New Fast Binary Pyramid Motion Estimation for MPEG2 and HDTV Encoding

Xudong Song, *Member, IEEE*, Tihao Chiang, *Senior Member, IEEE*, Xiaobing Lee, *Member, IEEE*, and Ya-Qin Zhang, *Fellow, IEEE*

***Abstract*—A novel Fast Binary Pyramid Motion Estimation (FBPME) algorithm is presented in this paper. The proposed FBPME scheme is based on binary multiresolution layers, exclusive-or (XOR) Boolean block matching, and a N*-scale* tiling search scheme. Each video frame is converted into a pyramid structure of $K-1$ binary layers with resolution decimation, plus one integer layer at the lowest resolution. At the lowest resolution layer, the N*-scale* tiling search is performed to select initial motion vector candidates. Motion vector fields are gradually refined with the XOR Boolean block-matching criterion and the N*-scale* tiling search schemes in higher binary layers.**

FBPME performs several thousands times faster than the conventional full-search block-matching scheme at the same PSNR performance and visual quality. It also dramatically reduces the bus bandwidth and on-chip memory requirement. Moreover, hardware complexity is low due to its binary nature.

Fully functional software MPEG-2 MP@ML encoders and Advanced Television Standard Committee High Definition Television encoders based on the FBPME algorithm have been implemented. FBPME Hardware architecture has been developed and is being incorporated into single-chip MPEG encoders. A wide range of video sequences at various resolutions has been tested. The proposed algorithm is also applicable to other digital video compression standards such as H.261, H.263, and MPEG4.

***Index Terms*—Binary pyramid, HDTV, motion estimation, MPEG-2, video compression, XOR Boolean block matching.**

I. Introduction

Motion estimation (ME) and compensation are critical components for digital video compression systems such as High Definition Television (HDTV) and Standard Definition Television (SDTV) broadcasting equipment, video conferencing transmitters, and multimedia servers for Web applications. Block-based ME has been widely adopted by several important international standards such as Motion Picture Expert Group (MPEG), Advanced Television Standard Committee (ATSC), Digital Video Broadcasting (DVB), and International Telecommunications Union (ITU). The relevant standards include ISO/IEC MPEG-1, MPEG-2, MPEG-4, MPEG-7, H.261, H.263, DVB, and ATSC specifications [1]–[5].

The optimal block-based ME is typically defined as the exhaustive search algorithm using minimum absolute difference (MAD) block-matching criterion. The technique evaluates motion vectors for all locations within the search window. Many fast ME algorithms with various sparse searching schemes have been developed to reduce computational complexity. Examples include the three-step search [6], the 2-D logarithmic search [7], the conjugate directional search [8], the genetic search [9], [10], the unrestricted center-biased diamond search [11], the feature-based block ME using integral projection [12], and sub-sampled motion field estimation with alternating pixel-decimation patterns [13]. These sparse searching schemes provided much reduced complexity at the expense of motion accuracy. In addition, fast search schemes are less robust in the sense that they are often trapped into local minimum.

The multi-resolution ME techniques [14]–[18] were developed to search a smaller window from lower to higher resolution layers similar to the hierarchical motion projection search. A small search window at reduced resolution can cover the same area as the full search in the full resolution with reduced complexity. With an efficient hierarchical structure, decimation and filtering algorithms, the multi-resolution ME also provides greater robustness than sparse search algorithms.

To further reduce the computation, binary ME algorithms have been developed. Binary ME algorithms use an exclusive-or (XOR) Boolean block-matching criterion instead of the integer MAD criterion for block matching. The binary representation provides much lower computational complexity and data bus throughputs for the basic block-matching module, [20]–[27]. The binary ME algorithms can be combined with any fast search schemes for increased speed.

In this paper, a new Fast Binary Pyramid ME (FBPME) algorithm is presented, based on a binary pyramid structure, XOR Boolean block matching, and N*-scale* tiling search scheme. The block matching now uses only bit-wise XOR logic operations that are much simpler and faster to implement than integer MAD operations. the n*–scale* tiling scheme provides multi-path hierarchical projection searches from layer to layer to achieve more robust performance. FBPME is about 4000 times faster than the integer MAD full search ME, and 16–256 times faster than the HFM-ME algorithm [21], [22]. It also requires much less data bus bandwidth due to the binary nature of the data transfer between frame and reference buffers.

This paper is organized as follows. Section II discusses previous efforts in binary ME algorithms, and briefly summa-

Manuscript received July 1999; revised May 2000. This paper was recommended by Guest Editor R. Schaefer.

X. Song was with Sarnoff Corporation, CN 5300, Princeton, NJ 08543-5300 USA. He is now with IVAST, San Jose, CA 95110 USA.

T. Chiang was with Sarnoff Corporation CN 5300, Princeton, NJ 08543-5300 USA. He is now with the Department of Electronics Engineering, National Chiao Tung University, Hsin-chu 30050, Taiwan, R.O.C. (e-mail: tchiang@cc.nctu.edu.tw).

X. Lee is with Sarnoff Corporation CN 5300, Princeton, NJ 08543-5300 USA (e-mail: xudong_song@hotmail.com).

Y.-Q. Zhang was with Sarnoff Corporation, Princeton, NJ 08543-5300 USA. He is now with Microsoft Research China, Beijing, China.

Publisher Item Identifier S 1051-8215(00)08199-4.

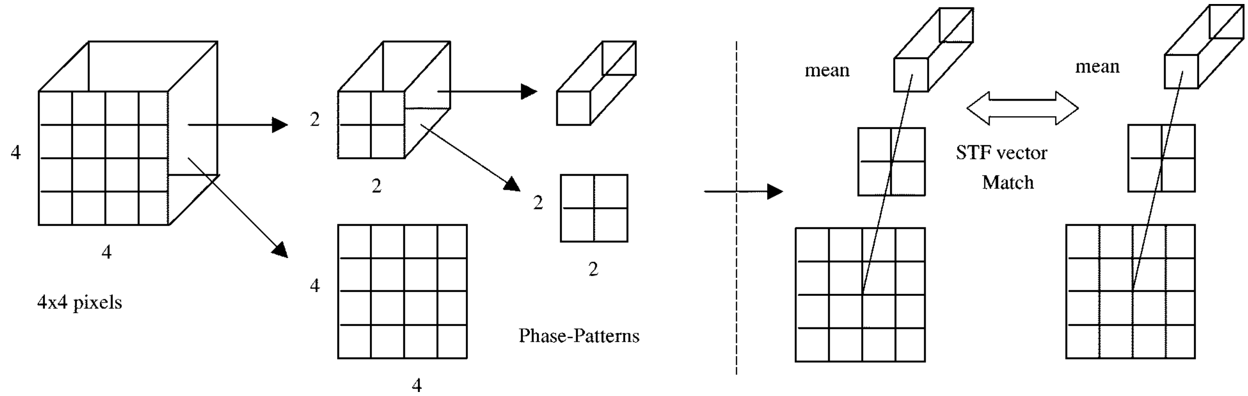

Fig. 1. Illustration of multi-resolution HFM block matching with STF vectors.

rizes the difference and advantages of the proposed FBPME. Section III describes the construction of the binary pyramid. Section IV presents the XOR Boolean block matching and its parallel hardware architecture. Section V describes the N-*scale* tiling multi-path search for the FBPME algorithm. Section VI derives the computation complexity and data bus bandwidth. Section VII shows the testing results of FBPME used in MPEG2 MP@ML and ATSC HDTV encoders for both STDV and HDTV video inputs.

II. Binary ME Algorithms

A binary-based fast hierarchical feature matching ME scheme (HFM-ME) was proposed to reduce the matching computational complexity and bus throughput in [21], [22]. The HFM-ME algorithm performs ME with feature vector matching using integer means and binary phases. Zakhor and Lari [26] used one-bit edge information for global motion parameters to achieve camera stabilization with comparable performance to that of intensity-based integer ME. Natarajan, Bhaskaran, and Konstantinides [23] presented a fast binary ME algorithm based on a simple one-bit transformation and conventional search schemes. This algorithm provides good performance for simple texture and slow motion sequences. For a complex texture scene sequence such as “Flowergarden,” it is reported in [23] that this binary ME is 2-dB below that for the full-search MAD. Gharavi and Mills [20] presented a threshold method to convert the absolute difference of pixel matching into “0” or “1” bit representation. Thus the accumulator of the mean absolute difference (MAD) becomes a simple counter for “1’s.” Feng *et al.* [27] used the mean of 8×8 pixels to threshold all the pixels to form an 8×8 bit-plane. The full search binary match is performed in the bit-planes to find several locations with mismatches less than a threshold. The full search MAD matching is then performed on these selected locations to obtain the best-matched position.

A. HFM-ME

Lee and Zhang explored the basic binary block-matching concepts as illustrated in Fig. 1. The 4×4 integer array of MAD block matching can be replaced by the sign truncated feature (STF) vectors matching with 8 bits of mean, 2×2 and 4×4 bits arrays of phases. As shown in Fig. 2, the HFM-ME algorithm is implemented as a fast motion tracker to provide coarse motion vectors and a refined motion vector with a small (3×3) pixels search window. A half-pixel mode decision module is used to find the final motion vector with half-pixel accuracy. The HFM-ME is executed as follows.

1) Build multi-resolution STF feature vectors, frame-by-frame.
2) Perform 16×16 XOR Boolean matching and 4×4 integer array of MAD matching at each search point
3) Use the relative full search scheme on the binary bit-planes.
4) Use the inner encoding loop MAD ME on reconstructed data within 3×3 pixels of the refined window.

In step 1, a 4×4 sub-block of luminance integers is represented as a feature vector of one integer of the mean, 2×2 bits array, and 4×4 bits array of binary phase pattern. They can be simplified as one mean with 4×4 bits array of phases. Then a 16×16 macro-block is expressed as 4×4 sets of the mixed resolution feature vectors. As the block matching, the 4×4 integer matching of MAD and 16×16 binary matching of XOR Boolean operations are performed. In the XOR Boolean block matching, a “0” stands for a good matched pixel, and a “1” for a mismatched pixel. A look-up table (LUT) of the counted “0’s” is developed for software emulation of the fast numeric mapping of the XOR matching results. The hardware logic implementation should be much simpler and faster.

