#2969

AIU-25 #2969

Modeling Chaos

A parallel CPU architecture can take you where shorter clock ticks, smarter instructions, and more on-chip memory can't go

Peter Wayner

WHEN SCIENTISTS ATTEMPT to model the flow of water or air or any fluid, they write a Navier-Stokes differential equation that describes how a small part of a continuous stream will behave. In easy cases, such as water flowing down a straight pipe, calculus and clever guessing can provide an exact solution that describes the flow. But in more difficult problems, such as the turbulent flow of air around the wing of a new airplane (see photo 2 on page 258), the answer cannot be found with paper-and-pencil mathematics; numerical analysis by computer is required.

Solving this type of problem on a traditional, serial-architecture computer—even a very fast one—can be impractical because of the large number of separate and independent calculations to be performed. The problem seems tailor-made for a parallel architecture, and, in fact, it has become a primary application for parallel-architecture machines. The approach has been so successful that many companies are replacing their wind-tunnel tests with computational models running on parallel-architecture computers.

A look at two such computers illustrates many of their strengths and some of the technical issues that come up in using them. In the Connection Machine, from Thinking Machines Inc. (Boston, Massachusetts), several thousand extremely simple CPUs are hooked into a large array with carefully arranged channels of communication between them. The design allows many thousands of similar calculations to be executed literally at once (not just apparently, as is the case with multitasking architectures). Another radical new design, implemented by researchers at Princeton, involves CPUs with only one instruction that is custom-designed to solve a single problem at a very high speed.

Instead of using standard numerical methods requiring accurate real-number arithmetic to simulate a process, both the Connection Machine and the Princeton computer use a cellular automaton to model the interaction between particles on a grid. It may not be elegant to the classical mathematician raised on smooth functions, but its simplicity makes it easy to compute in parallel. Before looking at the details of these machines, I'll briefly explain the cellular automaton model used on both of them to simulate fluid flows. Having the practical application in front of you makes the strengths and drawbacks of parallelism much clearer than would an abstract discussion.

Fitting the Problem to the Architecture

The cellular automaton model discussed here was proposed by a team of three scientists at the Los Alamos National Laboratory: Uriel Frisch, Brosl Hasslacher, and Yves Pomeau. Further studies have been made by others, including Jim Salem, Bruce Nemnich, and Steve Wolfram at Thinking Machines.

The model follows the movement of particles on a large hexagonal lattice. The particles interact according to a set of easily computed rules that specify the outcome for every possible collision. After each time step, the computer checks for particles colliding. The hexagonal lattice lets up to six particles collide at once from six different directions; this yields $2^6 = 64$ possibilities. However, the 64 possibilities can be reduced through reflection and rotation of axes to a more manageable 14, and that is the number of rules the working model actually contains (see figure 1).

All the particles are assumed to be moving at the same speed. Every rule conforms to the Newtonian law of conservation of momentum. Put differently, the vector sum of particles moving inside each grid is the same before and after each time step.

The model can be adjusted to handle fluids and gases with different viscosities by adjusting the density of the grid. The dynamics of a fluid are measured by a number known as the Reynolds number, which is proportional to the particle velocity and size, and inversely proportional to the viscosity of the fluid. Slow-moving objects in thick liquids, like raisins in molasses, have a low Reynolds number; fast objects moving through slippery fluids, like bullets through air, are described by large numbers.

In the simulation, the relative density of the grid determines the Reynolds number. If there are many nodes close together, the automaton behaves like a fluid with a high Reynolds number. If there are relatively fewer nodes, the simulated fluid will be thicker. Experiments have shown that the number of nodes per square inch is roughly proportional to the square of the fluid's Reynolds number.

