A Hierarchical N-Queen Decimation Lattice and Hardware Architecture for Motion Estimation

Chung-Neng Wang, *Member, IEEE*, Shin-Wei Yang, Chi-Min Liu, and Tihao Chiang, *Senior Member, IEEE*

***Abstract*—A subsampling structure, an N-Queen lattice, for spatially decimating a block of pixels is presented. Despite its use for many applications, we demonstrate that the N-Queen lattice can be used to speed up motion estimation with nominal loss of coding efficiency. With a simple construction, the N-Queen lattice characterizes the spatial features in the vertical, horizontal, and diagonal directions for both texture and edge areas. Especially in the 4-Queen case, every skipped pixel has the minimal and equal distance of unity to the selected pixel. It can be hierarchically organized for variable nonsquare block-size motion estimation. Despite the randomized lattice, we design compact data storage architecture for efficient memory access and simple hardware implementation. Our simulations show that the N-Queen lattice is superior to several existing sampling techniques with improvement in speed by about N times and small loss in peak SNR (PSNR). The loss in PSNR is negligible for slow-motion video sequences and is less than 0.45 dB at worst for high-motion estimation sequences.**

***Index Terms*—Decimation lattice, fast motion estimation, hierarchical decimation lattice, N-Queen pattern, pixel decimation, video coding.**

I. Introduction

SEVERAL video coding standards including MPEG-1/2/4 and H.261/3/4 contain block motion estimation as the most computationally intensive module. Reducing the number of operations for block matching can speed up motion estimation. We can improve motion estimation by reducing the number of search points [1], [2], the number of pixels from a block used for matching [3]–[8], and the number of operations for measuring the distortion [1], [9]. The MPEG-4 reference software has provided two fast algorithms that have significantly reduced the number of search points [1], [2]. The pixel decimation approaches can be easily combined with approaches from the remaining categories. In this paper, we will focus on the issue of pixel decimation to achieve further improvement.

The pixel decimation can be realized with either fixed patterns [3]–[6] or adaptive patterns [7], [8]. As shown in Fig. 1(b), Bierling used an orthogonal sampling lattice with a 4:1 subsampling structure [3], which is referred to as the Quarter pattern. Liu and Zaccarin achieved the pixel decimation that is similar to the Bierling's approach with four alternating subsampling patterns selected for each step so that all of the pixels in the current block are visited [6]. The adaptation of a pixel decimation pattern can be based on the spatial luminance variation within a picture [7], [8]. Adaptive techniques can achieve better coding efficiency than that of the uniform subsampling schemes [3]–[6] but with an overhead in selecting the most representative pattern. Due to the penalty caused by mispredicted branches, the irregular or adaptive sampling structure is undesirable for pipelined implementation and efficient hardware realization.

The Quarter pattern has advantages in scheduling the operations in a pipeline fashion with efficient memory access. It has a disadvantage of having pixels with irregular distances of both 1 and $\sqrt{2}$. It also lacks half of the coverage in the vertical, horizontal, and diagonal directions. To represent key features and maintain pipelined memory access, we will construct a family of lattices that retain the regularity and characterize more spatial luminance variation characteristics in all directions.

The goal is to find a sampling lattice that represents the spatial information in all directions and the pixels are distributed uniformly in the spatial domain. To enhance the Quarter pattern, we discovered that the N-Queen lattice [10], [11] improves the representation since it holds exactly one pixel for each row, column, and (not necessarily main) diagonal of a block, as illustrated in Fig. 1(c) and (d). Thus, there are exactly N pixels for each $N \times N$ block. The experimental results show that the visual quality is maintained about the same for N equal to 4 and 8. Based on the MMX single instruction multiple data (SIMD) architecture [12], we show an example of pipelined block matching using a 4-Queen pattern to speed up the conventional full search. As compared to the full search using the Full pattern, the proposed full search using the 4-Queen pattern shows an improvement 3.2 times in speed and occupies only 25% memory bandwidth.

This paper is organized as follows. In Section II, the N-Queen pixel decimation is described. Section III shows the hardware architecture of the N-Queen decimation approach. Section IV demonstrates the performance of the proposed pixel decimation approach versus the other pixel decimation approaches. The paper concludes in Section V.

Manuscript received April 16, 2003; revised June 26, 2003. This work was supported by the National Science Council, Taiwan, R.O.C., under Contract NSC 92-2220-E-009-016. This paper was recommended by Associate Editor X. Mouli.

C.-N. Wang is with the Department and Institute of Computer Science and Information Engineering and the Department and Institute of Electronics Engineering, National Chiao Tung University, Hsinchu 30050, Taiwan (e-mail: cnwang@ csie.nctu.edu.tw).

S.-W. Yang is with the Department and Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 30050, Taiwan. He is also with the Multimedia Technologies Laboratory, Institute for Information Industry, Taipei, Taiwan (e-mail: swyang@csie.nctu.edu.tw).

C.-M. Liu is with the Department and Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 30050, Taiwan (e-mail: cmliu@csie.nctu.edu.tw).

T. Chiang is with the Department and Institute of Electronics Engineering, National Chiao Tung University, Hsinchu 30050, Taiwan (e-mail: tchiang@mail.nctu.edu.tw).

Digital Object Identifier 10.1109/TCSVT.2004.825550

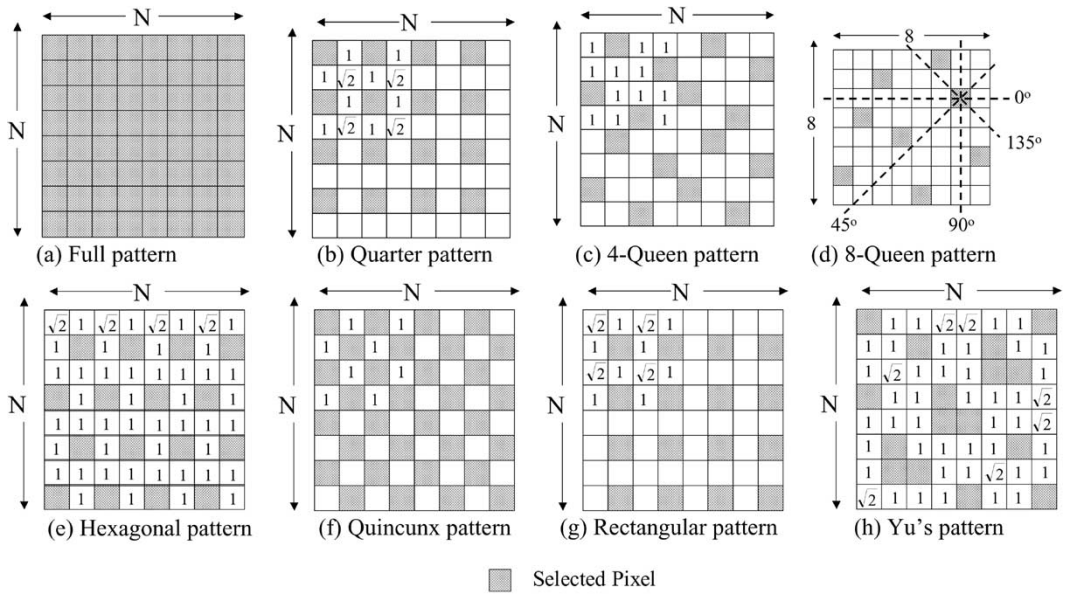


Fig. 1. Pixel patterns for decimation. (a) Full pattern with $N \times N$ pixels selected. (b) Quarter pattern uses 4:1 subsampling. (c) A 4-Queen pattern is tiled with four identical patterns. (d) An 8-Queen pattern. (e) Hexagonal pattern adopts 4:1 subsampling. (f) Quincunx pattern uses 2:1 subsampling. (g) Rectangular pattern takes 4:1 subsampling. (h) Yu's pattern [5] adopts 4:1 subsampling. (c) and (d) are derived from the N-Queen approach with $N = 4$ and $N = 8$, respectively.

