Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A parallel memory architecture for video coding[*]

Jian-ying PENG[†], Xiao-lang YAN[†‡], De-xian LI, Li-zhong CHEN
(*Institute of VLSI Design, Zhejiang University, Hangzhou 310027, China*)
[†]E-mail: pengjy@vlsi.zju.edu.cn; yan@vlsi.zju.edu.cn

Received Jan. 17, 2008; revision accepted June 8, 2008; CrossCheck deposited Nov. 10, 2008

Abstract: To efficiently exploit the performance of single instruction multiple data (SIMD) architectures for video coding, a parallel memory architecture with power-of-two memory modules is proposed. It employs two novel skewing schemes to provide conflict-free access to adjacent elements (8-bit and 16-bit data types) or with power-of-two intervals in both horizontal and vertical directions, which were not possible in previous parallel memory architectures. Area consumptions and delay estimations are given respectively with 4, 8 and 16 memory modules. Under a 0.18-μm CMOS technology, the synthesis results show that the proposed system can achieve 230 MHz clock frequency with 16 memory modules at the cost of 19k gates when read and write latencies are 3 and 2 clock cycles, respectively. We implement the proposed parallel memory architecture on a video signal processor (VSP). The results show that VSP enhanced with the proposed architecture achieves 1.28× speedups for H.264 real-time decoding.

Key words: Single instruction multiple data (SIMD), Video coding, Parallel memory, Skewing scheme
doi:10.1631/jzus.A0820052 **Document code:** A **CLC number:** TN47

INTRODUCTION

During the last decade we have been witnessing the high development pace of video coding algorithms, which results in a large number of latest standards, such as H.264/AVC, VC-1 and AVS. As well as the advances in VLSI technology, modern media processors, digital signal processors (DSPs) and general-purpose processors (GPPs) with instruction extensions can provide a programmable environment for challenging real-time video processing. Single instruction multiple data (SIMD) architectures exploit inherent data level parallelism and achieve high performance with low hardware cost and control complexity in video processing (Kozyrakis and Patterson, 2002). They use partitionable function units to operate on packed narrow subwords. As an example, a 64-bit register can be treated as two 32-bit, four 16-bit, or eight 8-bit segments.

Traditional SIMD processors employ the word-addressable memory architecture for simple implementation and low cost. They straightforwardly access data elements located next to each other in a correct order according to word boundaries. When the wanted data are not word aligned, two memory loads, two shifts and one logic "or" operation are needed. But motion estimation, the most computation intensive part of video coding, demands unaligned memory access. Many other applications, such as discrete cosine transform (DCT), matrix computation, require stride memory access. Up to 41% of the total instructions of some processors are used for bringing requested data in the amenable SIMD fashion (Talla *et al.*, 2003).

To decrease such overhead of rearrangement operations, stride memory access is supported in modern multimedia architectures, such as matrix oriented multimedia (MOM) (Corbal *et al.*, 1999), complex streamed instruction (CSI) (Cheresiz *et al.*, 2005), Imagine (Khailany *et al.*, 2001) and Vector Intelligent RAM (VIRAM) (Kozyrakis and Patterson, 2003). For example, MOM allows arbitrary stride between consecutive rows and the subwords in a row have to be adjacent. However, none of them utilizes the parallel memory architecture. They support stride

[‡] Corresponding author
[*] Project (No. 2005AA1Z1271) supported by the Hi-Tech Research and Development Program (863) of China

and possibly some irregular accesses by collecting data to dense groups in an additional buffer (Aho *et al.*, 2007). But parallel memories access only the required data in parallel without extra buffering or additional memory accesses. The parallel memory architecture with versatile access formats provides a high bandwidth of internal data memory for parallel processors, which is especially useful for subword SIMD parallel processing. Therefore, an efficient parallel memory architecture is essential for SIMD architectures in video processing.

In this paper, we propose a novel parallel memory architecture with power-of-two memory modules, which provides required parallel access formats of video coding. It supports conflict-free access with power-of-two strides in 2D directions for 8-bit pixels and 16-bit coefficients in video coding. The organization of the paper is as follows. Section 2 discusses the related works of parallel memory. Demanded access formats of video coding are presented in Section 3. Proposed skewing schemes are described in Section 4 whereas the proposed parallel memory architecture is depicted in detail in Section 5. Section 6 gives the ASIC results of hardware implementation with memory module numbers 4, 8 and 16, and comparisons with some reference architectures. The performance comparisons are shown in Section 7. Finally, Section 8 concludes the paper.

RELATED WORKS

Parallel memory architectures have been under active research for a long time (Sohi, 1993). The essential components of a parallel memory architecture are depicted in Fig.1. The address computation unit is hardwired to utilize predetermined module assignment function and address assignment function. The module assignment function (or skewing scheme) determines the distribution of data to the modules, whereas the address assignment function allocates the distribution of data to addresses within the memory modules. Using control signals, such as the access format and the position of the starting point, the address computation unit calculates the physical addresses and aligns them with right memory modules (S_0, S_1, …, S_{M-1}). The data permutation network shuffles the requested data into a correct order according to the module assignment function.

