A VLSI Architecture for the FCHC Isometric Lattice Gas Model

Fung F. Lee, Michael J. Flynn, and Martin Morf

Technical Report: CSL-TR-90-426

April 1990

This work was supported by NASA under contract NAG2-248, using facilities supported under NAGW 419.

A VLSI Architecture for the FCHC Isometric Lattice Gas Model

by

Fung F. Lee, Michael J. Flynn, and Martin Morf

Technical Report: CSL-TR-90-426

April 1990

Computer Systems Laboratory

Departments of Electrical Engineering and Computer Science

St anford University

St an ford, California 94305

Abstract

Lattice gas models are cellular automata used for the simulation of fluid dynamics. This paper addresses the design issues of a lattice gas collision rule processor for the four-dimensional FCHC isometric lattice gas model. A novel VLSI architecture based on an optimized version of Hénon's isometric algorithm is proposed. One of the key concepts behind this architecture is the permutation group representation of the isometry group of the lattice.

In contrast to the straightforward table lookup approach which would take 4.5 billion bits to implement this set of collision rules, the size of our processor is only about 5000 gates. With a reasonable number of pipeline stages, the processor can deliver one result per cycle with a cycle time comparable to or less than that of a common commercial DRAM.

Key Words and Phrases: cellular automata, collision rule, computer architecture, fluid dynamics, isometry, lattice gas, permutation group.

Contents

List of Tables

List of Figures

1 Introduction

Lattice gas models are cellular automata used for the simulation of fluid dynamics. (see [I] for an introduction to the subject). A two-dimensional model with the required degree of isotropy to simulate the full Navier-Stokes equations was proposed [2]. The subject is much less advanced in higher dimensions. Though no suitable three-dimensional lattice exists [3], one may use four dimensional models so that three-dimensional problems can then be easily simulated as special cases [l]. A four-dimensional lattice with the required properties has been proposed [3]: the *face-centered-hypercubic (FCHC) lattice* with 24 neighbors.

In two dimensional problems, one can easily implement the collision functions with either the table lookup approach or simple boolean expressions, since they are relatively simple, cheap and fast. However, this solution does not scale up.

The table lookup approach is expensive because it is general purpose. Basically, the size of such a table grows exponentially as the number of input bits. For an n-bit lattice gas model, the table size is at least $n2^n$ bits. For the 6-bit FHP model, the size is 384 bits. For the 24-bit FCHC model, the size is 384 Mbits! Even this high number does not account for the extra hardware required to handle non-deterministic aspects of the collision rules. So lattice gas models seem to lose their appeal when we move to higher dimensional problems.

In order to apply massive parallelism to this problem, we must first efficiently compute a single collision. If a computation primitive is important enough, we build special units to compute it, just as in the cases of integer and floating point adders and multipliers. Unfortunately, current special purpose cellular automata machines such as CAM-6 [4] and RAP-l [5], which use the table lookup approach, are limited to models with 16 or less input bits. How can we build machines which can handle lattice gas models with 24 or more number of bits by taking advantage of the special properties of the collision rules?

Hénon proposed the isometric collision algorithm mainly as a recipe to select collision rules [6]. We propose to actually implement the isometric algorithm in hardware. We will show that an implementation is feasible and cost effective with current technology. Furthermore it is much more effective than the table lookup in terms of area and speed.

We first describe the FCHC isometric lattice gas model from a computational point of view in Section 2. Section 3 introduces the isometric collision algorithm. Implementation issues and possible optimization of the algorithm are discussed in details in Section 4. Also introduced is the permutation group representation of the isometry group, which is a key concept for an efficient hardware implementation of the isometric algorithm. The hardware organization is described in Section 5. Section 6 gives a performance estimation of the proposed architecture in terms of speed and area, and a comparison with the table lookup approach. Finally, this work is summarized and further research opportunities are briefly discussed.

2 FCHC Isometric-Model

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 [l]. An isometric model is a lattice gas model with *isometric collision rules,* which are particularly interesting lattice gas models [6, 3], and they may lead to an efficient implementation.

2.1 FCHC Lattice

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) = x such that the sum $x_1 + x_2 + x_3 + x_4$ is even. Each node x is linked to its 24 nearest neighbors x' such that the vector $\mathbf{x}' - $ x corresponds to one of the following 24 values:

$$\begin{array}{lll} (\pm1,\pm1,0,0), & (\pm1,0,\pm1,0), & (\pm1,0,0,\pm1), \\ (0,\pm1,\pm1,0), & (0,\pm1,0,\pm1), & (0,0,\pm1,\pm1). \end{array} \tag{1}$$

These 24 nearest neighbors form a regular polytope. With time steps normalized to 1, the vectors in (1) are also the 24 possible velocities of the particles arriving at a node or leaving it. Let V be the set of such velocities, which are arbitrarily labeled as $\mathbf{v}_i$, with i = 1,. . . , 24. The *state* of a node can be denoted by the bit vector b = $(b_1, \ldots, b_{24})$, where b_i = 1 if a particle with the corresponding velocity $\mathbf{v}_i$ is present, and b_i = 0 otherwise.

2.2 Isometry Group

Associated with the FCHC lattice is the *isometry group G* of order 1152, which preserves the set of velocities *V.* Roughly speaking, an isometry is a rotation around the origin plus an optional mirror symmetry. More precisely, an isometry can be represented by a matrix M:

$$M = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \tag{2}$$

The group operator is the ordinary matrix multiplication operator. The image of a velocity v in the isometry M is Mv. Particular examples of isometries are

1. S,: the change of sign of one coordinate α, where a = $1,2,3,4$. For example:

$$S_1 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

2. $P_{\alpha\beta}$: the permutation of two coordinates a and β where a, β = $1,2,3,4$ and a $\neq \beta$. For example:

$$P_{12} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4}$$

3. Σ_1, Σ_2: the reflections with respect to the hyperplanes $x_1 + x_4 = x_2 + x_3$ and $x_1 = x_2 +$ x3 $+ x_4$ respectively:

$$\Sigma_1 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & -1 \\ 1 & & 1\text{-}1 & 1 \\ 1\text{-}1 & & 1 & 1 \\ -1 & 1 & 1 & 1 \end{pmatrix} \tag{5}$$

$$\Sigma_2 = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \tag{6}$$

It can be shown [6] that the set of 12 elements:

$$S_1, S_2, S_3, S_4, P_{12}, P_{13}, P_{14}, P_{23}, P_{24}, P_{34}, \Sigma_1, \Sigma_2 \tag{7}$$

form a generating set, and every isometry M can be uniquely expressed as a product of the form

$$M = \begin{pmatrix} I \\ S_4 \end{pmatrix} \begin{pmatrix} I \\ S_3 \end{pmatrix} \begin{pmatrix} I \\ S_2 \end{pmatrix} \begin{pmatrix} I \\ S_1 \end{pmatrix} \begin{pmatrix} I \\ P_{34} \end{pmatrix} \begin{pmatrix} I \\ P_{23} \\ P_{24} \end{pmatrix} \begin{pmatrix} I \\ P_{12} \\ P_{13} \\ P_{14} \end{pmatrix} \begin{pmatrix} I \\ \Sigma_1 \\ \Sigma_2 \end{pmatrix} \tag{8}$$

where, in each parenthesis, one of the factors is to be chosen, and I is the identity matrix.

2.3 Isometric Collision Rules

The isometric collision rules [6] 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: compute the momentum of the input state, and normalize it by taking advantage of the symmetries, and use it 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 Isometric Collision Algorithm

Hénon's isometric algorithm [6] shows how the output state is computed as a non-deterministic function of the input state.

