A MEMORY EFFICIENT ARRAY ARCHITECTURE FOR FULL-SEARCH BLOCK MATCHING ALGORITHM

Vasily G. Moshnyaga *Keikichi Tamaru*
Department of Electronics, Kyoto University
Sakyo-ku, Yoshida-Honmachi, Kyoto 606-01, JAPAN
{vasily,tamaru}@kuee.kyoto-u.ac.jp

ABSTRACT

This paper proposes a novel array architecture for full-search block matching motion estimation. The design efforts are focused on transforming the array computation in a way that minimizes the memory and I/O costs while satisfying the highest throughput requirements. Compared with the existing architectures, this one ensures feasible solutions for the HDTV picture format with twice lower memory requirements, minimal I/O pin count and 100% processor utilization. The architecture features regular and simple interconnects and is quite suitable for VLSI implementation.

1. INTRODUCTION

1.1. Motivation

Full Search Block Matching (FBM) is the basic algorithm adopted for video coding applications. Having successive video frames divided into blocks of $(N \times N)$ pixels, the FBM determines a *displacement vector* for every reference block in the current frame, by comparing it with all *candidate blocks* within the search area of size $(N+2p)^2$ surrounding the position of the reference block in the previous frame. The position (m,n) of a candidate block that results in the minimum distortion defines the motion vector v:

$$Z(m,n) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |y(i+m,j+n) - x(i,j)|, \quad -p \le m,n \le p-1 \quad (1)$$

$$v = \arg \min_{-p \le m,n \le p} Z(m,n) \quad (2)$$

The FBM algorithm provides optimal precision, regular data flow as well as higher parallelism, a characteristic that is advantageous for VLSI implementation. However, it is an extremely time consuming process, since $(2p)^2$ comparisons have to be computed for each of the displacement vectors in a frame. If a frame has 720×576 pixels (standard TV), and a frame rate of 25Hz, over 900 million operations on 6-16-bit data are required for $p = 8$, $N = 16$.

A number of hardware architectures have been proposed to cope with real time and high volume requirements. References [1-3] give good surveys of these. These architectures make use of massive pipelining and parallel processing which are provided by systolic arrays [4,7-10], linear arrays[6] or tree-like structures[5]. Despite differences, all of the architectures one feature in common: they all assume that image data is stored in a memory external to the processing array. Therefore the architectural solutions are focused on the data flow within processor element (PE) array to keep the PE's as busy as possible and at the same time minimize the required I/O bandwidth. In order to reduce the I/O count, the architectures include either an "on-chip" RAM memory to store the search area / reference block or a large number of line buffers or pipelined register chains to broadcast the data between the PEs during the computational process. As result, the total memory used for storing the search area / reference block is duplicated in the motion estimation system, increasing its size. Since all the existing architectures require two separate memories for storing the current frame and the previous frame of 900 KByte each (for the HDTV format), more than 70% of the total system area is occupied by memory units! Hence, efficient memory utilization becomes one of the most important design problems, especially for portable applications.

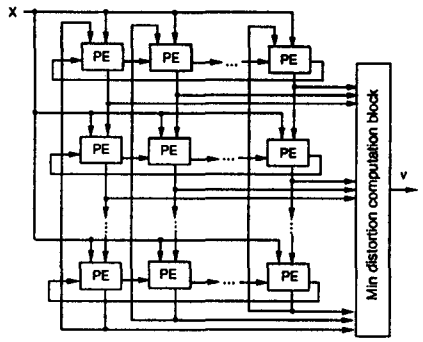


Figure 1. The proposed architecture

1.2. Contribution

In this paper, we propose a new distributed memory architecture for full-search block matching motion estimation. In our design, explicit provisions have been made to keep the total memory size and the number of transfers as small as possible, while maintaining the high throughput that is required for HDTV applications. Compared to previously proposed architectures, our architecture neither requires an external memory nor large register banks for data storage and transfer; it ensures minimal I/O bandwidth, provides 100% processor utilization and is linearly scalable. It features a regular and simple interconnect scheme and is suitable for VLSI implementation.

