A Modular High-Throughput Architecture for Logarithmic Search Block-Matching Motion Estimation

Hangu Yeo, *Student Member, IEEE*, and Yu Hen Hu, *Senior Member, IEEE*

Abstract— **In this paper, a high-throughput modular architecture for a logarithmic search block-matching algorithm is presented. The design efforts are focused on exploiting the search area data dependencies using special data input ordering constraints. The input bandwidth problem has been solved by a random access on-chip memory, and a simple address generation procedure has been described. Furthermore, this architecture can handle a large search range with unequal horizontal and vertical spans using a technique called *pipeline interleaving*. Compared to the existing architectures for the three-step search BMA, this architecture delivers a high throughput rate with fewer input lines, and is linearly scalable.**

Index Terms— **Block-matching algorithm, logarithmic search, motion estimation and compensation, systolic array architecture.**

I. Introduction

VIDEO compression standards such as MPEG [1] have fundamentally impacted the future direction of modern information technology. These standards made it possible to develop low-cost, high-performance, real-time visual communication systems to support emerging applications such as multimedia, digital library, video-on-demand (VoD), and high-definition television (HDTV).

A common approach of video compression is to exploit both spatial (intraframe) as well as temporal (interframe) redundancies. Transform coding is often used to serve the purpose of intraframe coding, and predictive coding with motion estimation is often used for interframe coding. In the context of video coding, motion estimation is concerned less about the movement of a specific object in each frame. Rather, within a predefined search area of the *reference frame*, the goal of motion estimation is to search for a block or a region which best matches, under certain matching criteria, a given block (or a region) in the current frame. The displacement between the coordinate of the block in the current frame and the matched block in the reference frame is called a *motion vector*. Given the block in the reference frame and the motion vector, as well as the difference between these two corresponding blocks, the block in the current frame can be recovered perfectly. Assuming the reference frame is available, then only the difference image between the two matched blocks and the corresponding motion vector needs to be transmitted to fully recover the current frame. This often leads to a significant reduction of the video data to be transmitted.

Motion estimation can be performed with different granuarities such as a pixel, a block, or a (irregular) region.

Among them, the block-based approach is considered the most matured and practically useful. A *block* refers to a small square of pixels (e.g., 8×8, 16×16) in a frame. During motion estimation, a distance measure, defined by the matching criterion, between the target block in the current frame and a candidate block within the search area of the reference frame will be computed. The candidate block with the smallest distance to the target block will be selected as the best matched block, and the displacement between these two blocks will be computed and transmitted as the motion vector.

Motion estimation is a computationally intensive task. The amount of computation is proportional to the number of candidate blocks in the search area and the size of the block. A full-search block-matching algorithm (FBMA) evaluates the distance between the target block and every candidate block within the search region. Hence, it is able to find the best matched block to give the highest peak-signal-to-noise ratio (PSNR) and reconstructed image quality. On the other hand, it also demands an enormous amount of computation. For example, for HDTV applications, it is estimated that 8 billion additions will need to be performed each second in order to sustain real-time compression of the video signals. Numerous special-purpose hardware, many feature systolic-array structures, have been proposed [8]–[13] to alleviate this problem. On the software side, there are also many *fast* block-matching algorithms being proposed which seek to reduce the computation time by searching only a subset of the eligible candidate blocks. These fast block motion estimation algorithms include the two-dimensional logarithm search algorithm [4], the three-step search algorithm [5], the conjugate direction search algorithm [6], and many variations. These algorithms search only a small fraction of available candidate blocks at the cost of moderate performance degradation in terms of PSNR and subjective picture quality. Even for this family of *fast* motion estimation algorithms, a hardware implementation will be beneficial in that a special-purpose motion estimation unit can spare the host processor to handle more complicated, but fewer computation-intensive tasks. However, unlike the FBMA algorithm which is extremely regular and suitable for

Manuscript received July 31, 1995; revised January 28, 1996. This paper was recommended by Associate Editor S. Panchanathan.

H. Yeo is with the IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA.

Y. H. Hu is with the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706 USA.

Publisher Item Identifier S 1051-8215(98)03979-2.

array structure implementation, these fast motion estimation algorithms have a much less regular structure, and hence are more difficult to implement. So far, there are two different architectures proposed for the implementation of the three-step search block-matching motion estimation algorithm.

In Jehng *et al.* [14], they proposed a low-latency and high-throughput *tree architecture* for the FBMA and three-step search BMA. This tree architecture can avoid the long data path length, data skewing, and hazards caused by data dependency. Nevertheless, it requires massive amounts of input ports, i.e., it requires N^2 input ports for the block of size $N \times N$ image pixels. To alleviate the problem of large input pin count, they proposed a *tree-cut technique* which folds a whole tree into a subtree with a reduced number of processing elements and input pin count by sacrificing the throughput rate. In [15], Jong *et al.* suggested a fully pipelined parallel architecture for the three-step search BMA. They map the three-step search BMA onto one-dimensional nine processing elements, and let each PE evaluate each candidate location. For the data reusability, half search area buffers have been proposed to utilize the overlapped search area region between the two neighboring target blocks. Both architectures, [14] and [15], introduced a technique called *memory interleaving* which distributes search area data to N^2 and nine memory modules, respectively, and allows parallel data accesses. Nonetheless, there is still room for further improvement. In particular, the overlap of search regions *within* the search area corresponding to a particular target block is not fully exploited in these architectures, resulting in excessive memory access.

In this paper, we discuss our effort to develop an array structure to implement a logarithmic search block-matching algorithm of which the three-step search algorithm is a special case. A key step in this work is to exploit the patterns of overlapped search area in the reference frame so as to derive an efficient architecture which reduces as much as possible the needs to reload or redistribute the same reference data over and over. As a consequence, our architectures offer more efficient processor utilization while achieving a very high throughput rate. The irregularity of the logarithmic search method is hidden behind the memory access pattern, which is how different parts of the memory (frame buffer) are accessed at different time. Previous implementations of the three-step search method did not address the issue explicitly, nor are their results conclusive. In this paper, we derive a formula explicitly describing the memory access pattern, and proposed an application-specific architecture to implement the memory access process. With these efforts, we estimate that the architecture we propose is capable of handling the progressive-scan HDTV format with a clock rate as low as 50 MHz.

The rest of this paper is organized as follows. The logarithmic search BMA is reviewed in Section II. The reference area overlapping pattern analysis is discussed in Section III. Furthermore, high-throughput architectures with reduced input pin count and memory bandwidth using special data input schemes are discussed in Section IV. Finally, comparisons with other existing architectures and conclusions are given in Section V.

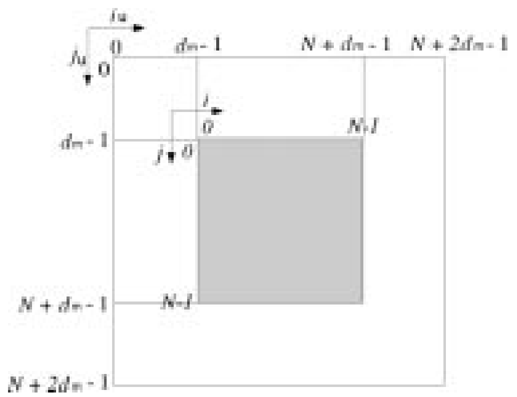


Fig. 1. Pixel coordinate index space.

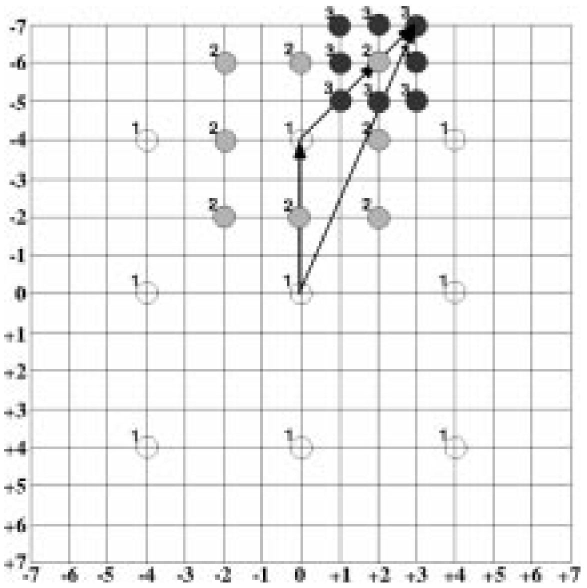


Fig. 2. Example of logarithmic search BMA where $N = 16$ and $d_m = 7$.

II. Review of the Logarithmic Search BMA

By default, the current frame and the reference frame have exactly the same coordinate for each pair of corresponding pixels. Depicted in Fig. 1 is the target block (shaded region) of the current frame overlaid on the search area (the clear region in the background) of the reference frame. Starting from the current frame position, the search region spans d_m pixels in each direction. Thus, each entry of the two dimensional motion vector (MV) has a range of $[-d_m \; d_m]$.

In a full search BMA, all $(2d_m + 1)^2$ positions are to be searched. Each search requires the calculation of the *total absolute difference* (TAD) between the pixels in the target block, denoted by $x(i,j)$, and the candidate block, denoted by $y(i,j)$:

$$\mathrm{TAD}(di, dj) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |x(i,j) - y(i + di, j + dj)| \tag{1}$$

where (di, dj) is the displacement between these two blocks. Thus, each search would require N^2 integer subtractions and additions.

In a logarithmic search BMA, the search is accomplished hierarchically in $\nu = \lceil \log_2 d_m \rceil$ steps.[1] During step s $(0 \le s \le \nu - 1)$, a partial motion vector $[MV1(s), MV2(s)]$ is determined by comparing the TAD's evaluated at exactly *nine* displacement vectors:

$$\mathrm{TAD}(s, m(s), n(s)) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |x(i,j) - y(i + O1(s) + m(s), j + O2(s) + n(s))| \quad (2)$$

where $m(s), n(s) \in \{-d(s), 0, d(s)\}\}$, with the local step size[2] $d(s) = 2^{\nu - s - 1}$, and the shifted origin $(O1(s), O2(s)) = \Sigma_{k=0}^{s-1} (MV1(k), MV2(k))$ for $s \ge 1$, and (0, 0) for $s = 0$. The local displacement $(m(s), n(s))$ which yields the smallest TAD will be chosen as $(MV1(s), MV2(s))$. The net motion vector then can be found as

$$\begin{aligned} MV &= (MV1, MV2) = (O1(\nu), O2(\nu)) \\ &= \sum_{k=0}^{s-1} (MV1(k), MV2(k)). \end{aligned} \quad (3)$$

Let us now consider an example where $N = 16$, and $d_m = 7$. Then $\nu = 3$, $d(0) = 4$, $d(1) = 2$, and $d(2) = 1$. This corresponds to the case of *three-step search BMA* method [5]. This is illustrated in Fig. 2 where the 225 $(= (2 \times 7 + 1)^2)$ candidate motion vectors correspond to the 225 grids. In the first step, $(s = 0)$, the TAD's of those nine clear circles labeled with 1 are evaluated. After comparison, it is decided $(MV1(0), MV2(0)) = (0, -d(0)) = (0, -4) = (O1(1), O2(1))$. In the second step, TAD's on the eight shaded circles as well as $\mathrm{TAD}(0, 0, -4)$ are compared, and the minimum is selected. This yields $(MV1(1), MV2(1)) = (-d(1), -d(1)) = (2, -2)$ in this example. Consequently, $(O1(2), O2(2)) = (0, -4) + (2, -2) = (2, -6)$. Finally, in the third step, $(MV1(2), MV2(2)) = (d(2), d(2)) = (1, -1)$, and $MV = (2, -6) + (1, -1) = (3, -7) = (O1(3), O2(3))$.

Note that the total number of distortion measure evaluations in a logarithmic search BMA method is $8\nu + 1$—orders of magnitude less than that of the full search BMA method. With the above example, only 25 out of the possible 225 (a ratio of nine to one reduction) distortion measures need to be evaluated for each target block in the current frame. On the other hand, since not all candidate displacement vectors are evaluated, the logarithmic BMA method often yields a suboptimal motion vector whose corresponding TAD may not be the global minimum among all $(2d_m + 1)^2$ candidate displacement vectors. Some proposals to modify the three-step search method to alleviate this problem can be found in [16]–[17].

The logarithmic search BMA corresponding to a single target block in the reference frame can be written as the five-level nested *Do* loop as depicted in Fig. 3. The outer most loop with index s performs the ν search steps. Loop indexes m and n refer to the nine displacement vectors to be searched during each step, and i and j loops compute the TAD for a given displacement vector.

[1] $\lceil x \rceil$ is the ceiling function.

