An Adaptive Parallel Memory System for Image Processing

Zhiyong Liu, Xiaobo Li , and Jia-Huai You
Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada T6G 2H1

Abstract

We present a flexible parallel storage scheme for image processing. The scheme features that it is adaptive to different parallel access patterns and different sizes of the image arrays to be processed on parallel processing systems. The storage scheme uses N memory modules where N is any complete power of two. By selecting different address generation function the scheme allows various subarrays of N elements to be accessed simultaneously without memory conflict. Address generation in the scheme needs only exclusive-or operations and can be completed in constant time. All the data alignments requirements in the scheme can be implemented through popular interconnection networks.

Key Words: *Image processing, Parallel processing, Memory conflict, Skewing scheme, Address generation.*

1 Introduction

In image processing systems, an image is expressed as an integer array, and matrix operations are used intensively. For parallel processing, some subsets of a matrix need to be accessed simultaneously in order to increase the effective memory bandwidth and processing speed. Consider an $N \times N$ matrix to be processed by a system with N processors and a number of memory modules. When accessing a subset of the $N \times N$ matrix simultaneously, if more than one element in the subset lies in the same memory module, memory conflict occurs. Memory conflicts result in degradation of effective memory bandwidth. Many parallel storage schemes, called *skewing schemes*, have been proposed to store different elements of a desired subset in different memory modules.

There are some basic considerations in the design of skewing schemes. The first is the requirements of the access patterns, i.e. what subsets of data can be accessed simultaneously. The second is address generation, i.e. the calculation of the memory module number and local address for each element to be accessed. The third is alignment of data, i.e. the arrangement of the elements in some desired order. While the first issue is about the functionality of a skewing scheme itself, the second and the third are considered implementation issues of a skewing scheme.

Skewing schemes are an important issue in image processing, pattern recognition, raster graphics, numerical analysis, signal processing, etc., and it has been studied by a number of researchers [2, 8, 18, 17, 9, 4, 15, 10, 11, 12, 7, 19, 16, 6, 3, 14]. Frailong et al present a general description method of a kind of skewing schemes — XOR-schemes — in [4], and construct an XOR-scheme, where rows, columns, squares, and rectangles of various shapes can be accessed simultaneously.

XOR-schemes have the advantages that they use the smallest number (N) of memory modules to store an $N \times N$ image array, and that they are easy to be implemented. The skewing schemes proposed in [4, 10, 11, 7, 16, 6, 14] are all XOR-schemes. Any particular XOR-scheme can be easily implemented in hardware for an array of particular size, and for limited patterns of subarray, but difficulty arises when the following facts are considered.

First, most proposed XOR-schemes do not handle the case that N is an *odd* power of two. Manipulation of an $N \times N$ matrix with N being an odd power of two is often indispensable in many applications. For example, in image processing, a digital image with a quality comparable to that of monochrome TV pictures consists of $2^9 \times 2^9$ pixels [5]. We will propose skewing schemes which can apply to $N \times N$ matrices with N being any power of two.

Secondly, in practice it is usually the case that a parallel processing system is used to process different sized arrays. If the size of the system is smaller than N, one can use several memory cycles to access N elements. This will not lower the utilization of the resources. Using smaller systems to process large matrices has been discussed in [2, 9, 7]. If the size of the system is larger than the size of the array, one may hope to access more than N elements at a time so that the array will be processed more efficiently. If arrays with different sizes need different *mapping matrices* (defined in Section 2), different address generation functions are necessary. Although address generation in some XOR-schemes is feasible for processing arrays of one fixed size, it is difficult to implement when arrays with different sizes are considered. It is preferable that a uniform address generation scheme, which is independent of the size of the array to be processed, can be used for arrays of different sizes. We will show that the skewing scheme developed in this paper can naturally satisfy these kinds of applications.

Thirdly, it is impossible to find a skewing scheme which allows arbitrary subarray with N elements to be accessed in parallel without memory conflict [2, 7]. However, for any particular application algorithm (or in any particular computation stage of an algorithm),

very limited patterns (or only one pattern) of subarrays are required to be accessed in parallel. Based on this observation, one may want to support different skewing schemes in one system, and allow different applications to use different schemes. Such a system must support different address generation functions, one for each skewing scheme. Because address generation is carried out at run time, the switch between different address generation functions should be simple. We will propose a simple mechanism which allows a large set of frequently used patterns to be accessed in parallel.

