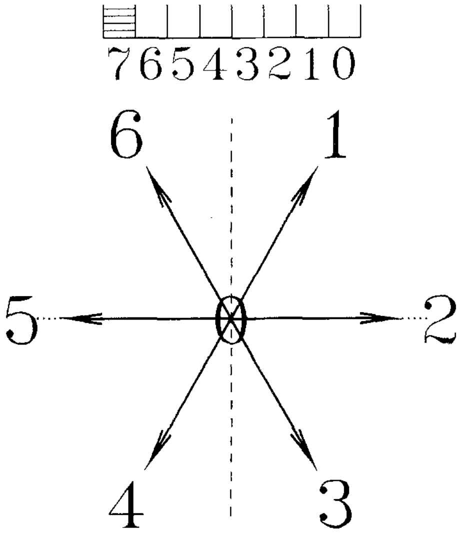

Fig. 2. The diagram of directions for the implementation of cellular automata as used in this work. Related to the seven directions are the seven low-order bits in a byte which is shown in the upper part of the figure. Bit 7 is used to ensure conservation of angular momentum.[5]

sion phase, there is no interaction among the nodes; the bits are reshuffled for every node separately. We do this with a look-up table.[5] Now collision means interaction between mobile particles. A boundary condition, in comparison, can be construed as interaction between one mobile particle and another fixed on the wall. Hence, the only difference between coding collisions and the boundary conditions is that a different look-up table has to be applied for the particles hitting the border. Since in the present example we have four types of boundary conditions, the construction of the various look-up tables can be difficult and may lead to well-concealed errors. Therefore it might be useful to know the following algorithms.

Let us start with boundary condition (2). In the language of cellular automata, it is equivalent to specular reflections at the vertical plane depicted in Fig. 2 by the dashed line. Hence an *arriving* 1 particle becomes a *leaving* 6 particle, an arriving 2 leaves as a 5, and so forth. The general rule is

$$\text{incoming IB moves out as } 7-\text{IB} \tag{9}$$

Hence, take the least significant byte of an integer II (representing the incoming particles) and check if the bit IB is set. If it is set, flag the bit 7 − IB in the integer IO (describing the outgoing particles). The entire algorithm might look like this:

```
      DO 3 II = 0, 255
      IO = IAND(II, 129)
      DO 1 IB = 1, 6                                    (10)
1     IF (BTEST(II, IB)) IO = IOR(IO, ISHFT(1, 7 - IB))
3     IREV(II) = IO
```

We use extended Fortran statements valid on Crays, IBM mainframes, with Microsoft Fortran and others; for, example BTEST(II, IB) checks if in the integer II the bit IB is set. The first line in (10) cranks through all possible input configurations. In the second line the zeroth and let seventh bits are captured from II and written on IO, as they belong to states that cannot move. The third and fourth lines are to account for the moving particles. The use of the bit-adding IOR is important because several particles may coexist in a node. If the reshuffle for one of them was determined in a previous pass through the DO 3 loop, this must not be destroyed in the present pass. The last line of the code establishes the desired look-up table for specular reflections at the vertical.

Specular reflection at the horizontal [see boundary condition (3)] is slightly more complicated to code. From Fig. 2 we find

$$\text{incoming IB moves out as } \begin{cases} 4-\text{IB} & \text{if IB} = 1, 2, 3 \\ 10-\text{IB} & \text{if IB} = 4, 5, 6 \end{cases} \tag{11}$$

The code is almost the same as in (10). Only the interior DO 1 loop has to be split.

For the nonslip boundary condition (1) it takes backward-reflection or inversion rules.[2] They can be described by

$$\text{incoming IB moves out as } \operatorname{mod}(\text{IB}+2, 6)+1 \tag{12}$$

so that in (10) just the $7-\text{IB}$ has to be replaced by the expression with the modulo function.

