On the Data Reuse and Memory Bandwidth Analysis for Full-Search Block-Matching VLSI Architecture

Jen-Chieh Tuan, Tian-Sheuan Chang, *Member, IEEE*, and Chein-Wei Jen

***Abstract*—This work explores the data reuse properties of full-search block-matching (FSBM) for motion estimation (ME) and associated architecture designs, as well as memory bandwidth requirements. Memory bandwidth in high-quality video is a major bottleneck to designing an implementable architecture because of large frame size and search range. First, memory bandwidth in ME is analyzed and the problem is solved by exploring data reuse. Four levels are defined according to the degree of data reuse for previous frame access. With the highest level of data reuse, one-access for frame pixels is achieved. A scheduling strategy is also applied to data reuse of ME architecture designs and a seven-type classification system is developed that can accommodate most published ME architectures. This classification can simplify the work of designers in designing more cost-effective ME architectures, while simultaneously minimizing memory bandwidth. Finally, a FSBM architecture suitable for high quality HDTV video with a minimum memory bandwidth feature is proposed. Our architecture is able to achieve 100% hardware efficiency while preserving minimum I/O pin count, low local memory size, and bandwidth.**

***Index Terms*—Architecture, block matching, memory management, motion estimation, video coding.**

I. Introduction

MOTION estimation has been widely employed in the H.26x, MPEG-1, -2, and -4 video standards [1] to exploit the temporal redundancies inherent within image frames. Block matching is the most popular method for motion estimation (ME). Numerous pixel-by-pixel difference operations are central to the block matching algorithms and result in high computation complexity and huge memory bandwidth. Owing to the rapid progress of VLSI technology, computation complexity requirements can easily be fulfilled by multiple PE's architecture, even for large frame sizes and frame rate. However, without enough data, these PEs can hardly be fully utilized and simply result in increased silicon area. The data rate is limited by available memory bandwidth. Therefore, straightforward implementation of ME is an I/O bound problem rather than a computation-bound one. The memory bandwidth problem may be solved by careful scheduling of the data sequence and setting up appropriate on-chip memories. Meanwhile, a well-designed ME architecture reduces the requirements of memory bandwidth and the I/O pin count, but still maintains high hardware efficiency.

Many algorithms and hardware have been dedicated to ME. The full-search block-matching (FSBM) algorithm is one these algorithms, and searches through every candidate location to find the best match. To reduce the computational complexity of FSBM, various fast algorithms were proposed [2]–[7], [19] that searched fewer points. However, these fast algorithms suffer from irregular control and lower video quality, and thus FSBM remains widespread owing to its simplicity, regularity, and superior video quality. Many architectures have been proposed for implementing FSBM. These architectures use systolic array [8]–[11], tree structure [12], or 1-D structure [14] to solve computational problem by providing enough PEs. Other architectures include a programmable architecture [31] and the integral projection-matching criterion architecture [32]. However, all of these architectures provide limited solutions to overcoming the memory bandwidth bottlenecks of high-quality video ME such as HDTV. Direct implementation is unrealistic without exploiting the locality or tricky designs. For example, the MPEG2 MP@ML format requires a memory bandwidth of tens of gigabytes per second, while the HDTV format with a large search range requires a terabytes per second bandwidth. This work only considers uni-directional ME, and bandwidth becomes even higher for bi-directional predictions. Redundancy relief is the solution to the huge bandwidth problem because many redundant accesses exist in the memory traffic of ME.

This work provides a data reuse analysis of FSBM to remove the redundancies caused by retrieving the same pixel multiple times. Regarding strength of data reuse, the present analysis extracts four data-reuse levels from the FSBM algorithm. Furthermore, a redundancy access factor Ra is provided to measure the degree of redundancy. Weaker data reuse level has a higher Ra value and demands a higher memory bandwidth. Meanwhile, a stronger data reuse level has a smaller Ra and demands lower memory bandwidth. The memory bandwidth of FSBM is a function of frame rate, frame size, search range, and Ra. The former three factors are generally fixed in video compression applications. Only Ra can be altered to accommodate the bandwidth requirement. Actually, Ra varies with the data reuse level; hence, the data-reuse level is important when designing a FSBM architecture. Besides bandwidth and data-reuse level, local memory analysis is also addressed. The local memory is set up for storing already loaded data. Local memory size increases or shrinks

Manuscript received December 15, 1999; revised October 22, 2001. This work was supported by the National Science Council of the Republic of China under Contract NSC87-2215-E009-039. This paper was recommended by Associate Editor I. Ahmad.

J.-C. Tuan is with the iCreate Technologies Corporation, Science-Based Industrial Park, Hsinchu 300, Taiwan, R.O.C. (e-mail: tuan@twins.ee.nctu.edu.tw).

T.-S. Chang is with the Global Unichip Corporation, Science-based Industrial Park, Hsinchu 300, Taiwan, R.O.C. (e-mail: tschang@globalunichip.com).

C.-W. Jen is with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: cwjen@twins.ee.nctu.edu.tw).

Publisher Item Identifier S 1051-8215(02)01122-9.

```
for (h = 0; h < (H/ N); h = h + 1) {
for (w = 0; w < (W/ N); w = w + 1) {
    MV(w, h) = (0, 0); tmpMV = (0, 0);
    MAD(w, h) = INFINITE; tmpMAD = 0;
    for (n = -p_V; n < p_V; n = n + 1) {
    for (m = -p_H; m < p_H; m = m + 1) {
        for (y = 0; y < N; y = y + 1) {
        for (x = 0; x < N; x = x + 1) {
        tmpMAD = tmpMAD + | I_k( w*N+x, h*N+y ) - I_k-1( w*N+x+m, h*N+y+n ) |;
        }    } /* end of x and y loops */
        if (tmpMAD < MAD(w, h) )
            MAD(w, h) = tmpMAD;
            MV(w, h) = (m , n);
        }
    }    } /* end of m and n loops */
}    } /* end of w and h loops */
```

Fig. 1. Motion-estimation algorithm through six nested loops.

with the data-reuse level, with a weaker data-reuse level consuming less local memory and a stronger data-reuse level requiring more. The relationship among local memory size, required memory bandwidth, and different data-reuse levels is analyzed herein. FSBM architecture designers can thus trade-off among these quantitative constraints and make suitable decisions.

The existing literature has largely overlooked the classification of FSBM arrays. [15] presented a classification derived from dependency graph (DG) transformation. This classification divides FSBM arrays into six types, each classified according to the distribution of the subtraction-absolute-accumulation (SAA) operation. However, such a classification reveals few implementation issues. Consequently, a new architectural classification for FSBM array is also proposed herein. The novel classification derives seven types of data-reuse-efficient architectures that are mutually transformable using simple transformation operators "*delay* and *rotate*". Designers can easily identify implementation considerations from the novel classification. Each type directly identifies the interconnection structure between PEs. The novel classification is based on three data sequence scheduling strategies, each representing an efficient data-reuse scheme. Since data reuse is involved, local memory is established, and the local memory issue can also be easily understood using the novel classification. The accumulation issue and its latency of different types are also addressed herein, and finally, the classified architectures are compared.

A one-access FSBM architecture for high quality video format is proposed herein. One-access indicates that each pixel is accessed just once, as in data reuse Level D and its $Ra = 1$. The memory bandwidth requirement is minimized when one-access is achieved, successfully overcoming the huge bandwidth problem in ME of high quality video. Features of the proposed design are then discussed.

The rest of this paper is organized as follows. Section II analyzes memory accesses and data reuse in ME. Section III then discusses scheduling strategies and presents a classification of data-reuse-efficient ME architectures. To maximize data reuse, Section IV proposes the one-access architecture. Conclusions are finally drawn in Section V.