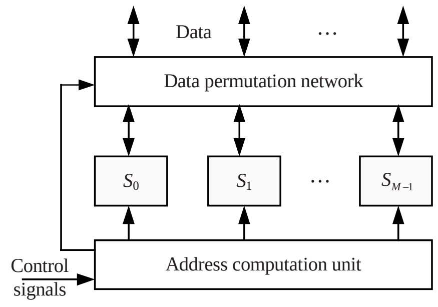


Fig.1 Diagram of parallel memory architecture

Researchers have proposed many skewing schemes to support versatile parallel access formats. For example, Park (2004) employed a prime number of memory modules to access data elements within a block, a row, a column, and a forward or backward diagonal subarray simultaneously. However, a prime number of memory modules complicated the hardware implementation and took redundant memory storage space. Ideally, the number of memory modules equals that of elements inside access formats. Processors usually employ parallel data path of power-of-two bytes, so we restrict our discussions to parallel memory schemes with power-of-two memory modules ($M=2^n$).

The XOR-scheme was generalized by Frailong *et al.*(1985), which computed a physical address within a module as a dot product of a logical address of a data element and a transformation matrix, described as

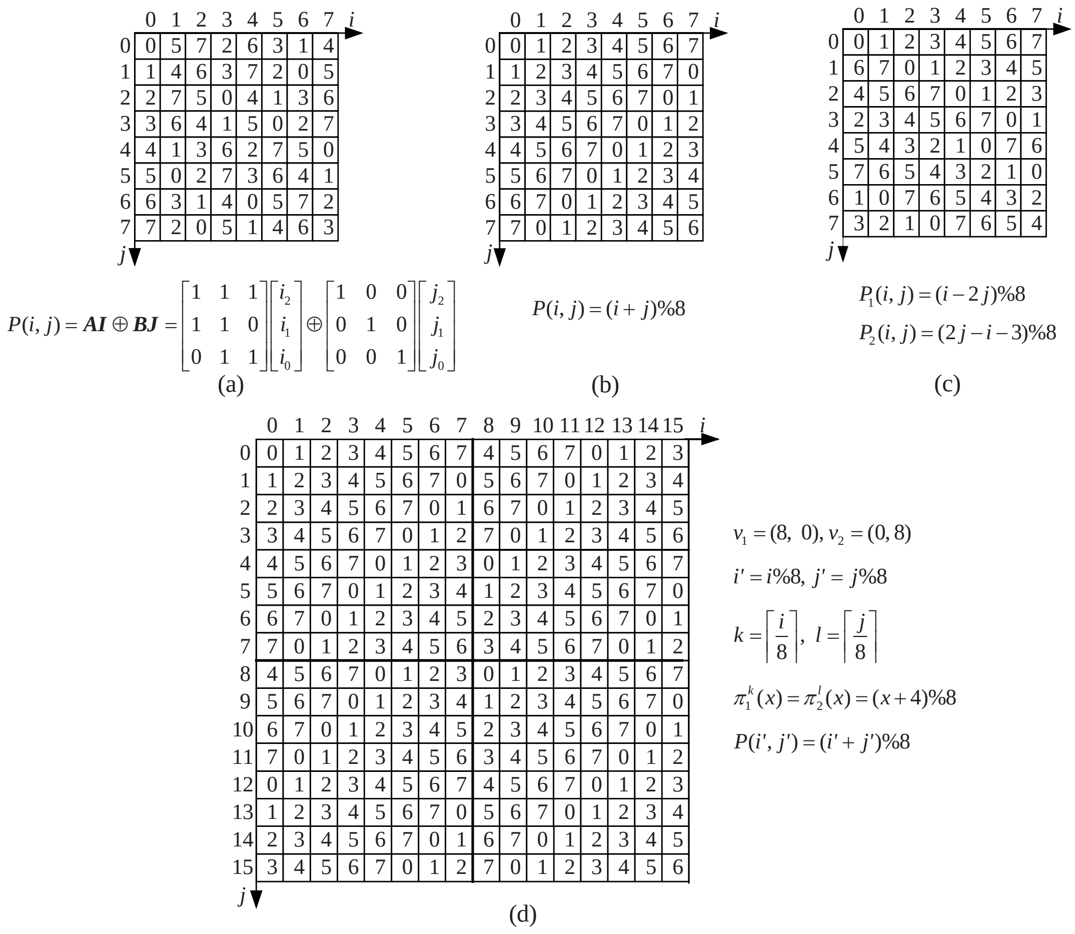
$$P(i,j)=\boldsymbol{AI}\oplus\boldsymbol{BJ}, \text{when } \boldsymbol{I}=(i\%2^u)_2, \boldsymbol{J}=(j\%2^v)_2, \quad (1)$$

where the number of memory modules is $M=2^n$, $\boldsymbol{A}$ and $\boldsymbol{B}$ are $n\times u$ and $n\times v$ binary matrices, respectively. Then calculations are performed by simple bitwise “XOR” and “AND” operations. An example of the XOR scheme is shown in Fig.2a.