Maintaining the correct boundary conditions at the open ends requires a different approach. It is closely related to the distribution of suitable initial values. If a velocity field $\mathbf{V}$ is given, it is in general not possible to construct a cellular automaton representing these velocities at every node. Only after averaging over many nodes (typically 400) may we obtain the desired velocities as mean values. Hence, we must trim the nodes in a stochastic way. A convenient formula for the probabilities[6] is

$$p_i=\frac{\rho}{7}\left(1+\frac{1}{3}\mathbf{e}_i\cdot\mathbf{V}\right) \tag{13}$$

which holds if ρ is much smaller than 7. The $\mathbf{e}_i$ are the unit vectors shown in Fig. 2. One verifies readily that the sum over the p_i, $i=0, 1, 2,..., 6$, gives the density ρ, while $\mathbf{V}$ is recovered from the mean over all $\mathbf{e}_i$. With this we can build a suitable initial distribution by a Monte Carlo algorithm: Let the velocity field be represented by the integer field IV(IX, IY), where (IX, IY) signifies the discretized (x, y). Get the deterministic velocity $\mathbf{V}(x, y)$ for the point (IX, IY). Compute the probabilities p_i from Eq. (13). Flag the ith bit in IV(IX, IY) if a uniformly distributed random number is smaller than p_i.

The same procedure may be used for the computation of velocity profiles as the open ends to satisfy the boundary conditions (4) and (5). This is described with more details in ref. 7; see also ref. 8 for the solution of a similar problem.

We finish the task demanded in Section 1 by solving a *noncompatible initial value problem.* That is, we put the boundary conditions (1)–(5) into force and distribute initial values according to (4) all over the channels, except at their downstream ends. Then particles get jammed for an initial period in the lower channel, while they become rarefied in the upper part of the device. Later, the flux through the membrane increases and the jam is released.

This process is shown in Fig. 3. Refer first to the lines distinguished by the asterisks. We observe their divergence for times between 0 and 1600; see the drop of density in the upper channel and its increase in the lower

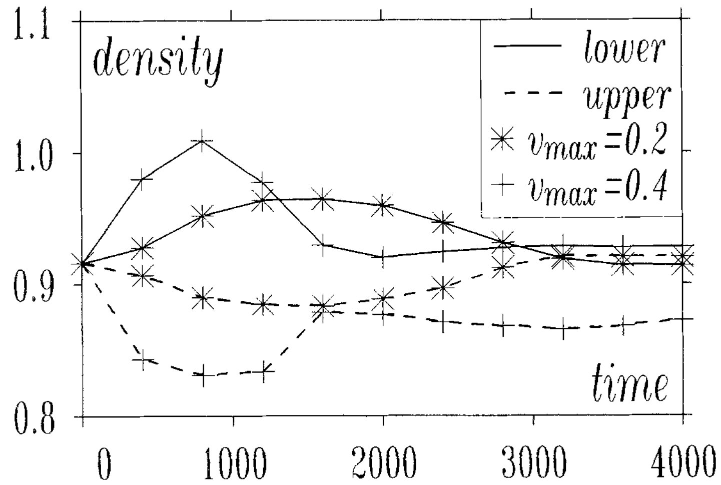

Fig. 3. The jam of particles in the downstream half of the two-channel device and its subsequent release. We use diagrams like this to find the time at which stationarity has been reached.

one. Then the lines converge and reunite at time 3200. So at least for velocities around 0.2 we find reasonable convergence and can thus solve the stationary problem.

The behavior is different for a bigger $v_{\max}$. Again we notice the initial divergence (see the lines marked by the crosses in Fig. 3), but in this case convergence is never attained, even at times much greater than those shown. The reason is clear: The Mach number of the cellular automata is about one. Yet even for velocities as small as 0.4, compressibility effects show up. The problem with this kind of compressibility, however, is that it has nothing in common with the compressibility of a real liquid.[9] Therefore, we must keep all velocities well below 0.4.

But also too small velocities are detrimental,[9] as one can see in Fig. 4. In its upper part, it displays results from a calculation with $v_{\max} = 0.05$: The deterministic motion is drowned in fluctuations. For stationary flow one can somewhat improve on that by averaging over flow fields at different instances. This is what was done to obtain the field shown in the lower part of Fig. 4.

Hence, for calculations with cellular automata, changes in Reynolds number are practically equivalent with changes of lattice size. We can quantify this with Eq. (6) when we substitute, instead of l, $\sqrt{3}/2 \cdot N_y$, N_y being the size of the grid in the lateral direction. With various collision rules one can change the viscosity ν from 0.5 to 1.0 (in the present case ν is 0.55[5]). $v_{\max}$ can vary, as was discussed above, from 0.1 to 0.3. This together gives less freedom than one order of magnitude. But also the range