II. Analysis of Data Reuse and Memory Accesses in Motion Estimation

A redundancy access factor Ra is defined to evaluate the efficiency of memory accesses used in FSBM

$$Ra = \frac{\text{total number of memory accesses in task}}{\text{pixel count in task}}. \tag{1}$$

The memory bandwidth of FSBM then becomes

$$\begin{aligned}\mathrm{BW} = f * W * H * Ra_{\text{current\_frame}} \\ + f * W * H * Ra_{\text{previous\_frame}}.\end{aligned} \tag{2}$$

Ra represents the average access count per pixel in FSBM processing, with a smaller value indicating greater reduction of memory bandwidth. When the Ra of an architecture equals one, the architecture is said to be one-access and minimum memory bandwidth is achieved.

The FSBM algorithm can be expressed using C-like language, as in Fig. 1.

In the above algorithm, p_H and p_V represent the horizontal and vertical search displacement, respectively. Meanwhile, $I_k(i, j)$ is the pixel intensity of the current frame and $I_{k-1}(i, j)$ is the pixel intensity of the previous frame.

For one frame ME, as described in Fig. 1, the operation count can range up to $W * H * \mathrm{SR}_H * \mathrm{SR}_V$ $(\mathrm{SR}_H = 2p_H, \mathrm{SR}_V = 2p_V)$ in terms of SAA operation, as indicated in (3). Each SSA operation requires two pixels. A total of $2 * W * H * \mathrm{SR}_H * \mathrm{SR}_V$ pixels are thus accessed from the frame memory. However, each frame contains only $W * H$ unique pixels. The excessive frame memory access results from each pixel having multiple accesses. Redundancy access is measured using the equation

$$Ra = \frac{W \times H \times \mathrm{SR}_H \times \mathrm{SR}_V}{W \times H} = \mathrm{SR}_H \times \mathrm{SR}_V.$$

Consequently, each pixel is accessed an average of $\mathrm{SR}_H * \mathrm{SR}_V$ times. These redundant accesses introduce a large memory bandwidth overhead.

A. Locality in Current Frame

In FSBM, each current block is independent, i.e., pixels of one current block do not overlap with other current blocks. Consequently, the lifetime of the pixels of the current block is just the time period of motion-estimating one current block. Each pixel in the block is used $\mathrm{SR}_H * \mathrm{SR}_V$ times during this period, showing that pixels of the current block have good locality compared with pixels of the search area. This characteristic allows the simple approach of keeping all $N * N$ current block pixels locally, allowing the design to result in a $1/(\mathrm{SR}_H \times \mathrm{SR}_V)$ reduction in memory accesses of the current frame. Therefore, the additional $N * N$ local memory reduces the access count of the current frame to just $W * H$ for each frame, which is also the maximum possible saving. This idea is widely applied in many proposed ME architectures for minimizing current frame bandwidth. Consequently, the redundancy access of the current frame is as shown in the equation at the bottom of the page.

B. Locality in Previous Frame

Each search area in the previous frame is a $(N + \mathrm{SR}_H - 1) * (N + \mathrm{SR}_V - 1)$ rectangle centered around related current blocks. Adjacent search areas thus overlap and are no longer independent. The locality of search area data has two types: local locality and global locality. Local locality covers data reuse within a single search area, regardless of overlapping among other search areas. Global locality covers data reuse among different search areas. Four data-reuse levels are defined according to the degree of data reuse: from Level A (weakest reuse degree) to Level D (strongest reuse degree). The data-reuse level is an important factor in dealing with the memory bandwidth requirements of ME. Because stronger reuse level reduces Ra, i.e., less memory bandwidth is required. Furthermore, in most applications, only Ra can be varied to reduce memory bandwidth, as shown in (2). The four reuse levels are defined below.

1) Level A—Local Locality Within Candidate Block Strip: Local locality can be further divided into locality within the same row and locality between adjacent rows. A row of candidate blocks is called a candidate block strip, as shown in upper half of Fig. 2. The extension direction of the candidate block strip is arbitrarily chosen. For the two consequent candidate blocks of two motion vectors (MVs) given, pixels are significantly overlapped. The pixel access count without any data reuse is $2 * N * N$ for calculating the two MVs, but only $N * (N + 1)$ individual pixels exists. Candidate blocks #1 and #2 in Fig. 2 demonstrates this situation. Consequently, while dealing with candidate block #2, the overlapped data of candidate block #1 can be reused without accessing the frame memory again. The data-reuse approach can be extended to the

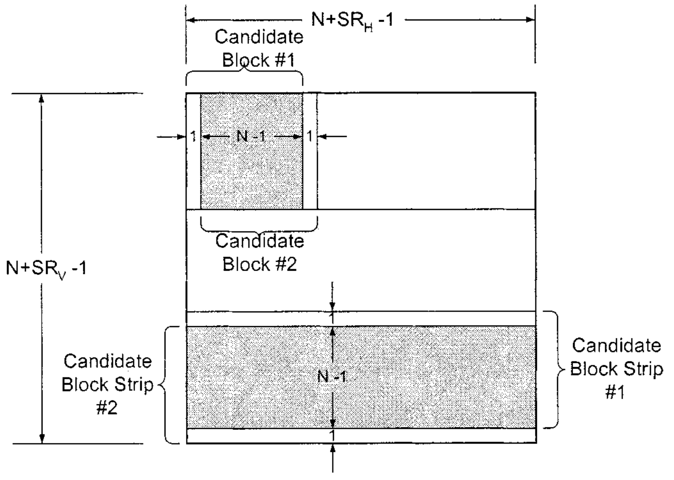


Fig. 2. Level A schematic: local locality within candidate block strip. Level B schematic: local locality among adjacent candidate block strips.

entire candidate block strip, i.e., pixels can be repeatedly used for adjacent MV calculations.

2) Level B—Local Locality Among Adjacent Candidate Block Strips: Vertically adjacent candidate block strips also overlap significantly, as demonstrated by candidate block strips #1 and #2 in the lower half of Fig. 2. The size of the overlapping region of two candidate block strips is $(N+2*p_H-1)*(N-1)$. The pixels in the overlapped region can be reused while processing the next candidate block strip, exploiting the locality of adjacent candidate block strips. This idea is extended to all candidate block strips within the search area.

3) Level C—Global Locality Within Search Area Strip: The global locality within the search area strip describes data reuse among different search areas corresponding to the location of their current blocks within the same search area strip. The search area strip is like an up-sampled version of the candidate block strip, and is formed by entire rows of search area. The derivation resembles the scheme in Level A, and thus the formula and derivations are ignored herein.

4) Level D: Global Locality Among Adjacent Search Area Strips: Level D resembles Level B, except that it applies to the search area strips. Level D repeatedly reuses data already loaded from former search area strips for latter search area strips. Applying this level, one-access is achieved.

C. Bandwidth Requirement of Data-Reuse Levels

Equation (2) reveals that required bandwidth is actually determined by Ra. The Ra factor for each data-reuse level can be calculated as shown in the equation at the bottom of the next page.

For simplicity, the above calculations assume $W \gg \mathrm{SR}_H$ and $H \gg \mathrm{SR}_V$. The above four levels provide a good indication of the amount of memory bandwidth required for a given FSBM

$$Ra = \frac{(\text{\# of current blocks}) \times (\text{\# of memory accesses per current block})}{W \times H} = \frac{(W/N) \times (H/N) \times N \times N}{W \times H} = 1$$

TABLE I
BANDWIDTH REQUIREMENTS AND DATA REUSE LEVEL OF VARIOUS VIDEO FORMATS

Format	Frame Rate	Search Range		Reuse Level vs. Bandwidth (MB/sec)			
	f	SR$_H$	SR$_V$	A	B	C	D
CIF	25	16	16	83	12	7.6	5.1
	25	32	32	245	25	10	5.1
NTSC	30	16	16	342	52	31	21
	30	32	32	1,006	104	41	21
GA HDTV (1280x720)	60	32	32	5,364	553	221	111
	60	64	64	17,750	1,438	332	111
	60	256	64	60,217	4,755	332	111
MP@HL	30	64	64	20,117	1,629	376	125
	30	256	64	68,246	5,390	376	125

design. Table I presents the bandwidth requirements for various common video formats corresponding to different data-reuse levels using (2). The required bandwidth easily reaches tens of gigabytes per second for high-quality video with weak data reuse, but falls significantly to sub-gigabytes per second when a higher data-reuse level is adopted. This phenomenon provides a good reason for considering stronger data-reuse level to lower frame memory bandwidth, or using low-cost memory modules or narrow bus width. Wide bus makes the chip package more costly and increases problems with skew. Furthermore, from a systematic point of view, memory traffic introduced by other components such as variable-length-codec, DCT/IDCT, and so on must also be considered. Additionally, the motion-estimation process only uses the luminance pixel data, while the other components also use chrominance data thus increasing the importance of strong data-reuse level.