1. Compute the momentum q = (q_1, q_2, q_3, q_4) of the input state: q = $\sum_{i=1}^{24} b_i \mathbf{v}_i$

2. Normalization: Apply the appropriate isometries to the input state and the momentum so that the *normalized momentum* satisfies the following condition:

$$q_1 \geq q_2 \geq q_3 \geq q_4 \geq 0 \text{ and } (q_4 = 0 \text{ or } q_1 + q_4 < q_2 + q_3) \tag{9}$$

   (a) Apply the isometry S_α if $q_\alpha < 0$, for $\alpha = 1, 2, 3, 4$.

   (b) Apply $P_{\alpha\beta}$ ($\alpha \neq \beta$, and $\alpha, \beta = 1, 2, 3, 4$) so that $q_1 \geq q_2 \geq q_3 \geq q_4 \geq o$.

   (c) If $q_4 > 0$ and $q_1 + q_4 = q_2$ t q_3, apply Σ_2. If $q_4 > 0$ and q_1 *t* $q_4 > q_2 + q_3$, apply Σ_1, and then apply S_4 if the new $q_4 < 0$.

3. Collision:

   (a) Determine the class of the normalized momentum according to Table 1.

   (b) Choose at random one of the optimal isometries of that class according to Table 2.

   (c) Apply this isometry.

4. Denormalization: Apply the isometries applied in step 2 in reverse order to obtain the output state.

In order to take advantage of the isometries inherited in the model, it is not necessary to restrict ourselves to the particular form of normalization momentum as defined in (9). However, this form is convenient for mapping to familiar hardware structures.

Table 1: Classes of normalized momenta

Class	Definition
1	$q_1 = q_2 > q_3 > q_4 > 0$
2	$q_1 = q_2 = q_3 > q_4 > 0$
3	$q_1 > q_2 > q_3 > q_4 = 0, \quad q_1 = q_2 + q_3$
4	$q_1 > q_2 > q_3 > q_4 = 0, \quad q_1 \neq q_2 + q_3$
5	$q_1 = q_2 > q_3 > q_4 = 0$
6	$q_1 > q_2 = q_3 > q_4 = 0, \; q_1 = 2q_2$
7	$q_1 > q_2 = q_3 > q_4 = 0, \; q_1 \neq 2q_2$
8	$q_1 = q_2 = q_3 > q_4 = 0$
9	$q_1 > q_2 > q_3 = q_4 = 0$
10	$q_1 = q_2 > q_3 = q_4 = 0$
11	$q_1 > q_2 = q_3 = q_4 = 0$
12	$q_1 = q_2 = q_3 = q_4 = 0$

Table 2: Classes of optimal isometries

Class	Optimal isometries	
1	1	P_{12}
2	2	$P_{23}P_{12}, P_{23}P_{13}$
3	2	$S_4\Sigma_1, S_4\Sigma_2$
4	1	S_4
5	1	S_4P_{12}
6	4	$S_4\Sigma_1, S_4\Sigma_2, S_4P_{23}\Sigma_1, S_4P_{23}\Sigma_2$
7	1	S_4P_{23}
8	4	$P_{23}P_{12}, P_{23}P_{13}, S_4P_{23}P_{12}, S_4P_{23}P_{13}$
9	3	$S_4S_3, S_3P_{34}, S_4P_{34}$
10	6	$S_3P_{34}P_{12}, S_4P_{34}P_{12}, S_4S_3\Sigma_1,$
		$S_4S_3P_{34}P_{12}\Sigma_1, S_4S_3\Sigma_2, P_{34}P_{12}\Sigma_2$
11	6	$S_4S_2P_{23}, S_4S_3P_{23}, S_3S_2P_{24},$
		$S_4S_3P_{24}, S_3S_2P_{34}, S_4S_2P_{34}$
12	12	$S_3S_1P_{34}P_{12}, S_4S_1P_{34}P_{12}, S_3S_2P_{34}P_{12}, S_4S_2P_{34}P_{12}$
		$S_2S_1P_{24}P_{13}, S_4S_1P_{24}P_{13}, S_3S_2P_{24}P_{13}, S_4S_3P_{24}P_{13}$
		$S_2S_1P_{23}P_{14}, S_3S_1P_{23}P_{14}, S_4S_2P_{23}P_{14}, S_4S_3P_{23}P_{14}$

4 Implementation Issues

The algorithm can be viewed as a description of how to generate the right control signals to transform the input state bits.

4.1 Data Transformation

As we see in the isometric collision algorithm, 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.

Let us examine carefully how this operation may be carried out. Suppose **b** and **b'** are the input state and output state respectively, and $\mathbf{b}'$ is deduced by applying the isometry M to **b.** First, we *decode* the actual set of input velocities from **b:** $\{\mathbf{v}_i|b_i = 1, i = 1,\ldots, 24)$. Then we apply the isometry M to each present velocity to compute the set of output velocities: $\{M\mathbf{v}_i|b_i = 1, i = 1,\ldots, 24)$. Finally, we *encode* the output velocities as the output state: $\mathbf{b}' = (b'_1,\ldots, b'_{24})$, where for all j, $b'_j = 1$ if $\mathbf{v}_j = M\mathbf{v}_i$ and $b_i = 1$ for some $i \in \{1,\ldots, 24\}$, and $b'_j = 0$ otherwise. Decoding is simple; matrix multiplication is relatively expensive; encoding may even involve searching or sorting in some implementation. In other words, the operation of applying an isometry to a state seems to be quite expensive. Fortunately, there is a much better way to approach the problem.

Since an isometry M preserves the set of velocities V, that is, $\mathbf{v}_i$ is mapped to $\mathbf{v}_j$, whether $b'_j = 1$ is equivalent to asking whether $b_i = 1$. Hence, applying an isometry to a state vector is equivalent to permuting the state components in a particular order, independent of the actual values of the components. This implies that the output state vector is a permutation of the input state vector. How an isometry is applied to a state vector is strongly related to how the isometry is represented.

4.1.1 Representation of Isometries

This section is written with general notations so as to be valid for any single-speed lattice gas model [1] with its associated isometry group G.

Let V be the set of all n distinct particle velocities. A *velocity labeling function*, f_V, is a bijective function $f_V : V \mapsto \mathrm{N}$, where $\mathrm{N} = \{1, 2, \ldots, n\}$, We write $V = \{\mathbf{v}_j | j \in \mathrm{N}\}$ with the implicit assumption that some f_V has already been chosen.

Let A be a set and consider the set S_A of all bijections f such that $f : A \mapsto A$. The set S_A under function composition, denoted by $[S_A, \circ]$, is called the group of permutations on A. Any subgroup of S_A is called a *permutation group*. Cayley's theorem states that every group is isomorphic to a permutation group [7].

Suppose G_M is the matrix group representation of the isometry group G, so that the image of a velocity v in the isometry M is Mv, where $M \in G_M$. We would like to find a permutation group isomorphic to the isometry group G. In the following, we will derive the corresponding permutation group G_π from the matrix group G_M by construction of an isomorphism.

Let G_π be the range of the function $f_G : G_M \mapsto G_\pi$ defined by $f_G(M) = \pi$ such that for all $i, j \in$ N, $\pi(i) =$ j if $M\mathbf{v}_i = \mathbf{v}_j$, where $\mathbf{v}_i$, $\mathbf{v}_j \in V$. Since an isometry preserves the set of velocities, π is indeed a permutation of N, and hence G_π is a subset of S_n. Moreover, it can be easily verified that G_π is a subgroup of S_n, that is, G_π is a permutation group (see Appendix A).