II. N-Queen Decimation Approach

A. Conventional Block Matching

The motion estimation module searches for a block from the reference frame such that it has the minimal distortion as compared to the current block. A distortion measure is used to derive the discrepancy between each pair of pixels from the current and reference blocks. For an input frame with height H and width W, the best motion vector is derived from the conventional full search strategy as shown in

$$J_c^* = arg\left(\min_{j=1\sim(N+2M)^2}\left\{\sum_{i=1}^{N^2} Diff\left(F_c(i), F_r(i+V_j)\right)\right\}\right), \quad c = 1, \ldots, N_B \tag{1}$$

where $N_B = (HW/(N \times N))$ is the total number of blocks of size $N \times N$ in a frame. M indicates the search range. The $Diff(\vec{f}, \vec{g})$ is the matching criterion for computing the difference between each pair of data from vectors $\vec{f}$ and $\vec{g}$. The vectors $F_c(.)$ and $F_r(.)$ represent the pixels from the current and reference frames, respectively. Thus, the motion vector with the minimal value J_c^* is viewed as the best candidate after the block matching and is used for motion compensation.

Finding the best motion vector from (1) is computationally expensive. In such a matching process, the computation of distortion is critical for the overall performance. Balancing the coding efficiency and the computational complexity, a simple block-matching metric, namely sum of absolute difference (SAD), is used in our experiments. To speed up the full search, various blocking matching techniques based on reduced SAD metrics have showed a significant improvement [3]–[9]. For example, the partial SAD metrics improve the motion search with a subset of the pixels in the block for finding the best motion vector [9]. The key issues are to determine how to select a representative subset of the pixels in a block and when to stop the computation of the SAD. Since a representative subset of pixels in a block can increase the convergence speed for blocking matching based on the reduced SAD metrics. Thus, our goal is to find the most representative sampling lattice for a block in order to accomplish a fast motion estimation algorithm.

B. Rationale

The pixel decimation process is to reduce the computation in calculating distortion measuring for each pair of blocks [3]. The basic idea is to find a sampling lattice with less SAD computation but still the best motion vector that can be derived. The results should be equivalent with or close to the motion vector found through the standard SAD metrics. Such a sampling lattice, named as the most representative sampling lattice, can be used to extract a subset of the pixels in a block for the reduced SAD metrics.

The most representative sampling lattice is selected based on how much the texture and edge information are preserved with a minimal number of pixels. The sampling lattice is analyzed with both the spatial homogeneity and directional coverage. The spatial homogeneity is based on the presence of significant intraframe correlations within an image [13]. The intraframe autocorrelation function for a less detailed image is higher when the horizontal and vertical spacing (in pixels) are close to 1. Similar observations can be found for highly detailed images. That is, the correlation coefficient between the neighboring pixel increases as their spacing decreases. Thus, the spatial homogeneity is measured by the mean (μ_d) and variance (σ_d^2) of spatial distances from each skipped pixel to its nearest selected pixel [11].

Smaller mean and variance indicate a more spatially homogeneous sampling lattice. The directional coverage is measured by the percentage of coverage in directions such as diagonal, vertical, and horizontal directions. Table I shows that the Quarter pattern has less spatial homogeneity and lacks half of the coverage in all directions. It also shows that the Quincunx pattern

TABLE I
COMPARISON OF THE SAMPLING LATTICES FOR EACH 8 × 8 BLOCK. IN MEASURING THE DIRECTIONAL COVERAGE, FOUR ORIENTATIONS DESCRIBED IN FIG. 1(d) ARE USED. FOR HORIZONTAL AND VERTICAL DIRECTIONS, THERE ARE EIGHT POSSIBLE EDGES WHILE FOR THE DIAGONAL DIRECTIONS THERE ARE 15 POSSIBLE EDGES

Pattern	Decimation Ratio	Spatial homogeneity		Directional coverage (θ)			
		μ_d	σ_d^2	0^o	90^o	45^o	135^o
Full	1	0	0	$\frac{8}{8}$	$\frac{8}{8}$	$\frac{15}{15}$	$\frac{15}{15}$
Quincunx [9]	2	1	0	$\frac{8}{8}$	$\frac{8}{8}$	$\frac{8}{15}$	$\frac{7}{15}$
Quarter[3]	4	1.14	0.04	$\frac{4}{8}$	$\frac{4}{8}$	$\frac{7}{15}$	$\frac{7}{15}$
Hexagonal [4]	4	1.03	0.11	$\frac{4}{8}$	$\frac{8}{8}$	$\frac{12}{15}$	$\frac{12}{15}$
4-Queen	4	1	0	$\frac{8}{8}$	$\frac{8}{8}$	$\frac{10}{15}$	$\frac{10}{15}$
Rectangular	4	1.14	0.04	$\frac{4}{8}$	$\frac{4}{8}$	$\frac{7}{15}$	$\frac{7}{15}$
Yu's [5]	4	1.10	0.03	$\frac{8}{8}$	$\frac{8}{8}$	$\frac{7}{15}$	$\frac{9}{15}$
8-Queen	8	1.32	0.14	$\frac{8}{8}$	$\frac{8}{8}$	$\frac{8}{15}$	$\frac{8}{15}$

[9] has the best homogeneity, but it lacks half of the coverage in both diagonal directions. To improve the spatial homogeneity and the directional coverage, we construct a new N-Queen decimation lattice [10], [11].

As illustrated in Fig. 1(d), an edge can have four orientations in the horizontal, vertical, and diagonal directions. To fully represent the spatial information of an $N \times N$ block, it is required that at least one pixel should be selected for each row, column, and diagonal. To meet such a constraint, the solution is identical to the problem of placing N queens on a chessboard, which is referred to as an N-Queen pattern. For an $N \times N$ block, as shown in Fig. 1(c) and (d), each selected pixel of the N-Queen pattern occupies a dominant position, which is located at the center. All of the pixels located on the four lines in the vertical, horizontal, and diagonal directions are deleted from the list of the selected pixels. With such elimination process, there is exactly one pixel selected for each row, column, and (not necessarily main) diagonal of the block. Thus, the N-Queen patterns present a $N:1$ subsampling lattice that can speed up the full search by approximately N times.

The N-Queen patterns are not unique. For instance, there are 92 8-Queen patterns for each 8 × 8 block. The remaining issue is to identify which one provides a better representation. For these 92 patterns, the mean values of spatial distances from each skipped pixel to its nearest selected pixel are distributed between 1.29 and 1.37 pixels. Thus, the maximum difference of the average distances is only 0.08 pixel. Fig. 2 shows that the 92 8-Queen patterns have almost identical performance, where the deviation in peak SNR (PSNR) is less than 0.1 dB.

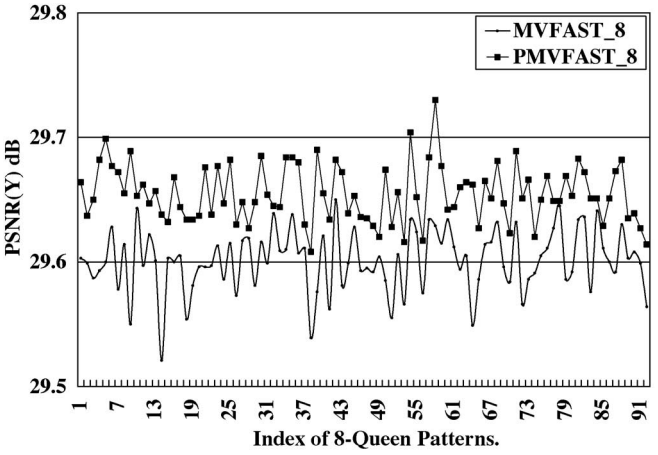


Fig. 2. Performances of the two techniques [1], [2] using 92 8-Queen patterns for the Y component of the Foreman sequence in CIF format. The bit rate is 112 kb/s and the frame rate is 10 fps. The block size is 16 × 16 and the search range is 32 × 32.

