Architectures for Hierarchical and other Block Matching Algorithms

Gagan Gupta and Chaitali Chakrabarti [1]

Department of Electrical Engineering
Telecommunications Research Center
Arizona State University
Tempe, AZ 85287-5706
Ph: (602) 965-9516
Fax: (602) 965-8325
chaitali@asu.edu

Abstract

Hierarchical block matching is an efficient motion estimation technique which provides an adaptation of the block size and the search area to the properties of the image. In this work, we propose two novel special-purpose architectures to implement hierarchical block matching for real-time applications. The first architecture is memory-efficient, but requires a large external memory bandwidth and a large number of processors. The second architecture requires significantly fewer processors, but additional on-chip memory. We describe in details the processor architecture, the memory organization and the scheduling for both these architectures. We also show how the second architecture can be modified to handle full-search and 3-step hierarchical search block matching algorithms, with significant reduction in the hardware complexity as compared to existing architectures.

[1] This work was supported in part by a grant from NSF MIP-9309504.

1 Introduction

In many applications of visual communication, moving image is transmitted over low transmission rate channels. Since the amount of image data to be transmitted is large, video compression techniques need to be employed. Video compression techniques reduce two types of redundancies in the image: (i) spatial and (ii) temporal. Spatial redundancy is the redundancy in a single frame, while temporal redundancy is the redundancy between successive frames. Spatial redundancy is exploited by a combination of differential pulse coded modulation (DPCM) and transform coding [1]. Temporal redundancy, on the other hand, is exploited by motion compensated prediction [2]. An important component in motion compensated prediction is estimating motion between successive frames [3]. In fact, in video compression coders, motion estimation takes up more than 50% of the computational needs of the entire compression process [4].

There are two main techniques for motion estimation [2]: pel recursive and block matching. Pel recursive algorithms estimate motion between successive frames on a pixel by pixel basis, whereas block matching algorithms (BMA) estimate motion on a block by block basis. In BMA, the image frame is divided into blocks (generally of size 8×8 or 16×16), and a motion vector is estimated for each block. BMAs are becoming more popular due to their relative computational simplicity, and have been adopted for motion estimation in the MPEG standards [5].

There are two classes of BMAs: fixed block size BMA [2], [3], and hierarchical BMA [6]. In a fixed block size BMA, a fixed size reference block is compared with candidate blocks in a fixed size search area in the previous frame. The matching criteria that is mostly used for finding the best match is the mean-of-the-absolute-differences (MAD) [3]. A full search technique or selective search technique is used to find the block with the minimum value of MAD in the search area. Fixed size BMAs give unreliable motion estimate for small block sizes and large displacements. They also fail to give accurate estimates if different parts within a block have motion in different directions. On the other hand, Hierarchical BMAs provide an adaptation of block size and search area to the properties of the image. In hierarchical BMAs, the search is carried out in a hierarchical fashion. The size of the block and the search area vary at different levels of hierarchy. The computation starts out with large blocks and large search areas, and in subsequent levels, the sizes are reduced. The search in each level is continued around the best fitting candidate block in the previous level. At each level of hierarchy, either a full search or selective search technique is employed. Bierling has proposed a hierarchical BMA that uses selective search scheme and has shown it to give more accurate motion estimates than fixed block size BMAs [6].

Block matching algorithms are computationally very demanding. This has necessitated the development of application-specific VLSI architectures for BMAs. A large number of architectures have been proposed for fixed size, full search BMA (FBMA) [7]-[9]. Most of these architectures use mesh-type systolic arrays to achieve high computation rates. Recently, Jehng et al. proposed a tree architecture for FBMA that is faster than the mesh-type systolic arrays [10]. Compared to FBMA, the hierarchical-search BMA (with fixed block size) as well as the hierarchical BMA (with variable block size) proposed by Bierling are computationally less intensive. However, both these algorithms require high memory bandwidth and irregular memory access. Architectures for hierarchical-search BMA with fixed block size have been proposed in [10] and [11]. Jehng's architecture [10] handles the high memory bandwidth by distributing the data in multiple memory modules such that multiple elements can be accessed simultaneously. To date, we are aware of only two architectures that have been proposed for hierarchical BMA [12], [13]. Both architectures use on-chip memory to reduce the memory bandwidth. Architectural details, such as, memory organization, scheduling, etc. have not been described in [12].

In this paper we present two architectures to implement the hierarchical BMA for real-time applications. This paper is an extension of our earlier work on architectures for the hierarchical BMA [6] that was presented in [14]. Here we propose architectures for hierarchical BMA for generalized parameters, that is, the parameters are not restricted to those in [6]. The proposed architectures differ in the number of processors and the size of on-chip memory. While Architecture 1 has no on-chip memory, and requires larger number of processors, Architecture 2 has a small on-chip memory and requires fewer processors. The on-chip memory reduces the large external memory bandwidth significantly. We describe in details the memory organization, scheduling and processor design for both the architectures. We propose a novel memory organization that supports simultaneous access of multiple pixels that are equidistant. For high speed operations, we propose a dynamic data routing network which routes data from the memory modules to the right processors. We also describe an architecture with on-chip memory for implementing full-search and hierarchical-search BMA. The hardware complexity of this architecture is significantly less compared to existing architectures.

The rest of the paper is organized as follows. In section 2 we describe the hierarchical BMA, and in section 3 we present Architectures 1 and 2 for implementing it. In section 4, we describe the architectures for full-search and hierarchical-search BMA.

2 Hierarchical Block Matching Algorithm

In hierarchical block matching (HBMA), the size of the block and the search area vary at different levels of hierarchy. At lower levels of hierarchy, larger block sizes are used in order to estimate motion of larger parts of the image, while at higher levels of hierarchy smaller block sizes are used in order to estimate motion of the smaller blocks. Fig. 1 shows the principle of hierarchical block matching for two levels of hierarchy. At the lowest level, a displacement estimate $\vec{x}$ is made for the block marked I. At the next level of hierarchy, the block size as well as the maximum possible displacement is reduced, and a displacement estimate $\vec{z}$ is made for the block marked II around the estimated displacement of block I. The displacement for the reference block is then the sum of the two displacements, $\vec{y} = \vec{x} + \vec{z}$.

2.1 Parameters

Table 1 gives the parameters of a simplified hierarchical block matching algorithm for two levels of hierarchy [6]. The parameters apply in both dimensions. The level 1 displacement is calculated

Parameter at level	1	2
Max. update displacement (d_l)	± 7	± 3
Measurement window size (W_l)	64	16
Step size (S_l)	32	16
Filter window size (F_l)	5	5
Subsampling (U_l)	8	2

Table 1: Parameters for a simplified HBMA [5]. The maximum displacement estimated is ± 10 pixels. d_l, W_l, S_l, F_l andU_l are parameters at level l, $l = 1, 2$.

for blocks corresponding to every $S_1 = 32^{nd}$ pixel in the image frame, and in each dimension, every $U_1 = 8^{th}$ pixel of the $W_1 \times W_1 = 64 \times 64$ block is used for match computation. The level 2 displacement is calculated for blocks corresponding to every $S_2 = 16^{th}$ pixel, and in each dimension, every $U_2 = 2^{nd}$ pixel of the $W_2 \times W_2 = 16 \times 16$ block is used for match computation. Similar parameters have been derived in [15]. The matching criteria that is used here is the mean-of-the-absolute-differences (MAD). Let $B_{m,n}$ be a block of size $W \times W$ whose top-left corner is located at (m, n) in the current frame X_t. Then for a subsampling of U

$$\mathrm{MAD}(i,j) = \sum_{x=0}^{\frac{W}{U}-1} \sum_{y=0}^{\frac{W}{U}-1} |X_t(m+Ux, n+Uy) - X_{t-T}(m+i+Ux, n+j+Uy)|, \qquad i,j = \{-d, 0, d\} \quad (1)$$

where X_{t-T} is the previous frame at time $t - T$, and d is the maximum possible displacement at that step. The (i, j) vector corresponding to the minimum MAD value for block $B_{m,n}$ is the estimated displacement vector or motion vector in that step. Since motion vectors are computed for blocks corresponding to every S_2 pixel, we say that the image is effectively divided into blocks of size $S_2 \times S_2$.

In Table 1, the number of operations [2] in a MAD computation is $\frac{W_l^2}{U_l^2}$=64, $l = 1, 2$. At each level of hierarchy, the $\log D$ step search technique (modified three step search) is used, where $D = max.$ $displacement + 1$ [16]. In level 1, the maximum possible displacement $d_1 = \pm 7$ pixels, and thus a $\log(7 + 1) = 3$ step search is carried out. In level 2, the maximum possible displacement $d_2 = \pm 3$ pixels, and a 2-step search is performed. In each step, the reference block is matched with 9 [3] candidate blocks in the search area. Thus $3 * 9 = 27$ MAD computations are carried out in level 1, and $2 * 9 = 18$ MAD computations are carried out in level 2. A low pass mean value filtering on $F_l \times F_l = 5 \times 5$ size block is carried out on the original image frames to improve the accuracy of the motion estimates [6].

Figure 2 shows part of the image frame for the case when $S_1 = 2S_2$. The level 1 displacement is estimated using MAD computations for blocks $B_{i,j}$, $B_{i,j+2S_2}$, $B_{i+2S_2,j}$, $B_{i+2S_2,j+2S_2}$, etc. and level 2 displacement is estimated for all the blocks. Since the level 1 displacement is required for the estimation of the level 2 displacement, the blocks which do not compute the level 1 displacement using MAD, estimate it using bilinear interpolation of the level 1 displacements of the four closest neighbors [6].

In the rest of the paper, we use the following set of parameters for blocks of size $S_2 \times S_2$ =8×8. The step size $S_1 = 16$, and the rest of the parameters, namely, d_l, W_l, F_l and U_l, are the same as that of Table 1. We use these parameters to calculate the hardware resources, memory bandwith, etc. for the proposed architectures.

2.2 Number of Processors

The minimum number of processors that would be required for real-time implementation of the two level hierarchical BMA is calculated as follows. Let the image size be $M \times N$. For level 1, $\frac{MN}{S_1^2}$ motion vector estimates are made, 27 blocks are matched per estimate, and $\frac{W_1{}^2}{U_1{}^2}$ operations are computed for every block match. For level 2, $\frac{MN}{S_2^2}$ estimates are made, 18 blocks are matched per

[2] Each operation consists of one subtraction, one absolute value computation, and one addition.

