JOURNAL OF COMPUTATIONAL PHYSICS **80**, 277–287 (1989)

Implementation and Performance of a Binary Lattice Gas Algorithm on Parallel Processor Systems

F. HAYOT, M. MANDAL, AND P. SADAYAPPAN

Ohio State University, 174 West 18th Avenue, Columbus, Ohio 43210

Received June 17, 1987; revised November 18, 1987

We study the performance of a lattice gas *binary* algorithm on a "real arithmetic" machine, a 32 processor INTEL iPSC hypercube. The implementation is based on so-called multi-spin coding techniques. From the measured perfomance we extrapolate to larger and more powerful parallel systems. Comparisons are made with "bit" machines, such as the parallel Connection Machine. © 1989 Academic Press, Inc.

I. INTRODUCTION

Lattice gas hydrodynamics is an approach to modeling fluid phenomena through the use of a simple microscopic model [1]. The fluid is described by particles located at the sites of a lattice, constrained to move in discrete fashion between lattice points. Macroscopic fluid properties are obtained from the microscopic model by averaging over collections of lattice sites. Computationally, the model is based on manipulations of binary variables. This is in contrast with the usual simulations of hydrodynamic behavior, which use real number arithmetic in solving the relevant partial differential equations, the Navier–Stokes equations. The main advantage of the lattice gas approach is that there is no round-off error and no error accumulation. Complicated boundary conditions for fluid flow are easy to take into account. Moreover, the approach is much more amenable to exploitation of massive parallelism.

Our aim is to study the implementation and performance of the lattice gas algorithm on "real arithmetic" parallel processor systems. A very efficient way to implement the binary algorithm on such a machine is through the so-called multi-spin coding techniques [2]. With these techniques, many independent binary variables are packed into each machine word. They are thus treated in parallel, and the dynamics of the physical model is represented by boolean operations on words. This technique is also very economical in terms of storage.

Parallelism is exploited at several levels: at one level through handling simultaneously all the binary variables in one word, and at another level through parallel execution on the processors of the system. Moreover, the technique is amenable to exploitation of pipelined parallelism with vector processors. It is thus very well suited for implementation on systems with multiple vector processors. Such systems clearly represent the emerging trend in supercomputing architectures, e.g., Cray XMP, INTEL iPSC-VX. We will evaluate the performance of the lattice gas algorithm on an INTEL iPSC 32 processor hypercube and extrapolate the obtained performance results to larger and more powerful systems of processors. This enables us to compare the performance potential of such parallel "real arithmetic" processors with that of parallel "bit" machines, such as the Connection Machine [3], for the binary algorithm under consideration. We have also implemented the algorithm on a shared memory concurrent/vector processor system, the ALLIANT Fx-8.

In Section II we will describe the lattice gas algorithm and, in Section III, the use of multi-spin coding techniques. Section IV contains results of performance evaluation and a comparison with "bit" machines and conclusions. Section V is devoted to discussions and outlook.

II. Lattice Gas Binary Model

The lattice gas is a model in which identical point particles move from site to site of a two-dimensional triangular lattice. (Extensions to 3 dimensions are discussed in Section V). At each site of the lattice there are six possible directions in which a particle can move, labelled *a*, *b*, *c*, *d*, *e*, *f* starting East and going counterclockwise (cf. Fig. 1). Particles move (they are never at rest) along lattice links and change