Since G_π is defined as the range of f_G, f_G is onto by definition. It is also one-to-one because of isometry. Hence, f_G is bijective.

Proposition 1 *For all* $M_1, M_2 \in G_M$, $f_G(M_2 \cdot M_1) = f_G(M_2) \circ f_G(M_1)$.

Proof: For all i $\in$ N, there exist j, $k \in$ N such that $M_1\mathbf{v}_i = \mathbf{v}_j$, and $M_2\mathbf{v}_j = \mathbf{v}_k$. Obviously, we have $(M_2M_1)\mathbf{v}_i = \mathbf{v}_k$, and hence $f_G(M_2 \cdot M_1)(i) = k$. Also we have $(f_G(M_2) \circ f_G(M_1))(i) = f_G(M_2)(f_G(M_1)(i)) = f_G(M_2)(j) = k$. □

Theorem 1 *f_G is an isomorphism from* $[G_M, \cdot]$ *to* $[G_\pi, \circ]$.

Proof: As G_π has already been verified to be a group under function composition, the theorem follows naturally since we have already shown that (a) the function f_G is a bijection and (b) for all $M_1, M_2 \in G_M$, $f_G(M_2 \cdot M_1) = f_G(M_2) \circ f_G(M_1)$ (*see* Proposition 1). □

Although Theorem 1 is valid independent of the particular choice of f_V, the velocity labeling function, the particular permutation functions in G_π do depend on f_V. Table 3 shows some images of f_G for the particular f_V chosen. The permutations in lower cases are the isomorphic images of their respective matrices in upper cases.

4.1.2 Applying Isometries

Recall that N $= \{1, 2, \ldots, n\}$, and $b_i \in \{0, 1\}$. Let $B = \{0, 1\}$. For each permutation π : N $\mapsto$ N in the permutation group G_π, let us define a function $\hat{\pi} : B^n \mapsto B^n$ such that

$$\hat{\pi}[(b_1, b_2, \ldots, b_n)] = (b_{\pi^{-1}(1)}, b_{\pi^{-1}(2)}, \ldots, b_{\pi^{-1}(n)}) \tag{10}$$

We can easily show that $\hat{\pi}$ is a permutation of B^n, and $\widehat{G_\pi}$, the set of such $\hat{\pi}$'s, is a permutation group of B^n under function composition (see Appendix B). We shall call $\widehat{G_\pi}$ the *induced permutation group* of G_π. Note that $|\widehat{G_\pi}| = |G_\pi|$.

Since an isometry M preserves the set of velocities V, that is, $\mathbf{v}_i$ is mapped to $\mathbf{v}_j$, whether $b'_j = 1$ is equivalent to whether $b_i = 1$. As $\pi = f_G(M)$ exactly represents the permutation of the *indices* of a state vector, applying an isometry M to a state vector $\mathbf{b}$ is thus equivalent to evaluating $\hat{\pi}(\mathbf{b})$, where $\pi = f_G(M)$. As a circuit, $\hat{\pi}$ is nothing more than a permutation of the n wires connecting the n input ports to the n output ports.

velocity labeling	Permutation functions												
	i	s_1	s_2	s_3	s_4	p_{12}	p_{13}	p_{14}	p_{23}	p_{24}	p_{34}	σ_1	σ_2
(1,1,0,0)	1	3	2	1	1	1	13	17	5	9	1	1	1
(1,-1,0,0)	2	4	1	2	2	3	15	19	6	10	2	22	21
(-1,1,0,0)	3	1	4	3	3	2	14	18	7	11	3	23	24
(-1,-1,0,0)	4	2	3	4	4	4	16	20	8	12	4	4	4
(1,0,1,0)	5	7	5	6	5	13	5	21	1	5	9	5	5
(1,0,-1,0)	6	8	6	5	6	14	7	23	2	6	10	18	17
(-1,0,1,0)	7	5	7	8	7	15	6	22	3	7	11	19	20
(-1,0,-1,0)	8	6	8	7	8	16	8	24	4	8	12	8	8
(1,0,0,1)	9	11	9	9	10	17	21	9	9	1	5	13	9
(1,0,0,-1)	10	12	10	10	9	18	22	11	10	2	6	10	13
(-1,0,0,1)	11	9	11	11	12	19	23	10	11	3	7	11	16
(-1,0,0,-1)	12	10	12	12	11	20	24	12	12	4	8	16	12
(0,1,1,0)	13	13	15	14	13	5	1	13	13	21	17	9	10
(0,1,-1,0)	14	14	16	13	14	6	3	14	15	23	18	14	14
(0,-1,1,0)	15	15	13	16	15	7	2	15	14	22	19	15	15
(0,-1,-1,0)	16	16	14	15	16	8	4	16	16	24	20	12	11
(0,1,0,1)	17	17	19	17	18	9	17	1	21	17	13	17	6
(0,1,0,-1)	18	18	20	18	17	10	18	3	22	19	14	6	18
(0,-1,0,1)	19	19	17	19	20	11	19	2	23	18	15	7	19
(0,-1,0,-1)	20	20	18	20	19	12	20	4	24	20	16	20	7
(0,0,1,1)	21	21	21	23	22	21	9	5	17	13	21	21	2
(0,0,1,-1)	22	22	22	24	21	22	10	7	18	15	23	2	22
(0,0,-1,1)	23	23	23	21	24	23	11	6	19	14	22	3	23
(0,0,-1,-1)	24	24	24	22	23	24	12	8	20	16	24	24	3

Table 3: Images of N under various permutation functions: $N \mapsto N$

4.1.3 Composition of Isometries

As G_M is isomorphic to G_π according to Theorem 1, π can be uniquely expressed as a composition of the form

$$\pi = \begin{pmatrix} i \\ s_4 \end{pmatrix} \begin{pmatrix} i \\ s_3 \end{pmatrix} \begin{pmatrix} i \\ s_2 \end{pmatrix} \begin{pmatrix} i \\ s_1 \end{pmatrix} \begin{pmatrix} i \\ p_{34} \end{pmatrix} \begin{pmatrix} i \\ p_{23} \\ p_{24} \end{pmatrix} \begin{pmatrix} i \\ p_{12} \\ p_{13} \\ p_{14} \end{pmatrix} \begin{pmatrix} i \\ \sigma_1 \\ \sigma_2 \end{pmatrix} \tag{11}$$

where, in each parenthesis, one of the factors is to be chosen. The factors in lower cases are permutation functions, the respective images of those matrices appeared in (8), and i is the identity permutation.

These generators have some interesting and useful properties. Each of them is an inverse of itself; some commute with each other, for example, $s_1 \circ s_2 = s_2 \circ s_1$. This may be useful when we consider *operator reordering* to reduce the critical path delay. The permutations $s_1, s_2, s_3, s_4, \sigma_1, \sigma_2$ are even, while p_{12}, p_{13}, p_{14}, p_{23}, p_{24}, p_{34} are odd. This is independent of the labeling function. Any one of these generators can be written as the product of disjoint transpositions, i.e., disjoint cycles of length 2. In general, every permutation can be written as the product of disjoint cycles in only one way (where the order of the factors does not matter) [8]. In other words, the cycle form of such a generator is unique.