2 Some Definitions

Suppose an $N \times N$ matrix X is to be processed on a system with N memory modules, where $N = 2^n$ and n is any integer. We use an $N \times N$ matrix M to describe a skewing scheme: element $x_{i,j}$ of X is stored in memory module $m_{i,j}$. We call matrix M the *mapping matrix*. We express an integer in its binary form. Any XOR-scheme [4] can be described by the following equation:

$$m^{\tau}_{i,j} = (A \times i^{\tau}) + (B \times j^{\tau}), \qquad (1)$$

where A and B are $n \times n$ binary matrices, and the operations "$\times$" and "$+$" are carried out over GF(2). We will use matrix E to denote the identity matrix. We denote column k of matrix A as $a_{*,k}$, for $0 \leq k \leq n-1$. Note that we will use $m_{i,j} = (A \times i + B \times j)$ to denote $m^{\tau}_{i,j} = (A \times i^{\tau}) + (B \times j^{\tau})$ in the following context when no confusion will arise. In this paper, the following subarrays of N elements are considered.

- Rows [2, 8].
- Columns [2, 8].
- Diagonals [2, 8].
- Blocks of $P \times Q$, where P and Q are integers and $P \times Q = N$ [2, 18, 4].
- "The Same points" in blocks [10, 7]. — We will call this type of patterns "scattered blocks".
- Red and black chessboards [4].
- Upper-fold and lower-fold lines [13].
- Partial-row-pairs and partial-column-pairs [13].

3 The Proposed Skewing Schemes

3.1 Scheme 1: the Exchange-Expansion (EE) Scheme

Intuitively, the EE scheme works as follows. Let the elements in the first row of M be numbered from 0 to $N-1$. The second row is formed by changing the first and the second halves of the first row, the next two rows are created from the first two rows by cutting the first two rows into quarters and exchanging the first and second quarters and the third and fourth, $\cdots$, the lower half matrix is formed by exchanging $N/2$ two adjacent columns of the up half matrix ($N/2$ rows).

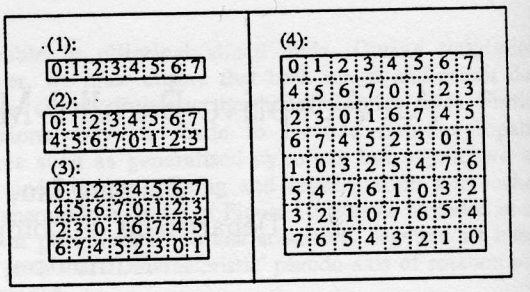

Figure 1: Exchange-Expansion Process for $N = 8$

Figure 2: A 16 × 16 Mapping Matrix for Scheme 1

The resulting matrix is our mapping matrix M. This expansion process is shown in Figure 1 for $N = 8$. Note that, theoretically, when we begin to construct the mapping matrix M, the elements of the first row are not necessarily numbered sequentially (the order can be arbitrary); but for the simplicity of implementation (address generation and data alignment), we need to give some desired order. The order for the first row in Figure 1 is just a naturally sequential one.

We propose that in any memory module the elements of a matrix are stored according to their row index, therefore address generation in our scheme refers to the memory module number computation.

Let $br(i)$ be the bit-reversal function of integer i, and let $\oplus$ be the bitwise exclusive-or operation. Given the indices of an element $x_{i,j}$, $m_{i,j}$ is decided by the following equation:

$$m_{i,j} = br(i) \oplus j. \qquad (2)$$

In terms of XOR-scheme expression the scheme can be expressed as

$$m_{i,j} = (R \times i) \oplus (E \times j), \qquad (3)$$

where matrix R is such that all the elements on its backward diagonal, and only these elements, are "1".

It is not difficult to see that Equation 2 does implement the idea of Exchange-Expansion. A mapping matrix generated by Equation 2 for $N = 16$ is given in Figure 2.

We will prove that in the EE scheme, for any integers P and Q ($PQ = N$), a continuous area of $P \times Q$

in matrix X can be accessed simultaneously in one memory cycle. We will prove that the simultaneously accessible blocks can move along vertical and horizontal stripes in our scheme. Also, we will prove that N points scattered over the matrix plane are accessible in parallel. Some continuous areas which are simultaneously accessible are shown in Figure 2.

Definition 1 *A main PQ-block (mb(PQ)) consists of all the elements* $x_{i+a,j+b}$

$$mb(PQ) = \{x_{i+a,j+b}\},$$

where $i = 0 \bmod P$, $j = 0 \bmod Q$, $0 \le a \le P-1$ *and* $0 \le b \le Q-1$.

Theorem 1 *With the memory module number generation scheme of Equation 2, any main PQ-block can be accessed in one memory cycle.*

Proof:

Let $x_{i,j}, x_{i',j'} \in mb(PQ)$. We need only to prove that $m_{i,j} = m_{i',j'}$ if and only if $(i = i') \wedge (j = j')$, i.e. all the elements in a main PQ-block are stored in different memory modules.

Because $PQ = N$, we can express P as $P = 2^p$, Q as $Q = 2^q$, and $p + q = n$. From Equation 2 we have

$$m_{i,j} = (i_0 \cdots i_{p-1} i_p \cdots i_{n-1}) \oplus (j_{n-1} \cdots j_q j_{q-1} \cdots j_0)$$

and

$$m_{i',j'} = (i'_0 \cdots i'_{p-1} i'_p \cdots i'_{n-1}) \oplus (j'_{n-1} \cdots j'_q j'_{q-1} \cdots j'_0).$$

Because $x_{i,j}$ and $x_{i',j'}$ are in the same main PQ-block, we know $i_{n-1} \cdots i_p = i'_{n-1} \cdots i'_p$ and $j_{n-1} \cdots j_q = j'_{n-1} \cdots j'_q$.

Considering $p + q = n$, we have

$$m_{i,j} \oplus m_{i'j'} = (i_0 \oplus i'_0) \cdots (i_{p-1} \oplus i'_{p-1})(j_{q-1} \oplus j'_{q-1}) \cdots (j_0 \oplus j'_0). \quad (4)$$

From Equation 4 we know that $m_{i,j} = m_{i',j'}$ if and only if $(i = i') \wedge (j = j')$.

Q.E.D.

The theorem guarantees that any one row, two adjacent half rows, four adjacent one fourth rows, $\cdots\cdots$, one column, can be accessed simultaneously in one memory cycle. The "shape" of the desired subset is variable.

We have the following stronger theorem for the scheme.

Theorem 2 *Any block of* $\{x_{(i+a) \bmod N, (j+b) \bmod N}\}$ *can be accessed simultaneously in one memory cycle, where* $i = 0 \bmod P$ **or** $j = 0 \bmod Q$; $0 \le a \le P-1$ *and* $0 \le b \le Q-1$.

This theorem says that not only are the shapes of simultaneously accessible blocks variable, but the blocks are also circularly movable along horizontal strips of width P, or vertical strips of width Q.

Corollary 1 *Any block of N elements starting from any arbitrary position can be accessed in at most two memory cycles.*

In image processing, some scattered image points need to be accessed simultaneously. These points are called points *at the same positions* [10, 7]. We will call them *scattered blocks* hereafter.

Definition 2 *Given integers a and b,* $0 \le a \le P-1$, $0 \le b \le Q-1$, *a scattered PQ-block (sb(PQ-ab)) is defined as the set consisting of all the elements* $x_{i,j}$ *with* $i = a \bmod P$ *and* $j = b \bmod Q$:

$$sb(PQ-ab) = \{x_{i,j} \mid (i = a \bmod P) \wedge (j = b \bmod Q)\}.$$

Theorem 3 *With the memory module number generation scheme of Equation 2, any scattered PQ-block (sb(PQ-ab)) can be accessed in one memory cycle.*

Notice that the sets of elements at the same positions in [10, 7] are only the elements at the same positions in each main $\sqrt{N} \times \sqrt{N}$ subsquares while scattered PQ-blocks here can be the elements at the same positions in main PQ-blocks with different shapes.

3.2 Scheme 2: the GRAY Scheme

Intuitively, the scheme is as follows: the first column is numbered from 0 to $N-1$, we exchange each pair of adjacent numbers to form the second column; then flip these two columns and exchange each pair of blocks with four adjacent numbers to form the third and the fourth columns, $\cdots\cdots$, at last, flip the first half matrix ($N/2$ columns) and exchange the two blocks of $2^{n-1} \times 2^{n-1}$ adjacent elements to form the last $N/2$ columns of the matrix. The resulting matrix is our mapping matrix. This corresponds to the following equation.

$$m_{i,j} = i \oplus (G \times j), \quad (5)$$

where matrix G has the form that $g_{n-1,n-1} = 1$, $g_{k,k} = g_{k,k+1} = 1$, for $0 \le k \le n-2$, and the remaining elements of G are all 0.

For example, $G_{4\times4}$ is

$$G_{4\times4} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

In fact $G \times j$ is the Gray code of j for $0 \le j \le N-1$, so that the skewing scheme is named GRAY scheme. The mapping matrix of 16×16 is shown in Figure 3. The scheme has the following characteristics.

Because $G_{n\times n}$ is nonsingular for any n, we have the following theorem.

Theorem 4 *In GRAY scheme, any row or any column can be accessed simultaneously in one memory cycle.*

Definition 3 *A CRS* $\bmod 2^c$ *(Complete Residue System* $\bmod 2^c$*) is a set of the integers from 0 to* $2^c - 1$, *where c is an integer.*

0	1	3	2	6	7	5	4	12	13	15	14	10	11	9	8
1	0	2	3	7	6	4	5	13	12	14	15	11	10	8	9
2	3	1	0	4	5	7	6	14	15	13	12	8	9	11	10
3	2	0	1	5	4	6	7	15	14	12	13	9	8	10	11
4	5	7	6	2	3	1	0	8	9	11	10	14	15	13	12
5	4	6	7	3	2	0	1	9	8	10	11	15	14	12	13
6	7	5	4	0	1	3	2	10	11	9	8	12	13	15	14
7	6	4	5	1	0	2	3	11	10	8	9	13	12	14	15
8	9	11	10	14	15	13	12	4	5	7	6	2	3	1	0
9	8	10	11	15	14	12	13	5	4	6	7	3	2	0	1
10	11	9	8	12	13	15	14	6	7	5	4	0	1	3	2
11	10	8	9	13	12	14	15	7	6	4	5	1	0	2	3
12	13	15	14	10	11	9	8	0	1	3	2	6	7	5	4
13	12	14	15	11	10	8	9	1	0	2	3	7	6	4	5
14	15	13	12	8	9	11	10	2	3	1	0	4	5	7	6
15	14	12	13	9	8	10	11	3	2	0	1	5	4	6	7

Figure 3: A 16 x 16 Mapping Matrix for Scheme 2

Let {} denote a set of integers.

Lemma 1 $\forall k,\ 0 \leq k \leq N-1,\ k = k_{n-1}\cdots k_{c+1}k_c k_{c-1}\cdots k_0$, $\forall c,\ 0 \leq c \leq n-1$, *if* $k_c = 1$, *we have* $\{m_{l,k} \mid l_{n-1}\cdots l_{c+1} = k_{n-1}\cdots k_{c+1},$ *and* $l_c = \bar{k}_c\} = \{m_{k,l} \mid l_{n-1}\cdots l_{c+1} = k_{n-1}\cdots k_{c+1},$ *and* $l_c = \bar{k}_c\}$.

Proof:

$\forall l,\ l_{n-1}\cdots l_{c+1} = k_{n-1}\cdots k_{c+1},\ l_c = \bar{k}_c$, we know $m_{l,k} = l \oplus (G \times k) = 0\bar{k}_{n-1}k_{n-2}\cdots k_{c+2}\bar{k}_{c+1}m_{l,k,c-1}\cdots m_{l,k,0}$, where $m_{l,k,t} = l_t \oplus (k_t \oplus k_{t+1})$ for $0 \leq t \leq c-1$.

Obviously, $\{l \mid l_{n-1}\cdots l_{c+1} = k_{n-1}\cdots k_{c+1},$ and $l_c = \bar{k}_c\}$ constitute a CRS mod 2^c. Thus $\{m_{l,k} \mid l_{n-1}\cdots l_{c+1} = k_{n-1}\cdots k_{c+1},$ and $l_c = \bar{k}_c\}$ constitute a CRS mod 2^c and all the 2^c numbers have the form that $m_{l,k,n-1}m_{l,k,n-2}\cdots m_{l,k,c} = 0k_{n-1}k_{n-2}\cdots k_{c+2}\bar{k}_{c+1}$.

Similar to the above, we know that $\{m_{k,l} \mid l_{n-1}\cdots l_{c+1} = k_{n-1}\cdots k_{c+1},$ and $l_c = \bar{k}_c\}$ constitute a CRS mod 2^c and all the 2^c numbers have the form that $m_{l,k,n-1}m_{l,k,n-2}\cdots m_{l,k,c} = 0k_{n-1}k_{n-2}\cdots k_{c+2}\bar{k}_{c+1}$.

Q.E.D.

Lemma 2 $\forall k, 0 \leq k \leq N-1,\ \{m_{l,k} \mid 0 \leq l \leq k\} = \{m_{k,l} \mid 0 \leq l \leq k\}$.

This lemma can be proved by using Lemma 1 repeatedly.

Definition 4 *Upper-fold(k) is the elements* $\{x_{0,k}, x_{1,k}, \cdots, x_{k-1,k}, x_{k,k+1}, x_{k,k+2}, \cdots, x_{k,N-1}\}$.

Definition 5 *Lower-fold(k) is the elements* $\{x_{k,0}, x_{k,1}, \cdots, x_{k,k-1}, x_{k+1,k}, x_{k+2,k}, \cdots, x_{N-1,k}\}$.

The elements of upper-fold(6) and lower-fold(9) are shown in Figure 3. From Theorem 4 and Lemma 2, we have the following two theorems.

Theorem 5 $\forall k, 0 \leq k \leq N-1$, *Upper-fold(k) can be accessed in one memory cycle.*

Theorem 6 $\forall k, 0 \leq k \leq N-1$, *Lower-fold(k) can be accessed in one memory cycle.*

Theorem 7 *The N elements on the forward or backward diagonal can be accessed in two memory cycles.*

Proof:

From Equation 5 we can see that any element on the forward diagonal can be expressed as

$$m_{k,k} = (G \times k) \oplus k = 0k_{n-1}k_{n-2}\cdots k_1.$$

Thus there are two and only two elements on the forward diagonal storing in the same memory module $0k_{n-1}k_{n-2}\cdots k_1$.

It can be proved similarly for the backward diagonal.

Q.E.D.

Although in this scheme it needs two memory cycles to access the elements on either diagonal, we have the following result:

Corollary 2 *The 2N elements on the forward and the backward diagonals can be accessed in two memory cycles.*

3.3 Scheme 3: the Combination of EE and GRAY

We investigate the combination of the schemes EE and GRAY, which is a new skewing scheme and is expressed as

$$m_{i,j} = (R \times i) \oplus (G \times j). \tag{6}$$

The scheme guarantees rows, columns, various blocks of N elements to be accessed simultaneously in one memory cycle, it also guarantees any diagonal to be accessed in two memory cycles and two diagonals ($2N$ elements) to be accessed in two memory cycles. This scheme has the additional feature that chessboard patterns of N elements in various areas can be accessed in one memory cycle.

According to [4], the *red chessboard* $RC(n,c)$ consists of the elements in the rectangle of $2^c \times 2^{n+1-c}$ such that the sum of their indices i and j is even. The *black chessboard* $BC(n,c)$ consists of the rest elements in the rectangle. The red chessboard $RC(n,c)$ based at (0,0) are accessible in one memory cycle if and only if the n column-vectors $(a_{*,0} \oplus b_{*,0}), b_{*,1}, b_{*,2}, \cdots, b_{*,c-1}, a_{*,1}, a_{*,2}, \cdots, a_{*,n-c}$ are linearly independent (notice that here the symbols a and b have been changed from [4] because i is the row index and j is the column index here). The scheme given in [4] guarantees conflict-free access to $RC(n,1)$, $RC(n,n)$, $RC(n,n/2)$ and $RC(n,n/2+1)$ for n being an even number.

The scheme expressed by Equation 6 has the following stronger feature.

Theorem 8 *In scheme 3, for any integer n and c,* $1 < c \leq n+1$, $RC(n,c)$ *(also* $BC(n,c)$*) can be accessed in one memory cycle.*

0	1	3	2	6	7	5	4
4	5	7	6	2	3	1	0
2	3	1	0	4	5	7	6
6	7	5	4	0	1	3	2
1	0	2	3	7	6	4	5
5	4	6	7	3	2	0	1
3	2	0	1	5	4	6	7
7	6	4	5	1	0	2	3

Figure 4: The 8 x 8 Mapping Matrix for Scheme 3

0		3		6		5	
	5		6		3		0
2		1		4		7	
	7		4		1		2
1		2		7		4	
	4		7		2		1
3		0		5		6	
	6		5		0		3

(1) RC(3,2)

	1		2		7		4
4		7		2		1	
	3		0		5		6
6		5		0		3	
	0		3		6		5
5		6		3		0	
	2		1		4		7
7		4		1		2	

(2) BC(3,3)

Figure 5: Chessboards in Scheme 3

Proof:

Given n column-vectors $(g_{*,0} \oplus r_{*,0}), r_{*,1}, \cdots, r_{*,c-1}, g_{*,1}, \cdots, g_{*,n-c}$, where $1 < c < n+1$, one can always add $r_{*,c-1}$ to $g_{*,n-c}$ then an $e_{*,n-c-1}$ is generated, add this to $g_{*,n-c-1}$ then $e_{*,n-c-2}$ is generated, $\cdots\cdots$, add $e_{*,0}$ to $g_{*,0} \oplus r_{*,0}$ then $e_{*,n-1}$ is generated. So we can get n column-vectors $e_{*,0}$ through $e_{*,n-1}$, which is obviously linearly independent.