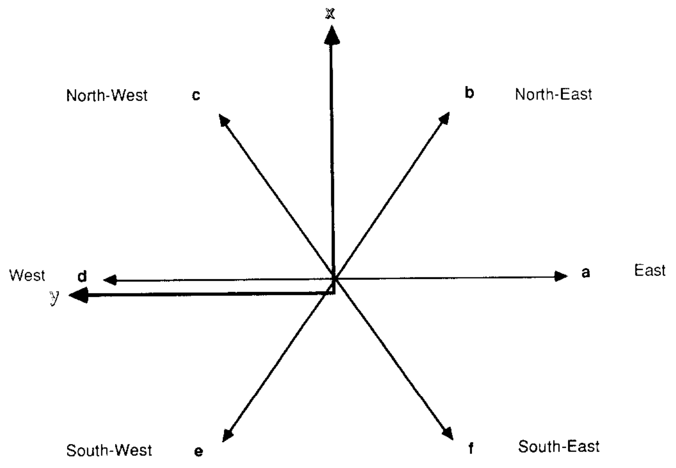

FIG. 1. Representation of a lattice site with the 6 directions emerging from it: East, Northeast, Northwest, West, Southwest, and Southeast. The binary variables in each direction are respectively *a*, *b*, *c*, *d*, *e*, and *f*. Also shown are our choice of *x* and *y* direction, the *x* direction indicating the direction of flow as in Fig. 2.

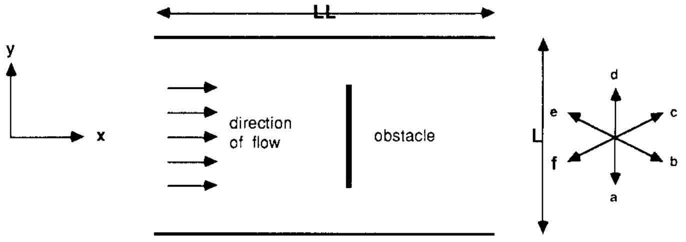

FIG. 2. Schematic drawing of our lattice system of length LL and width L, indicating the position of an obstacle set perpendicular to the flow. The 6 binary directions are again indicated as in Fig. 1.

direction only if, at a given site, a 2-body, 3-body, or 4-body collision occurs. The simulation of each time step involves a sequence of two operations, a translation and a collision computation. The translation computation transforms one lattice configuration to another, based on the current direction of motion of the particles. This is followed by a collision computation that at each site determines new directions of motion for the particles, based on momentum and energy-conserving collision rules. This definition of a time step prevents any two particles from moving in the same direction at any time step. Thus the six directions a, b, c, d, e, f (cf. Fig. 1) each have either one particle moving in that direction or none. The six directions can therefore be represented by binary variables, which denote the presence or absence of a particle, and the particle configuration at any site is represented, at a given time, by a 6-digit binary number (f, e, d, c, b, a).

A lattice of N sites can thus be represented by $6N$ binary variables and the evolution in time of the physical model is given by repeated local updating of these $6N$ variables. In the subsequent section, we describe in detail how the updating takes place for a more complex situation than the one given above: namely, we consider flow around a linear obstacle, represented as a contiguous set of lattice sites that form a straight segment. Now each site is described by seven binary variables, the seventh one specifying whether the site is part of an obstacle or not. If a particle hits an obstacle site, it bounces back reversing its direction. The physical, macroscopic situation, thus modelled (see Fig. 2) corresponds to flow in a channel around an obstacle. The flow far from the obstacle is parallel to the x direction. In Fig. 2 the obstacle is placed perpendicularly to the flow, i.e., along the y direction. This corresponds to the West–East direction of the lattice in Fig. 1.

III. MULTI-SPIN CODING FOR THE LATTICE GAS ALGORITHM

A sketch of the system is given in Fig. 3. The lattice points lie on L columns and LL rows and are numbered from $(1, 1)$ to (LL, L) in the usual notation, as indicated in the figure. Also represented there are the point (i, j) and its neighboring sites. The x–y system of axes is also shown here; the direction of the physical flow is along x. As mentioned, the six binary variables at site (i, j) are denoted by $a(i, j)$,

$b(i, j)$, $c(i, j)$, $d(i, j)$, $e(i, j)$, $f(i, j)$, where a value of 1 means that at site (i, j) a particle is moving in the indicated direction, a value of 0 means that no particle is present at site (i, j) which moves in that direction. At each site there is, moreover, an obstacle bit $O(i, j)$ such that

$$\begin{aligned} &O(i, j) = 0 \text{ means there is an obstacle at site } (i, j)\\ &O(i, j) = 1 \text{ means there is no obstacle at site } (i, j). \end{aligned} \tag{1}$$