Equation 11 suggests the use of some kind of multiplexers to specify the particular π to be composed. An implementation of this form may use 2-to-1 multiplexers, 3-to-l multiplexers, or 4-to-l multiplexers. Since the order of the group G is 1152, we need at least [log, 11521 = 11 control points. In fact, a more convenient and efficient composition form of π exists:

$$\begin{aligned}\pi = & \begin{pmatrix} i \\ s_4 \end{pmatrix} \begin{pmatrix} i \\ s_3 \end{pmatrix} \begin{pmatrix} i \\ s_2 \end{pmatrix} \begin{pmatrix} i \\ s_1 \end{pmatrix} \begin{pmatrix} i \\ p_{23} \end{pmatrix} \begin{pmatrix} i \\ p_{24} \end{pmatrix} \\ & \begin{pmatrix} i \\ p_{13} \end{pmatrix} \begin{pmatrix} i \\ p_{34} \end{pmatrix} \begin{pmatrix} i \\ p_{12} \end{pmatrix} \begin{pmatrix} i \\ \sigma_2 \end{pmatrix} \begin{pmatrix} i \\ \sigma_1 \end{pmatrix} \end{aligned} \tag{12}$$

Let $X(\pi, c)$ be a *conditional permutation* defined by

$$X(\pi, c) = \begin{cases} \pi & \text{if } c = 1 \\ i & \text{if } c = 0 \end{cases} \tag{13}$$

We can then rewrite Equation 12 more precisely as

$$\begin{aligned}\pi = & X(s_4, c_{s_4}) \circ X(s_3, c_{s_3}) \circ X(s_2, c_{s_2}) \circ X(s_1, c_{s_1}) \circ X(p_{23}, c_{p_{23}}) \circ X(p_{24}, c_{p_{24}}) \circ \\ & X(p_{13}, c_{p_{13}}) \circ X(p_{34}, c_{p_{34}}) \circ X(p_{12}, c_{p_{12}}) \circ X(\sigma_2, c_{\sigma_2}) \circ X(\sigma_1, c_{\sigma_1}) \end{aligned} \tag{14}$$

This form also requires 11 control signals, but it only uses 2-to-l multiplexers. The particular $p_{\alpha\beta}$'s are chosen because they correspond to a fast parallel momentum sorter used to implement step 2(b) of the isometric algorithm. To be consistent, some entries in Table 2 require modification: for the last 4 optimal isometries of class 12, replace the symbols $P_{23}P_{14}$ by $P_{24}P_{13}P_{34}P_{12}$.

4.2 Control Generation

Since the generation of the control signals has to precede the application of the corresponding isometries, they are in the critical path of the circuit. The general guideline of our design is to generate the control signals as early as possible with the minimum amount of hardware resources. We tend to trade area for speed if *or-parallelism* is useful, that is, if computing results for different cases in parallel helps to reduce the critical path delay. One significant demonstration of such tradeoff is described in the next section.

4.2.1 Sigma Optimization

From Table 1, we observe that the class of the normalized momentum is completely specified by the results of the following five boolean equality tests performed on the momentum components *after* step 2(c) of the isometric collision algorithm: $q_1 = q_2, q_2 = q_3, q_3 = q_4, q_4 = 0$, and $q_1+q_4 = q_2 + q_3$. In other words, computing the actual value of the normalized momentum is a means rather than an end. Can we somehow avoid computing the actual values of the momentum components under various cases in step 2(c)? The idea is to merge steps 2(c) and 3(a) of the isometric collision algorithm. Instead of actually calculating the (final) normalized momenta under different cases, testing the equalities, and then selecting the results conditionally, we have found that we can skip the calculation of the normalized momenta by the following analysis.

By the end of step 2(b) of the isometric collision algorithm, the momentum components are non-negative and *sorted:* $q_1 \geq q_2 \geq q_3 \geq q_4 \geq 0$. There are a total of 5 mutually exclusive cases:

1. $q_4 = 0$: apply I
2. $q_4 > 0$ and q_1 t $q_4 < q_2 + q_3$: apply I
3. $q_4 > 0$ and $q_1 + q_4 = q_2 + q_3$: apply Σ_2
4. $q_4 > 0$ and q_1 t $q_4 > q_2$ t q_3 and $q_1 - q_4 \leq q_2 + q_3$: apply Σ_1
5. $q_4 > 0$ and q_1 t $q_4 > q_2 + q_3$ and $q_1 - q_4 > q_2 + q_3$ apply $S_4\Sigma_1$

It is clear that the final momentum **q'** after application of the corresponding isometry in all 5 cases satisfy condition (9). For example, if case 3 applies, then

$$\mathbf{q}' = \Sigma_2\mathbf{q} = \frac{1}{2}\begin{pmatrix} q_1 \text{ t } q_2 \text{ t } q_3 \text{ t } q_4 \\ q_1 \text{ t } 42 - q_3 - q_4 \\ q_1 - q_2 \text{ t } q_3 - q_4 \\ q_1 - q_2 - q_3 \text{ t } q_4 \end{pmatrix} \tag{15}$$

case	equality tests				
	$q'_1 = q'_2$	$q'_2 = q'_3$	$q'_3 = q'_4$	$q'_4 = 0$	$q'_1 + q'_4 = q'_2$ t q'_3
1	$q_1 = 42$	$q_2 = q_3$	$q_3 = q_4$	true	$q_1 + q_4 = q_2$ t q_3
2	$q_1 = q_2$	$q_2 = q_3$	$q_3 = q_4$	false	false
3	false	$42 = q_3$	$q_3 = q_4$	true	false
4	$43 = q_4$	$q_2 = q_3$	$q_1 = 42$	$q_1 - q_4 = q_2$ t q_3	false
5	$43 = q_4$	$q_2 = q_3$	false	false	false

Table 4: Equality tests under the 5 cases

input		output	
t_1	$q_1 = 42$	e_1	$q'_1 = q'_2$
t_2	$42 = q_3$	e_2	$q'_2 = q'_3$
t_3	$43 = 44$	e_3	$q'_3 = q'_4$
t_4	$q_4 = 0$	e_4	$q'_4 = 0$
t_5	$q_1 + q_4 > q_2 + q_3$	e_5	$q'_1 + q'_4 = q'_2$ t q'_3
t_6	$q_1 + q_4 = q_2$ t q_3	N_Σ_2	case 3 applies
t_7	$q_1 - q_4 > q_2$ t q_3	N_Σ_1	case 4 or 5 applies
t_8	$q_1 - q_4 = q_2$ t q_3	N_S4A	case 5 applies

Table 5: Meaning of Sigma block variables

variable	expression
N_Σ_2	$\overline{t_4}\, t_6$
N_Σ_1	$\overline{t_4}\, t_5$
N_S4A	$\overline{t_4}\, t_5\, t_7$
e_1	$t_1\,(t_4 \vee \overline{t_5}\,\overline{t_6}) \vee t_3\,\overline{t_4}\, t_5$
e_2	t_2
e_3	$t_3\,(t_4 \vee \overline{t_5}) \vee t_1\,\overline{t_4}\, t_5\,\overline{t_7}$
e_4	$t_4 \vee t_6 \vee t_5\,\overline{t_7}\, t_8$
e_5	$t_4\, t_6$