All four of the above reuse levels are intra-frame data reuse. There is still another inter-frame reuse level. For example, if we motion estimate frame $\#(k)$, frame $\#(k)$ serves as the current frame and frame $\#(k-1)$ serves as the previous frame. When advancing to frame $\#(k+1)$, frame $\#(k+1)$ servers as the current frame and frame $\#(k)$ becomes the previous frame. Notably, frame $\#(k)$ is actually accessed twice, and this reuse can be defined as Level E, the ultimate data-reuse level. However, achieving data reuse Level E involves a significant penalty, i.e., storing at least one frame. Furthermore, bi-directional ME, such as that used in MPEG2, has to store more than one frame, because this data-reuse level predicts across several frames. Level E implementation is impractical, so only Levels A–D are considered herein.

D. Local Memory Size Derivation

To realize data reuse, local memory holding already loaded data for further access is required. The local memory required for the current frame, i.e., for storing one current block is $N*N$.

$$\begin{aligned}
\text{Level A: } Ra &= \frac{\text{(\# of candidate block strips)} \times \text{(\# of memory access per candidate block strip)}}{\text{Previous Frame Size}} \\
&= \frac{W/N \times H/N \times \mathrm{SR}_V \times N \times (N + \mathrm{SR}_H - 1)}{W \times H} \\
&\approx \mathrm{SR}_V \times \left(1 + \frac{\mathrm{SR}_H}{N}\right) \\
\text{Level B: } Ra &= \frac{\text{(\# of search area)} \times \text{(\# of access of memory per search area)}}{\text{Previous Frame size}} \\
&= \frac{W/N \times H/N \times (N + \mathrm{SR}_H - 1) \times (N + \mathrm{SR}_V - 1)}{W \times H} \\
&\approx \left(1 + \frac{\mathrm{SR}_V}{N}\right) \times \left(1 + \frac{\mathrm{SR}_H}{N}\right) \\
\text{Level C: } Ra &= \frac{\text{(\# of search area strips)} \times \text{(\# of memory access per search area strip)}}{\text{Previous Frame Size}} \\
&= \frac{H/N \times (W + \mathrm{SR}_H - 1) \times (N + \mathrm{SR}_V - 1)}{W \times H} \approx \left(1 + \frac{\mathrm{SR}_V}{N}\right) \\
\text{Level D: } Ra &= \frac{\text{(\# of memory access of all search area strips)}}{\text{Previous Frame Size}} = \frac{W \times H}{W \times H} = 1
\end{aligned}$$

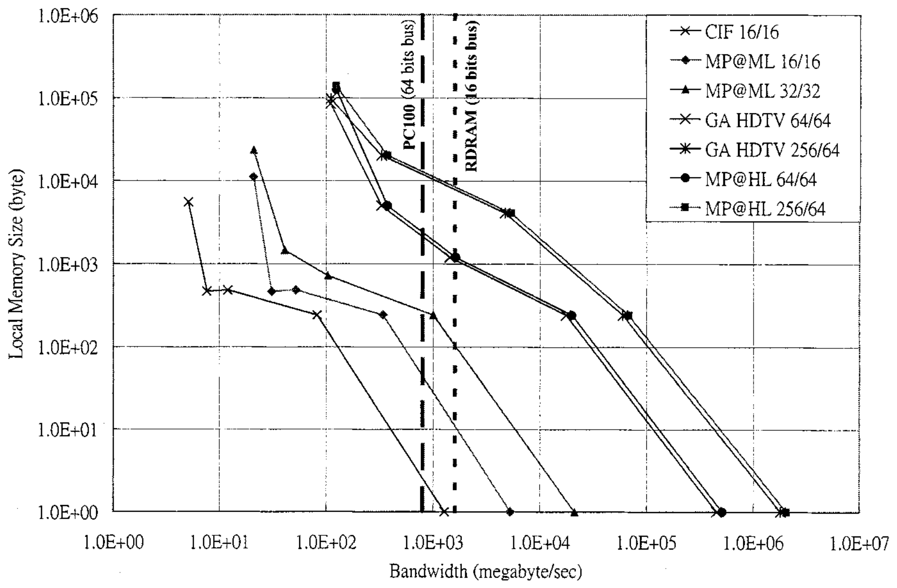


Fig. 3. Relationship between bandwidth requirements and local memory size for various video formats and data-reuse levels.

TABLE II
LOCAL MEMORY SIZE FOR DIFFERENT REUSE LEVELS OF PREVIOUS FRAMES

Reuse Level	Local Memory Size
A	N * (N - 1)
B	(N + SR_H) * (N - 1)
C	(N + SR_V - 1) * (SR_H - 1)
D	(W + SR_H - 1) * (SR_V - 1)

The local memory size for the previous frame equals the size of the overlapped region in each data-reuse level. Table II lists the derived size, which theoretically should be the upper limit of a novel design. Actually however, few overheads exist for preloading or scheduling data. A properly scheduled data sequence can eliminate the need for local memory. Section IV discusses the scheduling. Meanwhile, Fig. 3 presents a quantitative example of local memory size versus bandwidth requirements for different video formats. Each curve consists of five data points, with the upper four points representing Levels D (upper most) to A (lower most), while the lowest point represents no data reuse. The numbers labeling the different video formats are the search ranges, $\mathrm{SR}_h/\mathrm{SR}_v$. Meanwhile, the two dashed vertical lines, 1.6 GB/s for RDRAM (16-bit bus) and 800 MB/s for PC100 (64-bit bus), indicate the peak bandwidth of current commodity memories [34]. Even the RDRAM scheme was found to be unable to satisfy the Level B and A bandwidth requirements of high-resolution video formats. Meanwhile, the PC100 scheme cannot afford a bi-directional Level C requirement. The analysis presented in Fig. 3 outlines a good tradeoff decision.

III. CLASSIFICATION OF DATA REUSE-EFFICIENT ME ARCHITECTURES

This section provides a new classification that directly addresses data-reuse efficiency. While many ME architectures have been proposed, few ME classifications have been mentioned. DG transformation [9], [15], and [28] can serve as a basis for classification through different mapping procedures. [15] proposes a classification for FSBM architectures which is derived from DG transformation and which divides FSBM arrays into six different types. Different arrays of the same type are characterized by the distribution of SAA operations among PEs. However, such classifications reveal little about implementation. Numerous important implementation considerations are easily identified from our classification.

A. Scheduling Strategies

Three scheduling strategies for the array processor are derived according to the property of data reuse. The three strategies chosen have an important common property: heavy exploiting of the data reuse of FSBM in the spatial or temporal domains. This property removes redundancies in FSBM and helps to develop a data-reuse-efficient architecture. Most novel FSBM architectures, both 1- and 2-D, are covered by these three scheduling strategies, which are defined as follows.