C. Recursive Structure

The N-Queen lattice can also be used recursively to segment a frame into hierarchical layers. Each layer consists of m blocks of equal size $(N/m) \times (N/m)$. As shown in Fig. 3, we can select these m blocks using one of the sampling lattices at each

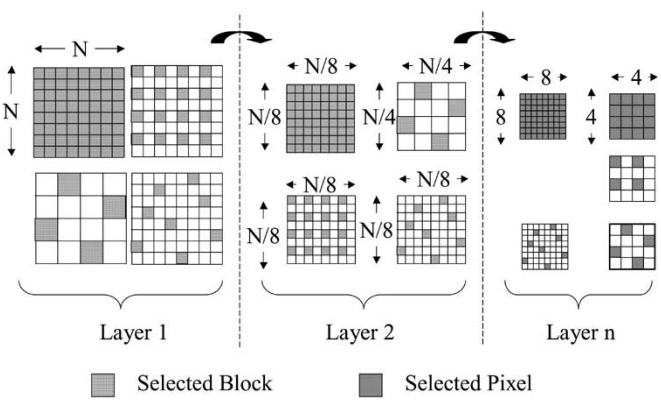


Fig. 3. Recursive structure of pixel decimation using various patterns at each layer.

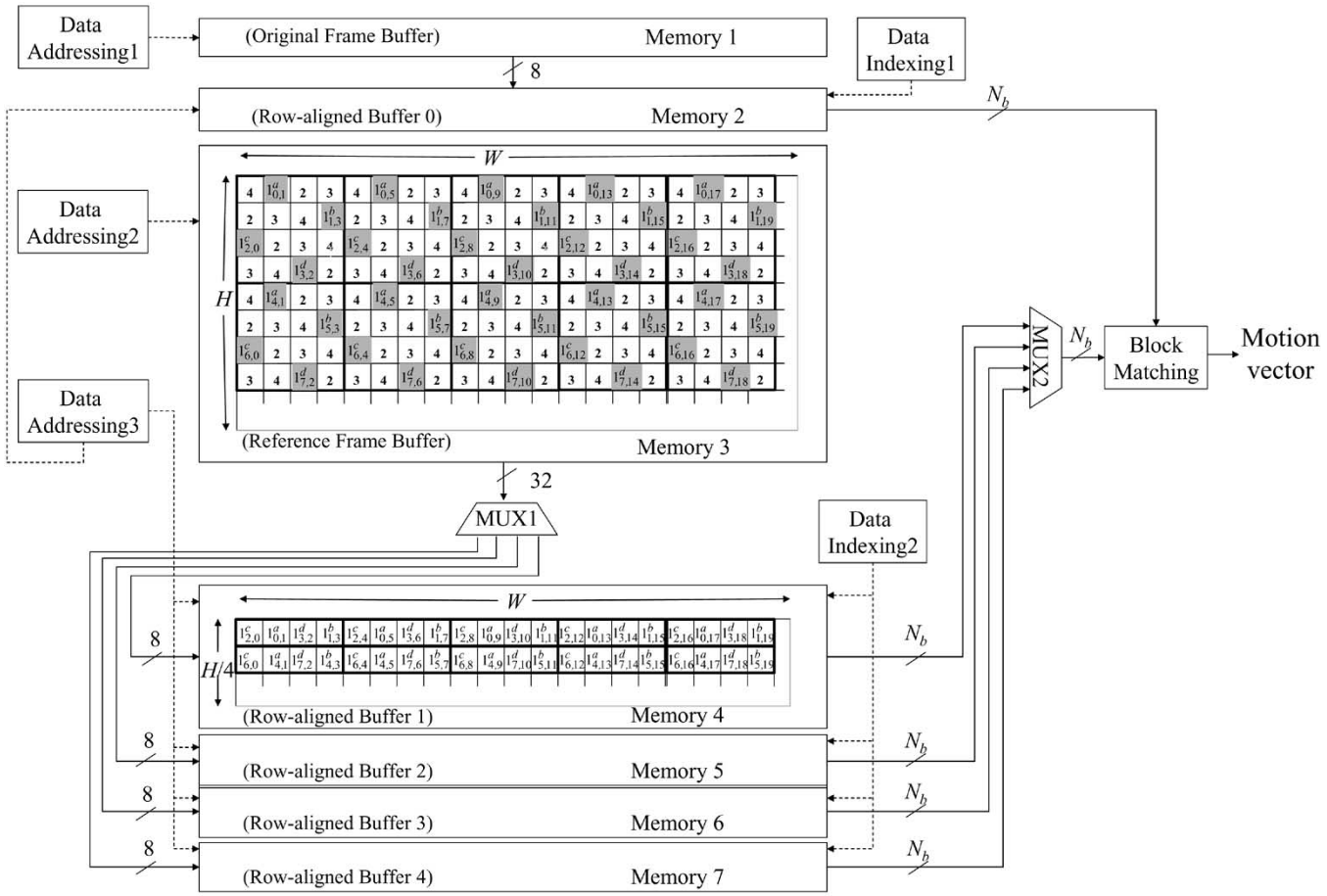


Fig. 4. Architecture example of the 4-Queen pattern motion estimation. For the compact storage, the row-alignment approach [10], [11] for transforming a two-dimensional 4 × 4 block in Memory 3 into a one-dimensional vector of 4 pixels in Memory 4 is illustrated.

layer except for layer n. For each selected block, the same pixel decimation process can be applied recursively. A combination of various lattices offers great flexibility in performing motion estimation especially when global motion estimation at a frame level is considered. In addition, the N-Queen lattice is applicable for motion search using nonsquare blocks, which are used in H.26L (H.264 or Advanced Video Coding) [14]. To fit variable block size motion search in H.26L, multiple small 4-Queen patterns are tiled to construct the final pattern.

III. Hardware Architecture: Case Study

The proposed motion estimation architecture using the N-Queen lattice is shown in Fig. 4. The architecture based on the N-Queen lattice consists of three parts: the reshuffling, data accessing, and pipelined matching modules. In the reshuffling module, we rearrange the pixels into smaller buffers for compact storage. Prior to the matching process, the pointers that store the starting addresses of the relevant pixels are computed and initialized for all buffers. With the starting addresses, we can overlap the memory access and block-matching pipelines. In this paper, we use the 4-Queen lattice to explain the architecture implementation based on the N-Queen lattice. A case study of the practical realization that adopts the Intel SIMD architecture using MMX technology [12] is provided for facilitating the speedup by parallel processing using N-Queen lattices.

A. Compact Storage

For each $N \times N$ block within a macroblock, the N-Queen subsampling process transforms a two-dimensional (2-D) $N \times N$ block to a one-dimensional (1-D) vector of N pixels in a sequential manner, which is desirable for both the software and hardware implementation. The row or column alignment approaches are used to transform a 2-D $N \times N$ block into a 1-D vector of N pixels. For the N-Queen lattice, the row alignment approach moves the selected pixels into a 1-D row vector and the column alignment approach moves the selected pixels into a 1-D column vector. A 4-Queen motion estimation architecture with the compact storage through the row alignment approach [10], [11] is shown in Fig. 4. The column alignment approach can be constructed similarly with this architecture. Applying either alignment approach periodically to every block in a raster scanning order, we can group the pixels that are located at nonoverlapping search points within a frame together. Consequently, the pixels of a frame can be split into N groups using either the row or column alignment approach for advanced processing.

To minimize the memory access bandwidth, a group number (1–4) is used to index each group of pixels that are placed in

a separate memory buffer, as shown in Fig. 4. There is a separate frame memory buffer allocated for each of the N groups based on the N-Queen lattice. For example, the 4-Queen lattice stores the nonoverlapping search positions pixels in four smaller buffers.

One of the special properties of this storage technique is that a macroblock resides in a continuous memory space for easy access. If we use a pipelined memory access strategy, a shift of one pixel in each frame buffer represents a spatial shift in $log(N)$ pixels in the original frame. Thus, this data storage architecture can facilitate a $log(N)$ search strategy easily. Another interesting observation is that each pixel is sequentially accessible even though the search strategy is hierarchical. This provides an elegant solution to improve both search strategy and memory access.