In step 2, the full-search scheme is applied to the 4×4 array of STF feature vector matching, vector-by-vector. As a straightforward implementation with a ± 1 pixel search step, each mean matching only needs 1/16 of integer computations and data retrieval as in the 16×16 array of integer MAD matching. The 4×4-integer array of matches can be performed with a ± 4 pixel search step. Around each step at location (x, y), 16×16 array of binary phase matching is performed with ± 1 pixel step skew of (dx, dy).

In step 3, the relative full search is performed around each checkpoint without phase skew as in step 2. The 4×4 array of STF feature of the reference frame is matched with the shifted integer means and binary phases of current blocks rather than the skewed reference blocks, in order to eliminate the phase skew errors without retrieving the shifted means of the reference block from the external memories. The STF vectors of (dx, dy) shifted current block can be derived from the input data in the on-chip memories. Therefore, the final motion vector

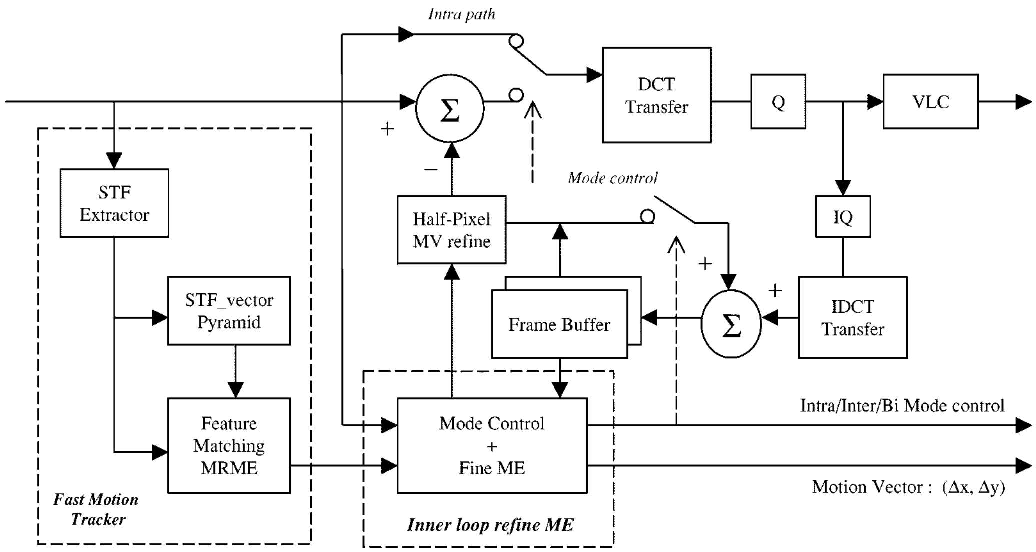

Fig. 2. HFM fast motion tracker and refinement ME of the inner encoder loop.

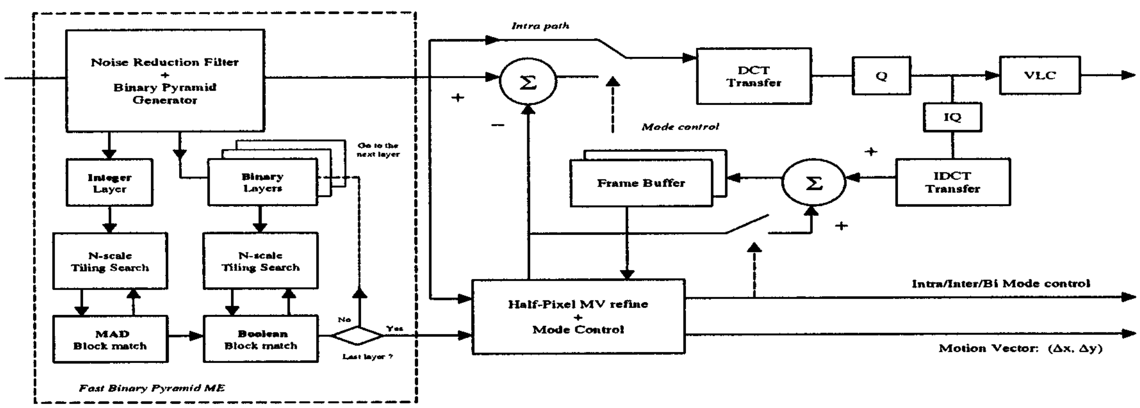

Fig. 3. Illustration of FBPME.

$(x+dx, y+dy)$ is shifted from the input block location (i, j) to $(i-dx, j-dy)$.

In step 4, a refined full-search ME module is done in the inner encoding loop to offset the (dx, dy) shift as shown in Fig. 2. Experiment results show that HFM-ME with relative full-search provides almost the same PSNR performance as the full search, but with much less computation complexity and bus bandwidth.

The disadvantages of the HFM-ME algorithm are: 1) difficulty in combination with sparse fast search; 2) inaccuracy of phase skewing; 3) limited number of layers due to the fixed MB size; 4) the need for the inner encoding-loop ME module; and 5) need for fusion of integer- and binary-matching results.

B. FBPME

In this paper, we propose a new FBPME technique, as shown in Fig. 3, which has low computational complexity, reduces data bandwidth, and requires simple hardware implementation. FBPME consists of the binary pyramid construction, XOR block-matching logic and N-*scale* tiling search scheme. FBPME eliminates the phase skew of the HFM-ME in the binary pyramid construction of the FBPME. The XOR matching is implemented with simple parallel hardware architecture. N-*scale* tiling search is used to refine the motion vector candidates obtained from the lowest resolution layer.

The tiling search scheme combines the advantages of the hierarchical motion projection search and multi-path search similar to the Viterbi algorithm. The multi-path tiling search can greatly reduce the computations with multiple smaller search windows to preserve the robust performance of full search block matching, eliminating the need for large windows using binary full search and the inner-loop refinement in the HFM-ME algorithm.

The FBPME is performed according to the following steps.

1) Build the binary pyramid structure.
2) Perform the N-*scale* tiling search with integer MAD block matching at the top integer layer.
3) perform the n-*scale* tiling search with XOR Boolean block matching at binary layers.
4) Perform full-search XOR Boolean matching with a ± 3 pixel refinement window at original resolution.

In step 1, the Binary Pyramid Generator performs k layer decimation filtering to build a conventional pyramid structure with k integer layers. Each integer layer (except the bottom layer) is up-sampled to approximate its lower layer data. The difference between the integer layer and its approximation is compared

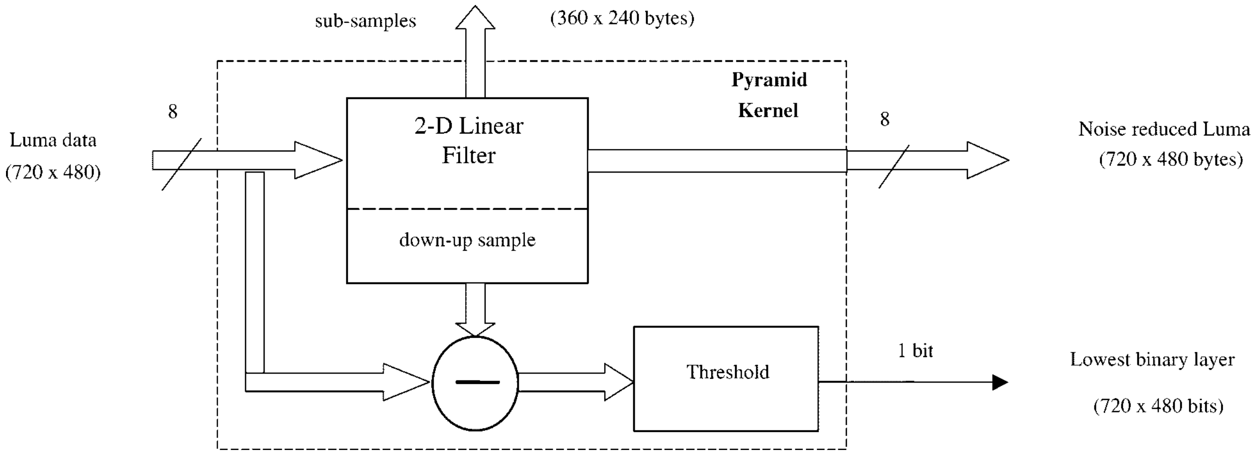

Fig. 4. Basic pyramid filtering and binary bit-plane construction.

to a threshold to derive the binary layer. The resultant binary pyramid structure consists of one top integer layer and $k-1$ binary layers. This new binary pyramid structure offers more precise representation for the residual information of the pyramid layers without the phase skew errors caused by the simple mean threshold methods.

In step 2, N-*scale* ($n = 4$) tiling search is performed for the top integer layer to locate four initial motion vector candidates corresponding to the best MAD block matching with different shapes of tiling super-blocks. these four initial motion vector candidates will be projected into the next binary layer for multi-path motion vector refinements. each candidate is related to one of the four shapes of tiling such as 8×8, 4×8, 8×4, and 4×4 blocks, as shown in Fig. 13. This tiling search with various shaped super-blocks covers greater geometrical region of multiple macroblocks in the original resolution layer to prevent the FBPME from being trapped into local minima. The FBPME scheme, therefore, can achieve more reliable and accurate ME with similar or smaller search windows.

In step 3, the XOR Boolean block match in combined with the N-*scale* tiling search at higher resolution binary layers. centering on each motion vector projection location, the tiling search is performed in a 3×3 pixel window. The four motion vector candidates with the best XOR block match will be projected into the next binary layer. The same process will be repeated, layer-by-layer, until the last binary layer. The XOR Boolean block match can be implemented with LUT methods using software emulation or hardware logic as indicated in Section IV.

In step 4, at the original resolution binary layer, the binary motion vector refinement is performed with the XOR Boolean block match of 16×16 bits and full search around the best block match of the four projected candidates from the previous layer. The search window is ± 3 pixels.

III. Construction of the Binary Pyramid

The binary pyramid is constructed as follows.