[3] For each reference block, 9 MAD computations are done, since in MAD(i, j), i and j can each take on 3 values.

estimate, and $\frac{{W_2}^2}{{U_2}^2}$ operations are computed for every block match. Thus for a frame frequency of f Hz, the number of operations that are carried out per second are $MNf(27(\frac{W_1}{S_1U_1})^2+18(\frac{W_2}{S_2U_2})^2)$. For the values in Table 1, the number of operations per second is $\frac{MNf}{16^2}(\frac{27}{4}+18)64 \ = MNf*6.1875$. For a picture format of 576×720, $f=25$ Hz. the operation rate is 64.2 million per second. In fact, the operation rate ranges from 6 to 1026 million operations per second for different standard picture formats. Clearly, multiple pipelined processors are required to meet such high processing requirements. If the processor cycle time is c, then the minimum number of processors that are needed for real-time computation is given by

$$p = \lceil 9MNfc(3(\frac{W_1}{S_1U_1})^2 + 2(\frac{W_2}{S_2U_2})^2)\rceil \tag{2}$$

3 Architectures for HBMA

In this section we present two architectures for two-level hierarchical BMA. We first describe the common architectural features such as processor configuration, memory organization, scheduling, etc, and then give the details of each architecture. We describe the architectures in terms of the general parameters (W_l, U_l, S_l, etc.) and then give details using the specific values from Table 1. We assume that the parameters in the two levels are related as follows:

1. $\frac{W_1}{U_1}=\frac{W_2}{U_2}=Q$, i.e. total number of MAD computations for blocks in levels 1 and 2 are the same.

2. U_1 and U_2 are either both odd or both even.

While the above assumptions are mildly constraining, they enable a very efficient architecture design. For instance, assumption 1 simplifies the scheduling, while assumption 2 enables efficient memory organization.

3.1 Processor Architecture

Let p be the minimum number of processors that are required for real-time computation. Then from equation 2, we have $p=\lceil 9MNfc(3(Q/S_1)^2+2(Q/S_2)^2)\rceil$. The time required to process a block is $n_c=\lceil Q^2/p\rceil$ cycles (assuming one operation is performed every cycle by each processor). If n_c is not an integer, then in the last cycle of processing, some processors will be idle. This implies that additional processors will be required to meet the computation rate. The number of processors is then chosen to be p', where p' is the smallest integral factor or multiple of Q that is greater than or equal to p, such that the computation rate is met.

Example: For the parameters in Table 1, if the image is of size 576×720, $f = 25$Hz, $c = 200$ns, then the minimum number of processors $p = 13$. Since every block match involves 64 computations, in the last cycle, one processor will be idle. The number of processors is chosen to be $p' = 16$, since p' is a multiple of $Q = 8$ and the computation rate is met. $\square$

The proposed architecture consists of p' processors which are interconnected by a tree structure as shown in Fig. 3. The tree architecture is adopted to achieve small latency and fast computation times [10]. The p' processors of type P_D form the leaves of the tree. These processors compute the absolute differences between pixels of the reference block (X) and the pixels of the candidate block (Y). The differences are added using a tree of adders. Processing element P_M keeps track of the minimum value of MAD. Each level of the tree can be considered as a pipeline stage and the latency is $\lceil \log p' \rceil + 3$. Fig. 3 shows the processor architecture for the case when $p' = 8$. Note that the tree of processors can be replaced by a systolic array at the expense of an increase in the latency and consequently, an increase in the pipeline refill time.

3.2 Memory Organization

The memory organization has to be such that multiple processors can access data simultaneously from the image memories. Recall from Section 2, that for a level 1 search every U_1^{th} pixel of a $W_1 \times W_1$ block is used, and for a level 2 search, every U_2^{th} pixel of a $W_2 \times W_2$ block is used. The memory organization schemes suggested in [10], [17]-[18] cannot support such data accesses. Our aim here is to design a new memory organization that supports simultaneous access of p' pixels that are U_1 apart for level 1 search and are U_2 apart for level 2 search.

The proposed memory organization consists of a set of memory banks, where each memory bank consists of a set of memory modules. If $p' \leq Q$, only one memory bank is required, and the bank consists of e memory modules, where e is the smallest integer greater than or equal to p' such that it is not a multiple or a factor of U_1 and U_2. If $p' > Q$, r memory banks are required, where r is smallest integer greater than or equal to $\frac{p'}{Q}$ such that r is not a multiple or factor of U_1 and U_2. Moreover, for $p' > Q$, each memory bank consists of e memory modules, where e is smallest integer greater than or equal to Q such that e is not a multiple or factor of U_1 and U_2.

Example: For the parameter values given in Table 1, $U_1 = 8$, $U_2 = 2$, and $Q = 8$. It can be shown that for $p' \leq 8$, only 1 bank with $p' + 1$ modules are needed. This is because p' is restricted to the set 1, 2, 4, or 8, and since $U_1 = 8$ and $U_2 = 2$, the smallest value of e that is not a multiple or a factor of U_1 and U_2 is $e = p' + 1$. Similarly, it can be shown that for $p' \geq 8$, $r = 1 + \frac{p'}{8}$ and there are 9 modules per bank. Figure 4 shows the memory organization for the case when $p' = 16$ ($r = 3, e = 9$). The module width is the same as the pixel width. $\square$

The image frames are distributed among the memory banks and modules in the following way. Pixel $I_{i,j}$ is allocated to memory bank b, and within bank b to memory module k. α is the address of pixel $I_{i,j}$ in module k.

$$\left.\begin{array}{l} b = i \bmod r \\ k = j \bmod e \\ \alpha = \lceil \frac{N}{e} \rceil * \lfloor \frac{i}{r} \rfloor + \lfloor \frac{j}{e} \rfloor \end{array}\right\} \text{ for } 0 \leq i \leq M-1,\ 0 \leq j \leq N-1 \tag{3}$$

The above distribution is such that each memory bank contains an entire row of pixels (i.e. N pixels) in the image frame, and each line is distributed in the cyclic mode among the memory modules. In Appendix 1 we prove that the distribution function of equation (3) supports simultaneous access of p' elements which are U_1 apart for level 1 search and U_2 apart for level 2 search.

3.3 Interpolator

Bilinear interpolation is required for all blocks for which level 1 displacements are not estimated by MAD computations. In particular, it is required for all blocks $\{B_{i,j}\}$ such that ($i \bmod S_2 = 0$ & $j \bmod S_2 = 0$) but ($i \bmod S_1 \neq 0$ & $j \bmod S_1 \neq 0$), for $0 \leq i \leq M-1, 0 \leq j \leq N-1$. In Figure 2 where $S_1 = 2S_2$, bilinear interpolation is required for blocks $B_{i,j+S_2}$, $B_{i,j+3S_2}$, $B_{i+S_2,j}$, $B_{i+S_2,j+S_2}$, $B_{i+S_2,j+2S_2}$, etc. If $E(\mathrm{B}_{i,j})$ is the level 1 displacement of $\mathrm{B}_{i,j}$, then the level 1 displacements of $B_{i+2S_2,j+S_2}$, $B_{i+S_2,j+S_2}$, and $B_{i+S_2,j+2S_2}$ are calculated as follows.

$$\begin{aligned} E(\mathrm{B}_{i+2S_2,j+S_2}) &= \frac{E(\mathrm{B}_{i+2S_2,j}) + E(\mathrm{B}_{i+2S_2,j+2S_2})}{2} \\ E(\mathrm{B}_{i+S_2,j+S_2}) &= \frac{E(\mathrm{B}_{i+2S_2,j+S_2})}{2} + \frac{E(\mathrm{B}_{i,j}) + E(\mathrm{B}_{i,j+2S_2})}{4} \\ E(\mathrm{B}_{i+S_2,j+2S_2}) &= \frac{E(\mathrm{B}_{i,j+2S_2}) + E(\mathrm{B}_{i+2S_2,j+2S_2})}{2} \end{aligned}$$

The bilinear interpolator consists of $\frac{N}{2S_2} + 2$ shift registers to hold the level 1 displacement vectors and four add-shift units as shown in Figure 5. Each add-shift unit adds two displacement vectors and right shifts the sum vector by 1 bit (equivalent to division by 2). Since the maximum estimated displacement for each vector component is ± 10 pixels, each displacement vector is housed in two 5-bit registers. Although the above scheme is for a two level HBMA, a similar scheme can be designed for 3 or more level HBMA. Recently, an interpolator has been proposed for 3 level HBMA in [19].

3.4 Scheduling

Hierarchical BMA has two types of computational dependencies: inter-level and intra-level. Inter-level dependency is the dependency between computations in different levels of search, while intralevel dependency is the dependency between computations in the same level.

Inter-level Dependency : For all the blocks, computations of the level 2 displacement cannot begin unless the level 1 displacements are available. For some blocks, the level 1 displacement is not computed by MAD, and the level 2 displacement computations have to wait until bilinear interpolation is done on the level 1 displacements of the neighboring blocks.

Intra-level Dependency : At each level of search, search for the next step is made around the estimated displacement of the previous step. For instance, for level 1 search, step 3 search cannot proceed before result from step 2 search is available, and similarly, search of step 2 cannot proceed before result from step 1 is available.

The scheduling schemes of both Architectures 1 and 2 employ interleaving to avoid the performance degradation due to these dependencies.

3.5 Architecture 1

Architecture 1 has an on-chip module which consists of a set of processors, an address generating circuit and an interpolator, and an off-chip module which consists of two image memories to store the current frame and the previous frame. The processors are organized in a tree structure as described in Section 3.1. The off-chip memories are organized into memory banks and memory modules as described in Section 3.2. The off-chip memory is the slowest unit, and hence the system operates at the rate of off-chip memory access times. For bit parallel operation, if b bits are used to represent every pixel, and p' pixels are accessed simultaneously, the processor chip pin count requirement ($> bp'$) will be very high. Since the memory access time is fairly high, accesses to modules within a bank can be pipelined and interleaved to reduce the pin count as in [10]. Also the large memory access time and the interleaved accesses can be used to route a pixel to the 'right' processor. Routing is required since the same pixel is required by different leaf processors at different instances of time while processing different blocks. Figure 6 shows the block diagram of this architecture for the case when $p' \leq Q$. The image memories contain data that has been filtered by a 5×5 low pass filter. The computation schedule for the case when $S_1 = 2S_2$ (see figure 2) is given in Table 2.