B. Data Access

In each buffer, the selected pixels can be accessed in two steps. The first step is to compute the buffer indices and the second step is to compute the starting addresses of the pixels that will be accessed first. In the first step, the buffer index can be retrieved from a lookup table. The table is predetermined based on the specific N-Queen patterns. For example, the 4×4 block at the upper left corner in the reference frame buffer of Fig. 4 is used to construct such an index table defined as follows:

$$I[4][4] = \{\{4,1,2,3\},\{2,3,4,1\},\{1,2,3,4\},\{3,4,1,2\}\}. \tag{2}$$

The buffer index is computed as

$$T(x,y)_k = I\left[(x + Q[k][0]) \bmod 4\right]\left[(y + Q[k][1]) \bmod 4\right] \tag{3}$$

where k is the index of the Nth group with $k = 0,\ldots,3$ for the 4-Queen case. The symbol "mod" denotes the arithmetic modulo operation. Let the coordinate of the pixel on the upper left corner of a frame be (0, 0) and the coordinates (x, y) represent the offset from (0, 0). The entries of the matrix $Q[.][.]$ represent the offsets of the selected pixels from the position (0, 0) of the 4×4 subblock on the upper left corner as shown in the reference frame buffer of Fig. 4. The matrix of the offsets is defined as

$$Q[4][2] = \{\{0,1\},\{1,3\},\{2,0\},\{3,2\}\}. \tag{4}$$

For each row of the matrix, the first number represents the offset in the vertical direction and the second number means the offset in the horizontal direction. In the second step, the starting addresses for the pixels of the indexed buffer $T(x,y)_k$ are computed as follows. For the column- and row-alignment buffers, the addresses are computed as

$$row_k = (x + Q[k][0]) \tag{5}$$

$$col_k = \left\lfloor \frac{(y + Q[k][1])}{4} \right\rfloor \tag{6}$$

and

$$row_k = \left\lfloor \frac{(x + Q[k][0])}{4} \right\rfloor \tag{7}$$

$$col_k = (y + Q[k][1]) \tag{8}$$

respectively.

The symbol row_k denotes the row number within the kth aligned memory buffer and the symbol col_k denotes the column number in the same buffer, where the value of k ranges from 0 to 3 for the 4-Queen pattern.

C. Parallel Block Matching

Based on our storage architecture, the continuous pixels in a row can be moved in a batch fashion from the buffer into a wide register of multiple bytes depending on the specific processor used. For example, two registers of 64 b can store four pixels from the current and reference blocks. Upon completion of the distortion computation, the reference register only needs to access next 64 b that are sequentially located to implement a shift of four pixels in search points. Thus, our data storage structure can easily support pipelined memory access and the register with the four pixels can be processed in parallel. With the sequential nature of the storage, the data retrieval and the matching can be overlapped in a pipelined fashion.

A parallel matching algorithm is presented for a group of the pixels within the search window. Assume that the search range is 32×32 and the block size is 16×16. To sequentially perform block matching, the SAD of each search point is derived in a row-first manner. Assume that several 64-b or 32-b registers are available and each pixel is stored in 16-b-wide register. Using a 64-b register, four pixels are retrieved sequentially from the same row or the same column within the search window and then the matching are simultaneously performed. In the row-aligned buffer 1 within Memory 4 of Fig. 4, the 16 pixels starting from the coordinate (0, 0) at the same row can be placed into multiple 64-b registers, where each register contains four pixels. To process the next search point starting at (0, 4), the contents inside the overlapping three registers can be kept and the content of the first register is updated with the next successive four pixels with the starting coordinate (0, 16) at the same row. The same procedures can be employed for each row within the search window. Similarly, we can transpose the column-aligned buffers into the row-aligned buffers and identical method can be used for the row-aligned buffers. Both the row-aligned and the column-aligned buffers have high reusability and no overhead for collecting pixels in each step. Thus, either approach can provide a pipelined memory access for parallel block matching.

In Fig. 4, we illustrate the architecture for compact storage, data access, and parallel block matching. For compact storage, the pixels from the original and reference frame buffers are reshuffled into the row-aligned buffers 0–4 separately. The reshuffled location for each pixel is controlled by the three modules, namely Data Addressing 1–3. To efficiently move the 4-Queen sampled pixels within the reference frame, a 32-b-wide bus is connected to each row of Memory 3. Subsequently, the pixels with the same buffer index are moved to the specified row-aligned buffer via a multiplexer MUX1. As shown in the block matching module, the number of bits N_b per data fetch varies among various architectures. A large N_b can increase the execution speed by reducing the time in computing the distortion in the parallel block matching module. The pixels from the current frame and the pixels from the reference frame are retrieved by the module Data Indexing 1 and 2, respectively. Since there are four aligned buffers from the reference frame, a multiplexer MUX2 is used to select the pixels from one of the four buffers for block matching.

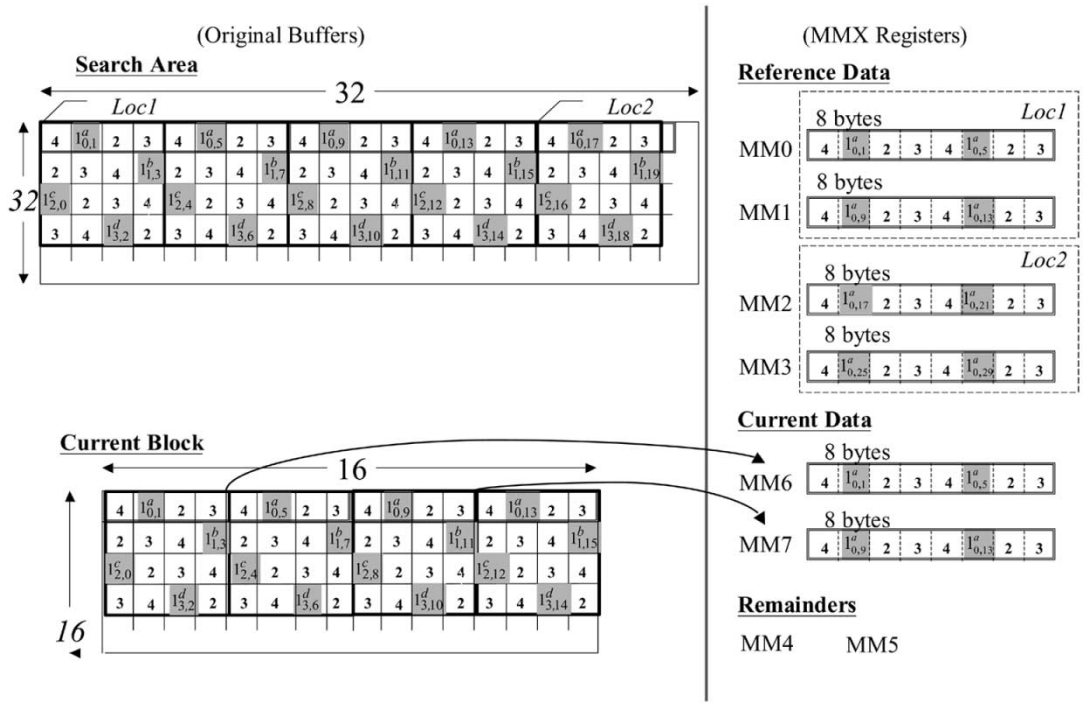


Fig. 5. Illustration of data loading into the MMX registers from the original buffers for the Full pattern block matching. The Registers MM0-MM3 are used for storing the reference data that from the search area. The current data is uploaded into Registers MM6 and MM7. The remaining registers are used for intermediate results.

D. MMX Pipelined Implementation for Full Search

We now describe an implementation for the full search, which will be used to compare against the case for the 4-Queen pattern. Assume the search range is 32 $\times$ 32 and the block size is 16 $\times$ 16. The absolute value of the difference signal can be stored in 8 b. For the MMX SIMD architecture [12], each 64-b register can store eight pixels.

As shown in Fig. 5, the Full pattern block matching uses a 16 $\times$ 16 block where each row has 16 pixels stored in 8-b bins. Two MMX registers are used to store pixels from a row of the target block and four MMX registers are used to store the 32 pixels of the row from the reference frame. The first eight pixels of the row from the target block are loaded in Register MM6 and the remaining pixels in the same row are stored in Register MM7. Each group of nonoverlapping eight pixels from the row within the search area is simultaneously loaded into four MMX Registers MM0-MM3. The other registers are reserved for temporary storage. The parallel computation of the SAD's is performed in a row-by-row fashion for each block.