Concurrent Mode: This strategy is a data broadcast scheme, i.e., the same pixel is used simultaneously by multiple PEs. Using the DG projection procedure mentioned in [27], [28] to demonstrate this strategy, the six-loop algorithm in Fig. 1 is folded into two loops: $[h, w, y, x, n, m] \rightarrow [(h, w, n, y, x), m]$, as shown in Fig. 4. Then, processor basis $\mathbf{P}^T = [0\ \ 1]$ and scheduling vector $\bar{s}^T = [1\ \ 0]$ are chosen. $\mathbf{P}^T$ defines the mapping from DG nodes to PEs and $\bar{s}^T$ defines the execution order of nodes following the procedure in [27], [28]. From a hardware perspective, $\mathbf{P}^T$ defines the number of PEs and $\bar{s}^T$ defines the PE interconnection and execution order. This projection defines the data-reuse property of current block pixel

```
for (i = 0; i < H*W*SR_V ; i = i + 1) {
    h = i / (W*SR_V*N), w = [ i / (SR_V*N*N) ] % (W/N);
    n = [ i / (N*N) ] % SR_V - p_V , y = ( i / N ) % N , x = ( i % N );
    for (m = -p_H; m < p_H; m = m + 1) {
        tmpMAD = tmpMAD + | I_k( w*N+x, h*N+y ) - I_k-1( w*N+x+m, h*N+y+n ) |;
    } /* end of m loop */
} /* end of i loop */
```

Fig. 4. Motion-estimation algorithm folded into two nested loops. The MV calculation is omitted.

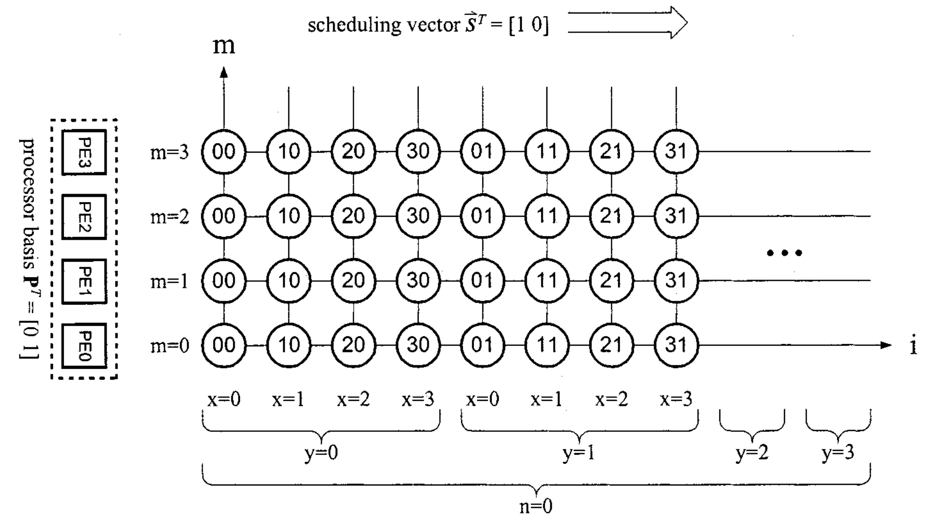

scheduling vector $\vec{s}^T = [1\ 0]$
m
processor basis $\mathbf{P}^T = [0\ 1]$
PE3
PE2
PE1
PE0
m=3
m=2
m=1
m=0
00
10
20
30
01
11
21
31
i
x=0
x=1
x=2
x=3
y=0
y=1
y=2
y=3
n=0

Fig. 5. Concurrent mode example for current block data, with the indexes being simplified from folded loops: $N = \mathrm{SR}_H = \mathrm{SR}_V = 4$.

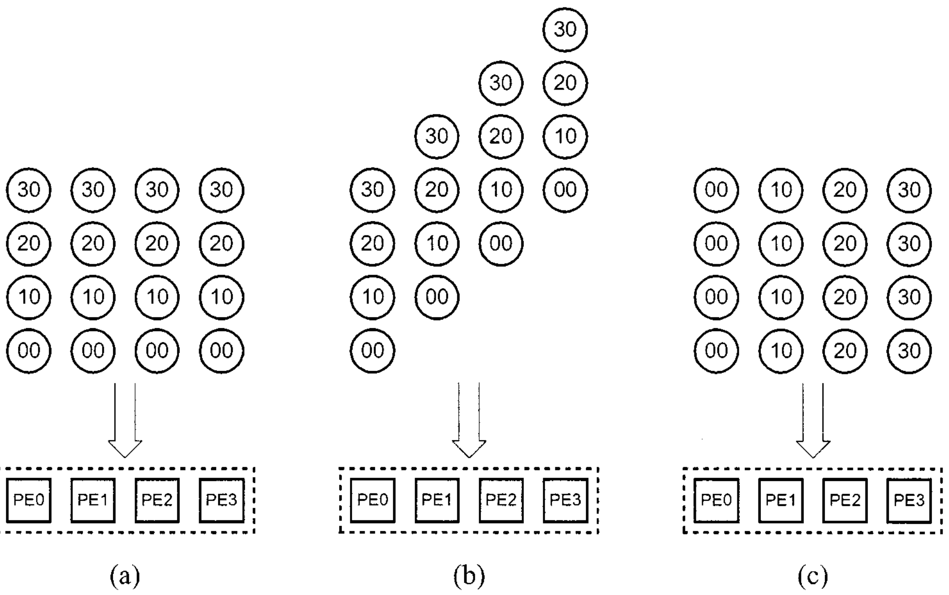


Fig. 6. Data sequence example involving three scheduling modes. (a) Concurrent mode. (b) Pipeline mode. (c) Store mode.

$I_k(i, j)$ as the concurrent mode, the data sequence example of which is displayed in Fig. 5. Owing to the broadcasting scheme of the concurrent mode, local memory and interconnection between PEs is unnecessary, regardless of the local memory size derived in Table II.

Pipeline Mode: The projection parameters for this strategy are $\mathbf{P}^T = [0\ 1]$ and $\vec{s}^T = [1\ 1]$, and the data-reuse property of current block pixel $I_k(i, j)$ is defined as the pipeline mode. Fig. 6(b) illustrates the data sequence example. For this strategy each PE must propagate its currently used data to the neigh-

TABLE III
CLASSIFICATION OF SEVEN ARRAY TYPES

Type	CCSP	CPSC	CSSC	CSSP	CCSS	CPSS	CPSP
Scheduling strategy	CB† concurrent, SA‡ pipeline	CB pipeline, SA concurrent	CB store, SA concurrent	CB store, SA pipeline	CB concurrent, SA store	CB pipeline, SA store	CB pipeline, SA pipeline
Folded loops	[(h,w,n,y,x), m]	[(h,w,n,y,x), m]	[(h,w,n,y,m), x]	[(h,w,n,y,m), x]	[(h,w,n,y,x), m]	[(h,w,n,y,m), x]	[(h,w,n,y,x), m]
$\mathbf{P}^T$	[0 1]	[0 1]	[0 1]	[0 1]	[1 −1]	[1 −1]	[0 1]
$\bar{s}^T$	[1 0]	[1 1]	[1 1]	[1 0]	[1 0]	[1 0]	[1 2]
1-D architecture	b.c. of [14], [16], [30]	b.s. of[14], [16], [23]	[15]	[22]			*AS1* of [9]
2-D architecture	type-2 of [8]	[11], [24], [25], [26]	[18], [19]	type-1 of [8], *AB2* of [9], [10]	[12], [21]	[29]	*AS2* of [9]

†: current block; ‡: search area

boring PE at the next cycle, meaning that both interconnection between neighbor PEs and local memory for holding data are required.

Store Mode: The folded algorithm is $[(h,w,n,y,m),x]$. With projection parameters of $\mathbf{P}^T = [1\ \ 0]$ and $\bar{s}^T = [0\ \ 1]$, the data-reuse property of current block pixel $I_k(i,j)$ is defined as the store mode. Fig. 6(c) shows the data sequence example. In this strategy, data are only saved and used within each PE, meaning that only local memory has to be set up.

Though only the current block data is demonstrated herein, the search area pixels are simultaneously co-scheduled using the three strategies to maximize data reuse. Various co-scheduled data-reuse-efficient architectures are classified below.