continued

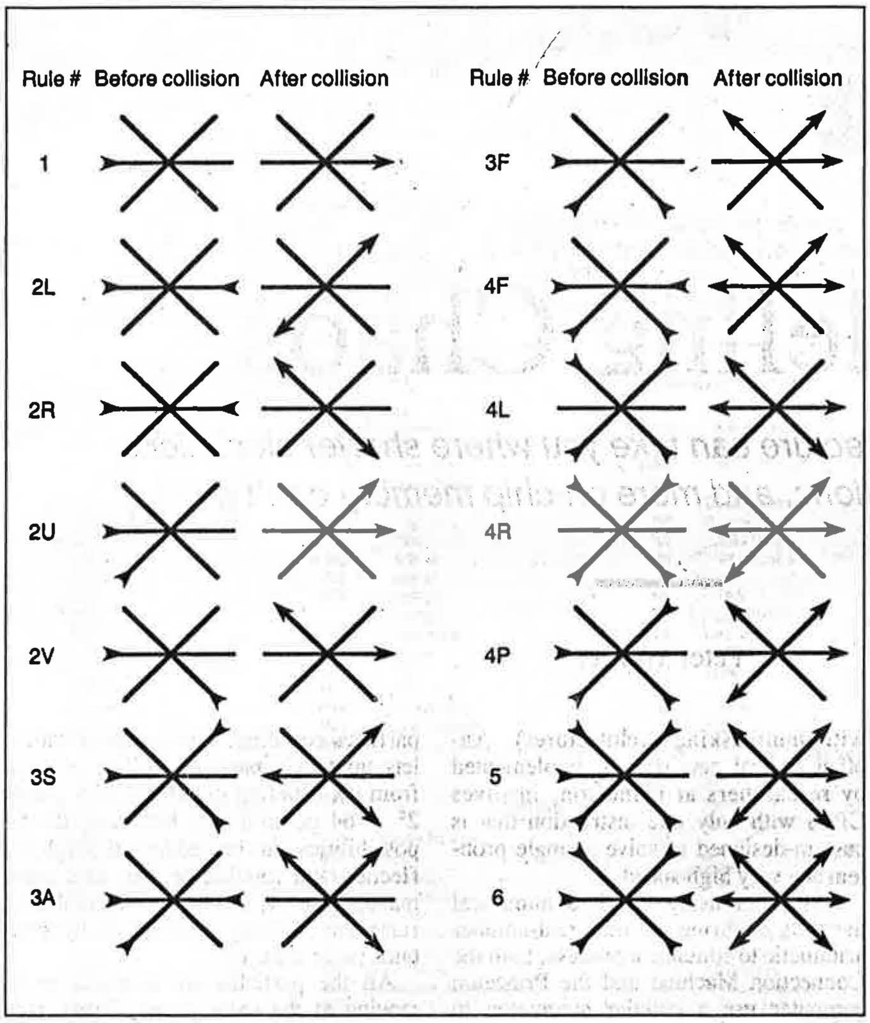


Figure 1: *These 14 rules determine the effects of particles colliding in the fluid-flow cellular automaton. By rotating and reflecting the axes, the rules cover all 64 possibilities.*

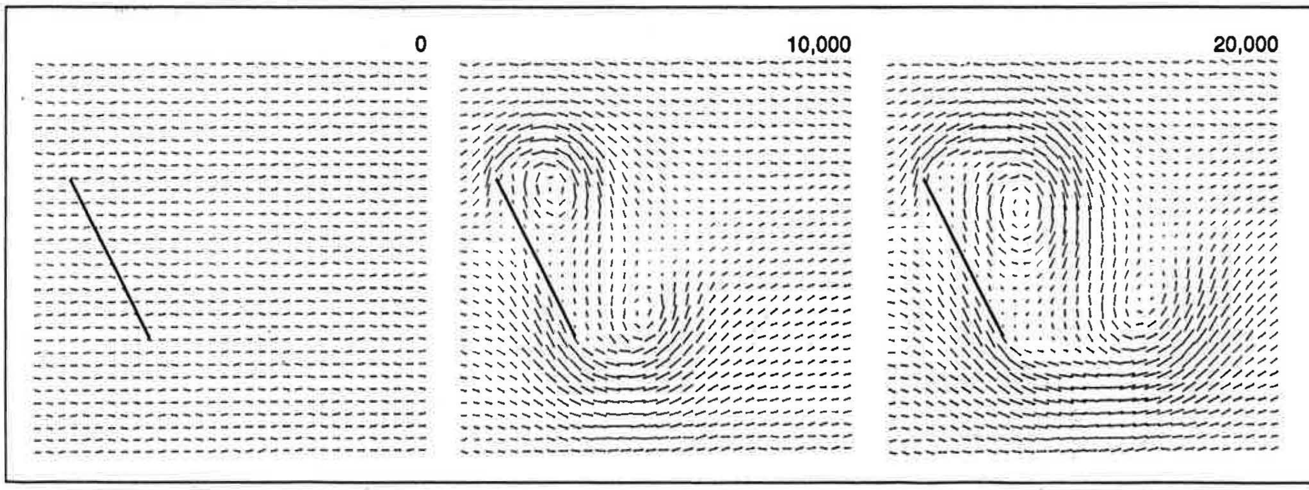
Figure 2: *The successive generations of a 2048- by 2048-cell automaton show how a plate can introduce turbulence into the model. Each arrow represents the average velocity computed over a 64- by 64-cell region. The frames are shown for 10,000-generation intervals, starting at generation 0.*

