VLSI ARCHITECTURES FOR HIERARCHICAL BLOCK MATCHING

Gagan Gupta
Dept. of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287, USA
gupta@dspsun.eas.asu.edu

Chaitali Chakrabarti
Dept. of Electrical Engineering and TRC
Arizona State University
Tempe, AZ 85287-5706, USA
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 for implementing hierarchical block matching. 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 the processor architecture, the memory organization and the scheduling details for both the architectures.

1. INTRODUCTION

In many applications of visual communication, moving image is transmitted over low transmission rate channels. Compression techniques reduce the amount of data to be transmitted by exploiting the spatial and temporal redundancies in images. Motion compensated prediction is widely used to exploit temporal redundancies in video signals. An important component in motion compensated prediction is estimating motion between successive video frames [3].

There are two main techniques for motion estimation: pel recursive and block matching [3]. Block matching is more popular due to its relative computational simplicity. There are two classes of block matching algorithms (BMA): fixed block size BMA [3] and hierarchical BMA [1]. In fixed size BMA, a fixed size reference block is compared with either all or selected candidate blocks in a fixed size search area in the previous frame. The matching criteria that is used for finding the best match is the mean-of-the-absolute-differences (MAD) [3]. Fixed size BMAs give unreliable motion estimates 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. Hierarchical BMA, on the other hand, provides an adaptation of block size and search area to the properties of the image, and has been shown to give more accurate estimates [1].

A large number of VLSI architectures have been suggested in the literature for fixed size BMAs [4], [6]. However very little work has been done in the area of hierarchical BMA. Recently, a VLSI architecture with on-chip memory has been proposed for hierarchical BMA in [7]. The architecture is not optimized for number of processors as well as size of on-chip memory. Also, architectural details including memory organization, scheduling, etc have not been described.

This research was supported in part by a grant from NSF MIP-9309504.

In this paper we present two architectures for implementing hierarchical BMA in real-time. The two architectures differ in the number of processors and the size of on-chip memory. While the first architecture has no on-chip memory and requires large number of processors, the second architecture 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. In both the architectures the number of processors is small enough to support single chip implementation.

2. HIERARCHICAL BLOCK MATCHING ALGORITHM

Fig. 1 shows the principle of hierarchical block matching for two levels of hierarchy. A displacement estimate is to be made for the block centered in pixel A in the current frame. At the highest level, an estimate ($\vec{x}$) is made for block I around the original location of pixel A in the previous frame. At the next level of hierarchy, the block size is reduced, and an estimate ($\vec{z}$) is made for block II around the estimated displacement of block I. The displacement for the block centered in pixel A, $\vec{y}$, is given by $\vec{y} = \vec{x} + \vec{z}$. We also refer to $\vec{y}$ as the displacement for pixel A.

At each level of hierarchy, a $\log D$ step search technique is used to find the best match [2]. Table 1 gives all the parameters of a simplified hierarchical block matching algorithm for two levels of hierarchy and a block size of 16×16 [1]. A level I estimate is made for every 32^{nd} pixel in the image frame, and every 8^{th} pixel of the 64×64 block is used for match computation. The maximum estimated displacement in level I is ± 7 pixels. In level II, an estimate is made for every 16^{th} pixel, and every 2^{nd} pixel of the 16×16 block is used for match computation. The maximum displacement estimated is ± 3 pixels. A low pass mean value filtering on 5×5 size window is carried out to improve accuracy of the motion estimates. Fig. 2 shows a part of the

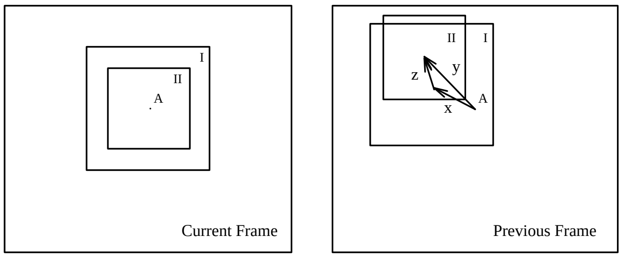

Figure 1: The HBMA for two levels of hierarchy.

Parameter at level	I	II
Max. update displacement	± 7	± 3
Block size	64	16
Step size	32	16
Filter window size	5	5
Subsampling	8	2

Table 1: Parameters for a simplified hierarchical BMA.

image frame. The level I estimate is done for pixels $I_{i,j}$, $I_{i,j+2S}$, $I_{i,j+4S}$, etc., and the level II estimate is done for all pixels. S is the step size used in level II search. Bilinear interpolation of the four closest neighbors is used to compute level I displacement estimate of the pixels for which the level I estimates are not known.