B. Classification of Seven Array Types

This classification covers all seven types of efficiently co-scheduled architecture. The folded loops and projection parameters $\mathbf{P}^T$ and $\bar{s}^T$ of each type are described below. While the parameters below are used for 1-D architectures, 2-D architectures can be obtained in a similar fashion. Using these parameters to derive the PE and interconnection structures through DG mapping methodology is a simple matter. The published 1- and 2-D architectures relating to each type are also included, as classified in Table III.

These seven types can be transformed using the transformation operators "*delay*" and "*rotate*". These two operators can convert one scheduling mode to another. The concurrent and pipeline modes are convertible through a delay operation, i.e., inserting delay elements into the data input sequence as illustrated in Fig. 7. The pipeline and store modes can be converted by a rotation operation as illustrated in Fig. 8. Each row of the input data sequence in Fig. 8 is "left-rotated" according to the rotation distance, with "0" meaning no rotation, "1" meaning left-rotate one node position, "2" meaning left-rotate two node positions, and so on.

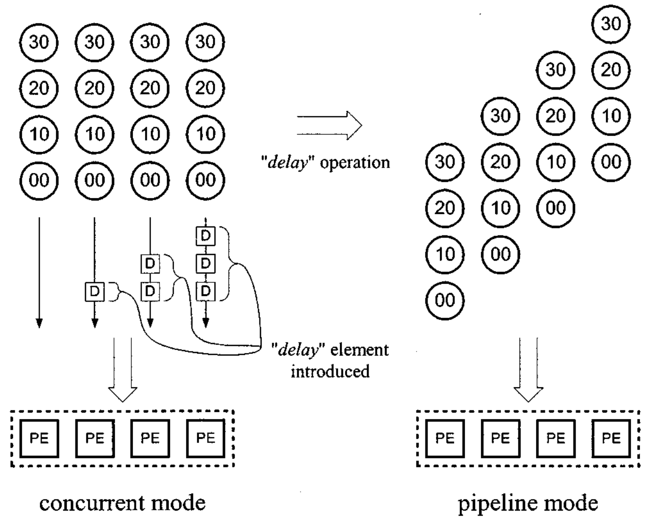


Fig. 7. The "*delay*" operation.

Owing to the complexity of conversion between the concurrent mode and the store mode, this conversion is not used herein. Table IV lists the transformations between different types. Actually, these conversions can be obtained via different traditional systolic array mapping methods, but the "*delay* and *rotate*" method is easier to understand and is a more intuitive approach.

C. Comparison of Architectures

Table V lists a theoretical 2-D architecture comparison. These architectures are designed to minimize I/O ports, while maintaining 100% hardware efficiency. The 1-D comparison is not

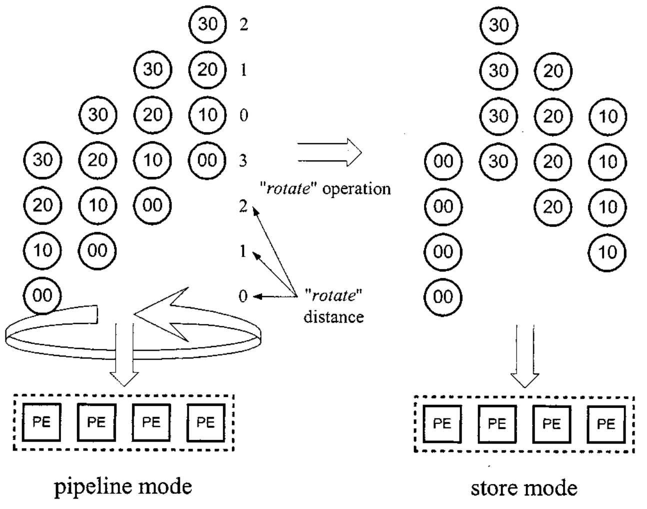


Fig. 8. The "*rotate*" operation.

TABLE IV
LIST OF TRANSFORMATIONS BETWEEN DIFFERENT TYPES

Concurrent $\xleftrightarrow{\text{DELAY}}$ Pipeline	Pipeline $\xleftrightarrow{\text{ROTATE}}$ Store
CCSP $\xleftrightarrow{\text{DELAY}}$ CPSC	CCSP $\xleftrightarrow{\text{ROTATE}}$ CCSS
CSSC $\xleftrightarrow{\text{DELAY}}$ CSSP	CPSC $\xleftrightarrow{\text{ROTATE}}$ CSSC
CPSC $\xleftrightarrow{\text{DELAY}}$ CPSP	CSSP $\xleftrightarrow{\text{ROTATE}}$ CPSS
CPSP $\xleftrightarrow{\text{DELAY}}$ CCSP	

TABLE V
COMPARISON WITH $N = 16, \mathrm{SR}_H = \mathrm{SR}_V = 32$

Architecture	# of PE	BW (Mbyte/sec)	Input Pin Count	Utilization
Type-2 of [8]	1024	247	32	100%
[11]	1024	564	72	100%
[18]	1024	564	72	100%
[10]	256	603	16	47%
[12]	256	247	16	100%
Ours	256	125	16	100%

included because it is simpler and less revealing. To allow a fair comparison, the parameters $N = \mathrm{SR}_H = \mathrm{SR}_V = 16$ are assumed. The table derives the theoretical local memory (L.M.), size of the current block (C.B.), size of the search area (S.A.), accumulation type (Acc. Type), cost of the accumulation type (Acc. Cost), and the latency for completing one current block task. The latency excludes the adder and comparator overheads. Four kinds of accumulation structures exist, namely self accumulation, propagating accumulation, circulating accumulation, and parallel accumulation, as shown in Fig. 9, and the actual

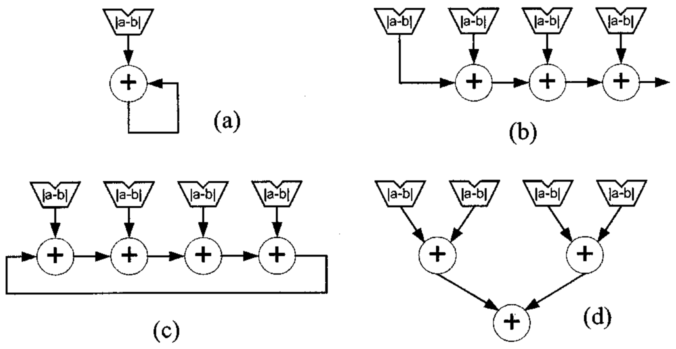


Fig. 9. Four kinds of accumulation structure. (a) Self accumulation. (b) Propagating accumulation. (c) Circulating accumulation. (d) Parallel accumulation.

accumulation structure can be found in related works. The accumulation cost is the total adder bits of accumulator. Notably, some fields of the local memory size are zero, a phenomenon that is caused by the broadcasting mechanism. From the perspective of VLSI implementation, the broadcast mechanism will cause significant load capacitance and skew problems. Architecture [33] uses a slower data input rate to overcome the problem of current block broadcasting. This approach is possible because the tasks in [33] are interleaving processed by fewer PEs, and thus the required data input rate of the current block is scaled down. But fewer PEs produce less computational power unless operating frequency is increased. A similar problem occurs when a shared bus is used to carry the final MADs, like the architectures in [12], [14], [16], [30], and [33]. This problem is solved in [33] by the slower data rate, as mentioned above.

All of the 2-D architectures in Table VI apply data reuse Level C, the data-reuse level achieved by most proposed architectures. Levels A and B are inefficient and cost more I/O pins for 2-D architecture design. Meanwhile, Level D needs more local memory because of the stronger data reuse involved. Consequently, the local memory structure and its interface to PEs must be carefully designed. Section IV discusses the Level D architecture design.

