JOURNAL OF COMPUTATIONAL PHYSICS 84, 311–325 (1989)

A Scalable Architecture for Lattice-Gas Simulations

STEVEN D. KUGELMASS AND KENNETH STEIGLITZ

Department of Computer Science, Princeton University, Princeton, New Jersey 08544

Received April 12, 1988; revised November 16, 1988

We describe a computer architecture for simulating the Frish–Hasslacher–Pomeau (FHP) lattice-gas model for fluid flow. It consists of a 1-dimensional pipeline of identical full-custom chips hosted by a general-purpose computer. Sixty-four-pin DIP chips for the site-update rule were fabricated by MOSIS in 3μ CMOS, each containing more than 65,000 transistors. The chips have been tested and perform reliably at a 7 MHz clock rate. There are two processors per chip, so each chip is capable of 14 million site-updates/s/chip. A 10-chip pipeline was interfaced and tested with a SUN workstation. The present test interface transfers a word at a time through the CPU, and also requires the host to handle boundary conditions, so it achieves only about 2 million site-updates/s. This is still about 16 times faster than a software simulation on the DEC VAX 8650. With external memory and a pipeline processor to handle boundary conditions, this 10-chip pipeline should be capable of 140 million site-updates/s. The pipelined architecture has a property we call *linear speedup*. That is, n processors of fixed size and cost provide n times the throughput of one processor on the same problem instance, with no increase in memory bandwidth.

1. INTRODUCTION

Many problems are characterized by the fact that they deal with data values distributed on a regular mesh or *lattice*. They arise in a wide variety of applications such as image processing, computer vision, physical modeling, and the solution of partial differential equations. These problems often require tremendous computational resources and are typically solved on large general-purpose supercomputers. Special purpose SIMD and MIMD processor systems have also been built for these problems [1–4].

The cellular automaton (CA) is a simple type of lattice computation that was studied in the early days of computer science as a model of self-replication [5]. A cellular automaton is built from the following components:

(1) A discrete *state space* Σ over $\{0, ..., k-1\}$.

(2) A set of sites called the *domain* Δ. This is a discrete regular lattice in n dimensions. Each site in the domain has associated with it a value from Σ and this is referred to as the *state* of the site. The state of all the sites taken collectively is called the state of the CA.

0021-9991/89 $3.00

(3) An *update rule* Φ, which gives the state at time $t+1$ from the state at time t using the states of sites from a fixed neighborhood in the n-dimensional lattice. Φ can therefore be thought of as a local re-writing rule.

More recently, people have studied cellular automata because they may be able to mimic the way that computations are performed in nature—they model the simultaneous local interaction of simple objects distributed over a large area or volume [6]. Stephen Wolfram [7, 8] published a series of papers describing an experimental study of some properties of a class of 1-dimensional cellular automata, a study made possible by vastly increased computational power. He showed that very simple cellular automata could exhibit kinds of behavior usually associated with much more complex systems.

In 1986, Frisch, Hasslacher, and Pomeau [9] showed that a cellular automaton with a hexagonally connected geometry can simulate fluid dynamics, extending previous work on lattice gasses (see [10], for example). More precisely, they showed that in the limit of large lattice size, appropriate state averages of their automaton converge to solutions to the 2-dimensional Navier–Stokes equation. Simulations of their automaton have verified many of the qualitative features of actual fluid flow [11] as well as some quantitative features, such as momentum density profile [12]. This was perhaps the earliest concrete example of a cellular automaton offering a competitive alternative to traditional methods for a problem as difficult and important as fluid dynamics [13–15], at least in certain regimes.

Toffoli and Margolus [16, 17] built a high-performance cellular-automaton machine that is programmable and very easy to use, but not easy to make faster. In this paper we describe the design and implementation of an architecture for lattice computations that can, in principle, be indefinitely pipelined and extended to provide arbitrarily high throughput for a given problem. It has the property we call *linear speedup*—a property not shared by other special purpose architectures dedicated to problems of this genre [16–20]. We use the FHP lattice-gas cellular automaton [9] as the touchstone for our work because it has many of the properties that are characteristic of lattice computations: small storage per site, large number of sites, small neighborhood per site, and complete uniformity of the algorithm.

2. The FHP Lattice Gas Model

The FHP lattice gas model is a discrete particle model for fluid dynamics [9]. It is based on a regular hexagonal lattice, giving each site six nearest neighbors. Particles move synchronously around the lattice at unit speed, colliding only at lattice sites, and possibly scattering as a result. The links along which they move are bi-directional, and particles can interact only at lattice sites. Two particles traveling in opposite directions on the same link are not considered to be colliding.