Using a Macintosh with Lightspeed Pascal, I wrote a program for calculating and displaying a microscopic version of the model (see the editor's note at the end of this article). At this cut-down level, the behavior of particles appears random. A better picture of what happens at the macroscopic level requires that we use a much larger grid, divided into quadrants, and calculate an average direction for the particles in a given quadrant. Figure 2 shows a sequence of "snapshots" made every 10,000 time steps using the Connection Machine. Each of the arrows in figure 2 represents the average direction of the particles in a 64 by 64 group of cells; the entire model consists of 2048 by 2048 cells altogether. At this level, the restrictions of the model begin to disappear and the behavior of the averages looks much more like a fluid. It would clearly be impractical to do work on this scale using a single-processor microcomputer.

A Special CPU for the Cellular Automaton

Parallel computing is the obvious solution to speeding up this problem, but it is not so easy as simply throwing more processors into the box. The chips must communicate with each other, and if the architecture of the machine is not carefully designed, most of the gain in computation power can be lost to communication time. In the fluid-flow automaton, the communications step is even more significant than in most parallel applications because the computation phase is almost trivial: Rules can be implemented by feeding the 6 bits that describe a node into a small set of Boolean gates.

At Princeton University, Professor Kenneth Steiglitz and graduate student Steve Kugelmass have built a specialized computer with custom very-large-scale-integration (VLSI) chips to process the fluid-flow automaton particles. The speed of the machine comes from lining



up many simple machines. In early designs of the chip, the individual processors were placed in a hexagonal lattice with data lines running between them. Each processor computed a single node.

This approach to the problem had an intuitive appeal, but it quickly ran into problems with the communications overhead. While it was quite easy to send the information about the particles between the processors on a chip, it was difficult to arrange the communications between two chips because of the physical limitations of the package. A simple chip with 37 processors arranged in concentric hexagons needed 84 pins just to handle the data coming to and from the neighboring chips. These obstacles could have been overcome using multiplexing and other techniques, but not without sacrificing speed and simplicity.

Steiglitz and Kugelmass scrapped the one-processor-per-site architecture in favor of a pipeline of slightly more sophisticated processors. Each processor has a shift register that holds three lines of the hexagonal grid (see figure 3). In an n by n array, the shift register holds $2n + 1$ cells at once, mapped as shown in figure 3 for a 4 by 4 array. It takes n^2 steps to compute an entire generation, but when x shift registers are lined up, the x generations are done simultaneously in the same n^2 steps. (This doesn't include the time required to fill the pipeline with initial data, of course.) The size of the largest feasible shift register limits the width of simulations that can be done with the pipeline machine, but not the overall length. Long wind-tunnel experiments are particularly easy.

This design removes the intercommunication bottleneck and interconnect difficulties associated with the original two-dimensional model. It also makes it easier to present a picture of the current state of the automaton for display or other

