High-Performance Multiprocessor Architecture for a 3-D Lattice Gas Model [1]

F. Lee, M. Flynn and M. Morf
Computer Systems Laboratory
Stanford University, Stanford, CA 94305

Abstract- **The lattice gas method has recently emerged as a promising discrete particle simulation method in areas such as fluid dynamics. We present a very high-performance scalable multiprocessor architecture, called ALGE, proposed for the simulation of a realistic 3-D lattice gas model, Hénon's 24-bit FCHC isometric model. Each of these VLSI processors is as powerful as a CRAY-2 for this application. ALGE is scalable in the sense that it achieves linear speedup for both fixed and increasing problem sizes with more processors.**

The core computation of a lattice gas model consists of many repetitions of two alternating phases: particle *collision* and *propagation*. *Functional decomposition by symmetry group* and *virtual move* are the respective keys to efficient implementation of collision and propagation.

1 Introduction

High performance computing has become a vital enabling force in the conduct of science and engineering research and development. In particular, simulations based on computational fluid dynamics are less costly and much faster than complex wind tunnel tests. In the past few years, the lattice gas method [3] has emerged as an attractive, robust and promising discrete particle simulation method for fluid flow simulations with complicated boundary conditions, that are difficult or impossible to solve with other methods. Various standard fluid dynamical equations, including the Navier-Stokes equations, can be obtained from lattice gas models after proper limits are taken [4].

The core computation of a lattice gas model is inherently suitable for execution on scalable parallel computing systems, without requiring floating point operations. Increasing amounts of computing power is needed to solve large scale simulation problems. It is believed that simple and practical application-specific computers (or co-processors) can achieve performance orders of magnitude higher than existing "general-purpose" supercomputers, that invariably focus on floating point operations. This belief has been confirmed in the case of two-dimensional simulation, but not in the case of three-dimensional simulation, which is much more important and challenging.

All existing special-purpose lattice gas computers such as CAM-6 [12], RAP1, RAP2 [1], and LGM-1 [6], deal with two-dimensional lattice gas models. Until today, only one other design, CAM-8 [10], proposed by Margolus and Toffoli, attempts to deal with three-dimensional models, but this proposal is limited to 16 or fewer state bits per lattice node.

[1]This work was supported by NASA Ames Research Center under contract NAGW 419.

Yet we need to simulate models with 24 bits or more per node in order to achieve realistic results in studying complex phenomena such as turbulent flow [2]. ALGE is to our knowledge the first special-purpose machine proposed to tackle a realistic 3-D lattice gas model.

2 Lattice Gas Models

In order to keep this paper self contained, we repeat some of the material from our previous publication [7], on which this paper is based.

In a lattice gas model, space and time are discretized. Time is divided into a sequence of equal time steps, at which particles reside only at the nodes of the lattice. The evolution consists of two alternating phases: (i) *propagation*: during one time step, each particle moves from one node to another along a link of the lattice according to its velocity; (ii) *collision*: at the end of a time step, particles arriving at a given node collide and instantaneously acquire new velocities. The properties of the lattice not only govern the propagation phase, but also significantly constrain the collision phase, because the *collision rules* must have the same symmetries as the lattice [4].

The *state* of a node can be denoted by the bit vector $\mathbf{b} = (b_1, \ldots, b_n)$, where $b_i = 1$ if a particle with the corresponding velocity $\mathbf{v}^i$ is present [2], and $b_i = 0$ otherwise. Let $\mathbf{b}(\mathbf{x}, t)$, and $\mathbf{b}'(\mathbf{x}, t)$ be the states of the node at position $\mathbf{x}$ and time t before and after the collision respectively. The collision phase specifies that, for all $\mathbf{x}$ and t,

$$\mathbf{b}'(\mathbf{x}, t) = \mathcal{C}(\mathbf{b}(\mathbf{x}, t)) \tag{1}$$

where $\mathcal{C}$ is a deterministic or non-deterministic n-input n-output boolean collision function. The propagation phase specifies that, for all $\mathbf{x}$ and t,

$$b_i(\mathbf{x} + \mathbf{v}^i, t + 1) = b'_i(\mathbf{x}, t) \tag{2}$$

An obstacle such as a plate, a wedge or an airplane wing is decomposed into a series of continuous links which approximate its geometrical shape. At nodes which represent an obstacle, particles are either bounced back or undergo specular reflection. This can be handled by adding one or more obstacle bits to the state of a node and adjusting the collision function appropriately.