Each site has six neighbors, and outgoing particles may be present on any of

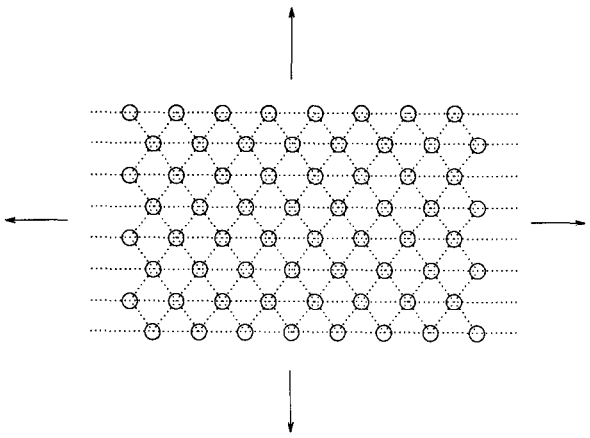

FIG. 1. The hexagonal lattice.

these links. (See Fig. 1.) Therefore each site in the lattice has 64 states. However, some models must also provide for the possibility that there is a particle at rest at the site (a "center") and, additionally, that the site may not be part of free space, but rather may be solid. The "solid" sites are used to compose boundaries and obstacles for the simulation. This raises the minimum number of bits required to describe the state of a site from six to eight.

3. SCATTERING RULES

The particles moving in the lattice scatter at the sites according to rules that are designed to make the simulation converge to solutions of the Navier–Stokes equation. The rules have two fundamental properties: conservation of mass (particle number), and conservation of momentum. The rest particles in Rules C1 and C2 are introduced to lower the effective viscosity. We can still consider energy to be conserved if we think of them as being internally excited.

Further, the collision rules are designed to minimize a particle's mean-free-path: the number of lattice edges it traverses without colliding and scattering with another particle. The smaller the mean-free-path, the lower the viscosity of the lattice gas, and the larger the Reynolds number of the simulation. In general, it is difficult to achieve high Reynolds numbers and we therefore try to have as many particle interactions (scattering rules) as possible. In the event that a particle cannot scatter with another when it reaches a lattice site, it continues its trajectory with its previous velocity undisturbed.

Figure 2 illustrates the particle collision rules. The configuration before the collision is shown on the left, with the configuration after the collision on the right. In the case where symmetry would imply the existence of more than one member of a rule family, only one member is shown. The complete set of rules can be derived

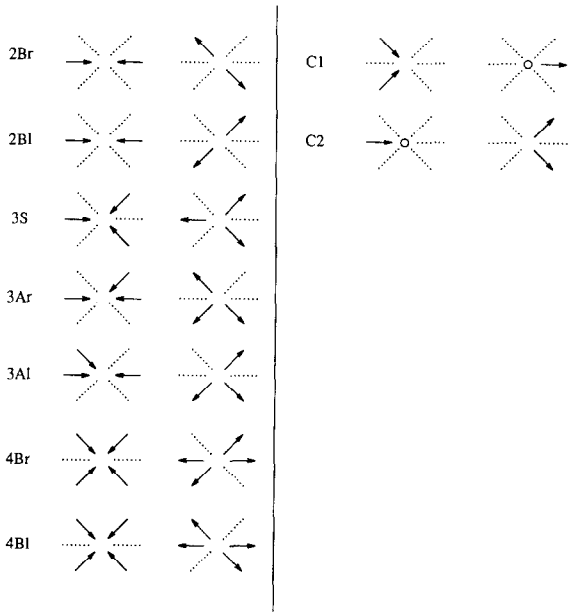

FIG. 2. Particle collision rule classes.

by rotation of the before and after configurations by any multiple of 60°. The small circle at the site means that a particle is present there at rest.

Obstacles, such as cylinders, plates, or wings, are placed in the lattice by changing a site from "free-space" to "solid". While a "free-space" site scatters particles according to the rules outlined above, a "solid" site behaves differently and there are several permissible choices. The simplest possibility is to have each particle reflect back along its incoming path. Another possibility is to have each particle reflect in such a way that its incident angle equals its reflected angle. Unfortunately, the latter entails having different kinds of solid sites, depending on position in an object.

These two choices also correspond to different kinds of boundary-layer effects [9]. In the first case, the rule leads to a "no-slip" boundary, where the velocity of the fluid near a surface is zero. The second case corresponds to a "slip" condition; the fluid next to a boundary can move. We adopt the "no-slip" boundary because it is simpler and more realistic for the kinds of fluid-dynamic phenomena that we expect to simulate.