Another class of frequently used parallel schemes is the linear skewing scheme, which was first introduced by Budnik and Kuck (1971). Linear schemes can be expressed as

$$P(i,j)=(ai+bj)\%M, \quad (2)$$

where a and b are constants. An example, when $a=b=1$ and $M=8$, is presented in Fig.2b. Linear functions have the important property—isotropic (Gossel *et al.*, 1994), which reduces the number of memory module permutations inside the access formats and



(a)

j \ i	0	1	2	3	4	5	6	7
0	0	5	7	2	6	3	1	4
1	1	4	6	3	7	2	0	5
2	2	7	5	0	4	1	3	6
3	3	6	4	1	5	0	2	7
4	4	1	3	6	2	7	5	0
5	5	0	2	7	3	6	4	1
6	6	3	1	4	0	5	7	2
7	7	2	0	5	1	4	6	3

$$P(i,j)=\boldsymbol{AI}\oplus\boldsymbol{BJ}=\begin{bmatrix}1&1&1\\1&1&0\\0&1&1\end{bmatrix}\begin{bmatrix}i_2\\i_1\\i_0\end{bmatrix}\oplus\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}j_2\\j_1\\j_0\end{bmatrix}$$

(b)

j \ i	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

$$P(i,j)=(i+j)\%8$$

(c)

j \ i	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	6	7	0	1	2	3	4	5
2	4	5	6	7	0	1	2	3
3	2	3	4	5	6	7	0	1
4	5	4	3	2	1	0	7	6
5	7	6	5	4	3	2	1	0
6	1	0	7	6	5	4	3	2
7	3	2	1	0	7	6	5	4

$$P_1(i,j)=(i-2j)\%8$$
$$P_2(i,j)=(2j-i-3)\%8$$

(d)

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

$$v_1=(8,\ 0),\ v_2=(0,8)$$
$$i'=i\%8,\ j'=j\%8$$
$$k=\left\lceil\frac{i}{8}\right\rceil,\ l=\left\lceil\frac{j}{8}\right\rceil$$
$$\pi_1^k(x)=\pi_2^l(x)=(x+4)\%8$$
$$P(i',j')=(i'+j')\%8$$

Fig.2 Previous skewing schemes. (a) XOR scheme; (b) Linear scheme; (c) Multi-skewing scheme; (d) Multi-periodic diamond scheme

simplifies the address calculation circuit and the permutation network. But, it has been proven that it is not possible for a linear function to obtain conflict-free access for rows, columns, and forward and backward diagonals without redundant memory modules. A multi-skewing method was proposed by Deb (1996), which employed several linear module assignment functions to provide conflict-free access for various access formats of an $M\times M$ matrix, as shown in Fig.2c.

The multi-periodic diamond scheme was proposed in (Gossel *et al.*, 1994) and described by

$$P(i,j)=P(i'+k\alpha,\ j'+l\beta)=\pi^k\pi^l P(i',j'),\quad (3)$$

where $P(i',j')$ determines the memory mapping of the basic domain. The position (i,j) in other blocks is permutated by the functions π^k and π^l along the i- and j-axis, respectively. For example, in Fig.2d, the basic domain is the upper-left 8×8 block, and the other blocks are permutations of that block. Well chosen permutations enable the needed parallel access formats.

It is well known that access format combinations are quite limited for a single module assignment function without data redundancy and extra memory space (Gossel *et al.*, 1994). However, solutions might be found for specific applications, such as placement restrictions and several module assignment functions. Aho *et al.*(2004) presented the configurable parallel memory architecture (CPMA) that allowed several data access formats and skewing schemes to be implemented by a single hardware when the number of memory modules is arbitrary—But CPMA is too complicated for a specific application, while our work only concentrates on video coding. They proposed a configurable parallel memory system with arbitrary stride accessibility for multimedia processing (Aho *et* *al.*, 2007). Parallel memory schemes for H.263 encoder are proposed in (Tanskanen *et al.*, 2000), but they did not give the implementation architecture. Tanskanen *et al.*(2004; 2005) presented two parallel memory architectures for video coding, which employed several XOR-schemes. But all the previous designs only support parallel access formats of 8-bit subword, and can not provide stride accessibility of 16-bit element, which is often demanded in video processing. Moreover, they just support 1D stride accesses, which are not suitable for 2D video coding. Based on the analysis of access formats in video coding, this paper presents a novel parallel memory architecture allowing conflict-free stride accesses in both horizontal and vertical directions.

ACCESS FORMATS OF VIDEO CODING

Video coding proceeds macroblock by macroblock, and macroblocks are further constructed by tree-based blocks of different sizes, such as 16×8, 8×8, 8×4, and 4×4. There are two data types of video processing: byte (8-bit) for pixels and half-word (16-bit) for coefficients. Most pixel computation kernels in video coding, including motion estimation, compensation, padding, etc., need to access adjacent M pixels in horizontal or vertical directions without restricted start point placement. Meanwhile, alternate pixels in a row or column are demanded to be accessed in fractional pixel interpolation (Tanskanen *et al.*, 2000). In the 'in-place' computations of 2D orthogonal transforms, conflict-free parallel accesses to 16-bit elements with power-of-two strides in both a row and a column are required (Trenas *et al.*, 1998). Desired parallel access formats of video coding are shown in Fig.3. The memory module is to be accessed in a 2D scanning field. The pointed square presents the starting point location. UN stands for placement unrestricted. But shown as (5)~(8) in Fig.3, parallel accesses with power-of-two strides (2^l, $l\neq0$) from 2 to M, need restrictions that the horizontal and vertical coordinates in the local $M\times M$ matrix should be less than 2^l respectively for row and column accesses.