We next estimate the minimum number of processors that would be required for real-time implementation of the hierarchical BMA. Let the image size be $M \times N$, and the block size be $B \times B$, then 64 operations [1] are carried out per block (refer to Table 1). For level I, $\frac{M \times N}{2B \times 2B}$ motion vector estimates are made, and 27 blocks are matched per estimate. For level II, $\frac{M \times N}{B \times B}$ estimates are made, and 18 blocks are matched per estimate. Thus for a frame frequency of f Hz, the number of operations that are carried out per second are $\frac{MNf}{B^2} * (\frac{27}{4} + 18) * 64 \ = \frac{MNf}{B^2} * 24.75 * 64$. For different standard picture formats and block sizes ranging from (4×4) to (16×16), this gives a requirement of 6 to 1026 million operations per second. Multiple pipelined processors operating in parallel 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

$$p = \frac{MNfc}{B^2} * 24.75 * 64 \qquad (1)$$

3. ARCHITECTURE I

In this section we describe an architecture which consists of a tree of processors and two image memories to store the current frame and the previous frame. Since the processors need to access data from the image memories simultaneously, the image memories are distributed among multiple memory modules. Fig. 3 describes the block diagram of the proposed architecture.

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

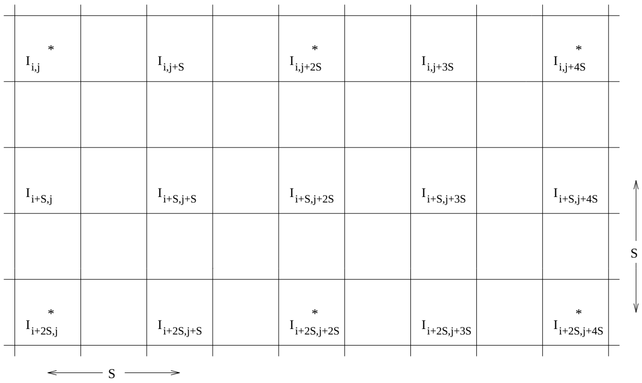

Figure 2: A part of the picture frame. S is the step size.

Processor Configuration

Let p' be the smallest power of 2 that is $\geq p$, where p is obtained from eqn 1. The proposed architecture consists of p' processors which are interconnected by a tree structure. The tree architecture is adopted to achieve small latency and fast computation times [6]. The p' processors of type P_D form the leaves of the tree. These processors compute the absolute differences between pixels of the current frame (X) and the pixels of the previous frame (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 delay in each stage is equal to the memory access time.

Bilinear interpolation is carried out using shift registers and adders.

Memory Organization

The p' processors of type P_D need to access p' data elements simultaneously. These p' data elements are 8 apart for level I search and 2 apart for level II search (see Table 1). To support this kind of access, the image frames are distributed among $\lfloor \frac{p'}{8} \rfloor + 1$ memory banks. If $p' \leq 8$, only one memory bank is required and the bank consists of $p' + 1$ memory modules. If $p' > 8$, each bank consists of 9 memory modules. The distribution is as follows: pixel I(i,j) is allocated to memory module k of memory bank b, where $b = i \bmod (\lfloor \frac{p'}{8} \rfloor + 1)$ and $k = j \bmod (p' + 1)$, $0 \leq i \leq M-1$, and $0 \leq j \leq N-1$. Within module k, the address assigned to pixel I(i,j) is $\lfloor \frac{i}{\lfloor \frac{p'}{8} \rfloor + 1} \rfloor + M * \lfloor \frac{j}{p'+1} \rfloor$. In [5] we show that such a distribution is sufficient to support simultaneous acces of p' elements that are either 8 apart or 2 apart. The distribution is the same for both current and previous image frames.

Fig.3 shows the block diagram of the memory system for the case $p' \leq 8$. For bit parallel operation, if the processor chip pin count is high, then accesses to modules within a bank can be pipelined and interleaved to reduce the pin count as in [6].

Schedule

Table 2 shows a schedule for the computation of the level I and level II estimates for the different pixels shown in Fig. 2. This schedule achieves 100% pipeline utilization.