Three of the rule classes, 2B, 3A, and 4B have L and R variants, so called because they scatter left-ward and right-ward, respectively. It is critical that both of these scatterings take place with equal probability or a systematic bias will be introduced that will destroy the convergence properties of the simulation. When the simulation is performed in software, it is a simple matter to "flip a coin" and choose either an L or an R rule when appropriate. Alternatively, the software simulation

can use the R rule during even-numbered generations and the L rule during the odd-numbered ones (or vice versa). In hardware, however, it is undesirable to "flip a coin" or to change rules on every generation. Instead, one of the chip's two processors performs the R rules all the time, and the other performs the L rules. This has the effect of placing the L and R rules throughout the lattice in a manner similar to the squares of a checkerboard. The unbiased spatial distribution is equivalent (informally) to an unbiased temporal distribution (coin flipping) because the events of interest occur randomly throughout the lattice and the simulation.

The sequence in which particle collisions are computed at lattice sites is also critical to the success of the simulation. In concept, the entire lattice is updated simultaneously, with each update called a *generation.* In practice, the new site values can be computed in any sequence provided that particles from generation i are the only ones used to compute particle velocities for generation $i+1$. If this rule is violated, it is easy to give a configuration and a site update sequence that violates the mass (and therefore the momentum) conservation law [9].

The net velocity of an ensemble of particles over a small region of the lattice (spatial averages) yields the hydrodynamic behavior that is sought. This function is not computed by the special purpose chip; it is done as part of a post-processing phase.

4. Serial Pipelined Architectures for Lattice Processing

The architecture that we adopted is a single serial pipeline (see Fig. 3). This has the benefit that the bandwith to the processor system is small even though the number of active processing elements (PE's) is large. The serial technique has been used for image processing where the size of the 2-dimensional grid is small and fixed [21–23], and has also been used to design a high-performance custom processor for a 1-dimensional cellular automaton [24]. The appealing aspects of the serial

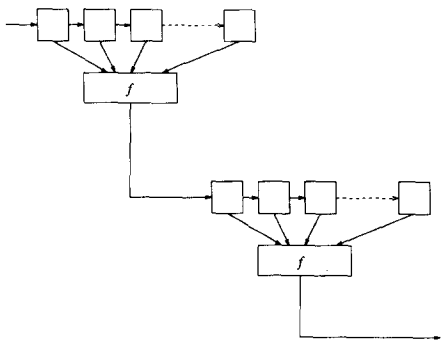

FIG. 3. One-dimensional pipeline. The data enters the shift registers (represented by the square boxes) and is accessed by the next-state function f.

architecture are the simplicity of its design, its small area in comparison to other architectures, and the small input/output bandwidth requirement.

Consider what is required to pipeline a computation. We must guarantee that the appropriate site values of the correct ages are presented to the computing elements. In the case of the lattice gas cellular automaton, we can express this data dependency as

$$a_i^{t+1} = f(N(a_i^t))$$

where a_i^t is the value at lattice site a_i at time t, $N(a_i^t)$ is the *neighborhood* of the lattice site a_i at time t, and f is the function that determines the new value of a_i based on its neighborhood. The cellular automaton requires all the points in the neighborhood of a_i to be the same age in order to compute the new value, a_i^{t+1}.

One-dimensional pipelining also requires a linear ordering of the sites in the array. That is, we wish to send the values associated with the sites one at a time into the 1-dimensional pipeline and receive the sequence of sites in the same order possibly some generations later. Therefore, we would like sites that are close together in the lattice to be close together in the stream, so that the serial PE can use as little local memory as possible. Unfortunately, there is no sub-linear embedding of an array into a list [25, 26] and so we must store two raster lines on each chip.

5. Wide Serial Architecture

Assume that data from the 2-dimensional array is serialized by a row-major raster scan. Throughput in a serial architecture can be improved by adding concurrency at each level of the pipeline. One way to accomplish this is to have each pipeline stage compute the new value of more than one site each clock period. For example, if the computation at PE j is at the point where site a, circled, is to be updated, then PE j contains the data indicated by strike-out in Fig. 4. We could allow a second PE j' to compute site $a+1$ at the same time if we store just one more data point (see Fig. 5).

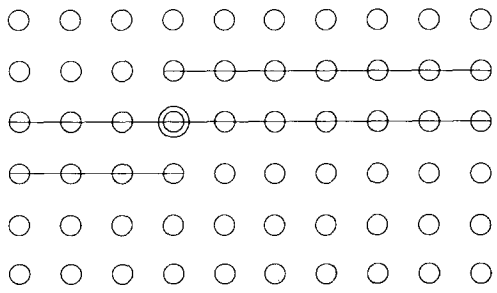

FIG. 4. Data used by one processing element. A Rectangular lattice is shown for simplicity.

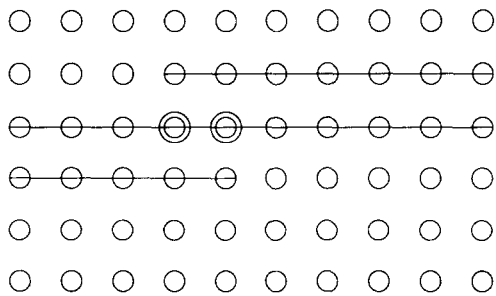

FIG. 5. Data used by two parallel processing elements.