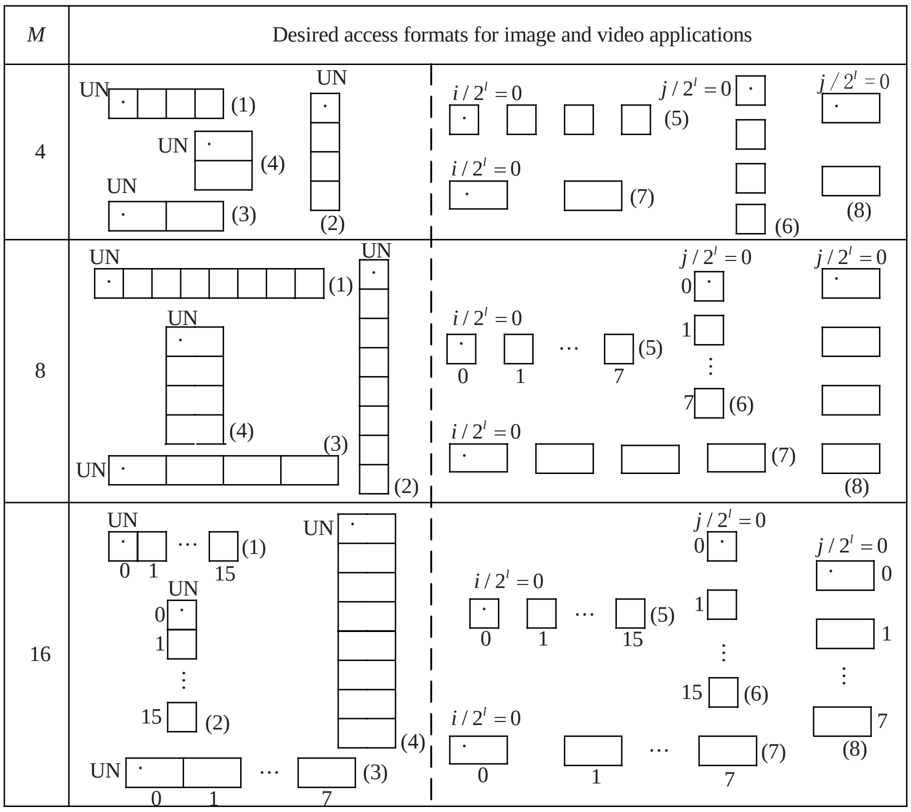


Fig.3 Desired parallel access formats of video coding (M=4, 8 and 16). The memory module is to be accessed in a 2D scanning field. The pointed square presents the starting point location. UN stands for placement unrestricted

SKEWING SCHEMES FOR VIDEO CODING

Since all the stream data of video coding are stored in the single data memory, traditional 2D addressing mode just using the coordinates (*i*,*j*) to locate a position is not appropriate. A new 2D parallel memory access pattern is proposed in Fig.4. Two parameters determine the position of the starting point: (1) *base*, the base address of a data block, which is restricted to multiples of *M*; (2) 2D offsets in horizontal and vertical directions (*ox*,*oy*) between the base address and the starting point. Other definitions are given as follows:

xstride: the horizontal stride in bytes between neighbor elements. It should be power-of-two.

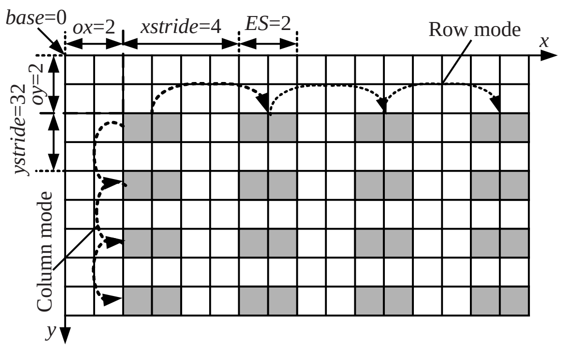


Fig.4 Proposed 2D parallel memory access pattern

ystride: the stride in bytes between consecutive rows. For hardware simplicity, it is restricted to be multiples of *M* and be power of two.