1) Initialization: Let k be the number of pyramid level. Let $l = 0$ and level 0 be the original image, i.e., $\mathbf{X_l} = \mathbf{X_0}$.
2) Generate level $l+1$: An image $\mathbf{X_l}$ is filtered by a low-pass filter $\mathbf{H}$, the output of a low-pass filter $\mathbf{H}$ is $\mathbf{H}(\mathbf{X_l})$. $\mathbf{H}(\mathbf{X_l})$ is decimated by a factor of M in each dimension which produces a reduced image $\mathbf{X_{l+1}} = \mathbf{D_M}(\mathbf{H}(\mathbf{X_l}))$. The reduced image $\mathbf{X_{l+1}}$ is interpolated by an interpolating filter $\mathbf{I_M}$ giving the expanded image $\mathbf{I_M}(\mathbf{X_{l+1}})$.
3) Form the Binary Pyramid $\mathbf{B_l}$
$$\mathbf{B_l} = \begin{cases} 1, & \text{if } (\mathbf{X_l} - \mathbf{I_M}(\mathbf{X_{l+1}})) \geq \mathbf{T} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$
where T is a threshold.
4) Termination: Let $l = l + 1$ and if $l \neq k - 1$, go to step 2; otherwise, $\mathbf{B_l} = \mathbf{X}(\mathbf{k-1})$, stop.

Fig. 4 illustrates the basic functions to construct the binary pyramid. The 2-D filter and down-sampling operations produce the conventional pyramid data structures. The difference between the input data and the up-sampling approximations is compared with a threshold to generate the binary plane with the same spatial resolution as the input data. This process is repeated on the down-sampled data, layer-by-layer, until reaching the desired binary level. An example of 4-layer binary pyramid construction is shown in Fig. 5.

An example to illustrate the binary pyramid construction process follows. A 4×4 sub-block of input data is filtered and decimated by a factor of 2, as shown in (2). The 2×2 decimated data is up-sampled as shown in (3). Finally, the difference between (2) and (3) is compared with a threshold of zero to give a binary plane, as shown in (4)

$$\begin{pmatrix} 74 & 59 & 100 & 158 \\ 74 & 69 & 59 & 80 \\ 87 & 86 & 65 & 69 \\ 100 & 118 & 72 & 60 \end{pmatrix} \quad \mathit{sub\text{-}sampling} \quad \begin{pmatrix} 70 & 94 \\ 87 & 72 \end{pmatrix} \tag{2}$$

$$\begin{pmatrix} 70 & 94 \\ 87 & 72 \end{pmatrix} \quad \mathit{up\text{-}sampling} \quad \begin{pmatrix} 70 & 82 & 94 & 94 \\ 78 & 80 & 83 & 83 \\ 87 & 79 & 72 & 72 \\ 87 & 79 & 72 & 72 \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 74 & 59 & 100 & 158 \\ 74 & 69 & 59 & 80 \\ 87 & 86 & 65 & 69 \\ 100 & 118 & 72 & 60 \end{pmatrix} - \begin{pmatrix} 70 & 82 & 94 & 94 \\ 78 & 80 & 83 & 83 \\ 87 & 79 & 72 & 72 \\ 87 & 79 & 72 & 72 \end{pmatrix}. \tag{4}$$

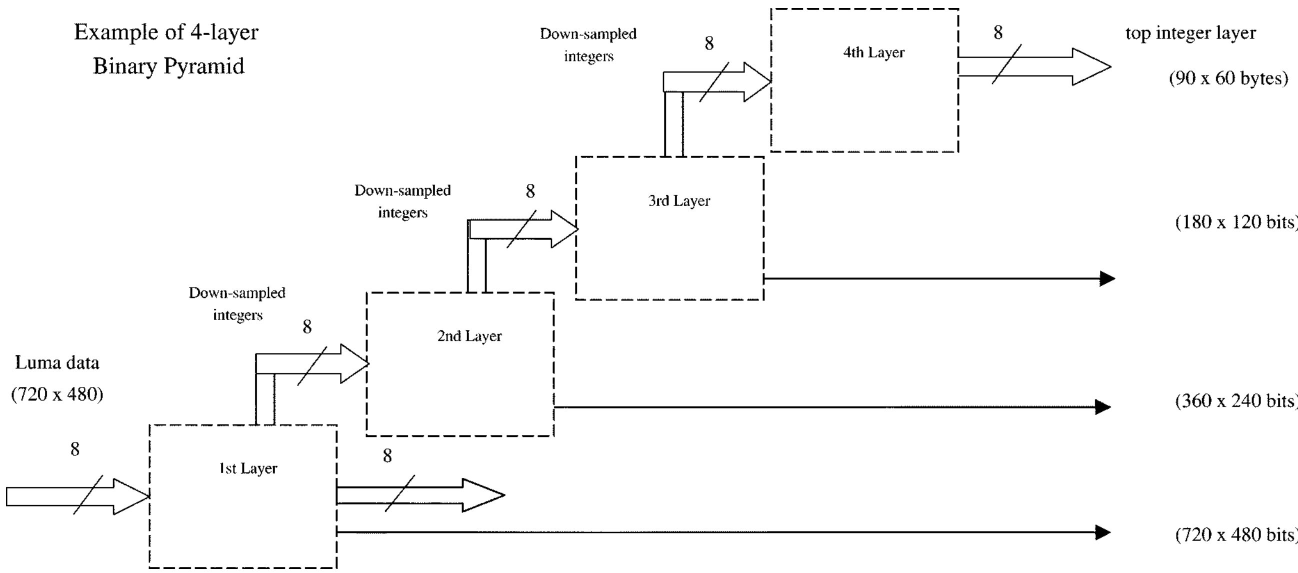

Fig. 5. Illustration of 4-layer binary pyramid construction.

Fig. 6. Image of level 1 of the binary pyramid generated from the second frame in the "Flowergarden" sequence.

Fig. 6 shows the first binary layer image of the second frame in the "Flowergarden" sequence. Figs. 7 and 8 show the second and third binary layer images, respectively. Fig. 9 shows the top layer integer image, in 8 bits per pixel, as the fourth layer of conventional sub-samples. The binary layer images have not only preserved the edge information but also emphasized the complex details from relatively flat areas, such as the cloud area in the "Flowergarden" sequence.

IV. XOR Boolean Block-Matching Criterion and Parallel Architecture

Block-matching algorithms are widely used in motion-compensated video-coding applications. In block-matching ME, a frame F_c is divided into blocks with size of $G \times G$. For each block in the current frame F_c, a block from the reference frame F_r (a previous or future frame) with corresponding displacement (x, y) is selected by satisfying the block-matching criteria. This displacement (x, y) in the reference block shifted from the current block location is defined as the motion vector.

A. Block-Matching Criteria

MAD is often used as the block-matching criterion. MAD calculation involves one subtraction, one absolute value operation, and one accumulation per pixe .

$$\mathrm{MAD}(x, y) = \frac{1}{G^2} \sum_{i=0}^{G-1} \sum_{j=0}^{G-1} |I_c(i, j) - I_r(i+x, j+y)| \quad (5)$$

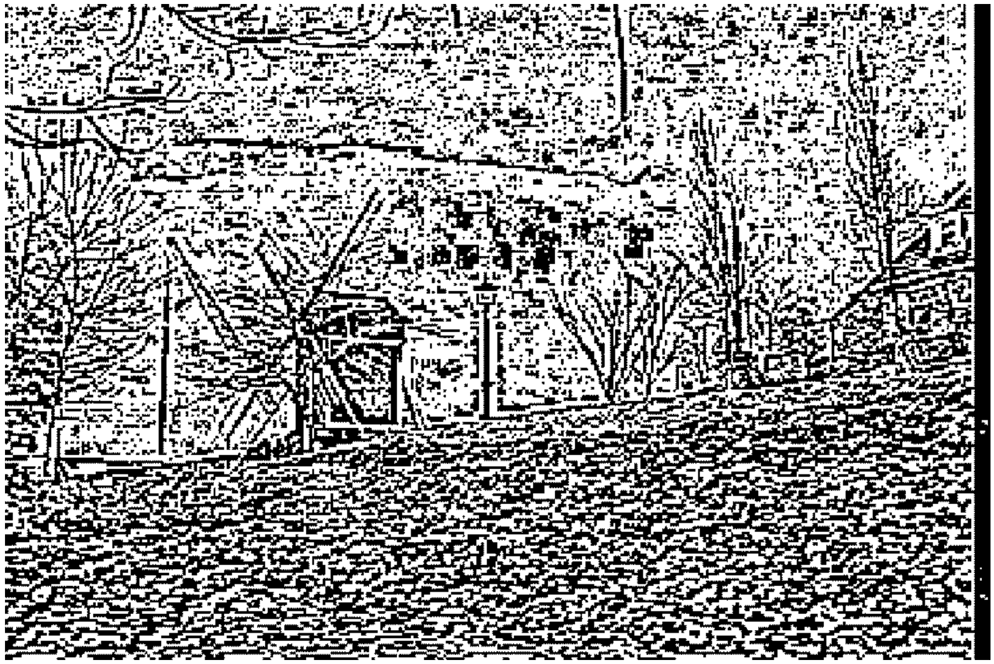

Fig. 7. The image of level 2 of the binary pyramid generated from the second frame in the "Flowergarden" sequence.