Table 6: Sigma block outputs

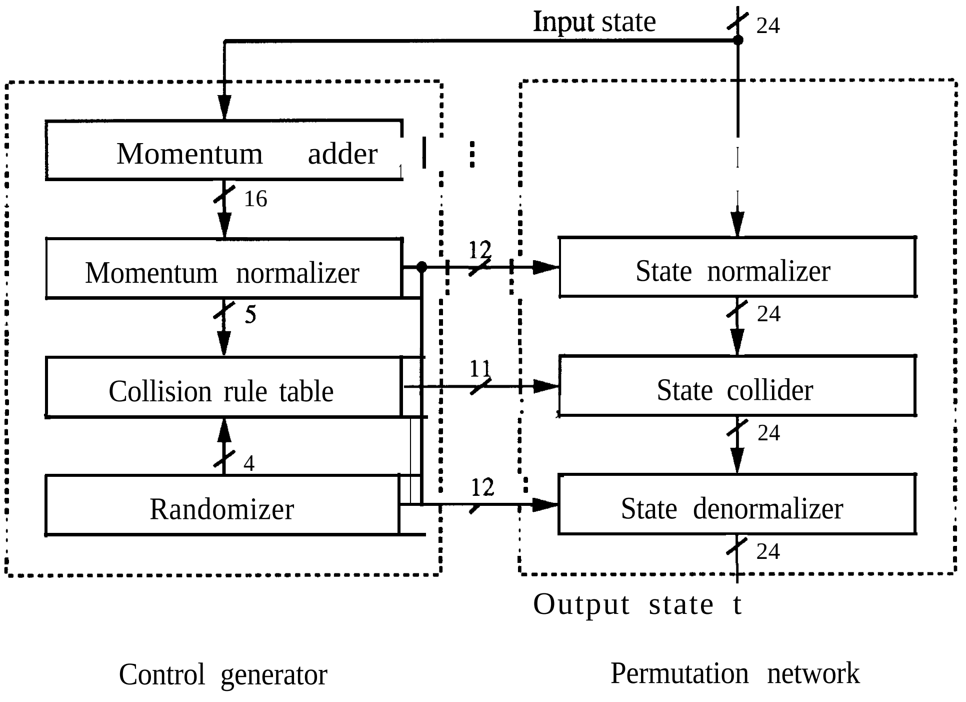

Figure 1: A processor architecture for the FCHC isometric model

As shown in Table 4, performing the equality tests on the final momentum q' is equivalent to performing some other tests on the "sorted momentum" before the possible application of Σ_1 or Σ_2. For example, if case 4 applies, then testing whether $q'_1 = q'_2$ is equivalent to testing whether $q_3 = q_4$. As another example, if case 3 applies, then $q'_4 = 0$ is known to be true. If we make the variable assignment as shown in Table 5, we can express the output variables as in Table 6 (also see Section 5.1.2).

5 Hardware Organization

The processor consists of two parts, namely, the control generator and the permutation network. Figure 1 shows the block diagram of the top level architecture. In this paper, it is described basically as a combinational circuit. However, this structure is easily pipelined to achieve higher throughput. How the pipeline feature can be utilized in a system environment will be discussed in another paper.

5.1 Control Generator

The control generator is composed of 4 functional blocks, namely, the momentum adder, momentum normalizer, collision rule table, and randomizer. It generates all the control signals required to control the settings of the permutation network. It accepts 24 input state bits and generates 23 distinct control signals.

5.1.1 Momentum Adder

The momentum adder computes the four momentum components from the input state bits, as specified in step 1 of the isometric algorithm. Using the same velocity labeling as in Table 3, we have

$$q_1 = b_1 + b_2 - b_3 - b_4 + b_5 + b_6 - b_7 - b_8 + b_9 + b_{10} - b_{11} - b_{12} \tag{16}$$

$$q_2 = b_1 - b_2 + b_3 - b_4 + b_{13} + b_{14} - b_{15} - b_{16} + b_{17} + b_{18} - b_{19} - b_{20} \tag{17}$$

$$q_3 = b_5 - b_6 + b_7 - b_8 + b_{13} - b_{14} + b_{15} - b_{16} + b_{21} + b_{22} - b_{23} - b_{24} \tag{18}$$

$$q_4 = b_9 - b_{10} + b_{11} - b_{12} + b_{17} - b_{18} + b_{19} - b_{20} + b_{21} - b_{22} + b_{23} - b_{24} \tag{19}$$

Note that all operands are 1 bit wide, and there are many common sub-expressions. Such operands may be added by using carry save adders [9].

5.1.2 Momentum Normalizer

Figure 2 shows the complete design of the momentum normalizer in terms of common functional blocks such as adders and comparators. Most of the operands are 3 bit wide, and only a few of them are 4 bit wide. Hence, they are small and fast. The momentum normalizer accepts the 16 momentum bits, generates 12 control signals to drive the state normalizer of the permutation network, and outputs 5 bits to drive the collision rule table. The hardware implements the control decisions made in steps 2 and 3(a) of the isometric algorithm. The 4 blocks in the first level correspond to step 2(a). They generate the four control signals, N_S1,N_S2,N_S3 and N_S4. The 5 sorters in levels 2, 3 and 4 correspond to step 2(b). This structure is chosen because it is the fastest and smallest parallel sorter [10] for 4 numbers. It generates the five control signals, N_P12, N_P34, N_P13, N_P24 and N_P23.

At the output of the fourth level, the momentum components are sorted: $q_1 \geq q_2 \geq q_3 \geq q_4 \geq 0$. The rest of the normalizer implements the Sigma optimization as discussed in Section 4.2.1. The Sigma block generates the last 3 control signals, namely $N\_\Sigma2$, N-Cl and N_S4A, and the 5 bits, e_1, e_2, e_3, e_4, e_5, which encode the class, according to Tables 5 and 6.

5.1.3 Randomizer

Suppose we have a good quality pseudo-random number generator. Since the maximum number of optimal isometries of any one class is 12, the randomizer must have 4 output bits. Note that 12 is divisible by all n_c, where n_c is the number of optimal isometries corresponding to any one class (see Table 2). The random number generator can be realized as a chain of simple linear feedback shift registers. It generates one new random number for each input state vector. It is not in the critical path.