We will now describe the basic operations of the lattice gas algorithm which change configurations at the sites of the triangular grid. Boolean operations between 2 binary variables are represented symbolically as

$$\begin{aligned} \wedge &= \text{XOR}\\ \& &= \text{AND}\\ | &= \text{OR}\\ \sim &= \text{NOT}. \end{aligned} \tag{2}$$

How does the multi-spin coding proceed? Take the same binary variable, say a, at 32 successive sites, along a given row, for example. Put these variables into one 32-bit long word. Then every operation involving a is done on 32 binary variables a at 32 successive sites simultaneously. The same is done for b, c, d, e, f, O. Then every configuration change at those 32 sites can be implemented through boolean operations on seven 32-bit words, and thus done in parallel. The gain in storage and in time compared to any other implementation is obvious.

Let us now discuss, in turn, collisions and translations of particles.

(i) *Collisions.* Collisions considered are either of the 3-body, 2-body, or 4-body type. The last two can be handled together.

(a) *3-body collisions.* At a given lattice site a 3-body collision corresponds to the following (cf. Fig. 1): if particles arrive from a, c, e directions (no particles from b, d, f directions), they scatter into b, d, f directions (no particles in a, c, e directions). In terms of the binary representation of lattice site states this consists in going from state (01 01 01) to state (10 10 10) or vice versa. The computation at each site consists in determining whether the current configuration is either of these two and, if so, switch to the other one. The key to the spin-coding scheme is the replacement of all conditional (IF–THEN) operations by boolean operations. The computation for a set of lattice points can then be performed simultaneously at the word level using such operations. An auxiliary binary variable u_3 is computed:

$$u_3 = ((a \wedge b) \& (c \wedge d) \& (e \wedge f)) \& (\sim((a \wedge c) \mid (c \wedge e))),$$

where the letters a, b, c, d, e, f represent each a word of 32 binary variables of the same name operated on simultaneously. $u_3 = 1$ for a 3-body collision configuration and 0 otherwise. The action taken, once the relevant 3-body configuration is detected, is to selectively replace a by $\sim a$, b by $\sim b$, c by $\sim c$, etc. It describes the passage from (10 10 10) to (01 01 01) and vice versa.

It can be implemented by the XOR operation

$$\begin{aligned} u_3 \wedge a = {}&\sim a && \text{if } u_3 = 1 \\ & a && \text{if } u_3 = 0. \end{aligned}$$

(b) *2-body and 4-body collisions.* Here the rules are the following: if 2 particles collide along b and e (cf. Fig. 1) (and no other particles present) they scatter into a and d, if they collide along c and f they scatter into b and e and if they collide along a and d they scatter into c and f.

One such possible collision configuration is (010 010) and the resulting configuration after collision is obtained by right-shifting this configuration by one unit. Similarly other possible 2-body collisions are:

$$(010\,010) \rightarrow (001\,001) \rightarrow (100\,100) \rightarrow (010\,010)$$

For 4-body scattering, the applicable configurations are the complementary ones to the 2-body ones. For example, the possible 4-body collision (101 101) is complementary to the 2-body one (010 010). The transformation rules remain the same, namely, right-shifting the configuration by one unit [(101 101) → (110 110)].

This collision description for 2- and 4-body scattering is not quite complete since each final configuration could equally well be obtained by left-shifting rather than right-shifting. This is not taken into account here but could be implemented easily by alternating between left and right shifts on even and odd time updates.

A 2- or 4-body collision configuration is detected through computing an auxiliary variable u_{24}:

$$u_{24} = ((a \wedge b) \mid (b \wedge c)) \,\&\, (\sim((a \wedge d) \mid (b \wedge e) \mid (c \wedge f)))$$

which is equal to 1 for a 2- or 4-body collision configuration and 0 otherwise.