2. THE ARCHITECTURE

The proposed architecture consists of a two-dimensional mesh array of $(2p)^2$ PEs with wrap-around connections and the minimum displacement block (MDB) at the output of the array, as it illustrated in Figure 1. The mesh array computes equation (1) while the MDB implements (2). In the sequel, we will assume that $p = N/2$ which is almost always acceptable in practice.

Two main features distinguish our architecture from other 2-D mesh-arrays: (1) non-overlapping memory distribution between PEs; (2) dynamic memory sharing between current image and previous image.

2.1. Memory distribution

The system memory is distributed between the PEs such a way that each PE is given a dedicated portion of the search area. The data

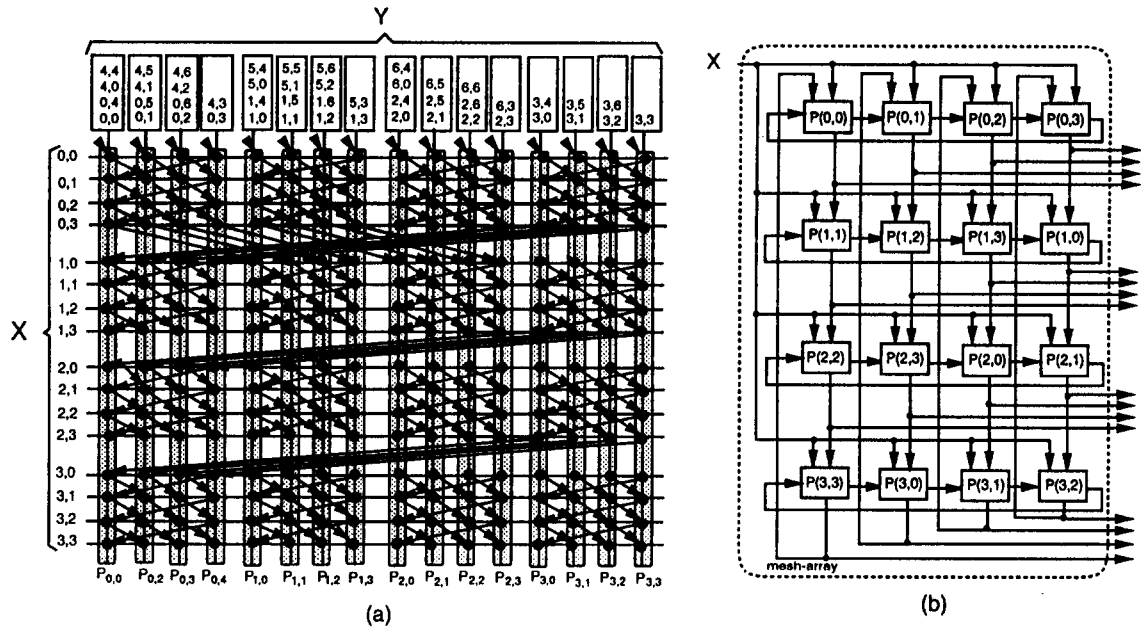


Figure 2. Hardware mapping for $N = 4$ and $p = 2$: (a) grouping of computations and data; (b) allocation of the groups in array

to be processed by the $PE(n, m)$ is stored in its local memory, $LM(m, n)$. In order to obtain a non-overlapping data distribution over the LMs, we group the computations that are executed on the same search area pixels or on pixels whose coordinates differ by $q \times (2p), (q = 1, 2, \ldots)$ and then map each group to a proper PE of the array. Figure 2(a) illustrates the assignment for N=4 and p=2. Here, vertical lines denote the search area pixels, horizontal lines denote the current block pixels, black circles depict the AD-operators, patterns show the PEs, rectangles represent contents of the LMs in the corresponding PEs. Since, there is no data overlap between the LMs, matching of all the search area pixels against a reference block pixel can be done in parallel. This releases us from the burden of iterative search area broadcast (another big limitation of other architectures) and allows us to reduce both the number of latches in the design and, which is very important, power consumption. In general, the number of transfers is reduced by a factor of $(2p)^2$. The only data which needs to be moved between the PEs is the accumulated AD-term. So, the number of wiring interconnections which have to be routed at the layout stage is also decreased. As result, the VLSI routing becomes regular and simple.