Before simulation, the states of the nodes are initialized according to the initial distribution of particle densities and velocities. After simulation, nodes within a volume of tens of nodes on each side are averaged to compute the macroscopic density and momentum.

There are two types of boundary conditions on the lattice edges we are concerned with. The first type is the *periodic boundary condition*: the particles exiting from one edge are reinjected into the other edge in the same direction. The second type, related to a wind-tunnel experiment, consists in providing a flux of fresh particles on one side of the lattice and allowing an output flux on the other side. In this paper, we focus on the first type of

[2] In this paper, Roman and Greek indices refer respectively to labels and components.

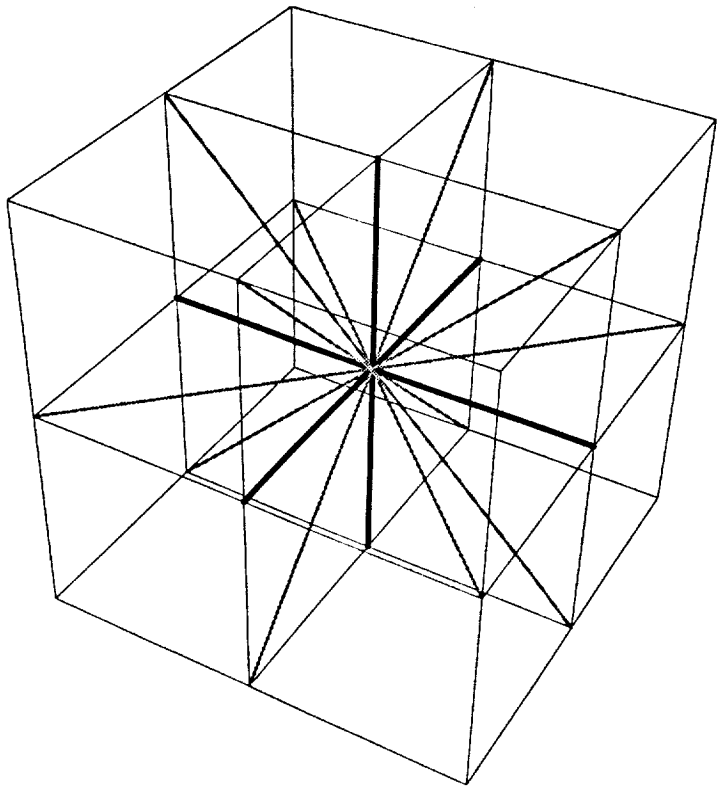

Figure 1: The pseudo-four-dimensional FCHC model. Only the neighbors of one node are shown as connected.

boundary condition, because it is basic: it requires no special treatment for nodes on the *edges*, as there are no edges in a wraparound lattice space. The second type can be dealt with as a simple extension.

2.1 Three-Dimensional Lattice

The particular lattice we are most interested in is the FCHC lattice used in three dimensional simulations [4,11]. A FCHC (face-centered hypercubic) lattice consists of those nodes, which are the points with signed integer coordinates $(x_1, x_2, x_3, x_4) = \mathbf{x}$ such that the sum $x_1 + x_2 + x_3 + x_4$ is even. Each node $\mathbf{x}$ is linked to its 24 nearest neighbors $\mathbf{x}'$ such that the vector $\mathbf{x}' - \mathbf{x}$ corresponds to one of the following 24 values:

$$\begin{aligned} &(\pm 1, \pm 1, 0, 0), \quad (\pm 1, 0, \pm 1, 0), \quad (\pm 1, 0, 0, \pm 1), \\ &(0, \pm 1, \pm 1, 0), \quad (0, \pm 1, 0, \pm 1), \quad (0, 0, \pm 1, \pm 1). \end{aligned} \tag{3}$$

These 24 nearest neighbors form a regular polytope. With time steps normalized to 1, the vectors in (3) are also the 24 possible velocities of the particles arriving at or leaving from a node.

The pseudo four-dimensional FCHC model is derived by projecting the four-dimensional FCHC lattice to three dimension so that the fourth dimension has a periodicity of 1. Each node of a regular cubic lattice is a node in the model. Figure 1 shows the neighborhood of a node: along the gray links, connecting to 12 neighbors, at most one particle can propagate, with component $v_4 = 0$; along the thick black links, connecting to 6 neighbors, up to two particles can propagate, with components $v_4 = \pm 1$.

2.2 Isometric Collision Rules

Associated with the FCHC lattice is the *isometry group G* of order 1152. Roughly speaking, an isometry is a symmetry operation such as rotation and reflection about the origin.