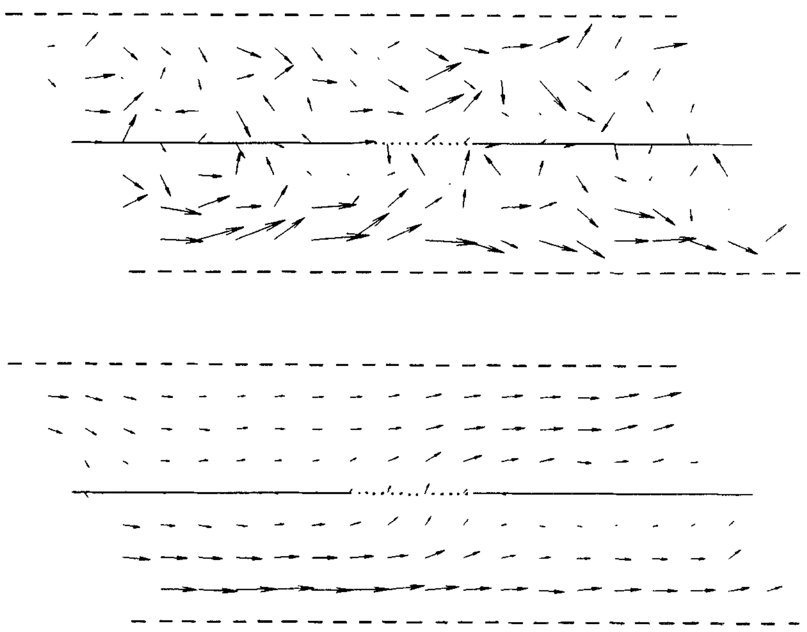

Fig. 4. Flow fields according to cellular automata at Re = 10. The upper picture was taken from a single instance *time* = 1000, while the lower one emerged from averaging over flow fields at *time* = 100, 120,..., 2000. Here $a = 2.7$ and $p = 0.48$.

of lattice sizes is limited. N_y must not be much less than 100, as otherwise the statistics becomes poor. On the other hand, N_y cannot be much bigger than 1000 due to finite computer memory However, for the present problem computations with Reynolds numbers between 10 and 1000 were performed, which should have a statistical accuracy better than 10%.

3. SOLUTION BY FINITE ELEMENTS

The method of finite elements has been proved over the years to be an adequate method for solving fluid flow problems.[10] Therefore we apply it to our problem of flow in a channel with porous walls.

Since the problem is two-dimensional, the stream function–vorticity formulation of the Navier–Stokes equation is used, thus avoiding the explicit calculation of pressure. This leads to the following two equations:

$$\nabla^2\psi + \omega = 0 \tag{14}$$

$$\frac{\partial\omega}{\partial t} + \frac{\partial\psi}{\partial y}\cdot\frac{\partial\omega}{\partial x} - \frac{\partial\psi}{\partial x}\cdot\frac{\partial\omega}{\partial y} - \nu\nabla^2\omega = 0 \tag{15}$$

with ψ as the stream function defined by

$$v_x = \frac{\partial \psi}{\partial y} \quad \text{and} \quad v_y = -\frac{\partial \psi}{\partial x} \tag{16}$$

and ω as the vorticity. The two equations are solved simultaneously, and boundary conditions are required for both.

The method used for solving these equations is based on a computer program developed by Reiners.[11]

As one can see, the time-dependent version of the formulation is required. This facilitates the implementation of boundary conditions at the permeable wall, which in part are calculated with the help of a time iteration.

The equations are solved on a finite-element grid that is formed by two axisymmetric parts, each one consisting of 56 elements. The two parts touch at the permeable wall. The elements are of rectangular shape, but of isoparametric type. The interpolation functions for both the stream function and the vorticity are quadric, i.e., of second order with two variables.

The boundary conditions are as follows, first for the stream function.

The fluid velocities at inlet and outlet cross sections [boundary conditions (4) and (5)] are prescribed by the parabolic profile of the laminar flow in a channel. The maximal velocities must be set according to the mass balance, which is based on the two inflowing streams (characterized by $v_{\max}$) and the portion of flow that passes through the permeable wall ($\Delta v_{\max}$). A cubic profile of the stream function is calculated from those velocities. The stream function on the symmetry axes [boundary condition (3)] and on the solid walls [boundary condition (1)] is set to constant values.

The vorticity on the symmetry axes is set to zero and changes as a linear function along the inlet and outlet boundaries. On the permeable and solid walls the vorticity boundary conditions are calculated from a Taylor series of the stream function,[12] supposing that the nonslip condition is valid on both types of wall. At the permeable wall, a velocity perpendicular to the wall does exist.