The computation of multiple blocks are interleaved to avoid pipeline stalls that would otherwise be caused by the inter-level and intra-level dependencies. For instance (see Table 2), after the level 1-step 1 search of block $\mathrm{B}_{i+2S_2,j+2S_2}$, instead of waiting for the search results to become available (this is equivalent to emptying the pipeline in the tree of processors) and then proceeding with the level 1-step 2 search of $\mathrm{B}_{i+2S_2,j+2S_2}$, the level 1-step 1 search of the next block, $\mathrm{B}_{i+2S_2,j+4S_2}$ is computed. After the level 1 vectors for blocks $\mathrm{B}_{i+2S_2,j+2S_2}$ and $\mathrm{B}_{i+2S_2,j+4S_2}$ are available, the level 2 search for all adjacent blocks are scheduled. The computations for the two steps in level 2 search can once again be interleaved among these blocks to avoid any pipeline stalls.

Time	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$
Block	$\mathrm{B}_{i+2S_2,j+2S_2}$	$\mathrm{B}_{i+2S_2,j+4S_2}$	$\mathrm{B}_{i+2S_2,j+2S_2}$	$\mathrm{B}_{i+2S_2,j+4S_2}$	$\mathrm{B}_{i+2S_2,j+2S_2}$	$\mathrm{B}_{i+2S_2,j+4S_2}$
Level	1	1	1	1	1	1
Step	1	1	2	2	3	3
Time	$t+6$	$t+7$	$t+8$	$t+9$	$t+10$	$t+11$
Block	$\mathrm{B}_{i+2S_2,j+2S_2}$	$\mathrm{B}_{i+2S_2,j+S_2}$	$\mathrm{B}_{i+S_2,j+S_2}$	$\mathrm{B}_{i+S_2,j+2S_2}$	$\mathrm{B}_{i+2S_2,j+2S_2}$	$\mathrm{B}_{i+2S_2,j+S_2}$
Level	2	2	2	2	2	2
Step	1	1	1	1	2	2
Time	$t+12$	$t+13$	$t+14$	$t+15$	$t+16$	$t+17$
Block	$\mathrm{B}_{i+S_2,j+S_2}$	$\mathrm{B}_{i+S_2,j+2S_2}$	$\mathrm{B}_{i+2S_2,j+4S_2}$	$\mathrm{B}_{i+2S_2,j+3S_2}$	$\mathrm{B}_{i+S_2,j+3S_2}$	$\mathrm{B}_{i+S_2,j+4S_2}$
Level	2	2	2	2	2	2
Step	2	2	1	1	1	1
Time	$t+18$	$t+19$	$t+20$	$t+21$	$t+22$	$t+23$
Block	$B_{i+2S_2,j+4S_2}$	$B_{i+2S_2,j+3S_2}$	$B_{i+S_2,j+3S_2}$	$B_{i+S_2,j+4S_2}$	$B_{i+2S_2,j+6S_2}$	$B_{i+2S_2,j+8S_2}$
Level	2	2	2	2	1	1
Step	2	2	2	2	1	1

Table 2: Time schedule at which level 1 and level 2 displacements for the blocks get computed in Architecture 1 when $S_1 = 2S_2$. One time unit corresponds to the time taken to carry out 9 block matches.

Table 3 gives the minimum number of processors and memory modules required to implement the two level HBMA for the following standard video picture formats.

1. 288×352, 10 Hz. — video telephone format.

2. 288×352, 30 Hz. — teleconferencing video format.

3. 576×720, 25 Hz. — broadcast video format.

The parameter values have been described in Section 2. The clock cycle time of the processors is dictated by the external memory access time. Here we have assumed a conservative memory access time of 200 ns. as in [10]. The table also compares the resource requirements for HBMA (Architecture 1) with those of FBMA and 3HSA (architecture of [10]). Note that for a block of size $S_2 \times S_2 = 16 \times 16$, hierarchical BMA requires significantly fewer resources compared to both FBMA and 3HSA for all the three picture formats.

Picture format	Block size $S_2 \times S_2$	FBMA [10] # of procs	FBMA [10] # of mem modules prev+curr	3HSA [10] # of procs	3HSA [10] #of mem modules prev+curr	HBMA (Arch. 1) # of procs	HBMA (Arch. 1) # of mem modules prev+curr
288×352, 10 Hz.	8×8	64	64+64	8	8+8	8	9+9
288×352, 10 Hz.	16×16	64	64+64	8	8+8	2	3+3
288×352, 30 Hz.	8×8	256	256+256	16	16+16	16	27+27
288×352, 30 Hz.	16×16	256	256+256	16	16+16	4	5+5
576×720, 25 Hz.	8×8	512	512+512	64	64+64	64	81+81
576×720, 25 Hz.	16×16	512	512+512	64	64+64	16	27+27

Table 3: The number of processors and the number of memory modules that are required for FBMA, 3HSA (by Jehng's architecture) and hierarchical BMA (by Architecture 1).

3.6 Architecture 2

Picture format	Block size $S_2 \times S_2$	FBMA without	FBMA with	3HSA without	3HSA with	HBMA without	HBMA with
288×352, 10 Hz.	8×8	228.1	2.8	27.4	2.8	25.1	4.5
288×352, 10 Hz.	16×16	228.1	1.9	27.4	1.9	6.3	2.3
288×352, 30 Hz.	8×8	684.3	8.4	82.1	8.4	75.3	13.5
288×352, 30 Hz.	16×16	684.3	5.7	82.1	5.7	18.8	6.8
576×720, 25 Hz.	8×8	2332.8	28.5	279.9	28.5	256.6	46
576×720, 25 Hz.	16×16	2332.8	19.4	279.9	19.4	64.2	23

Table 4: Memory bandwidth requirement in Mbytes/s for the three algorithms without and with on-chip memory.

One major disadvantage of Architecture 1 is that it has very high memory bandwidth requirements (see Table 4). This is because the same pixels are used for multiple computations, and are consequently accessed as many times from the external memory. In this section we propose an architecture that on the one hand, reduces the external memory bandwidth, and on the other hand, requires fewer processors. This is achieved by storing the search area on-chip in a local memory, and by operating the processors at the faster local memory access rate. Since the processors are essentially word-level subtractors, they can be easily clocked at local memory access times of (say) 70ns.

As in Architecture 1, the computation unit of Architecture 2 consists of a tree of processors working in parallel. To support simultaneous data accesses, the on-chip memory is distributed among multiple memory modules. Moreover, since the same pixel is accessed by different leaf processors at different times (for different computations), a **data shuffler** unit is required to reorder the p' data elements accessed from the local memory, before they are forwarded to the leaf processors. The local memory, the data shuffler, the processor tree, and the address generator units may be housed on the same chip. Each of these units are pipelined so that they can be clocked at the local memory access rate. Fig. 7 shows the proposed system architecture.

3.6.1 Processor Architecture

The computation unit consists of a tree with p' leaf processor as described in Section 3.1. The processors as well as the adders are clocked at the rate of on-chip memory access times. As a result, the number of processors in Architecture 2 is significantly smaller than that of Architecture 1 (where the processors are clocked at the rate of off-chip memory access time).

3.6.2 Memory Organization

The memory system consists of two external image memories to store the current and previous image frames. In addition, it consists of two on-chip memories to store the data that would be otherwise accessed multiple times from the external memories.

On-chip Memory Organization

On-Chip Memory 1: On-chip Memory 1 is used to store the search area from the previous image frame. For a block whose top leftmost pixel is $I_{i,j}$, the level l search area includes pixels

$$\{I_{i+m,j+n}\} \quad \text{where} \quad 0 \le m,n \le W_l - U_l + 2\sum_{k=1}^{l} d_k$$

and d_l, U_l, W_l are the parameters for level l. Thus the search area at level l is of size

$$(W_l + 1 - U_l + 2\sum_{k=1}^{l} d_k)^2$$

In HBMA, not only does the level 1 and level 2 search areas of a block overlap, the search areas of neighboring blocks also overlap. This fact can be exploited to reduce the memory bandwidth. The nonoverlapping part of the search area for the next block can be loaded while the current block is being matched. The amount of overlap between search areas of neighboring blocks at level l is

$$(W_l + 1 - U_l + 2\sum_{k=1}^{l} d_k - S_l) * (W_l + 1 - U_l + 2\sum_{k=1}^{l} d_k).$$

The exact amount of overlap between the level 1 and level 2 search areas as well as the overlap between the search areas of neighboring blocks, depends on the actual values of the various parameters.

Example : For the parameter values in Section 2, the on-chip memories store the data that is required to process a group of 4 neighboring blocks. In each group, level 1 and level 2 displacements are required for one block, and level 2 displacements are required for the remaining three blocks. For instance, in Figure 2, blocks $B_{i+2S_2,j+2S_2}$, $B_{i+2S_2,j+S_2}$, $B_{i+S_2,j+S_2}$, and $B_{i+S_2,j+2S_2}$ form a group, where the level 1 and level 2 displacements are required for $B_{i+2S_2,j+2S_2}$, and only level 2 displacements are required for $B_{i+2S_2,j+S_2}$, $B_{i+S_2,j+S_2}$, and $B_{i+S_2,j+2S_2}$. For the parameters given in Table 1, the search area for the level 1 displacement of a block includes the search areas for the 3 other blocks in the same group. We justify this statement by studying the following group of blocks $B_{i+32,j+32}$, $B_{i+32,j+16}$, $B_{i+16,j+16}$, and $B_{i+16,j+32}$. The search area for $B_{i+32,j+32}$ is given by pixels

$$\mathrm{B}_{i+32,j+32}: \ \{I_{i+m+9,j+n+9}\}, \ \ 0 \le m,n < 71$$

while the search areas for $\mathrm{B}_{i+32,j+16}$, $\mathrm{B}_{i+16,j+16}$ and $\mathrm{B}_{i+16,j+32}$ are given by pixels

$$\left.\begin{array}{ll} \mathrm{B}_{i+32,j+16}: & \{I_{i+m+30,j+n+14}\} \\ \mathrm{B}_{i+16,j+16}: & \{I_{i+m+14,j+n+14}\} \\ \mathrm{B}_{i+16,j+32}: & \{I_{i+m+14,j+n+30}\} \end{array}\right\} 0 \le m,n < 35$$

Thus the search areas for $\mathrm{B}_{i+16,j+16}$, $\mathrm{B}_{i+16,j+32}$ and $\mathrm{B}_{i+32,j+16}$ are completely encampassed within the search area for $\mathrm{B}_{i+32,j+32}$. Hence it is sufficient to load only the search area for $B_{i+32,j+32}$. The size of the search area for one group of blocks is $71 * 71$. The search areas for two consecutive groups of blocks overlap by $(71 - 32) * 71$ pixels. Thus only $32 * 71$ pixels from the search area of the next group have to be loaded during the computation of the current group. The total size of Memory 1 is thus $71 * 71 + 32 * 71 = 7313$ pixels. □

The organization of on-chip Memory 1 is the same as that described in section 3.2. The search area is distributed among memory banks and modules as per the distribution function of equation (3). The only difference is that for a search area of size (say) $K \times L$, the parameters M and N in equation 3 have to be replaced by K and L. Since the system now operates at a much faster clock rate, it may not be possible to pipeline on-chip memory accesses within memory banks. Hence an address generating circuit is required for every on-chip memory module.