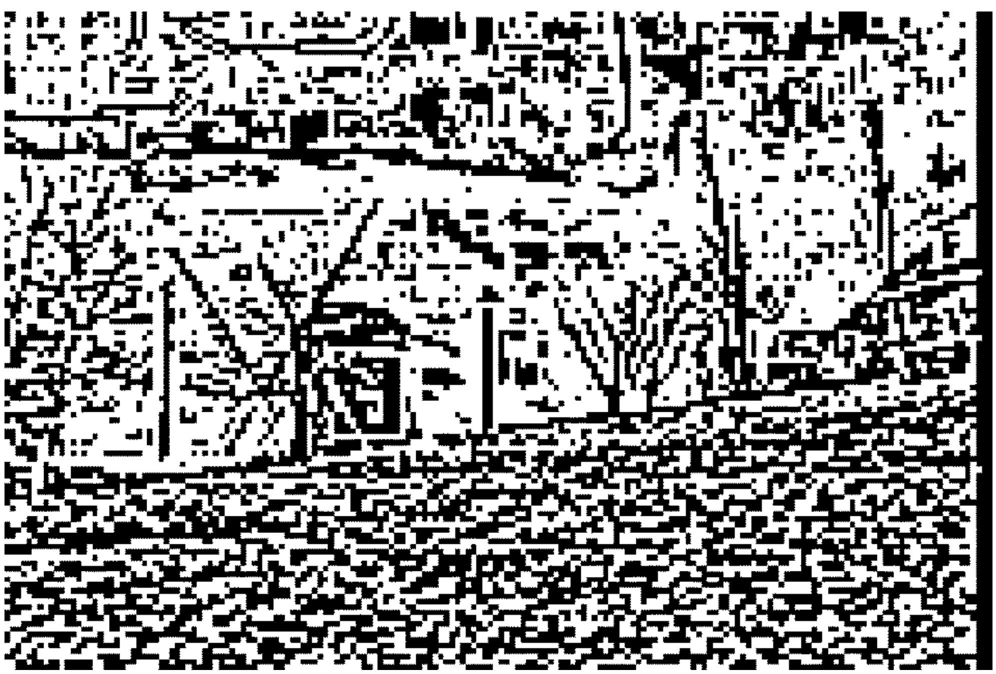

Fig. 8. The image of level 3 of the binary pyramid generated from the second frame in the "Flowergarden" sequence.

where $I_c(i, j)$ is the value of pixel intensity at location (i, j) within the current frame. $I_r(i+x, j+y)$ is the value of pixel intensity at location $(i+x, j+y)$ within the reference frame.

In MAD block matching, each pixel is represented as an 8-bit integer. In binary pyramid, a byte is reduced to a bit so only bit-wise operations are needed. The absolute difference computation then becomes a simple logic XOR operation, such as

$$\begin{aligned} A-B &= A \oplus B, \quad \text{with a carry bit} \quad Cy = \overline{A}B \\ |A-B| &= A \oplus B, \quad \text{without any carry bit} \end{aligned}$$

where $\oplus$ denotes the XOR operation.

The XOR binary block match is the simplest case of the MAD matching criterion with the minimum hardware complexity. The XOR match criterion is expressed as the following:

$$\mathrm{XOR}(x, y) = \sum_{i=0}^{G-1} \sum_{j=0}^{G-1} \phi(I_c(i, j) \oplus I_r(i+x, j+y)) \quad (6)$$

$$\phi(I_c(i, j) \oplus I_r(i+x, j+y)) = \begin{cases} 0, & \text{if } I_c(i, j) = I_r(i+x, j+y) \\ 1, & \text{otherwise.} \end{cases} \quad (7)$$

Fig. 9. Image of level 4 of the binary pyramid generated from the second frame in the “Flowergarden” sequence.

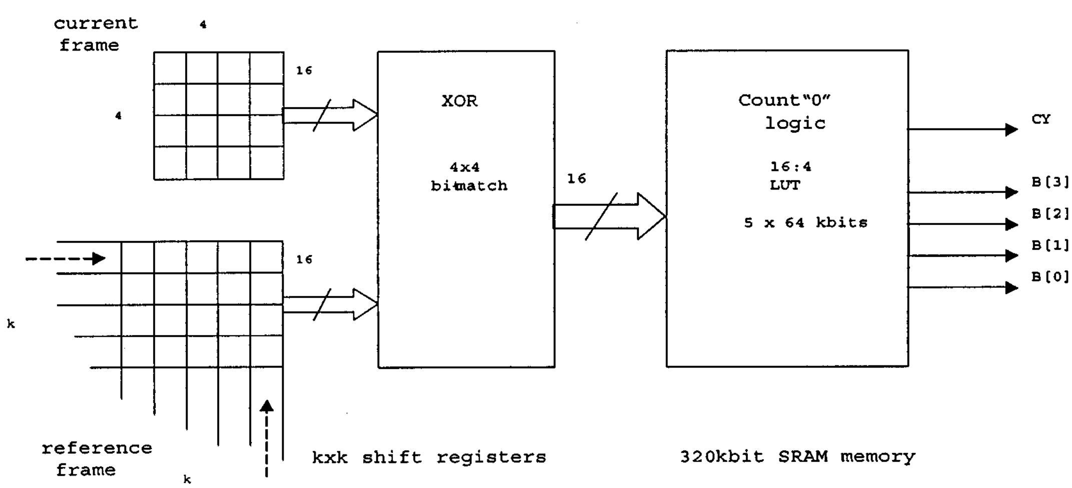

Fig. 10. Block diagram of the 4×4 kernel operator of 4×4 arrays of XOR Boolean block matching.

It shows that the binary block matching becomes simply counting the number of “0”s in the $G \times G$ match pattern between the current and the reference binary blocks. The more the “0”s, the better the two binary blocks are matched.

B. The Parallel XOR Matching Architecture

The binary XOR matching operations can be easily built into $N \times N$ arrays of parallel logic architecture. In a simple field programmable gate array (FPGA) design, the kernel operator can be 4×4 arrays and 4×4 such kernel operators can perform the 16×16 of Boolean XOR match of a macro-block in a single clock cycle.

Fig. 10 shows a block diagram of the 4×4 kernel operator. It consists of a 4×4 register file, a $k \times k$ shift register arrays, a 4×4 XOR match operator, and a $16:4$ LUT memory. The 4×4 register file holds a 4×4 binary sub-block from the current frame F_c and the $k \times k$ shift register arrays hold the bits of the search window from the reference frame F_r. The 4×4 XOR operator performs the 4×4 Boolean match, and the $16:4$ LUT converts the number of “0” in the 16 bit match result into the 4-bit numeric data. All the registers, shift registers, and LUT memory cells can be implemented by SRAM (Static Random Access Memory) technologies. The $16:4$ LUT memory utilizes 320 kbits SRAM cells, as $(4 + 1) \times 2^1 6$ bits and most of them saved the same value. Using two of $8:3$ LUT and a 4-bit adder can greatly reduce the SRAM size with com-

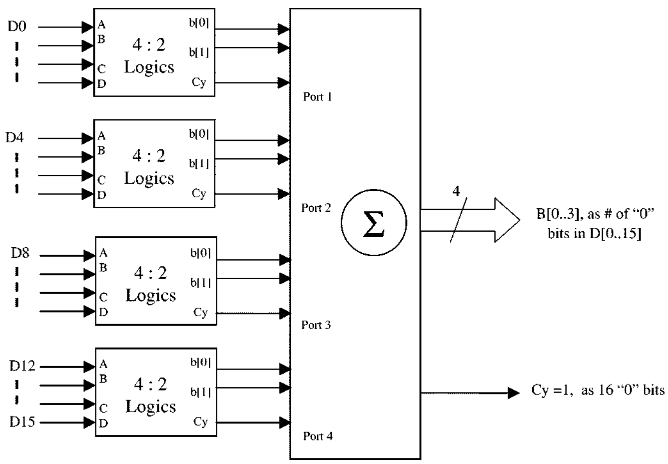

Fig. 11. Partitions of combinatorial logic for the 16 : 4 count "0" logic.

promising the speed. A 4-bit counter and 16 bit shift register can easily count the number "0" of the matched bits in 16 clock cycles with much less logic cells.

The "Hard-Wire" application-specific integrated circuits (ASICs) solutions can implement the same 16 : 4 logic of counting "0" with much less logic gates and the fast performance. Fig. 11 illustrates the basic functions of the 16 : 4 mapping logic with the "Hard-Wire" combinatorial logic implementations. The 16 : 4 logic can be partitioned into 4 of 4 : 2 count "0" logic circuits and a summation logic to add up the 4 of 2 bit numerical numbers.

The 4 : 2 count "0" logic converts the 4 input bits $[A, B, C, D]$ to 2 output bits $b[1]$, $b[0]$, and Cy (carry bit). When the $Cy = 1$, the $b[1]$ and $b[2]$ will be equal to 0, because there are only 4 input bits. The 4 : 2 count "0" logic is expressed as

$$\begin{aligned} Cy &= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \\ b[0] &= A \oplus B \oplus C \oplus D \\ b[1] &= \overline{Cy}\,\overline{ABC}\,\overline{ABD}\,\overline{ACD}\,\overline{BCD} \end{aligned}$$

where the $b[0]$ shows there are 1 or 3 "0"s and the $b[1]$ with 2 or 3 "0" bits in the 4 input bits, respectively. The $b[1]$ can be illustrated as there are no 1 "0," nor 4 "0"s, nor 4 "1"s.

The summation logic can be further partitioned into three count "1" logic circuits, as illustrated in Fig. 12. At first, the $B[0]$ count "1" logic adds all the least significant bits $\{b1[0], b2[0], b3[0], b4[0]\}$ from the outputs of four count "0" circuits. Then, the $B[1]$ logic adds the second bits $\{b1[1], b2[1], b3[1], b4[1]\}$ and the carry bit from $B[0]$. Finally, the $B[2]$ logic adds all the carry bits $\{Cy1, Cy2, Cy3, Cy4\}$ plus carry bits from $B[0]$ and $B[1]$, respectively. They are expressed as follows:

$$\begin{aligned} B[0] &= b_1[0] \oplus b_2[0] \oplus b_3[0] \oplus b_4[0] \\ B[1] &= b_1[1] \oplus b_2[1] \oplus b_3[1] \oplus b_4[1] \oplus C_{10} \\ B[2] &= Cy1 \oplus Cy2 \oplus Cy3 \oplus Cy4 \oplus C_{20} \oplus C_{21} \\ B[3] &= C_{31} \oplus C_{32} \\ CY &= Cy1Cy2Cy3Cy4 \end{aligned}$$

where the C_{10} and C_{20} are the carry bits from the $B[0]$ logic. The bits C_{21}, C_{31} and C_{32} are carry bits from the $B[1]$ logic and $B[2]$ logic circuits, respectively. When $CY = 1$, the set $\{B[3], B[2].B[1], B[0]\}$ equals $\{0, 0, 0, 0\}$ because this 16 : 4 count "0" logic has only 16 input bits.

The carry bits from the $B[0]$ logic are expressed as

$$\begin{aligned} C_{20} &= b_1[0]b_2[0]b_3[0]b_4[0] \\ C_{10} &= \overline{C_{20}}\,\overline{\overline{b_1[0]}\,\overline{b_2[0]}\,\overline{b_3[0]}}\,\overline{\overline{b_1[0]}\,\overline{b_2[0]}\,\overline{b_4[0]}} \\ &\quad \cdot \overline{\overline{b_1[0]}\,\overline{b_3[0]}\,\overline{b_4[0]}}\,\overline{\overline{b_2[0]}\,\overline{b_3[0]}\,\overline{b_4[0]}} \end{aligned}$$

where the C_{10} can be interpreted as that there exist no four "0"s nor one "1" nor four "1"s in the least significant bits $\{b1[0], b2[0], b3[0], b4[0]\}$.