IV. ONE-ACCESS ARCHITECTURE

This section presents the FSBM architecture design of data reuse Level D, i.e., the one-access architecture. The architecture is of the CPSS type. The proposed architecture mostly comprises a ME core and some on-chip memory. Since it is inefficient to store entire frame in on-chip memory, off-chip memory (such as SDRAM) is also used. Off-chip memory bandwidth in high quality video ME with a large frame size and wide search range is the main barrier in designing a cost-efficient architecture. The proposed architecture eliminates the frame memory bandwidth bottleneck by exploiting the maximum data reuse and successfully reduces bandwidth to a level easily handled by commodity memory. Carefully local memory setup optimizes the local memory size to a near minimum value and few overheads are introduced. Simple and regular interconnections ensure high-speed operation, while efficient and distributed local memory organization is used to provide enough data. Just two I/O ports are required, one for the current block data and the other for the search area, but 100% hardware efficiency is still achieved.

TABLE VI
COMPARISON OF 2-D ARCHITECTURES

	CCSP	CPSC	CSSC	CSSP	CCSS	CPSS	CPSP
# of PE	256	256	256	256	256	256	256
L.M. of C.B.	0	256	256	512	0	512	512
L.M of S.A.	512	0	0	512	512	512	512
Acc. Type	Self Acc.	Self Acc.	Prop. Acc.	Para. Acc.	Circ. Acc	Para. Acc.	Self Acc.
Acc. Cost	4096	4096	3825	2287	4096	2287	4096
Latency	256	511	511	256	256	256	766

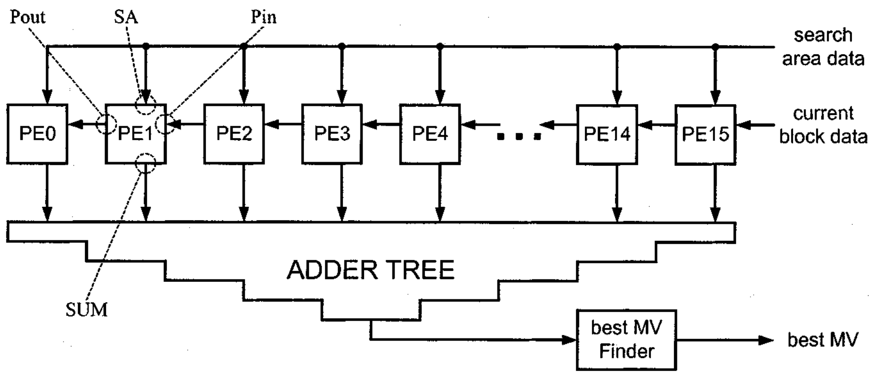


Fig. 10. Overview of the proposed CPSS type architecture and the interconnections of PE ports Pin, Pout, SA, and SUM.

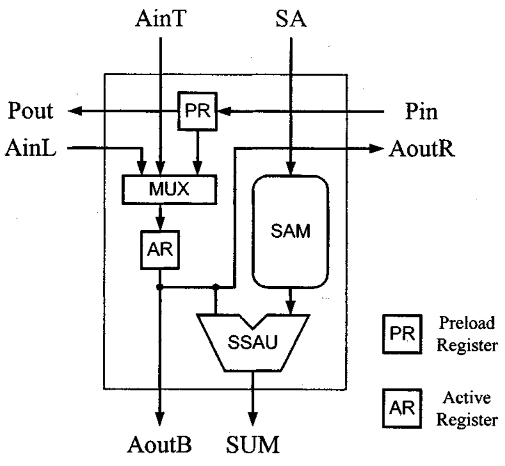


Fig. 11. PE structure.

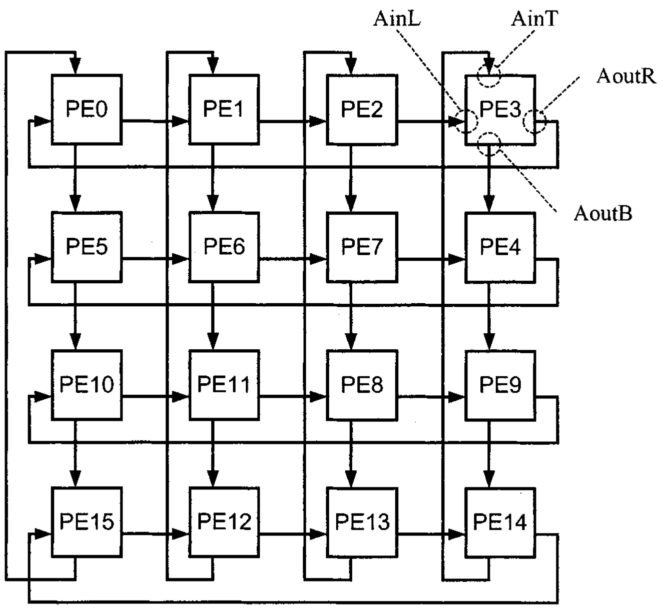


Fig. 12. Data path of current block data and the interconnections of ports AinT, AoutB, AinL, and AoutR.

A. Architecture Overview

The architecture proposed herein is a 2-D design with $N * N$ PEs. For ease of illustration, $N = 4$ and $p_V = p_H = 4$ were selected. Consequently, total of 16 PEs exist, and the search range is 8. Fig. 10 presents an overview of the one-access architecture. The overview shows the three top module I/O ports and their connections, and reveals the two data-loading paths of the search area and the current block data. The search area data are loaded into PEs in series using a shared bus, and the current block data are loaded in a propagating fashion. The data path of MAD calculation is also displayed, and is a parallel adder tree design. At the output of the final stage of the adder tree, the best MV Finder determines the best motion vector. Notably, the

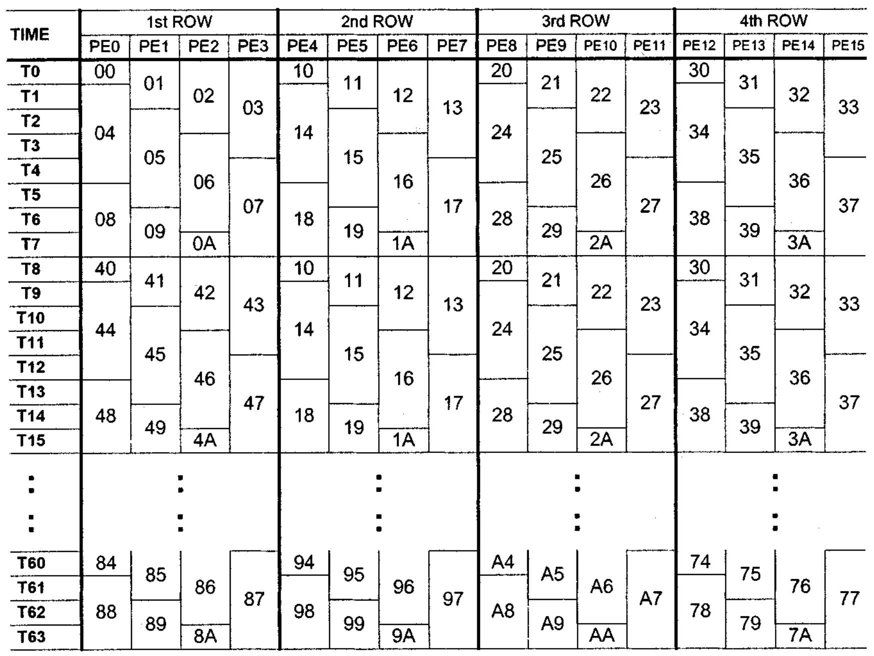

TIME
1st ROW
2nd ROW
3rd ROW
4th ROW
PE0 PE1 PE2 PE3 PE4 PE5 PE6 PE7 PE8 PE9 PE10 PE11 PE12 PE13 PE14 PE15
T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15
T60 T61 T62 T63

Fig. 13. Data sequence of the search area pixels.

architecture is illustrated in a 1-D scheme for simplicity; it is actually a 2-D architecture.