continued

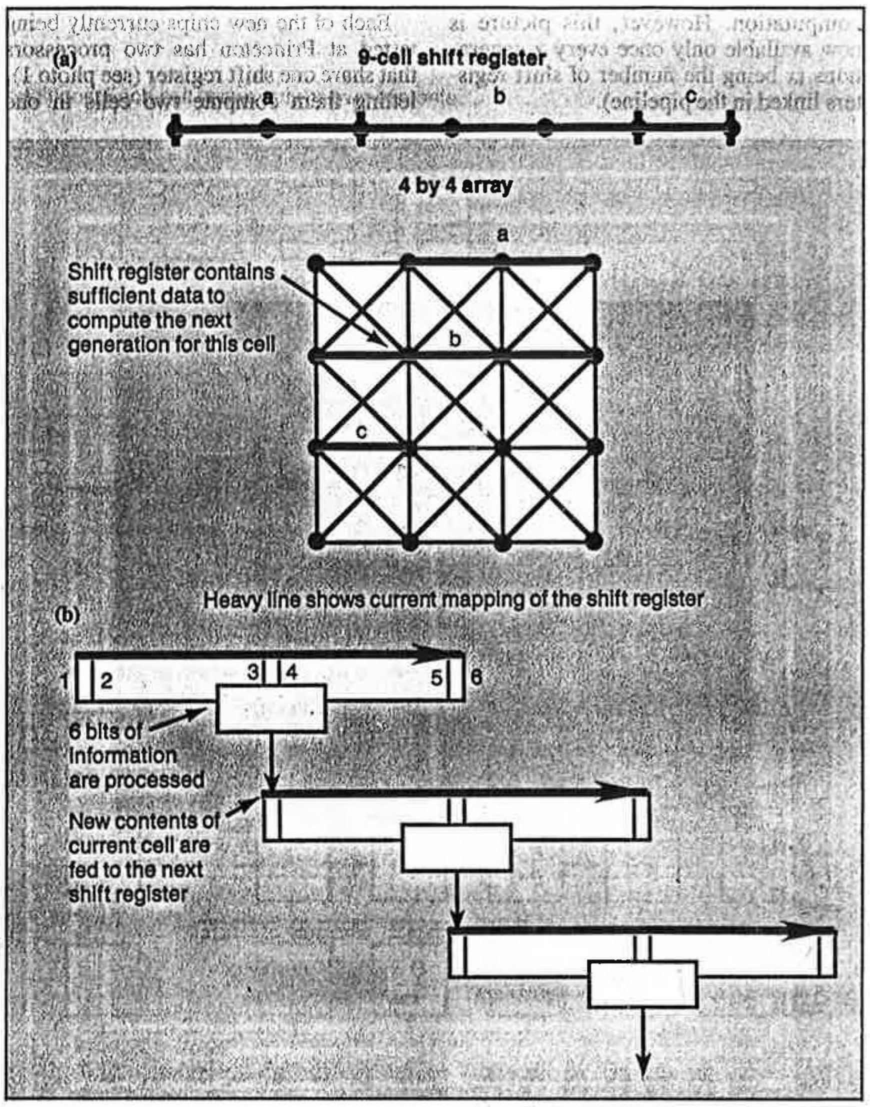
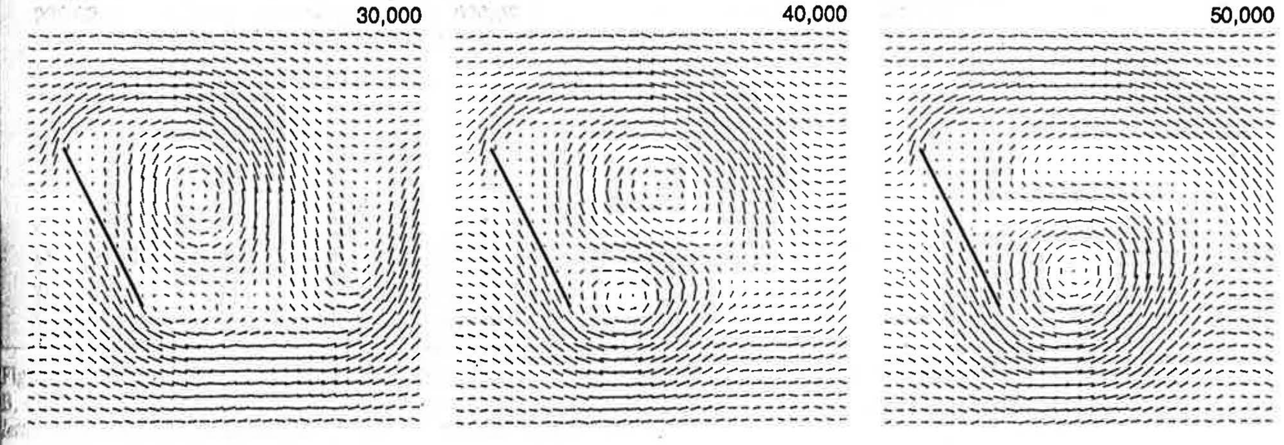

(a)
9-cell shift register
a
b
c
4 by 4 array
a
Shift register contains sufficient data to compute the next generation for this cell
b
c
(b)
Heavy line shows current mapping of the shift register
1
2
3
4
5
6
6 bits of information are processed
New contents of current cell are fed to the next shift register

Figure 3: *The Princeton machine's pipelined architecture.* **(a)** *The two-dimensional array of cells is mapped onto a shift register. The illustration shows the case for a 4 by 4 array and a 9-cell shift register. At any given time step, the register has enough information to compute one cell's next generation. It takes 16 time steps to do the entire 4 by 4 array.* **(b)** *By pipelining multiple shift registers, multiple generations can be computed at once.*



computation. However, this picture is now available only once every *x* generations (*x* being the number of shift registers linked in the pipeline).