E. MMX Pipelined Implementation for 4-Queen Search

Now we introduce the block-matching algorithm and the pipelined scheme based on a 4-Queen pattern. Since the row- and column-alignment approaches are similar, our discussions only focus on the improved parallel block matching using the column-aligned buffer. As shown in Fig. 6, the block matching using a 4-Queen pattern adopts a 16 $\times$ 4 block that has four pixels in each row, where each pixel is stored in an 8-b bin. To minimize the memory access, we take one MMX register to store the pixels at every row of the current block and their duplicated pixels. We use four MMX registers to store the 32 pixels from the reference frame. Initially, we load the successive 32 pixels into the four MMX Registers MM0-MM3 and compute the two SADs concurrently for these 32 pixels. Next, we move to the next location in the same row within the search area by right-shifting one pixel and loading the next consecutive 32 pixels into the MMX registers. The same steps are repeated until all of the search positions are visited. The remaining registers are reserved for temporary storage. Consequently, the parallel computation of the SADs is performed in a row-by-row manner for each block.

For each block, we use the table lookup approach to compute the buffer index to retrieve the pixels. The table of indexing buffer is derived from (3), where the arithmetic modulo operation can be implemented as a logical AND ("&") operation with reduced complexity as follows:

$$T(x,y)_k = I\left[(x+Q[k][0])\ \&0x3\right]\left[(y+Q[k][1])\ \&0x3\right]. \quad (9)$$

Thus, the operation "mod 4" is replaced by the operation AND with a constant of value 3. The indices can be found by directly searching the lookup tables as illustrated in Table II. Each indexing approach needs $(2 \times 2 + N_s)$ operations, where the symbol N_s denotes the total number of operations required for loading the data from memory according to the MMX technology [12]. In addition, we need to compute the starting addresses for loading the reference data into the MMX registers prior to blocking matching. The starting addresses at the specified buffer are defined in (7) and (8) for column-aligned buffers, which can be realized by replacing the division with a right-shift ($\gg$) operation as follows:

$$col_k = \lfloor (x + Q[k][0]) \gg 2 \rfloor \quad (10)$$

$$row_k = (y + Q[k][1]). \quad (11)$$

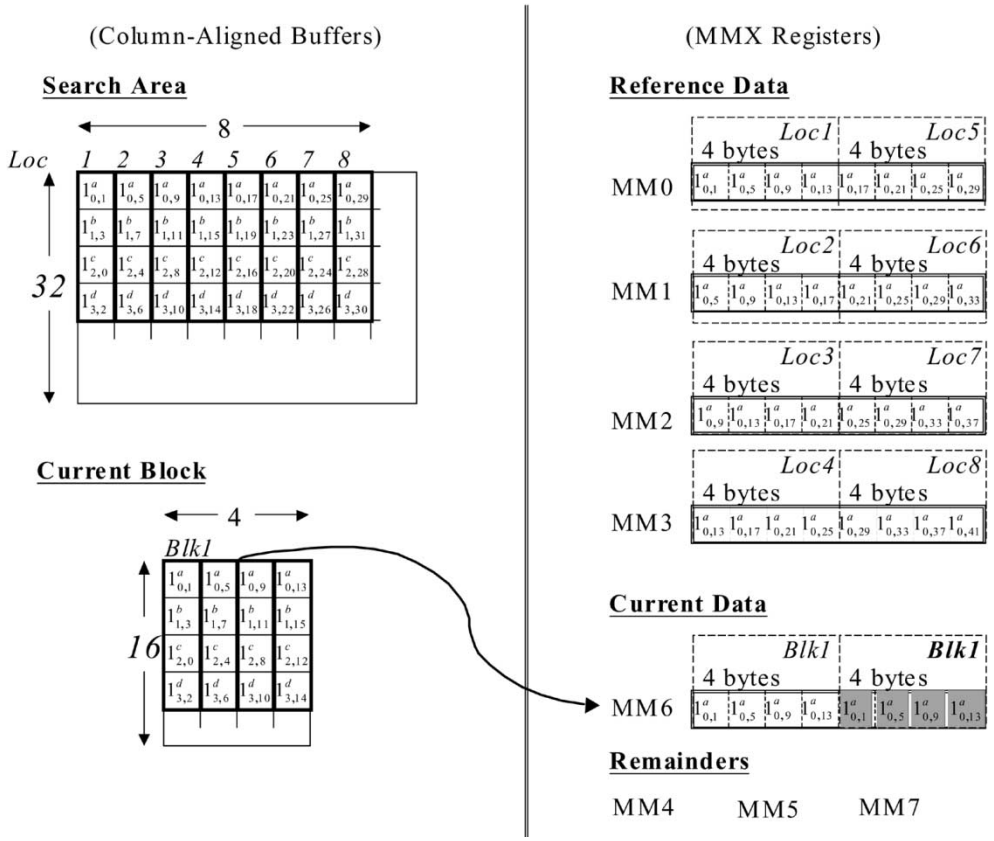


Fig. 6. Illustration of data loading from the column-aligned buffers into the MMX registers for the 4-Queen pattern blocking matching. The Registers MM0-MM3 are used for storing the reference data from the search area. The current data is loaded into the Register MM6. The remaining registers are used for intermediate results.

TABLE II
BUFFER INDEXING FOR COLUMN-ALIGNED STORAGE. THE COORDINATE (x, y) INDICATES THE POSITION OF THE UPPER LEFT CORNER OF THE BLOCK

y	$T(x, y)_k$			
	k=0	*k=1*	*k=2*	*k=3*
4*v*	1	1	1	1
4*v* +1	2	2	2	2
4*v* +2	3	3	3	3
4*v* +3	4	4	4	4

(a) (x, y)=(4*u*, :)

y	$T(x, y)_k$			
	k=0	*k=1*	*k=2*	*k=3*
4*v*	3	4	3	2
4*v* +1	4	1	4	3
4*v* +2	1	2	1	4
4*v* +3	2	3	2	1

(b) (x, y)=(4*u*+1, :)

y	$T(x, y)_k$			
	k=0	*k=1*	*k=2*	*k=3*
4*v*	2	2	4	4
4*v* +1	3	3	1	1
4*v* +2	4	4	2	2
4*v* +3	1	1	3	3

(c) (x, y)=(4*u*+2, :)

y	$T(x, y)_k$			
	k=0	*k=1*	*k=2*	*k=3*
4*v*	4	3	2	3
4*v* +1	1	4	3	4
4*v* +2	2	1	4	1
4*v* +3	3	2	1	2

(d) (x, y)=(4*u*+3, :)

Thus, the address computation needs $(2 + 1)$ operations.

F. Computational Complexity

The computational complexity is compared based on the total operations used per block. Thus, for an $N \times N$ block and an $M \times M$ search area, the pipelined block matching using the Full pattern takes

$$O_{full} = \left\{2 + [4 + 4 * 5 + 2 * 20 + N_s] * \left(\frac{M}{2}\right)\right\} * N * M \tag{12}$$

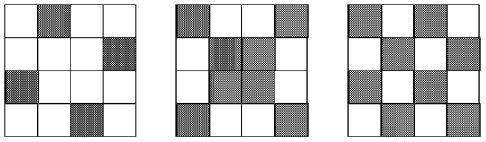
operations per macroblock in total.

For the 4-Queen pattern using the column-aligned storage, the migration of the pixels from the frame buffers to the aligned buffers for both the current and reference images needs

$$O_{\text{alignment}} = 1\frac{1}{4} * N^2 + 2 * (N_s - 1) \tag{13}$$

TABLE III
CYCLE COUNTS FOR THE BLOCK MATCHING USING THE FULL PATTERN AND THE 4-QUEEN PATTERN

Methods	Full pattern	4-Queen pattern	Speedup
Ns=4	558080	164864+5632+326=170822	3.27
Ns=2	541696	164864+4608+322=169794	3.19



(a) 4-Queen (b) A (c) B

Fig. 7. Decimation patterns at the block layer. The marked block means the selected 4×4 block.