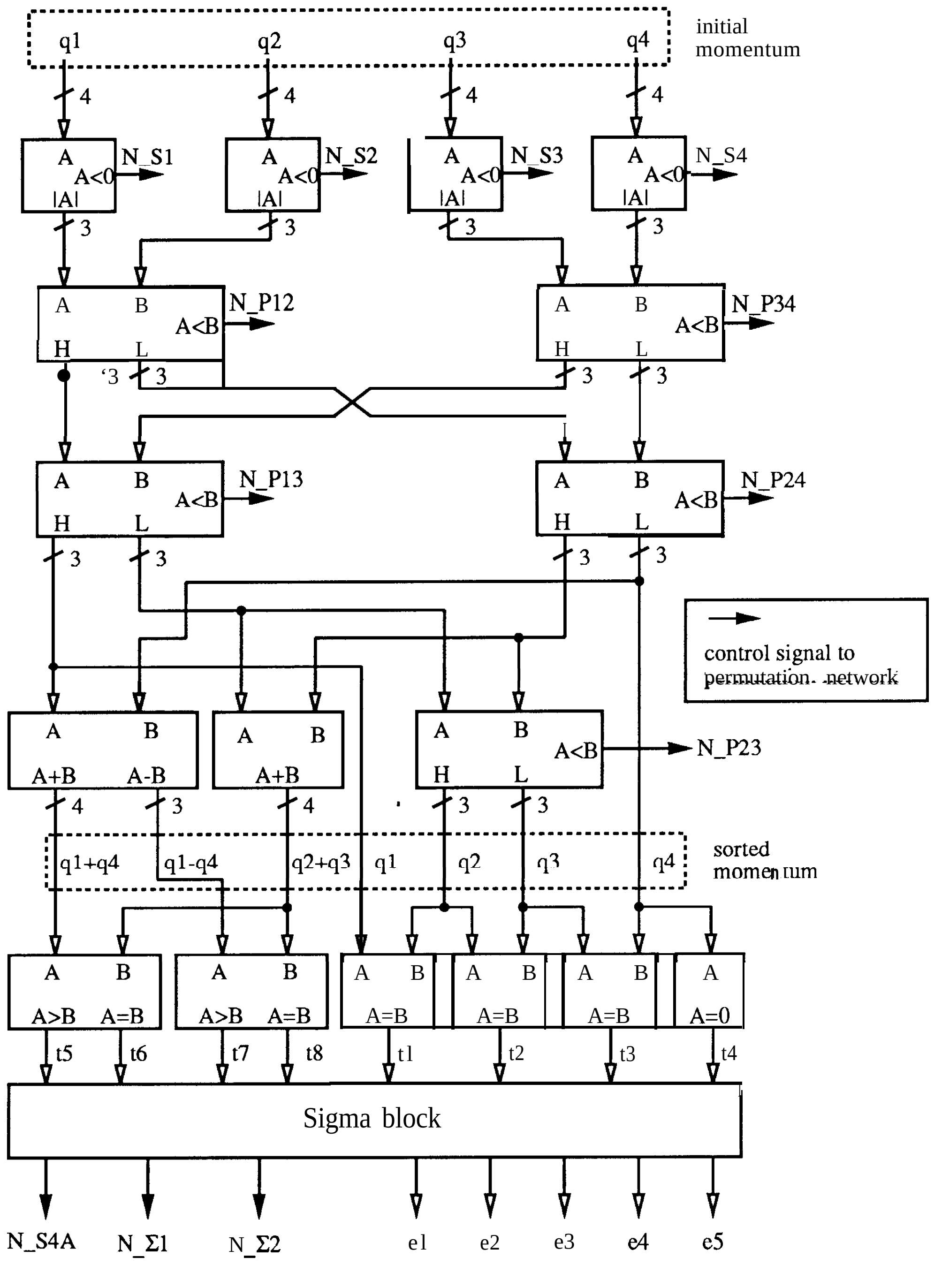

Figure 2: Momentum normalizer

5.1.4 Collision Rule Table

The collision rule table generates the 11 control signals, namely, $C\_\Sigma1$, $C\_\Sigma2$, C_P12, C_P34, C_P13, C_P24, C_P23, C-Sl, C_S2, C_S3, and C_S4. All these signals can be generated at about the same time. The table is the "programmable" component of the processor (see Table 2 and the required modification as mentioned in Section 4.1.3). It can be realized as a PLA with 9 inputs, 11 outputs, and at most 79 product terms. It can also easily be split into a few smaller PLAs or logic circuits to further optimize speed and area.

5.2 Permutation Network

The permutation network is divided into 3 sections, namely, the state normalizer, state collider, and state denormalizer, that are very similar in structure. The network takes 24 state bits as inputs, permutes the bits according to the conditional signals from the control generator, and provides 24 state bits as outputs. The width of the data path is uniformly 24 bit all through the network. The structure is very regular. Each *conditional permutation operator* is a hardware realization of some function $\hat{\pi}$ of $\widehat{G_\pi}$. It is a combinational circuit which realizes some $\hat{\pi} : B^n \mapsto B^n$, so that the corresponding isometry is *applied* to the input bits. In Figure 3, say, N_S1 is a control signal to the conditional permutation operator s_1.

Such a conditional permutation unit can be realized as a row of multiplexers. Figure 4 shows the implementation of some of the permutation functions listed in Table 3. Alternatively, since each element of the permutation group can be written as products of disjoint transpositions, it can also be realized as a row of 2x2 switch boxes (see Figure 5). A 2x2 switch box copies its 2 inputs to outputs if the control is 0, and switches the outputs if the control is 1. For a given permutation function, the number of multiplexers is exactly twice the number of switch boxes, which is equal to the number of 2-cycles in its cycle representation. These numbers are independent of velocity labeling.

Some permutation units require longer wires and wider wiring channels than others. As the choice of the velocity labeling function, f_V, determines the exact permutation function representation of an isometry, the labeling function affects the wiring complexity of the permutation network.

The cascade order of the operators in the state normalizer matches the order in which the control signals are generated so as to minimize the critical path delay.

The state denormalizer is the inverse of the state normalizer, so that the order of the signals and the permutation operators are exactly reversed.

5.3 Enhancement for Collisions with Obstacles

So far, we have dealt with a model with no obstacle bits. An obstacle at a node indicates the presence of solid rather than free space. Effectively, we have a different set of collision rules at