On-Chip Memory 2 : On-chip Memory 2 is used to store the reference blocks which are being matched. For the parameter values in Section 2, Memory 2 stores the five reference blocks (of the current frame) corresponding to a group. These 5 blocks are the blocks used for calculating level 1 and level 2 displacements of one block, and the level 2 displacements for the remaining 3 blocks in a group. Thus the size of on-chip Memory 2 is $5Q^2$ bytes, since Q^2 pixels need to be stored per block. While the displacements of one group are being computed, the five blocks of the next group are loaded. For instance (see Figure 2), the level 1 block for $B_{i+4S_2,j+4S_2}$ is loaded only after level 1 displacement for $B_{i+2S_2,j+2S_2}$ is computed, the level 2 block for $B_{i+4S_2,j+3S_2}$ is loaded only after level 2 displacement for $B_{i+2S_2,j+S_2}$ is computed, and so on. If we define the time that is required to carry out one step search of a hierarchy level (9 block matches) as one time unit, then it takes a total of 11 time units to estimate the vectors of a group of 4 blocks. The 5 blocks for the next group are loaded in 10 (out of 11) time units.

Since the number of processors is p', on-chip Memory 2 consists of p' memory modules. The data can be distributed in the cyclic or in the consecutive mode. Since there is an overlap in the successive reference blocks, the size of on-chip Memory 2 can also be reduced if the common pixels are stored only once.

External Memory Organization

Previous Frame: If an image is made up of $S_2 \times S_2$ blocks, then $\frac{MNf}{S_2^2}$ vectors are computed every second. While one block is being matched, if A is the search area size to be loaded for the next block, and if m is the external memory access time, then the number of memory modules required to meet the bandwidth is: $\lceil \frac{mMNf*A}{S_2^2} \rceil$. For the parameters in Table 1, $32 * 71$ pixels need to be accessed during the computation of 4 blocks, and thus $A = 8*71$. Therefore the number of external memory modules is $\lceil \frac{mMNf*71}{32} \rceil$. Table 4 shows how the external memory bandwidth reduces after on-chip memory is added to the system.

The image data needs to be distributed in the modules so as to facilitate multiple accesses. In fact, the distribution function (eqn. 3) of section 3.2 can be used here too. To reduce the number of memory ports, image memory modules can be grouped into banks and accesses can be pipelined as in [10].

Current Frame: Earlier we described how for the parameters in Section 2, 5 blocks are fetched from the current frame during 10 time units (each time unit = 9 block matches). Since each block consists of Q^2 pixels, $5 * Q^2$ pixels need to be accessed in $\frac{10}{11} * \frac{S_2^2}{4MNf}$ time units. Therefore the time to fetch one pixel is $\Delta T = \frac{10}{11} * \frac{S_2^2}{4MNf} * \frac{1}{5Q^2}$. The number of external memory modules needed to store the current frame is then $\lceil \frac{m}{\Delta T} \rceil = \lceil \frac{mMNf*Q^2*11}{8{S_2}^2} \rceil$.

3.6.3 Data Shuffler

The mapping between the memory modules of on-chip Memory 1 and the leaf processors is different at different time instants. This is because the same pixel occupies different (relative) positions in different candidate blocks. Thus while a pixel is accessed from the same memory module every time, it has to be routed to different leaf processors at different times. In Architecture 1, the routing was achieved by reordering the pipelined accesses. Such a method cannot be applied to Architecture 2 since it operates at a higher speed. Note that the reference block pixels always occupy the same position, and hence once brought into on-chip Memory 2, do not require any subsequent reordering. In this section we describe a dynamic routing network called the data shuffler which parallely routes p' pixels from the memory modules to the right leaf procesors. While the routing required in Architecture 2 could be implemented by a general purpose interconnection network such as the Omega network, we choose to implement it using the specialized data shuffler unit. This is because the characteristics of the memory organization can be exploited to design a router that is simpler and more area efficient than the general purpose networks.

In the memory organization described in section 3.2, if $p' \leq Q$, only 1 bank of e memory modules is required, and each row of the block being matched is distributed among modules. The data shuffler then reorders p' (out of e) elements that are accessed in parallel from the memory modules. If $p' \geq Q$, there is more than 1 bank ($r > 1$) and each bank contains an entire row of the block being matched. In this case, $\frac{p'}{Q}$ banks are connected to the processor tree at any given time, and the data shuffler reorders $\frac{p'}{Q}$ (out of r) bank level accesses as well. In any cycle, the memory modules and the banks are accessed in an order that is a function of e, r and U. Moreover, the order of accesses of different blocks is closely related. Specifically, the order of accesses during the computation of $B_{i,j}$ is a circular shifted version of the order of accesses during the computation of $B_{i1,j1}$, where $i \neq i1$ and $j \neq j1$. This property enables us to design the data shuffler unit by a dynamic routing network.

Without loss of generality, consider the case when $p' \leq Q$. If $I_{i,j}$ is the top-left pixel of block $B_{i,j}$, and if U is the subsampling, then the Q pixels of a row that get accessed are from modules $(j + vU) \bmod e$, where $0 \leq v \leq Q - 1$. Let $m(q1)$ be the module from which $I_{i,j1}$ is accessed, where $q1 = j1 \bmod e$. Let $SEQ(m(q1))$ be the sequence of modules that are accessed when the leftmost pixel of the row is module $m(q1)$. $SEQ(m(q1))$ is obtained by varying v from 0 to $Q-1$ in the expression $(j1+vU) \bmod e$. The elements of $SEQ(m(q1))$ need not be distinct. However, $SEQ(m(q1))$ consists of a sequence of size e whose elements are distinct. We refer to this sequence $S(m(q1))$ as the *kernel* sequence ($S(m(q1))$ is obtained by varying v from 0 to $e-1$ in the expression $(j1+vU) \bmod e$.). For instance, in an example with $p'=4$, $Q=8$ and $e=5$, $SEQ(m(q1)) = \{0,3,1,4,2,0,3,1\}$ and $S(m(q1)) = \{0,3,1,4,2\}$. It can be shown that for $j1 \neq j2$, the kernel sequence $S(m(q2))$ for $B_{i,j2}$, where $q2 = j2 \bmod e$, is a circular shifted version of $S(m(q1))$. If we denote one of the kernel sequences as the *master* sequence, then all the other kernel sequences can be obtained by circular shifting the master sequence.

The same idea can be extended for the case when $p' > Q$. In this case, both the bank level accesses and the row level accesses have to be reordered. If pixel $I_{i1,j}$ is the top-left pixel of block $B_{i1,j}$, then the Q rows that get accessed are from banks $(i1+vU) \bmod r$, where $0 \leq v \leq Q-1$. Let $b(q1)$ be the bank that contains the topmost row (row $i1$) of the block, where $q1 = i1 \bmod r$. Let $SEQ(b(q1))$ be the sequence of banks that are accessed when the topmost row is in $b(q1)$. Then $S(b(q1))$ is a kernel sequence of size r whose elements are distinct. It can be shown that for $i1 \neq i2$, the sequence $S(b(q2))$ for block $B_{i2,j}$, where $q2 = i2 \bmod r$, is a circular shifted version of $S(b(q1))$. The proof for both the cases have been included in Appendix 2.

For $p' \leq Q$, the sequences corresponding to different values of $m(q)$ can be obtained by rotating the master sequence by a maximum of $\lfloor \frac{e}{2} \rfloor$ times to the left or right. This procedure is implemented by the data shuffler unit as follows. There are $\lceil \frac{e}{2} \rceil$ stages of multiplexers which are used to route the data through this network. Each stage consist of e 3:1 multiplexers which shift the data up or down or not at all. Since the amount of shift along with the direction of shift with respect to the master sequence is known apriori, a finite state machine can be designed to control each stage of switches. Fig. 8 shows the 3 types of multiplexers that are used for the implementation of the data shuffler. Switch type A is a 2 to 1 multiplexer, whereas switch type B is a 3 to 1 multiplexer. Both switches have 3 outputs which always hold the same values. Switch type C is the same as switch B except for the fact that it has only one output. A similar procedure for reordering the data is applied for the case when $p' > Q$. Here $r > 1$, and the bank level accesses are reordered by $\lceil \frac{r}{2} \rceil$ additional stages with rp' $3:1$ multiplexers per stage.

Example: We next describe the function of the data shuffler unit for the case when $p'=4$, $e=5$, $r=1$, $Q=8$ and $U_2=2$. The leftmost pixel in a row of the block being matched can lie in any of the

5 modules. Let the master sequence be $S(m(0)) = \{0, 2, 4, 1, 3\}$. Note from Table 5, that sequences $S(m(1))$ through $S(m(4))$ are rotated versions of $S(m(0))$. For instance, $S(m(1)) = \{1, 3, 0, 2, 4\}$ can be obtained by right rotating $S(m(0))$ two times. □

Module # of top left pixel	Memory module number for pixel								$S(m(\#))$
	0	1	2	3	4	5	6	7	
m(0)	m(0)	m(2)	m(4)	m(1)	m(3)	m(0)	m(2)	m(4)	$\{0,2,4,1,3\}$
m(1)	m(1)	m(3)	m(0)	m(2)	m(4)	m(1)	m(3)	m(0)	$\{1,3,0,2,4\}$
m(2)	m(2)	m(4)	m(1)	m(3)	m(0)	m(2)	m(4)	m(1)	$\{2,4,1,3,0\}$
m(3)	m(3)	m(0)	m(2)	m(4)	m(1)	m(3)	m(0)	m(2)	$\{3,0,2,4,1\}$
m(4)	m(4)	m(1)	m(3)	m(0)	m(2)	m(4)	m(1)	m(3)	$\{4,1,3,0,2\}$

Table 5: Example where $p' = 4$, $e = 5$, $r = 1$, $Q = 8$ and $U_2 = 2$. The entries indicate which memory modules supply the 8 pixels of a row of a block, given the module number of the top left pixel. m(#) is the memory module number.