Fig. 11 shows the PE structure. Each PE contains one active register (AR) for holding active current block data, and one preload register (PR) for propagating the current block data of the next macroblock. The data provided by PRs fill the time gap between successive macroblocks. The multiplexer (MUX) selects data from either AR or PR. A search area memory (SAM) stores necessary search area data for data reuse. Finally, an SSA unit (SSAU) calculates the absolute difference between current block and search area data. Eight ports exist per PE, and Figs. 10 and 12 illustrate the interconnections of these ports. Notably, the adder tree is ignored in Fig. 12 for simplicity.

B. ME Operation

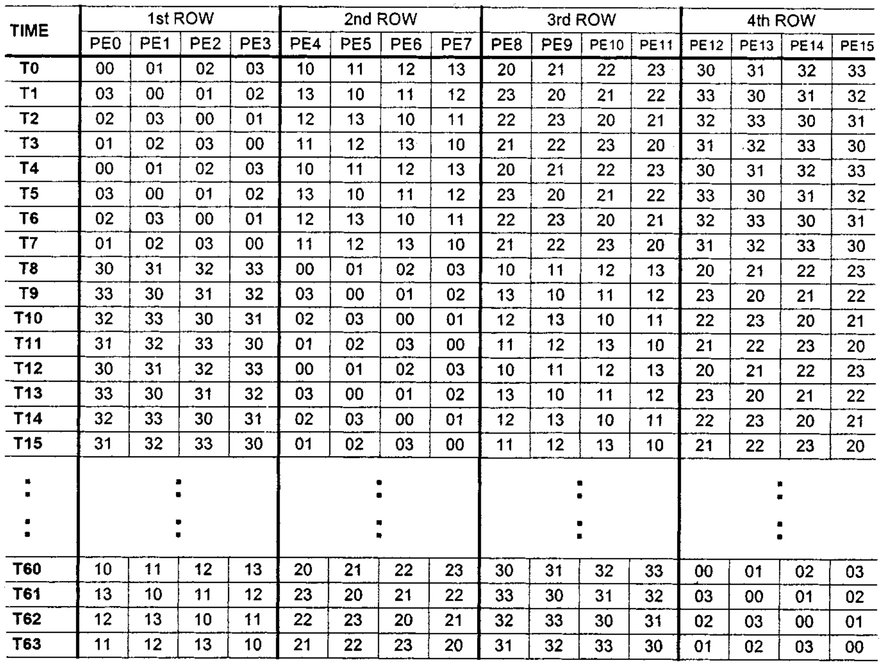
The partial MAD calculation (3) is distributed among all PEs in the novel architecture. The SSAU is responsible for calculating the absolute difference between current block pixel and search area pixel, and thus it defines which of the pixels are paired with PEs and how. The search area data are distributed among SAMs. If the search area pixel are labeled $I_{k-1}(i,j)$, where $0 \le W, 0 \le j \le H$, then $I_{k-1}(i,j)$ will be distributed to SAM of PE#$((j\%N) * N + i)$. Consequently, SAMs differ in size, and their size can be calculated according to the pixel distribution just mentioned. Total on-chip SAM size can then be calculated by summing all SAMs. The search area data are static within the SAM and will never be passed to another SAM, making the connection between SSAU and SAM very simple. Fig.13 lists the sequence of the search area data, while Fig. 14 lists the sequence of the corresponding current block data. Examining the data sequence in Fig. 14 and the circulating path in Fig. 12 easily reveals the horizontal and vertical circulation property of the current block.

At the beginning of a macroblock processing, both current block data and search area data must be prepared immediately to avoid idle cycles. The current block data are pumped in parallel from PRs into ARs, and the search area data are prepared concurrently from the SAMs. [29] lists further transaction details.

C. Performance and Comparison

The proposed ME architecture is characterized by low memory bandwidth, low latency, minimum I/O port, 100% hardware efficiency, and simple interconnections among PEs. Memory bandwidth is reduced to just twice the luminance bit rate of video sequence by using the data reuse Level D. The memory bandwidth of ME is only influenced by frame rate and frame size, regardless of changes in search range. The low latency delay is due to the parallel loading of both the search area and current block data. The theoretical latency for the processing of each macroblock is $((\mathrm{SR}^2 \times N^2)/\mathrm{PE\ count})$. For $N = 16, \mathrm{SR}_H = \mathrm{SR}_V = 32$ and $\mathrm{PE\ count} = 256$ (a real video compression case), this architecture can complete each macroblock in just 1024 clock cycles, plus the overhead introduced from the adder tree and MV finder. This architecture has only two I/O ports, one for loading current block data and the other for loading search area data. Avoiding transition gap between macroblocks is essential to 100% utilization. Both current block and search area data should be prepared before next macroblock to avoid transition gap. The current block data are prepared by PRs and the search area data are prepared by SAMs. The data preparation also applies to macroblocks at frame boundaries, thus the one-access architecture is able to be 100% utilized via a suitable data loading sequence. The detailed transitions were provided in reference [29]. The interconnection linking PEs is simple and local.

Tables V and VII compare the proposed architecture and other architectures under different parameters. The comparisons use the HDTV video format $(f = 30, W = 1920, H = 1088)$ to show the impact of large frame size. The moving distance



TIME	1st ROW				2nd ROW				3rd ROW				4th ROW			
	PE0	PE1	PE2	PE3	PE4	PE5	PE6	PE7	PE8	PE9	PE10	PE11	PE12	PE13	PE14	PE15
T0	00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33
T1	03	00	01	02	13	10	11	12	23	20	21	22	33	30	31	32
T2	02	03	00	01	12	13	10	11	22	23	20	21	32	33	30	31
T3	01	02	03	00	11	12	13	10	21	22	23	20	31	32	33	30
T4	00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33
T5	03	00	01	02	13	10	11	12	23	20	21	22	33	30	31	32
T6	02	03	00	01	12	13	10	11	22	23	20	21	32	33	30	31
T7	01	02	03	00	11	12	13	10	21	22	23	20	31	32	33	30
T8	30	31	32	33	00	01	02	03	10	11	12	13	20	21	22	23
T9	33	30	31	32	03	00	01	02	13	10	11	12	23	20	21	22
T10	32	33	30	31	02	03	00	01	12	13	10	11	22	23	20	21
T11	31	32	33	30	01	02	03	00	11	12	13	10	21	22	23	20
T12	30	31	32	33	00	01	02	03	10	11	12	13	20	21	22	23
T13	33	30	31	32	03	00	01	02	13	10	11	12	23	20	21	22
T14	32	33	30	31	02	03	00	01	12	13	10	11	22	23	20	21
T15	31	32	33	30	01	02	03	00	11	12	13	10	21	22	23	20
:		:				:				:				:		
T60	10	11	12	13	20	21	22	23	30	31	32	33	00	01	02	03
T61	13	10	11	12	23	20	21	22	33	30	31	32	03	00	01	02
T62	12	13	10	11	22	23	20	21	32	33	30	31	02	03	00	01
T63	11	12	13	10	21	22	23	20	31	32	33	30	01	02	03	00

Fig. 14. Data sequence of the current block pixels.

TABLE VII
COMPARISON WITH $N = 16, \mathrm{SR}_H = \mathrm{SR}_V = 64$

Architecture	# of PE	BW (Mbyte/sec)	Input Pin Count	Utilization
Type-2 of [8]	4096	372	48	100%
[11]	4096	2,068	264	100%
[18]	4096	2,068	264	100%
[10]	256	1,590	16	66%
[12]	256	372	16	100%
Ours	256	125	16	100%

of a macroblock increases with the frame size (assuming the same macroblock size), thus enlarging the search range of the high-resolution video. To show the influence of the large search range, $N = 16$ and $\mathrm{SR}_H = \mathrm{SR}_V = 32$ are used in Table V, while $N = 16$ and $\mathrm{SR}_H = \mathrm{SR}_V = 64$ are used in Table VII. The comparisons focus on frame memory bandwidth, input pin count, and hardware utilization; each pixel is assumed to have 8-bit width, and the bandwidth is measured in megabytes per second. Tables V and VII display that the novel architecture maintains the same low memory bandwidth and input pin count when the search range changes, while still achieving 100% hardware utilization, making it suitable for high quality video formats.