Each of the new chips currently being tested at Princeton has two processors that share one shift register (see photo 1), letting them compute two cells in one clock cycle. To accomplish this, the shift register is simply extended in length to 2*n* + 2. Using this design, each chip is capable of doing 20 million cell updates per second. The prototype machine, however, is connected to a Sun-3 workstation with a bus that can process only ⅓ million sites per cycle. Once the pipeline is filled with data, the machine can process ⅔ million sites per second per chip.

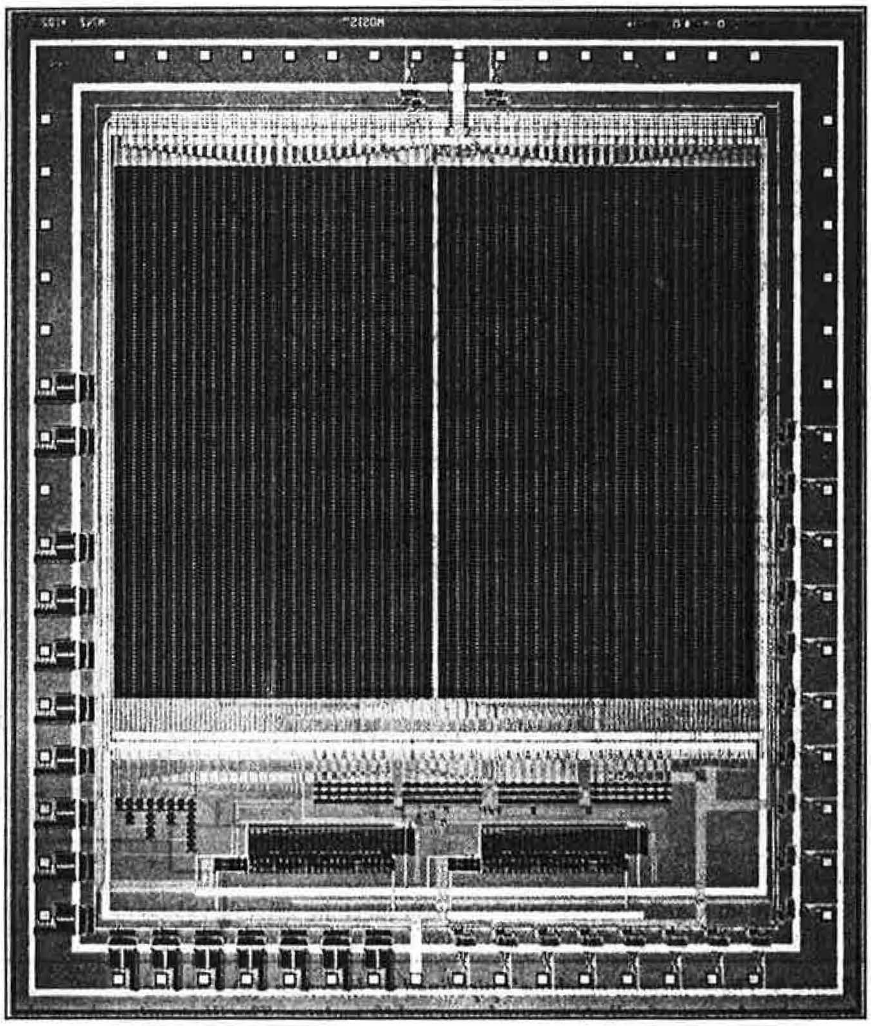


Photo 1: *The Princeton parallel-processing chip. The shift register dominates the top three-quarters of the photograph while the two processors are at the bottom. The shift register contains 512 memory words. The entire chip contains the equivalent of 68,000 transistors implemented in 3-micron technology.*