Initial values for the vorticity—resulting from the linear distribution with respect to the y axis of the undisturbed channel flow—are given, and with them the stream function is calculated from Eq. (14). Using the stream function, we obtain the values for the time derivative of the vorticity using Eq. (15). The vorticity itself is calculated by multiplying the derivative with a discrete time step—a point in connection with this will be noted below. With the new vorticity we get the new stream function. The time iteration goes on until two vorticities differ by less than a certain limit. This is considered to be the steady state.

Since the whole finite-element domain consists of two parts, a coupling condition for the stream function at the permeable wall is employed at each time step: the value of the stream function at the permeable wall is averaged from the values of the upper and the lower domains.

The program increases the flow rate through the permeable wall in steps of 1 % of the inflow, at each step reaching the steady state. Difficulties with the stability of the algorithm occur when the calculation goes on in steps that are too large of flow rate through the permeable wall.

There are two other phenomena which can lead to instabilities. The first concerns the size of the grid which determines the size of the matrices that are used in the course of the calculations. If the matrices are too big, they cannot be inverted with the Gauss algorithm (as is done now), but show up as singular. This is certainly a numerical problem, for the sort of matrices does not change essentially when the grid becomes bigger. Therefore, the finite-element method in our case is limited to rather small grids at present.

A second problem is the size of the time steps. The higher the Reynolds number of the flow, the smaller the time steps must be. Nevertheless, it can happen that suddenly, at a certain flow rate through the wall, the calculations stop due to numerical overflow: the algorithm becomes unstable. This problem might be overcome by averaging the vorticity values of the actual and of the preceding time step. The time steps can then be chosen much longer, but this procedure has not yet been performed in the calculations presented in this paper.

4. COMPARISON OF RESULTS

When we compare finite-element results with those from cellular automata, our main result is a negative one. Clearly, if $\Delta v_{\max} > 0$, some liquid must flow through the membrane. This induces certain global features which are reproduced by both computational methods (see Fig. 1). However, details of the distribution of flux over the membrane are important. Here our two methods give, for Reynolds numbers well above 100, opposite results. According to cellular automata, the main flux passes the membrane close to its downstream edge. The finite elements, in contrast, suggest that the membrane is strained mostly in its upstream part (see Fig. 1).

The reason for the discrepancy is not clear. Both codes were carefully tested and applied to nontrivial examples (see, e.g., ref. 7). Differences in the implementation of boundary conditions based on the two very different methods may be responsible. In addition, it is possible that the flow at higher Reynolds numbers ($\mathrm{Re} > 500$) is no longer laminar, setting limits to the finite-element method, which assumes laminar flow. At Reynolds number bigger than 500, the mentioned instabilities also occur due to the size of the time steps. This might be another hint on the flow changing its characteristics in this region of Reynolds numbers.

We are not aware of experimental data to decide the conflict.

The discrepancy is sizable only for Reynolds numbers well above 100. In the opposite case we find by both methods a more uniform flow through the porous material, as shown in Fig. 4. The differences are then negligible as compared with the inherent inaccuracies of our two methods.

5. THE TABLE OF USEFULNESS

Another kind of result derives from our immediate experience with different computational schemes for hydrodynamics, viz. cellular automata,(5,7) finite elements, and spectral methods,(13,14) from which we derive a "table of usefulness." We hope that this table might serve as a first orientation for those wishing to enter the field.

We base our comparison on four criteria: *stability*, *flexibility*, *efficiency*, and *simplicity*.

When a numerical instability, i.e., amplification of rounding errors, occurs, a supercomputer is not more worth than a pocket calculator. Hence *stability* is prime.

A property very relevant for practical problems is *flexibility*. By this we mean the adaptability to different boundary conditions. For example, a code which permits the simulation of a turbulent boundary layer over a flat plate is fine, but a code by which one can do the same thing with a realistic airfoil is better.

Next comes *efficiency*. It is defined as the ratio of accuracy over computational time. Hence, for many problems with partial differential equations, higher efficiency just allows for more accurate solutions. In hydrodynamics, changes in efficiency can cause qualititive changes: With a more efficient code we can do computations with better resolution. On the other hand, fluids at higher velocities develop structures of increasing complexity, especially in turbulence. For complex structures we need high resolution. Therefore, when we have only an inefficient code, we cannot treat turbulence.