The isometric collision rules [5] require that

1. Every collision is an isometry: the output velocities are images of the input velocities in an isometry.

2. The isometry depends on the momentum only: the momentum of the input state is computed, and then normalized by taking advantage of the symmetries, and finally used for classification.

3. The isometry is randomly chosen among all optimal isometries: this is why non-determinism comes into play. (An optimal isometry is one which minimizes the viscosity of the lattice gas, so that higher Reynolds numbers can be reached.)

3 System Overview

This is an updated version of the design of ALGE as presented in [7]. The machine is organized as an array processor, which serves as a special purpose high performance computing engine to a host computer. The host computer downloads the problem (data) into the engine and offloads the engine-produced solution. The host provides the user interface to the computing engine and performs the pre-processing and post-processing phases of the simulation.

Figure 2 shows a 4x4 configuration of ALGE. The processors are connected as the nodes of a 2-D toroid. Each identical processor [3] (P) has its own local memory (M). In a simulation the 3-D problem space is decomposed into non-overlapped equal-sized partitions such that nodes with the same Z-coordinates map to the same memory space, and adjacent partitions map to adjacent memory spaces.

4 Processor Architecture

Figure 3 shows the functional block diagram of the processor. The processor contains the following units: several collision units, an address generator, a transposer, a switch, a

[3] It may contain several processing elements (PE) as referred in [7].

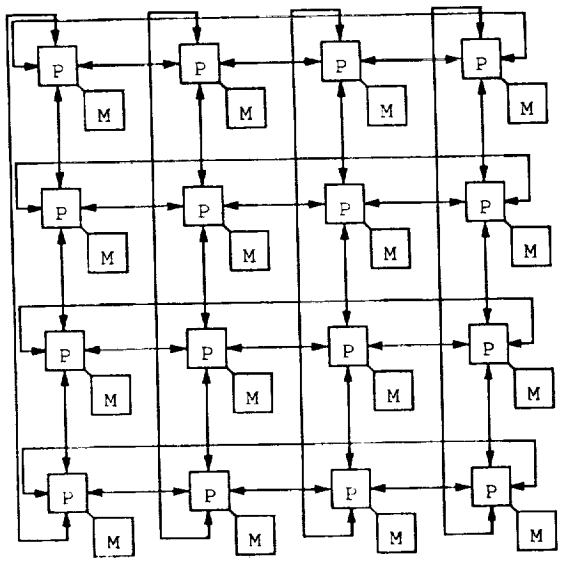

Figure 2: A 4x4 configuration of ALGE

memory address register (MAR), a memory data register (MDR) and a control.

The collision units can be viewed as the "arithmetic" units of a "superscalar" processor. Each unit is capable of computing one collision function per cycle. The address generator contains a register file and some modified adders. It is responsible for generating the proper address sequences for reading and writing data from and to memory. Each local memory can supply one word of k bits per cycle. The n bits of any given node is stored at a different word address. Hence, it takes n cycles to read all n bits of each of the k nodes. The transposer is a two-way shift register array. Actually, there are two transposing buffers so that one can be emptied (written back to memory) and filled (read from memory), while the other is accessed by the collision units. The switch exchanges data with neighboring processors if necessary. At any time, the processor either reads or writes. Since the procedure is deterministic, and the access sequence is data independent, all operations (AG, RD, etc.) are deeply pipelined in order to achieve maximum throughput.

The parameter k is is the number of partitions mapped to a processor. The optimal choice of k depends on n, the number of bits per node, the delay through the switch, and the number of I/O pins and area of the VLSI implementation. Some typical numbers we are considering are: $n = 25$ (1 obstacle bit), $k = 192$ for a processor with 4 collision units.

4.1 Collision Unit

The properties of a lattice not only govern the propagation phase, but also significantly constrain the collision phase, because the *collision rules* must have the same symmetry as the lattice [4]. How the underlying symmetry group of a lattice gas model can be exploited to derive compact and high performance processing elements to handle collision functions of potentially exponential complexities ($O(n2^n)$) was posed as a major challenge in this area of research (see the Preface of [3]). The FCHC isometric model proposed by Hénon [5] was