In addition, explicit provisions are made during hardware mapping to localize the PE interconnections in the mesh. Due to associativity of summation (1), a group of computations, $P(m, n)$, can be assigned to any PE in the row m. However, the closest neighboring connections emerge only if the assignment is done in a "left-rotate" fashion, that is by mapping the group $P(m, n)$ to the element $(m, n - m)$, as it shown in Fig.2(b). In this way, a regular and simple VLSI implementation is ensured.

2.2. Memory sharing

In order to efficiently utilize the memory space, we apply a gradual memory renewal concept, when pixels of the current frame replace pixels of the previous frame that are no longer needed. At every moment in time, pixels of a certain region of the previous frame are being read by the PEs, while another region is being overwritten with new pixels. Figure 3 illustrates this concept for two adjacent reference blocks (labeled with thick solid lines) of size $N \times N$ and displacement p. Due to displacement in the search areas for these blocks, the region patterned by black becomes vacant after processing of the block 1. In case when $N \leq 2p$, the region size

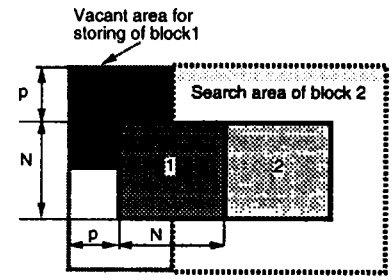


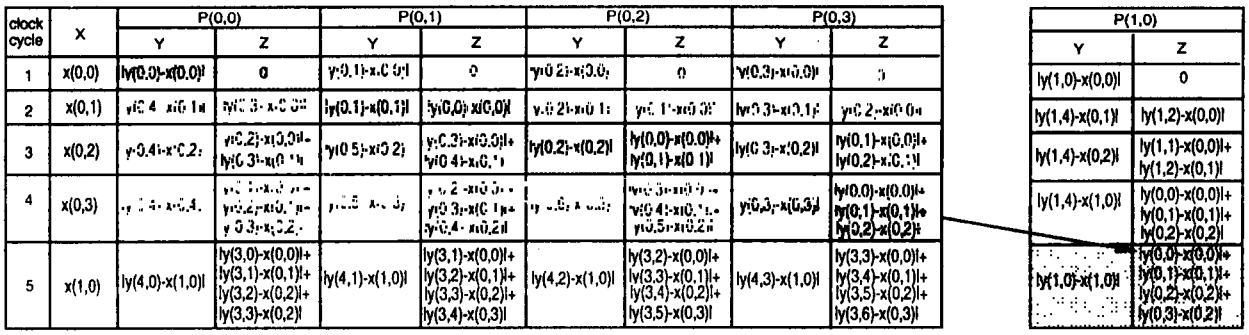
Figure 3. An illustration of the memory sharing concept.

becomes larger than N^2. Hence, memory allocated for storing this part of the previous frame can be efficiently used for storing the current block data without degrading the quality of the algorithm. As result, no other memory units are needed except input buffers to store N rows of the current frame.

In the proposed architecture, motion estimation is performed block by block in a raster scan order. Fortunately, for the block size of $N \times N$ pixels and the search range of $p = N/2$ pixels, the data required for the adjacent blocks can share the same memory and need not to be loaded. Figure 4 shows a general memory mapping concept. The bold lines in this figure define the pixels belonging to the search area of the first block, dotted lines define the pixels covered by the search area of the second block. As can be seen, the local memories are of different sizes, with the largest of the PE(0,0). Note, that for the standard TV format of 720 x 576 pixels, N=16, p=8, the size of the largest memory unit is 3 KByte only, while majority of the LMs are of 1.5 KByte each.