From Table 5 it is possible to derive the mapping between the processors and the memory modules. Since $p' = 4$, in the first cycle, pixels 0 through 3 are processed, and in the second cycle pixels 4 through 7 are processed. If the leftmost pixel is in $m(1)$, then in the 1st cycle, processor 0 gets its pixel from $m(1)$, processor 1 gets its pixel from $m(3)$, and so on. In the second cycle, processor 0 gets its pixel from $m(4)$, processor 1 gets its pixel from $m(1)$ and so on. Table 6 describes the mapping between the processors and the memory modules.

Module # of top left pixel	Processor #									
	cycle 1					cycle 2				
	m(0)	m(1)	m(2)	m(3)	m(4)	m(0)	m(1)	m(2)	m(3)	m(4)
m(0)	P(0)	P(3)	P(1)		P(2)	P(1)		P(2)	P(0)	P(3)
m(1)	P(2)	P(0)	P(3)	P(1)		P(3)	P(1)		P(2)	P(0)
m(2)		P(2)	P(0)	P(3)	P(1)	P(0)	P(3)	P(1)		P(2)
m(3)	P(1)		P(2)	P(0)	P(3)	P(2)	P(0)	P(3)	P(1)	
m(4)	P(3)	P(1)		P(2)	P(0)		P(2)	P(0)	P(3)	P(1)

Table 6: Example where $p' = 4$, $e = 5$, $r = 1$, $Q = 8$ and $U_2 = 2$. The entries indicate which processors get pixels from which memory modules, given the module number of the leftmost pixel in a row. m(#) is the memory module number and P(#) is the processor number.

Fig. 9 shows a 3 stage data shuffler for the above example. The type A switches latch the data corresponding to the master sequences: 0-3-1-4-2 for $U_2 = 2$ and 0-2-4-1-3 for $U_1 = 8$. The control signal L is reset to 0 for blocks subsampled by U_2 and is set to 1 for blocks subsampled by U_1. The multiplexers of type B and C then shift the data up or down by a maximum of $\lfloor e/2 \rfloor = 2$ positions. The control signals S_1, S_2, S_3 and S_4 are set by a finite state machine. VLSI implementation of the data shuffler unit shows that the delay introduced by the multiplexer switches is very small and hence the entire unit can be considered as one pipeline stage.

3.6.4 Scheduling

Since on-chip Memory 1 contains the search area for only 4 neighbouring blocks, the data hazards caused due to inter-level and intra-level dependencies cannot be avoided altogether. Extra clock cycles have to be added to refill the pipeline during the intra-level computations of a group of blocks. Table 7 describes this schedule. The number of pipeline stages in Architecture 2 (see fig. 7) is $\lceil \log_2 p \rceil + 6$ [4]. Thus the time required to refill the pipeline is $\Delta = \lceil \log_2 p \rceil + 6$ cycles.

The minimum number of processors (p) now required for the two level HBMA of Table 1 can be found from the equation

$$\frac{256}{MNfc} = 99 * \lceil \frac{64}{p} \rceil + 3 * (\lceil \log_2 p \rceil + 6) \tag{4}$$

and p' is the smallest integral multiple or factor of Q that is greater than or equal to p.

Time Int.	T	Δ	T	Δ	T	Δ	T	T	T
Block	$B_{i+2S_2,j+2S_2}$		$B_{i+2S_2,j+2S_2}$		$B_{i+2S_2,j+2S_2}$		$B_{i+2S_2,j+2S_2}$	$B_{i+2S_2,j+S_2}$	$B_{i+2S_2,j+2S_2}$
Level	1		1		1		2	2	2
Step	1		2		3		1	1	2

Time Int.	T	T	T	T	T	T	Δ
Block	$B_{i+2S_2,j+S_2}$	$B_{i+S_2,j+S_2}$	$B_{i+S_2,j+2S_2}$	$B_{i+S_2,j+S_2}$	$B_{i+S_2,j+2S_2}$	$B_{i+2S_2,j+4S_2}$	
Level	2	2	2	2	2	1	
Step	2	1	1	2	2	1	

Table 7: Time schedule at which displacements for the blocks get computed in Architecture 2 when $S_1 = 2S_2$. T is the time required for 9 block matches and Δ is the time required to refill the pipeline.

If the on-chip memory size is increased to accomodate the level 1 search area of the next group of 4 blocks, the above data hazard can be totally avoided. In that case the schedule of Architecture 1 (Table 2) can be used here too.

[4] The processor tree has $\lceil \log_2 p \rceil + 3$ pipeline stages. The address generator, the data shuffler and memory 1 each contribute to 1 pipeline stage.

3.6.5 An Example

Table 8 shows the number of processors, the number of main memory modules and the extra amount of on-chip memory needed for Architecture 2. We assume a conservative on-chip memory access time of 70 ns., and main memory access time of 200 ns. Note that there is a drastic reduction in number of processors and memory modules at the cost of a few kbytes of on-chip memory.

Picture format	Block size $S_2 \times S_2$	HBMA		
		# of procs	# of external mem modules prev + curr	on-chip memory size in bytes Mem 1 + Mem 2
288×352, 10 Hz.	8×8	2	1+1	6177+320
288×352, 10 Hz.	16×16	1	1+1	7313+320
288×352, 30 Hz.	8×8	8	3+1	6177+320
288×352, 30 Hz.	16×16	2	2+1	7313+320
576×720, 25 Hz.	8×8	32	10+2	6177+320
576×720, 25 Hz.	16×16	8	5+1	7313+320

Table 8: Hardware resources required by Architecture 2.

3.6.6 Implementation

A processing unit that can handle block sizes of 16×16 for the teleconferencing video format for 8-bit parallel data has been implemented using Berkeley CAD Tools. The technology used was $1\mu m$ CMOS. The unit consists of a processor tree, a data shuffler, an address generating circuit, an address shuffler and the necessary glue logic. The address generating circuit employs a table lookup ROM to carry out the division. It generates multiple addresses which are routed to the memory modules using an address shuffler that is similar to the data shuffler. All the subunits are clocked at 70ns. The processor tree with four leaf processors is of size 6.83 sq.mm. The data shuffler which routes data from 5 memory modules to 4 leaf processors is of size 2.45 sq.mm. The layout of these subunits have not been compacted.

In this implementation, the on-chip memories are required to support simultaneous read and write operations. This can be achieved by using either dual port memories or single port memories with a modified organization. In our design, the two on-chip memories were implemented by single port memories divided into multiple pages to allow simultaneous read and write operations in different pages. On-chip memory 1 was split into 5 modules, where each module consisted of 22 pages of 72 bytes each. On-chip Memory 2 was split into 4 modules, where each module consisted of 5 pages of 16 bytes each. However, such an implementation of the on-chip memory required a very large chip area. In retrospect, a dual port static RAM such as the one available commercially from Integrated Devices Technology Inc. [20] would have been a better choice. The $16K \times 8$ dual port static RAM (IDT7006) has a 70 ns access time and is ideal for such implementations.

4 Architectures for FBMA and 3HSA

In this section we propose architectures with on-chip memory to implement FBMA and 3HSA. The proposed architectures have significantly smaller number of processors and memory modules than the existing architectures of [10], [7].

4.1 Full Search Block Matching Algorithm (FBMA)

Processor Architecture:

In FBMA, for an image of size $M \times N$ pixels, and block size of $W \times W$, $(2d+1)^2$ block matches are required for every motion vector estimate, where d is the maximum estimated displacement. If p is the number of processors, c is the on-chip memory access time, and f is the frame refresh rate, then the minimum value for p can be found from the equation

$$\frac{W^2}{MNfc} = (2d+1)^2 * \lceil \frac{W^2}{p} \rceil, \tag{5}$$

The tree architecture with p leaf processors (see section 3.1) can be used here too.

Memory Organization:

The memory organization for FBMA is similar to the one used by Architecture 2 for HBMA. It consists of two local on-chip memories to house the search area and the reference block, and two external image frame memories to store the current and the previous image frames. The on-chip memories are organized into p modules since p pixels have to be accessed in parallel.

For FBMA, the size of the search area is $(2d+W)^2$. Since consecutive search areas for FBMA overlap, the fresh amount of data to be loaded for the next search area is $(2d+W)W$. Therefore the total size of on-chip memory 1 is $(2d+W)^2 + (2d+W)W = 2(d+W)(2d+W)$. On-chip Memory 2 holds two blocks and hence is of size $2W^2$. $(2d+W)W$ data elements need to be accessed from the external previous image frame memory in the time available for one motion estimate. Therefore, for external memory access time of m, the number of external memory modules needed for the previous image frame is $\lceil \frac{mMNf*(2d+W)}{W} \rceil$. W^2 data elements need to be fetched from the current image frame memory in the same time. Therefore $\lceil mMNf \rceil$ memory modules are required for the current image frame.

The data is distributed in the on-chip memory and in the external memory as described in section 3.2 (see equation 3). A data shuffler unit similar to the one described for HBMA can be used for FBMA as well. Since FBMA does not have any computational dependencies, the scheduling is very straight-forward. Table 9 shows the hardware resources that are required for the FBMA algorithm for the three standard video formats (assuming on-chip memory access time of c = 70 ns and $d = 7$). Note that the number of processors is significantly lower than that of [10] (compare with Table 3). One and two dimensional systolic array architectures have also been proposed in [7] where the number of PEs is a function of the block size. The number of processors in the architecture of [7] is higher than ours for the same cycle time.

Picture format	Block size $W \times W$	FBMA # of procs	FBMA # of external mem modules prev + curr	FBMA on-chip Memory 1 (bytes)
288×352, 10 Hz.	8×8	16	1+1	660
288×352, 10 Hz.	16×16	16	1+1	1380
288×352, 30 Hz.	8×8	64	2+1	660
288×352, 30 Hz.	16×16	64	2+1	1380
576×720, 25 Hz.	8×8	256	6+1	660
576×720, 25 Hz.	16×16	256	4+1	1380

Table 9: Hardware resources required by the proposed architecture for FBMA

4.2 3 Step Hierarchical Search Algorithm (3HSA)

Processor Architecture:

For an image of size $M \times N$ pixels, block size of $W \times W$, 3HSA carries out 27 block matches per block. If c is the on-chip memory access time and f is the frame refresh rate, the minimum number of processors (p) required would be

$$\frac{W^2}{MNfc} = 27 * \lceil \frac{W^2}{p} \rceil$$

A tree architecture with p leaf processors can be used for 3HSA as well. The memory size and organization for 3HSA is the same as that of FBMA for the case when $d = 7$.