The fourth criterion is *simplicity*. According to common experience, the frequency of errors increases dramatically with the complexity of the code. A wrong program, however, has little value even if it outperforms everything else by speed.

Except for stability, the importance of the criteria depends on the peculiarities of the problem at hand (see Table I). Each of the numerical

Table I. Strengths and Weaknesses of Three Numerical Methods in Hydrodynamics

Algorithm	Stability	Flexibility	Efficiency	Simplicity
Cellular automata	+	+	– –	+ +
Finite elements	–	+	–	+
Spectral methods	–	–	+	–

methods is good (+) with respect to some aspects, but bad (–) or even dreadful (– –) with respect to others. "Dreadful" means that there exists at present no remedy to cure the evil.

Due to their simple implementation and excellent stability, cellular automata seem to be valuable to test other more effective but also more involved and critical numerical methods. On the other hand, the efficiency of cellular automata is so low that there seems to be no chance to use them for the simulation of high-Reynolds-number flow, especially for 3D turbulence. (Ref. 15 does not contain a counterexample, as just the slackening of a 2D velocity field was described.) Likewise, due to the limitations discussed in Section 2, cellular automata are not useful for flow at very low Reynolds numbers.

Finite elements are already more difficult to code, but their main advantage is a better efficiency combined with excellent flexibility. For example, to obtain the results shown in Fig. 1 took on a Cray YMP 1 hr per processor when cellular automata were used, while with finite elements less then 5 min had to be spent. The finite-element method is somewhat imperilled by instabilities, but we think that these jeopardies can be tamed.

The spectral methods seem to be very much on the negative side. Nevertheless, due to their high efficiency, they furnish the only approach to real turbulence simulations.(16,17,13)

The three numerical methods may be ranked according to *continuity*: The velocity field of cellular automata can be thought as a superposition of δ-functions. The fields represented by finite elements are at least once differentiable, and those of the spectral methods are perfectly smooth. It is obvious that this is the reason for the decrease of flexibility as well as for the increase of efficiency: With the cellular automata we follow the chaotic motion of single particles, while in reality small pieces of a fluid have only few degrees of freedom.

ACKNOWLEDGMENTS

We thank M. Kremer and D. Stauffer for suggestions and support. The work would probably never have been accomplished without J. A. M. S. Duarte's experience concerning initial values and boundary conditions in cellular automata.

REFERENCES

1. U. Frisch, B. Hasslacher, and Y. Pomeau, *Phys. Rev. Lett.* **56**:1505 (1986).
2. D. d'Humieres and P. Lallemand, *Complex Systems* **1**:599 (1987).
3. H. A. Lim, *Phys. Rev. A* **40**:968 (1989); H. A. Lim, *Complex Systems* **2**:45 (1988).
4. F. Hayot, M. Mandal, and P. Sadayappan, *J. Comp. Phys.* **80**:277 (1989).
5. U. Brosa and D. Stauffer, *J. Stat. Phys.* **57**:399 (1989).
6. S. Wolfram, ed., *Theory and Applications of Cellular Automata* (World Scientific, Singapore, 1986).
7. J. A. M. S. Duarte and U. Brosa, *J. Stat. Phys.* **59**:501 (1990).
8. F. Hayot and M. Raj Lakshmi, *Physica D* **40**:415 (1989).
9. J. P. Dahlburg, D. Montgomery, and G. D. Doolen, *Phys. Rev. A* **36**:2471 (1987).
10. T. J. Zhung, *Finite Element Analysis in Fluid Dynamics* (McGraw-Hill, New York, 1978).
11. U. Reiners, Simulation laminarer Strömung in Kanälen mit permeablen Wänden, Dissertation, Universität Dortmund (1989).
12. P. J. Roache, *Computational Fluid Dynamics* (Hermosa, Albuquerque, 1972).
13. L. Boberg and U. Brosa, *Z. Naturforsch.* **43a**:697 (1988).
14. U. Brosa, *J. Stat. Phys.* **55**:1303 (1989).
15. S. Succi, P. Santangelo, and R. Benzi, *Phys. Rev. Lett.* **60**:2738 (1988).
16. J. Kim, P. Moin, and R. Moser, *J. Fluid Mech.* **177**:133 (1987).
17. E. Laurien and L. Kleiser, *J. Fluid Mech.* **199**:403 (1989).

Communicated by D. Stauffer