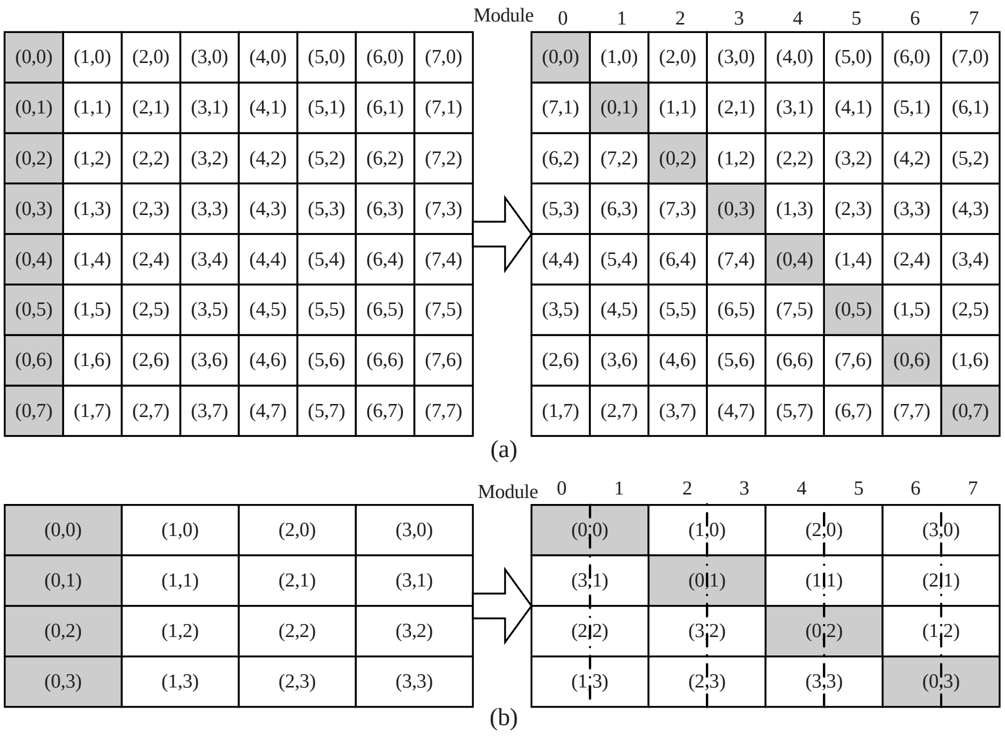
ES: element size, 1 for 8-bit pixels and 2 for 16-bit coefficients.

A/*S*: 0 for accessing adjacent elements, when *l*=0; 1 for accessing sparse elements with power-of-two strides 2^l.

column: 0 for row access and 1 for column access.

As one single module assignment function cannot support all parallel access formats shown in Fig.3, we propose two parallel schemes (named A-scheme and S-scheme) based on the popular linear shift/rotate scheme (Li *et al.*, 2005) to meet the requirements of video coding. As shown in Fig.5, pixels (*ES*=1) within an 8×8 block and coefficients (*ES*=2) within a 4×4 block are mapped in the shift/rotate manner, in which the *i*th row of data is sifted/rotated by *i*×*ES* bytes, respectively. The memory module assignment function for the position (*x*, *y*) based on byte in an *M*×*M* matrix (*ES*=1) or an *M*/2×*M*/2 matrix (*ES*=2) is

$$m(x,y) = (x + y \cdot ES)\%M. \tag{4}$$



(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)	(6,0)	(7,0)
(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)	(7,1)
(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)	(7,2)
(0,3)	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)	(7,3)
(0,4)	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)	(7,4)
(0,5)	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)	(7,5)
(0,6)	(1,6)	(2,6)	(3,6)	(4,6)	(5,6)	(6,6)	(7,6)
(0,7)	(1,7)	(2,7)	(3,7)	(4,7)	(5,7)	(6,7)	(7,7)

Module 0	1	2	3	4	5	6	7
(0,0)	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)	(6,0)	(7,0)
(7,1)	(0,1)	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)
(6,2)	(7,2)	(0,2)	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)
(5,3)	(6,3)	(7,3)	(0,3)	(1,3)	(2,3)	(3,3)	(4,3)
(4,4)	(5,4)	(6,4)	(7,4)	(0,4)	(1,4)	(2,4)	(3,4)
(3,5)	(4,5)	(5,5)	(6,5)	(7,5)	(0,5)	(1,5)	(2,5)
(2,6)	(3,6)	(4,6)	(5,6)	(6,6)	(7,6)	(0,6)	(1,6)
(1,7)	(2,7)	(3,7)	(4,7)	(5,7)	(6,7)	(7,7)	(0,7)

(a)

(0,0)	(1,0)	(2,0)	(3,0)
(0,1)	(1,1)	(2,1)	(3,1)
(0,2)	(1,2)	(2,2)	(3,2)
(0,3)	(1,3)	(2,3)	(3,3)

Module 0, 1	2, 3	4, 5	6, 7
(0,0)	(1,0)	(2,0)	(3,0)
(3,1)	(0,1)	(1,1)	(2,1)
(2,2)	(3,2)	(0,2)	(1,2)
(1,3)	(2,3)	(3,3)	(0,3)

(b)

Fig.5 Linear shift/rotate scheme with *M*=8. (a) 8×8 pixel block; (b) 4×4 coefficient block

By using this scheme, each data element belonging to a row or a column in the matrix is stored in the distinct modules. As a result, parallel memory accesses in horizontal and vertical directions can be supported.