(c) *Obstacle.* This is a special case of collisions, in the sense that if a site is part of an obstacle (obstacle bit = 0 at this site) then any particle hitting this site bounces back, reversing its direction. In terms of binary variables this consists in interchanging a and d, b and e, c and f.

If no collision configuration of type (a), (b), or (c) is encountered at a given site, no action is taken: particles move on with their direction unchanged, i.e., the binary configuration at the site remains the same. The results pertaining to the collision case, with or without an obstacle, can be summarized in one single formula for the changes occurring in each word a, b, c, d, e, f,. This formula is the following, given here for completeness:

$$a \leftarrow (\sim O \,\&\, d) \mid (O \,\&\, ((u_{24} \,\&\, b) \mid (\sim u_{24} \,\&\, (u_3 \;\hat{}\; a))))$$
$$b \leftarrow (\sim O \,\&\, e) \mid (O \,\&\, ((u_{24} \,\&\, c) \mid (\sim u_{24} \,\&\, (u_3 \;\hat{}\; b))))$$
$$c \leftarrow (\sim O \,\&\, f) \mid (O \,\&\, ((u_{24} \,\&\, d) \mid (\sim u_{24} \,\&\, (u_3 \;\hat{}\; c))))$$
$$d \leftarrow (\sim O \,\&\, a) \mid (O \,\&\, ((u_{24} \,\&\, e) \mid (\sim u_{24} \,\&\, (u_3 \;\hat{}\; d))))$$
$$e \leftarrow (\sim O \,\&\, b) \mid (O \,\&\, ((u_{24} \,\&\, f) \mid (\sim u_{24} \,\&\, (u_3 \;\hat{}\; e))))$$
$$f \leftarrow (\sim O \,\&\, c) \mid (O \,\&\, ((u_{24} \,\&\, a) \mid (\sim u_{24} \,\&\, (u_3 \;\hat{}\; f)))).$$

(ii) *Translations.* After collisions have taken place, translation of the particles by one lattice unit in the direction where their velocity points, completes a time step. In terms of multi-spin coding this means essentially that entire words are shifted by one bit, the last bit in one word replacing the first one in the next word.

Boundary conditions imposed on the flow are such that the flow is periodic in all directions. Clearly the lengths L and LL are multiples of a word length in bits, which here is 32.

The rules for translations are then the following: Denote by $a(i, *)$ all the words representing particles moving East in row i of our system (cf. Fig. 3). The rules are

1. a [east direction]
   $a(i, *) \leftarrow$ right-rotate $(a(i, *))$
2. d [west direction]
   $d(i, *) \leftarrow$ left-rotate $(d(i, *))$.

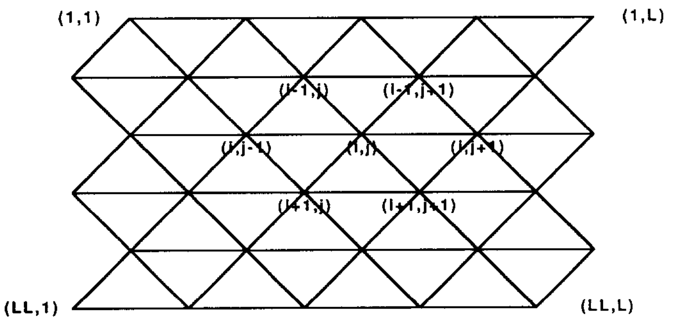

FIG. 3. Schematic drawing of a part of our triangular lattice system, where site numbered (i, j) and 6 neighboring sites are shown. The system has L columns and LL rows and sites are consequently numbered from $(1, 1)$ to (LL, L), as indicated.