To simplify the logic, three of 4 : 2 count "1" circuits are used. The $B'[1]$ and $B'[2]$ logic are expressed as

$$\begin{aligned} B'[1] &= b_1[1] \oplus b_2[1] \oplus b_3[1] \oplus b_4[1] \\ C'_{31} &= b_1[1]b_2[1]b_3[1]b_4[1] \\ C'_{21} &= \overline{\overline{b_1[1]}\,\overline{b_2[1]}\,\overline{b_3[1]}}\,\overline{\overline{b_1[1]}\,\overline{b_2[1]}\,\overline{b_4[1]}}\,\overline{\overline{b_1[1]}\,\overline{b_3[1]}\,\overline{b_4[1]}}\,\overline{C'_{31}} \\ B'[2] &= Cy1 \oplus Cy2 \oplus Cy3 \oplus Cy4 \\ C'_{32} &= \overline{\overline{Cy1}\,\overline{Cy2}\,\overline{Cy3}}\,\overline{\overline{Cy1}\,\overline{Cy2}\,\overline{Cy4}} \\ &\quad \cdot \overline{\overline{Cy1}\,\overline{Cy3}\,\overline{Cy4}}\,\overline{\overline{Cy2}\,\overline{Cy3}\,\overline{Cy4}}\,\overline{CY}. \end{aligned}$$

Consequently, the carry bits from $B[1]$ and $B[2]$ circuits become the follows:

$$\begin{aligned} C_{21} &= C'_{21} \oplus (C_{10}B'[1]) \\ C_{31} &= C'_{31} \oplus (C_{10}B'[1]C'_{21}) \\ C_{32} &= C'_{32} \oplus (C_{20}C_{21} + C_{20}B'[2] + C_{21}B'[2]) \end{aligned}$$

The special "Hard-Wire" ASIC implementation of this parallel logic architecture for the Boolean XOR block matching can be much faster than the FPGA LUT designs and the software emulations of the LUT algorithms with the general purpose DSP and CPU processors.

V. Bimary Pyramid ME with N-Scale Tilings

Let F_m denote the mth image frame in a video sequence. The binary pyramid F_m is generated using the algorithm described in Section III. Each pyramid layer is partitioned into nonoverlapping blocks of size $\mathbf{s}$. The resulting partition or *tiling* at layer l is represented by $F_m^{l,\mathbf{s}}$. The intensity value of the pixel with coordinates $\mathbf{x} = [x_1, x_2]^T$ in an image frame m at layer l is denoted by $F_m^l(\mathbf{x})$ where x_1, x_2, and T denote the row index, the column index, and "transpose," respectively. For a given $\mathbf{s} = [s_1, s_2]^T$, the block of pixels with upper left corner at image position $\mathbf{x}$ at layer l is referred to as $B_m^l(\mathbf{x}, \mathbf{s}) = \{F_m^l(\mathbf{q}) \in F_m^l | \mathbf{x} \le q < \mathbf{x} + \mathbf{s}\}$. The sum of the absolute differences between pixels from a block $B_m^l(\mathbf{x}, \mathbf{s})$ from frame m and corresponding pixels in block $B_{m-1}^l(\mathbf{x}, \mathbf{s})$ from frame $m-1$ can be represented as

$$\begin{aligned} B_m^l(\mathbf{x}, \mathbf{s}) &\ominus B_{m-1}^l(\mathbf{x}, \mathbf{s}) \\ &= \sum_{\mathbf{0} \le \mathbf{d} < \mathbf{s}} |F_m^l(\mathbf{x}+\mathbf{d}) - F_{m-1}^l(\mathbf{x}+\mathbf{d})|. \end{aligned} \quad (8)$$

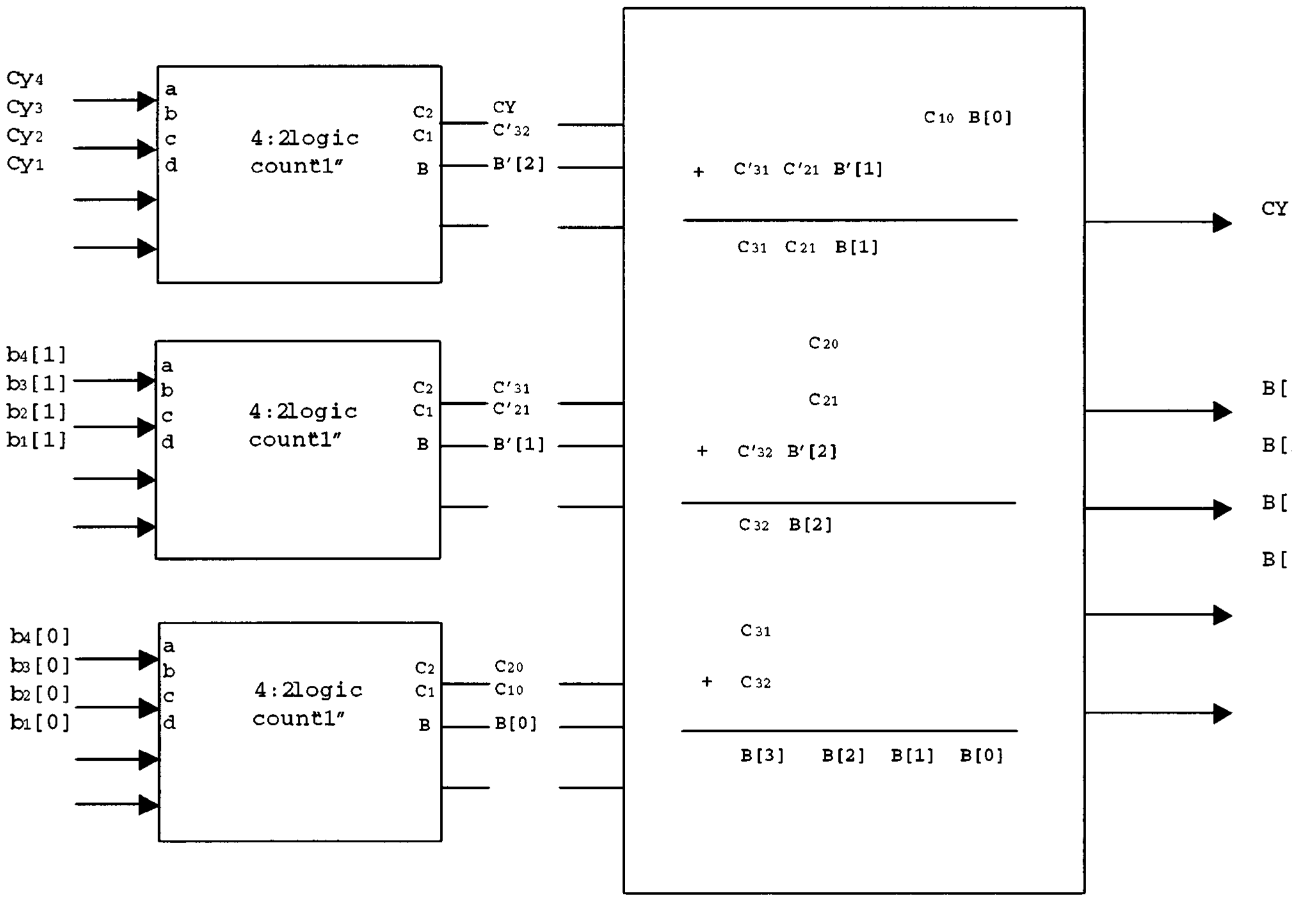

Fig. 12. Partition of the summation logic into three count 1 logic and a complex carry logic.

The sum of *XOR Boolean* operations between pixels from a block $B_m^l(\mathbf{x}, \mathbf{s})$ from frame m and corresponding pixels in block $B_{m-1}^l(\mathbf{x}, \mathbf{s})$ from frame $m-1$ can be denoted as

$$\begin{aligned} B_m^l(\mathbf{x}, \mathbf{s}) &\oplus B_{m-1}^l(\mathbf{x}, \mathbf{s}) \\ &= \sum_{\mathbf{0} \le \mathbf{d} < \mathbf{s}} \phi(F_m^l(\mathbf{x}+\mathbf{d}) \oplus F_{m-1}^l(\mathbf{x}+\mathbf{d})) \end{aligned} \tag{9}$$

$$\begin{aligned} \phi(F_m^l(\mathbf{x}+\mathbf{d}) &\oplus F_{m-1}^l(\mathbf{x}+\mathbf{d})) \\ &= \begin{cases} 0, & \text{if } F_m^l(\mathbf{x}+\mathbf{d}) = F_{m-1}^l(\mathbf{x}+\mathbf{d}) \\ 1, & \text{otherwise} \end{cases} \end{aligned} \tag{10}$$

Let $F^{0,\mathbf{s}}$ be a tiling defined on the full resolution image with cardinality κ. Let $F^{0,\mathbf{s}/2}$ be a tiling defined on level 1. For the lower resolution image, N-*scale* tilings are considered. They are $\{F^{l,\mathbf{s1}}\}_{l=2}^{k-1}$, $\{F^{l,\mathbf{s2}}\}_{l=2}^{k-1}$, …, and $\{F^{l,\mathbf{sN}}\}_{l=2}^{k-1}$.