2.3. PE organization

Figure 5 shows the internal structure of a PE. Each PE is divided in a two stage pipeline. The first stage is constructed by Absolute Difference Calculator (ADC) and the second by the adder and RAM. The ADC computes the absolute difference (AD) between the search pixel datum, $y(m + i, n + j)$, stored in the register (y) and the reference pixel, $x(i, j)$, fed from the input bus, X. After



clock cycle	X	P(0,0)		P(0,1)		P(0,2)		P(0,3)		P(1,0)	
		Y	Z	Y	Z	Y	Z	Y	Z	Y	Z
1	x(0,0)	|y(0,0)-x(0,0)|	0							|y(1,0)-x(0,0)|	0
2	x(0,1)			|y(0,1)-x(0,1)|	|y(0,0)-x(0,0)|					|y(1,4)-x(0,1)|	|y(1,2)-x(0,0)|
3	x(0,2)					|y(0,2)-x(0,2)|	|y(0,0)-x(0,0)|+ |y(0,1)-x(0,1)|			|y(1,4)-x(0,2)|	|y(1,1)-x(0,0)|+ |y(1,2)-x(0,1)|
4	x(0,3)									|y(1,4)-x(1,0)|	|y(0,0)-x(0,0)|+ |y(0,1)-x(0,1)|+ |y(0,2)-x(0,2)|
5	x(1,0)	|y(4,0)-x(1,0)|	|y(3,0)-x(0,0)|+ |y(3,1)-x(0,1)|+ |y(3,2)-x(0,2)|+ |y(3,3)-x(0,2)|	|y(4,1)-x(1,0)|	|y(3,1)-x(0,0)|+ |y(3,2)-x(0,1)|+ |y(3,3)-x(0,2)|+ |y(3,4)-x(0,3)|	|y(4,2)-x(1,0)|	|y(3,2)-x(0,0)|+ |y(3,3)-x(0,1)|+ |y(3,4)-x(0,2)|+ |y(3,5)-x(0,3)|	|y(4,3)-x(1,0)|	|y(3,3)-x(0,0)|+ |y(3,4)-x(0,1)|+ |y(3,5)-x(0,2)|+ |y(3,6)-x(0,3)|	|y(1,0)-x(1,0)|	|y(0,0)-x(0,0)|+ |y(0,1)-x(0,1)|+ |y(0,2)-x(0,2)|+ |y(0,3)-x(0,2)|

Figure 6. Data flow in the 4x4 array architecture

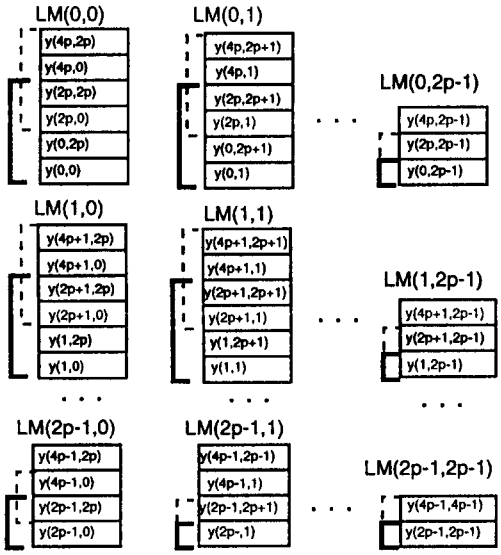


Figure 4. General data distribution between the LMs

being delayed one clock cycle by latch, Y, the AD value is added to the partial sum stored in the register Z, and the result is written to the register Z of its right or down-neighbor, depending on the control of the multiplexor (mx). At the same stage, the RAM reads its address (A) to the register y, or receives in (A) data stored in the FIFO file x. Due to regular data flow in the diagonal direction (from bottom-left to the top-right), no complicated address generators are needed.

2.4. The system operation

We assume that before a new current block processing, all PEs have zeros in registers Z and initial search area pixels $y(0+m, 0+n), -p \leq m, n \leq p-1$, in the registers y. The operation begins with broadcasting a current block pixel, $x(i,j)$, to all the PEs.

In the clock cycle (t) each PE executes in parallel three disjoint operations: (1) computes the AD value, $Y_t = |y_{t-1} - x(i,j)|$, to be stored in its register Y; (2) adds the content of its register Z to the AD value, Y_{t-1}, calculated at the previous clock cycle $(t-1)$ and writes the result to Z registers of its right or down-neighboring PEs; (3) reads a new search area pixel to the register y, or writes the pixel from the FIFO file x to the RAM. Since we assume that the current block pixels enter the system in a row-based fashion, the right neighbors are selected each of $[0, N-1]$ cycles, while the down-neighbor is chose at the end of the row, that is in each N cycle. Thus during the N steps, the $(2p)^2$ absolute differences calculated in the first clock step are iteratively moved along the