The most attractive feature of this scheme is that throughput is increased, but at a cost of only the incremental amount of memory needed to store the extra sites. However, there is a price to pay: two new site values are required every clock period so that two site updates can be performed. The extra PEs require added bandwidth to and from the chip and this implies that the main memory system must provide that bandwidth as pins or wires.

As an example, Fig. 6 shows how two PEs on the same chip can cooperate on a computation. Each square of the shift register holds the value of one site in the lattice. Every clock period, two new site values are input to the chip, two sites updated, and their values output to the next chip in the pipeline.

A particle-collision rule-set is chosen from a set of four possibilities by applying control voltages to two pins of the chip. This permits a small degree of programmability of the device and some control over the viscosity of the lattice gas. The four particle collision rule sets are numbered (refer to Fig. 2 for descriptions of the rules 2B, 3S, etc.):

0. 2-body, 3-body, centers (2B, 3S, C1, C2)
1. 2-body, 3-body, centers, 3-body-asymmetric (2B, 3S, 3A, C1, C2)
2. 2-body, 3-body, centers, 4-body (2B, 3S, C1, C2, 4B)
3. 2-body, 3-body, centers, 3-body-asymmetric, 4-body (2B, 3S, C1, C2, 3A, 4B).

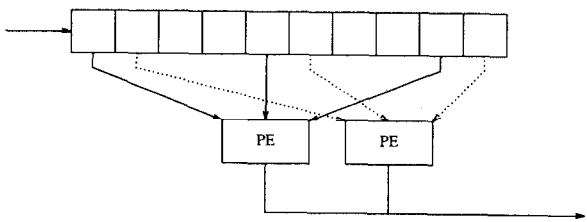

FIG. 6. Wide serial architecture. The two processing elements operate in parallel and update two successive sites.

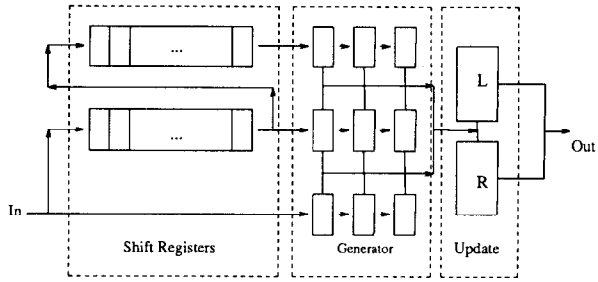

FIG. 7. Basic architectural plan. The boxes marked "R" and "L" correspond to the two processing elements in Fig. 6.

Each chip consists of three principal parts: the shift-register memory, the neighborhood generator, and the update processors. The basic architectural plan is shown in Fig. 7.

The shift registers provide delay to permit 2-dimensional processing of the 1-dimensional data stream. When a data value is entered into a shift register, the registers emit the value of the site in the same column of the previous line. The shift registers are not of programmable length; chip area considerations led to a register length of 256. Each stage in the shift register is wide enough to hold two sites (16 bits) so that the values of 512 sites fit within each of the two shift registers.

The *neighborhood generator* converts *outgoing* particle information from the neighbors of the two sites to be updated into *incoming* particle information for those sites. The neighborhood generator is composed of a set of shift registers in which the required bits are extracted and brought out to the update processors. These processors map from a site's incoming particle configuration to its new state (outgoing particle configuration).

The chip contains two update processors, which compute left- and right-variants of a rule, respectively. As we mentioned above, this is required for isotropy of solutions.

With each major clock cycle of the chip, the following activities take place: A new 16-bit word, comprising the states of two lattice sites, is latched into the chip's shift registers and *neighborhood generator*. All other data values in the registers are shifted forward one stage. When this shift is completed, the neighborhood generator is then presenting extracted incoming state information for the "current sites" to the *update processors* which place the new site values on the output pins. Because of the latency in the shift registers, the site values emitted are one row above and one column behind those just accepted.