V. Conclusion

This work analyzes the I/O bound problem in ME and proposes several methods of reusing data to solve this problem. Different degrees of data reuse are classified into four levels. Greater data reuse requires lower frame memory bandwidth, but uses more local memory. Data reuse Level D achieves the one-access goal of maximum saving in memory bandwidth. Local memory sizes are derived according to different data-reuse levels. Three elegant scheduling strategies dedicated to the interface between PEs and local memories are then discussed, and a simple mapping procedure is adopted to map the strategies into the hardware. These selected interface varieties can simplify interconnection networks. Applying the three modes to schedule data sequences of both the previous and current frames results in a classification of seven implementation types, from which the costs of different interface modes can easily be depicted. This classification can serve as a basis for analyzing or evaluating existing ME architectures and makes it easier for new designs. Finally, a one-access FSBM architecture that minimizes bandwidth requirements is proposed. This architecture is characterized by: 1) minimum memory bandwidth requirements; 2) low latency; 3) minimum I/O pin count; 4) 100% hardware efficiency; and 5) simple and regular interconnections, features that make it suitable for high quality video ME. The low memory bandwidth also benefits from lower power consumption due to fewer bus transition activities and the adoption of inexpensive frame memory modules. Besides, other modules can obtain more bandwidth owing to memory bandwidth relief from ME in a complete video compression system.

References

[1] K. R. Rao and J. J. Hwang, *Techniques and Standards for Image, Video and Audio Coding*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[2] 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.

[3] T. Koga *et al.*, "Motion compensated interframe coding for video conferencing," in *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, Nov. 1981, pp. G5.3.1–G5.3.5.

[4] L. D. Vos, "VLSI architecture for the hierarchical block matching algorithm for HDTV applications," *SPIE/VCP*, vol. 1360, pp. 398–409, Nov. 1990.

[5] B. M. Wang, "An efficient VLSI architecture of hierarchical block matching algorithm in HDTV applications," in *Int. Workshop on HDTV*, Oct. 1993.
[6] B. M. Wang, J. C. Yen, and S. Chang, "Zero waiting cycle hierarchical block matching algorithm and its array architectures," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, pp. 18–28, Feb. 1994.
[7] R. Srinivasan and K. R. Rao, "Predictive coding based on efficient motion estimation," *IEEE Trans. Commun.*, vol. COM-33, pp. 888–896, Aug. 1985.
[8] L. De Vos and M. Stegherr, "Parameterizable VLSI architectures for the full-search block-matching algorithm," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1309–1316, Oct. 1989.
[9] T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1302–1308, Oct. 1989.
[10] C.-H. Hsieh and T.-P. Lin, "VLSI architecture for block-matching motion estimation algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, pp. 169–175, June 1992.
[11] H. Yeo and Y. H. Hu, "A novel modular systolic array architecture for full-search block matching motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 407–416, Oct. 1995.
[12] Y.-K. Lai *et al.*, "A novel scaleable architecture with memory interleaving organization for full search block-matching algorithm," *Proc. 1997 IEEE Int. Symp. Circuits and Systems*, pp. 1229–1232, June 1997.
[13] Y.-S. Jehng *et al.*, "An efficient and simple VLSI tree architecture for motion estimation algorithms," *IEEE Trans. Signal Processing*, vol. 41, pp. 889–900, Feb. 1993.
[14] K.-M. Yang *et al.*, "A family of VLSI designs for the motion compensation block-matching algorithm," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1317–1325, Oct. 1989.
[15] S. Chang *et al.*, "Scaleable array architecture design for full search block matching," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 332–343, Aug. 1995.
[16] S. Dutta and W. Wolf, "A flexible parallel architecture adapted to block-matching motion-estimation algorithms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 74–86, Feb. 1996.
[17] G. Gupta and C. Chakrabarti, "Architectures for hierarchical and other block matching algorithms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 477–489, Dec. 1995.
[18] E. Iwata and T. Yamazaki, "An LSI architecture for block-matching motion estimation algorithm considering chrominance signal," *VLSI Signal Processing VIII*, pp. 421–430, 1995.
[19] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, pp. 148–157, Apr. 1993.
[20] M.-C. Lu and C.-Y. Lee, "Semi-systolic array based motion estimation processor design," *IEEE Proc. 1997 Int. Symp. Circuits and Systems*, pp. 3299–3302, 1995.
[21] V.-G. Moshnyaga and K. Tamaru, "A memory efficient array architecture for full-search block matching algorithm," *1997 Proc. IEEE Int. Symp. Circuits and Systems*, pp. 4109–4112, 1997.
[22] S.-H. Nam and M.-K. Lee, "Flexible VLSI architecture of motion estimator for video image compression," *IEEE Trans. Circuits Syst. II*, vol. 43, pp. 467–470, June 1996.
[23] B. Natarajan *et al.*, "Low-complexity algorithm and architecture for block-based motion estimation via one-bit transforms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 3244–3247, 1995.
[24] P. A. Ruetz *et al.*, "A high-performance full-motion video compression chip set," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, pp. 111–122, June 1992.
[25] C. Sanz *et al.*, "VLSI architecture for motion estimation using the block-matching algorithm," in *Proc. Eur. Design and Test Conf.*, 1996, pp. 310–314.
[26] C.-L. Wang *et al.*, "A high-throughput, flexible VLSI architecture for motion estimation," *1995 Proc. IEEE Int. Symp. Circuits and Systems*, pp. 3289–3295, 1995.
[27] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1988.
[28] Y.-K. Chen and S. Y. Kung, "A systolic design methodology with application to full-search block-matching architectures," *J. VLSI Signal Processing*, pp. 51–77, 1998.
[29] J.-C. Tuan and C.-W. Jen, "An architecture of full-search block matching for minimum memory bandwidth requirement," in *Proc. IEEE 8th Great Lake Symp. VLSI*, 1998, pp. 152–156.
[30] Y.-K. Lai and L.-G. Chen, "A flexible data-interlacing architecture for full-search block-matching algorithm," in *IEEE Int. Conf. Application-Specific Systems, Architectures and Processors*, 1997, pp. 96–104.
[31] H.-D. Lin *et al.*, "A 14-Gops programmable motion estimator for H.26X video coding," *IEEE J. Solid-State Circuits*, vol. 31, pp. 1742–1750, Nov. 1996.
[32] S. B. Pan *et al.*, "VLSI architectures for block matching algorithms using systolic arrays," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 67–73, Feb. 1996.
[33] J. You and S. U. Lee, "High throughput, scalable VLSI architecture for block matching motion estimation," *J. VLSI Signal Processing*, vol. 19, pp. 39–50, 1998.
[34] Next-Generation DRAM Comparison (1999). [Online]. Available: http://www.micron.com

Jen-Chieh Tuan received the B.S. degree in electrical engineering from National Tsing Hua University in 1995 and the M.S. degree in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1997.

He is currently a Logic Design Engineer of iCreate Technologies Corporation, Hsinchu, Taiwan, R.O.C.

Tian-Sheuan Chang (S'93–M'00) received the B.S., M.S., and Ph.D. degrees in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1993, 1995, and 1999, respectively.

He is currently a Principle Engineer at Global Unichip Corporation, Hsinchu, Taiwan, R.O.C. His research interests include VLSI design, digital signal processing, and computer architecture.

Chein-Wei Jen received the B.S. degree from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1970, the M.S. degree from Stanford University, Stanford, CA, in 1977, and the Ph.D. degree from National Chiao Tung University in 1983.

He is currently with the Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University, as a Professor. During 1985–1986, he was with the University of Southern California, Los Angeles, as a Visiting Researcher. His current research interests include VLSI design, digital signal processing, processor architecture, and design automation. He has held six patents and published over 40 journal papers and 90 conference papers in these areas.