Proposed A-scheme for adjacent access

To provide parallel access to adjacent both 8-bit and 16-bit elements in a row or a column of a $U\times V$ image matrix, as shown by (1)~(4) in Fig.3, the A-scheme is proposed. A-scheme partitions the image matrix into size $M\times M$ (ES=1) or $M/2\times M/2$ (ES=2) sub-matrices, and all sub-matrices employ the linear shift/rotate scheme, which is described as Eq.(4). Then the module assignment function for position (x,y) is given as

$$\begin{aligned}\phi(x,y)&=\phi(x_0+KM,y_0+jM)\\&=m(x_0,y_0)=(x_0+y_0)\%M,\end{aligned}\tag{5}$$

where (x_0,y_0) is the new coordinates in a local sub-matrix of the original position (x,y). A-scheme ϕ is periodic, as $\phi(x,y)=\phi(x+kM,y+jM)$, for $k, j=\ldots, -2, -1, 0, 1, 2, \ldots$. As an example, when M=8, the memory module distributions of a 16×16 pixel block (ES=1) and an 8×8 coefficient block (ES=2) using A-scheme are shown in Figs.6a and 6c, respectively.

In our defined 2D access pattern, the new coordinates in local sub-matrix of i-byte ($0\le i<M$) element from the starting position *base* and (ox,oy) is represented as followings:

$$\begin{aligned}&\text{row: }\begin{cases}x_i=(ox+i)\%M,\\y_i=oy\%M,\end{cases}\\&\text{column: }\begin{cases}x_i=(ox+i\%ES)\%M,\\y_i=(oy+i/ES)\%M.\end{cases}\end{aligned}\tag{6}$$

So the module assignment function of A-scheme can be deduced by combining Eqs.(5) and (6) as:

$$m(i)=\begin{cases}[(ox+i)\%M+(oy\%M)\cdot ES]\%M, & \text{row,}\\ [(ox+i\%ES)\%M+(oy+i/ES)\cdot ES]\%M, & \text{column.}\end{cases}\tag{7}$$

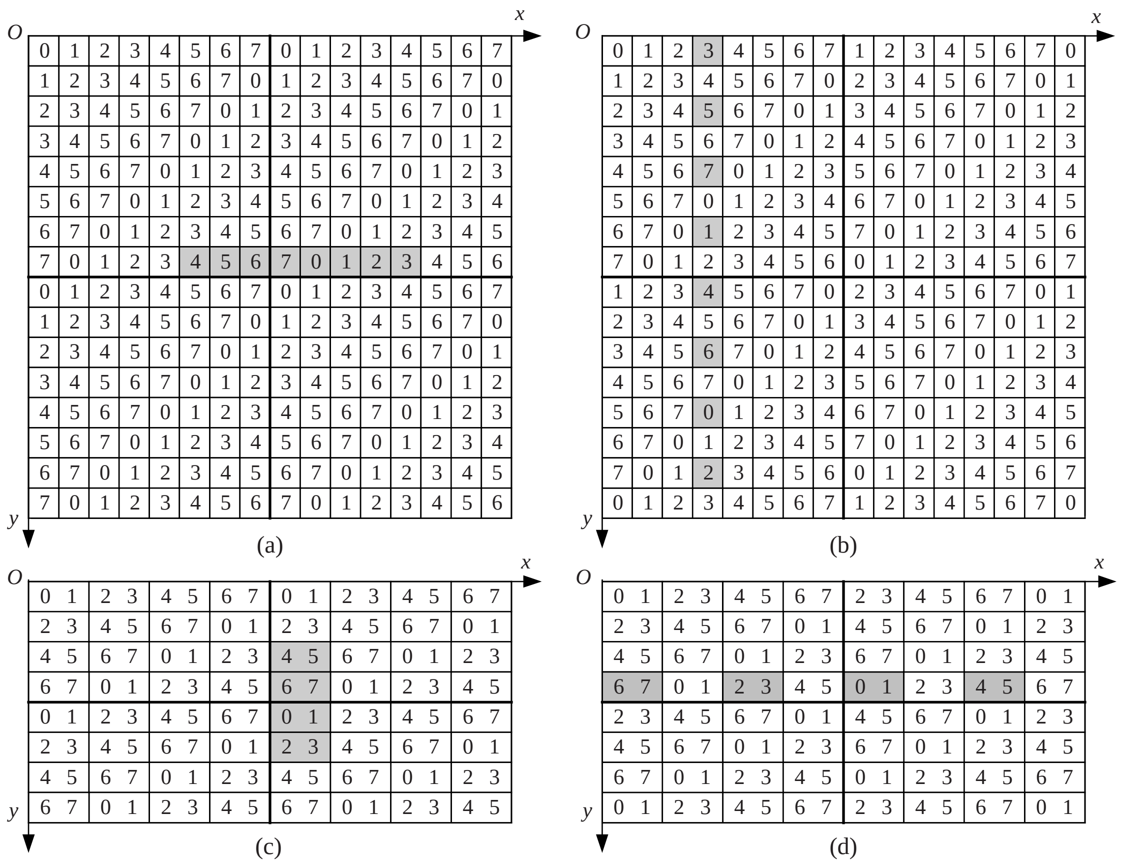


Fig.6 Module distributions of A- and S-schemes (*M*=8). (a) 16×16 pixel block using A-scheme; (b) 16×16 pixel block using S-scheme; (c) 8×8 coefficient block using A-scheme; (d) 8×8 coefficient block using S-scheme