Fig. 8. Reconstructed Y components for the 102nd frame of the Foreman sequence in CIF format. (a) The original frame. (b)–(d) Constructed pictures with Full Search and N-Queen patterns.

operations. With the aligned buffer, the block matching takes

$$O_{\mathrm{SAD}} = \{(1+1) + [4 + 4 * (5 + 12 + 2)] * 4\} * N * M \quad (14)$$

operations. Additional buffer address computation and memory access require

$$O_{\mathrm{index}} = \{[(2 \times 2 + N_s) + (2+1)] * 4\} * \frac{N}{4} * M \quad (15)$$

operations for each marcoblock. Consequently, the overall computational complexity for the pipelined block matching using a 4-Queen pattern is $O_{4-\mathrm{Queen}} = O_{\mathrm{alignment}} + O_{\mathrm{SAD}} + O_{\mathrm{index}}$. The comparison of the computational costs for the pipelined schemes using the Full and 4-Queen patterns is shown in Table III, where the block size is 16 × 16 and the search area is 32 × 32. Based on the MMX technology, the 4-Queen scheme has improved the motion estimation by about 3.2 times in speed. The reason for not achieving the theoretic improvement of four times comes from the fact that extra buffer indexing and pixel address computations are required.

G. Memory Bandwidth

In this section, we analyze total memory bandwidth for loading the current and reference frames. For the Full pattern, the total bandwidth consumption in bytes per second is

$$\zeta_{\mathrm{full}} = (\zeta_{\mathrm{full\_cur}} + \zeta_{\mathrm{full\_sa}}) \times \mathrm{Fps} \quad (16)$$

where the symbol $\zeta_{\mathrm{full\_cur}}$ denotes the bandwidth to access the data from the current frame and $\zeta_{\mathrm{full\_sa}}$ is the bandwidth for loading the data from the search window of the reference frame. Let the symbol Fps be the frame rate. Assume that the processing pixels of the current block are loaded simultaneously into an on-chip memory with $N \times N$ data bins. Thus, to access the data from the current frame of size $H \times W$ requires

$$\zeta_{\mathrm{full\_cur}} = H \times W \quad (17)$$

bytes.

For an $N \times N$ block and a $2M \times 2M$ search area, assume the reference data are simultaneously loaded into the on-chip memory with size $(N + 2M) \times (N + 2M)$. Since the overlapped data can be reused for the next location, only partial data are reloaded. Specifically, there are $N(N + 2M)$ data need to be updated. Thus, the memory consumption for storing the reference data from the whole search area needs additional

$$\zeta_{\mathrm{full\_sa}} = \frac{H}{N} \times \left[(N+2M)^2 + \left(\frac{W}{N} - 1 \right) \times (N + 2M) \times N \right] \quad (18)$$

bytes. The first term in the parenthesis accounts for the first search point, which takes more operations due to the memory stall in the initialization stage of pipelining. Less access is needed since the neighboring search windows overlap in area.

For the proposed 4-Queen algorithm, the total memory bandwidth (in bytes) is

$$\zeta = (\zeta_{\mathrm{cur}} + \zeta_{\mathrm{sa}}) \times \mathrm{Fps}. \quad (19)$$

Obviously, only a quarter of the reference and current data are needed for block matching. Thus,

$$\zeta = \frac{\zeta_{\mathrm{full}}}{4}. \quad (20)$$

TABLE IV
PERFORMANCE OF THE FOUR PIXEL PATTERNS AND RECURSIVE STRUCTURE, THE THREE SEARCH STRATEGIES FOR VARIOUS VIDEO SEQUENCES UNDER DIFFERENT TESTING CONDITIONS

Sequence	*Format*	*BR*	*Fps*	*SA*	*Methods*	*PSNRY*	*PSNRU*	*PSNRV*	*Δ PSNRY*	*ChkPt*	*Pixels*	*Ratio*
News	CIF	112	15	16	Full Search	34.06	38.08	39.02		60420096	256	1
					Full_Qu	34.05	38.01	38.94	-0.01	60420096	128	2
					Full_Q	33.95	37.87	38.82	-0.11	60420096	64	4
					Full_H	33.99	37.89	38.86	-0.07	60420096	64	4
					Full_4	34.02	37.94	38.90	-0.04	60420096	64	4
					Full_Rt	33.99	38.00	38.96	-0.07	60420096	64	4
					Full_Yu	33.99	37.86	38.80	-0.07	60420096	64	4
					Full_8	33.85	37.69	38.71	-0.21	60420096	32	8
					MVFAST_F	33.95	37.97	38.96	-0.11	263681	256	229.14
					MVFAST_Qu	33.91	38.02	39.01	-0.15	264539	128	456.80
					MVFAST_Q	33.88	37.95	38.91	-0.18	258705	64	934.19
					MVFAST_H	33.96	38.04	38.99	-0.10	264009	64	915.42
					MVFAST_4	33.97	38.05	38.97	-0.09	259359	64	931.83
					MVFAST_Rt	33.94	38.12	39.10	-0.12	261529	64	924.11
					MVFAST_Yu	33.93	38.03	38.95	-0.13	261222	64	925.19
					MVFAST_8	33.85	37.96	38.89	-0.21	257952	32	1873.84
					PMVFAST_F	33.89	37.99	38.94	-0.17	181338	256	333.19
					PMVFAST_Qu	33.91	38.00	38.95	-0.15	190347	128	634.84
					PMVFAST_Q	33.87	38.04	38.94	-0.19	178329	64	1355.25
					PMVFAST_H	33.94	38.03	38.98	-0.12	180333	64	1340.19
					PMVFAST_4	33.95	38.02	39.00	-0.11	177280	64	1363.27
					PMVFAST_Rt	33.93	38.01	38.96	-0.13	178563	64	1353.47
					PMVFAST_Yu	33.92	38.07	39.05	-0.14	177955	64	1358.10
					PMVFAST_8	33.83	37.94	38.93	-0.23	177968	32	2715.99
Foreman	CIF	112	10	16	Full Search	30.05	36.73	37.49		40144896	256	1
					Full_Qu	29.99	36.75	37.50	-0.06	40144896	128	2
					Full_Q	29.51	36.47	37.30	-0.54	40144896	64	4
					Full_H	29.69	36.68	37.44	-0.36	40144896	64	4
					Full_4	29.79	36.70	37.49	-0.26	40144896	64	4
					Full_Rt	29.65	36.57	37.28	-0.40	40144896	64	4
					Full_Yu	29.68	36.66	37.40	-0.37	40144896	64	4
					Full_8	29.33	36.43	37.06	-0.72	40144896	32	8
					MVFAST_F	29.89	36.88	37.66	-0.16	473299	256	84.82
					MVFAST_Qu	29.92	36.92	37.76	-0.13	469747	128	170.92
					MVFAST_Q	29.61	36.80	37.72	-0.44	446359	64	359.75
					MVFAST_H	29.73	36.87	37.81	-0.32	452763	64	354.67
					MVFAST_4	29.83	36.82	37.73	-0.22	453247	64	354.29
					MVFAST_Rt	29.73	36.92	37.80	-0.32	444403	64	361.34
					MVFAST_Yu	29.76	36.90	37.81	-0.29	452431	64	354.93
					MVFAST_8	29.60	36.80	37.66	-0.45	444744	32	722.12
					MVFAST_A4	29.56	36.87	37.69	-0.49	417201	32	769.79
					MVFAST_B4	29.51	36.79	37.63	-0.54	450904	32	712.26
					MVFAST_4R	29.06	36.56	37.41	-0.99	447622	16	1434.96
					PMVFAST_F	29.97	36.94	37.76	-0.08	378736	256	105.99
					PMVFAST_Qu	29.94	36.95	37.78	-0.03	374847	128	214.19
					PMVFAST_Q	29.66	36.83	37.71	-0.39	364766	64	440.22
					PMVFAST_H	29.75	36.94	37.82	-0.30	364512	64	440.53
					PMVFAST_4	29.88	36.91	37.82	-0.17	366304	64	438.38
					PMVFAST_Rt	29.80	39.96	37.86	-.025	357737	64	448.88
					PMVFAST_Yu	29.81	36.97	37.76	-0.25	365998	64	438.74
					PMVFAST_8	29.65	36.71	37.65	-0.40	364337	32	881.49
					PMVFAST_4R	29.10	36.54	37.35	-0.85	374118	16	1716.89
					PMVFAST_A4	29.63	36.85	37.65	-0.42	364845	32	880.26
					PMVFAST_B4	29.57	36.68	37.56	-0.48	370089	32	867.79

For each method, the first symbol denotes the search strategy and the remaining symbols denote the sampling patterns. For example, the notation "PMVFAST_8" represents the motion estimation using the PMVFAST approach and 8-Queen pattern. "BR" indicates the bit rate in kb/s, the frame rates are represented in frames per second "Fps", and the search range is denoted as "SA". The column "PSNRY" denotes the average PSNR for the luminance component and the "ChkPt" indicates the actual number of search points used. The "Pixels" means the number of pixels per search point. The final column "RatioO" is the improvement in speed as compared to the full search.