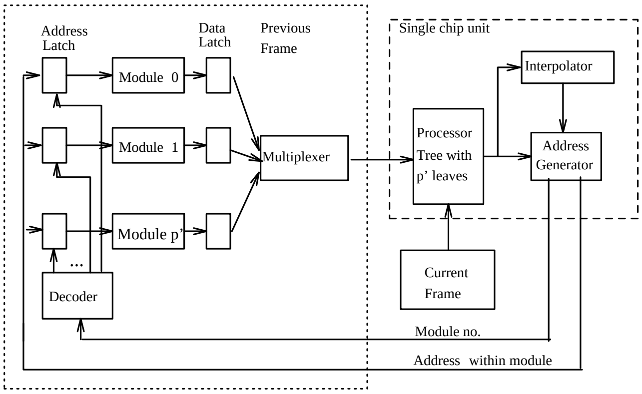

Figure 3: Block diagram of Architecture I. $p' \leq 8$.

One time unit in the table corresponds to the time taken to carry out one search step of a hierarchy level, i. e. 9 block matches.

4. ARCHITECTURE II

One major disadvantage of Architecture I is that it has very high memory bandwidth requirements (6 to 1000 Mbytes/s). This is because the same pixels are used for multiple computations, and are consequently accessed multiple times. 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 a faster on-chip memory access rate. Since the processors are essentially word-level subtractors, they can be easily clocked at access times of (say) 70ns.

Processor Configuration:

Let $\tilde{p}$ be the smallest power of $2 \geq p$ where p is the minimum number of processors that are required for real-time computation (refer to eqn 1). Then, as in Architecture I, the processors are organized in the form of a tree structure with p' leaf processors of type P_D. Processors of types P_D and P_M as well as the adders are all clocked at the rate of on-chip memory access times. For $\tilde{p}$ simultaneous accesses, the search area needs to be distributed among atleast $\tilde{p}$ on-chip memory modules. Moreover, since the same pixel may be accessed by different leaf processors at different times, a **data shuffler** unit is required to reorder the $\tilde{p}$ data elements accessed from the on-chip memory before forwarding them to the leaf processors.

Data Shuffler:

The data shuffler unit consists of multiple stages of multiplexer switches to carry out the reordering. Specifically, $\frac{m-1}{2}$ stages are required to reorder $m-1$ data elements from each bank, and each stage consists of m 3 to 1 multiplexers. In addition, for a system with b banks, $\frac{b-1}{2}$ additional stages of $(m-1)b$ 3 to 1 multiplexer switches are required to reorder the bank level accesses. Since the reordering is known apriori, a finite state machine can be designed to control each stage of switches.

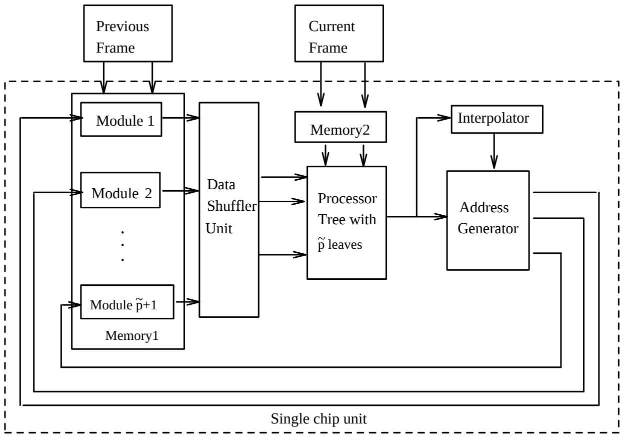

Figure 4: Block diagram of Architecture II. $p' \leq 8$.

Memory Organization:

In this architecture, there are two on-chip memory modules: Memory1 and Memory2. Memory1 is used to store the entire search area of the block whose displacement vector is being computed, while Memory2 is used to store the current block. Since there is a significant overlap between search areas for two consecutive blocks, Memory1 loads only that part of the search area which is not common. This drastically reduces the external memory bandwidth requirement.

We next estimate the minimum size of Memory1 by considering a part of the image frame shown in Fig. 2. For $B = 8$ or $B = 16$, search areas for $\mathrm{I}_{i+S,j+S}$, $\mathrm{I}_{i+S,j+2S}$ and $\mathrm{I}_{i+2S,j+S}$ are completely encompassed within the search area for $\mathrm{I}_{i+2S,j+2S}$. Hence, if Memory1 is loaded with the search area of $\mathrm{I}_{i+2S,j+2S}$, estimates for $\mathrm{I}_{i+S,j+S}$, $\mathrm{I}_{i+S,j+2S}$ and $\mathrm{I}_{i+2S,j+S}$ can be computed without loading any more data. When estimates for $\mathrm{I}_{i+2S,j+2S}$, $\mathrm{I}_{i+S,j+S}$, $\mathrm{I}_{i+S,j+2S}$ and $\mathrm{I}_{i+2S,j+S}$ are being computed, $2B*71$ pixels from the search area of $\mathrm{I}_{i+2S,j+2S}$ is loaded. Therefore Memory1 is of size $71*71+2*B*71$. Memory2 is used to store five blocks of the current frame (in this example, level I and level II blocks for $I_{i+2S,j+2S}$, level II blocks for $I_{i+S,j+S}$, $I_{i+S,j+2S}$, and $I_{i+2S,j+S}$). Thus Memory2 is of size $5*64 = 320$ bytes.