The Connection Machine

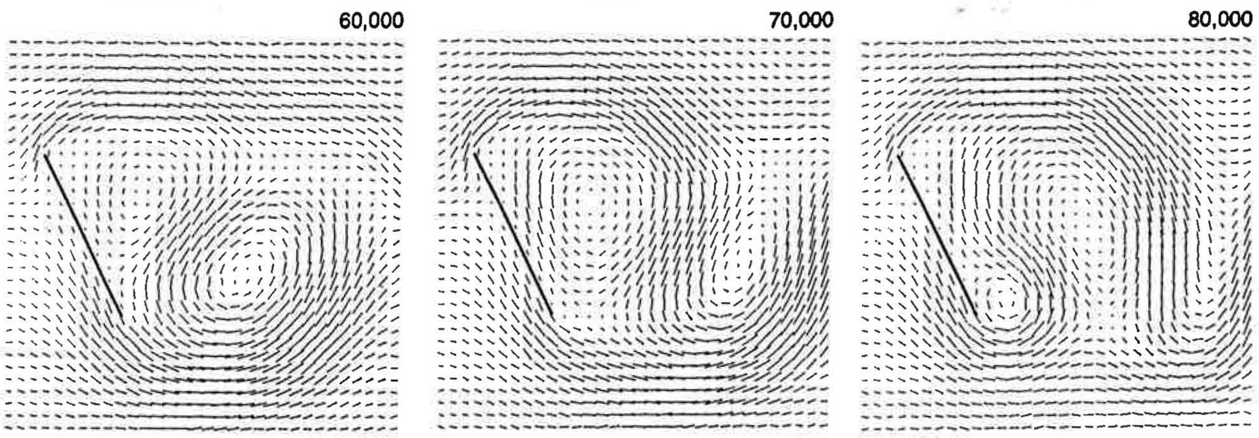
One of the original implementations of the fluid-flow automaton was done on the Connection Machine (CM) by Jim Salem, Bruce Nemnich, and Steve Wolfram. The CM uses 65536 processors in a very flexible architecture. Each processor has links to 12 neighbors, letting it act as a "hypercube" in up to 12 dimensions. A front-end computer compiles the program and loads the code into the parallel processors. Each time step consists of a communications phase and a computation phase.

Each processor is a simple bit-oriented computer with its own 8192 bytes of memory. In the newer CM-2, each processor also has its own floating-point chip attached for very fast scientific computing. Extensions and new data types are provided for Lisp, C, and FORTRAN to make the parallelism transparent or at least accessible to programmers.

Most of the limitations that exist with Steiglitz's initial prototype of one processor per site are removed by the general-purpose architecture of the CM and its special software. The machine was designed to be a parallel-processing computer and can easily be programmed for any purpose, so wires run from each of the processors back to the front-end computer handling the input and output.

For the fluid-flow automaton, the CM was configured as a plane with links between each processor and its four neigh-

continued

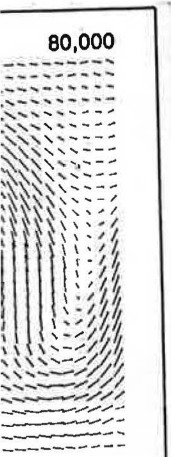


is, the shift
length to 2*n*
chip is capa-
updates per
chine, how-
workstation
only ⅓ mil-
e pipeline is
can process
r chip.

entations of
s done on the
y Jim Salem,
Volfram. The
a very flex-
ocessor has
g it act as a
mensions. A
les the pro-
o the parallel
consists of a
a computa-

e bit-oriented
192 bytes of
-2, each pro-
Floating-point
cientific com-
w data types
d FORTRAN
nsparent or at
mers.
hat exist with
of one proces-
y the general-
e CM and its
chine was de-
ocessing com-
ogrammed for
from each of
front-end com-
d output.
maton, the CM
with links be-
its four neigh-