Proposed S-scheme for sparse access

We propose another parallel scheme, S-scheme, to support accesses with power-of-two intervals from 2 to M between two neighbor elements in a row or a column, as depicted by (5)~(8) in Fig.3. The image matrix is also divided into blocks of size $M\times M$ (ES=1) or $M/2\times M/2$ (ES=2). The top left block ($x,y<M$ when ES=1; $x<M$, $y<M/2$ when ES=2) applies the basic linear shift/rotate scheme as Eq.(4). The skewing scheme for other sub-matrices at position (s,t) is defined as

$$\varphi(x,y)=[m(x_0+y_0)+(s+t)\%M\cdot ES]\%M, \quad (8)$$

where (x_0,y_0) are the new coordinates in a local sub-matrix of the original position (x,y), $s=s_{k-1}+s_{k-2}+\ldots+s_1$ ($k=\lceil U/M\rceil$), $t=t_{j-1}+t_{j-2}+\ldots+t_1$ ($j=\lceil V/M\rceil$, ES=1; $j=\lceil 2V/M\rceil$, ES=2).

S-scheme is not periodic, but local skewing is linear and periodic. It permits simultaneous access to M-byte elements in a row or a column with power-of-two intervals from 2 to M. The initial position (x,y) must satisfy the following requirement:

$$\begin{aligned}\text{row: } & (x\%M)/2^l=0,\\ \text{column: } & (y\%M)/2^l=0.\end{aligned} \quad (9)$$

The memory module distributions of a 16×16 pixel block (ES=1) and an 8×8 coefficient block (ES=2) by S-scheme are given in Figs.6b and 6d, respectively. For example, in Fig.6b, we can access elements in a column, with initial position (0,3) and strides equal to 2. The coordinates of accessed elements are {(0,3), (2,3), (4,3), (6,3), (8,3), (10,3), (12,3), (14,3)}, which are assigned to the memory 3, 5, 7, 1, 4, 6, 0 and 2 respectively.

In the S-scheme, the new coordinates in the local sub-matrix of i-byte ($0\le i<M$) element from the starting position *base* and (ox,oy) are represented as

$$\begin{aligned}&\text{row: } \begin{cases} x_i=[ox+i\%ES+(i/ES)\cdot xstride]\%M,\\ y_i=oy\%M,\end{cases}\\ &\text{column: } \begin{cases} x_i=(ox+i\%ES)\%M,\\ y_i=(oy+i/ES)\%M.\end{cases}\end{aligned} \quad (10)$$

And the positions of the sub-matrix are

$$\begin{aligned}&\text{row: } \begin{cases} s_i=[ox+i\%ES+(i/ES)\cdot xstride]/M,\\ t_i=oy/M,\end{cases}\\ &\text{column: } \begin{cases} s_i=(ox+i\%ES)/M,\\ t_i=(oy+i/ES)/M.\end{cases}\end{aligned} \quad (11)$$

Therefore the module assignment function of S-scheme based on our defined 2D access pattern is

$$m(i)=[x_i+(y_i+s_i+t_i)\%M\cdot ES]\%M. \quad (12)$$

PROPOSED HARDWARE ARCHITECTURE

The block diagram of the proposed parallel memory architecture is shown in Fig.7. The input parameters including *base*, (*ox*,*oy*), *xstride*, *ystride*, *ES*, *A*/*S* and *column*. Instead of the bi-directional crossbar in Fig.1, two data permutation networks (DPNs) are employed to permute the written or read data according to the control signals from the address generation unit (AGU).

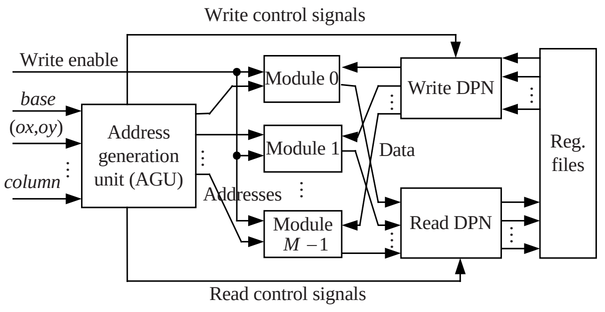


Fig.7 Block diagram of the proposed architecture

Fig.8 depicts a block diagram of AGU. The outputs of the unit are the physical addresses for each memory module and read and write control signals for the two DPNs. Inside AGU, the effective address unit (EAU) computes the effective addresses EA_i of each accessed data. The low n ($n=\log_2 M$) bits of EA_i determine the index of the assigned memory module for element i, and the remaining bits determine the real inside address within the assigned module. The permutation control unit (PCU) reorders the memory module numbers. According to these values, the address permutation network (APN) directs the physical addresses to the proper memory modules.