1) Initialization: Let k be the number of binary pyramid level and $l = k-1$. At the Level $k-1$, the motion vector fields $\mathbf{v}_{s1}^{k-1}$, $\mathbf{v}_{s2}^{k-1}$, $\cdots$, and $\mathbf{v}_{sN}^{k-1}$ are defined as follows:

$$\mathbf{v}_{sj}^{k-1} = \arg \min_{v \in \Omega^{k-1}} B_m^{k-1}(\mathbf{x}, \mathbf{s}j) \ominus B_{m-1}^{k-1}(\mathbf{x}+\mathbf{v}, \mathbf{s}j) \tag{11}$$

where

$$1 \le j \le N \quad \Omega^{k-1} = \{\mathbf{v}\colon -\mathbf{d}^{k-1} \le \mathbf{v} \le \mathbf{d}^{k-1}\}.$$

2) Projection and Refinement: From the lower resolution $l+1$ to the higher resolution l, the motion vector fields are projected according to the following:

$$\mathbf{u}_{sj}^l = 2\mathbf{v}_{sj}^{l+1} \tag{12}$$

where

$$1 \le j \le N.$$

The motion vector fields at the resolution l are refined as follows:

$$\begin{aligned} \mathbf{v}_{sj}^l = \arg &\min_{\mathbf{v} \in \{\mathbf{w}_{j1}^l, \mathbf{w}_{j2}^l, \cdots, \mathbf{w}_{jN}^l\}} B_m^l(\mathbf{x}, \mathbf{s}j) \\ &\oplus B_{m-1}^l(\mathbf{x}+\mathbf{v}, \mathbf{s}j) \end{aligned} \tag{13}$$

where

$$\begin{aligned} \mathbf{w}_{ji}^l = \arg &\min_{v \in \Omega^l} B_m^l(\mathbf{x}+\mathbf{u}_{si}^l, \mathbf{s}j) \\ &\oplus B_{m-1}^l(\mathbf{x}+\mathbf{u}_{si}^l+\mathbf{v}, \mathbf{s}j) \\ 1 \le j \le N \quad 1 \le i \le N \quad &\Omega^l = \{\mathbf{v}\colon -\mathbf{d}^l \le \mathbf{v} \le \mathbf{d}^l\}. \end{aligned} \tag{14}$$

3) Let $l = l-1$ and if $l \ne 1$, go to step 2; otherwise, go step 4, continue.
4) Level 1:

$$\mathbf{u}_{sj}^1 = 2\mathbf{v}_{sj}^2 \tag{15}$$

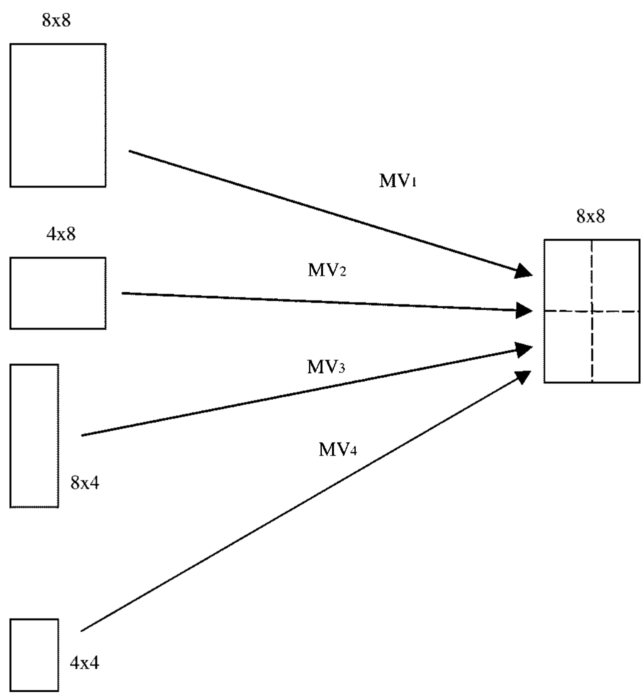

Fig. 13. Illustration of 4–scale tilings search scheme.

where

$$1 \leq j \leq N$$

$$\mathbf{v}_{s/2}^{1} = \arg \min_{\mathbf{v} \in \{\mathbf{w}_1^{\ell}, \mathbf{w}_2^{\ell}, \cdots, \mathbf{w}_N^{\ell}\}} B_m^1(\mathbf{x}, \mathbf{s/2}) \oplus B_{m-1}^1(\mathbf{x}+\mathbf{v}, \mathbf{s/2}) \tag{16}$$

where

$$\mathbf{w}_i^1 = \arg \min_{v \in \Omega^1} B_m^1(\mathbf{x}+\mathbf{u}_{si}^1, \mathbf{s/2}) \oplus B_{m-1}^1(\mathbf{x}+\mathbf{u}_{si}^1+\mathbf{v}, \mathbf{s/2}) \quad 1 \leq i \leq N \quad \Omega^1 = \{\mathbf{v}\colon -\mathbf{d}^1 \leq \mathbf{v} \leq \mathbf{d}^1\}. \tag{17}$$

5) Termination: Level 0

$$\mathbf{u}^0 = 2\mathbf{v}_{s/2}^1 \tag{18}$$

$$\mathbf{v}^0 = \arg \min_{v \in \Omega^0} B_m^0(\mathbf{x}+\mathbf{u}^0, \mathbf{s}) \oplus B_{m-1}^0(\mathbf{x}+\mathbf{u}^0+\mathbf{v}, \mathbf{s}) \tag{19}$$

where

$$\Omega^0 = \{\mathbf{v}\colon -\mathbf{d}^0 \leq \mathbf{v} \leq \mathbf{d}^0\}.$$

Fig. 13 shows a *4-scale* tiling example. A binary pyramid layer can be tiled into 8×8, 8×4, 4×8, 4×4 super-blocks with various shapes, with respect to the scale $N = 4$. There are three advantages in the N*-scale* tiling search in the binary pyramid structure: 1) large super-blocks can be used for a more hierarchical layer ME; 2) natural scenes frequently contain motion at different scales—various shapes can be better fitted with motion objects; 3) viterbi-like multi-path hierarchical projection search can prevent the ME from being trapped in local minima. The tiling multi-path search is much more important for binary based ME, as illustrated in Fig. 14.

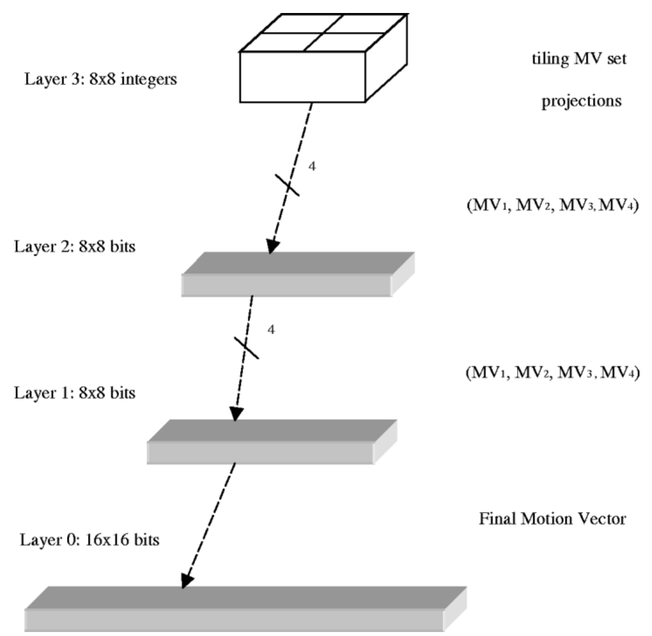

Fig. 14. Tiling set of motion vector projection into multi-layer pyramid.

Fig. 14 shows the FBPME using *4-scale* tiling. ME is first performed at level 3 using four different tiling block sizes as shown in Fig. 13 in the conventional block matching. Each detected motion vector from each scale is propagated to the next lower level and is refined using XOR matching criterion at four different scales. This process repeats once until level 1 is reached. At level 1, the four motion vectors projected from level 2 are refined using XOR matching criterion at the 8×8 tiling block. The best motion vector from these four motion vectors, based on the most “0’s” from XOR matching, is projected to level 0. At level 0, the final motion vector is obtained refining the motion vector projected from level 1 using XOR matching criterion for the 16×16 tiling block.

VI. Complexity and Bus Bandwidth

A. Complexity of the FBPME

The width and the height of the image sequence is W and H, respectively. The search window has $\pm M$ pixels. In the conventional full search block-matching algorithm, it requires one subtraction, one addition, and one absolute value operation for a single pixel matching. The computational complexity (operations per frame) is approximated as

$$C_{FULL} \simeq W \times H \times 4 \times M^2 \times 3 = 12WHM^2. \tag{20}$$

In FBPME, we use $k = 4$ and $N = 4$. For the FBPME the same full-search scheme is used. The tiling block size at level 0 is 16×16. The tiling block size at level 1 is 8×8. The same tiling block sizes of 8×8, 8×4, and 4×8, 4×4 are used at level 2, and level 3. The effective search range at level 0, level 1, and level 2 is set to ± 3 pixels. The effective search range at level 3 is set to $\pm M/8$ pixels.

At level 3, the computational complexity is approximated as

$$C_3 \simeq \frac{WH}{64} \times \frac{M^2}{64} \times 4 \times 3 = \frac{3WHM^2}{1024}. \quad (21)$$

At level 2 and level 1, each 8×8 block can be represented by four 16-bit vectors. A 16-bit XOR Boolean operator can be implemented using one 16-bit exclusive-or arrays, a dual-port LUT with 65 536 entries. The 8×8 block needs four additions for the 64 pixels for the basic matching. Likewise, the 8×4 block needs two additions for the 32 pixels basic matching. Then, the computational complexity at level 2 and level 1 is approximated as

$$C_2 \simeq \frac{WH}{16} \times \left(\frac{4}{64} + \frac{2}{32} \times 2\right) \times 49 = \frac{147WH}{256}. \quad (22)$$

At level 1, The 8×8 block then needs four additions for the 64 pixels for the basic matching. The computational complexity at level 1 is approximated

$$C_1 \simeq \frac{WH}{4} \times \frac{4}{64} \times 49 = \frac{49WH}{64}. \quad (23)$$

At level 0, the 16×16 block needs 16 additions for the 256 pixels for the basic matching. The computational complexity is approximated as

$$C_0 \simeq WH \times \frac{16}{256} \times 49 = \frac{49WH}{16}. \quad (24)$$

Therefore, the computational complexity of the FBPME can be approximated as

$$C_{\mathrm{FBPME}} = C_3 + C_2 + C_1 + C_0 = \frac{3WHM^2}{1024} + \frac{1127WH}{256}. \quad (25)$$

The ratio between C_{FBPME} and C_{FULL} is as follows:

$$\frac{C_{\mathrm{FBPME}}}{C_{FULL}} = \frac{\frac{3WHM^2}{1024} + \frac{1127WH}{256}}{12WHM^2} = \frac{1}{4096} + \frac{1127}{3072M^2}. \quad (26)$$

It is seen from (26) that the computational complexity of FBPME is very low compared with the full search.