Scheduling:

The intra-level computational dependencies in HBMA are also present in 3HSA. For instance, step 3 of 3HSA can proceed only after results from step 2 are available, and step 2 can not proceed until step is 1 over. Therefore the pipeline needs to be stalled between step 1 and step 2, and between step 2 and step 3 of every block match. Table 10 shows the time schedule for the blocks centered around pixels shown in Fig. 2. Time interval T is equivalent to time required for 9 block matches, and Δ is the time required to refill the pipeline.

Time Interval	T	Δ	T	Δ	T	T	Δ	T	Δ	T	T	Δ
Block	$B_{i,j}$		$B_{i,j}$		$B_{i,j}$	$B_{i,j+W}$		$B_{i,j+W}$		$B_{i,j+W}$	$B_{i,j+2W}$	
Step	1		2		3	1		2		3	1	

Table 10: Time schedule at which displacements for the blocks get computed in 3HSA.

The minimum number of PEs (p) now required for the 3HSA algorithm, can be found from the equation

$$\frac{W^2}{MNfc} = 27 * \lceil \frac{W^2}{p} \rceil + 2 * (\lceil \log_2 p \rceil + 6) \tag{6}$$

As in the HBMA, if the on-chip memory size is increased to accomodate search areas for two consecutive blocks, the pipeline stalls can be eliminated by employing pipeline interleaving.

Table 11 shows the hardware resources that are required for the 3HSA algorithm for the three standard video formats (assuming c = 70 ns., and d = 7). It can be seen that the proposed

Picture format	Block size $W \times W$	3HSA # of procs	# of external mem modules prev + curr	on-chip Memory 1 (bytes)
288×352, 10 Hz.	8×8	2	1+1	660
288×352, 10 Hz.	16×16	2	1+1	1380
288×352, 30 Hz.	8×8	7	2+1	660
288×352, 30 Hz.	16×16	6	2+1	1380
576×720, 25 Hz.	8×8	32	6+1	660
576×720, 25 Hz.	16×16	22	4+1	1380

Table 11: Hardware resources required by the proposed architecture for 3HSA

architecture requires fewer processors and drastically fewer external memory modules for 3HSA compared to [10] (see Table 3).

5 Conclusion

We have presented two area efficient, high throughput special purpose architectures for hierarchical BMA. Architecture 1 uses no on-chip memory, but has a very high memory bandwidth. Architecture 2 on the other hand, has a small on-chip memory which helps reduce the memory bandwidth. It also requires fewer processors compared to Architecture 1. We have suggested a new memory organization scheme for both the architectures that permits parallel access of multiple pixels that are equidistant. We have also proposed a new dynamic data routing network for Architecture 2 which parallely routes data from the memory modules to the right processors. Finally, we have shown how an architecture similar to Architecture 2 can be used to implement full search and 3-step hierarchical search BMA, with significant reduction in the hardware complexity.

References

[1] A. Habibi, "An adaptive strategy for hybrid image coding," *IEEE Trans. Comm.,* vol. COM-29, pp. 1736-1740, Dec. 1981.

[2] A. N. Netravali and B. G. Haskell, *Digital Picture Representation and Compression.* New York: Plenum, 1988.

[3] H. G. Musmann, P. Pirsch, and H. J. Grallert, "Advances in picture coding," *Proc. IEEE,* vol. 73, no. 4, pp. 523-548, Apr. 1985.

[4] K. Guttag, R. J. Gove, and J. R. Van Aken, "A single chip multiprocessor for multimedia: The MVP," *IEEE Computer Graphics and Application,* vol. 12, no. 6, pp. 53-64, Nov. 1992.

[5] D. Gall, "MPEG: A video compression standard for multimedia applications," *Comm. ACM,* vol. 34, no. 4, Apr. 1991.

[6] M. Bierling, "Displacement estimation by hierarchical blockmatching," *SPIE Visual Comm. and Image Processing,* vol. 1001, pp. 942-951, 1988.

[7] T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Trans. Circuits and Systems,* vol. 36, no. 10, pp. 1301-1308, Oct. 1989.

[8] K. Y. Yang, M-T. Sun, and L. Wu, "A family of VLSI designs for the motion compensation block matching algorithm," *IEEE Trans. Circuits and Systems,* vol. 36, no. 10, pp. 1317-1325, Oct. 1989.

[9] L. De Vos and M. Stegherr, "Parameterizable VLSI architectures for the full-search block-matching algorithm," *IEEE Trans. Circuits and Systems,* vol. 36, no. 10, pp. 1309-1316, Oct. 1989.

[10] Y. S. Jehng and L. G. Chen, "An efficient and simple VLSI tree architecture for Motion Estimation Algorithms," *IEEE Trans. Signal Processing,* vol. 41, no. 2, Feb. 1993.

[11] Y. S. Jehng, L. G. Chen, T-D. Chiueh, and T-H. Chen, "Realization of array architectures for video compression algorithms," *Proc. of the IEEE Int. Symp. on Circuits and Systems,* pp. 1672-1675, 1992.

[12] T. Komarek, P. Pirsch, "VLSI architectures for hierarchical block matching algorithm," *IFIP Workshop, Grenoble, France,* pp. 168-181, Dec. 1989.

[13] L. DeVos, "VLSI architectures for the hierarchical block-matching algorithms for HDTV applications," *Proceedings of SPIE vol. 1360 Visual Communications and Image Processing'90*, pp.398-409.

[14] G. Gupta and C. Chakrabarti, "VLSI Architectures for Hierarchical Block Matching Algorithm," *1994 IEEE Int. Symp. on Circuits and Systems,* vol. 4, pp. 215-218.

[15] P. Gerken and H. Amor, "Hierarchical displacement estimation with low computational effort for applications in 64Kbits/s moving video coding," *Second International Workshop on 64Kbits/s Coding of Moving Video*, Hannover, Germany, Sept 89.

[16] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion-compensated interframe coding for video conferencing," *Proc. of the NTC 81,* pp. G5. 3. 1-G5. 3. 5, Dec. 1981.

[17] D. C. Van Voorhis and T. H. Morrin, "Memory systems for image processing," *IEEE Trans. Comput.* vol. COM-27, no. 2, pp. 113-125, Feb. 1978.

[18] J. W. Park, "An efficient memory system for image processing," *IEEE Trans. Comput.* vol. COM-35, no. 7, pp. 669-674, July 1986.

[19] H. C. Kim and S. R. Maeng, "Pipelined Systolic Arrays Architecture for the Hierarchical Block-Matching Algorithm," *1994 IEEE Int. Symp. on Circuits and Systems,* vol. 3, pp. 221-224.

[20] Integrated Devices Technology Inc., *High Performance CMOS Data Book Supplement,* pp. s5109-s5111, 1989.

Appendix 1: Memory Organization

In hierarchical BMA, if $I_{i,j}$ is the top left-most pixel of a block being matched, then the pixels that are used for a single MAD computation are (i) $I_{i+U_1m,j+U_1n}$, $0 \leq m,n \leq Q$, for level 1 search, and (ii) $I_{i+U_2m,j+U_2n}$, $0 \leq m,n \leq Q$, for level 2 search. The pixels of the image frames are distributed among the memory modules in the following way (see section 3.2). Pixel $I_{i,j}$ is assigned to location α in module k in memory bank b, where $b = i \bmod r$, $k = j \bmod e$, and $\alpha = \lceil \frac{N}{e} \rceil * \lfloor \frac{i}{r} \rfloor + \lfloor \frac{j}{e} \rfloor$. Here r is the smallest integer greater than or equal to $\lfloor \frac{p'}{Q} \rfloor$ such that r is not a multiple or factor of U_1 and U_2, and e is the smallest integer greater than or equal to p' for $p' \leq Q$ and greater than or equal to Q for $p' > Q$, such that it is not a multiple or a factor of U_1 and U_2. In this Appendix, we prove that the above distribution supports simultaneous access of p' pixels of a block. We study 2 cases: (i) $p' \leq Q$ and (ii) $p' > Q$.

Case I ($p' \leq Q$)

When $p' \leq Q$, the memory is organized into one bank, with e memory modules. Thus all rows of the image frame lie in the same bank. It is sufficient to prove that, pixels $I_{i,j+U_qn}$, where $q = 1, 2$ and $0 \leq n < p'$, lie in different memory modules. The proof is by contradiction.

Assume that for some $n = n_1$, $I_{i,j}$ and $I_{i,j+U_qn_1}$ are mapped to the same memory module. Let $k = j \bmod e$ and $k_1 = (j + U_qn_1) \bmod e$. Then $k = k_1$ implies that

$$(j + U_qn_1) \bmod e = j \bmod e \Rightarrow (U_qn_1) \bmod e = 0$$

This implies that U_qn_1 is a multiple of e, i.e., $n_1 = \frac{e}{U_q} * l$, where l is an integer.
But, since e is not a multiple or factor of U_q and n_1 is an integer $\leq p' - 1$, there exists no l other than 0 which satisfies the above equation. Therefore, p' memory elements can be accessed simultaneously from e memory modules when the elements are U_q apart, $q = 1, 2$.

Case II ($p' > Q$)

When $p' > Q$, the memory is organized into r memory banks, with e modules per bank. Recall that an entire row of the image is contained in the same memory bank. We first prove that for the above distribution, rows that are U_q apart lie in different memory banks, and thus a rows can be accessed in parallel. Thus it is sufficient to prove that pixels $I_{i+U_qm,j}$ where $0 \leq m < r$, lie in different banks. The proof is by contradiction.

Assume for some $m = m_1$, $I_{i,j}$ and $I_{i+U_q m_1,j}$ lie in the same bank. Therefore,

$$(i + U_q m_1) \bmod r = i \bmod r \Rightarrow (U_q m_1) \bmod r = 0$$

This implies that $m_1 = \frac{r}{U_q} * l$, where l is an integer.
But r is not a multiple or factor of U_q and there exists no l other than 0 that satisfies the above equation.