For a and d the entire rows are right- or left-shifted. Thus the last bit in one word takes the place of the first bit in an adjacent word and vice versa. The first and last word in a given row are connected this same way, as is required by periodic boundary conditions (cf. Fig. 3).

```
3. $b$ [northeast direction]
   if $i$ is odd then
   $b(i, *) \leftarrow b(i+1, *)$
   else $b(i, *) \leftarrow$ right-rotate $(b(i+1, *))$
4. $c$ [northwest direction]
   if $i$ is odd then
   $c(i, *) \leftarrow$ left-rotate $(c(i+1, *))$
   else $c(i, *) \leftarrow c(i+1, *)$
5. $e$ [southwest direction]
   if $i$ is odd then
   $e(i, 1*) \leftarrow$ left-rotate $(e(i-1), *))$
   else $e(i, *) \leftarrow e(i-1, *)$
6. $f$ [southeast direction]
   if $i$ is odd then
   $f(i, *) \leftarrow f(i-1, *)$
   else $f(i, *) \leftarrow$ right-rotate $(f(i-1, *))$.
```

This completes the set of binary rules for updating the system. Care must be taken to impose periodic boundary conditions for all translations. Whereas collision rules consist in rearranging words a, b, c, d, e, f, locally, at a site, translation rules involve neighboring sites and left or right bit shifts between successive words.

IV. Performance and Comparisons

The system is 2-dimensional, enclosed in a rectangle of length LL and width L. LL denotes the number of rows of the triangular lattice, L the number of columns (cf. Fig. 3). The flow that is mimicked corresponds to flow along the long dimension of the rectangle, around an obstacle perpendicular to this direction placed in the middle. The initial random distribution of particles (with an average number of 2 particles per site) is such that there is a net velocity. Boundary conditions are periodic. We have checked that the simulation of the flow reproduces known results [4]. We are interested here not in the physics of the flow, but in the performance of the binary algorithm which describes the microscopic lattice gas dynamics.

The technique of implementing the binary algorithm on "real artithmetic" machines is through multi-spin coding, described in the previous section. Let us also recall that the algorithm involves two basic sets of operations, one associated with collisions, the other one with translations. The number of boolean operations for collisions is the following: there are 10 each for calculating the intermediate binary variables u_3 and u_{24} (see Section III), 9 operations for updating each of the six directions, which makes a total of 74 binary operations. Translations involve only right- or left-rotation operations on words.

We will first discuss performance of the algorithm on the INTEL hypercube and on an ALLIANT Fx-8. We will later make comparisons with simulations on other machines and draw some conclusions.

(a) *Performance on the 32-processor INTEL hypercube and an ALLIANT.* The lattice gas algorithm has been implemented on a 32-processor INTEL iPSC hypercube. A system of 4096×2048 sites was simulated, represented as a 2-dimensional array of 4096×64 32-bit words. This array was partitioned 32 ways into strips of 128×64 words each. A subset of the hypercube interconnection scheme between processors is used to create a ring of 32 processors. This purely local communication between processors is required to implement a translation step. The partitioning was done by slicing rows and not columns in order to minimize the number of words to be communicated between processors.

Each update of the 4096×2048 systems takes 3592 ms, of which 3565 ms is computation time and 27 ms is interprocessor communication time. The parallel implementation thus achieves an efficiency $\varepsilon(\varepsilon = 1/(1 + t_{\text{com}}/t_{\text{cal}}))$ of 0.99, and a site update rate of 2.3×10^6 sites per second.

The algorithm was also implemented on the ALLIANT Fx-8 shared-memory architecture, where each processor is, moreover, capable of speedup through vectorization. So far, only the collision part of the update computation has been completely vectorized and parallelized. The system size considered is large, 4096×4096 sites. With no vectorization and no concurrency the collision part of the algorithm takes 35,766 ms for one system update. With vectorization this drops to 9950 ms, corresponding to a speedup of 3.6. With vectorization and full concurrency, the collision part now takes only 1613 ms, corresponding to a further speedup of 6.2. The total speedup on the collision part, using vectorization and concurrency, is thus a very considerable 22.3. The translation part, which at the moment is neither vectorized nor executed concurrently takes 21,700 ms for a lattice update. It dominates in the vectorized, concurrent mode the update rate of 0.7×10^6 site updates per second. (The corresponding rate on a much smaller system, 512×256, is 1.3 times larger).