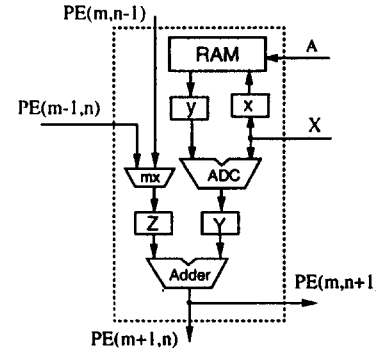


Figure 5. The PE structure

rows N times; each time accumulating the result computed at the PE they visit.

Figure 6 illustrates the data flow in array Fig.2(b) by outlining the results of computations in the 5 elements $P(0,0), P(0,1), P(0,2), P(0,3)$ and $P(1,0)$ during the first 5 clock steps. The column (X) in this figure shows the current block data. The PE columns define contents of the registers Y and Z for each PE. As can be seen, the accumulation process is dynamic; unlike those of other architectures. The sum, $Y+Z$, computed in the PE(0,0) at the clock cycle 1 travels the PEs in the right direction, each time accumulating a new AD-term. (The patterns show the operands added in the corresponding clock step). At the end of cycle 3, the PEs write the sums to the registers Z of their down-adjacent PEs. Thus the results accumulated in PEs of the first row will be stored in the Z registers of the second row, while the Z registers of $P(0,0), P(0,1), P(0,2), P(0,3)$ will receive data accumulated in $P(3,0), P(3,1), P(3,2), P(3,3)$, respectively.

Generally, after N^2 clock cycles, all pixels of the current block are processed and $(2p)^2$ distance measures are available simultaneously. The minimum of them is determined in the min.displacement block in parallel to processing of a new current block in the array.

Figure 7 shows the distribution of memory accesses in the PEs during the computations. The first two columns in this figure are the same, as in the Fig.6, the P-columns shows data to be read from or write to (grey patterns) the local memory of the $P(m,n)$ in the corresponding clock cycle. Due to RAM's inability to perform simultaneous read/write, the current block data is written to the memory with a delay to its arrival time on the system input. For example, the $x(0,2)$ pixel arrives in clock cycle 3, but the $PE(0,2)$ writes it to memory only in the clock cycle 4. Every PE makes one write and several read accesses to its RAM during one block processing. However, the overall number of accesses to the distributed memory is quite low, yet all the PEs are 100% utilized.

Table 1. Comparison results: Video Format CCIR Rec.601, $N = 16, p = 8$

Reference	$\#PE$	Memory size (KB)	# I/O	# clock cycles per block		per frame	f_{max} /sec
[1]	256	2(450)	136	$2p(N+2p-1)$	496	803520	49
[4]	256	2(450)	16	$(N+2p-1)^2$	961	1,556820	25
[5]	512	2(450)	2,056	$(2p)^2+log_2N+2$	262	433,872	92
[6]	16	2(450)	32	$N^2(2p)$	4,096	6,635,520	6
[7]	256	2(450)	136	$N^2+2p(N+2p-1)+N$	768	1,244,160	32
[8]	256	2(450)	32	N^2	256	423,936	94
Our	256	455	16	N^2	256	414,720	96

Table 2. Comparison results: HDTV Format, $N = 16, p = 16$

Type	$\#PE$	Memory size (KB)	# I/O	# clock cycles per block	per frame	f_{max} /sec
[1]	256	2(900)	136	1,504	5,414,400	7
[4]	256	2(900)	16	2,209	7,952,400	5
[5]	512	2(900)	2,056	1,034	3,686,410	10
[6]	16	2(900)	56	4,096	14,745,600	3
[7]	256	2(900)	136	1,776	6,393,600	6
[8]	1024	2(900)	80	256	933,120	42
Our	1024	924	16	256	921,600	42

3. COMPARISON

The comparison of the proposed architecture to the other existing architectures is presented in Table 1-2. The video broadcast format (CCIR Rec.601) with $N = 16$, $p = 8$ and the HDTV picture format with $N = 16$, $p = 16$, and clock period of 25 ns are used. In Table 1,2, the number of PEs, the total memory size, the I/O count, the number of clock cycles required to estimate a reference block and a frame, and the maximum number of frames to be estimated per second (f_{max}/s) have been compared based on the technique proposed in [8].