[2] Note that $d_m = \Sigma_{s=0}^{\nu-1} d(s)$.

```
$MV1 = MV2 = 0,\ MV1(0) = MV2(0) = 0;$
Do $s = 0$ to $\nu - 1$
    $D_{min} = \infty$
    $d(s) = 2^{\nu - s - 1}$
    Do $m = -1$ to 1;                /* $m = -1, 0, 1$ */
        Do $n = -1$ to 1;                /* $n = -1, 0, 1$ */
            $TAD(m,n) = 0$
            Do $i = 0$ to $N - 1$
                Do $j = 0$ to $N - 1$
                    $TAD(m,n) = TAD(m,n)$
                              $+|x(i,j) - y(i + m \cdot d(s) + MV1, j + n \cdot d(s) + MV2)|$
                End Do $j$
            End Do $i$
        End Do $n$
    End Do $m$
    Do $m = -1$ to 1
        Do $n = -1$ to 1
            If $D_{min} > TAD(m,n)$
                $D_{min} = TAD(m,n)$
                $MV1(s+1) = m \cdot d(s)$
                $MV2(s+1) = n \cdot d(s)$
            End If
        End Do $n$
    End Do $m$
    $MV1 = MV1 + MV1(s+1)$
    $MV2 = MV2 + MV2(s+1)$
End Do $s$
```

Fig. 3. Logarithmic search BMA.

Given a nested *Do* loop, it is possible to *map* its corresponding *dependence graph* onto an array structure with processing nodes occupying a lower dimension index space [18]. However, due to the dependency of $d(s)$ on s, the corresponding five-dimensional dependence graph is *not* regular. Thus, an existing systolic design procedure cannot be applied directly. On the other hand, the inner four-level nested *Do* loop with s being fixed indeed forms a regular dependence graph, and systolic mapping procedures can be applied.

III. Reference Area Overlapping Pattern Analysis

Given a fixed $d(s)$, an $(N + 2d(s)) \times (N + 2d(s))$ area in the reference frame will be searched to compute the nine $\mathrm{TAD}(m,n)$; $m, n = -1, 0, 1$. This is illustrated in Fig. 4(a). For example, the search area for $\mathrm{TAD}(0,1)$ corresponds to the shaded area in this region. Clearly, in the nine possible search directions, their search areas have significant overlap. Under the assumption that N is an integral multiple of $d(s)$, we may partition this overall search area into $(N/d(s)+2) \times (N/d(s)+2)$ segments, each of size $d(s)$ pixels $\times$ $d(s)$ pixels. Some of these segments will be used only once by one particular search direction (m,n), and others will be used many more times. In Fig. 4(b), we list the number of search directions for which the segment will be accessed for the case of $N = 16$, $d(s) = 4$. This figure suggested an interesting memory loading strategy: if each of these segments is loaded simultaneously pixel by pixel, then the entire search area will need $(N/d(s) + 2)^2$ I/O channels (8 bits wide each), and the loading of data into the array can be accomplished in $d(s)^2$ clock cycles. Once the

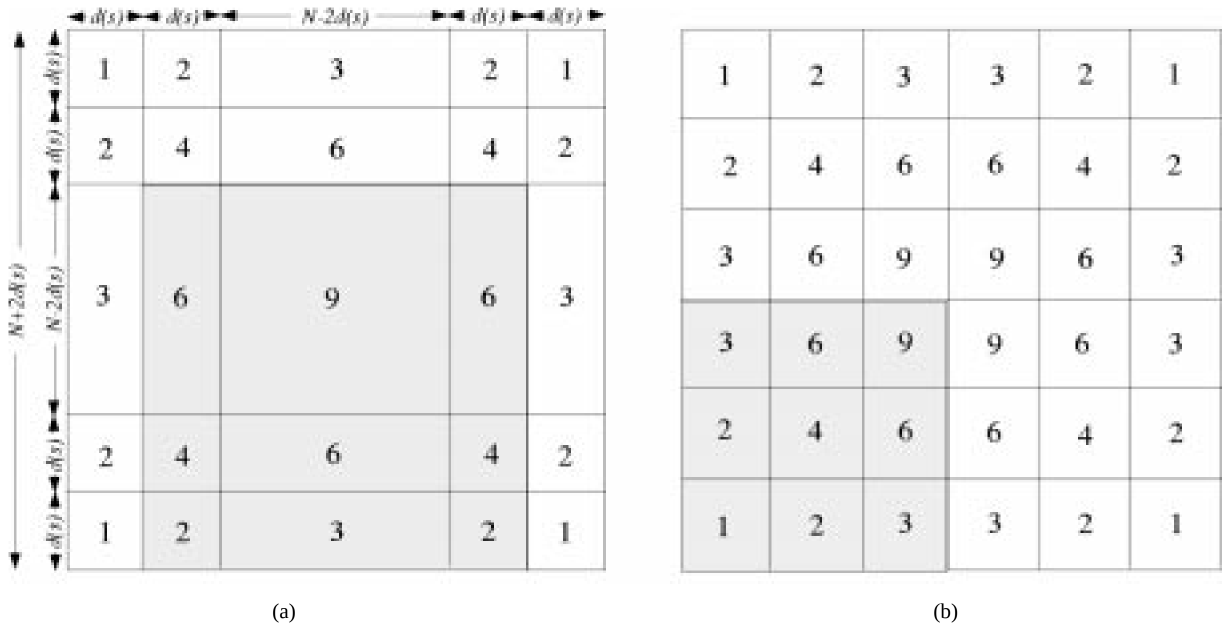


Fig. 4. (a) Search area for a target block at search step s. (b) Example when $N = 16$ and $d(s) = 4$.

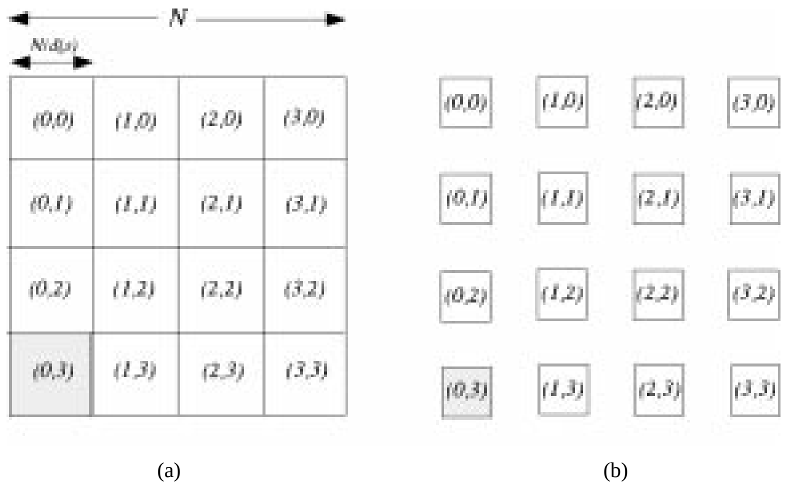


Fig. 5. (a) Target block is subdivided into subblocks. (b) Each PE assigned to each subblock.

pixel is loaded into the array, they will be routed to the proper processing element for computation. As such, each pixel will be loaded only *once* regardless of whether it is to be used once or nine times during the computation of the nine TAD's. This is the motivation behind the architecture proposed below.

Before we present the formal derivation, let us consider a heuristic architecture. Assume that the target area is also decomposed into $N/d(s) \times N/d(s)$ subblocks (in this example, 16/4 × 16/4 = 16 subblocks), and assign one processing element (PE) to compute the 16 ADA (absolute difference and accumulation) operations within a subblock as depicted in Fig. 5(a) and (b). Then each PE will need to access a 3 × 3 subblock array in the reference frame. For example, the PE (0, 3) will process pixels in reference subblocks shaded in Fig. 4(b). By analyzing this pattern, an interconnection pattern within the array structure can be devised for a given search step $d(s)$.

Obviously, this is only a basic idea. The actual implementation will be more complicated. First, recall that $d(s)$ changes its value for different search steps s. But reconfiguring the array during the program execution may not be the best idea. Also, when $d(s)$ becomes small (i.e., $d(s) = 1$), assigning $d(s)^2$ pixels to a single PE will no longer be economical. In this case, a larger subblock size will be used, and of course, the internal control for each PE will be somewhat complicated as a result.

IV. Algorithm Transformation and Architecture

A. Algorithm Transformation

Based on Fig. 4(a) above, we can summarize the overlap between search areas in the following lemma.

Lemma 1: Let c and c' be integers satisfying $1 \le c' \le c < 3$. If $d(s) \le N$, then the data dependency of $y(m, n, i, j)$ can be expressed as follows.

Case I: For $0 \le i < N - c \cdot d(s)$ and $m \ne 1$,

$$y(m, n, i, j) = y(m - c', n, i + c' \cdot d(s), j). \tag{4}$$

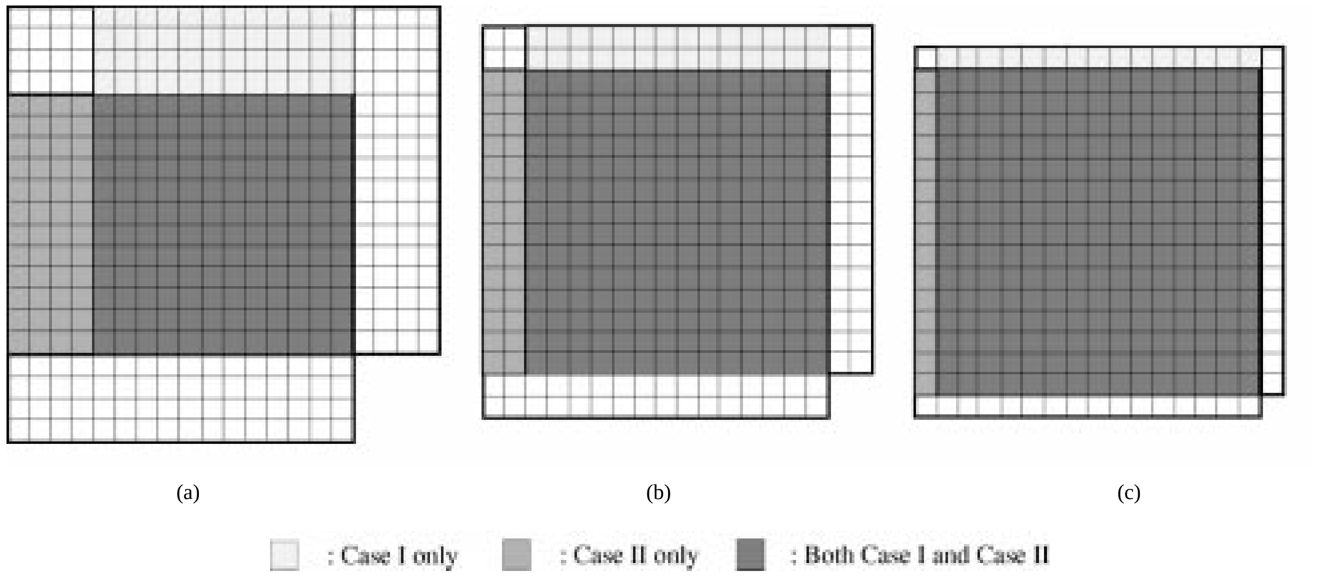


Fig. 6. Three search areas corresponding to (m, n), $(m + 1, n)$, and $(m, n + 1)$. (a) Step 0. (b) Step 1. (c) Step 2.

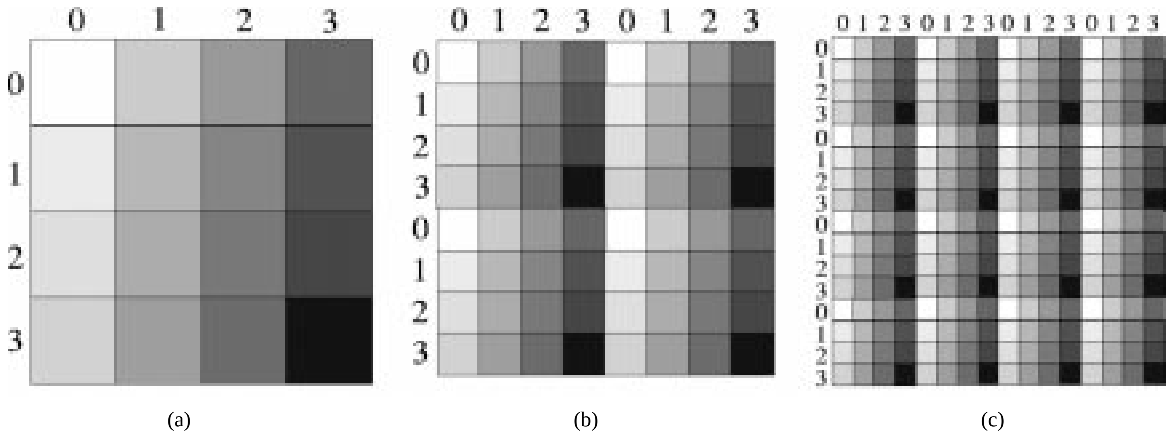
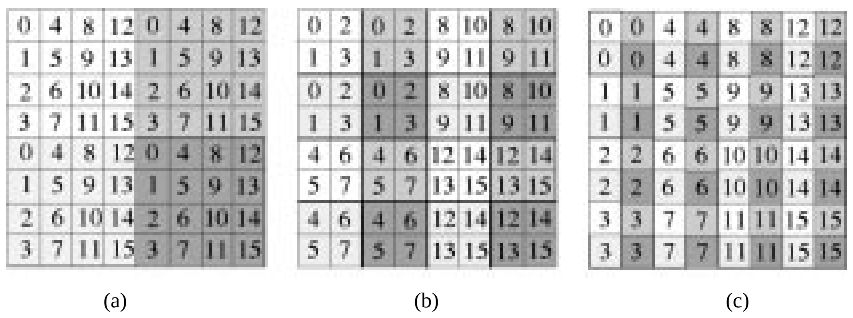


Fig. 7. Image pixel segmentation when $N = 16$. (a) Step 0. (b) Step 1. (c) Step 2.



(a)

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

(b)

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

(c)

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

Fig. 8. Example: Type I input ordering scheme. (a) Step 0. (b) Step 1. (c) Step 2.

Case II: For $0 \leq j < N - c \cdot d(s)$ and $n \neq 1$,

$$y(m, n, i, j) = y(m, n - c', i, j + c' \cdot d(s)). \tag{5}$$

Proof: The proof is given in the Appendix. ■

To illustrate, consider the case of $N = 16$, $d_m = 7$, and $c = 1$. Depicted in Fig. 6 are the three search areas corresponding to (m,n), $(m + 1, n)$, and $(m, n + 1)$. The overlapped area is the shadowed region. It can be verified that the conditions stated in (4)–(5) in Lemma 1 are correct.

As mentioned in Section III, the inner four-level nested *Do* loop with the s being fixed indeed forms a regular dependence graph, and systolic mapping procedures can be applied. Assuming that the 3 × 3 array of search points is to be searched column by column, the pair of indexes m and n in the original formulation in Fig. 3 can be combined into a composite index l using the following transformation formula:

$$\begin{cases} l = 3(m+1) + (n+1), & 0 \leq l \leq 8 \\ m = \left(\left\lfloor \dfrac{l(s)}{3} \right\rfloor - 1 \right) = \{-1, 0, 1\} \\ n = (l \bmod 3 - 1) = \{-1, 0, 1\}. \end{cases} \tag{6}$$

```
$D_{min} = \infty$, $d(s) = 2^{\nu-s-1}$;
Do $l = 0$ to 8
    $TAD(l) = 0$
    Do $j_1 = 0$ to $d(0)^2 - 1$
        Do $i_1 = 0$ to 15
            $TAD(l) = TAD(l) + |x_s(i_1, j_1) - y_s(l, i_1, j_1)|$
        End Do $i_1$
    End Do $j_1$
End Do $l$
Do $l = 0$ to 8
    If $D_{min} > TAD(l)$
        $D_{min} = TAD(l)$
        $MV1 = (\lfloor \frac{l}{3} \rfloor - 1)d(s)$
        $MV2 = (l \bmod 3 - 1)d(s)$
    End If
End Do $l$
```

Fig. 9. Three-level nested *Do* loops after algorithm transformation.

Moreover, the pair of indexes i and j can be transformed into i_1 and j_1 as summarized in the following lemma, and the search area data dependencies in Lemma 1 can be exploited.

Lemma 2: For each step s, the indexes i and j $(0 \leq i, j < N)$ introduced in Section III are transformed into the indexes i_1 $(0 \leq i_1 < (N/d(0))^2)$ and j_1 $(0 \leq j_1 < d(0)^2)$. The i_1 subdivides a block of size $N \times N$ into $(N/d(0))^2$ subregions of $d(0)^2$ image pixels which are indexed by j_1 as depicted in Fig. 7:

$$i_1 = 4\left(\left\lfloor \frac{i}{d(s)} \right\rfloor \bmod 4\right) + \left\lfloor \frac{j}{d(s)} \right\rfloor \bmod 4 \tag{7}$$

$$j_1 = d(s)^2 \cdot \left(\frac{N}{4d(s)} \left\lfloor \frac{i}{4d(s)} \right\rfloor + \left\lfloor \frac{j}{4d(s)} \right\rfloor\right) + d(s) \cdot (i \bmod d(s)) + j \bmod d(s) \tag{8}$$

$$i = 4d(s)\left\lfloor \frac{j_1}{4d(s)} \right\rfloor + d(s)\left\lfloor \frac{i_1}{4} \right\rfloor + \left\lfloor \frac{j_1}{d(s)} \right\rfloor \bmod d(s) \tag{9}$$

$$j = 4d(s) \cdot \left(\left\lfloor \frac{j_1}{d(s)^2} \right\rfloor \bmod \frac{N}{4d(s)}\right) + d(s) \cdot (i_1 \bmod 4) + j_1 \bmod d(s). \tag{10}$$

Proof: The proof is given in the Appendix. ■

This index transformation implies that when pixels of an image block are read into the array structure, they are loaded sequentially according to the following data input ordering scheme. The target block is decomposed into $N/d(0)$ by $N/d(0)$ subregions[3] according to the search step as depicted in Fig. 7, and the individual pixels within each subregion are loaded column by column. Depicted in Fig. 8 is an example when $N = 8$ and $d_m = 7$. The number within each subregion depicts j_1, which is the input order of image pixels within each subregion.

With the above data input ordering scheme, we proceed to replace $x(i, j)$ in Fig. 3 by $x_s(i_i, j_1)$, and denote the serialized reference block pixel $y(i+m \cdot d(s)+MV1, j+n \cdot d(s)+MV2)$ by $y_s(i_1 + \Delta i_1, j_1 + \Delta j_1)$, and $\mathrm{TAD}(m, n)$ by $\mathrm{TAD}(l)$. This yields the three-level nested *Do* loop formulation as depicted in Fig. 9, where

[3] $N/d(0) = 4$.

```
$D_{min}(0, 17, d(0)^2) = \infty$, $d(s) = 2^{\nu-s-1}$;
Do $l = 0$ to 8
    $c = \lfloor \frac{l}{3} \rfloor$
    Do $j_1 = 0$ to $d(0)^2 - 1$
        $TAD(l, 0, j_1) = 0$
        Do $i_1 = 0$ to 15
            $x_s(l+1, i_1, j_1) = x_s(l, i_1, j_1)$
            If $0 \leq i_1 < 4(4-c)$ and $3c \leq l < 3(c+1)$
                $y_s(l, i_1, j_1) = y_s(l-3c, i_1+4c, j_1)$
            If $0 \leq i_1 \bmod 4 < 4-c$ and $l(s) \bmod 3 \neq 0$
                $y_s(l, i_1, j_1) = y_s(l-c, i_1+c, j_1)$
            End If
            $TAD(l, i_1+1, j_1) = TAD(l, i_1, j_1) + |x_s(l, i_1, j_1) - y_s(l, i_1, j_1)|$
        End Do $i_1$
        $TAD(l, 17, j_1+1) = TAD(l, 17, j_1) + TAD(l, 16, j_1+1)$
    End Do $j_1$
End Do $l$
Do $l = 0$ to 8
    If $D_{min}(l, 17, d(0)^2) > TAD(l, 17, d(0)^2)$
        $D_{min}(l+1, 17, d(0)^2) = TAD(l, 17, d(0)^2)$
        $MV1(l+1, 17, d(0)^2) = (\lfloor \frac{l}{3} \rfloor - 1)d(s)$
        $MV2(l+1, 17, d(0)^2) = (l \bmod 3 - 1)d(s)$
    End If
End Do $l$
```

Fig. 10. Three-level nested *Do* loops after algorithm transformation (partially localized code).

$$x_s(i_1, j_1) = x\left(4d(s) \cdot \left\lfloor \frac{j_1}{4d(s)} \right\rfloor + d(s) \cdot \left\lfloor \frac{i_1}{4} \right\rfloor + \left\lfloor \frac{j_1}{d(s)} \right\rfloor \bmod d(s) \quad 4d(s) \cdot \left(\left\lfloor \frac{j_1}{d(s)^2} \right\rfloor \bmod \frac{N}{4d(s)}\right) + d(s) \cdot (i_1 \bmod 4) + j_1 \bmod d(s)\right) \tag{11}$$

$$\begin{aligned} y_s(l, i_1, j_1) &= y(i + m \cdot d(s) + MV1, j + n \cdot d(s) + MV2) \\ &= y\left(\left(\left\lfloor \frac{i_1}{4} \right\rfloor + 4\left\lfloor \frac{j_1}{4d(s)} \right\rfloor + \left\lfloor \frac{l}{3} \right\rfloor - 1\right) \cdot d(s) + \left\lfloor \frac{j_1}{d(s)} \right\rfloor \bmod d(s) + MV1, \left(i_1 \bmod 4 + 4\left(\left\lfloor \frac{j_1}{d(s)^2} \right\rfloor \bmod \frac{N}{4d(s)}\right) + l \bmod 3 - 1\right) \cdot d(s) + j_1 \bmod d(s) + MV2\right). \end{aligned} \tag{12}$$

We use $y_s(l, i_1, j_1)$ to emphasize that this transformed variable depends on all three indexes: l, i_1, and j_1.

B. Systolic Mapping

In order to apply systolic mapping, a computing algorithm must satisfy two conditions, namely, *single assignment form*

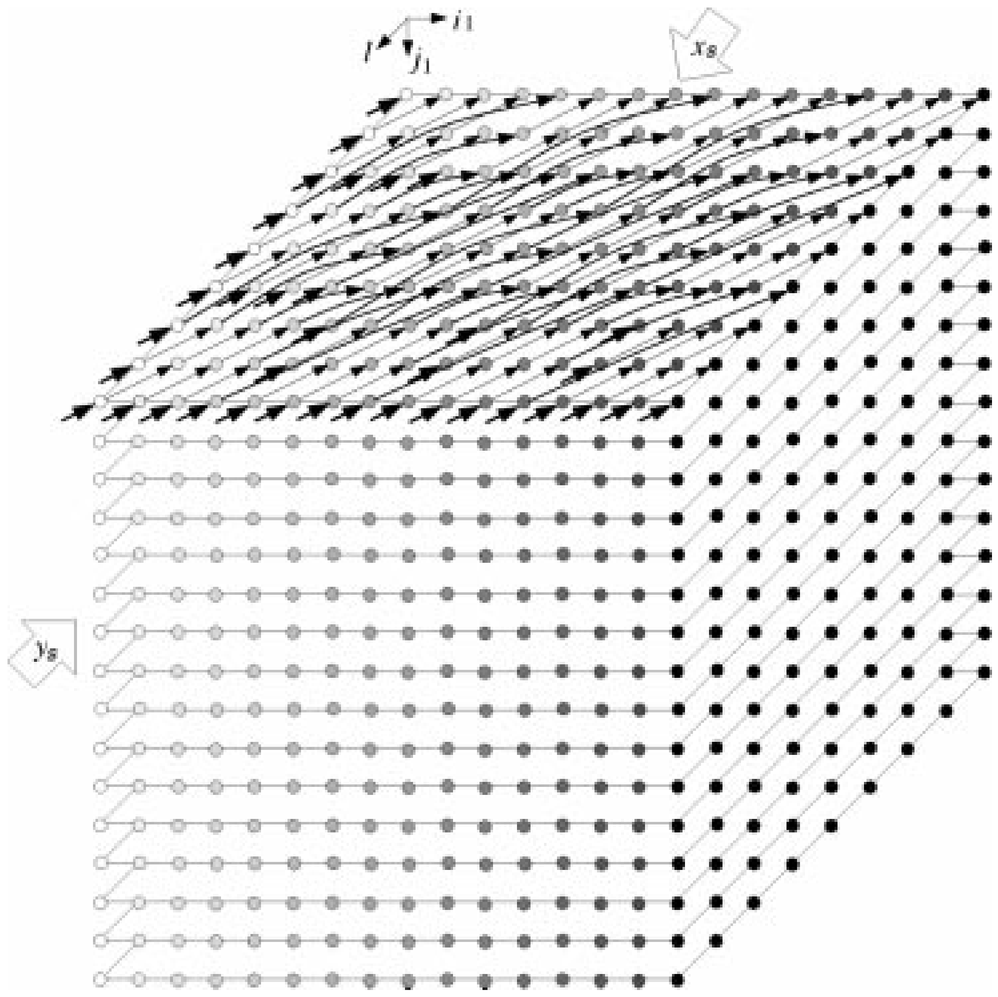


Fig. 11. Partially localized three-dimensional DG (Type I).

and *locally recursive dependency*. By single assignment form, each variable can be assigned to a single value during the execution of the entire algorithm. By locally recursive dependency, the data dependency cannot be a function of the loop iteration bounds. To satisfy these two constraints, we change the variables $x_s(i_1, j_1)$ and $\mathrm{TAD}(l)$, respectively, into a three-dimensional variables $x_s(l, i_1, j_1)$, and $\mathrm{TAD}(l, i_1, j_1)$, and impose the following data transmission rules:

$$x_s(l, i_1, j_1) = x_s(l-1, i_1, j_1) \tag{13}$$

$$\begin{aligned}\mathrm{TAD}(l, i_1+1, j_1) &= \mathrm{TAD}(l, i_1, j_1) \\ &\quad + |x_s(l, i_1, j_1) - y_s(l, i_1, j_1)| \end{aligned}\tag{14}$$

$$\begin{aligned}\mathrm{TAD}(l, 17, j_1+1) &= \mathrm{TAD}(l, 17+1, j_1) \\ &\quad + \mathrm{TAD}(l, 16, j_1+1).\end{aligned}$$

Compared to x_s and TAD, the data propagation pattern of y_s is much more complicated. We note that the same pixel in the reference frame $y(i, j)$ may be used several times during the logarithmic search as summarized in Lemma 1. If we can carefully design the input ordering of the pixel stream of the reference frame, much input/output bandwidth may be saved, resulting in more efficient and less costly implementation. This can be accomplished by propagating the reference frame pixels along the proper direction so that the data can be made available without requesting external input/output operations. After careful analysis of the overlaps of the search region for different values of l, we summarize the result in Corollary 1.

Corollary 1: Let c and c' be integers satisfying $1 \le c' \le c < \min(3, N)$. Then the data dependency of $y_s(l, i_1, j_1)$ can be expressed as follows:

Case I: For $0 \le i_1 < 4(4-c)$ and $3c \le l < 3(c+1)$,

$$y_s(l, i_1, j_1) = y_s(l - 3c', i_1 + 4c', j_1). \tag{15}$$

Case II: For $0 \le i_1 \bmod 4 < 4 - c$ and $l \bmod 3 \ne 0$,

$$y_s(l, i_1, j_1) = y_s(l - c', i_1 + c', j_1) \tag{16}$$

Proof: The proof is given in the Appendix. ■

Incorporating (13), (14), and Corollary 1, the logarithmic search BMA can be transformed into a localized, regular iterative formulation as depicted in Fig. 10.

In the above formulation, there are five dependence vectors, and these dependence vectors can be represented in a matrix form $(\mathcal{D}_1)$ with each column representing each dependence vector:

$$\mathcal{D}_1 = \begin{bmatrix} 1 & 0 & 0 & -3 & -1 \\ 0 & 1 & 0 & 4 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{vmatrix} l \\ i_1 \\ j_1 \end{vmatrix}. \tag{17}$$

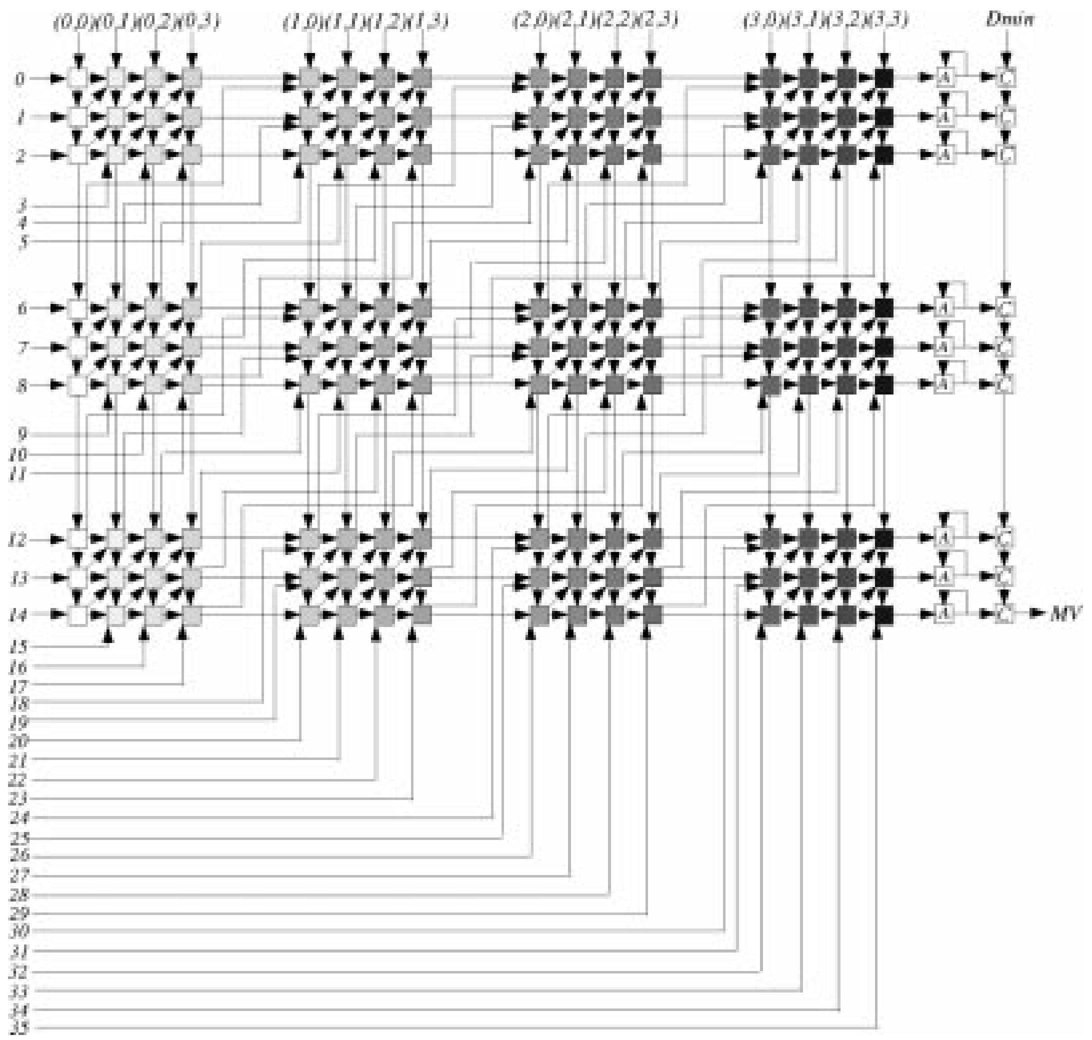


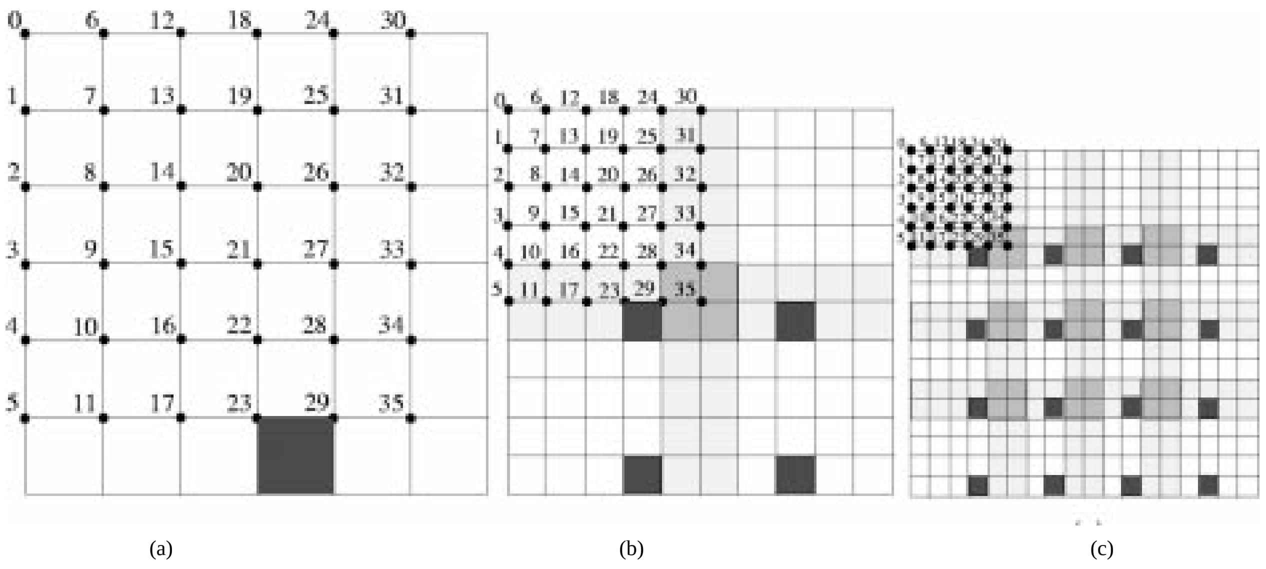
Fig. 12. Type I architecture.

The first column denotes the dependence vector for variables MV, $D_{\min}$, and x_s while the second and third columns denote dependence vectors for a variable TAD. The next two columns denote the dependence vectors of y_s indicated by (15) and (16) and depicted in Fig. 11. By projecting the three-dimensional dependence graph (Fig. 11) along a projection vector $\vec{d}^T = [0\ \ 0\ \ 1]$ with the so-called *default schedule* direction $\vec{s}^T = [0\ \ 0\ \ 1]$, we obtain a two-dimensional array structure, as depicted in Fig. 12, consisting of 9 $\times$ 16 processing elements, each responsible for a subregion depicted in Fig. 7 for a particular search direction (the l index) during the s step search.

The two-dimensional array structure requires 16 input ports for loading the 16 subregions within the target area depicted in Fig. 7, and each of these subregions is loaded simultaneously, pixel by pixel. Each pixel loaded will be broadcasted to the nine processing elements. The search area data are loaded through the 36 input ports simultaneously, and they will be routed to the proper processing element for computation. The search area of size $(N + 2d(s)) \times (N + 2d(s))$ in the reference frame is partitioned into $(N/d(s)+2)\times(N/d(s)+2)$ segments, each of size $d(s)$ pixels $\times$ $d(s)$ pixels as depicted in Fig. 13. For each step s, the $d(0)^2/d(s)^2$ number of segments composes a subregion of $d(0)^2$ image pixels according to the target area subregion pattern depicted in Fig. 7, and each of these subregions is loaded simultaneously pixel by pixel through each input port. At search step 0, since $N/d(0) = 4$, the search area is composed of 6 $\times$ 6 subregions, each of size $d(0)$ pixels $\times$ $d(0)$ pixels. As such, each pixel within the search area for search step 0 will be loaded only once regardless of whether it is to be used once or nine times. However, at search steps $s > 0$, the subregions overlap since the search area size $N + 2d(s)$ is smaller than $N + 2d(0) = 6d(0)$. Fig. 13 depicts the search area for each search step when $N = 16$ and $d_m = 7$. The black dots indicate the base of each subregion, and the area shaded with black depicts the subregion which will be loaded through input port 23 of the two-dimensional array depicted in Fig. 12. As such, the data within the shaded area will be loaded more than once. This brings the total number of pixels to be loaded to the array, for each target block in the current frame, to $36\cdot\nu\cdot d(0)^2 = (9/4)\cdot\nu\cdot N^2$. Compared to the number $9\cdot\nu\cdot N^2$, this novel approach can alleviate the I/O bandwidth problem.

C. Architectures with Reduced Input Pin Count

In this section, we propose two different data input ordering schemes (Types II and III), and the corresponding architectures



□ : Search area data accessed once.
▨ : Search area data accessed twice.
▩ : Search area data accessed four times.
■ : Sixteen search area data loaded through input pin number 23.

Fig. 13. Search area for each search step when $N = 16$ and $d_m = 7$. (a) Search step 0 with $d(0) = 4$. (b) Search step 1 with $d(1) = 2$. (c) Search step 2 with $d(2) = 1$.

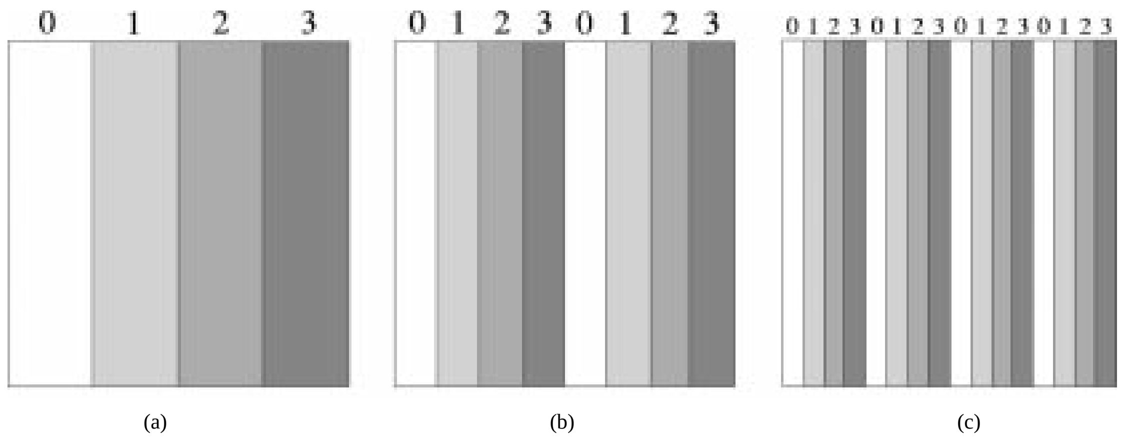


Fig. 14. Image pixel segmentation when $N = 16$. (a) Step 0. (b) Step 1. (c) Step 2.

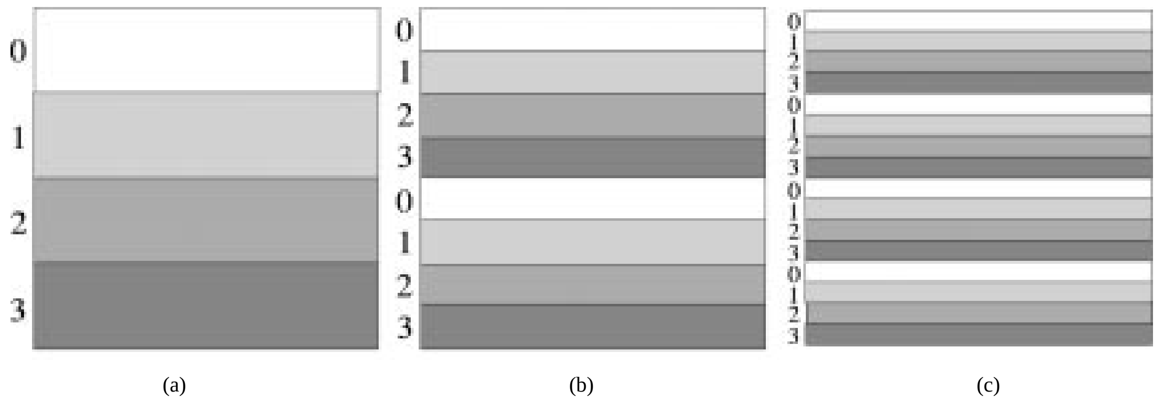
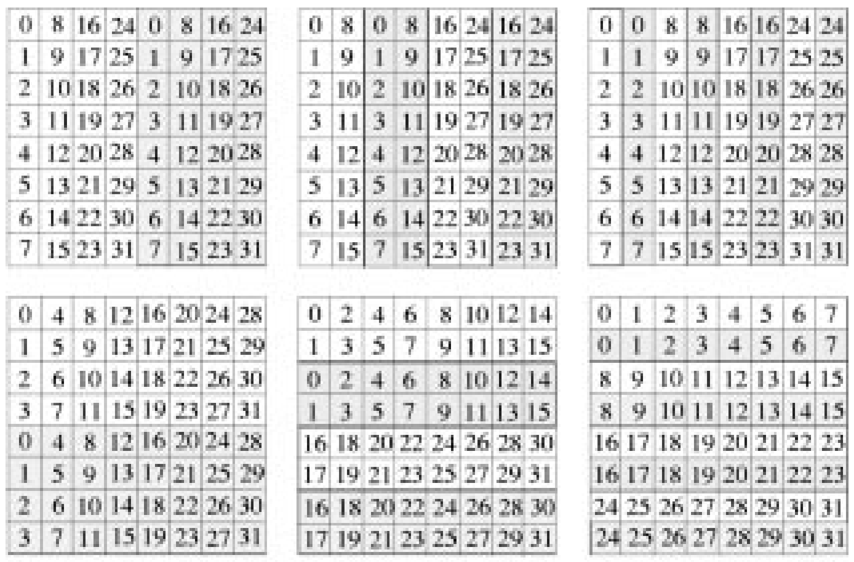


Fig. 15. Image pixel segmentation when $N = 16$. (a) Step 0. (b) Step 1. (c) Step 2.



0	8	16	24	0	8	16	24
1	9	17	25	1	9	17	25
2	10	18	26	2	10	18	26
3	11	19	27	3	11	19	27
4	12	20	28	4	12	20	28
5	13	21	29	5	13	21	29
6	14	22	30	6	14	22	30
7	15	23	31	7	15	23	31

0	8	0	8	16	24	16	24
1	9	1	9	17	25	17	25
2	10	2	10	18	26	18	26
3	11	3	11	19	27	19	27
4	12	4	12	20	28	20	28
5	13	5	13	21	29	21	29
6	14	6	14	22	30	22	30
7	15	7	15	23	31	23	31

0	0	8	8	16	16	24	24
1	1	9	9	17	17	25	25
2	2	10	10	18	18	26	26
3	3	11	11	19	19	27	27
4	4	12	12	20	20	28	28
5	5	13	13	21	21	29	29
6	6	14	14	22	22	30	30
7	7	15	15	23	23	31	31

0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31
0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

0	2	4	6	8	10	12	14
1	3	5	7	9	11	13	15
0	2	4	6	8	10	12	14
1	3	5	7	9	11	13	15
16	18	20	22	24	26	28	30
17	19	21	23	25	27	29	31
16	18	20	22	24	26	28	30
17	19	21	23	25	27	29	31

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
24	25	26	27	28	29	30	31

(a) (b) (c)

Fig. 16. Example: Types II and III input ordering schemes. (a) Step 0. (b) Step 1. (c) Step 2.

to lessen the burden of large input pin count of Type I architecture sacrificing the throughput rate. The pair of indexes i and j in the original algorithm depicted in Fig. 3 will be transformed into i_2 (i_3) and j_2 (j_3), respectively as summarized in Lemmas 3 and 4.

Lemma 3: For each step s, the indexes i and j $(0 \leq i, j < N)$ introduced in Section III are transformed into two indexes i_2 $(0 \leq i_2 < N/d(0))$ and j_2 $(0 \leq j_2(s,k) < N \cdot d(0))$. The i_2 subdivides a block of size $N \times N$ into $N/d(0)$ subregions of $N \cdot d(0)$ image pixels which are indexed by j_2 as depicted in Fig. 14.

$$i_2 = \left\lfloor \frac{i}{d(s)} \right\rfloor \bmod 4 \tag{18}$$

$$j_2 = N \cdot d(s) \cdot \left\lfloor \frac{i}{4d(s)} \right\rfloor + N \cdot (i \bmod d(s)) + j \tag{19}$$

$$i = 4d(s) \cdot \left\lfloor \frac{j_2}{d(s)N} \right\rfloor + i_2 d(s) + \left\lfloor \frac{j_2}{N} \right\rfloor \bmod d(s) \tag{20}$$

$$j = j_2 \bmod N. \tag{21}$$

Proof: The proof is given in the Appendix. ■

Lemma 4: For each step s, the indexes i and j $(0 \leq i, j < N)$ introduced in Section III are transformed into two indexes i_3 $(0 \leq i_3 < N/d(0))$ and j_3 $(0 \leq j_3 < N \cdot d(0))$. The i_3 subdivides a block of size $N \times N$ into the $N/d(0)$ subregions of $N \cdot d(0)$ image pixels which are indexed by j_3 as depicted in Fig. 15.

$$i_3 = \left\lfloor \frac{j}{d(s)} \right\rfloor \bmod 4 \tag{22}$$

$$j_3 = N \cdot d(s) \cdot \left\lfloor \frac{j}{4d(s)} \right\rfloor + i \cdot d(s) + j \bmod d(s) \tag{23}$$

$$i = \left\lfloor \frac{j_3}{d(s)} \right\rfloor \bmod N \tag{24}$$

$$j = 4d(s) \cdot \left\lfloor \frac{j_3}{d(s)N} \right\rfloor + i_3 \cdot d(s) + j_3 \bmod d(s). \tag{25}$$

Proof: The proof is given in the Appendix. ■

The target block is decomposed into $N/d(0)$ subregions according to the search step as depicted in Figs. 14 and 15. Again, the individual pixels within each subregion are loaded column by column. Each index transformation procedure exploits the search area data dependencies (4) and (5) in Lemma 1, respectively. The two types of data input ordering schemes are depicted in Fig. 16 where $N = 8$ and $d_m = 7$ have been considered. The number within each block depicts j_2 (j_3), which is the input order of image pixels within each subregion.

After careful analysis of the overlaps of the search region for different values of l, the data propagation pattern of y_s is summarized in the following corollary.

Corollary 2: Let c and c' be integers satisfying $1 \leq c' \leq c < \min(3, N)$. Then the data dependency of $y_s(l, i_2, j_2)$ can be expressed as follows.

For $0 \leq i_2 < 4 - c$ and $3c \leq l < 3(c+1)$,

$$y_s(l, i_2, j_2) = y_s(l - 3c', i_2 + c', j_2). \tag{26}$$

Proof: The proof is given in the Appendix. ■

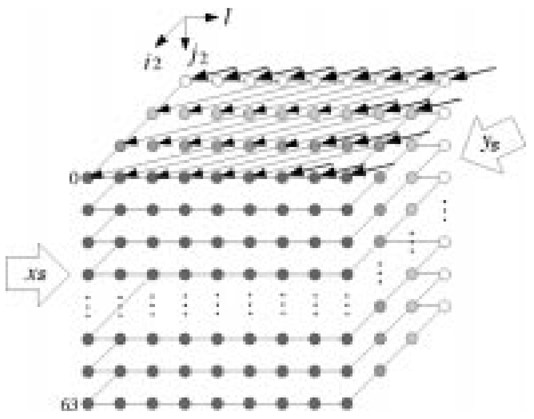
Corollary 3: Let c and c' be integers satisfying $1 \leq c' \leq c < \min(3, N)$. Then the data dependency of $y_s(l, i_3, j_3)$ can be expressed as follows.

For $l \bmod 3 \neq 0$ and $0 \leq i_3 < 4 - c$,

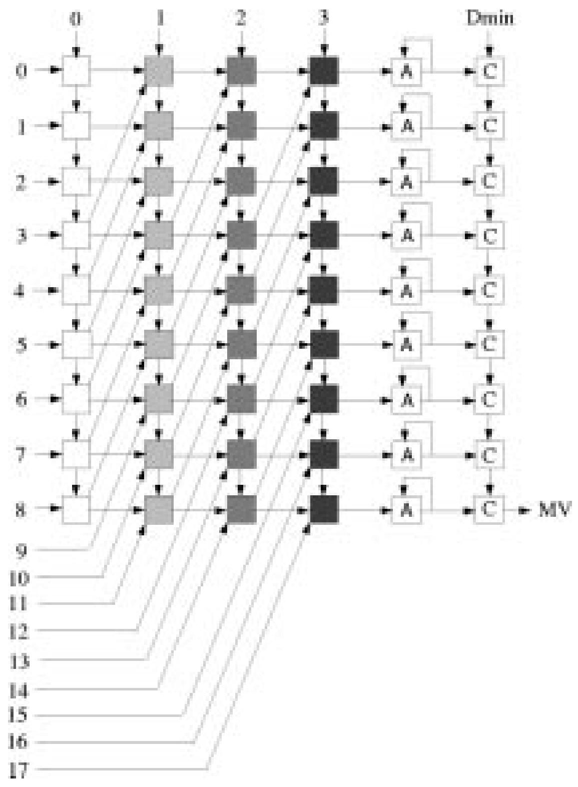
$$y_s(l, i_3, j_3) = y_s(l - c', i_3 + c', j_3). \tag{27}$$

Proof: The proof is given in the Appendix. ■

For each formulation, there are four dependency vectors which can be represented in matrix forms $\mathcal{D}_2$ and $\mathcal{D}_3$ where the first column denotes the dependency vector for variables MV, $D_{\min}$, and x_s. The next two columns denote the dependency vectors for TAD, and the last column denotes the dependency

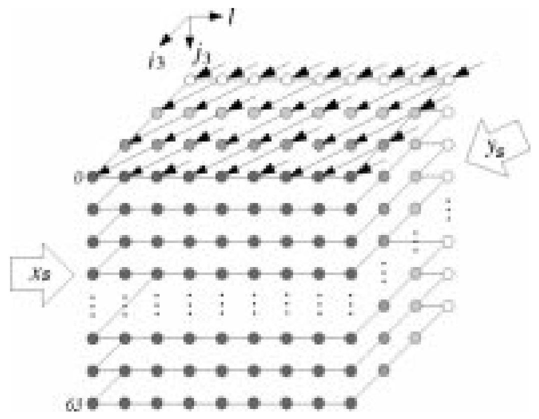


(a)

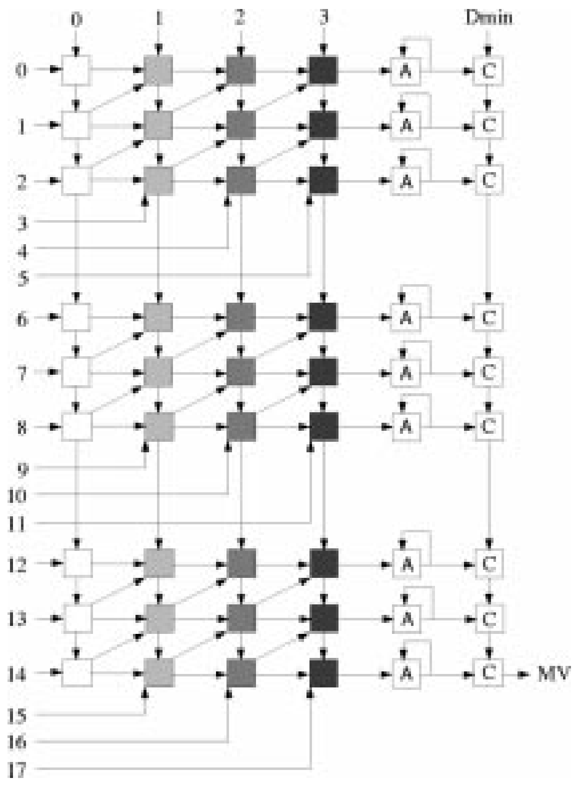


(b)

Fig. 17. (a) Partially localized three-dimensional DG (Type II). (b) Type II architecture.



(a)



(b)

Fig. 18. (a) Partially localized three-dimensional DG (Type III). (b) Type III architecture.

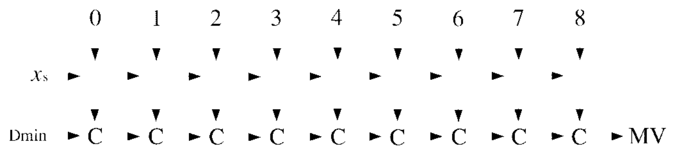


Fig. 19. One-dimensional systolic array after "multiprojection."

vector of y_s:

$$\mathcal{D}_2 = \begin{bmatrix} 1 & 0 & 0 & -3 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{vmatrix} l \\ i_2 \\ j_2 \end{vmatrix}$$

$$\mathcal{D}_3 = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{vmatrix} l \\ i_3 \\ j_3 \end{vmatrix}.$$

Three-dimensional DG's with a dependency vector of y_s only with $N = 16$ and $c = 1$ are depicted in Figs. 17(a) and 18(a) where each vertical layer of 9×64 nodes handles the computation for each subregion. By projecting the three-dimensional DG's along a projection vector $\vec{d}_1^T = [0\ 0\ 1]$ with the so-called *default schedule* direction $\vec{s}_1^T = [0\ 0\ 1]$, we obtain a two-dimensional array structure as depicted in Figs. 17(b) and 18(b). Furthermore, a one-dimensional systolic array with nine processing elements as in Fig. 19 can be obtained by using a technique called *multiprojection* which projects the two-dimensional DG along a *projection vector* $\vec{d}_2^T = [0\ \ 1]$ with a default *scheduling vector* $\vec{s}_2^T = [0\ 1]$. Compared to the Type I architecture, the throughput rate decreases by a factor

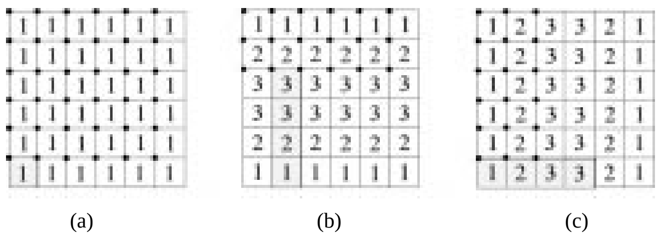


Fig. 20. Total number of search area data access. (a) Type I. (b) Type II. (c) Type III.

of 4 while decreasing the input pin count for the search area data by a factor of 2. Since the Types II and III architectures exploit only one of the two search area data dependencies in Lemma 1, multiple access of the data is inevitable. The numbers of search area data accesses for search step 0 when $N = 16$ are compared in Fig. 20. The number within each $d(0) \times d(0)$ area indicates the number of data access, and the shaded area indicates the subregion which will be loaded through input port 5 for each architecture. An example of the Type III architecture with an input data path for search step 0 when $N = 8$ is depicted in Fig. 21.

D. Address Generation

Compared to the FBMA, the data flow of the logarithmic search BMA is less regular. Random-access on-chip local memory [19] can be a feasible solution to overcome the irregular data flow and high input/output memory bandwidth. Fig. 22 depicts a block diagram of the proposed architecture with the random access on-chip local memories when $N = 16$. High throughput has been achieved by pipelining the three stages using three modules, each of which handles different target blocks in the current frame in parallel. As such, a throughput rate as high as $N^2/16$ block per clock cycle can be achieved with the Type I architecture. There are three 8×256 RAM's (2 kbits) for the target area data (RRAM) and three 8×1024 RAMs (8 kbits) for the search area data (SRAM) in the reference frame when $N = 16$. As such, each module imports target area data and search area data from the different RRAM's and SRAM's in a cyclic manner.

Assuming that the image pixels are stored column by column in a raster scan order in both local memories, the target area data addresses (REF) and search area data addresses (SA) of the proposed architectures are generated automatically by Corollary 4.

Corollary 4: At step s $(0 \le s < \nu)$, the input data address of the Type I $(\mathrm{REF}(i_1, j_1), \mathrm{SA}(r_1, j_1))$, Type II $(\mathrm{REF}(i_2, j_2), \mathrm{SA}(r_2, j_2))$, and Type III $(\mathrm{REF}(i_3, j_3), \mathrm{SA}(r_3, j_3))$ architectures are

$$\begin{aligned}\mathrm{REF}(i_1, j_1) &= i \cdot N + j \\ &= 4N \cdot d(s) \cdot \left\lfloor \frac{j_1}{4d(s)} \right\rfloor + N \cdot d(s) \cdot \left\lfloor \frac{i_1}{4} \right\rfloor \\ &\quad + N \cdot \left(\left\lfloor \frac{j_1}{d(s)} \right\rfloor \bmod d(s) \right) \\ &\quad + 4d(s) \cdot \left(\left\lfloor \frac{j_1}{d(s)^2} \right\rfloor \bmod \frac{N}{4d(s)} \right) \\ &\quad + d(s) \cdot (i_1 \bmod 4) + j_1 \bmod d(s) \end{aligned} \tag{28}$$

$$\begin{aligned}\mathrm{SA}(r_1, j_1) &= \mathrm{REF}(0, j_1) + N \cdot \left\lfloor \frac{\mathrm{REF}(0, j_1)}{N} \right\rfloor \\ &\quad + d(s) \cdot \left(r_1 \bmod 6 + 2N \cdot \left\lfloor \frac{r_1}{6} \right\rfloor \right) \\ &\quad + (2N+1) \cdot \sum_{i=0}^{s} d(i) \\ &\quad + 2N \cdot \sum_{i=0}^{s} MV1(i) + \sum_{i=0}^{s} MV2(i) \\ &= \mathrm{SA}_{\mathrm{base}}(r_1, j_1) \\ &\quad + 2N \cdot \sum_{i=0}^{s} MV1(i) + \sum_{i=0}^{s} MV2(i) \end{aligned} \tag{29}$$

$$\begin{aligned}\mathrm{REF}(i_2, j_2) &= i \cdot N + j \\ &= 4N \cdot d(s) \cdot \left\lfloor \frac{j_2}{d(s) \cdot N} \right\rfloor + i_2 \cdot d(s) \cdot N \\ &\quad + N \cdot \left(\left\lfloor \frac{j_2}{N} \right\rfloor \bmod d(s) \right) + j_2 \bmod N \end{aligned} \tag{30}$$

$$\begin{aligned}\mathrm{SA}(r_2, j_2) &= \mathrm{REF}(0, j_2) + N \cdot \left\lfloor \frac{\mathrm{REF}(0, j_2)}{N} \right\rfloor \\ &\quad + d(s) \cdot \left(r_2 \bmod 3 + 2N \cdot \left\lfloor \frac{r_2}{3} \right\rfloor \right) \\ &\quad + (2N+1) \cdot \sum_{i=0}^{s} d(i) \\ &\quad + 2N \cdot \sum_{i=0}^{s} MV1(i) + \sum_{i=0}^{s} MV2(i) \\ &= \mathrm{SA}_{\mathrm{base}}(r_2, j_2) \\ &\quad + 2N \cdot \sum_{i=0}^{s} MV1(i) + \sum_{i=0}^{s} MV2(i) \end{aligned} \tag{31}$$

$$\begin{aligned}\mathrm{REF}(i_3, j_3) &= iN + j \\ &= N \cdot \left(\left\lfloor \frac{j_3}{d(s)} \right\rfloor \bmod N \right) + 4d(s) \cdot \left\lfloor \frac{j_3}{d(s) \cdot N} \right\rfloor \\ &\quad + i_3 \cdot d(s) + j_3 \bmod d(s) \end{aligned} \tag{32}$$

$$\begin{aligned}\mathrm{SA}(r_3, j_3) &= \mathrm{REF}(0, j_3) + N \cdot \left\lfloor \frac{\mathrm{REF}(0, j_3)}{N} \right\rfloor \\ &\quad + d(s) \cdot \left(r_3 \bmod 6 + 2N \cdot \left\lfloor \frac{r_3}{6} \right\rfloor \right) \\ &\quad + (2N+1) \cdot \sum_{i=0}^{s} d(i) \\ &\quad + 2N \cdot \sum_{i=0}^{s} MV1(i) + \sum_{i=0}^{s} MV2(i) \\ &= \mathrm{SA}_{\mathrm{base}}(r_3, j_3) \\ &\quad + 2N \cdot \sum_{i=0}^{s} MV1(i) + \sum_{i=0}^{s} MV2(i) \end{aligned} \tag{33}$$

where i_1, i_2, i_3 indicate the input port of the target area data, and r_1, r_2, r_3 $(0 \le r_1 < 36,\ 0 \le r_2, r_3 < 18)$ indicate the input port of the search area data. j_1, j_2, j_3 $(0 \le j_1 < (N/d(0))^2$, $0 \le j_2, j_3 < N \cdot d(0)$ are the input order of image pixels within each subregion as discussed in Lemmas 2–4.

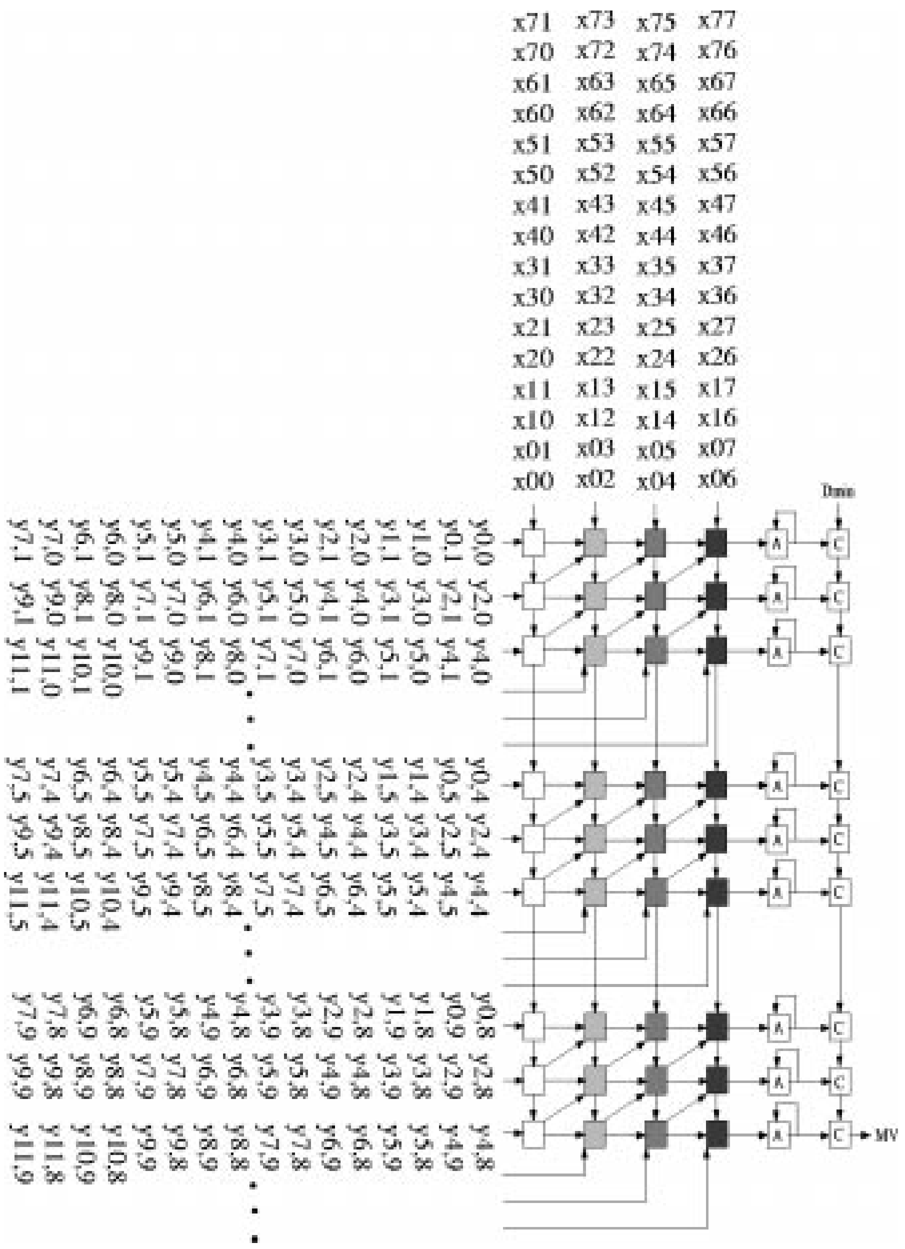


Fig. 21. Two-dimensional Type III architecture with input data path for search step 0 when $N = 8$.

Proof: The proof is given in the Appendix. ∎

Fortunately, the target area data addresses (28), (30), and (32), and the base addresses in (29), (31), and (33) are predictable. Therefore, when $N = 16$, the target area data address table can be stored in 8×256 ROM (2 kbits), and the search area data address table can be stored in advance in 10×1152 ROM (12 kbits) and 10×576 ROM (6 kbits) for Types I–III architectures. As such, the address generator calculates the search area data address locally with the base address and motion vector of the previous step. Because N is a power of two, at each step, $2N \times MV1(s)$ can be calculated recursively using shift operations.

E. Pipeline Interleaving

In practice, the search range is larger than $d_m = N/2 - 1$ for a reference block of size $N \times N$. For example, according to the Grand Alliance HDTV specification, the architecture should support the larger search range with unequal vertical and horizontal spans in the search area. An example of the search range of $-6N$ to $6N - 1$ pixels horizontally and $-N$ to $N-1$ pixels vertically is depicted in Fig. 23(a). The problem of a large search range can be solved by partitioning the search points into six subregions of $(2N)^2$ search points, and each subregion of $(2N)^2$ search points can be handled by combining the logarithmic search and the telescopic search [7] as depicted in Fig. 23(a).

Fig. 23(b) depicts the data flow diagram of the pipelined computation using three modules of our proposed architecture. Because both the logarithmic search and the telescopic search are based on the results of the previous search, delays must be inserted to prevent data hazards, which may cause performance degradation. This performance degradation caused by the idle

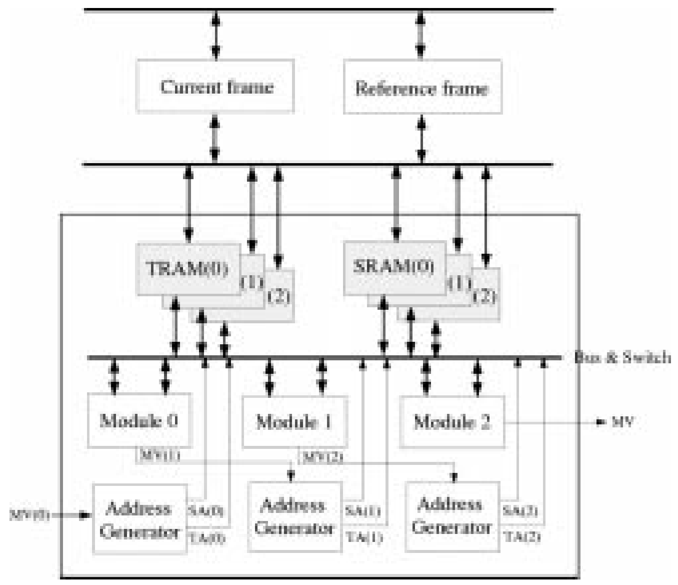


Fig. 22. Block diagram of the proposed architecture ($N = 16$).

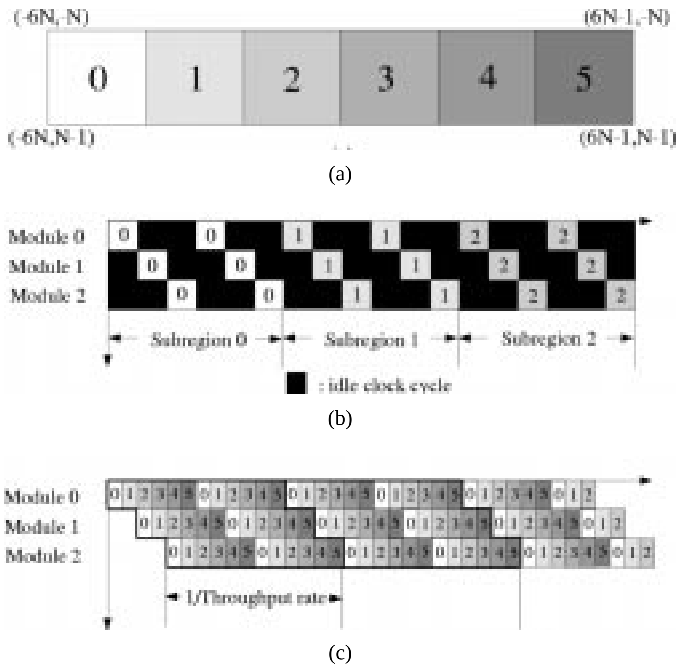


Fig. 23. Pipeline interleaving for large search range. (a) $12N \times 2N$ search points of an $N \times N$ reference block are partitioned into six subregions with $(2N)^2$ search points, respectively. (b) Data flow diagram without pipeline interleaving. (c) Pipeline interleaved data flow diagram.

clock cycles can be solved by pipeline interleaving because each subregion can be processed independently. Fig. 23(c) depicts the pipeline interleaved data flow diagram of the Type I architecture. Since the latency for the logarithmic search using the Type I architecture is $3 \times 2 \times N^2/16$ clock cycles when $N = 16$, the task of the six subregions can be interleaved to utilize the PE's 100%. By doing so, each search step can be performed every $N^2/16$ clock cycles, and the throughput rate of the Type I architecture becomes $(1/12) \times (16/N^2) = (4/3) \times (1/N^2)$ blocks/clock cycle. Hence, with a clock rate as low as 50 MHz and a search range of $-6N$ to $6N - 1$ pixels horizontally and $-N$ to $N - 1$ pixels vertically, the proposed Type I architecture can handle the Grand Alliance picture format up to 72 frames/s when $N = 16$.

TABLE I
COMPARISON: TARGET BLOCK SIZE: 16 × 16, SEARCH RANGE: $d_m = 7$

Search Method	Architecture	# PE /module	Pin count /module (×8)
FBMA	1-D array [11]	16	3
	2-D array [12]	256	3
Three-step	Jong's [15]	9	10
	Jong's [15]	36	40
	Jong's [15]	144	160
Log-step ($N = 16$)	1-D array	9	10
	Type I	144	52
	Type II,III	36	22

TABLE II
COMPARISON: THROUGHPUT RATE AND NUMBER OF DATA ACCESS WITHIN REFERENCE AREA

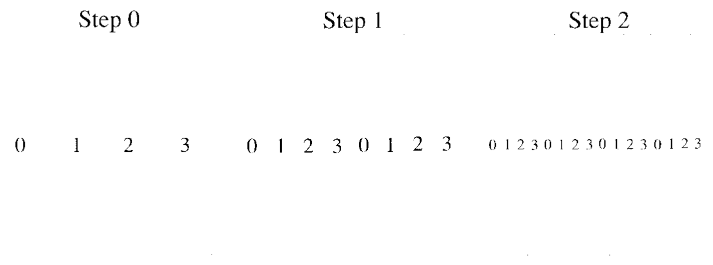
Search Method	Architecture	Throughput (block/clock)	# Access
FBMA	1-D array [11]	1/4,096	8,192
	2-D array [12]	1/256	512
Three-step	Jong's [15]	1/256	2,304
	Jong's [15]	1/64	N/A
	Jong's [15]	1/16	N/A
	Jehng's [14]	1/256	2,304
Log-step ($N = 16$)	1-D array	1/256	2,304
	Type I	1/16	576
	Type II,III	1/64	1,152

V. COMPARISON AND CONCLUSION

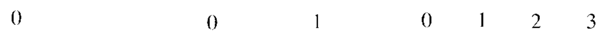
In this paper, a high-throughput modular architecture for the logarithmic search BMA has been proposed. Depending on the data input ordering schemes, we proposed a one-dimensional linear array, two-dimensional Type I, and Type II (Type III) architectures which feature throughput rates of $1/N^2$, $16/N^2$, and $4/N^2$ blocks/clock cycle, respectively. A comparison of our proposed architecture with the other existing architectures for the two different search methods is presented in Tables I and II. Table II compares the total number of data accesses within the reference area for a target block when $N = 16$ and $d_m = 7$. The comparison shows that with the same number of PE's,[4] Jong's and our proposed architecture have an identical throughput rate. However, input pin count and memory bandwidth have been reduced dramatically by exploiting the reference area overlapping pattern. Special data input ordering constraints corresponding to the reference area overlapping pattern have been proposed.

The proposed architecture can handle the large search range with unequal vertical and horizontal spans in the search area with 100% processor utilization by using the technique called pipeline interleaving. As discussed in Section IV-E, the proposed Type I architecture can handle real-time high-volume video processing such as HDTV. However, our proposed architecture requires a rather large number of input ports, which may render this architecture impractical. Architectures with a reduced number of input port have been proposed in Section IV-C to lessen the burden of large input pin count. Moreover, different matching criteria other than the TAD are

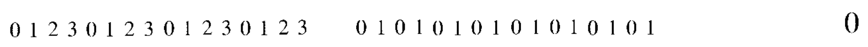
[4]The complexity of PE for both architectures is the same.



(a)



(b)



(c)

Fig. 24. Image pixel segmentation according to i $(0 \leq i < N)$, j_1 $(0 \leq j_1 < (N/d(0))^2)$, and j_2 $(0 \leq j_2 < N \cdot d(0))$. (a) $\lfloor (i/d(s)) \rfloor \bmod 4$. (b) $\lfloor (i/4d(s)) \rfloor$, $\lfloor (j_1/4d(s)) \rfloor$, $\lfloor (j_2/d(s) \cdot N) \rfloor$. (c) $i \bmod d(s)$, $\lfloor (j_1/d(s)) \rfloor \bmod d(s)$, $\lfloor (j_2/N) \rfloor \bmod d(s)$.

being applied to reduce the number of input ports and hardware complexity for low-power application.

Appendix

A. Proof of Lemma 1

I) When $i \geq N - c \cdot d(s)$ or $m = 1$,

$$i + c' \geq N \text{ or } m - c' < 0.$$

Therefore, there is no common region between two candidate blocks indexed by (m, n) and $(m - c', n)$.

II) In the same way, when $j \geq N - c \cdot d(s)$ or $n = 1$,

$$j + c' \geq N \text{ or } n - c' < 0.$$

Therefore, there is no common region between two candidate blocks indexed by (m, n) and $(m, n - c')$.