IV. EXPERIMENTAL RESULTS

In our simulation, we use the MPEG-4 reference software [15] and the distortion measure is SAD. Only the 16 × 16 motion vector mode is enabled for demonstrating the performance. The coding efficiency is analyzed based on the three parameters: sampling patterns, search strategies, and testing conditions.

TABLE IV *(Continued.)*
PERFORMANCE OF THE FOUR PIXEL PATTERNS AND RECURSIVE STRUCTURE, THE THREE SEARCH STRATEGIES FOR VARIOUS VIDEO SEQUENCES UNDER DIFFERENT TESTING CONDITIONS

Stefan	CCIR 601	1M	10	16	Full Search	26.43	32.60	32.87		136857600	256	1
					MVFAST_F	26.58	33.07	33.38	0.15	1584656	256	86.36
					MVFAST_Qu	26.60	33.10	33.44	0.17	1446579	128	189.22
					MVFAST_Q	26.38	32.87	33.18	-0.05	1376083	64	397.82
					MVFAST_H	26.60	33.19	33.52	0.17	1446944	64	378.34
					MVFAST_4	26.60	33.13	33.48	0.17	1397127	64	391.83
					MVFAST_Rt	26.60	33.18	33.54	0.17	1418394	64	385.95
					MVFAST_Yu	26.60	33.15	33.50	0.17	1406342	64	389.26
					MVFAST_8	26.47	32.99	33.30	0.04	1365344	32	801.89
					MVFAST_A4	26.46	32.99	33.29	0.03	1375583	32	795.92
					MVFAST_B4	26.43	32.95	33.25	0	1384446	32	790.83
					MVFAST_4R	26.24	32.76	33.02	-0.19	1385464	16	1580.50
					PMVFAST_F	26.57	33.03	33.37	0.14	1414828	256	96.73
					PMVFAST_Qu	26.58	33.08	33.40	0.15	1295846	128	211.23
					PMVFAST_Q	26.35	32.84	33.13	-0.08	1222771	64	447.69
					PMVFAST_H	26.57	33.16	33.49	0.14	1288854	64	424.74
					PMVFAST_4	26.58	33.10	33.43	0.15	1251042	64	437.58
					PMVFAST_Rt	26.57	33.16	33.50	0.14	1260784	64	434.20
					PMVFAST_Yu	26.59	33.13	33.46	0.16	1256161	64	435.80
					PMVFAST_8	26.46	32.98	33.28	0.03	1227809	32	891.72
					PMVFAST_A4	26.44	32.95	33.26	0.01	1235739	32	885.99
					PMVFAST_B4	26.43	32.93	33.26	0	1240245	32	882.78
					PMVFAST_4R	26.23	32.73	33.00	-0.20	1244059	16	1760.14
Stefan	CCIR 601	1M	10	32	Full Search	27.11	33.05	33.43		547430400	256	1
					MVFAST_F	27.13	33.47	33.92	+ 0.02	1603677	256	341.36
					MVFAST_Qu	27.08	33.50	33.93	-0.03	1455269	128	752.34
					MVFAST_Q	26.72	33.15	33.51	- 0.39	1375883	64	1591.50
					MVFAST_H	26.98	33.48	44.89	-0.13	1461897	64	1497.86
					MVFAST_4	27.00	33.48	33.80	-0.11	1398414	64	1568.86
					MVFAST_Rt	26.99	33.49	44.91	-0.12	1430278	64	1530.98
					MVFAST_Yu	27.00	33.46	33.90	-0.11	1411268	64	1551.60
					MVFAST_8	26.77	33.25	33.62	- 0.34	1369241	32	3198.45
					MVFAST_4R	26.44	32.93	33.24	-0.67	1386207	16	6318.60
					MVFAST_A4	26.73	33.23	33.59	- 0.38	1375583	32	3170.13
					MVFAST_B4	26.70	33.18	33.54	- 0.41	1384446	32	73154.81
					PMVFAST_F	27.17	33.50	33.94	+ 0.06	1425441	256	384.04
					PMVFAST_Qu	27.19	33.53	33.97	+0.08	1299292	128	842.66
					PMVFAST_Q	26.75	33.16	33.52	- 0.36	1226344	64	1785.57
					PMVFAST_H	26.98	33.48	33.89	-0.13	1289554	64	1698.05
					PMVFAST_4	27.00	33.48	33.90	- 0.11	1253491	64	1746.90
					PMVFAST_Rt	26.99	33.49	33.91	-0.12	1257592	64	1741.20
					PMVFAST_Yu	27.00	33.46	33.90	-0.11	1262730	64	1734.12
					PMVFAST_8	27.02	33.41	33.85	- 0.09	1221419	32	3585.54
					PMVFAST_4R	26.73	33.17	33.53	-0.38	1237161	16	7079.83
					PMVFAST_A4	27.00	33.41	33.83	- 0.11	1235739	32	3563.83
					PMVFAST_B4	27.00	33.40	33.81	- 0.11	1240245	32	3557.70

For each method, the first symbol denotes the search strategy and the remaining symbols denote the sampling patterns. For example, the notation "PMVFAST_8" represents the motion estimation using the PMVFAST approach and 8-Queen pattern. "BR" indicates the bit rate in kb/s, the frame rates are represented in frames per second "Fps", and the search range is denoted as "SA". The column "PSNRY" denotes the average PSNR for the luminance component and the "ChkPt" indicates the actual number of search points used. The "Pixels" means the number of pixels per search point. The final column "RatioO" is the improvement in speed as compared to the full search.

As for the sampling patterns, we use eight patterns as described in Fig. 1. The Full pattern ("F") selects all pixels in the current block. The Quarter pattern ("Q") is described in [3] and Quincunx pattern ("Qu") is from [9]. The Hexagonal pattern ("H") and the Yu's pattern ("Yu") are described in [4] and [5], respectively. The 4-Queen ("4") pattern is constructed by tiling multiple small 4-Queen patterns for each 16 × 16 macroblock. The 8-Queen ("8") pattern and Rectangular ("Rt") pattern are similarly tiled. Additionally, in order to examine the performance of the pixel decimation using recursive structure, Fig. 7 demonstrates three patterns used for two-layer recursive sampling. The two-layer recursive scheme ("4R") employs the same 4-Queen pattern at both the block and pixel layers. The second two-layer recursive sampling approach ("A4") adopts the pattern "A" and the last approach ("B4") takes the pattern "B" in Fig. 7 at the block layer. At the pixel layer, the 4-Queen pattern is applied for each subblock.

As for the search strategies, we tested the full search and two approaches adopted by the MPEG-4 committee. These approaches are often referred to as motion vector field adaptive

search technique (MVFAST) [1] and predictive MVFAST (PMVFAST) [2]. The Y component of the 102nd reconstructed frame of the Foreman sequence encoded using Full search with N-Queen decimation patterns and constant quantization is illustrated in Fig. 8. The bit rate is around 125 kb/s at 10 Hz. The quantization parameter (QP) of the intracoded video object plane (I-VOP) is 16. The QPs of the predicted video object planes (P-VOPs) are 16, 17, and 18 for the Full, 4-Queen, and 8-Queen patterns, respectively. The respective average PSNRs are 31.77, 31.33, and 30.93 dB.