continued



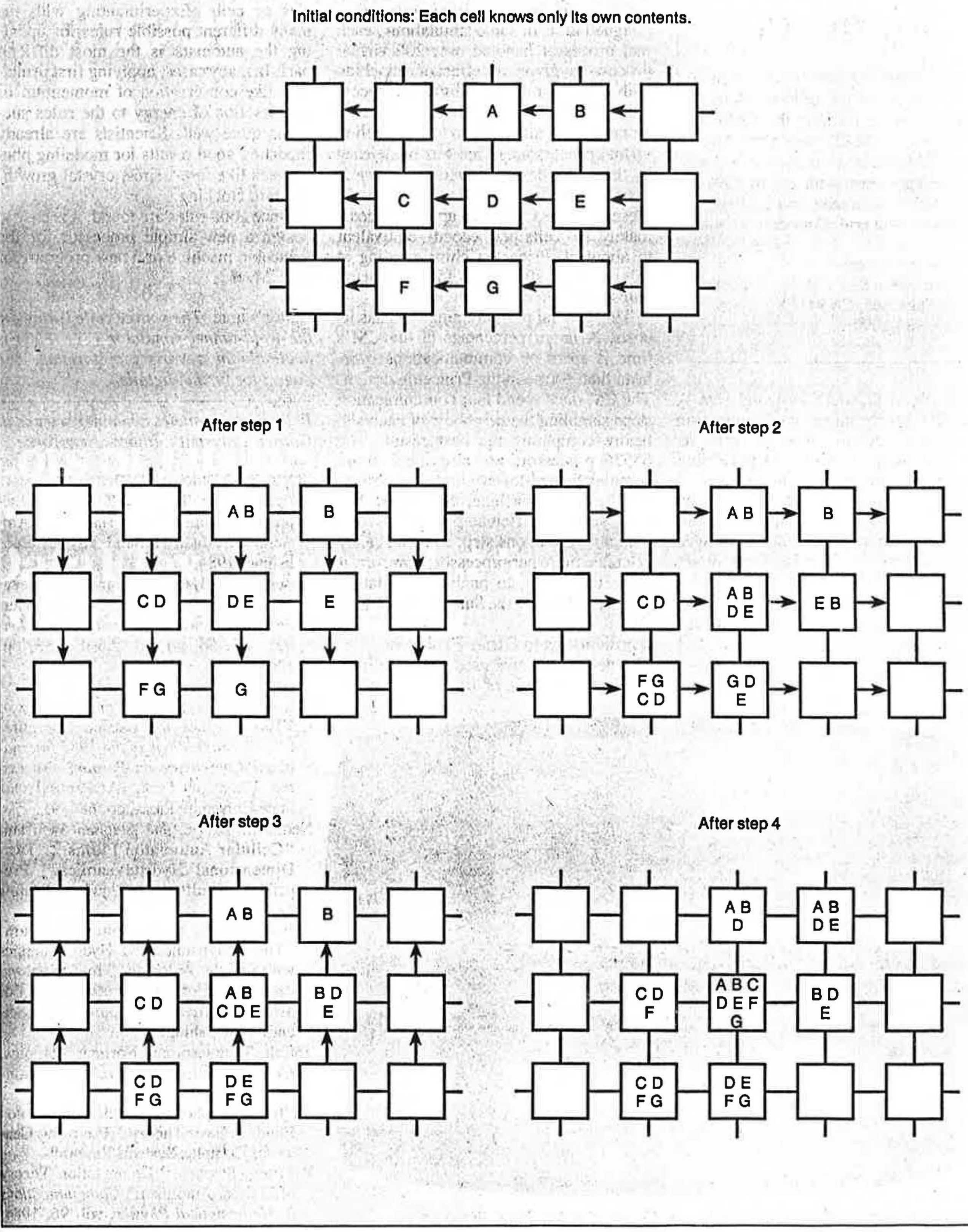


Figure 4: *A section of the square grid of the Connection Machine, illustrating how information about a cell's six neighbors (A, B, C, E, F, and G; D is the center cell) is funneled through four pathways per cell using a four-step process. At each stage, the letter inside the cell indicates how much is "known" to that cell.*

A Processor for the PC

Personal computer owners aren't left out of the cellular-automaton game altogether; the CAM-6 Processor, a $1500 plug-in board for the IBM PC, includes a specially designed processor with a 2- by 4096- by 4-bit lookup table and a 256- by 256- by 4-bit grid. Contact:

Systems Concepts
55 Francisco St.
San Francisco, CA 94133
(415) 984-1000

bors to the north, south, east, and west. A specially synchronized communication phase simulated the hexagonal grid, as shown in figure 4. (The CM is flexible enough to be programmed for any type of grid or random connections, such as my hexagonal arrangement, but not without slowing communications for messages that pass between chips lacking a direct wire. In these cases, intermediate processors must act as messengers. The planar grid with four connections keeps the process moving at the maximum capacity of the machine.)

One particularly clever addition to the software lets the programmer define virtual processors. The front-end computer turns this into code that lets each real processor multitask the virtual processors assigned to it. In some simulations, each real processor handled over 200 virtual processors, giving the effect of a machine with over 14 million individual processors. (The CM-2, with its increased memory, can simulate up to 550 million virtual processors.) This was made easy by the CM's design for straightforward, easy-to-program parallel processing. In these tests, the CM could update approximately 10^9 cells per second—equivalent to about 50 Princeton chips running at full capacity. (But about 1500 Princeton chips are hooked up to the Sun-3.)

This general programming ease has its costs. A larger percentage of the CM's time is spent on communications overhead than is true in the Princeton design. The CM must spend four communication steps sampling the neighbors of each site before computing the next result. The 65536 processors are also much more complex than necessary for this problem. The Princeton machine, by contrast, has very simple, efficient processors that communicate in one step, so it can easily calculate more per processor; however, it lacks the ability to analyze the data—hence the need for the Sun-3 workstation.