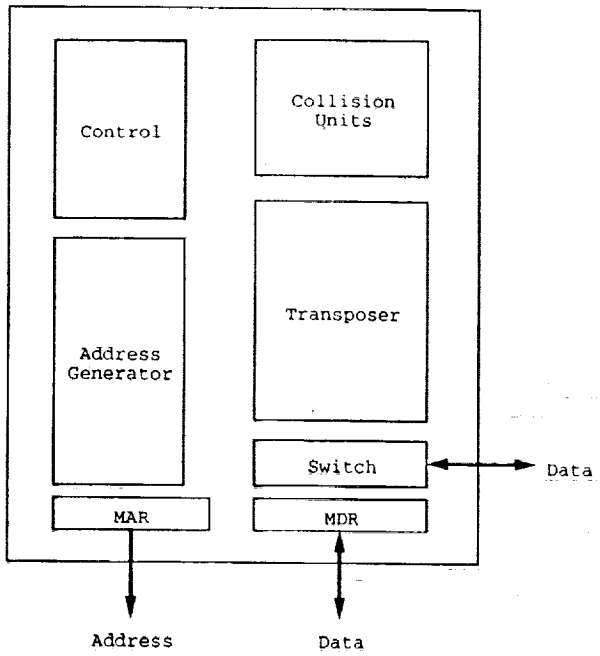

Figure 3: Functional block diagram of the processor

the first real 24-bit three-dimensional model with a detailed specification of an optimized non-deterministic collision function. Therefore, it was chosen as our first case study. A VLSI architecture for the FCHC isometric model has been designed and implemented as an ASIC. We have shown that a 4000 gate chip can replace the equivalent of 4.5 billion bits (rather than 384 million bits due to non-determinism) of a lookup table used to solve this problem. Because the architecture is derived by considering the symmetry properties rather than by brute force logic synthesis, it can be generalized to other classes of lattice gas models. We present the main ideas in this paper. (Please see [8,9] for more details).

Hénon's isometric algorithm [5] shows how the output state of a node is computed as a non-deterministic function of the input state:

1. Compute the momentum of the input state
2. Normalization: Apply the appropriate isometries (symmetry transformations) to the input state and the momentum, so that the momentum is *normalized*.
3. Collision: Choose at random one of the optimal isometries of the class to which the normalized momentum belongs, and apply this isometry.
4. Denormalization: Apply the isometries applied in step 2 in reverse order to obtain the output state.

The application of an isometry to a state is the most frequent and important operation. An efficient implementation of this operation is thus most crucial. Cayley's theorem states that every group is isomorphic to a permutation group, hence it is not too surprising that conditional application of isometries can be implemented as conditional permutations, which in turn map to simple multiplexers. In essence, the algorithm can be viewed as a description of how to generate the right control signals to permute the input state bits.

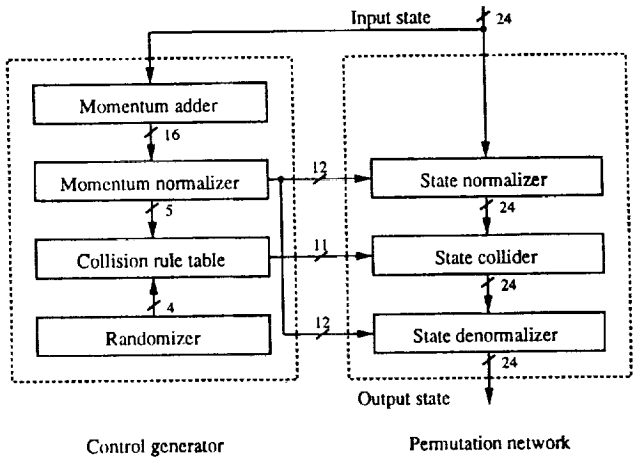

Figure 4: A collision unit

The organization of a collision unit (Figure 4) follows classical lines: the *control path* consisting of a momentum adder, a momentum normalizer, a small collision rule table, and a randomizer; the *data path* is a conditional permutation network composed of a state normalizer, a state collider and a denormalizer (inverse-normalizer). The overall feed-forward character of this unit makes it easy to design a *highly pipelined* version with a proportional increase in throughput.

A CMOS field programmable gate array implementation of the unit with a non-pipelined latency of 460 ns has been completed. A CMOS gate array implementation is estimated to have a non-pipelined latency below 50 ns. A collision unit capable of 20 million node updates per second (MNUPS) or more is clearly feasible. This is comparable to CRAY-2's performance of approximately 30 MNUPS [11].

4.2 Address Generator

As large simulation problems require the use of a huge amount of memory, memory chips can easily become the dominant cost factor of the system. Our solution avoids the common but expensive alternative of double buffering the complete memory space, while retaining a high degree of flexibility in the choice of lengths of each dimension of the (simulation) problem space. This is made possible by the *virtual move* addressing mechanism, which exploits the true data dependency of the computation steps involved. Data movement implied by propagation but not by communication requirements can thus be eliminated.

The address generator contains a number of registers and an arithmetic datapath (adder) to generate the complex sequence required.

4.2.1 Virtual Move

Although the FCHC models are our major concerns, the mechanisms described below apply to a larger class of models with other possible velocities. The following section is written with general notations so as to be valid for any D-dimensional space and arbitrary velocity.

The propagation equation (2) seems to suggest that at every time step, state bits of all the nodes have to be moved. However, a closer examination reveals that the equation actually represents an invariant relationship. If we choose to observe in a *frame of reference* moving at the velocity $\mathbf{v}^i$ with respect to the *rest* frame of the lattice, the particles with velocity $\mathbf{v}^i$ are obviously stationary! Hence, there is no need to actually *move* the bits in memory, as long as we keep track of the *Galilean transformation*. The coordinate $\mathbf{x}^i$ of the moving frame is related to the rest coordinate $\mathbf{x}$ by the transformation:

$$\mathbf{x}^i = \mathbf{x} - \mathbf{v}^i t \tag{4}$$

Suppose the space-time point $(\mathbf{x}, t)$ corresponds to $(\mathbf{x}^i, t)$, then the point $(\mathbf{x} + \mathbf{v}^i, t + 1)$ corresponds to $((\mathbf{x} + \mathbf{v}^i) - \mathbf{v}^i(t + 1), t + 1) = (\mathbf{x} - \mathbf{v}^i t, t + 1) = (\mathbf{x}^i, t + 1)$. Hence, (1) and (2) can be written as

$$\mathbf{b}'(\mathbf{x}^i, t) = \mathcal{C}(\mathbf{b}(\mathbf{x}^i, t)) \tag{5}$$

$$b_i(\mathbf{x}^i, t + 1) = b'_i(\mathbf{x}^i, t) \tag{6}$$

If we interpret $\mathbf{x}^i$ as the *physical address* used to address memory module i, then $\mathbf{x}$ can be treated as the *virtual address*. Equation (6) says that we do not have to move the bits at all in the propagation phase. We refer to this technique as *virtual move*. The cost of implementing virtual move is to have a slightly more complicated address generation scheme. For each virtual address $\mathbf{x}$, we need to generate n physical addresses, $\mathbf{x}^i\,(i = 1, \ldots, n)$ according to (4). However, only one address has to be generated per cycle, if we access one bit plane at a time. Multiple bit planes can be stored in the same memory space by interleaving.

4.2.2 Multi-dimensional Modulo Adder

The transformation (4) requires D modulo subtractions for each $\mathbf{v}^i$. How can one proceed to implement the address generators in hardware?

Suppose we have a wrap-around lattice space of dimension D, implied by the basic type of periodic boundary condition (see section 2). Equation (4) can be written as

$$\begin{aligned} x^i_\alpha &= (x_\alpha - v^i_\alpha t) \bmod n_\alpha \quad (\alpha = 1, \ldots, D) \\ &= (x_\alpha + (-v^i_\alpha t \bmod n_\alpha)) \bmod n_\alpha \\ &= (x_\alpha + d^i_\alpha(t)) \bmod n_\alpha \end{aligned} \tag{7}$$

where n_α is the length of x_α-dimension, and

$$d^i_\alpha(t) = -v^i_\alpha t \bmod n_\alpha \tag{8}$$

Note that d^i_α has to be recomputed only once per time step by addition:

$$d^i_\alpha(t+1) = (d^i_\alpha(t) + (-v^i_\alpha \bmod n_\alpha)) \bmod n_\alpha \tag{9}$$

In order to use conventional RAM, we need to map $\mathbf{x}$ ($\mathbf{x}^i$) to a linear address. We choose the conventional one-to-one mapping

$$A : \{0,1,\ldots,n_1\} \times \{0,1,\ldots,n_2\} \times \{0,1,\ldots,n_D-1\} \mapsto \{0,1,\ldots,n_1 n_2 \cdots n_D - 1\}$$

such that

$$A(\mathbf{x}) = A((x_1, x_2, \ldots, x_D)) = x_1 + x_2 n_1 + \ldots + x_D \prod_{\alpha=1}^{D-1} n_\alpha \tag{10}$$

Assume that all n_α's are powers of 2, such that $m_\alpha = \log_2 n_\alpha$ and $\sum_{\alpha=1}^{D} m_\alpha = m$. The mapping A can then be performed trivially by concatenating the binary representations of $\mathbf{x}$ such that $x_{\alpha+1}$ is on the left of x_α. Similarly, we can obtain the linear address of d^i. Let $a = A(\mathbf{x})$, and $b = A(d^i)$, and define e as

$$e_j = \begin{cases} 0 & \text{if } j = \sum_{\kappa=1}^{\alpha} m_\kappa \text{ for some } \alpha \in [0, D-1] \\ 1 & \text{otherwise} \end{cases} \tag{11}$$

The purpose of e is to mark the boundary bits of dimensions so that *carry-out* from lower dimensions would not be propagated to higher dimensions. The value of e does not change during a simulation.

We can calculate all D components of $\mathbf{x}^i$ according to (7) in one step by using a *multi-dimensional modulo adder*, which takes three m-bit inputs, a, b, and e, and computes the sum as $s = A(\mathbf{x}^i)$. The adder can be built according to the new definitions of p_i, propagate, and s_i, sum:

$$p_i = (a_i \vee b_i)\, e_i \tag{12}$$

$$s_i = a_i \oplus b_i \oplus c_i\, e_i \tag{13}$$

and our usual definitions of g_i, generate, and c_i, carry:

$$g_i = a_i\, b_i \tag{14}$$

$$c_0 = 0 \tag{15}$$

$$c_{i+1} = g_i \vee p_i c_i \tag{16}$$

Hence, this modified adder can be implemented in various ways, such as ripple carry adder, carry lookahead adder, or carry select adder, as deemed appropriate for the system requirement and implementation technology.

4.2.3 Example

Let us illustrate the idea by a small example. Suppose we have the 2-D square lattice with $n = 4$ bits per node, and $\mathbf{v}^1 = (1,0)$, $\mathbf{v}^2 = (-1,0)$, $\mathbf{v}^3 = (0,1)$, and $\mathbf{v}^4 = (0,-1)$.

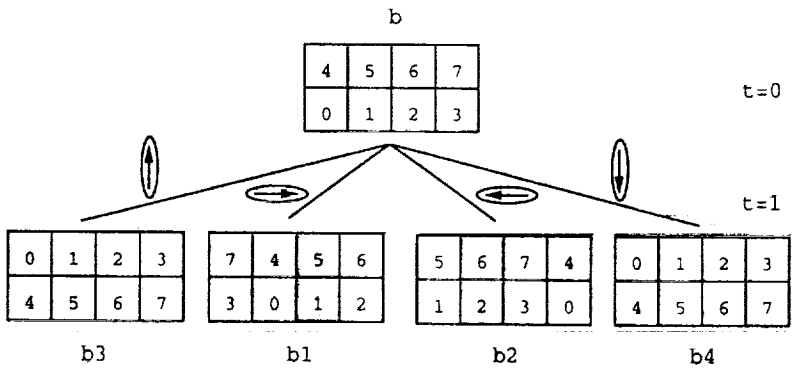

Figure 5: State bit propagation

$\mathbf{x}$	$A(\mathbf{x})$
(0,0)	0
(1,0)	1
(2,0)	2
(3,0)	3
(0,1)	4
(1,1)	5
(2,1)	6
(3,1)	7

Figure 6: Address mapping A

$A(\mathbf{x})$	$A(\mathbf{x}^1)$	$A(\mathbf{x}^2)$	$A(\mathbf{x}^3)$	$A(\mathbf{x}^4)$
0	3	1	4	4
1	0	2	5	5
2	1	3	6	6
3	2	0	7	7
4	7	5	0	0
5	4	6	1	1
6	5	7	2	2
7	6	4	3	3

Figure 7: Virtual to real address translation for $t = 1$

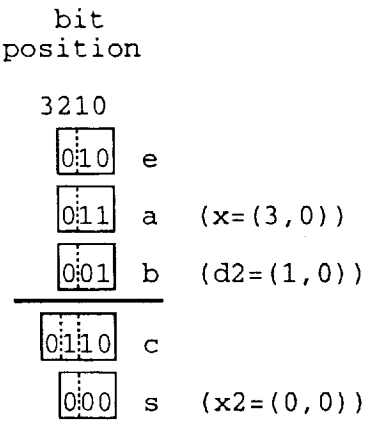

Figure 8: Operation of a multi-dimensional modulo adder.