B. Bus Bandwidth

Bus throughput and on-chip memory requirement are often bottlenecks for cost-effective real-time MPEG encoder implementation. This subsection estimates the required data throughput for the proposed FBPME scheme.

The frame rate, width. and height of the image sequence is F_r, W, and H, respectively. The size of the image block is $G \times G$. A picture frame contains (H/G) pictures slices, and there are (W/G) blocks in each slice. The search window has $\pm M$ pixels. In a block-matching ME, search areas of adjacent blocks overlap quite a bit. This overlapped area data can be stored inside the on-chip memory buffer to reduce external memory bandwidth. We assume an on-chip memory buffer "$\mathbf{D}$" whose size equals to the search area, $(G+2M) \times (G+2M)$ bytes. The new loading data size for buffer $\mathbf{D}$ is $G \times (G + 2M)$ bytes when the next block is on the same picture slice. We need to load the complete buffer at the beginning of a slice while processing one picture slice. Thus, the total external memory bandwidth per slice is approximately $((G + 2M)^2 + ((W/G) - 1) \times G \times (2M + G))$ bytes if boundary block cases are neglected. So, the bus bandwidth (bytes/s) of the Full search is approximated as

$$MB_{full} \simeq \frac{H}{G}\left((G+2M)^2 + \left(\frac{W}{G} - 1\right) \times G \times (2M+G)\right) \times F_r. \quad (27)$$

A derivation of the memory bandwidth requirement for FBPME is given in the following.

At layer 3, the bus bandwidth (bytes) is approximated as

$$MB_3 \simeq \frac{H}{64} \times \left((8+M/4)^2 + \left(\frac{W}{64} - 1\right) \times 8 \times (M/4+8)\right). \quad (28)$$

At layer 2, the bus bandwidth (bytes) is approximated as

$$MB_2 \simeq \left(\frac{HW}{16 \times 32} \times (8+6) \times (4+6) \times 2 + \frac{HW}{32 \times 32} \times (8+6)^2 + \frac{HW}{16 \times 16} \times (4+6)^2\right) \times \frac{1}{8}. \quad (29)$$

At layer 1, the bus bandwidth (bytes) is approximated as

$$MB_1 \simeq \frac{H}{16} \times \frac{W}{16} \times (8+6)^2 \times 4 \times \frac{1}{8}. \quad (30)$$

At layer 0, the bus bandwidth (bytes) is approximated as

$$MB_0 \simeq \frac{H}{16} \times \frac{W}{16} \times (16+6)^2 \times \frac{1}{8}. \quad (31)$$

Therefore, the bus bandwidth (bytes/s) of the FBPME is approximated as

$$MB_{\mathrm{FBPME}} \simeq F_r \times (MB_0 + MB_1 + MB_2 + MB_3). \quad (32)$$

TABLE I
BUS BANDWIDTH IN (MBYTES/S) FOR THE TWO ALGORITHMS
(F_r = 30 Hz, W = 720, H = 480)

Algorithms	BS(M = 128)	BS(M = 64)
FBPME	8.73	8.21
FullSearch	238.93	109.90

Table I lists bus bandwidth requirements in Mbytes/s for the FBPME and full search, where "BS" represents Bus Bandwidth. It is showing that the bus bandwidth requirement of the proposed algorithms is much smaller than that of full search.

VII. EXPERIMENTAL RESULTS

The proposed FBPME algorithm was implemented in the MPEG-2 framework. The SDTV MPEG sequences "Flowergarden," "Mobi" of size 720×480 consisting of 150 frames each and the HDTV sequences "Jeep," "Mask" of size

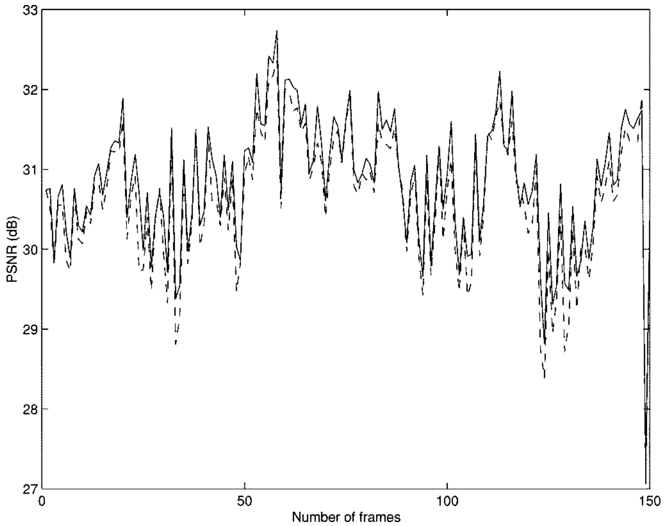

Fig. 15. PSNR versus frame number for an MPEG-2 encoder, for test sequence Flowergarden. Solid line: full Search. Dashed line: FBPME.

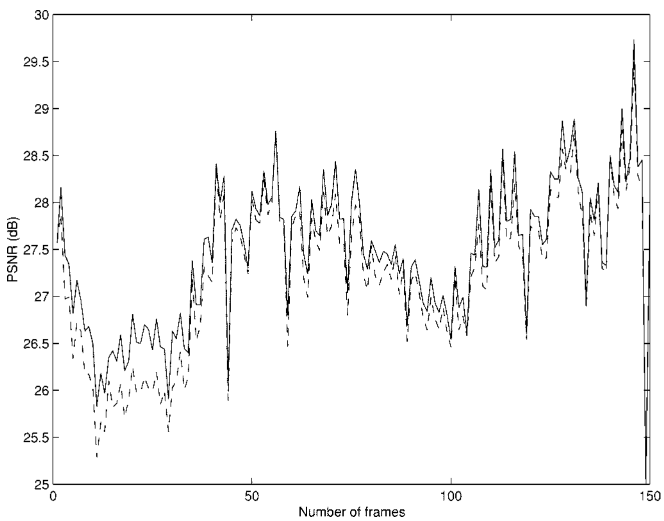

Fig. 16. PSNR versus frame number for an MPEG-2 encoder, for test sequence Mobi. Solid line: full search. Dashed line: FBPME.

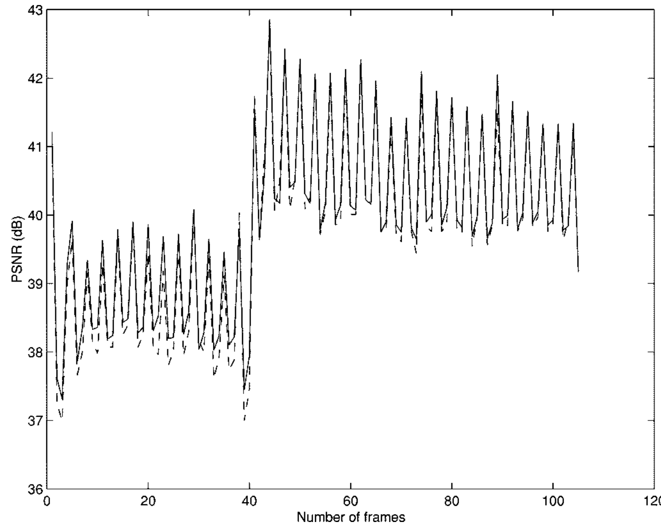

Fig. 17. PSNR versus frame number for an MPEG-2 encoder, for test sequence Jeep. Solid line: full search. Dashed line: FBPME.

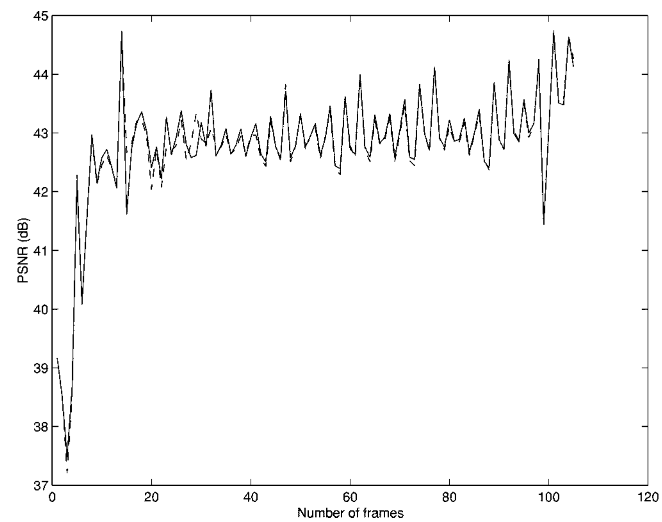

Fig. 18. PSNR versus frame number for an MPEG-2 encoder, for test sequence Mask. Solid line: full search. Dashed line: FBPME.

1920×1080 consisting of 105 frames each were used in this simulation. In our experiment, the size of group of the pictures (GOP) was set to 15. The prediction distance between I frame and P frame was 3. The effective search range was set to ± 128. The coding rate was 4 Mbytes/s for SDTV and 19 Mbytes/s for HDTV. The buffer size for SDTV and HDTV was set to 1.79 Mbits and 7.81 Mbits, respectively.

Our motion estimator computes all encoding modes including forward, backward, bi-directional, and dual prime but only one mode will be selected. The mode decision for each macro-block is critical for the encoder performance. In our experiment, we use an optimized mode selection that considers the rate-distortion behavior of each coding mode. For example, the bi-directional interpolative mode uses more motion vectors, although it typically yields less residual. Thus, our experimental results will give a better representation of the encoding results of an optimized encoder. The comparison between full search and FBPME uses the same optimized mode selection.

TABLE II
PSNR (DECIBELS) OF MPEG-2 COMBINED WITH FBPME METHOD AND FULL SEARCH

	PSNR (dB)			
Methods	Flowergarden	Mobi	Jeep	Mask
FBPME	30.63	27.22	39.74	42.75
Full Search	30.84	27.41	39.83	42.78

The PSNR results using the proposed FBPME and full search for the "Flowergarden," "Mobi," "Jeep," and "Mask" are shown in Figs. 15–18, respectively. Table II shows the average PSNR of the MPEG-2 with FBPME and full search.