We follow the recommended testing conditions as prescribed by the MPEG committee [16]. As shown in Table IV and our previous work presented in [10] and [11], we reach the following conclusions.

1) The N-Queen patterns have negligible video quality degradation. With PMVFAST, the loss in PSNR is less than 0.23 dB for slow motion video such as the "News" sequence. The loss in PSNR is less than 0.45 dB at worst for fast motion video. The 4-Queen pattern is better than the Quarter, Quincunx, Rectangular, Yu's, and Hexagonal patterns by about 0.01–0.28 dB for the noninterlaced coded sequences.
2) When various search strategies are compared for the same pattern, the degradation using Full Search is more for the N-Queen, Quincunx, and Quarter patterns. Such a phenomenon may be caused by the predictive vector technique used by MVFAST and MVFAST that yield smoother vector fields.
3) It is advantageous to use the recursive structure "4R" for larger picture sizes such as CCIR-601 format because the 16×16 block corresponds to 32×32 block at CCIR-601 resolution. For the same content, there is less spatial luminance variation at larger picture sizes.
4) Observing the performance of the two-layer subsampling approaches, namely "A4" and "B4," the recursive structures of pixel decimation perform better when the picture size is increased to a format such as CCIR-601 format. For the video with higher resolution such as the "Stefan" sequence, the worst-case loss in PSNR is less than 0.41 dB. The improved performance of the two-layer decimation for larger frame sizes may be attributed to the stronger spatial correlations at larger picture sizes.
5) When we compare the 4-Queen pattern and the Quincunx pattern, which have the same spatial homogeneity but different directional coverage, as shown in Table I, the 4-Queen pattern has about twice the speedup than the Quincunx pattern with a video quality degradation of less than 0.2 dB in PSNR. Such a result shows that the directional coverage of the sampling lattice is useful.

V. Conclusion

This paper has presented a novel and simple pixel decimation technique using an N-Queen lattice with an application for block-based motion estimation. The complexity and memory bandwidth can be arbitrarily reduced by a factor of N. It is superior in terms of spatial homogeneity and directional coverage.

It is advantageous that such a randomized lattice can be stored compactly with a sequential pipelined buffer access as demonstrated in this paper. In our approach, $log(N)$ search can be implemented sequentially. The recursive nature of the N-Queen sampling lattice is flexible when the block size is variable including a full picture size and nonsquare block for applications such as global motion estimation and sprite generation. With improvement in speed by a factor of N, our experimental results show no significant degradation in PSNR and have consistent performance for extensive tests as recommended by the MPEG committee.

Acknowledgment

The authors wish to thank the anonymous reviewers for their insightful comments to improve the initial draft of this paper.

References

[1] A. M. Tourapis, O. C. Au, M. L. Liou, G. Shen, and I. Ahmad, "Optimizing the MPEG-4 encoder- advanced diamond zonal search," in *Proc. 2000 Int. Symp. Circuits Systems*, vol. 3, Geneva, Switzerland, May 2000, pp. 674–677.

[2] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Trans. Image Processing*, vol. 9, pp. 287–290, Feb. 2000.

[3] M. Bierling, "Displacement estimation by hierarchical block matching," in *Proc. SPIE Conf. Visual Commun. Processing*, vol. 1001, 1988, pp. 942–951.

[4] K. T. Choi, S. C. Chan, and T. S. Ng, "A new fast motion estimation algorithm using hexagonal subsampling pattern and multiple candidate search," in *Proc. IEEE Int. Conf. Image Processing*, 1996, pp. 497–500.

[5] Y. Yu, J. Zhou, and C. W. Chen, "A novel fast block motion estimation algorithm based on combined subsamplings on pixels and search candidates," *J. Vis. Commun. Image Representation*, vol. 12, pp. 96–105, 2001.

[6] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vector," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, pp. 148–157, Apr. 1993.

[7] Y.-L. Chan and W.-C. Siu, "New adaptive pixel decimation for block motion vector estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 113–118, Feb. 1996.

[8] Y. K. Wang, Y. Q. Wang, and H. Kuroda, "A globally adaptive pixel-decimation algorithm for block-motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, pp. 1006–1011, Sept. 2000.

[9] K. Lengwehasatit and A. Ortega, "Probabilistic partial-distance fast matching algorithms f or motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, pp. 139–152, Feb. 2001.

[10] S.-W. Yang, C.-N. Wang, C.-M. Liu, and T. Chiang, "Fast motion estimation using N-Queen pixel decimation," in *Proc. 2nd IEEE Pacific-Rim Conf. Multimedia*, Beijing, China, Oct. 24–26, 2001.

[11] C.-N. Wang, S.-W. Yang, C.-M. Liu, and T. Chiang. (2003, Aug.) A hierarchical decimation lattice based on N-Queen with an application for motion estimation. *IEEE Signal Process Lett.*, pp. 228–231

[12] *Intel Architecture Software Developer's Manual*, vol. 1–3, Intel Corp., 1999.

[13] N. S. Jayant and P. Noll, *Digital Coding of Waveforms—Principles and Applications to Speech and Video*. Englwood Cliffs, NJ: Prentice-Hall, 1984.

[14] *Text of ISO/IEC FDIS 14496-10: Information Technology—Coding of Audio-Visual Objects—Part 10: Advanced Video Coding*, Mar. 2003.

[15] *ISO/IEC 14 496-5:2001/FPDAM1*, July 2001.

[16] *Experimental Conditions for Evaluating Encoder Motion Estimation Algorithms*, Dec. 1999.

Chung-Neng Wang (M'03) was born in PingTung, Taiwan, R.O.C., in 1972. He received the B.S. degree and Ph.D. degree in computer science and information engineering from National Chiao-Tung University (NCTU), Hsinchu, Taiwan, in 1994 and 2003, respectively.

He joined the faculty at NCTU in January 2003. Since 2001, he has actively parcticipated in ISO's Moving Picture Experts Group (MPEG) digital video coding standardization process, and he has made more than 18 contributions to the MPEG committee over the past four years. He has published over 20 technical journal and conference papers in the field of video and signal processing. His current research interests are video/image compression, motion estimation, video transcoding, and streaming.

Shin-Wei Yang was born in TaoYuan, Taiwan, R.O.C., in 1977. He received the B.S. degree in computer science and information engineering and the M.S. degree from National Chiao-Tung University, Hsinchu, Taiwan, in 2000 and 2002, respectively.

He is currently with the Institute for Information Industry, Taipei, Taiwan.

Chi-Min Liu received the B.S. degree in electrical engineering from Tatung Institute of Technology, Taiwan, R.O.C., in 1985 and the M.S. degree and Ph.D. degree in electronics from National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 1987 and 1991, respectively.

He is currently a Professor with the Department of Computer Science and Information Engineering, NCTU. His research interests include video/audio compression, speech recognition, radar processing, and application-specific VLSI architecture design.

Tihao Chiang (S'91–M'95–SM'99) was born in Cha-Yi, Taiwan, R.O.C., in 1965. He received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, in 1987, and the M.S. and Ph.D. degrees in electrical engineering from Columbia University, New York, NY, in 1991 and 1995, respectively.

In 1995, he joined the David Sarnoff Research Center, Princeton, NJ, as a Member of Technical Staff. Later, he was promoted to technology leader and a program manager at Sarnoff. While at Sarnoff, he led a team of researchers and developed an optimized MPEG-2 software encoder. For his work in the encoder and MPEG-4 areas, he received two Sarnoff achievement awards and three Sarnoff team awards. Since 1992, he has actively participated in ISO's Moving Picture Experts Group (MPEG) digital video coding standardization process with particular focus on the scalability/compatibility issue. He is currently the co-editor of part 7 on the MPEG-4 committee. He has made more than 90 contributions to the MPEG committee over the past 11 years. His main research interests are compatible/scalable video compression, stereoscopic video coding, and motion estimation. In September 1999, he joined the faculty at National Chiao-Tung University, Hsinchu, Taiwan. He has published over 50 technical journal and conference papers in the field of video and signal processing. He holds 13 U.S. patents and 30 European and worldwide patents.

Dr. Chiang was a co-recipient of the 2001 best paper award from the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY.