계층적 블럭매칭 알고리즘을 위한 파이프라인식 VLSI 아키텍쳐

김 형 철[†] · 맹 승 렬[††]

요 약

본 논문에서는 계층적 블록매칭 알고리즘(HBMA)을 위한 두 가지 병렬 VLSI 아키텍쳐를 제안한다. HBMA는 계층에 따른 반복 수행과 공간 인터폴레이션을 기반으로 수행되며, 이러한 수행 특성은 병렬처리의 장애요소인 데이타 종속성을 내재하고 있다. 제안된 아키텍쳐는 HBMA의 계층간 데이타 종속성을 해결하기 위하여 기본적으로 파이프라인 구조를 채택하고 있으며, HBMA에서 주어진 매개변수에 따라 세단계의 스테이지로 구성된다. 제안된 아키택쳐는 입력 프레임 데이타의 흐름을 제어하는 방식에 따라 두 가지 종류로 구분된다. U-Architecture는 단방향 스캔 순서를 따르도록 설계되었으며, B-Architecture는 양방향 스캔 수서를 따르도록 설계되었다. 각 아키텍쳐의 내부 메모리와 인터폴레이션 모듈은 해당 스캔 순서에 따라 동기적으로 동작할 수 있는 구조를 가진다.

성능분석의 결과로서 본 논문에서 제안한 두 가지 아키텍쳐가 모두 방송용 비디오 포맷을 실시간으로 처리할 수 있음을 보이고, HDTV 포맷은 가까운 장래의 VLSI 기술로 실시간 성능을 얻을 수 있음을 보였다. 또한, B-Architecture는 공간 연결성 내부 메모리 구조를 채택함으로써 입력 데이타의 재활용도를 높이고, 이에 따라 U-Architecture에 비해서 데이타 입출력 핀의 개수를 약 반정도 줄일 수 있는 특성을 보이고 있다.

Pipelined VLSI Architectures for the Hierarchical Block-Matching Algorithm

Hyung-Chul Kim[†] · Seung-Ryoul Maeng[††]

ABSTRACT

This paper presents two parallel VLSI architectures for the hierarchical block matching algorithm (HBMA). The repeated procedure of HBMA and the bilinear interpolation cause the data dependencies that are obstacles in parallel processing. The proposed architectures have the pipeline scheme to mainly solve the interlayer data dependency. From two possible order of vector flow, two specific three-stage architectures are designed based on the given parameters of HBMA. U-Architecture follows the unidirectional scan order and B-Architecture follows the bidirectional scan order. The internal memory and the interpolation unit can fulfill the designated scan order in synchronous way. The performance results show that both architectures can process in real-time up to the broadcast video format under the current VLSI technology, and the HDTV video format with the near future VLSI capabilities. Both architectures also achieve nearly linear speedup over an assumed non-pipelined VLSI architecture. Compared to U-Architecture, B-Architecture reduces 50% of pin count owing to the wraparound memory scheme.

[†]정 회 원 : 한국전자통신연구원 컴퓨터 · 소프트웨어기술연구소 멀티미디어연구부 선임연구원
[††]정 회 원 : 한국과학기술원 전산학과 교수
논문접수 : 1997년 12월 8일, 심사완료 : 1998년 1월 26일

1. Introduction

Many video coding techniques have been bro -ught out in order to deliver the huge amount of video data via the channel or storage of res -tricted bandwidth. One of the highly promising and most powerful methods in video coding is motion compensation [1]. It uses a motion esti -mation scheme to reduce the temporal redund -ancies of consecutive image frames. Motion compensation can be applied to two areas of motion-compensated prediction (MCP) and motion -compensated interpolation (MCI). MCP is used in the interframe predictive coding where the prediction error is used for reconstructing the original data. Since the prediction error term is quantized and transmitted at the low bit rate, motion estimation tries to look for the low error term instead of the exact motion field. Hence, a highly accurate motion estimation may not be crucial in predictive coding [2]. On the other hand, MCI is used in the interframe interpolative coding. The skipped frames at the transmitter are recovered by interpolating along mot-i on trajectories at the receiver. For MCI, highly accurate estimates are required to give good image quality since the error terms would cause highly visible artifacts that are not adjusted [3].

Our target application is MCI for the following reasons. It is a very attractive method of video coding since it can be combined with the known coding techniques to reduce the bit rate further. Many literatures have demonstrated that MCI is a novel method to recover the skipped frames than any other simple methods [4][5]. Accurate motion field is strongly required in MCI and an accuracy constraint of motion estimation can be defined by following two conditions: 1) moti -on vectors should imply the actual motion of moving objects as exactly as possible rather than the minimal error term, and 2) a motion vector should be assigned to a small group of pixels as possible; ultimately, a unique motion vector for a pixel [3][6]. There is also a real-time constraint on motion estimation to satisfy the frame rate for proper presentation. Because the required frame rate is 30 frames/sec in usual and every other frame should be recovered by MCI at the worst case, motion estimation for MCI should provide the frame rate of 15 frames/ sec.

There have been various approaches on motion estimation : a gradient-based approach [7], a feature-based approach [2], and a block-match -ing approach [2]. In video coding area, a block-matching algorithm (BMA) is widely used due to the simplicity of hardware implementation encouraged by the regular data flow. However, BMA has limitations on being directly applied to MCI since 1) it tends to fall into local minimum regardless of exact motion of moving obje -cts, and 2) it also fail to give accurate estimates if different parts of a block have various motions [8][9].

The *hierarchical block-matching algorithm* (HBMA)[10] is an improved algorithm of BMA in terms of accuracy because it copes with global motion of objects as well as local motion. Further, HBMA generates a dense motion field than BMA since it eventually gives every pixel a motion vector [5]. HBMA is a sort of multilaye -red algorithm based on both BMA and a spatial interpolation; it computes motion vectors by proceeding from higher layers (coarse grid) to lower layers (fine grid). HBMA repeatedly app -lies BMA with various set of parameter values in order to process a frame. That is, pixel data of a frame should pass all layers to complete the computation of the frame. It is well establi -shed that BMA itself is computationally very complex and a lot of researches on parallel VL SI architecture based on systolic array have endeavored to process BMA in real-time [11]

[12][13][14][15]. On the other hand, it is noticed that the computational complexity of HBMA is several tens times higher than that of BMA to process a frame [16]. Therefore, a specialized parallel VLSI architecture for HBMA is required to meet the real-time constraint.

Relatively less work has been done on the parallel VLSI architecture for HBMA. In recent, Gupta and et al. [17] propose a tree-type parallel VLSI architecture that can process two-layer HBMA in real-time. However, we argue that their architecture is not sufficient to process HBMA correctly since the inherent data dependency of HBMA is not considered. That is, all layers are processed at once to compute a motion vector without any refinement of the vector even though every vector should be updated along the lower layer goes on.

The data dependency of HBMA resides in the repeated procedure on various grid of motion field. A motion vector that is computed at a layer is updated at the next layer (interlayer data dependency). Multiple motion vectors are used to interpolate other motion vectors at a layer (intralayer data dependency). To resolve these data dependen- cies the flow of motion vectors between processing components should be maintained to be in a particular order. Further, the flow of motion vectors is tightly dependent on the way that the frame data are scanned to successively compute the motion vectors. Based on the horizontal scan scheme, there are two possible scan order of an unidirectional scan order (rightward only) and a bidirectional scan order (rightward and leftward). These scan orders allow a parallel architecture to be designed in different ways and these are able to effect the processing performance and the complexity of the architecture.

This paper presents two pipelined VLSI architectures for HBMA that resolve the data dependencies. The pipelined configuration can partly solve the interlayer data dependency and can simply provide the scalability if the number of layers of HBMA is increased. U-Architecture is based on the unidirectional scan order and B-Architecture is based on the bidirectional scan order. The proposed U-Architecture is an extended one of our previous work[16] that was designed without consideration of any memory scheme for frame data. Both of architectures in this paper are in three-stage pipeline according to the given parameters of three-layer HBMA as special cases. The detailed design of each component is focused on that the flow of the intermediate result of motion vectors could be regular and synchronous. The multi-stage configuration and the interpolation unit cure interlayer data dependency. The interpolation unit including latch mechanism satisfies Intralayer data dependency. The performances of U-Architecture and B-Architecture are analyzed in terms of real-time processing aspect, pipeline processing aspect, and VLSI aspect.

2. Data Dependency and Our Approach

2.1. Data Dependency of HBMA

HBMA basically performs Block-Matching Algorithm (BMA) and bilinear interpolation in a recursive way. In BMA, a reference block of $(n \times n)$ pixels in the present frame is compared with the candidate blocks within a search area of $(n+2p) \times (n+2p)$ pixels of the reference frame. The motion vector corresponding to each block is estimated for the best match. To process a frame, HBMA of r layers performs BMA with a different set of parameters $\{n_i, p_i, s_i\}$ on each different layer i, where $1 \leq i \leq r$. n_i and p_i determine the sizes of reference block and search area, respectively. Hence the possible maximum value of a vector is $\sum_{i=1}^{r} p_i$ pixels.

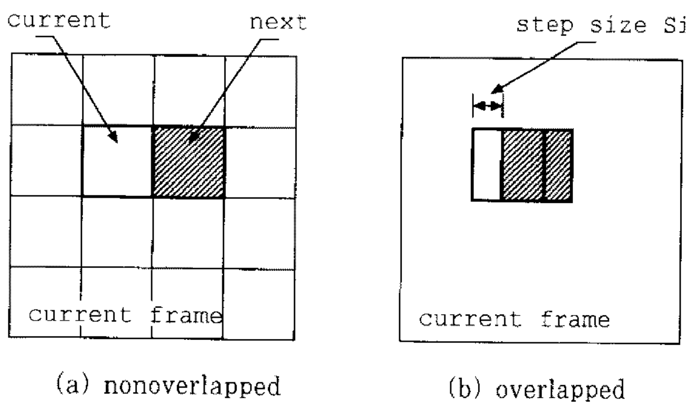
s_i is the step size of layer i that determines the grid size of vector field: i.e., it is the distance between pixels where a reference block is made at layer i. Figure 1 illustrates the concept of the step size.

Figure 2 illustrates the conceptual procedure of HBMA. A black square denotes the pixel position on which an estimated vector is given by BMA. A white square denotes the pixel position on which an interpolated vector is given by bilinear interplation.

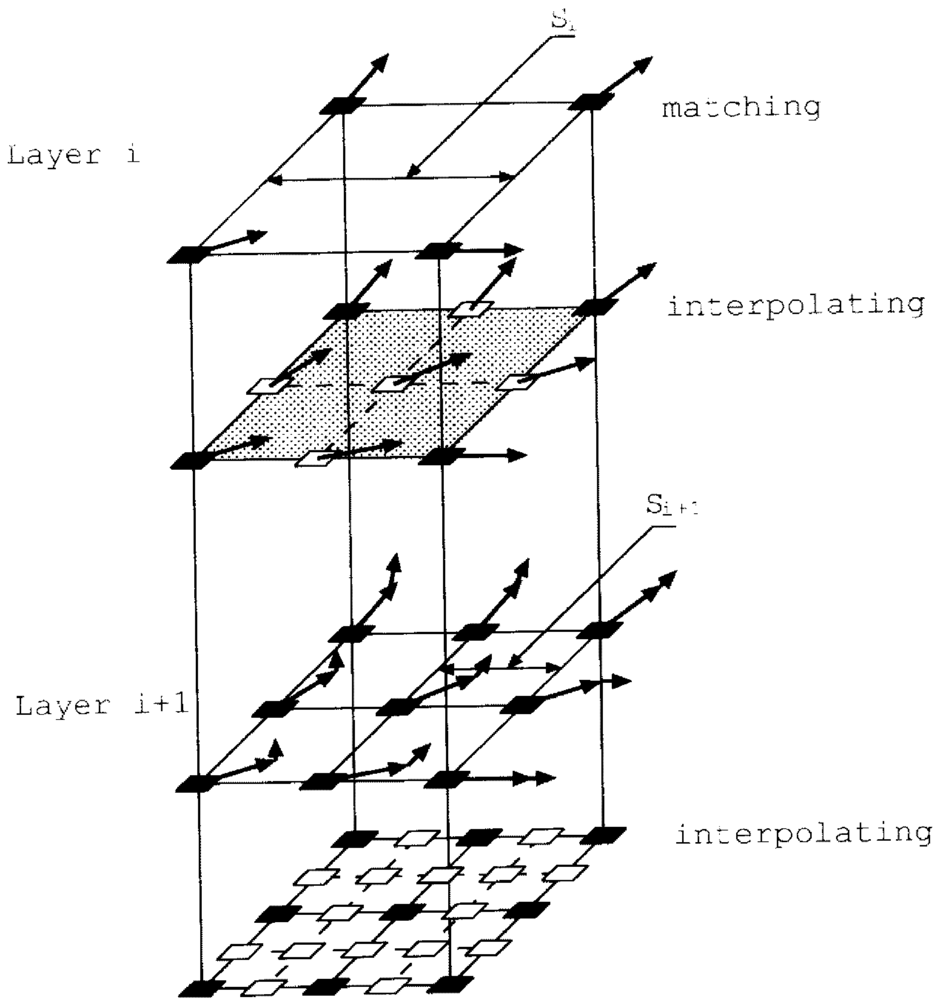
On layer i, a vector is estimated by BMA of using a large reference block and a large search area to cope with global motion of objects. The estimated vector is given to a central pixel of the reference block. BMA is made for every s_ith pixel along horizontal as well as vertical direction. After completion of the estimation procedure on layer i, every s_ith pixel has its own *estimated vector*. For every s_{i+1}th pixel that does not have its estimated vectors, an interpolated vector is given by bilinear interpolation as its initial value. Completing the procedure of layer i, every s_{i+1}th pixel has a vector that will also be used to determine the new coordinates of the search area for BMA of layer $i+1$. On layer $i+1$, BMA is performed again to estimate another motion vectors to find out local motion of objects. For local motion, the sizes of the reference block and the search area are reduced. The location of the reference block is centered at s_{i+1}th pixel, but the location of the search area is centered at the position displaced by the given vector value of the same pixel. The estimated vector at layer $i+1$ and the given vector at layer i are added to form an updated vector of the s_{i+1}th pixel. After bilinear interpolation for every s_{i+2}th pixel is completed, the next layers are repeated until s_{i+n} becomes one. Table 1 shows the given parameters of HBMA for three layers [10].

〈Table 1〉 Parameters of HBMA for three layers

Parameters at layer	i	1	2	3
Maximum update displacement	p_i	±7	±3	±1
Reference block size	n_i	64	28	12
Step size	s_i	8	4	2



(Fig. 1) Strategies for arranging two consecutive reference blocks : (a) in case of BMA and (b) in case of HBMA based on step size



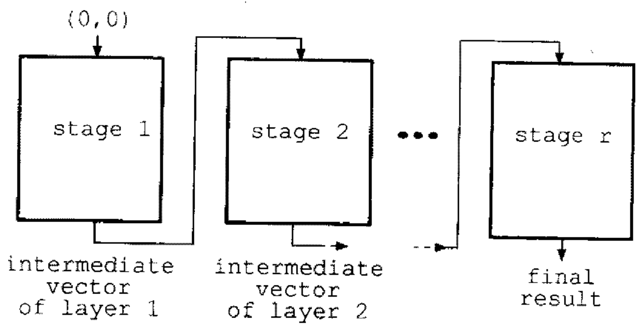
(Fig. 2) Hierarchical block-matching algorithm

Bilinear interpolation on layer i gives initial vectors to the pixels whose vectors are not esti -mated on layer i, but will be processed on layer $i+1$. In general, bilinear interpolation is given by $\hat{d} = d_1 hv + d_2 h^* v + d_3 hv^* + d_4 h^* v^*$, where $\hat{d}$ is the vector to be interpolated, d_i is the estimated vector, h is the horizontal off set, v is the vertical offset, and r^* is $(1-r)$. For instance, in the shadowed region in Figure 2, d_i denotes the vector of the pixel at the black square and $\hat{d}$ is the vector of the pixel at one of white squares. h (or v) is offset bet -ween a black square and a white square in ho r-izontal (or in vertical).

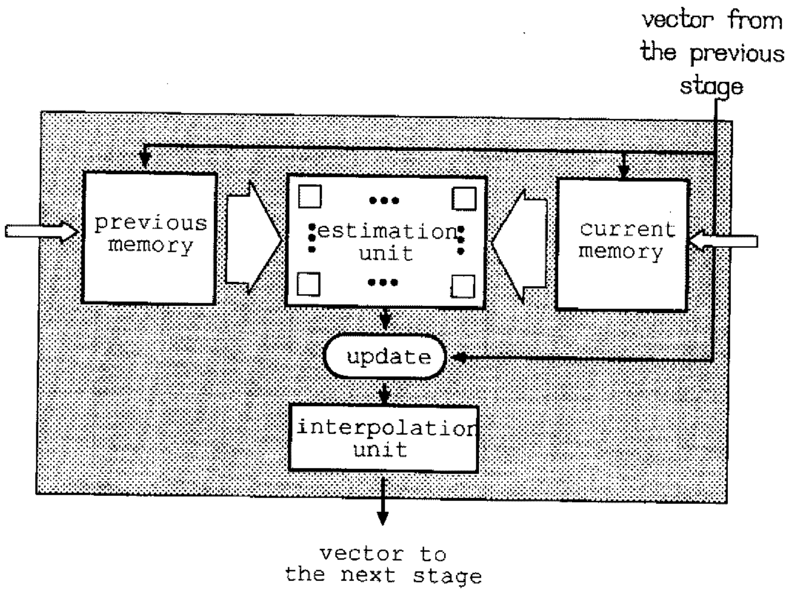
HBMA has two inherent data dependencies : *interlayer data dependency* and *intralayer data dependency*. Let $d(i,x,y)$ be a motion vector of the pixel (x,y) at layer i. Then the equationof $d(i,x,y) = d(i-1,x,y) + u(i,x,y)$ is acco-mplished, where $u(i,x,y)$ is the updated term of the motion vector for the pixel (x,y) at layer i. It shows that HBMA has interlayer data dependency. On the other hand, in bilinear interpolation, $\hat{d}$ can not be computed without d_is of the same layer. It means that HBMA has intralayer data dependency. The data dependency of an algorithm is a main obstacle in parallel processing and it should be satisfied in order to process the algorithm correctly [18].

2.2. Our Approach

Our approach is focused on mapping each layer of HBMA into a separate stage module. A stage module may be packaged onto an indiv-idual VLSI module for the scalability of HBMA. As a result, an r-stage pipeline is obtained as shown in Figure 3-(a). Figure 3-(b) illustrates the block diagram of a stage module. It consists of five major components : an estimation unit for BMA, a previous memory for search area data, a current memory for reference block data, an update subblock for add operation of two ve-ctor values, and an interpolation unit for exec-uting bilinear interpolation and managing the intermediate vectors. Based on the general module, two specific architectures are derived


(0,0)
stage 1
stage 2
stage r
intermediate vector of layer 1
intermediate vector of layer 2
final result

(a) block diagram of an r-stage pipeline



(b) block diagram of a stage module

(Fig. 3) Overview of the pipelined architecture according to the scan order in the following se ctions.

3. U-Architecture

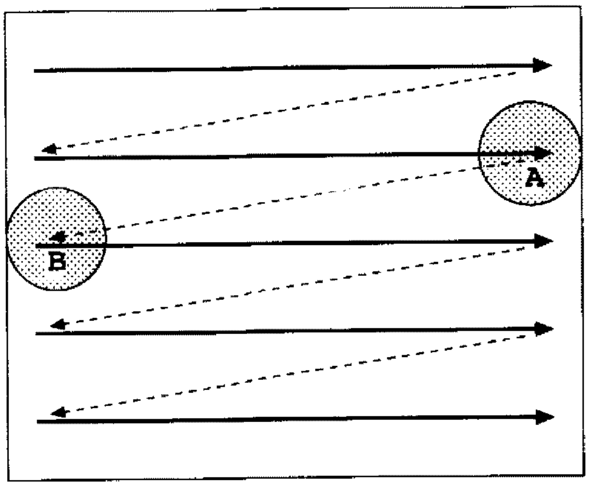
U-Architecture follows the unidirectional scan order and has three stages according to the given parameters of HBMA for three layers (Table 1). The unidirectional scan order in scanning the frame data and generating motion vectors is shown in Figure 4. All stages of the pipeline work in fully synchronous way. The flow of the intermediate vector is also sophisticatedly sche-duled to support the synchronous operation. Nevertheless, the transfer delay of frame data from the external memory for the subsequent matching would break down the fully synchronous pipeline. Hence the architecture is designed to avoid the delay by transferring image data in advance for the subsequent matching.

3.1. Estimation Unit

The estimation unit of stage i performs BMA to compute a motion vector for each reference block according to the following equations :

$$MAD = \sum_{x=1}^{n_i} \sum_{y=1}^{n_i} \left| I(x,y,t) - I(x+u, y+v, t-1) \right|$$

$$V = (u,v) \mid {}_{\min} MAD(u,v)$$



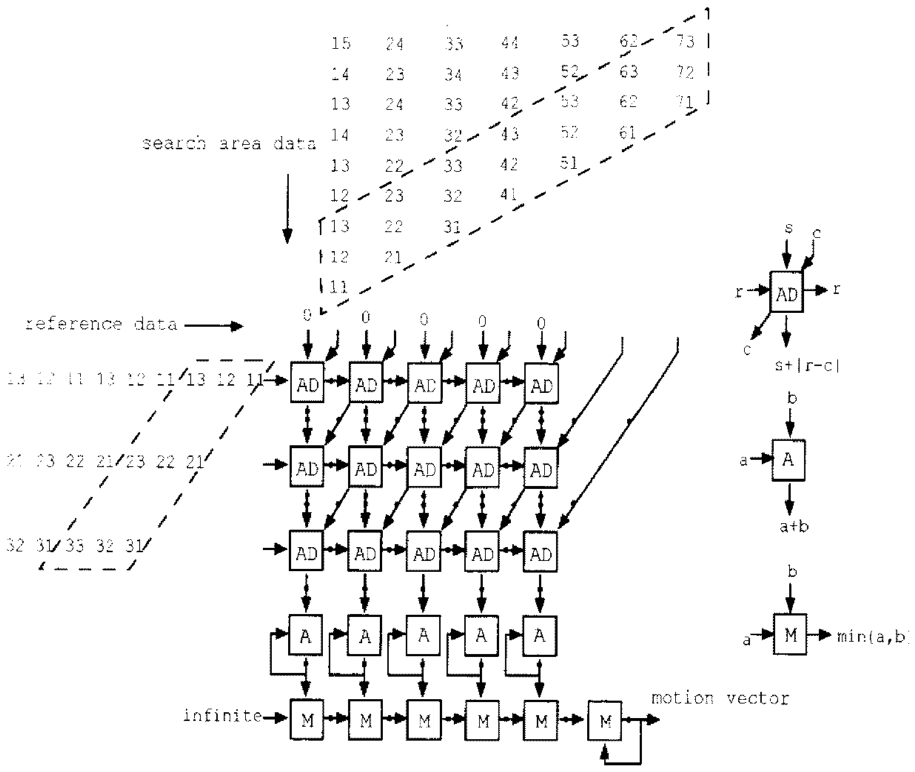
(Fig. 4) Access a frame in the unidirectional scan order

, where $I(x,y,t)$ is a luminance value of a pixel (x,y) at time t and $-p_i \le u, v < p_i$. Since BMA itself is computationally very demanding, a lot of researches have endeavored to develop the parallel VLSI architectures that can process BMA in real-time [11, 12, 13, 14, 15]. Because the parallel VLSI architectures for BMA is already abundant, we simply use Komarek and et al's systolic architecture as our estimation unit for its well-defined mapping procedure and scalability. Figure 5 shows the estimation unit and the operations of cells. The total clock cycles required for computing a motion vector at stage i is $E_i = (n_i + 1)(2p_i + 1) + 2n_i$.

3.2. Internal Memory Unit for Image Data

To compute a motion vector, the estimation unit requires frame data of a search area and a reference block. Hence, it accesses the external previous frame memory for search area data, and the external current frame memory for reference block data.

Most data of a search area as well as a reference block are accessed over and over again to compute a motion vector. For instance, Figure 6-(a) illustrates a search area of size $(n_i + 2p_i)^2$ and Figure 6-(b) shows the access pattern of the systolic array. A two-digit figure denotes a relative position of a pixel in the search area and a dashed box denotes a group of candidate blocks that are used for a matching. From Figure 6-(a) and -(b), we investigated the features of data access pattern as follows: 1) almost all of the pixels are accessed n_i times to compute a motion vector, where the size of a reference block is $n_i \times n_i$. 2) multiple systolic cells of the estimation unit do simultaneously access the frame memory. 3) data access on a horizontal line is skewed along the time line. 4) at most one pixel is accessed from a

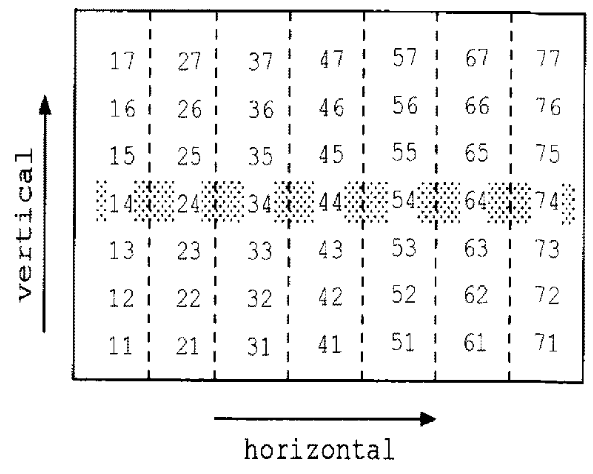


(Fig. 5) Systolic array for the full-search BMA (in case of $n_i=3, P_i=2$)[11]

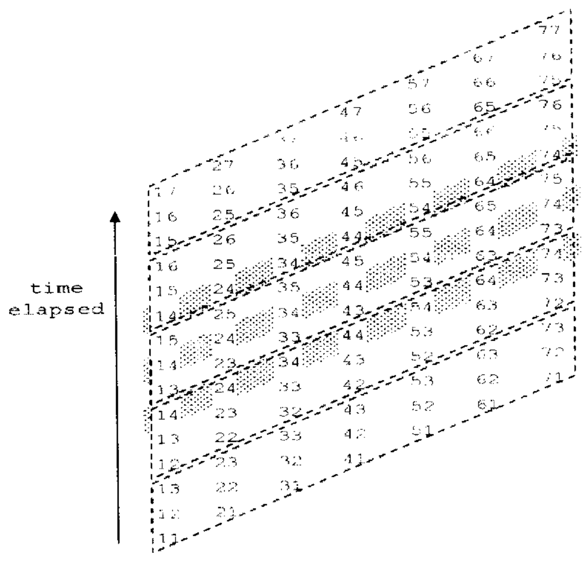
vertical line of frame data at a time, and 5) n_i contiguous pixels are sequentially accessed from a vertical line, irrespective of search area data or reference block data.

Using the on-chip internal memory can satisfy the first two requirements. Making the internal memories are of multi-module organization can satisfy the last three requirements. Even though some initial delays per data block are possible, double buffering can cure it.

The n-module memory organization is illustrated in Figure 7. It consists of n dual-port RAM modules and thus it has an input port and n output ports. It has an input logic and n output logics. The input logic selects an appropriate module where the input pixel value is stored according to its relative position within the search area or the reference block. Each module has its own output logic in order to generate an offset in the module independently from other modules. In this way, n pixels that are allocated in n separate modules can be simultaneously accessed. The function of the output logic is based on modular operation and increment operation.



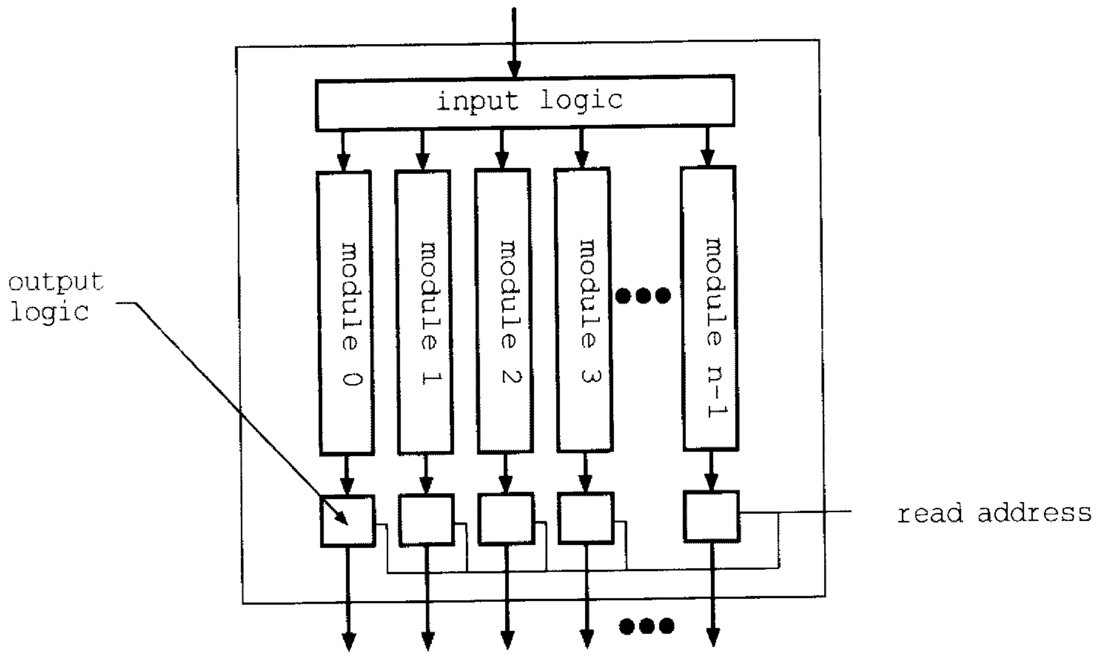
(a)



(b)

(Fig. 6) Data access pattern : (a) a search area where $n_i=3$ and $p_i=2$ and (b) data access pattern of systolic array

Two types of the internal multimodule memories are in a stage module : the *Previous Memory* (PM) and the *Current Memory* (CM). For the sake of legibility, we will consider only search area data and PM without any remark from now on. The sizes of PM and CM are $(n_i+2p_i)^2$ pi-xels and n_i^2 pixels, respectively, where the size of a search area is $(n_i+2p_i)^2$ pixels and that of a reference block is n_i^2 pixels. Although only one unit of PM or CM of those sizes are required to compute a motion vector, it is insu-fficient to store the frame data to be transfer-red in advance for the next search area or the next reference block. In the worst case at the start of a new block line, a whole search area (or reference block) should be transferred in advance. That is, in Figure 4, all pixels of the search area at B should be transferred dur-ing the search area at A is being processed since there are no overlapped block data. Ther-efore, a double buffering scheme is introduced and then there are a couple of PMs (PM-A and PM-B) and a couple of CMs (CM-A and CM-B) in a stage module. For instance, frame data for the next search area is being transferred to PM-B while frame data in PM-A is being pr

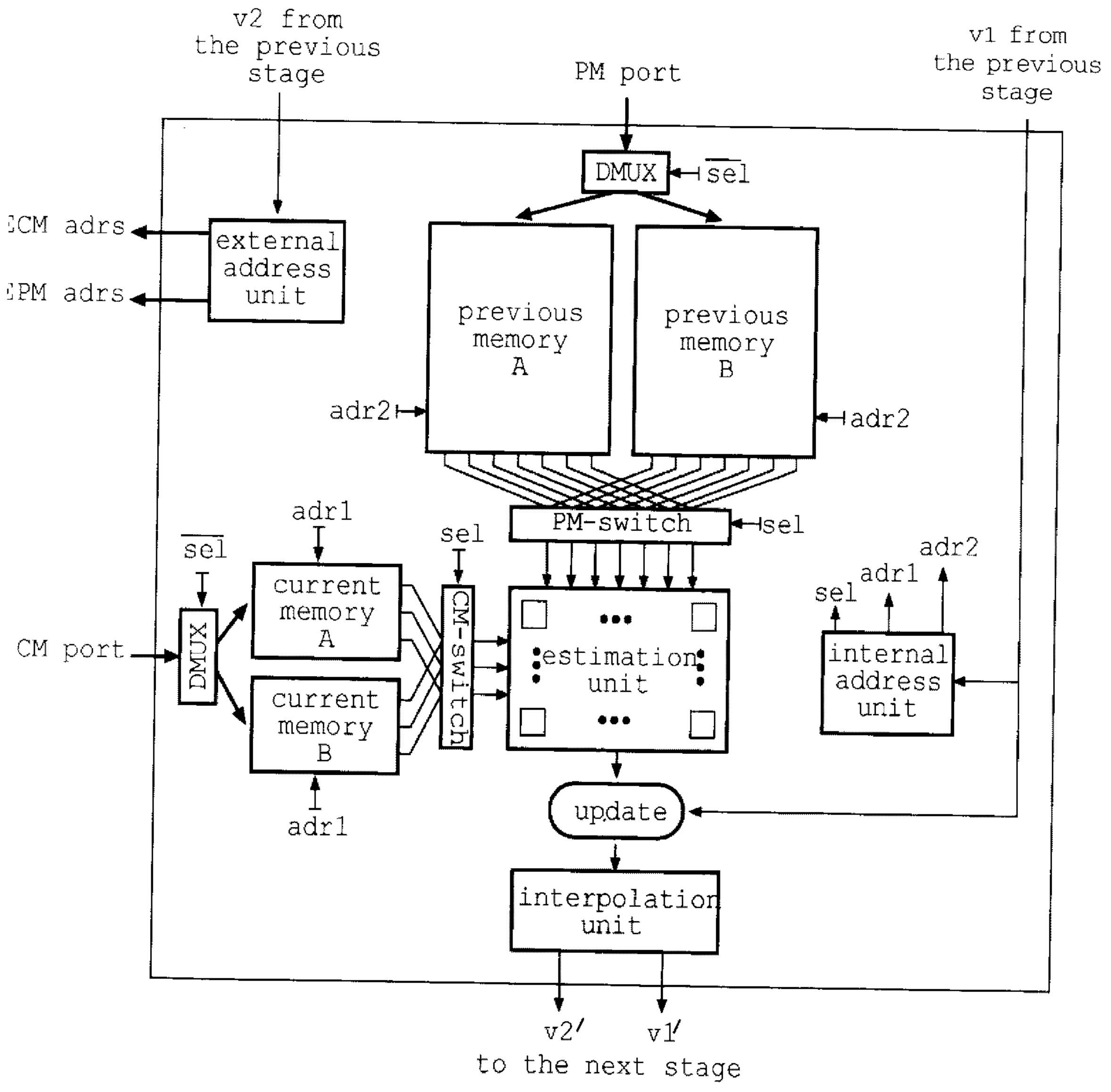


(Fig. 7) *n*-module internal memory

ocessed, and vice versa.

Figure 8 illustrates the block diagram of a stage module of U-Architecture that is focused on the memory configuration. Each stage module receives two intermediate vectors (*v1* and *v2*) from the previous stage module and supplies another two intermediate vectors (*v1′* and *v2′*) to the next stage module. *v1* is the intermediate vector to be processed in the stage. It determines the addresses of internal memories (*adr1* and *adr2*) and is updated by the result of the estimation unit. *v2* is the intermediate vector used to transfer the next search area data and reference block data in advance. It determines the address of external frame memories : *ECM address* for the external current frame memory, and *EPM address* for the external previous frame memory. Data from the external previous frame memory via *PM port* are stored in one of PMs that is used for pre-read memory.

There are two memory-to-EU switches : *PM-switch* and *CM-switch*. These need only 2-to-1 switching functions from two memory units to



(Fig. 8) Block diagram of a stage module of U-Architecture a processing unit. Thus PM-switch is an array of k 2-to-1 multiplexors and it determines that one of PM-A and PM-B is accessed by the esti-mation unit at an instance, where k is the number of read ports of a PM: $k=(n_i+2p_i)$. For CM-switch, k is n_i. The control signal, *sel*, simultaneously controls the switching of all multiplexors.

3.3. Interpolation Unit

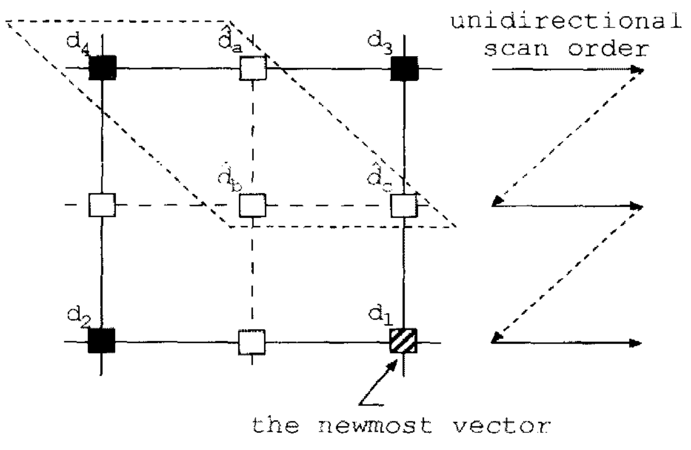
Besides performing bilinear interpolation, the interpolation unit provides a mechanism to supply the results of a stage to the next stage at correct time and another mechanism to resolve *intralayer data dependency* that bilinear interpolation would cause.

With the pipelined architecture, interlayer data dependency is resolved so naturally since the vectors are updated by simply passing through the pipeline. It implies that the intermediate results of the vectors at one stage should be precisely arranged to be updated at the next stage. Because the pipeline operates in fully synchronous way, the data flow (i.e., the flow of intermediate vectors) should keep a particular order. Furthermore, the number of the vectors to be processed at stage i is four times as many as that of generated ones at stage $i-1$ since the density of vector field at layer i is four times as many. That is, the density of vector field of layer i is determined by the step size s_i and $s_i=\frac{1}{2}s_{i-1}$ in both vertical and horizontal direction. For these reasons, an interpolation unit consists of a bilinear interpolator, an input latch mechan-ism, and an output latch mechanism.

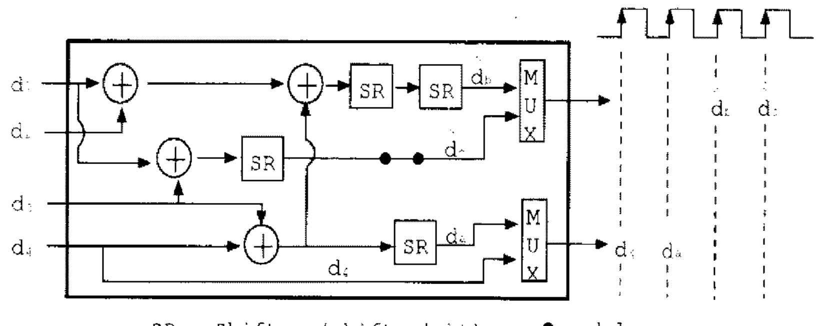
3.3.1. Bilinear Interpolator

The general bilinear interpolation can be simplified to Figure 9 from $s_i=\frac{1}{2}s_{i-1}$. The black squares denote the estimated vectors and the white squares denote the interpolated vectors. Among all possible interpolated vectors, only $\widehat{d}_a$, $\widehat{d}_b$, and $\widehat{d}_c$ are interpolated since these make it both efficient and systematic to implement than any other combinations.

Bilinear interpolation raises data depen-dencies that are obstacles against pipelined processing. One is, as shown in Figure 9, four



(Fig. 9) Simplified bilinear interpolation : $\widehat{d_a}=\frac{1}{2}(d_3+d_4)$, $\widehat{d_b}=\frac{1}{4}(d_1+d_2+d_3+d_4)$ and $\widehat{d_a}=\frac{1}{2}(d_1+d_3)$



(Fig. 10) Design of the interpolator

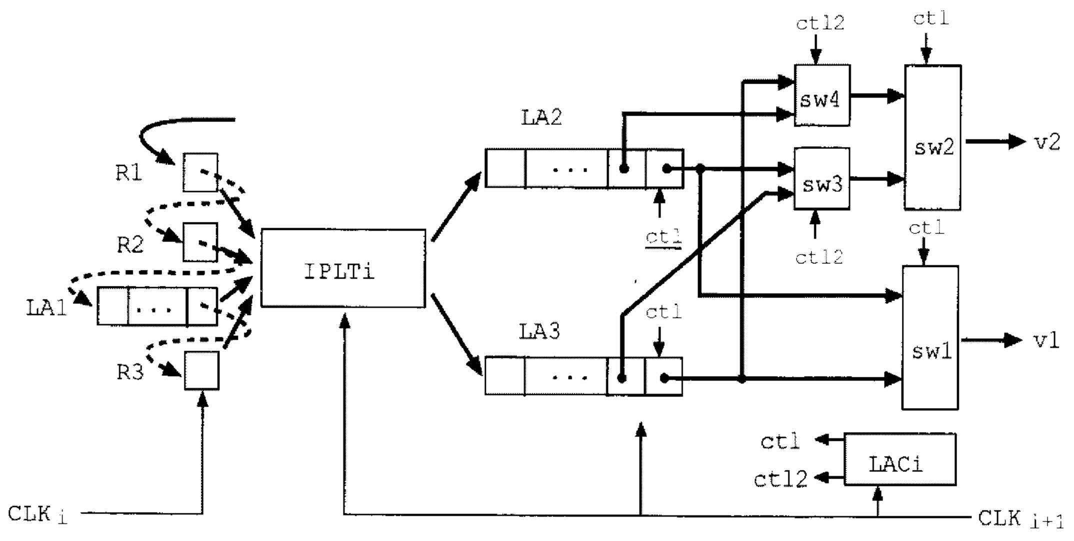
estimated vectors lying over two lines are required for bilinear interpolation. The other is that three interpolated vectors, generated after completion of bilinear interpolation, are spread over two lines. These data access pattern would cause the data flow over the whole pipeline stages to be irregular. Hence the interpolation unit is designed to alleviate these problems.

Figure 10 shows the bilinear interpolator of U-Architecture that computes $\hat{d}_a$, $\hat{d}_b$, and $\hat{d}_c$ at once. It works whenever the newmost vector, d_1, in the unidirectional scan order is available from the estimation unit. Recall that the estimation unit computes the motion vectors by scanning a frame in the unidirectional scan order. The outputs of the interpolator are clas-sified into two groups according to whether they are on the same horizontal line of frame or not.

3.3.2. Latch Mechanism

The latch mechanism regulates the flow of intermediate results of vectors. The relation-ship of the bilinear interpolator and the latch mechanism of stage i is shown in Figure 11.

The input latches take role of 1) supplying four inputs to the bilinear interpolator at once



(Fig. 11) Latch mechanism of stage i : input latches (R1, R2, R3, LA1), output latches (LA2, LA3), latch-array controller (LAC_i), and interpolator ($IPLT_i$)

by four separate latches, 2) ensuring no loss of already estimated vectors by LA1 of size N_p/s_i-1 vectors, where N_p is the number of pixels in a line of frame and s_i is the step size, and 3) keeping them on the proper input ports of the bilinear interpolator at correct time by configuring all input latches in linear connection.

The output latches store the results of the interpolator in line-by-line basis. There are two output latches of LA2 and LA3 that are first-in-first-out latch arrays. They are connected to the respective output ports of the inter-polator: $\hat{d}_b$ and $\hat{d}_c$ are forwarded to LA2 during $\hat{d}_1$ and $\hat{d}_a$ are forwarded to LA3. In this way, the flow of vectors to the next stage potentially keeps the unidirectional scan order. The sizes of LA2 and LA3 are $(N_p/s_{i+1}+1)$ vectors and $(N_p/2s_{i+1})$ vectors, respectively. The sizes are so minimal that they would neither overflow nor underflow. The function tables for the switches are defined in Table 2.

⟨Table 2⟩ Function tables for the switches

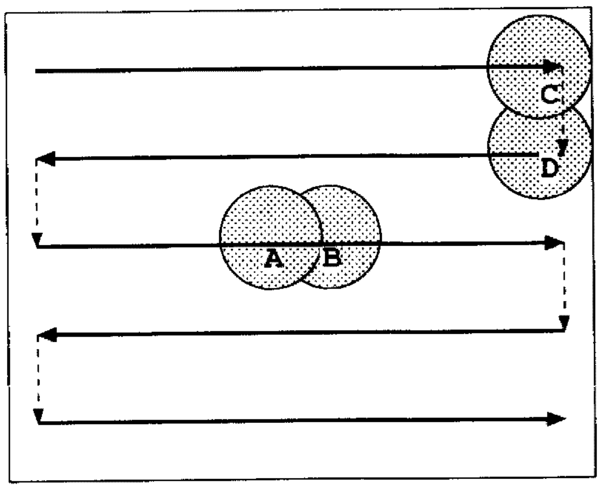
ctl	1	0
sw1	⧄	⧅
sw2	⧄	⧅

ctl2	1	0
sw3	⧄	⧅
sw4	⧄	⧅

The latch-array controller (LAC) takes role of supplying the results of a stage to the next stage in the unidirectional scan order at correct time. It generates two control signals : *ctl* and *ctl2*. LAC exclusively opens the output ends of output latches; it keeps one of output latches being opened during the vectors on a horizontal line are exhausted. The initial values of both control signals are 1s. Therefore, the synchronous data flow in the unidirectional scan order along the whole pipeline stages is achieved. The input latches use the clock CLK_i, while the int-erplator and the output latches use the clock CLK_{i+1}. CLK_{i+1} is four times faster than CLK_i and the clock subsystem is derived in our previous work [16].

4. B-Architecture

This section presents the other three-tage pipelined VLSI architecture that is referred to *B-Architecture*. B-Architecture is based on the *bidirectional scan order*. Figure 12 illustrates that a frame is accessed in the bidirectional scan order.



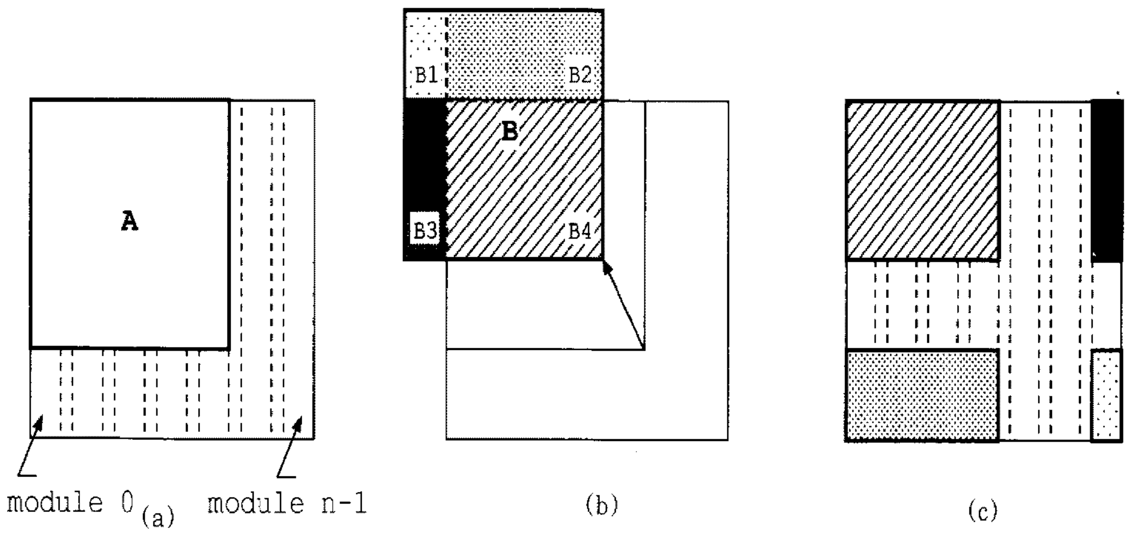
(Fig. 12) Access a frame in the bidirectional scan order

4.1. Wraparound Internal Memory Unit

With the bidirectional scan order, two blocks of frame data for consecutive estimations are always overlapped. From Figure 12, search are as (or reference blocks) at region A and at region B are overlapped. Region C and region D are overlapped even though the scan line is changed to vertical direction. This fact means that a considerable portion of two consecutive frame data blocks can be reused without retransfer. The fundamental structures of the current memory (CM) and the previous memory (PM) are the n-module memory organizations of U-Architecture as shown in Figure 7.

Furthermore, the memory units of B-Architecture have the wraparound structure in order to fully utilize the feature of data overlapping. The wraparound memory structure means that 1) the topmost cell of a module is logically adjacent to the bottommost cell of the module, and 2) the left most module is logically adjacent to the rightmost module. Hence two consecutive data blocks can be allocated in a memory unit without explicit double buffering as in U-Architecture. Figure 13 illustrates a data block is allocated in a wraparound manner. A data block A is already allocated in a memory unit and being used by the estimation unit (Figure 13-(a)). When the other data block B that is displaced by the motion vector is transferred in advance, only some part of data (B1, B2, and B3) need to be transferred since a part of data (B4) do already exist (Figure 13-(b)). Because B1, B2, and B3 are out of boundary of the memory module, they are allocated in a wraparound way (Figure 13-(c)). The horizontal wraparound scheme can be achieved by simple modular operation of the module selection logic of the address generator. The vertical wraparound scheme is the addressing matter localized within the output logic of a module.

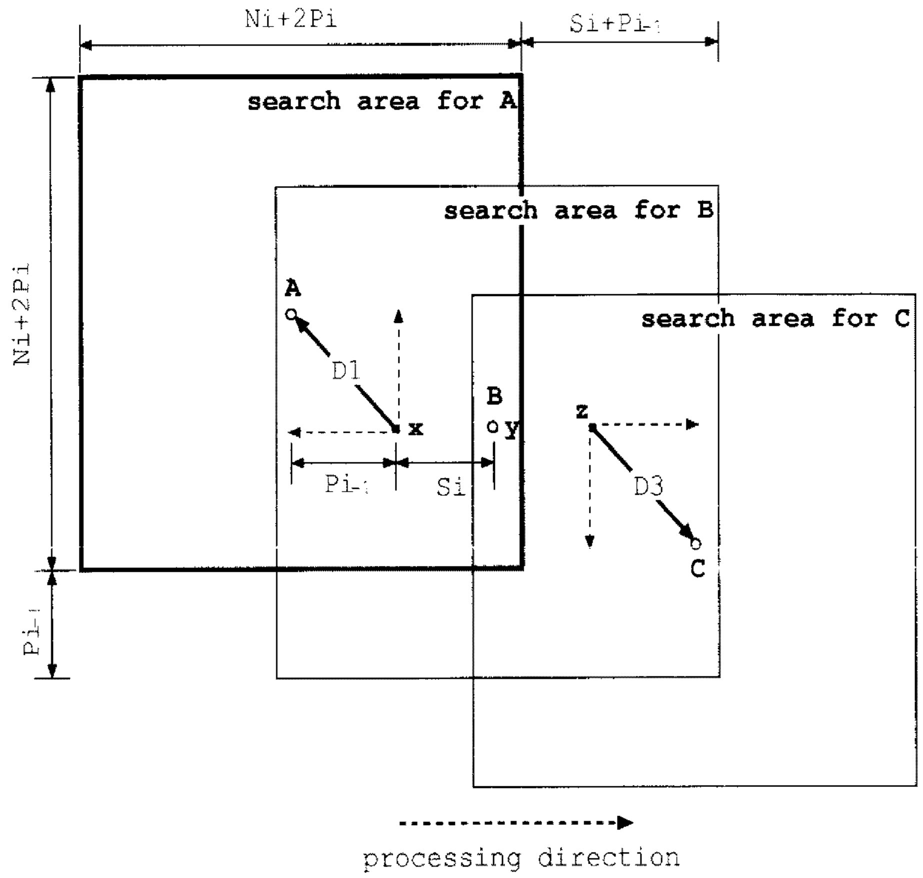
The size of a memory module can be gracefully reduced than explicit double buffering by the wraparound scheme. Let the size of a data block (a search area or a reference block) be $(m \times m)$ pixels. Let the size of a memory unit (PM or CM) be $(n \times n)$ bytes, where $n > m$. To get the n term, we should obtain $(n-m)$ term that is the possible maximum offset between two adjacent data blocks. Figure 14 shows three consecutive search areas for pixel locations of **x**, **y**, and **z**. After completion of layer $i-1$, **x** and **z** have the estimated vectors, **D1** and **D3**, respectively, which are incidentally completely opposite from each other. **y** has an interpolated vector from them that is a zero vector. The



(Fig. 13) Wraparound allocation scheme : (a) data block A is allocated already, (b) before the allocation of block B, and (c) after the wraparound allocation of block B locations of A, B, and C are displaced from **x**, **y**, and **z** by their intermediate vectors, respect-ively. Then, the offset term of $(n-m)$ is the offset between A and B. As shown in Figure 14, the horizontal offset term is $(p_{i-1}+s_i)$, where p_{i-1} is the possible maximum displacement of a motion vector at layer $i-1$ and s_i is the step size at layer i. The vertical offset term is also $(p_{i-1}+s_i)$ when the processing direction is ver-tical (as in case of C and D in Figure 12). The same result is obtained for the reference block. Therefore, a PM of size $(n_i+2p_i+s_i+p_{i-1})^2$ pixels is enough for two consecutive search areas at stage i. Similarly, the required size of CM for two consecutive reference blocks is $(n_i+s_i+p_{i-1})^2$ pixels, where the size of a ref-erence block is n_i^2 pixels.

The features of connection between the wraparound memory unit and the estimation unit are as follows.

- The number of the output ports of PM is $\alpha=(n_i+2p_i+s_i+p_{i-1})$, while the number of the input ports of the estimation unit for search area data is $\beta=(n_i+2p_i)$. That is, only β output ports of PM are enabled among α output ports at a time.
- Each enabled output port of PM is connected to a distinct input port of the estimation unit : *one-to-one connection*.
- An input port of the estimation unit is connected to an output port, $(h+\kappa)$ mod β, where h is a port id of PM and κ is



(Fig. 14) Possible maximum offset between two adjacent search areas at layer *i* the horizontal offset between the origins of the current search area and that of the next search area : *permutation*.

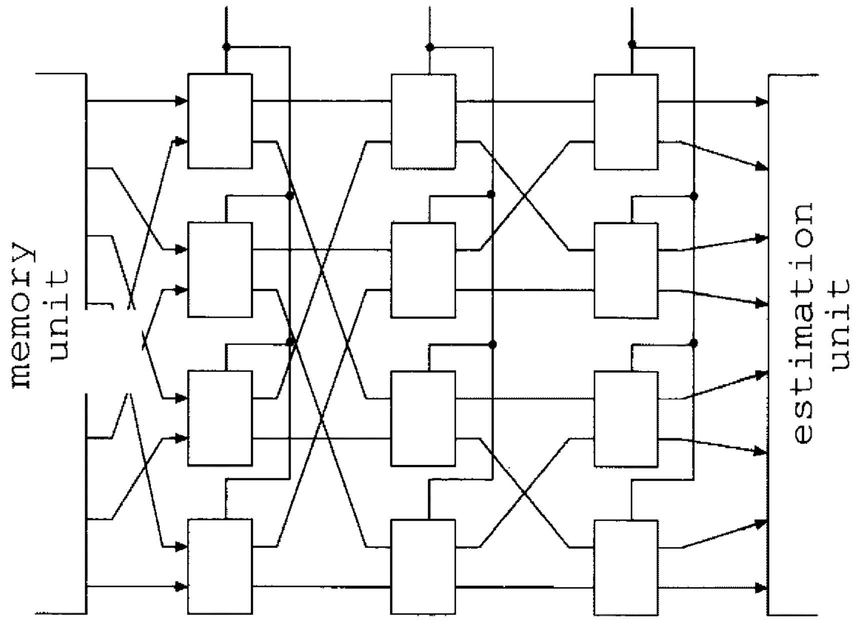
- The connection may be dynamically changed whenever the processing on a search area is completed.

Hence, the *memory-to-EU switch* should support permutation, one-to-one connection, the dynamic switching. Although the crossbar network and the Multistage Generalized Cube Network (MGCN) [19] satisfy these requirements, $a \times a$ MGCN is used in this paper for the sake of hardware cost. An $n \times n$ MGCN requires only $\frac{1}{2} n \log n$ switch elements. Figure 15 depicts a 8×8 modified MGCN. It is slightly different from the original MGCN in the respect of the control signal on every stage. The control signal arbitrates the switch elements in a stage for synchronous operation of the switch network in cooperating with the estimation unit.

Figure 16 illustrates the block diagram of a stage module of B-Architecture. There are two memory-to-EU switches of PM-switch and CM-switch for the paths of search area data and reference block data, respectively. PM-switch of stage i needs $\frac{1}{2} \lambda \log \lambda$ switch elements, where $\lambda = n_i + 2p_i + s_i + p_{i-1}$. For CM-switch, $\lambda = n_i + s_i + p_{i-1}$. The blocks of P and C are the control logics to arbitrate each stage of the memory-to-EU switches.

4.2. Interpolation Unit

Since the motion vectors are computed by accessing frame data in the bidirectional scan order, the flow of the intermediate vectors between stages should follow the order. However, unlike the unidirectional scan order, the results of bilinear interpolation build up a new line of the interpolated vectors between two lines of the estimated vectors from the fact of $s_{i+1} = \frac{1}{2} S_i$. The new line of the interpolated vectors would change the direction of existing flow of the estimated vectors as shown in Figure 17. Hence the interpolation unit should be designed to keep the whole flow of vectors being in the bidirectional scan order even though the new interpolated vectors are generated.



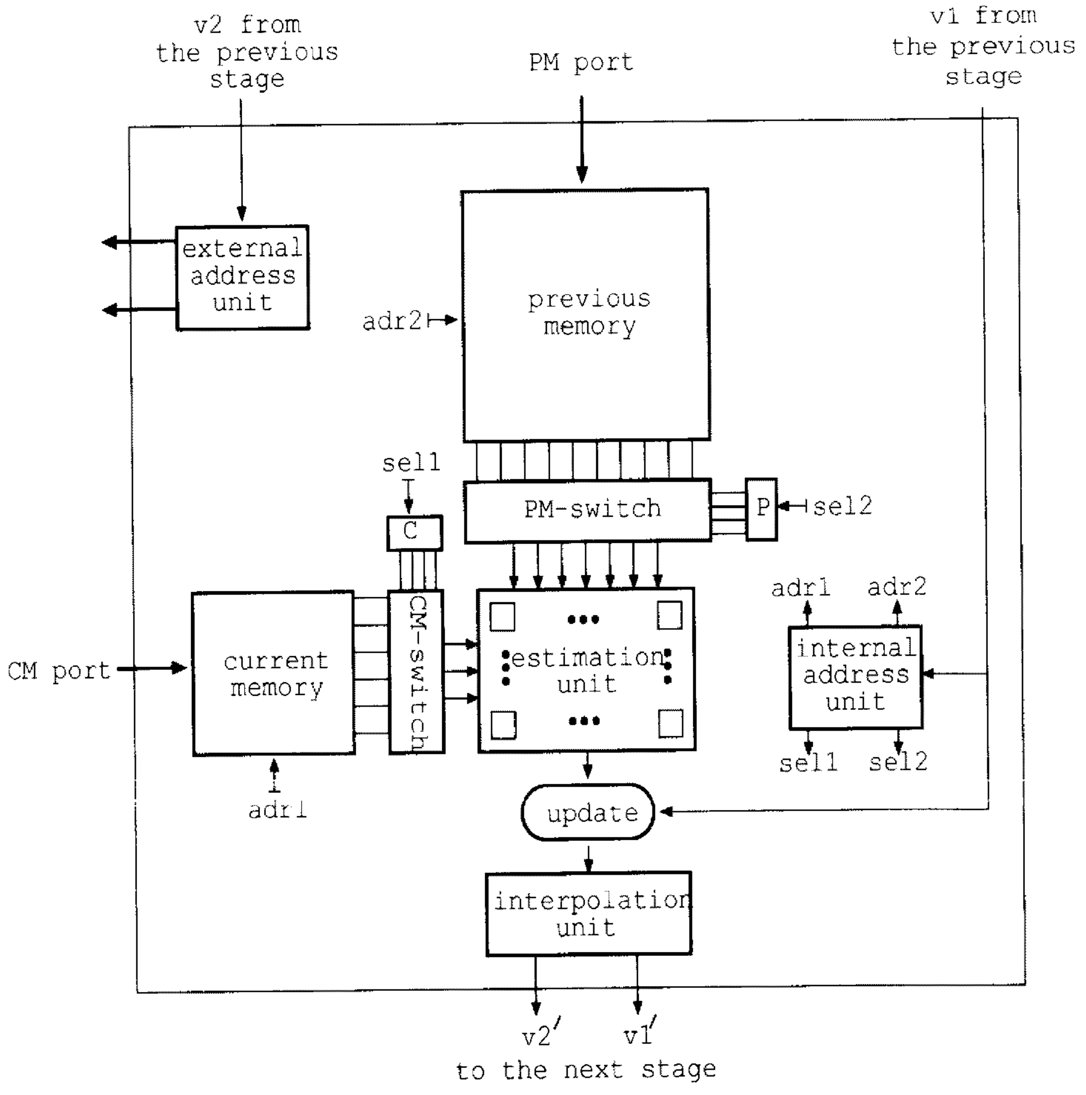
(Fig. 15) Memory-to-EU switch structure : 8×8 Modified MGCN

4.2.1. Bilinear Interpolator

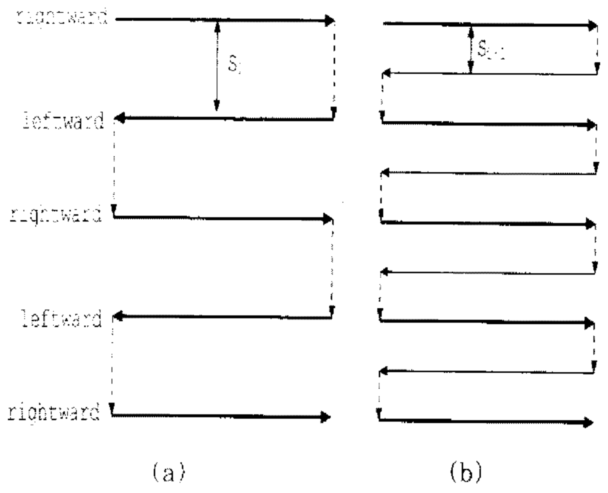
There are two cases in sequencing the output vectors depending on whether the newmost vec tor is on the rightward scan line or not. In case of the rightward line, $\hat{d}_b$ should be in pr -ior to $\hat{d}_c$, and d_2 should be in prior to $\hat{d}_a$ in accordance for the changed bidirectional scan order (Figure 18-(a)). In case of the leftward line, $\hat{d}_c$ should be in prior to $\hat{d}_b$, and $\hat{d}_a$ should be in prior to d_2 as shown in Figure 18-(b). These requirements on the output sequ ence are applied in design of the bilinear inte -rpolator of Figure 19. However, in case of the leftward line, we can see that the order of $\hat{d}_a$ and d_2 of Figure 19-(b) is conflict with the changed bidirectional scan order of Figure 18-(b). This matter is not simply cured by the bil -inear interplator and it will be done by the output latch mechanism.

4.2.2. Latch Mechanism

The functionality of the input latch is ident- ical to that of U-Architecture : it provides the



(Fig. 16) Block diagram of a stage module of B-Architecture



(Fig. 17) The order of the vector flow is changed by bilin-ear interpolation : (a) before bilinear inter-polation, and (b) after bilinear interpolation

correct estimated vectors to the bilinear interpolator at correct time. However, unlike U-Architecture, the direction of vector flow on a horizontal line is periodically reversed owing to the bidirectional scan order.. Since the estimated vectors are on two horizontal lines, two Last-In-First-Out (LIFO) structures are required as the input latch mechanism. The input latch mechanism has two sets of latches (Figure 20). Each group has a latch (RA or RB) and a LIFO stack (ISA or ISB) of size N_p/s_i-1 vectors, where N_p/s_i is the number of the estimated vectors on a line of a frame at layer i. The fuctions of the switches are shown in Table 3. The initial value of ctl is 0 and it is toggled whenever $\kappa \bmod (N_p/s_i)$ becomes 0, where κ is the accumulated number of vectors supplied by the estimation unit.

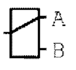
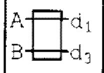
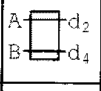
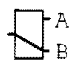
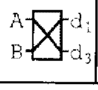
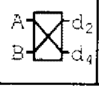
The output latches store the results of the bilinear interpolator and provide them to the next stage in line by line based on the bidirectional scan order. Figure 21 shows the output latch mechanism. The outputs of the bilinear interpolator are already arranged in the bidirectional scan order except only one case. That is, the bilinear interpolator supplies its outputs in the reverse bidirectional scan order only when it generates the outputs to the output port r1 during the newmost vector is on the leftward line as described in Section 4.2.1. Hence OQ1 should toggle its work from FIFO manner to LIFO manner, and vice versa. Preserving neither underflow nor overflow, the size of OQ1 is N_p/s_{i+1} vectors and the size of OQ2 is $N_p/s_{i+1}+1$ vectors.

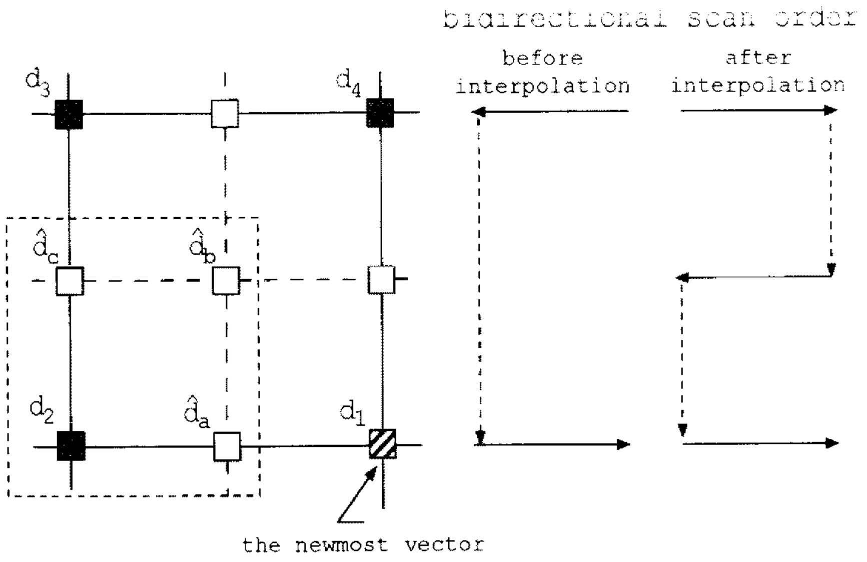
5. Performance Results

5.1. Real-Time Processing Aspect

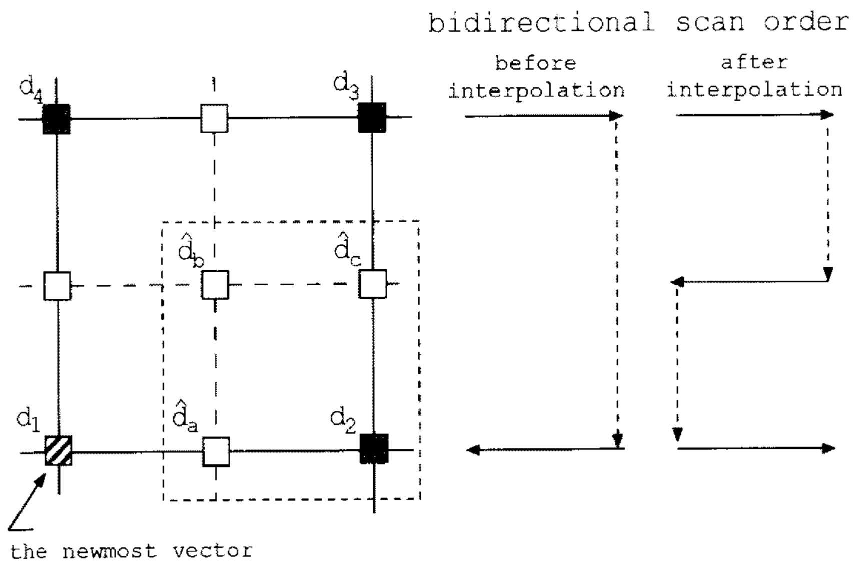
The major objective of this paper is to design a parallel VLSI architecture that can process HBMA in real-time. The feasibility of real-time processing by an architecture is analyzed by comparing the required cycle time to process a frame by the architecture with the given cycle time to a frame from the

〈Table 3〉 Function table for switches of the input latch mechanism

	ctl	S0	S1	S2	A ↕	B ↕
rightward	0				↓	↑
leftward	1				↑	↓



(a) the newmost vector on the rightward line



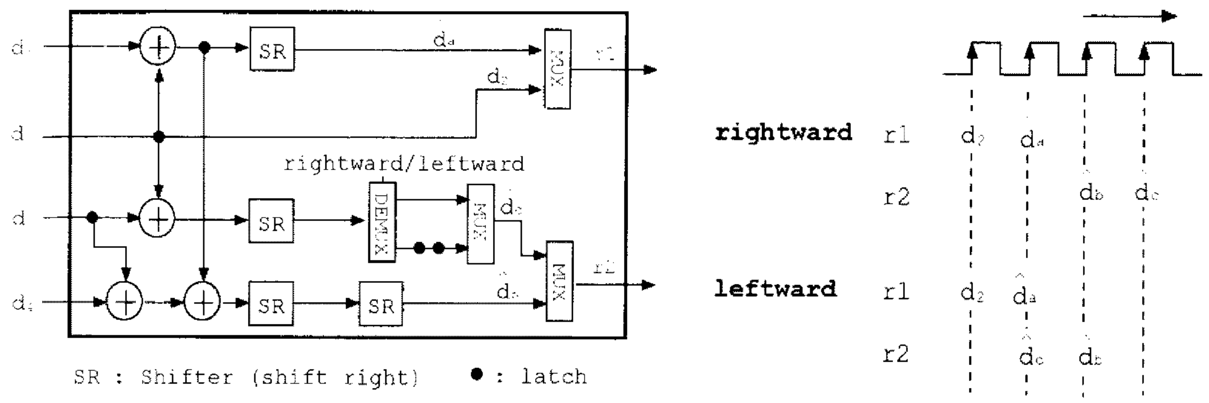
(b) the newmost vector on the leftward line

(Figure 18) Bilinear interpolation and the changed bidirectional scan order

real-time constraint.

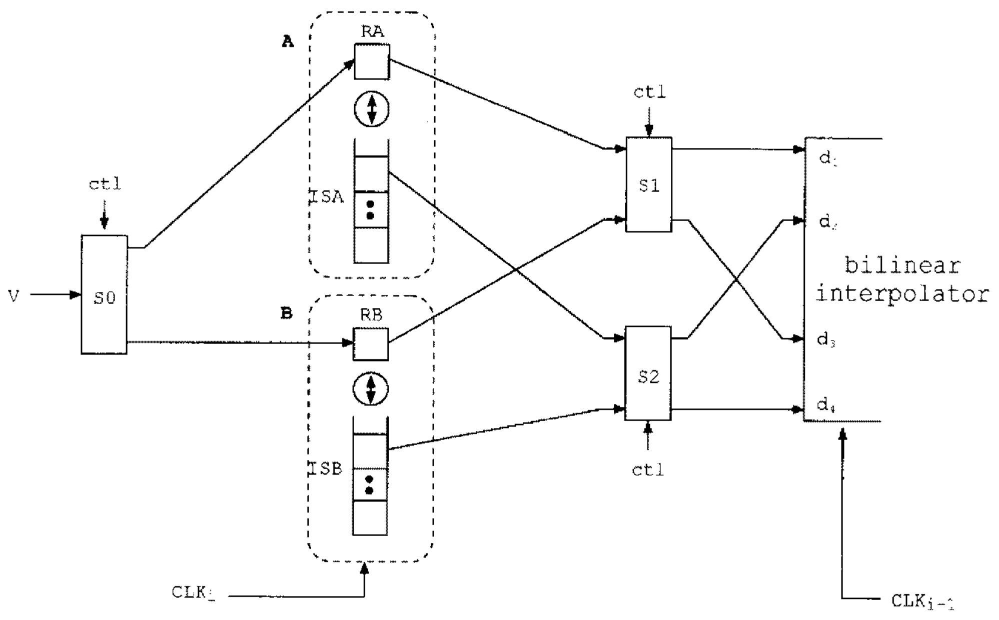
Let T_f be the required cycle time to process a frame of size $N_l \times N_p$ pixels by a pipelined architecture. To evaluate T_f, we first examine the critical path of the architecture. A stage begins its processing only after the first result of the previous stage would be available. Thus, the latency of stage i, F_i, is the sum of the fill-up delay of all of the input latches and the first activated one of two output latch arrays. The processing of a frame can be finished when the final result is generated from the last stage. Hence, we obtain

$$T_f = \sum_{i=1}^{3} F_i + G \qquad (1)$$



(a) block diagram of the bilinear interpolator (b) timing chart of the outputs

(Fig. 19) The bilinear interpolator of B-Architecture



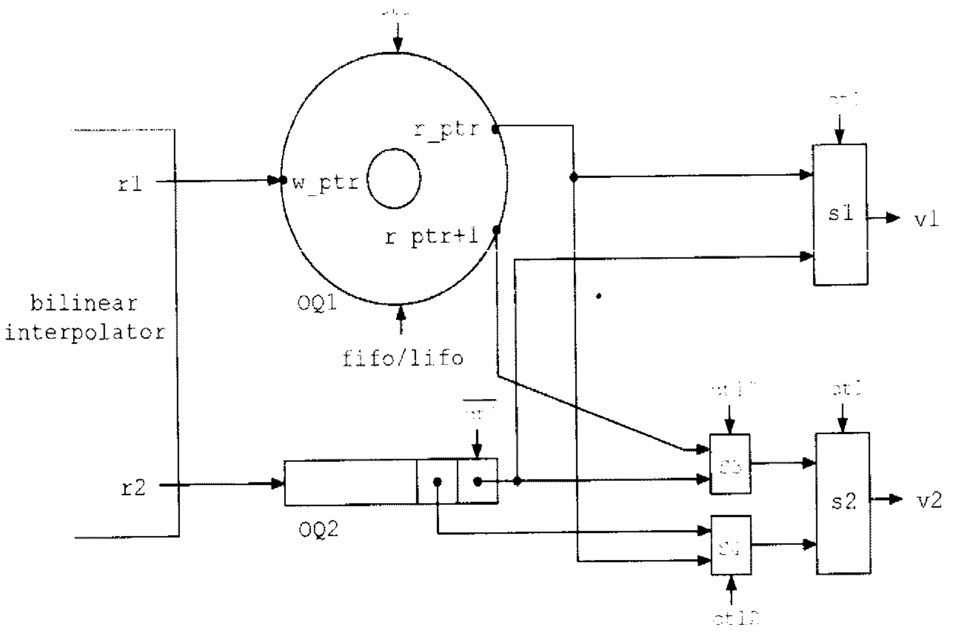
(Fig. 20) The input latch mechanism of stage i

, where G is the duration to generate all results from the last stage. F_i is $(N_p/s_i+2)t_i+2(N_p/2s_{i+1}+1)t_{i+1}$ and $(N_p/s_i+2)t_i+2(N_p/s_{i+1}+1)t_{i+1}$ and for U-Architecture and B-Architecture, respectively. G is $N_lN_pt_3$ for both architectures.

Let f_{rate} be the required frame rate for proper animation of motion video. Let k_{rate} be the frame skipping rate and it is given by 2 as the worst case. Then real-time constraint on T_f is given by

$$T_f \leq \frac{k_{rate}}{f_{rate}} \qquad (2)$$

By applying various video formats to Eq. (2), the required period t of the base clock is given by Table 4. Now the relation of $t \cong t_4/18$ is accomplished from the derivation of the clock subsystem, where t is the period of the base clock. The table shows both architect-



(Fig. 21) The output latch mechanism of stage i

ures can achieve real-time processing performance for up to the broadcast video format under the current VLSI technology. For HDTV format, the required clock speed is 2.7 ns that can be achieved VLSI technology about to be available [20].

5.2. Pipeline Processing Aspect

The speedup factor is evaluated for the pipeline processing performance. We assume a non-pipelined architecture that has only a single stage of our architecture and operates with the base clock of ours. The total cycle time of the non-pipelined architecture for a frame is given by $T_{non-pipe} = \sum_{i=1}^{3} E_i \cdot N_l N_p / s_i^2 \cdot t \cong 49 N_l N_p t$. Therefore, the speedup factor of the pipelined architecture over the non-pipelined architecture is given by

$$S_{pipe} = \frac{T_{non-pipe}}{T_f} \qquad (3)$$

Table 5 shows that both of ours achieve nearly linear speedup from the fact that the theoretical maximum speedup is k that a k-stage pipelined architecture can achieve.

5.3. VLSI Aspect

The hardware cost in terms of the pin count and the gate count for the proposed architectures is evaluated. The pin count is determined by the function of the required processing time of given data and the required transfer time of the data. The necessary amount of data (a sea

⟨Table 4⟩ Comparison of real-time processing performances

video format	$(N_l \times N_p)$	f_{rate} (Hz)	k_{rate}	U Arch (ns)	B-Arch (ns)
video conference	352x288	30	2	34.41	33.86
broadcast video	512x480	30	2	14.46	14.30
HDTV	1408x960	30	2	2.70	2.68

〈Table 5〉 Comparison of speedup factors

video format	U-Arch	B-Arch
video conference	2.56	2.52
broadcast video	2.61	2.58
HDTV	2.68	2.67

-rch area or a reference block) should be comp-letely transferred to the internal memory before it begins to process them in order to maximize the efficiency of pipeline and not to break the fully synchronous pipeline.

In analysis of the pin count, following assumptions are made : 1) the access time of the external memory and the access time of the internal memory are equal to each other, 2) only one pixel is transferred in a clock and thus the transfer time is equal to the amount of transferred data, and 3) the access times of the external memory and the internal memory are equal to the base clock period. These assumptions are reasonable since the access time of 3.5 ns is about to be commercially available [21].

Let κ_i and ρ_i be the transfer time of a search area and the transfer time of a reference block, respectively. In U-Architecture, even though the amount of data to be transferred could be smaller except at the start of a new block line, the worst case should be taken into account in determining the pin count. Thus $\kappa_i = (n_i+2p_i)^2$, and $\rho_i = n_i^2$ for U-Architecture since a whole data block should be transferred. For B-Architecture, only additional frame data are required to be transferred. Thus we obtain $\kappa_i = (s_i+p_{i-1})$ $(n_i+2p_i-p_{i-1})+p_{i-1})+p_{i-1}(n_i+2p_i)$ and $\rho_i = (s_i+p_{i-1})(n_i+p_{i-1})+p_{i-1}n_i$ at the worst case.

Then the pin count of input port for a search area at stage i, $\Im_i$, is given by

$$\Im_i = \left\lceil \frac{\kappa_i}{E_i} \right\rceil \times 8 \tag{4}$$

,where E_i is the clock cycles for computing a motion vector in Section 3.1 and 8 bits/pixel for a luminance signal.

Similarly, the pin count of input port for a reference block at stage i, $\Re_i$, is given by

$$\Re_i = \left\lceil \frac{\rho_i}{E_i} \right\rceil \times 8 \tag{5}$$

By applying the parameters of HBMA (Table 1) to Eq. (4) and Eq. (5), the required pin count for only data input is obtained as shown in Table 6. By comparing the pin counts for the data input ports, B-Architecture reduces about 50% of pins in compared to that of U-Architecture.

The evaluation of the gate count is based on the equivalent gate occupation of CMOS techno-logy (Table 7) [22]. Table 8 shows the gate count needed in each component of U-Architec-ture and Table 9 illustrates the result of B-Architecture. The gate counts of both architect-ures are compared in Table 10. For the on-chip memories, B-Architecture requires around 30% less than U-Architecture. However, this gain in memory size is achieved at the cost of the memory-to-EU switch. To sum up all comp-onents, B-Architecture requires about 10% less gates than U-Architecture. Even though the pr-oposed architectures are too complex to implem-ent with the current VLSI technology, it is exp ected that the complexity can be achieved near future.

6. Conclusion

This paper proposes a set of pipelined VLSI architectures to process HBMA in real-time. We look for the inherent data dependencies of HBMA : the interlayer data dependency and the intralayer data dependency. Based on two horizontal scan schemes, we investigate how the scan order could result in both the processing performance and the hardware complexity. Two specific three-stage architectures are designed according to the applied scan order. U-Architecture is based on the unidirectional scan order and B-Architecture is based on the bidirectional scan order. The scan order affects the designs of the bilinear interpolator, the latch mechanism, and the internal mémory.

The performance results show that both

〈Table 6〉 Comparison of the pin counts

stage		*i*	1	2	3	total
	port for search area		48	40	32	
U-Arch	port for reference block		32	32	24	
			82	72	56	208
	port for search area		8	24	16	
B-Arch	port for reference block		8	16	16	
			16	40	32	88

〈Table 7〉 Equivalent gate occupation for CMOS technology

Item (1-bit)	Equivalent gates
adder/subtractor	13
inverter	1
comparator	7
2×1 MUX	3
RAM	4
latch	5
up/down shifter	6

〈Table 8〉 Gate evaluation for U-Architecture

stage module *i*	1	2	3	subtotal
estimation unit	421,712	86,096	15,856	523,664
on-chip memory	651,520	124,160	21,760	797,440
latch Np=288	5,904	11,664	23,184	40,752
latch Np=480	9,744	19,344	38,544	67,632
latch Np=960	19,344	38,544	76,944	134,832
interpolator	768	768	768	2,304
memory-to-EU switch	3,408	1,488	624	5,520
total Np=288	1,083,312	224,176	62,192	1,369,680
total Np=480	1,087,152	231,856	77,552	1,396,560
total Np=960	1,096,752	251,056	115,952	1,463,760

〈Table 9〉 Gate evaluation for B-Architecture

stage module *i*	1	2	3	subtotal
estimation unit	421,712	86,096	15,856	523,664
on-chip memory	402,560	113,472	20,800	536,832
latch Np=288	9,032	14,792	29,192	53,016
latch Np=480	17,672	29,192	57,992	104,856
latch Np=960	34,862	57,992	115,592	208,446
interpolator	976	976	976	2,928
memory-to-EU switch	56,824	25,837	8,562	91,223
total Np=288	891,104	241,173	75,386	1,207,663
total Np=480	899,744	255,573	104,186	1,259,503
total Np=960	916,934	284,373	161,786	1,363,093

〈Table 10〉 Comparison of the gate counts

	U-Arch	B-Arch	B-Arch/U-Arch
estimation unit	523,664	523,664	1.0
on-chip memory	797,440	536,832	0.673
latch Np=288	40,752	53,016	1.301
latch Np=480	67,632	104,856	1.550
latch Np=960	134,832	208,446	1.546
interpolator	2,304	2,928	1.271
memory-to-EU switch	5,520	91,223	16.526
total Np=288	1,369,680	1,207,663	0.882
total Np=480	1,396,560	1,259,503	0.902
total Np=960	1,463,760	1,363,093	0.931

architectures achieve the real-time processing performance for the broadcast video format: U-Architecture required the clock speed of 14.46 ns and B-Architecture required the clock speed of 14.30 ns. These clock speeds are available under the current VLSI technology. For the HDTV video format, U-Architecture and B-Architecture required the clock speed that can be achieved with the near future VLSI technology. In the pipeline processing aspect, both architectures achieve nearly linear speedup over an assumed non-pipelined architecture. In the VLSI aspect, compared to U-Architecture, B-Architecture saves about 50% of pin count for the data input port due to the wraparound memory scheme at the cost of the memory-to-PE switch.

References

[1] A.K. Jain, "Image data compression : A review," in *Proceedings of the IEEE*, Vol. 69, No.3, pp.349-389, Mar. 1981.

[2] H.G. Musmann, P. Pirsch, and H.J. Grallert, "Advances in picture coding," in *Proceedings of the IEEE*, Vol.73, No.4, pp.523-

548, Apr. 1985.

[3] C. Cafforio, F. Rocca, and S. Tubaro, "Motion compensated image interpolation," *IEEE Trans. On Communications*, Vol.38, Feb. 1990.

[4] G. Bergeron and E. Dubois, "Gradient-based algorithms for block-oriented MAP estimation of motion and application to motion-compensated temporal interpolation," *IEEE Trans. On Circuits and systems for Video Technology*, Vol.1, pp.378-385, Mar. 1991.

[5] S.W. Wu and A. Gersho, "Joint estimation of forward and backward motion vectors for interpolative prediction of videl," *IEEE Trans. On Image Processing*, Vol.3, No.5, pp.684-687, 1994.

[6] R.J. Moorhead, S.A. Rajala, and L.W. Cook, "Image sequence compression using a pel-recursive motion-compensated technique," *IEEE Journal on Selected Areas in communications*, Vol.SAC-5, pp.1100-1114, Aug. 1987.

[7] A.N. Netravali and J.D. Robbins, "Motion-conpensated television coding : Part I," *Bell System Technical Journal*, Vol. 58, pp.632-670, Mar. 1979.

[8] W. Li and F. Dufaux, "Image sequence coding by multigrid motion estimation and segmentation-based coding of prediction errors," in *Visual Communications and Image Processing*, Vol.2094, pp.542-552, SPIE, 1993.

[9] G.T. Kim, H.C. Kim, S.R. Maeng, and J.W. Cho, "A motion estimation method based on pel-matching algorithm," *Journal of the Korea Information Science Society*, Vol.21, No.1, pp.94-103, 1994.

[10] R. Thoma and M. Bierling, "Motion compeating interpolation considering covered and uncovered background," *Signal Processing*, Vol.1, Oct. 1989.

[11] T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Trans. on Circuits and Systems*, Vol. 36, pp.1301-1308, Oct. 1989.

[12] Y.S. Jehng, L.G. Chen, and T.D. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms," *IEEE Trans. on Signal Processing*, Vol.4, No.2, pp.889-900, 1993.

[13] B.M. Wang, J.C. Yen, and S. Chang, "Zero waiting-cycle hierarchical block matching algorithm and its array architectures," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol.4, No.1, pp.18-28, 1994.

[14] T.P. Lin and C.H. Hsieh, "A moudlar and flexible architecture for real-time image template matching," *IEEE Trans. on Circuits and Systems-I : Fundamental Theory and Applications*, Vol.41, No.6, pp.457-461, 1994.

[15] H.M. Jong, L.G. Chen, and T.D. Chiueh, "Parallel architectures for 3-step hierarchical search block-matching algorithm," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol.4, No.4, pp.407-416, 1994.

[16] H.C. Kim, S.R. Maeng, and J.W. Cho, "A design of pipelined architecture for hierarchical block-matching algorithm," *IEICE Transactions on Information and Systems*, Vol.E78-D, pp.586-595, May 1995.

[17] G. Gupta and C. Chakrabarti, "VLSI architectures for hierarchical block matching," in *IEEE Int'l Symp. on Circuits and Systems*, pp.4.215-4.218, 1994.

[18] D.I. Moldovan, *Parallel Processing : From Applications to Systems*. San Mateo, CA, USA: Morgan Kaufmann Publishers, 1993.

[19] H.J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing*. McGraw-Hill, second ed., 1990.

[20] J.E. Smith and S. Weiss, "PowerPC 601

and Alpha 21064: A tale of two RISCs," *IEEE Computer Magazine*, Vol.27, pp.46-58, Jun. 1994.

[21] ----, *Cypress Semiconductor : High Performance Data Book*, 3901 North First St., San Jose, CA 95134: Cypress Semiconductor Co., Aug. 1993.

[22] ----, *1.0 Micron CMOS VDP370 Datapath Cell Library Rev 1.1 V8*R3, ASIC Division, 1109 McKay Drive, San Jose, CA 95131, USA: VLSI Technology, Inc., Apr. 1992.

김 형 철

1986년 인하대학교 전자계산학과 (학사)
1988년 한국과학기술원 전산학과 (석사)
1995년 한국과학기술원 전산학과 (박사)
1988년~현재 한국전자통신연구원 컴퓨터·소프트웨어 기술연구소 멀티미디어연구부 선임연구원
관심분야 : Parallel Architectures for Video Coding, Video Media Processing, Multimedia Group Communication

맹 승 렬

1977년 서울대학교 전자공학과(학사)
1979년 한국과학기술원 전산학과 (석사)
1984년 한국과학기술원 전산학과 (박사)
1984년~1989년 한국과학기술원 전산학과 조교수
1989년~1994년 한국과학기술원 전산학과 부교수
1994년~현재 한국과학기술원 전산학과 정교수
1990년~현재 한국과학기술원 인공지능연구센타 시스템 연구실장
1988년~1989년 Univ. of Pennsylvania 교환교수
1994년~1995년 Univ. of Texas 교환교수
관심분야 : Processor Architectures, Parallel Processing, Dataflow Machines, Vision Architectures, Microarchitectures and Microprogramming