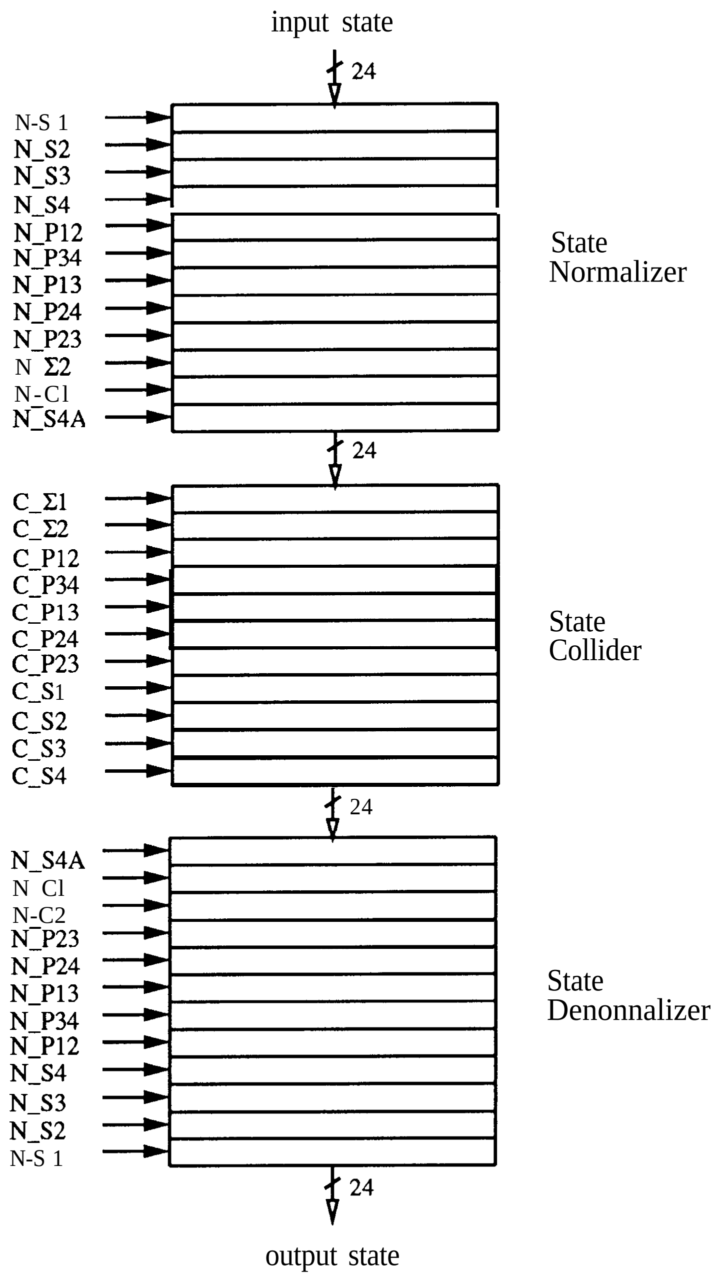

Figure 3: Permutation network

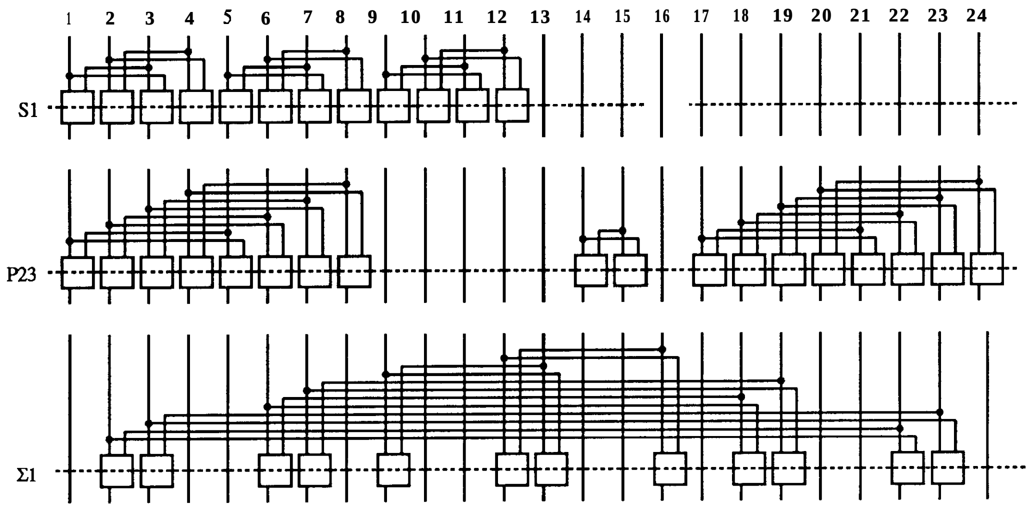

Figure 4: Realization of permutation functions by multiplexers

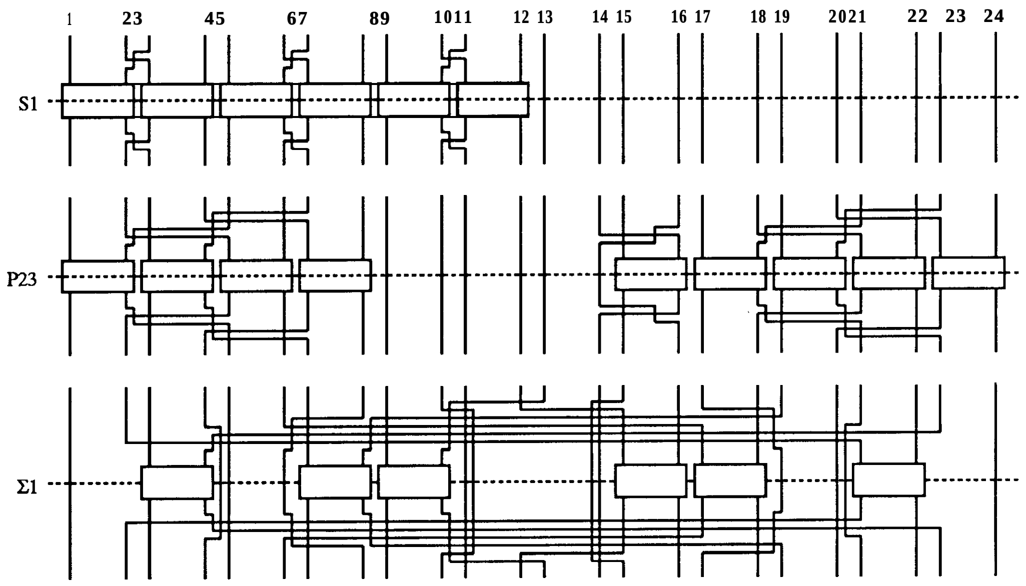

Figure 5: Realization of permutation functions by 2x2 switch boxes

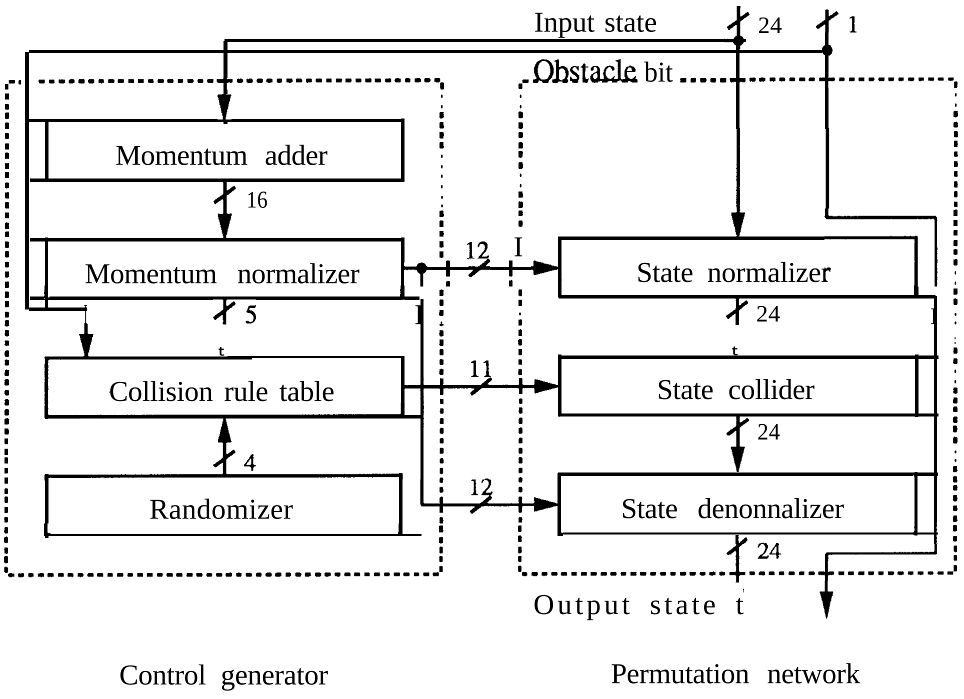

Figure 6: An enhanced processor architecture for the FCHC isometric model

such a node. Figure 6 shows how our design can be modified to accommodate a FCHC model with obstacle bits. Since an obstacle bit does not move, the bit value corresponding to the obstacle bit of a cell does not change in a collision. As an obstacle bit is in its own equivalence class under the isometry group, it can bypass the normalization steps and can be used to lookup the collision rule table directly. An implementation may choose to split the rule table into two, for the cases with and without the obstacle bit.

6 Discussions

The critical path of the whole circuit is the path from input to momentum adder, momentum normalizer, collision rule table, state collider, state denormalizer, and then to output. Assuming 1 μm CMOS technology, the critical path delay is estimated to be about 100 ns, which translates to a throughput of about 10 million operations per second. With a reasonable number of pipeline stages, the throughput can easily be increased a few times. A non-pipelined (minimal) implementation of this architecture would require about 5000 gates, and 50 pins (including clock and reset, but excluding power and ground and other pins for testing). One or several such processors can easily fit on a single chip.

Compare these estimates with those of the table lookup approach using RAM or ROM. For a deterministic collision function of 24 bits, the general table lookup approach requires 24×2^{24}

bits = 384 Mbits = 48 Mbytes. With one obstacle per cell, the memory requirement is doubled. To implement the same isometric collision rules, the straightforward extension of this approach requires at least 12 times the memory requirement, that is, 4.5 Gbits = 576 Mbytes, since the maximum number of optimal isometries for any state is 12 (see Table 2). This takes 1152 4-Mbit DRAM chip; the typical chip access time of which is about 80 ns.