B. Proof of Lemma 2

1) (7): By Figs. 24(a) and 25(a).
2) (8): Because each subblock is composed of $d(s)^2$ pixels, this can be proved by Fig. 24(b), (c) and Fig. 25(b), (c).
3) (9): By Fig. 24(b), (c).
4) (10): By Fig. 25(b), (c).

C. Proof of Corollary 1

I) When $i_1 \geq 4(4 - c)$ or $0 \leq l < 3c$,

$$l - 3c' < 0 \text{ or } i_1 + 4c' > 15.$$

Therefore, there is no common region between two $N \times N$ candidate blocks indexed by l and $l - 3c'$ in the reference frame.

Otherwise, by (12),

$$y_s(l, i_1, j_1) = y_s(l - 3c', i_1 + 4c', j_1).$$

This is because

$$\left\lfloor \frac{i_1}{4} \right\rfloor + \left\lfloor \frac{l}{3} \right\rfloor = \left\lfloor \frac{i_1 + 4c'}{4} \right\rfloor + \left\lfloor \frac{l - 3c'}{3} \right\rfloor$$
$$i_1 \bmod 4 + l \bmod 3 = (i_1 + 4c') \bmod 4 + (l - 3c') \bmod 3.$$

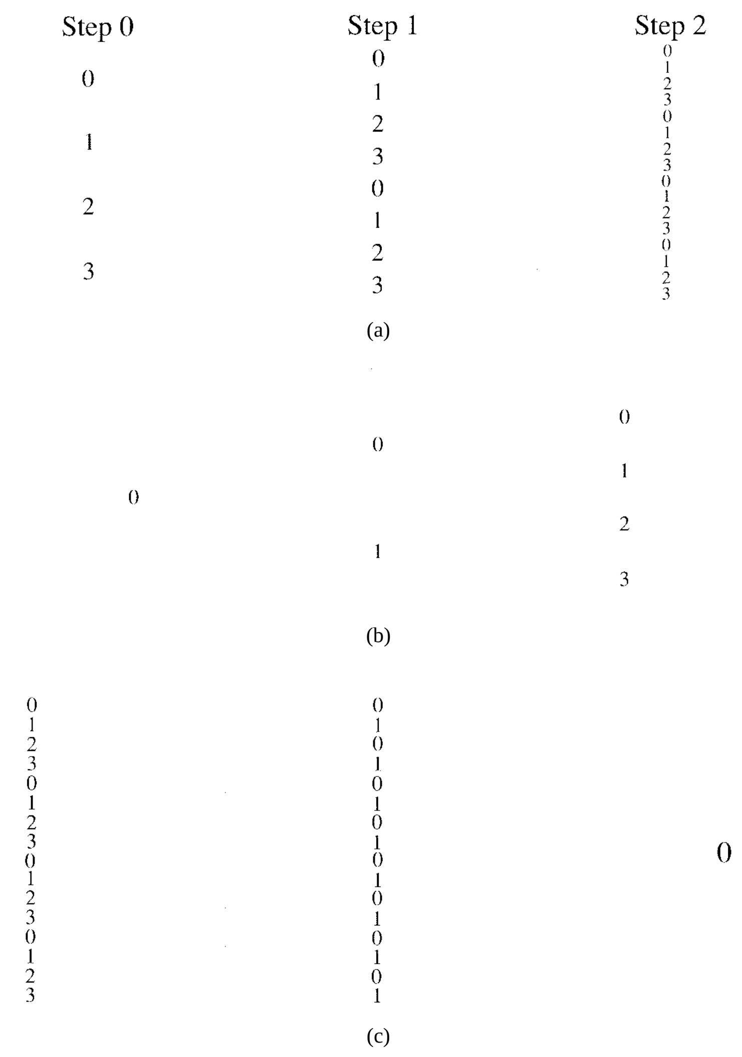


Fig. 25. Image pixel segmentation according to i $(0 \le i < N)$, j_1 $(0 \le j_1 < (N/d(0))^2)$, and j_3 $(0 \le j_3 < N \cdot d(0))$. (a) $\lfloor (j/d(s)) \rfloor \bmod 4$. (b) $\lfloor (j/4d(s)) \rfloor$, $\lfloor (j_1/d(s)^2) \rfloor \bmod \lfloor (N/4d(s)) \rfloor$, $\lfloor (j_3/d(s) \cdot N) \rfloor$. (c) $j \bmod d(s)$, $j_1 \bmod d(s)$, $j_3 \bmod d(s)$.

II) When $l \bmod 3 = 0$ or $i_1 \bmod 4 \ge 4 - c$

$$\left\lfloor \frac{i_1}{4} \right\rfloor + \left\lfloor \frac{l}{3} \right\rfloor \ne \left\lfloor \frac{i_1 + c'}{4} \right\rfloor + \left\lfloor \frac{l - c'}{3} \right\rfloor$$
$$i_1 \bmod 4 + l \bmod 3 \ne, (i_1 + c') \bmod 4 + (l - c') \bmod 3.$$

On the other hand, when $0 \le i_1 \bmod 4 < 4 - c$ and $l \bmod 3 \ne 0$, we have

$$y_s(l, i_1, j_1) = y_s(l - c', i_1 + c', j_1).$$

D. Proof of Lemma 3

1) (18): By Fig. 24(a).
2) (19): Because each column is composed of N pixels, j_2 is the data input order within each subregion indexed by i_2 by Fig. 24(c).
3) (20): By Fig. 24(b), (c).
4) (21): Because each column is composed of N pixels and j_2 is the input order within each subregion, $j = j_2 \bmod N$.

E. Proof of Lemma 4

1) (22): By Fig. 25(a).
2) (23): By Fig. 25(b), (c).
3) (24): Because each column within each subregion is composed of $d(s)$ pixels, $i = \lfloor (j_3/d(s)) \rfloor \bmod N$.
4) (25): By Fig. 25(b), (c).

F. Proof of Corollary 2

When $i_2 \ge 4 - c$ or $0 \le l < 3c$,

$$l - 3c' < 0 \text{ or } i_2 + c' > 3.$$

Therefore, there is no common region between two $N \times N$ candidate blocks indexed by l and $l - 3c'$ in the reference frame.

Otherwise

$$y_s(l, i_2, j_2) = y_s(l - 3c', i_2 + c', j_2).$$

This is because

$$i_2 + \left\lfloor \frac{l}{3} \right\rfloor = i_2 + c' + \left\lfloor \frac{l - 3c'}{3} \right\rfloor.$$

G. Proof of Corollary 3

When $i_3 \geq 4 - c$ or $l \bmod 3 = 0$

$$l \bmod 3 + i_3 \neq (l - c') \bmod 3 + i_3 + c'.$$

On the other hand, when $l \bmod 3 \neq 0$ and $0 \leq i_3 < 4 - c$, we have

$$y_s(l, i_3, j_3) = y_s(l - c', i_3 + c', j_3).$$

H. Proof of Corollary 4

1) (28): By (9) and (10).
2) (30): By (20) and (21).
3) (32): By (24) and (25).
4) (29), (31), (33): By Fig. 20, the offsets of the search area data address with respect to the target block data address are $d(s) \cdot (r_1 \bmod 6 + 2N \cdot \lfloor (r_1/6) \rfloor) + (2N + 1) \cdot \Sigma_{i=0}^{s} d(i)$, $d(s) \cdot (r_2 \bmod 3 + 2N \cdot \lfloor (r_2/3) \rfloor) + (2N + 1) \cdot \Sigma_{i=0}^{s} d(i)$, and $d(s) \cdot (r_3 \bmod 6 + 2N \cdot \lfloor (r_3/6) \rfloor) + (2N + 1) \cdot \Sigma_{i=0}^{s} d(i)$, respectively. Furthermore, $N \cdot \lfloor (\text{REF}(0, j_1)/N) \rfloor$, $N \cdot \lfloor (\text{REF}(0, j_2)/N) \rfloor$, and $N \cdot \lfloor (\text{REF}(0, j_3)/N) \rfloor$ have been added because the search area of size $2N \times 2N$ is larger than the target block of size $N \times N$. As such, the search area data addresses can be calculated by adding the displacement $(= 2N \cdot \Sigma_{i=0}^{s} MV1(i) + \Sigma_{i=0}^{s} MV2(i))$ to the fixed offsets reculsively.

References

[1] "Generic coding of moving pictures and associated audio information: Video," *ISO/IEC 13818-2: Draft International Standard*, Nov. 1993.

[2] "Grand Alliance HDTV system specification version 2.0," Dec. 1994.

[3] H. G. Musmann, P. Pirsch, and H. J. Grallert, "Advances in picture coding," *Proc. IEEE*, vol. 73, pp. 523–548, Apr. 1985.

[4] 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.

[5] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *Proc. NTC'81*, New Orleans, LA, Nov. 1981, pp. G5.3.1–G5.3.5.

[6] R. Srinivasan and K. R. Rao, "Predictive coding based on efficient motion estimation," in *Proc. ICC 1984*, May 1984, pp. 516–520.

[7] X. Lee, P. Zhou, and A. Leon-Garcia, "Efficient MPEG motion compensation scheme by motion trajectory tracking method," *Proc. SPIE*, vol. 1818, no. 2, pp. 594–605, 1992.

[8] T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1301–1308, Oct. 1989.

[9] C. H. Hsieh and T. P. Lin, "VLSI architecture for block-matching motion estimation algorithm," *IEEE Trans. Circuits Systems Video Technol.*, vol. 2, pp. 169–175, June 1992.

[10] L. D. 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.

[11] K. M. Yang, M. T. Sun, and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1317–1325, Oct. 1989.

[12] 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.

[13] E. Chan and S. Panchanathan, "Motion estimation architecture for video compression," *IEEE Trans. Consumer Electron.*, vol. 39, pp. 292–297, Aug. 1993.

[14] Y. S. Jehng, L. G. Chen, and T. D. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms," *IEEE Trans. Signal Processing*, vol. 41, pp. 889–899, Feb. 1993.

[15] H. M. Jong, L. G. Chen, and T. D. Chiueh, "Parallel architectures for 3-step hierarchical search block-matching algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, pp. 407–416, Aug. 1994.

[16] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, pp. 438–442, Aug. 1994.

[17] H. M. Jong, L. G. Chen, and T. D. Chiueh, "Accuracy improvement and cost reduction of 3-step search block matching algorithm for video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, pp. 88–90, Feb. 1994.

[18] S. Y. Kung, *VLSI Array Processors.* Englewood Cliffs, NJ: Prentice Hall, 1988.

[19] H. Jeschke, K. Gaedke, and P. Pirsch, "Multiprocessor performance for real-time processing of video coding applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, pp. 221–230, June 1992.

Hangu Yeo (S'94) received the B.S. degree in electronic engineering from Yonsei University, Seoul, Korea, in 1985, the M.S. degree in electrical engineering from Columbia University, New York, NY, in 1987, and the Ph.D. degree in electrical and computer engineering from University of Wisconsin, Madison, in 1997.

From 1990 to 1992, he worked as a Computer Assistant at the Ministry of National Defense, Korea. He is currently with IBM T. J. Watson Research Center, Yorktown Heights, NY. His research interests include video/image compression and the design of VLSI architectures for video and image processing.

Yu Hen Hu (S'79–M'80–SM'87) received the B.S.E.E. degree from National Taiwan University, Taipei, Taiwan, in 1976 and the M.S.E.E. and Ph.D. degrees in electrical engineering from the University of Southern California, Los Angeles, in 1980 and 1982, respectively.

From 1983 to 1987, he was an Assistant Professor in the Electrical Engineering Department, Southern Methodist University, Dallas, TX. He joined the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, in 1987 as an Assistant Professor (1987–1989), and is currently an Associate Professor. His research interests include multimedia signal processing, artificial neural networks, fast algorithms, and design methodology for application specific microarchitectures, as well as computer-aided design tools for VLSI using artificial intelligence. He has published more than 150 journal and conference papers in these areas. He is currently Associate Editor of *Journal of VLSI Signal Processing.*

Dr. Hu is a former Associate Editor (1988–1990) for the IEEE TRANSACTIONS OF ACOUSTICS, SPEECH, AND SIGNAL PROCESSING in the areas of system identification and fast algorithms. He is a founding member of the Neural Network Signal Processing Technical Committee of the IEEE Signal Processing Society, and served as Chair from 1993 to 1996. He is a former member of the VLSI Signal Processing Technical Committee of the Signal Processing Society. Currently he serves as the Secretary of the IEEE Signal Processing Society.