Memory1 is organized into memory banks and modules as in Architecture I, while Memory2 consists of one memory bank with $\tilde{p}$ memory modules. The image data is distributed in the cyclic mode in both the cases.

External Memory: If m is the external memory access time, then $\lceil \frac{mMNf*71}{2S} \rceil$ modules are needed for previous frame and $\lceil \frac{mMNf*64*11}{8B^2} \rceil$ are needed for current frame.

Schedule:

Since Memory1 contains search area for only 4 neighbouring pixels, the data hazards caused due to computational dependencies can not be avoided altogether. Hence for some cycles, pipeline needs to stall. However, if the on-chip memory size is increased to accomodate search areas of 8 consecutive blocks, all data hazards can be avoided, and 100% pipeline utilization can be achieved.

Fig. 4 describes the proposed system architecture. The on-chip memory, the data shuffler, the processor tree, and

Table 2: Time schedule for Architecture I

Time	t	$t+1$	$t+2$	$t+3$	$t+4$	$t+5$	$t+6$	$t+7$	$t+8$
Pixel	$I_{i+2S,j+2S}$	$I_{i+2S,j+4S}$	$I_{i+2S,j+2S}$	$I_{i+2S,j+4S}$	$I_{i+2S,j+2S}$	$I_{i+2S,j+4S}$	$I_{i+2S,j+2S}$	$I_{i+2S,j+S}$	$I_{i+S,j+S}$
Level	I	I	I	I	I	I	II	II	II
Step	1	1	2	2	3	3	1	1	1

Time	$t+9$	$t+10$	$t+11$	t+12	$t+13$	$t+14$	$t+15$	$t+16$	$t+17$
Pixel	$I_{i+S,j+2S}$	$I_{i+2S,j+2S}$	$I_{i+2S,j+S}$	$I_{i+S,j+S}$	$I_{i+S,j+2S}$	$I_{i+2S,j+4S}$	$I_{i+2S,j+3S}$	$I_{i+S,j+3S}$	$I_{i+S,j+4S}$
Level	II	II	II	II	II	II	II	II	II
Step	1	2	2	2	2	1	1	1	1

Table 3: Resources required for the two proposed architectures for HBMA.

Picture format	Block size	Architecture I		Architecture II			
		# of procs	# of external mem modules prev + curr	# of procs	# of on-chip mem modules	# of external mem modules prev + curr	Memory1 size in bytes
288 × 352, 10 Hz.	8 × 8	8	9+9	2	3	1+1	6177
288 × 352, 10 Hz.	16 × 16	2	3+3	1	1	1+1	7313
288 × 352, 30 Hz.	8 × 8	16	27+27	8	9	3+1	6177
288 × 352, 30 Hz.	16 × 16	4	5+5	2	3	2+1	7313
576 × 720, 25 Hz.	8 × 8	64	81+81	32	45	10+2	6177
576 × 720, 25 Hz.	16 × 16	16	27+27	8	9	5+1	7313

the address generator units can be housed in the same chip. Each of these units are pipelined so that they can be clocked at the on-chip memory access rate.

5. CONCLUSION

We have presented two architectures for implementing the hierarchical block matching algorithm. Architecture I uses no on-chip memory, but has high memory bandwidth, while Architecture II has minimum memory bandwidth but uses a 6-7 Kbytes of on-chip memory. Table 3 shows the number of processors and memory modules required for the two architectures for three different picture standards. An external memory access time of 200 ns. and on-chip memory access time of 70ns have been assumed. Note that Architecture II requires significantly fewer processors and fewer external memory modules compared to Architecture I. Both have simple and modular structures and can be easily implemented in VLSI. Architecture II can also be used to implement other block matching algorithms such as full-search and hierarchical search with a significant reduction in the number of processors.

6. REFERENCES

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

[2] 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.

[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] J. E. Franca and Y. Tsividis, *Design of analog-digital VLSI circuits for telecommunication and signal processing,* Englewood Cliffs, NJ: Prentice Hall, 1994.

[5] G. Gupta and C. Chakrabarti, "Architectures for Hierarchical Block Matching Algorithms", submitted to IEEE Trans on Circuits and Systems for Video Technology.

[6] 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.

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