One can see from Table II and Figs. 15–18 that the proposed FBPME achieves comparable performance with full search but with much less complexity.

VIII. CONCLUSION

In this paper, we presented a FBPME algorithm that not only significantly reduces the computational complexity, but also greatly reduces the bus bandwidth requirement. The FBPME takes advantages of XOR Boolean matching and N-*scale* tiling multi-path search scheme. it achieves the same level of PSNR and visual quality at less than 1/4000 of the computational load of the conventional full-search algorithm. Its binary nature allows it to be implemented very easily in hardware. the proposed scheme was implemented in an MPEG-2 MP@ML and ATSC HDTV encoder framework. extensive test results have indicated the superior performance of the proposed fbme in speed, memory requirement, as well as data throughput.

REFERENCES

[1] *Recommendation H.263 Video Coding for Low Bit Rate Communication*, ITU-T, H263, Jan. 1998.

[2] *Information Technology—Coding of MovingPctures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s—Part 2: Video*, ISO/IEC, 11172-2, 1990.

[3] *Information Technology—Generic Coding of Moving Pictures and Associated Audio Information—Part 2: Video*, ISO/IEC, International Standard 13 818-2, 1994.

[4] *Coding of Moving Pictures and Associated Audio*, MPEG-4 FDIS ISO/IEC JTC1/SC29/WG11, ISO/IEC 14 496-2, Mar. 1999.

[5] *ATSC Digital Television Standard, ATSC specifications*, Advanced Television Standard Committee, Sept. 1995.

[6] T. Koga *et al.*, "Motion-compensated interframe coding for video conferencing," in *Proc. Nat. Telecommunications Conf.*, Nov./Dec. 1981, pp. G 5.3.1–G 5.3.5.

[7] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. Commun.*, vol. COM-29, pp. 1799–1808, Dec. 1981.

[8] R. Srinivasan and K. R. Rao, "Predictive coding based on efficient motion estimation," *IEEE Trans. Commun.*, vol. COM-33, pp. 1011–1014, Sept. 1985.

[9] K. Chow and M. L. Liou, "Genetic motion search algorithm for video compression," *IEEE Trans. Circuits Syst. Video Technol.*, no. 6, pp. 440–445, Dec. 1993.

[10] C. H. Lin and J. L. Wu, "A lightweight genetic block-matching algorithm for video coding," *IEEE Trans. Circuits Syst. Video Technol.*, pp. 386–392, Aug. 1998.

[11] J. Y. Tham, S. Ranganath, M. Ranganath, and A. Kassim, "A novel unrestricted center-biased Diamond search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, pp. 369–377, Aug. 1998.

[12] J. S. Kim and R. H. Park, "A fast feature-based block matching algorithm using integral projections," *IEEE J. Select. Areas Commun.*, vol. 10, pp. 968–971, June 1992.

[13] B. Liu and A. Zaccarin, "New fast algorithm for estimation of block motion vectors," *IEEE Trans. Circuits Syst. Video Technol.*, pp. 148–157, Apr. 1993.

[14] M. Bierling, "Displacement estimation by hierarchical blockmatching," *Proc. SPIE Visual Communication and Image Processing*, vol. 100, pp. 942–951, 1988.

[15] Y.-Q. Zhang and S. Zafar, "Motion-compensated wavelet transform coding for color video compression," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, pp. 285–296, Sept. 1992.

[16] B. B. Paul and E. Viscito, "Hierarchical motion estimation with 2-scale tilings," in *Proc. IEEE Int. Conf. Image Processing*, Nov. 1994, pp. 260–264.

[17] X. Song, T. Chiang, and Y.-Q. Zhang, "A hierarchical motion estimation algorithm using nonlinear pyramid for MPEG-2," in *Proc. IEEE Int. Symp. Circuits and Systems*, June 1997, pp. 1165–1168.

[18] J. Chalidabhongse and C.-C. J. Kuo, "Fast motion vector estimation using multiresolution-spatio-temporal correlations," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 477–488, June 1997.

[19] P. Burt and E. H. Adelson, "The Laplacian pyramid as a compact image code," *IEEE Trans. Commun.*, vol. COM-31, Apr. 1983.

[20] H. Gharavi and M. Mills, "Blockmatching motion estimation algorithms—New results," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 649–651, May 1990.

[21] X. Lee, "A fast feature matching algorithm of motion compensation for hierarchical video codec," *Proc. SPIE Conf. Visual Communication and Image Processing*, vol. 1818, pp. 1462–1474, Nov. 1992.

[22] X. Lee and Y.-Q. Zhang, "A fast hierarchical motion-compensation scheme for video coding using block feature matching," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 627–635, Dec. 1996.

[23] B. Natarajan, V. Bhaskaran, and K. Konstantinides, "Low-Complexity block-based motion estimation via one-bit transforms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 702–706, Aug. 1997.

[24] X. Song, Y.-Q. Zhang, and T. Chiang, "Hierarchical motion estimation algorithm using binary pyramid with 3-scale tilings," in *Proc. SPIE Visual Communications and Image Processing Conf.*, Jan. 1998, pp. 80–87.

[25] X. Song, T. Chiang, and Y.-Q. Zhang, "A scalable hierarchical motion estimation algorithm for MPEG-2," in *Proc. IEEE Int. Symp. Circuits and Systems*, June 1998.

[26] A. Zakhor and F. Lari, "Edge-based 3-D camera motion estimation with application to video coding," *IEEE Trans. Image Processing*, vol. 2, no. 4, pp. 481–498, Oct. 1993.

[27] J. Feng, K.-T. Lo, H. Mehrpour, and A. E. Karbowiak, "Adaptive block matching motion estimation algorithm using bit-plane matching," in *IEEE Int. Conf. Image Processing*, Washington, DC, 1995, pp. 496–499.

Xudong Song (M'96) received the M.S. degree from the Institute of Information Science, Northern Jiaotong University, Beijing, China, in 1988, and the Ph.D degree in electrical engineering from Tampere University of Technology, Tampere, Finland, in 1993.

From 1993 to 1995, he was a Research Associate in the Department of Neurosurgery, University of Illinois at Chicago, in the area of medical imaging. From 1996 to 2000, he was with the Sarnoff Corporation (formerly David Sarnoff Research Center), Princeton, NJ, as a Member of the Technical Staff, where he was actively engaged in research and development of Electronic Cinema, H.263, MPEG, real-time HDTV encoding, and digital television. He joined IVAST, San Jose, CA, in 2000 as a Senior Technical Staff, where he works on multimedia. He has six US patents granted or pending. His current research interests include multimedia and video compression over Internet Protocol based networks.

Dr. Song received the Sarnoff Technical Achievement Award in 1999.

Tihao Chiang (S'90–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, Taiwan, in 1987, and the M.S. and Ph.D. degrees in electrical engineering from Columbia University, New York, in 1991 and 1995, respectively.

In 1995, he joined David Sarnoff Research Center, Princeton, NJ, as a Member of Technical Staff, and was later promoted as a Technology Leader and Program Manager. While at Sarnoff, he led a team of world-class researchers and developed an optimized MPEG-2 software encoder. Since 1992 he has actively participated in ISO's MPEG digital video-coding standardization process, with particular focus on the scalability/compatibility issue. He is currently the Co-Chair for encoder optimization on the MPEG-4 committee. He has made more than 40 contributions to the MPEG committee over the past 8 years. He has co-authored the rate control technology that was adopted as part of the MPEG-4 International Standards in 1998. 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, Taiwan, R.O.C. He currently holds two U.S. patents and has more than ten patents pending. He published over 20 technical journal and conference papers in the field of video and signal processing.

Dr. Chiang received two Sarnoff Achievement Awards and three Sarnoff Team Awards for his work in the encoder and MPEG-4 areas.

Xiaobing Lee (S'84–M'88) received the B.S.E.E. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 1981, the M.S.E.E. degree from Columbia University, New York, in 1984, and the Ph.D. degree in electrical engineering from Clarkson University, Potsdam, NY, in 1989.

He is currently Director of Research at SeaChange Systems, Greenville, NH. During 1998–1999, he was with Sarnoff Corporation, Princeton, NJ. He was a Principal Engineer at Wegener Communication Corporation during 1994–1998, a Senior Research Engineer at Tee-Comm Electronics during 1993–1994, a Research Scientist at the University Toronto, Toronto, Canada, during 1991–1992, and a Research Technologist at Southern-Westen Bell Corporation during 1989–1990. His current responsibility is the research and development of integrated MPEG multiplex and QAM devices for video servers and multimedia IP systems. His research interests include digital video streaming and video servers, video processing/compression, cable, and satellite IP/multimedia systems.

Ya-Qin Zhang (F'98) was born in Taiyuan, China, in 1966. He received the B.S. and M.S. degrees in electrical engineering from the University of Science and Technology of China (USTC) in 1983 and 1985, and the Ph.D idegree n electrical engineering from George Washington University, Washington, DC, in 1989.

He joined Microsoft Research, China, in January 1999. He was previously the Director of Multimedia Technology Laboratory at Sarnoff Corporation in Princeton, NJ (formerly David Sarnoff Research Center, and RCA Laboratories), a world leader in MPEG2/DTV, MPEG4/VLBR, and multimedia information technologies. He was with GTE Laboratories Inc., Waltham, MA, and Contel Technology Center, VA, from 1989 to 1994. He has authored and co-authored over 150 refereed papers and has 30 U.S. patents granted or pending in digital video, internet, multimedia, wireless, and satellite communications. Many of the technologies he and his team developed have become the basis for start-up ventures, commercial products, and international standards. He has been an active contributor to the ISO/MPEG and ITU standardization efforts in digital video and multimedia.

Dr. Zhang served as the Editor-In-Chief for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY from July 1997 to July 1999. He was a co-Guest Editor for the Special Issue on Advances in Image and Video Compresstion for the PROCEEDINGS OF THE IEEE in February 1995. He serves on the Editorial boards of seven other professional journals and over a dozen conference committees. He received numerous awards, including several industry technical achievement awards and IEEE awards. He was awarded as the Research Engineer of the Year in 1998 by the Central Jersey Engineering. He recently received the national award as The Outstanding Young Electrical Engineering of 1998, by Eta Kappa Nu.