We next prove that within a bank, the Q pixels along a row (of a block) can be accessed in parallel. This proof is same as that of Case I when $p' = Q$.

$\square$

Appendix 2: Data Shuffler

In this Appendix, we show that for any U, r and e, the sequence of modules and banks that are accessed during different computations are related. Let $I_{i,j}$ be the top-left pixel of block $B_{i,j}$. Then the sequence of modules and banks accessed by block $B_{i1,j1}$ is a rotated version of the sequence of modules and banks accessed by block $B_{i2,j2}$, where $i1 \neq i2$ and $j1 \neq j2$. We consider two cases: (i) $p' \leq Q$ and (ii) $p' > Q$.

Case I ($p' \leq Q$)

In this case, the Q pixels of a row are accessed from modules $(j + vU) \bmod e$, $0 \leq v \leq Q - 1$. Let $q = j \bmod e$. Then $SEQ(m(q))$ is a sequence of size Q corresponding to the modules that are accessed when the leftmost pixel of the row is in module $m(q)$. Let $S(m(q))$ be the kernel sequence of $SEQ(m(q))$. To prove that for any $j = j1$ and $j = j2$, the sequence $S(m(q1))$ can be obtained by rotating the sequence $S(m(q2))$, where $q1 = j1 \bmod e$ and $q2 = j2 \bmod e$. Since both the sequences consist of e distinct elements, without loss of generality we assume that the element corresponding to $v = v1$ in $S(m(q1))$ matches the element corresponding to $v = v2$ in $S(m(q2))$. Thus it is sufficient to prove that the element corresponding to $v = v_1 + a$ of $S(m(q1))$ is the same as the element corresponding to $v = v_2 + a$ of $S(m(q2))$ for $0 \leq a \leq e - 1$.

By definition,

$$(j_1 + v_1 U) \bmod e = (j_2 + v_2 U) \bmod e.$$

This implies that

$$j_1 + v_1 U = l + e\alpha, \;\; j_2 + v_2 U = l + e\beta,$$

where α, β are integers.

$$(j_1 + (v_1 + a)U) \bmod e = (j_1 + v_1 U + aU) \bmod e = (l + e\alpha + aU) \bmod e = (l + aU) \bmod e.$$

$$(j_2 + (v_2 + a)U) \bmod e = (j_2 + v_2 U + aU) \bmod e = (l + e\beta + aU) \bmod e = (l + aU) \bmod e$$

Thus $(j_1 + (v_1 + a)U) \bmod e = (j_2 + (v_2 + a)U) \bmod e$ for any a, and $S(m(q2))$ is a rotated version of $S(m(q1))$.

Case II ($p' \geq Q$)

In this case we prove that during the computation of two distinct blocks, the sequence of banks as well as the sequence of modules in a bank are related. If $I_{i,j}$ is the top-left pixel of block $B_{i,j}$, then the Q banks that get accessed are $(i + vU) \bmod r$, $0 \leq v \leq Q - 1$. To prove that for any $i = i1$ and $i = i2$, the sequence $S(b(q1))$ is obtained by rotating the sequence $S(b(q2))$ where $q1 = i1 \bmod r$ and $q2 = i2 \bmod r$.

Let the element corresponding to $v = v1$ in $S(b(q1))$ match the element corresponding to $v = v2$ in $S(b(q2))$. Then it can be shown that the element corresponding to $v = v1 + a$ in $S(b(q1))$ is the same as the element corresponding to $v = v2 + a$ in $S(b(q2))$ for $0 \leq a < r$. The proof is the same as Case I (replace e by r, $j1$ by $i1$ and $j2$ by $i2$). Within a bank, the module level accesses are the same as Case I. $\square$

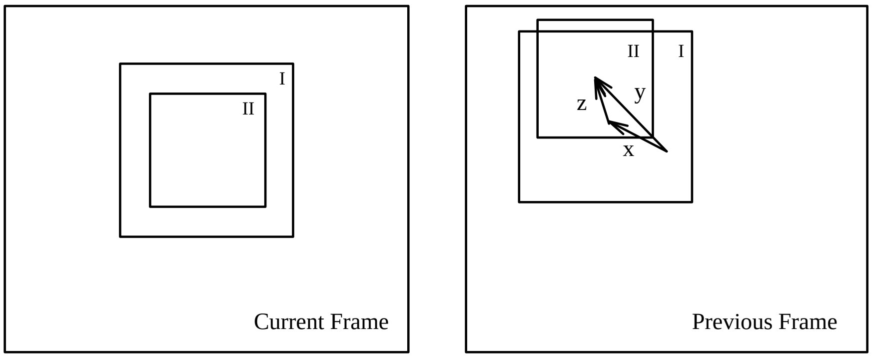

Figure 1: Hierarchical Block Matching Algorithm for two levels of hierarchy.

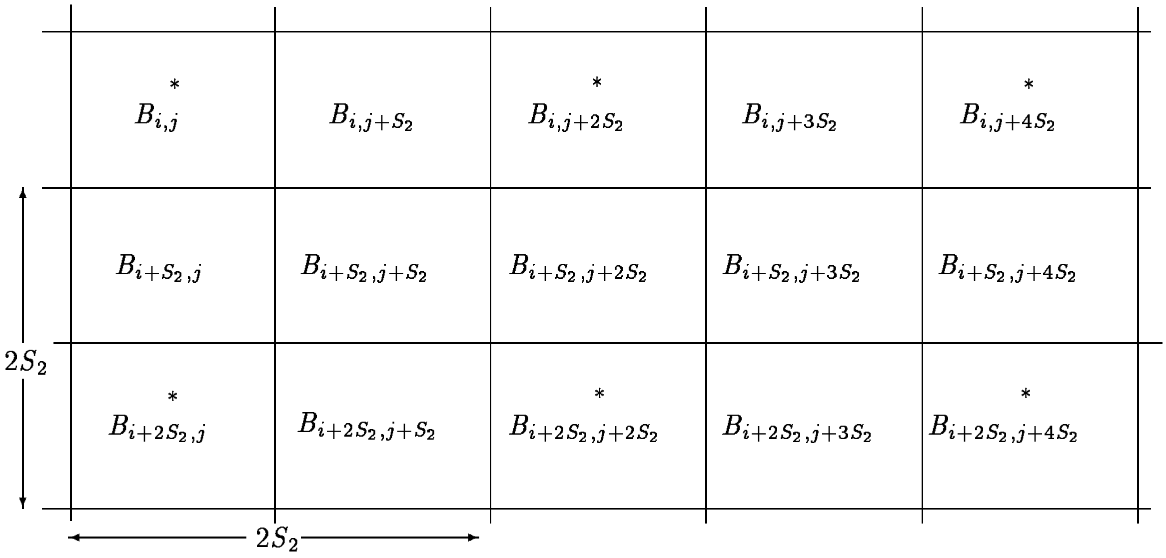

Figure 2: A part of the image frame when $S_1 = 2S_2$. The level 1 displacement is estimated using MAD computations for only the blocks that are marked with $*$. The level 2 displacement is estimated for all the blocks.

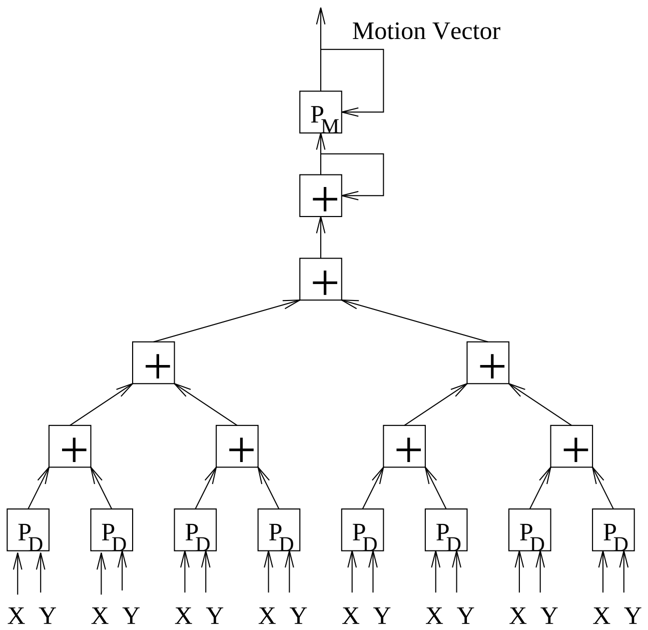

Figure 3: A tree of processors with $p' = 8$. X is the data from the reference block and Y is the data from the candidate block.