6. HARDWARE IMPLEMENTATION

The floor plan of the chip is shown in Fig. 8. The majority of the chip real estate is dedicated to the shift-register memory. The neighborhood generator and the

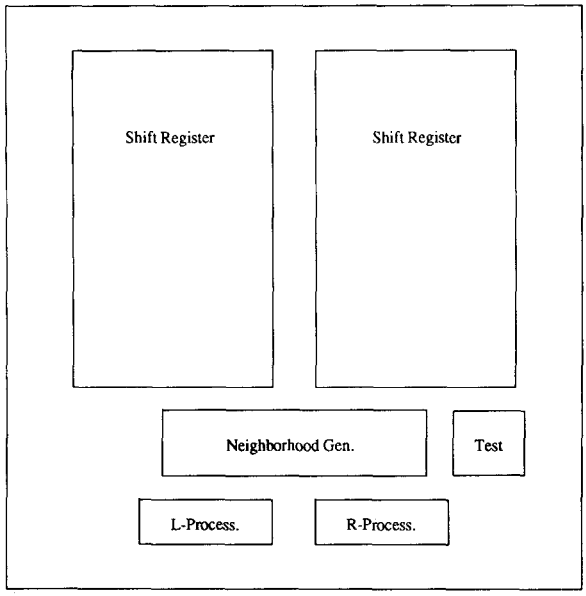

FIG. 8. Chip floor plan.

update processors are much smaller. The test circuitry aids in the testing and verification of the chip behavior.

7. UPDATE PROCESSORS

The update processors map from the input configuration of a site to its output configuration (new state). We wanted to provide some degree of flexibility or programmability over the chip, but at the same time, we wanted to retain the simplicity, ease of design, and small size of a fixed rule implementation.

There is a spectrum of possible structures, ranging from a Turing machine equivalent processor to combinational logic. The former offers very general programmability and flexibility, whereas the latter is very restricted in the functions that it can compute. What we really want is something between these two extremes.

We chose four sets of collision rules which we knew worked by having simulated them. We built a PLA that implemented all of them, and found that this PLA was only twice the size of a PLA that implemented any one of the rule sets. The PLA is "programmed" by selecting a rule set at the time the inputs are presented. This is accomplished via two control lines that determine which rule-set number is active at that time.

One PLA performs R rotations for the 2-body and 4-body cases (rules 2Br and 4Br), and the other performs L rotations (rules 2Bl and 4Bl). Their layouts were built by programs, no hand layout was necessary in this case. Two programs, *allL.c*

and *allR.c*, were used to generate truth tables for the functions. These truth tables were "minimized" by the program *espresso*, and the results fed to a PLA generator, *mpla*. They use a static structure, with pseudo-nMOS pull-ups for the AND and OR planes.

8. Comparison with Other Machines

We refer to the ten-chip prototype constructed at Princeton as LGM-1 (for lattice-gas machine). CAM-6 [16, 17] scans the 2-dimensional array to create a serial stream of data in the same way as LGM-1, but it is not a pipelined architecture. It is also designed and tuned for a particular host architecture, the IBM PC. Even though it is extensible to larger problems, it cannot provide increased throughput on the same problem. This is where the LGM-1 architecture excels, because it can be pipelined to achieve nearly an arbitrary speedup on the same problem instance.

Sternberg's machines [22, 27] are significantly more complicated, but they are pipelined. Even though they are more powerful than the special-purpose processor of LGM-1, they obtain this power by using more input/output bandwidth. See [26] for a treatment of computation rate versus input/output bandwidth trade-offs.

In spirit, our work is closest to that of Broderson and Ruetz [21]. They built separate chips for the various functions of their image processor, including a line-delay chip that we implemented as shift register. They concatenated their chips in a pipeline to achieve a final function, which is the composition of the functions of each chip in the pipeline. Since we perform the same function at each stage of the computation, we achieve high performance from chip concatenation, whereas Broderson and Ruetz were forced to make all their chips fast in order for their system to function at all.

Most of the other work on lattice-gas cellular automata implementation has been done in software on general purpose computers. The only directly comparable figures therefore are total throughput. We defer this comparison until the next section where we discuss the performance of our prototype system, LGM-1.

9. Performance Levels

We built the 10-chip pipeline and tested it by interfacing it with a SUN 3/160C workstation. The chips themselves perform reliably at a clock rate of 7 MHz, or 14 million site-updates/s/chip. The ultimate speed limit of this 10-chip pipeline is therefore 140 million site-updates/s, with speedup linear in the number of chips. This figure is not obtained with the current system for two reasons, neither of which is fundamental and each of which could be eliminated with further work on interface and support hardware. Because we view this project primarily as a verification of the architecture and not as the construction of a production machine, we have not

expended the effort necessary to realize this performance. Such effort would be better spent on a second-generation chip design which includes boundary processing, as explained below.

The first serious limit to the speed is the interface with the SUN workstation. Each site value is read and written to the bus, one-at-a-time, by the supervisor program. The 10-chip pipeline itself can be stepped by the host workstation every $3\mu s$, which, since there are 20 updates in the pipeline in one basic cycle, amounts to 7 million site-updates/s. However, because the complete lattice does not fit in the pipeline, it is necessary to read and write to the memory of the SUN, which further slows down the computation. A production machine would eliminate these problems by using an external memory to hold the complete lattice, so that the data can be circulated through the pipeline without passing through the host, which would then be used only for down-loading initial conditions and up-loading results.