Assuming 4 transistors per gate, and 1 transistor per DRAM cell, the table lookup approach takes about 4.5×10^9 transistors (including address decoders and other necessary glue circuits), while our design takes about 2×10^4 transistors.

The above comparison shows that our design is about 5 orders of magnitude more efficient than the general RAM table lookup approach in terms of active silicon area. The latency is roughly comparable to that of the lookup table. However, it is clear that our design can be easily pipelined with at least a 2 to 5 times increase in throughput, whereas it is not possible to do so in the table lookup approach.

The area efficiency of this architecture is due to the effective ways that *symmetry* properties are exploited. It is interesting to observe that the collision computation structure, though purely combinational in nature, can be nicely broken down into two parts, namely, the control path and the data path. The data path contains a cascade of conditional permutation operators as defined here, which are not present in any existing general purpose computer or in any other special purpose cellular automat a machine.

7 Summary

We have systematically derived a new VLSI architecture for the FCHC isometric lattice gas model from Hénon's isometric collision algorithm. With current technology, the architecture can be implemented at least as fast as the table lookup approach while requiring about 10^5 times fewer transistors. This is the first step toward creating a very high performance machine containing many such processors working in parallel to solve three dimensional lattice gas models.

This architecture is well matched to the algorithm and demonstrates how other architectures may be derived for other models of lattice gas. It is also interesting to generalize the architecture to non-isometric models.

Acknowledgements

We would like to thank Tin-Fook Ngai and Nhon Quach for their comments on the paper. This work was supported by NASA under contract NAG2-248, using facilities supported under NAGW 419.

A Proof: G_π is a permutation group of N

We can easily verify that G_π is a group under function composition. Associativity always holds for function composition. The identity permutation i is in G_π because $I \in G_M$. G_π *is closed* because for all $M_1, M_2 \in G_M$, we have $M_2M_1 \in G_M$. Therefore, for all $f_G(M_1), f_G(M_2) \in G_\pi$, we have $f_G(M_2) \circ f_G(M_1) = f_G(M_2M_1) \in G_\pi$ *by* Proposition 1. Since for all $M \in G_M$, there exists M^{-1}, and $f_G(M) \circ f_G(M^{-1}) = f_G(MM^{-1}) = f_G(I)$. Therefore, each $f_G(M) \in G_\pi$ has an inverse element $f_G(M^{-1})$. Hence, G_π is indeed a permutation group. • I

B Proof: $\widehat{G_\pi}$ is a permutation group of B^n

We first show that $\hat\pi$ is a permutation of B^n, that is, $\hat\pi$ is one-one and onto. To show that $\hat\pi$ is one-one, we only have to prove that no two tuples in B^n are mapped into the same tuple by ii. Suppose there are two tuples x and y both of which are mapped into z under $\hat\pi$; that is,

$$\hat\pi(\mathbf{x}) = \hat\pi(\mathbf{y}) = z$$

$$(x_{\pi^{-1}(1)}, \ldots, x_{\pi^{-1}(n)}) = (y_{\pi^{-1}(1)}, \ldots, y_{\pi^{-1}(n)}) = (z_1, \ldots, z_n)$$

$$x_{\pi^{-1}(j)} = y_{\pi^{-1}(j)} = z_j \text{ for all } j$$

Hence, $x_k = y_k$ for all k, since π is a permutation of N. This means that x and y are the same tuple. Since for all x = $(x_1, \ldots, x_n)$, there exists a tuple $(x_{\pi(1)}, \ldots, x_{\pi(n)})$ = x' such that $\hat\pi(\mathbf{x}')$ = x, $\hat\pi$ is clearly onto.

As associativity always holds for function composition, and the identity $\hat\imath$ is in $\widehat{G_\pi}$ (because i is in G,), in order to show that $\widehat{G_\pi}$ is a group, we must prove that $\widehat{G_\pi}$ is closed under function composition, and each element of $\widehat{G_\pi}$ has an inverse. $\widehat{G_\pi}$ is closed, because for all $\hat\pi_1, \hat\pi_2 \in \widehat{G_\pi}$,

$$\begin{aligned}
(\hat\pi_2 \circ \hat\pi_1)(\mathbf{x}) &= \hat\pi_2(x_{\pi_1^{-1}(1)}, \ldots, x_{\pi_1^{-1}(n)}) \\
&= (x_{\pi_1^{-1}(\pi_2^{-1}(1))}, \ldots, x_{\pi_1^{-1}(\pi_2^{-1}(n))})^1 \\
&= (x_{(\pi_1^{-1}\circ\pi_2^{-1})(1)}, \ldots, x_{(\pi_1^{-1}\circ\pi_2^{-1})(n)}) \\
&= (x_{(\pi_2\circ\pi_1)^{-1}(1)}, \ldots, x_{(\pi_2\circ\pi_1)^{-1}(n)}) \\
&= (\widehat{\pi_2 \circ \pi_1})(\mathbf{x})
\end{aligned}$$

For all $\hat\pi \in \widehat{G_\pi}$, there exists an inverse $\hat\pi^{-1} = \widehat{\pi^{-1}}$ so that

$$\begin{aligned}
(\widehat{\pi^{-1}} \circ \hat\pi)(\mathbf{x}) &= \widehat{\pi^{-1}}(x_{\pi^{-1}(1)}, \ldots, x_{\pi^{-1}(n)}) \\
&= (x_{\pi^{-1}(\pi(1))}, \ldots, x_{\pi^{-1}(\pi(n))}) \\
&= \hat\imath(\mathbf{x})
\end{aligned}$$

□

[1]Suppose $\hat\pi_2(y_1, \ldots, y_n) = (y_{\pi_2^{-1}(1)}, \ldots, y_{\pi_2^{-1}(n)})$. If $y_j = x_{\pi_1^{-1}(j)}$, then $y_{\pi_2^{-1}(k)} = x_{\pi_1^{-1}(\pi_2^{-1}(k))}$.

References

[1] U. Frisch, D. d'Humières, B. Hasslacher, P. Lallemand, Y. Pomeau, and J. Rivet, "Lattice Gas Hydrodynamics in Two and Three Dimensions," *Complex Systems,* vol. 1, no. 4, pp. 649–707, 1987.

[2] U. Frisch, B. Hasslacher, and Y. Pomeau, "Lattice-Gas Automata for the Navier-Stokes Equation," *Physical Review Letters,* vol. 56, no. 14, pp. 1505-1508, 1986.

[3] D. d'Humières, P. Lallemand, and U. Frisch, "Lattice Gas Models for 3D Hydrodynamics," *Europhysics Letters,* vol. 2, pp. 291-297, August 1986.

[4] T. Toffoli and N. Margolus, *Cellular Automata Machines – A New Environment for Modeling.* MIT Press, 1987.

[5] A. Clouqueur and D. d'Humières, "RAP1, a Cellular Automaton Machine for Fluid Dynamics ," *Complex Systems,* vol. 1, pp. 585-597, 1987.

[6] M. Hénon, "Isometric Collision Rules for the Four-Dimensional FCHC Lattice Gas," *Complex Systems,* vol. 1, pp. 475-494, June 1987.

[7] J. L. Gersting, *Mathematical Structures for Computer Science.* W. H. Freeman and Company, second ed., 1987.

[8] D. I. A. Cohen, *Basic Techniques of Combinatorial Theory.* John Wiley & Sons, 1978.

[9] S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers.* CBS College Publishing, 1982.

[10] D. E. Knuth, *The Art of Computer Programming.* Vol. 3, Addison-Wesley Pub. Co., 1968. Sorting and Searching.