Applications to Other Problems

The design precepts used in the cellular automaton models of liquids and gases can easily be adapted to simulate any system whose global behavior is determined by the local behavior of many small particles or cells. Experimenting with the many different possible rules for updating the automata is the most difficult part. In many cases, applying first principles like conservation of momentum or conservation of energy to the rules succeeds quite well. Scientists are already reporting good results for modeling phenomena like forest fires, crystal growth, and bird flocking.

Once good rules are found, it is easy to design a new simple processor for the Princeton machine or a new program for the CM. ■

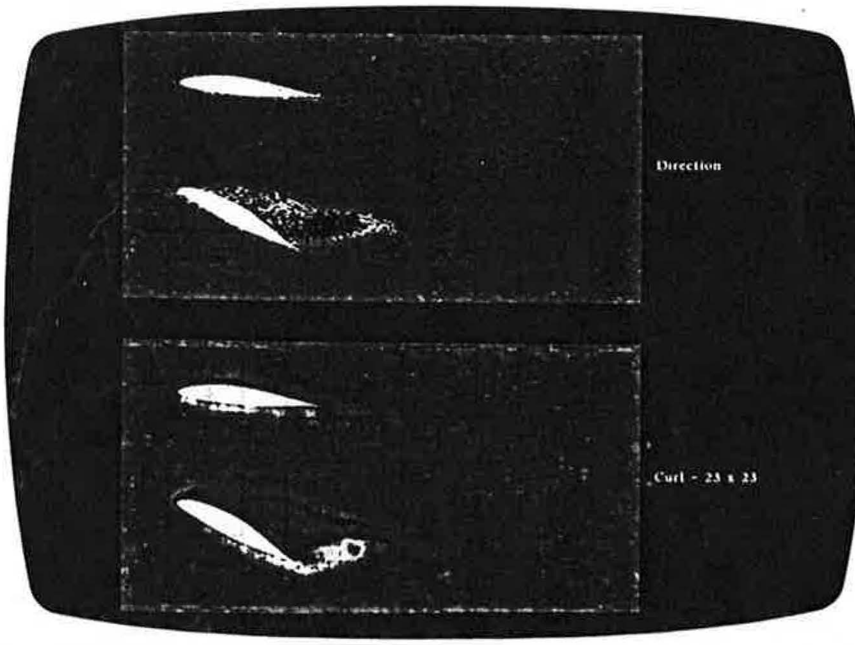


Photo 2: *Airfoil simulation made using the Connection Machine. In the upper section, red represents cells moving to the left, green is cells moving up, and blue is cells moving down. In the lower section, red represents straight flow, green is counterclockwise rotation, and blue is clockwise rotation.*

Editor's note: *The source code listing for the wind-tunnel simulator (*`FFA.PAS`*) is available in a variety of formats. See page 3 for further details.*

Peter Wayner studies computer science at Cornell University, Ithaca, New York.

SUGGESTED READING

Farmer, Doyne, Tommaso Toffoli, and Stephen Wolfram. *Cellular Automata*. Amsterdam: North-Holland Physics Publishing, 1984.

Frisch, U., B. Hasslacher, and Y. Pomeau. "Lattice Gas Automata for the Navier-Stokes Equation." *Physical Review Letters*, vol. 56, no. 14, April 1986, pp. 1506–8.

Kugelmass, Steven, Richard Squier, and Kenneth Steiglitz. "Performance of VLSI Engines for Lattice Computations." *Proceedings of the 1987 International Conference on Parallel Processing*. University Park, PA: Pennsylvania State University Press, pp. 684–91.

Nemnich, Bruce, and Stephen Wolfram. "Cellular Automaton Fluids 2: Two-Dimensional Hydrodynamics." Preprint. Submitted to *Physical Review Letters*.

Salem, James, and Stephen Wolfram. "Thermodynamics and Hydrodynamics with Cellular Automata." In *Theory and Applications of Cellular Automata*. Ed. Stephen Wolfram. Singapore: World Scientific Publishing, 1986.

Toffoli, Tommaso, and Norman Margolus. *Cellular Automata Machines*. Cambridge, MA: MIT Press, 1987.

Wolfram, Stephen. "Cellular Automaton Fluids 1: Basic Theory." Pre-print. Center for Complex Systems Research.

Wolfram, Stephen. "Computation Theory of Cellular Automata." *Communications in Mathematical Physics*, vol. 96, 1984, pp. 15–57.

Wolfram, Stephen, ed. *Theory and Application of Cellular Automata*. Singapore: World Scientific Publishing, 1986.