The second factor slowing the machine down is the need to treat the boundary sites differently from interior sites, and that affects not only the speed, but the extensibility. The boundary conditions of the lattice are not now handled in the pipeline, but they need to be if the pipeline is to be deepened. Right now, the sites at the top, bottom, and sides of the lattice are loaded with randomly moving particles in the host. These particles are moving with an appropriately biased velocity, and we can think of this computation as a "fan" for the "wind-tunnel." (Thus, the finite lattice simulates a section of an infinite lattice without periodic boundary conditions.) With a 10-chip pipeline these boundaries are accessible to the host only every 10 generations, and if the pipeline is deepened much beyond that, the particle distribution becomes seriously biased and the simulation does not converge. The extra time in the host to detect boundaries and insert random numbers slows the 10-chip machine down further to about 2 million site-updates/s, still about 16 times faster than a software simulation on the DEC VAX 8650. Thus despite the fact that the interface achieves only a small fraction of the potential speed and should be considered only a testbed, the machine is still useful for production.

To get around this limitation, both on speed and extensibility, we must provide fan processors at least every 10 or so chips inside the pipeline because these intermediate generations are not otherwise accessible. Such a fan processor can be inserted in the pipeline with no degradation of speed and can in fact be incorporated on the chip if space permits. The design of such a fan processor is described more fully in [30, 31], and sections of it have been built and tested in TTL. The next generation of the processor chip will use a denser technology and should be able to include such a fan.

It is difficult to put exact figures on the cost of hardware for a lattice-gas machine based on this architecture, but some idea of its scale can be given as follows: The custom chip could be fabricated in quantity (by MOSIS) at less than \$100 per chip. Each 10 chips in the pipeline requires a circuit board, which should cost no more than another \$500. External memory for the lattice sites, holding perhaps 1 or 2 Mbytes, is relatively inexpensive. It is important to realize here that the bandwidth

of this memory (2 bytes wide) need be only the 7 MHz clock rate of the chips and is independent of the length of the pipeline. The cost of the raw materials is therefore about $1500 for each 10 chips in the pipeline, plus a few thousand dollars for the external memory and interface with the SUN, which can now be slow without interfering with the production rate of the machine. The site values are available in raster-scan order in a stream and can therefore be used to drive a real-time display. This is the cost of only the raw materials, assuming the MOSIS fabrication facility is used for a quantity run of chips, so these figures must be viewed with caution.

Hayot *et al.* [28] discuss the implementation and performance of the best known algorithms for the FHP lattice-gas software simulation as executed on several different general-purpose machines. They also extrapolate to predict the performance of larger sibling computers on the same problem.

Hayot *et al.* report a site-update rate of 2.3 million sites/s on a 32-processor INTEL iPSC hypercube. An Alliant FX-8 shared memory vector supercomputer gave only 0.7 million sites/s on the same problem because a large part of the code had not been vectorized nor made concurrent. We summarize the reported measured performance of various machines in Table I (from [28]).

The RAP1 machine [29] is a dedicated FHP lattice-gas machine designed and built at École Normale Supérieure in Paris. It uses the same basic architecture as CAM-6. The Connection Machine is a general-purpose computer composed of 65536 one-bit processors arranged in a multi-dimensional hypercube.

Hayot *et al.* were able to predict the performance of more powerful hypercube processor systems by extrapolation. They report a predicted performance of 86 million sites/s on a 128-processor INTEL iPSC-VX, and 120 million sites/s on a 1024-processor N Cube-10. As mentioned above, the pipeline architecture described here, if properly supported with external memory and fan processing in the pipeline, could achieve a throughput of 14 million site-updates/s/chip.

The 10-chip pipeline interfaced with the SUN workstation was operated continuously for more than a week without any detected hardware failures and is

TABLE I

Reported Performance of Different Machines on Lattice-Gas Problems (after [28])

System	Lattice size	Site updates/s
32-processor iPSC	$4K \times 2K$	2.3×10^6
Alliant *FX*-8	$4K \times 4K$	0.7×10^6
CAM-6	256×256	4.0×10^6
RAP1(ENS Paris)	512×256	6.5×10^6
Connection Machine	$4K \times 4K$	10^9
Pipeline Architecture	512×1024	14×10^6 per chip (projected)

Note. The figure for the pipeline architecture is projected from the speed and size of the chip described in this paper.