Comparing with other architectures, our architecture ensures feasible solutions for the HDTV picture format with the minimum number of I/O pins and as twice as less memory size. In contrast to [8], where each first block in a row is evaluated two times slower than the others, our architecture processes all the image blocks with equal speed (256 clock cycles per block). As result, it can estimate the maximum number of frames per second.

Preliminary layouts indicate that the physical area of a motion estimation chip with $N = 16$, $p = 8$ is approximately 2/3 smaller than the total area of a conventional motion estimation system. Detailed chip design is in progress.

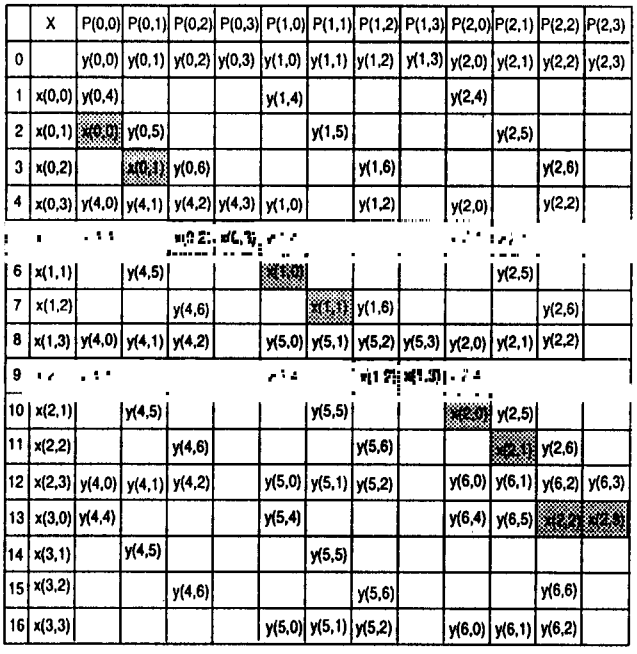


Figure 7. An illustration of read-write sequence in the array

REFERENCES

[1] T.Komarek and P.Pitsch, "Array architectures for block-matching algorithms", *IEEE Trans. CAS*, Vol.36, No.10, Oct. 1989, pp.1301-1308.

[2] P.Pirsch, N.Demassieux, W.Gehrke, "VLSI architectures for video compression", *Proceedings of the IEEE*, Vol.83, No.2, Feb. 1995, pp.220-246.

[3] M.Sung, "Algorithms and VLSI architectures for motion estimation", *VLSI Implementations for Image Communications*, P.Pirsch (Ed.), 1993, pp.251-2281.

[4] C.Hsieh and T.Lin, "VLSI Architecture for block-matching motion estimation algorithm", *IEEE Trans.CAS for Video Technology*, Vol.2, No.2, June 1992, pp.169-175.

[5] Y.Jehng, et al., "An efficient and simple VLSI tree architecture for the motion compensation block-matching algorithms", *IEEE Trans. Signal Process.*, Vol.41, Feb. 1993, pp.889-899.

[6] K.Yang, et al., "A family of VLSI designs for the motion compensation block-matching algorithm", *IEEE Trans. CAS*, Vol.36, Oct. 1989, pp.1317-1325.

[7] E.Chan, et al., "Motion estimation architecture for video compression", *IEEE Trans. Consumer Electron.*, Vol.39, No.3, Aug. 1993, pp.292-297.

[8] H.Yeo and Y.Hu, "A novel modular systolic array architectures for full-search block matching motion estimation", *IEEE Trans. CAS for Video Technology*, Vol.5, No.5, Oct. 1995, pp.407-416.

[9] J.Baek, et al, "A fast array architecture for block matching algorithm", *Proc. IEEE ISCAS*, 1994, pp.211-214.

[10] S.Pan, S.Chae, R.Park, "VLSI Architectures for block matching algorithm using systolic arrays", *IEEE Trans. CAS for Video Technology*, Vol.6, No.1, Feb. 1996, pp.67-73.