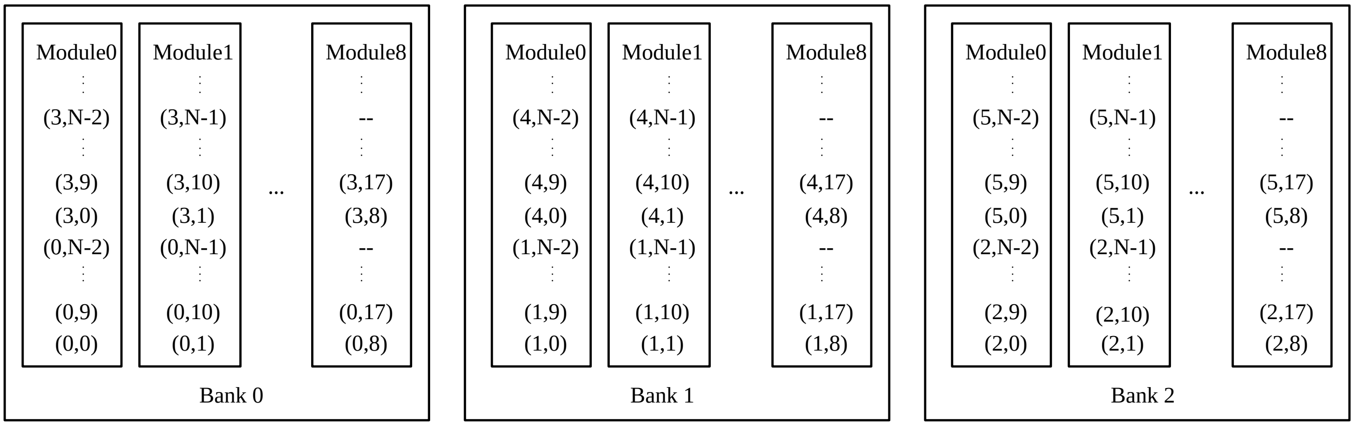

Figure 4: Memory organization for the case when $p' = 16$, $r = 3$ and $e = 9$

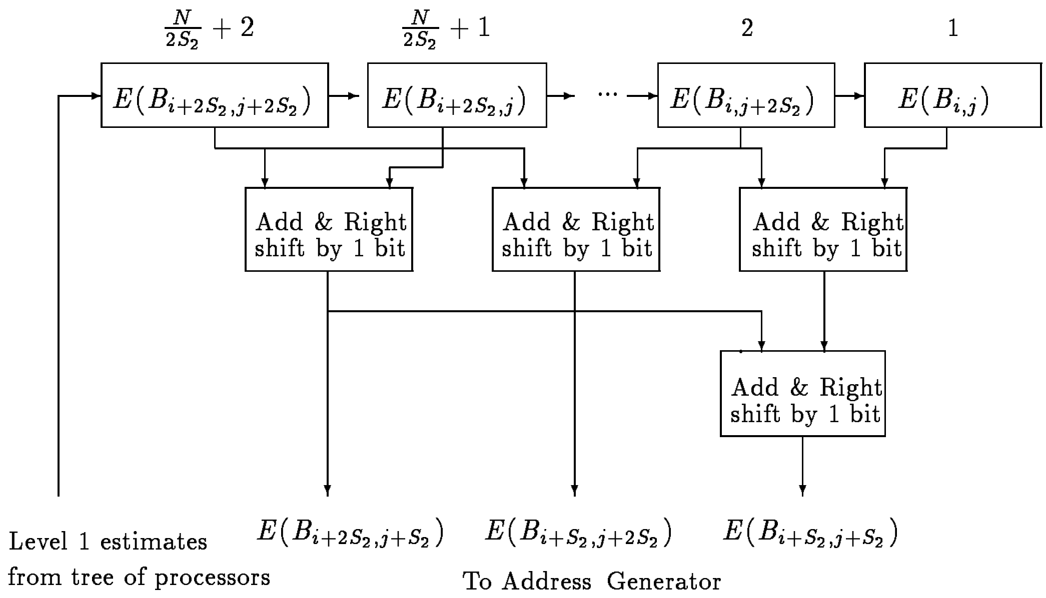

Figure 5: Block diagram of the bilinear interpolator

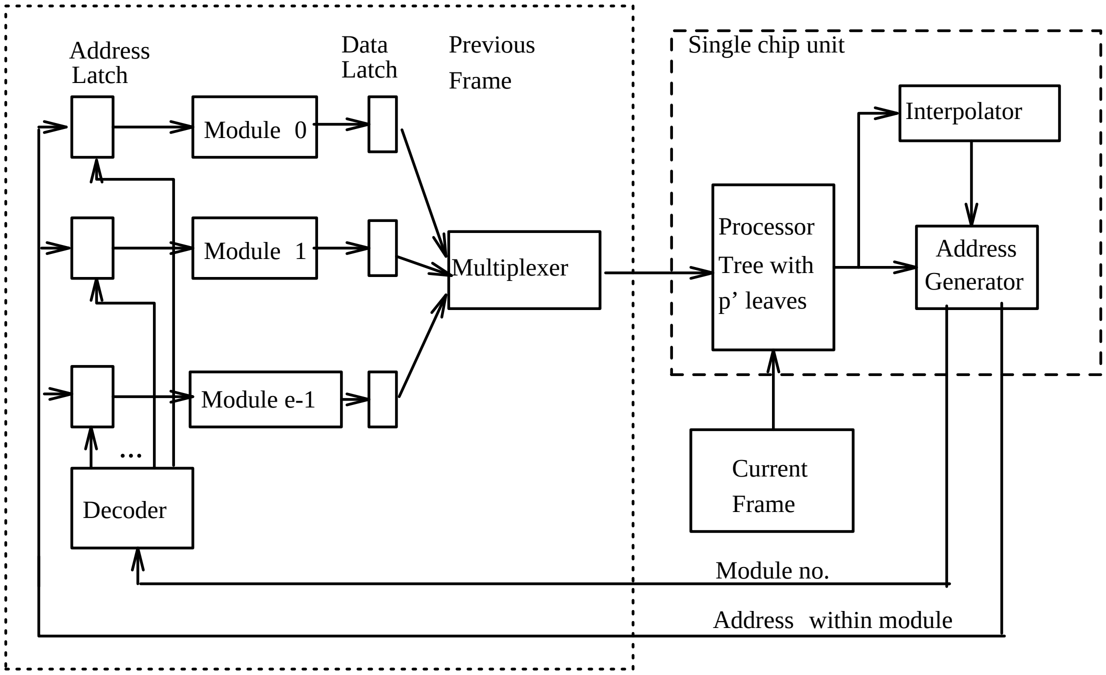

Figure 6: Block diagram of Architecture 1 for $p' \leq Q$.

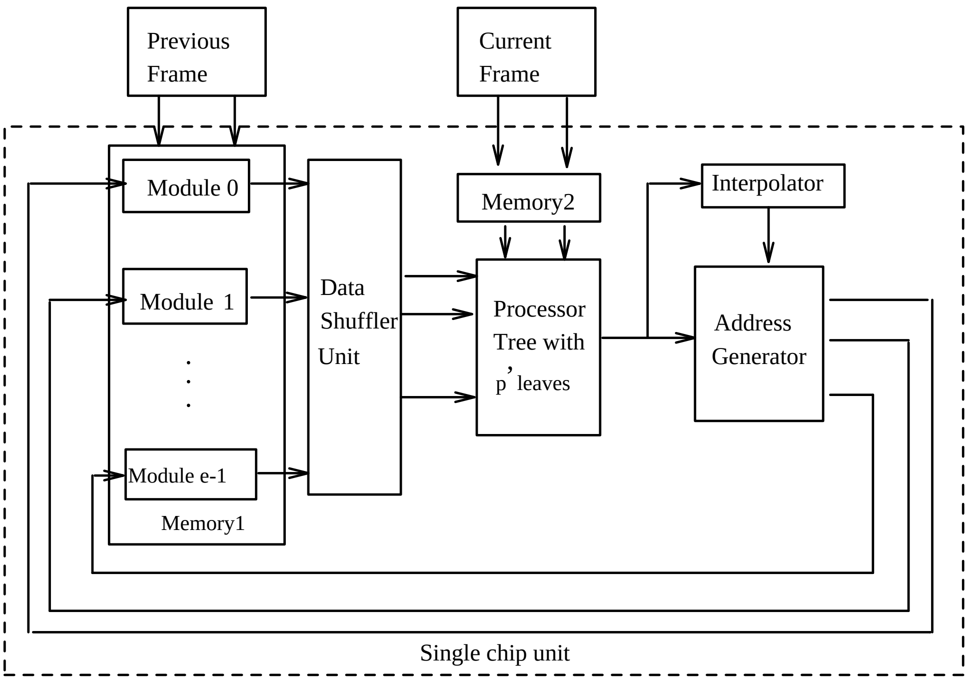

Figure 7: Block diagram of Architecture 2 for $p' \leq Q$.

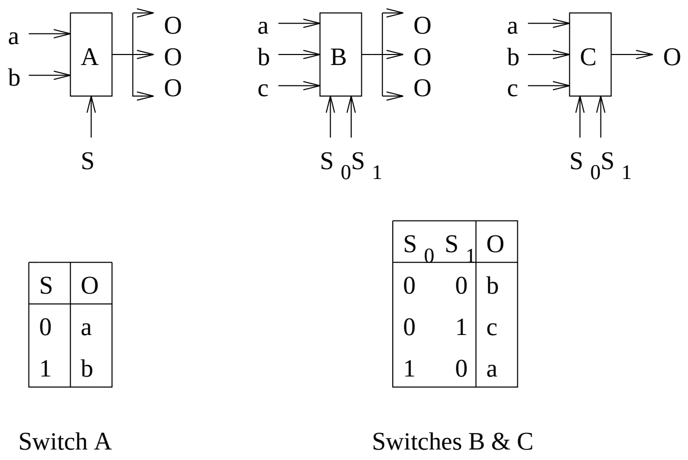

Figure 8: The switches used in the data shuffler unit.

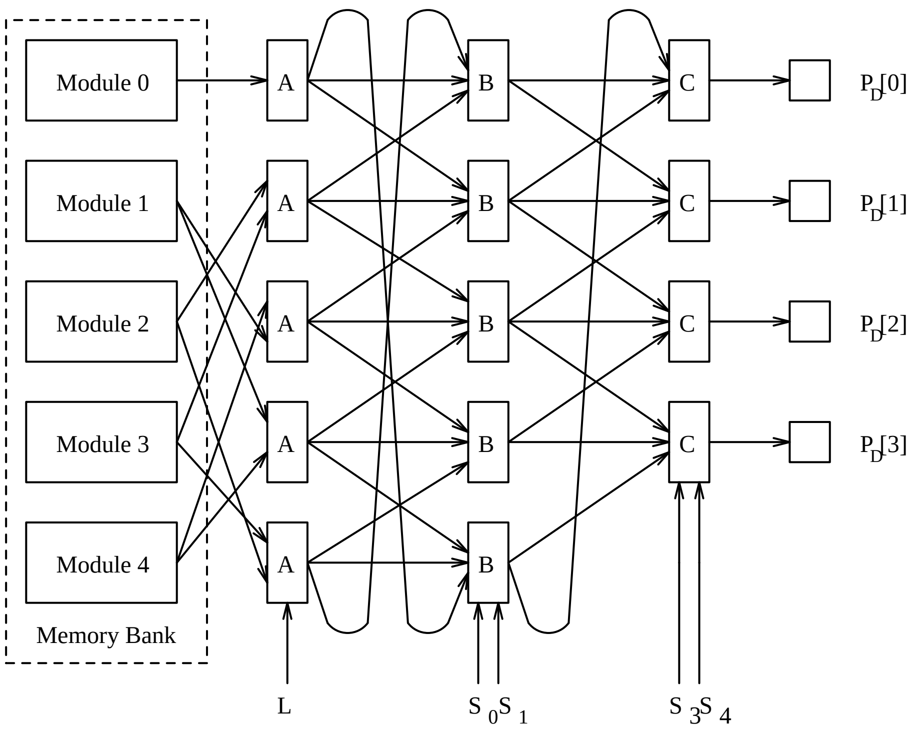

Figure 9: A 3-stage data shuffler for an example with $Q = 8$, $p' = 4$, and $e = 5$.