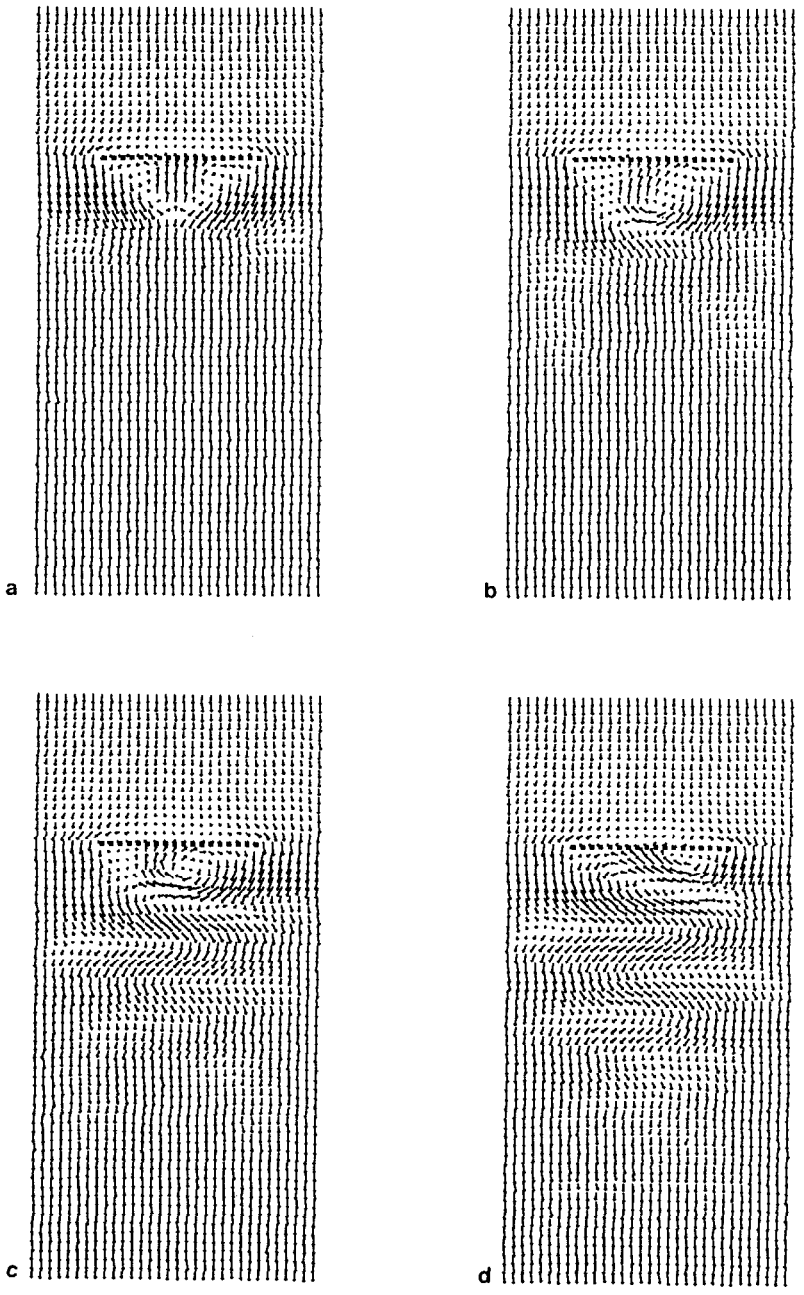

FIG. 9. Four snapshots of the velocity field of a flow simulation using LGM-1, showing flow from top to bottom past a bluff plate: (a) top left, after 1000 generations; (b) top right, after 2000; (c) bottom left, after 3000; (d) bottom right, after 4000.

still in use. We performed simulations of flows over variously shaped objects, and examples of the results are shown in Fig. 9. The particle density in each case was 0.2, which we know to yield good results and a relatively small fluid viscosity. We estimate the Reynolds number at 40 for these simulations.

Figure 9 shows snapshots of the velocity field of a flow simulation, taken at generations 1000 to 4000, for a 512×1024 lattice. The fluid flows from top to bottom past a bluff plate and develops asymmetric vortices in the shadow of the plate.

We should mention at this point that the 10-chip pipeline built at Princeton, as well as any other scalable pipeline machine, is controlled entirely from the host. The user writes a small, simple program in a high-level language that has access to the external hardware and the lattice data via a few parameters and a virtual address.

10. Concluding Remarks

The important feature of the pipeline architecture is its extensibility with linear speedup. More throughput can be bought by adding identical chips to the pipeline, with no additional demands on memory bandwidth or size.

The 10-chip pipeline built at Princeton is intended primarily as an experiment in architecture, rather than a serious tool for lattice-gas experimentation, although it is being used daily. As mentioned, the present system is slowed down considerably by the interface with the host and its extensibility and speed are both limited by the absence of a fan processor. If we follow this approach to achieve really massive parallelism, we also need to study the limiting effects of clock distribution and reliability in very long pipelines.