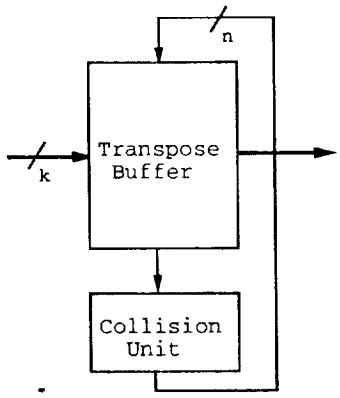

Figure 9: Operation of the transposer

The lattice (problem) space has 8 nodes with $n_1 = 4$ and $n_2 = 2$. In figure 5, each box represents a physical memory location, the linear address mapping of the coordinate of which is given in Figure 6, as calculated by (10). At $t = 0$, the virtual address $\mathbf{x}$ and physical addresses $\mathbf{x}^i$ $(i = 1, 2, 3, 4)$ for any given node are the same. At $t = 1$, they are different, as governed by (4) and shown in Figure 7. The numeric label within each box represents a binary value. Suppose we are interested in the *bit plane* of b_2. The label "3" of the b boxes represents the binary value of b_2 at $A(\mathbf{x}) = 3$ just after collision at $t = 0$. At $t = 1$ just before collision, the label "3" of the b2 boxes is at a different location, because the bit has been moved to the left by one position, where $A(\mathbf{x}) = 2$. However, the move can be avoided if the virtual address 2 is somehow translated into the physical address 3, so that access to $A(\mathbf{x}) = 2$ becomes access to $A(\mathbf{x}^2) = 3$. Figure 8 shows how the translation can be computed by a multi-dimensional modulo adder.

4.3 Transposer

It contains 2 transpose buffers, to be filled and emptied alternately. Figure 9 shows the operation of the transposer. It affects memory addressing, data structure to store the array.

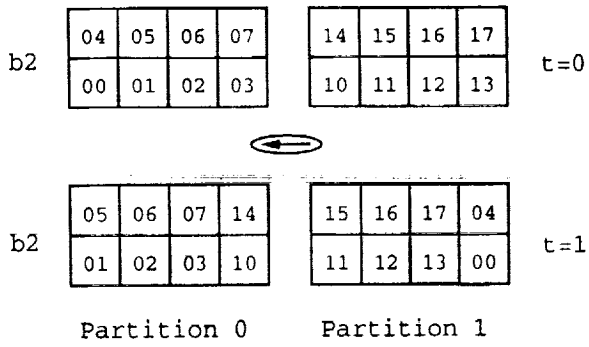

Figure 10: Propagation of bits of b_2 across partitions

During the i-th cycle, the i-th bits of the k words are read and shifted into a transpose buffer. At the end of the n cycles, the n bits of one node are shifted out per cycle. A collision unit takes the bits as input, and the output is written back to the buffer. It acts as a circular shift register. With multiple (u) collision units, multiple updates (collisions) can be executed in parallel per cycle. This update continues for k/u cycles until all k nodes are processed. They are then shifted out bit-by-bit to memory. While one buffer is busy acting as an n-wide circular shift register to serve the collision units, the other can be emptied and refilled just in time to take the turn, if k is chosen appropriately.

4.4 Switch

Updating a node at the border of a partition requires reading values from one or more adjacent partitions. We need to know when to select data bits from which partitions.

According to (2), we know where the neighboring nodes are in the problem space:

$$b_i(\mathbf{x}, t+1) = b'_i(\mathbf{x} - \mathbf{v}^i, t) \tag{17}$$

Let us define three coordinate systems, namely, the *global, partition*, and *local* coordinates such that they satisfy the following relationship:

$$\mathbf{x^G} = \mathbf{P}\mathbf{x^P} + \mathbf{x^L} \tag{18}$$

where $\mathbf{P}$ is a diagonal matrix with $p_{\alpha\alpha} = n_\alpha$, and the following conditions are satisfied:

$$0 \le x^L_\alpha < n_\alpha,\ 0 \le x^P_\alpha < p_\alpha,\ 0 \le x^G_\alpha < n_\alpha p_\alpha \tag{19}$$

Alternatively, we can write for any α

$$x^G_\alpha \bmod n_\alpha p_\alpha = n_\alpha (x^P_\alpha \bmod p_\alpha) + x^L_\alpha \bmod n_\alpha \tag{20}$$

Then we can show that for any α,

$$(x^G_\alpha - v^i_\alpha) \bmod n_\alpha p_\alpha = n_\alpha((x^P_\alpha + \mathrm{bound}(0, x^L_\alpha - v^i_\alpha, n_\alpha)) \bmod p_\alpha) + (x^L_\alpha - v^i_\alpha) \bmod n_\alpha \tag{21}$$

where bound is defined as

$$\text{bound}(L,k,U) = \begin{cases} -1 & \text{if } k < L \\ 0 & \text{if } L \le k < U \\ 1 & \text{if } k \ge U \end{cases} \tag{22}$$

For any given $\mathbf{v}^i$, $\text{bound}(0, x_\alpha^L - v_\alpha^i, n_\alpha)$ is either non-negative or non-positive. Hence, it is only necessary to distinguish whether the value is zero or non-zero.