Thus we know that $\forall c, 1 < c < n+1$, the n column-vectors $(g_{*,0} \oplus r_{*,0}), r_{*,1}, \cdots, r_{*,c-1}, g_{*,1}, \cdots, g_{*,n-c}$ are linearly independent.

Q.E.D.

The mapping matrix for $N = 8$ is give in Figure 4. Two kinds of chessboards are given in Figure 5, one is $RC(3,2)$, the other is $BC(3,3)$.

4 Adapting to the Schemes in One System

Memory module number calculation needs to be carried out in run time, so that it should be implemented it in hardware. From Equations 3, 5, and 6 we can see that the three schemes proposed in Section 3 employ very simple memory module number calculation. In the schemes proposed in Section 3, only one or two "1"s exist in any row of any matrix, thus the memory number calculation functions here can be completed in constant time. Further more, it is easy to synthesize the three schemes in one system so that the system can support all the frequently used access patterns mentioned in Section 2. Only two bits are needed to select the desired scheme in the application algorithms. Let's call the two bits *"the scheme selection register"*, and "01" be for scheme 1, "10" for scheme 2, and "11" for scheme 3. The pattern "00" can be used for implementing the scheme $m_{i,j} = i \oplus j$, which is first stated in [1] and used in STARAN, thus all the simultaneously accessible patterns (e.g. upper and lower folds, partial-row and partial-column pairs) in the scheme of $m_{i,j} = i \oplus j$ can all be retained in one system. The structure of the mechanism to implement the memory module calculation is pictured in Figure 6.

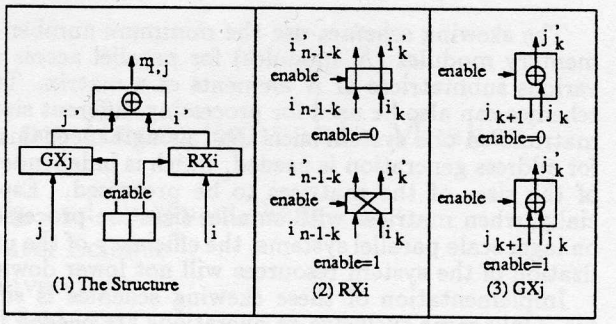

Figure 6: The Address Generation Mechanism

When the system is used for a particular algorithm, one only need to set the "scheme selection register" to a particular value.

5 Processing Different Sized Arrays on Large Systems

In practice, the number of processing elements may not be the same as the size of the matrix being processed. Let X be a matrix of $N \times N$ to be processed on a system with W memory modules and W processing elements. Let $U = W/N$.

Processing large matrices on small systems has been discussed in [2, 9, 7]. If $W \leq N$, it is simple to use several $((N/W)^2)$ mapping matrix planes to tessellate the plane of the array of $N \times N$ to be processed. This will not lower the utilization of the system resources. When $W > N$, if we produce an $N \times N$ mapping matrix, and still only N elements (e.g. one row, or one column) can be accessed in parallel, the efficiency of the system will be lowered down. Our goal here is to access $U \times N (= W)$ elements of matrix X simultaneously in one memory cycle. We will see that scheme 1, and scheme 3 naturally satisfies this requirement.

With scheme 1 or scheme 3, it is easy to deal with the case of $W > N$. One can simply use a part of the mapping matrix of $W \times W$ (for example, the most convenient is the top left part of $N \times N$) to be the mapping matrix of $N \times N$. This implies that for different sized matrices we need only a single mapping matrix of $W \times W$ on a system of size W, thus only a single mechanism for address generation is needed, which is independent of the size of the matrix to be processed. All the characteristics listed in the previous sections are retained except the scattered blocks, because we do not have $P \times Q = W$ points scattered over an $W \times W$ plane on the $N \times N$ plane if $W > N$.

6 Conclusions

We have presented some parallel skewing schemes with simple implementation methods for image processing, and presented a mechanism which can efficiently adapt to the skewing schemes in one system.