Acknowledgments

We thank Mark Taylor for writing the display software; Tarun Khanna for designing and building a prototype fan processor, Chumki Basu for documentation; Richard Squier for test coding and timing; and Mark Greenstreet for helpful discussions. This work was supported in part by NSF Grant MIP-8705454, U.S. Army Research Office - Durham Contract DAAG29-85-K-0191, and DARPA Contract N00014-82-K-0549. The chips were fabricated by MOSIS.

References

1. G. Barnes, R. Brown, M. Kato, D. Kuck, D. Slotnick, and R. Stokes, *IEEE Trans. Comput.* **C-17**, 746 (1968).
2. K. E. Batcher, *IEEE Trans. Comput.* **C-29**, 836 (1980).
3. D. M. Nosenchunck and M. G. Littman, in *Proceedings, 23rd Annual Space Conference, John F. Kennedy Space Center, FL, 1986.*
4. S. Manohar and G. M. Baudet, Technical Report CS-86-06, Department of Computer Science, Brown University, Providence, RI, 1986 (unpublished).

5. J. VON NEUMANN, *Theory of Self-Reproducing Systems*, edited by A. W. Burks (University of Illinois Press, Urbana, IL, 1966).
6. R. P. FEYNMAN, *Int. J. Theoret. Phys.* **21**, 467 (1982).
7. S. WOLFRAM, *Nature* **311**, 419 (1984).
8. S. WOLFRAM (Ed.), *Theory and Applications of Cellular Automata* (World Scientific, Singapore, 1986).
9. U. FRISCH, B. HASSLACHER, AND Y. POMEAU, *Phys. Rev. Lett.* **56**, 1505 (1986).
10. S. HARRIS, *Phys. Fluids* **9**, 1328 (1966).
11. M. VAN DYKE, *An Album of Fluid Motion* (Parabolic, Stanford, CA, 1982).
12. L. P. KADANOFF, G. R. MCNAMARA, AND G. ZANETTI, *Complex Systems* **1**, 791 (1987).
13. J. R. HERRING, S. A. ORSZAG, R. H. KRAICHNAN, AND D. G. FOX, *J. Fluid Mech.* **66**, 417 (1974).
14. S. A. ORSZAG, *J. Fluid Mech.* **41**, 363 (1970).
15. P. J. ROACHE, *Computational Fluid Dynamics* (Hermosa, Alburquerque, NM, 1972).
16. T. TOFFOLI, *Physica B* **10D**, 195 (1984).
17. T. TOFFOLI AND N. MARGOLUS, *Cellular Automata Machines: A New Environment for Modeling* (MIT Press, Cambridge, MA, 1987).
18. R. B. PEARSON, J. L. RICHARDSON, AND D. TOUSSAINT, *J. Comput. Phys.* **51**, 241 (1983).
19. A. HOOGLAND, J. SPAA, B. SELMAN, AND A. COMPAGNER, *J. Comput. Phys.* **51**, 250 (1983).
20. N. MARGOLUS, T. TOFFOLI, AND G. VICHNIAC, *Phys. Rev. Lett.* **56**, 1694 (1986).
21. P. A. RUETZ AND R. W. BRODERSON, *IEEE J. Solid-State Circuits* **SC-22** (1987).
22. S. R. STERNBERG, "Computer Architectures Specialized for Mathematical Morphology," in *Algorithmically Specialized Parallel Computers*, edited by H. J. Siegel (Academic Press, New York, 1985), p. 169.
23. J. KITTLER AND M. J. B. DUFF (Eds.), *Image Processing System Architectures* (Research Studies Press, Wiley, New York, 1985).
24. K. STEIGLITZ AND R. R. MORITA, in *Proceedings, 1985 IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, Tampa, FL, March.*
25. A. L. ROSENBERG, *SIAM J. Comput.* **4**, 443 (1975).
26. S. D. KUGELMASS, R. SQUIER, AND K. STEIGLITZ, *J. Complex Syst.* **1**, 939 (1987).
27. S. R. STERNBERG, "Pipeline Architectures for Image Processing," in *Multicomputers and Image Processing, Algorithms and Programs*, edited by L. Uhr (Academic Press, New York, 1982), p. 291.
28. F. HAYOT, M. MANDAL, AND P. SADAYAPPAN, "Implementation and Performance of a Binary Lattice Gas Algorithm on Parallel Processor Systems," Department of Physics, Ohio State University, 1987 (unpublished).
29. A. CLOUQUEUR AND D. D'HUMIÈRES, *Complex Syst.* **1**, 585 (1987).
30. S. D. KUGELMASS, Thesis, Department of Computer Science, Princeton University, Princeton, NJ, 1988 (unpublished).
31. T. KHANNA, Report of Senior Independent Work, Department of Computer Science, Princeton University, Princeton, NJ, 1988 (unpublished).