In equation (21), the partition coordinate determines the source partition, and the local coordinate determines the bit within a partition. Since the machine is synchronous, at any one time, all x_α^L have the same value for all partitions. This exactly matches the requirement implied by (21).

Figure 10 shows an example. It shows the data distribution at $t = 1$ for b_2 for the same example shown in Figure 5. The first digit of a label represents the partition coordinate mapping, while the second one represents the local coordinate mapping.

The function bound can be computed as a *carry-out* of the multi-dimensional modulo adder. Let $a = A(\mathbf{x^L})$, $d_\alpha^i = -v_\alpha^i$, and e as defined before, as the inputs to a multi-dimensional modulo adder, then for each α, $|\text{bound}(0, x_\alpha^L - v_\alpha^i, n_\alpha)|$ is exactly the carry-out bit which is to be blocked so that it will not flow across the dimension boundary. In Figure 8 the carry bit $c_2 = 1$ indicates that the local virtual address 3 of a partition maps to the local physical address 0 of its adjacent partition, as shown in Figure 10.

5 Summary

We have outlined a number of unique architectural features of a very high performance pipelined array processor dedicated to lattice gas simulation. The architecture is truly scalable in the sense that it achieves linear speedup for both fixed and increasing problem sizes with more processors. It is necessary and possible to take advantage of the special properties of the application to design application-specific computers that are a thousand times more powerful than existing supercomputers.

The driving limitation of ALGE is memory bandwidth. This situation becomes more severe as the processing elements run faster and the clock cycle gets shorter. This may be an ideal project for the use of high density mounting and packaging technology such as multiple chip modules.

Current work is focusing on resolving finer issues of design and implementation with the goal of building a prototype system.

The promise of powerful VLSI processors for digital wind tunnels opens up the potential for desk-top and onboard applications.

References

[1] Andre Clouqueur and Dominique d'Humières. R.A.P., A Family of Cellular Automaton Machines for Fluid Dynamics. *Helvetica Physica Acta*, 62:525–541, 1989.

[2] K. Diemer, K. Hunt, S. Chen, T. Shimomura, and G. Doolen. Density and velocity dependence of reynolds numbers for several lattice gas models. *Lattice Gas Methods for Partial Differential Equations*, pages 137–177, 1990.

[3] Gary D. Doolen, editor. *Lattice Gas Methods for Partial Differential Equations*, volume IV of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, 1990.

[4] Uriel Frisch, Dominique d'Humières, Brosl Hasslacher, Pierre Lallemand, Yves Pomeau, and Jean-Pierre Rivet. Lattice Gas Hydrodynamics in Two and Three Dimensions. *Complex Systems*, 1(4):649–707, 1987.

[5] Michel Hénon. Isometric Collision Rules for the Four-Dimensional FCHC Lattice Gas. *Complex Systems*, 1(3):475–494, June 1987.

[6] Steven D. Kugelmass. *Architectures for Two-Dimensional Lattice Computations with Linear Speedup*. PhD thesis, Princeton University, June 1988.

[7] Fung F. Lee and Michael J. Flynn. Architectural Mechanisms to Support Three-Dimensional Lattice Gas Simulations. *Third Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 115–122, July 1991.

[8] Fung F. Lee, Michael J. Flynn, and Martin Morf. A VLSI Architecture for the FCHC Isometric Lattice Gas Model. Technical Report CSL-TR-90-426, Computer Systems Laboratory, Stanford University, April 1990.

[9] Fung F. Lee, Michael J. Flynn, and Martin Morf. Design of Compact High Performance Processing Elements for the FCHC Lattice Gas Models. *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing*, March 1991.

[10] Norman Margolus and Tommaso Toffoli. Cellular Automata Machines. *Lattice Gas Methods for Partial Differential Equations*, pages 219–249, 1990.

[11] Jean-Pierre Rivet, Michel Hénon, Uriel Frisch, and Dominique d'Humières. Simulating Fully Three-Dimensional External Flow by Lattice Gas Methods. *Proceedings of the Workshop on Discrete Kinetic Theory, Lattice Gas Dynamics and Foundations of Hydrodynamics*, pages 276–285, September 1988.

[12] Tommaso Toffoli and Norman Margolus. *Cellular Automata Machines - A New Environment for Modeling*. MIT Press, 1987.