(b) *Results on "bit" machines.* The algorithm has been implemented on three "bit" machines:

(1) CAM-6, a general cellular automaton machine built at MIT, on which lattice gas algorithms have been implemented. The rate obtained for a 2-dimensional square lattice realization is 4×10^6 site updates/s for a lattice size of 256×256 [5].

(2) a dedicated processor for the hexagonal lattice gas algorithm, built at ENS (Paris). Here the update rate is 6.5×10^6 site updates/s for system size of 512×256 [6].

(3) the 65,536 processor Connection Machine [3] on which the highest performance has been achieved. Here, for a system of size 4000×4000, the update rate is 10^9 site updates per second. This is certainly the standard against which any performance on any other machine has to be measured.

(c) *Comparison and conclusions.* Let us mention here that the size of the lattice plays a crucial role if one wishes to model real flows. Whereas small sizes are sufficient for the study of simple, laminar flows, large sizes are required for high Reynolds number flow (e.g., turbulent flow), the reason being that the principal means to increase Reynolds number for the lattice gas is by increasing the size.

From our performance evaluation it is clear that the site update rate for the INTEL hypercube (2.3×10^6) is comparable to that obtained in the cellular automaton and dedicated lattice gas processor mentioned above. The advantage of the "real arithmetic" machines considered is that their large memories make it possible to consider lattice systems of large size. (The efficient use of storage in the multi-spin coding technique is essential.)

However the performance on the Connection Machine is much higher, a factor of 500 better than that achieved on the 32-processor INTEL iPSC hypercube. Which way can performance be improved on the latter?

The individual processor of the iPSC system, the INTEL 80286, is not very powerful. It is interesting to extrapolate from our results and estimate the performance possible with more powerful parallel processor systems, such as the CRAY XMP and INTEL iPSC-VX, where the individual processor is a high-performance vector machine.

With 80 word operations during a time step and an update rate of 2.3×10^6/s, the performance achieved in our current implementation corresponds to 5.75×10^6 operations/s, which compared to the 25 MIPS specification for the iPSC hypercube gives a 23% realized performance. Let us assume that a similar fraction of the peak performance can be obtained with more powerful systems.

Consider, for example, the 1280 MIPS, 128 processor INTEL iPSC-VX hypercube. Compared to the 32-node INTEL iPSC hypercube on which the algorithm was implemented, the total computational power is increased by a factor of 51 (1280 MIPS/25 MIPS). For the same lattice size the computational time for one complete update of the system is now going to drop from 3565 ms to 70 ms. The communication time will remain close to 27 ms. The update rate is then given by $4086 \times 2048/97 \times 10^{-3}$, which is 86×10^6 site updates per second. For the 2048 MIPS, 1024 processor *N* Cube-10, the same calculation leads to a site update of 120×10^6 sites/s, assuming communication times comparable to those of the INTEL hypercubes.

These extrapolated performance rates are very respectable for these general purposes, "real arithmetic" machines: they are only a factor of 8 (or 12) smaller than the performance obtained on the Connection Machine [3].

For the more powerful hypercubes the efficiency is lower than the 99% we measured, because the communication time stays constant even though the computational time decreases. However, with increasing power of the machine available, it will be attractive to simulate larger physical systems. The achieved processor utilization will then increase.

One aspect of the lattice gas computation has been neglected in our discussions up to now, because the quoted performance rates do not take it into account. Whereas a simulation for a large system can run up to 10^5 time updates, every so often—every 5000 or 10,000 time steps—average densities and velocities over regions of the lattice must be computed. Such regions involve typically 16×16 or 32×32 lattice sites. For each region three real numbers are computed, corresponding to density and two velocity components. On the hypercube such calculations (which do not involve interprocessor communication) are done on each processor and stored. They take up negligible time, corresponding to 3 to 4 time steps. However, if a more interactive mode is desired, where the simulated flow is displayed graphically as the calculation proceeds, the manner and rapidity at which data are collected from the hypercube processors will become important. The INTEL hypercube is notoriously inadequate in this respect. Therefore an improved hypercube would also need improved input/output performance.

V. Discussion and Outlook

The measured performance results and extrapolations to more powerful systems that already exist, show that word-level vector/concurrent computers are very promising for the implementation of bit-level algorithms (such as the one of lattice gas hydrodynamics) through the use of multi-spin coding techniques. Their performance is quite respectable compared to that of a massively parallel, general purpose "bit" machine, such as the Connection Machine. On the 65,536 processor Connection Machine, the implementation of the algorithm under consideration requires only bit transfers and comparisons [3]. There are no lengthy boolean operations such as those described in Section III. The Connection Machine is therefore *intrinsically* faster.

At the core of the successful implementation of the algorithm on "real arithmetic" machines lies the multi-spin coding technique. This technique has been used recently to obtain impressive results on a Cray-2 for the lattice Ising spin model [7] ($\sim 15 \times 10^3$ binary variables for the largest lattice size, and an update rate of 4.3×10^9 sites/s). The Ising system is less complex as far as boolean operations are concerned than the algorithm considered in this work.

Until now, we have been concerned with a 2-dimensional system only. If the lattice gas algorithm provides a new, powerful way to simulate certain flows, an extension to three dimensions becomes crucial. Such extension is not trivial and, instead of 6, about 20 binary variables per site are now required [8]. The number of possible configurations per site becomes enormous. Broadcasting all of these configurations to all processors during the updating, as is done for the 2-dimensional case on the Connection Machine [3], is not possible. As to the multi-spin coding approach, it can be straightforwardly extended, at least in principle, to the 3-dimensional case. However, the large number of binary variables, and of boolean operations to be performed on them in the collision part of the algorithm, will make the implementation extremely (and maybe forbiddingly) complex. In contrast, the translation part of the algorithm will take comparatively no time, because the computational time for this part is simply proportional to the number of binary variables required at a site. The extension to three dimensions thus presents a challenge and novel ways to handle the lattice gas algorithm way be required [9].

ACKNOWLEDGMENTS

One of us (F. H.) expresses his gratitude to Dr. Thomas E. Williams for his great interest and support. We also thank Argonne National Laboratory for the use of the ALLIANT Fx-8.

REFERENCES

1. U. FRISCH, B. HASSLACHER, AND Y. POMEAU, *Phys. Rev. Lett.* **56**, 1505 (1986).
2. J. HARDY, O. DE PAZZIS AND Y. POMEAU, *Phys. Rev. A* **13**, 1949 (1976); M. CREUTZ, L. JACOBS, AND C. REBBI, *Phys. Rev. Lett.* **42**, 1390 (1980).
3. Thinking Machines Corporation (Cambridge, MA), technical report, april 1986 (unpublished).
4. D. D'HUMIERES, P. LALLEMAND, AND T. SHIMOMURA, Los Alamos Report, April 1986 (unpublished).
5. N. MARGOLUS, T. TOFFOLI, AND G. VICHNIAC, *Phys. Rev. Lett.* **56**, 1694 (1986).
6. Y. POMEAU, Laboratoir de Physique, Ecole Normale Superieure, Paris, France, private communication (1987).
7. J. G. ZABOLITZKY AND J. H. HERRMANN, preprint (1987).
8. D. D'HUMIERES, P. LALLEMAND, AND U. FRISCH, *Europhys. Lett.* **2**, 291 (1986).
9. M. HENON, *Complex Systems* **1**, 475 (1987); J. P. RIVET, *C. R. Acad. Sci. Paris II* **305**, 751 (1987).