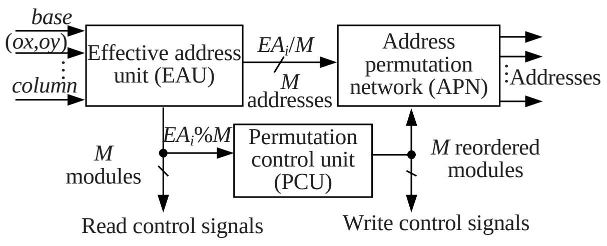


Fig.8 Block diagram of AGU

The detailed circuit of the EAU is shown in Fig.9. According to above discussion in Section 4, the effective address of each element EA_i ($0\le i<M$) is described as Eq.(13). EA_i is obtained by the addition of a pre-skewed base-stride (BS), the sum of strides in horizontal and vertical directions (S_i) and the memory-bank offsets (OF_i).

$$EA_i = BS + S_i + OF_i. \tag{13}$$

Fig.9a computes the first addendum term BS using the following relation: $BS=base+oy\times ystride$. Fig.9b shows the computation of S_i, described as

$$S_i=\begin{cases} ox+i\%ES+(i/ES)\cdot xstride, & \text{row,}\\ ox+i\%ES+(i/ES)\cdot ystride, & \text{column.}\end{cases} \tag{14}$$

The logic for computing the offset value OF_i is presented in Fig.9c. The computations of OF_i are different for A-scheme and S-scheme, represented by Eqs.(15) and (16), respectively:

$$OF_i=\begin{cases} y_i\cdot ES, & x_i+y_i<M,\\ -(M-y_i)\cdot ES, & x_i+y_i\ge M,\end{cases} \tag{15}$$

$$OF_i=\begin{cases} (y_i+s_i+t_i)\cdot ES, & x_i+y_i+s_i+t_i<M,\\ -(M-y_i-s_i-t_i)\cdot ES, & x_i+y_i+s_i+t_i\ge M,\end{cases} \tag{16}$$

where (x_i,y_i) and (s_i,t_i) are defined and calculated by Eq.(6), Eq.(10) and Eq.(11), respectively. Finally, the additions of effective addresses for M memory modules are shown in Fig.9d. Since the value of ES is only 1 or 2, the logic for pre-processing in Fig.9 is implemented just by multiplexers. Due to the restrictions of *base* and *ystride* value, $s(EA_i)=EA_i\%M$ is equivalent to the module assignment functions described as Eq.(7) and Eq.(12) for A- and S-scheme, respectively. And their address assignment function is $a(EA_i)=EA_i/M$.

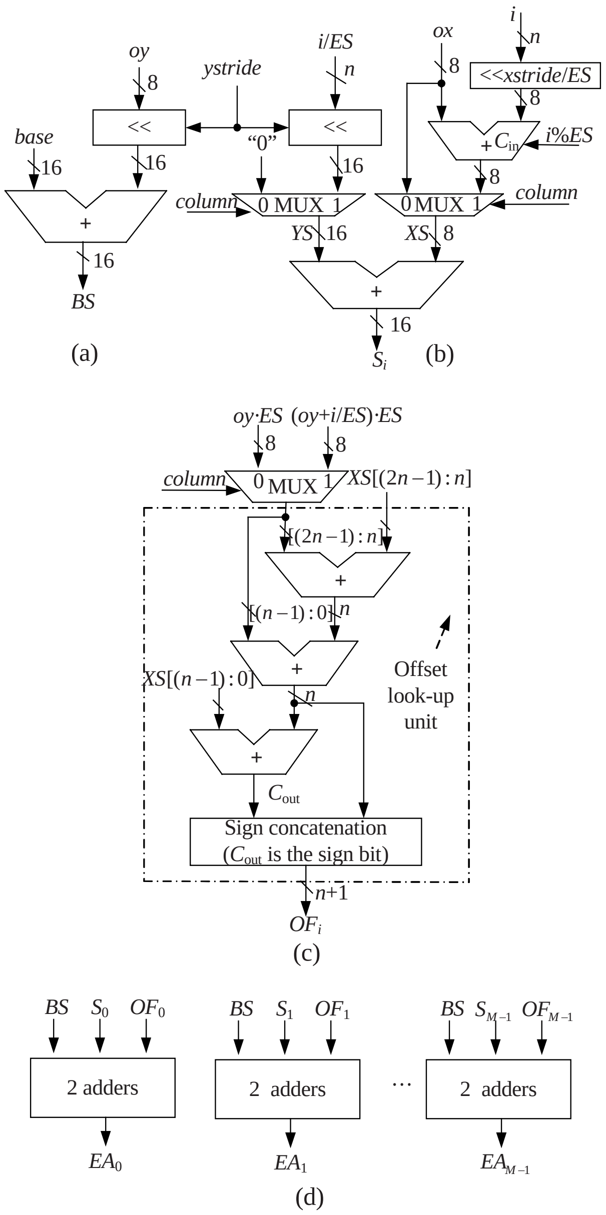


Fig.9 Circuits of EAU. (a) Adder for *BS* computing; (b) Logic for S_i computing; (c) Logic for OF_i computing; (d) Addition of EA_i computing

The logic of PCU is shown in Fig.10. PCU changes the order of the module numbers according to the low n bits of EA_i. The comparators are used to find the memory module number $EA_i\%M$ that equals the constant c, $c\in[0, M-1]$. The corresponding data element of index i is forwarded to the output Idx_c. For example, with four memory module numbers $\{s(EA_0), s(EA_1), s(EA_2), s(EA_3)\}=\{1, 2, 3, 0\}$, the reordered numbers are $\{Idx_0, Idx_1, Idx_2, Idx_3\}=\{3, 0, 1, 2\}$.

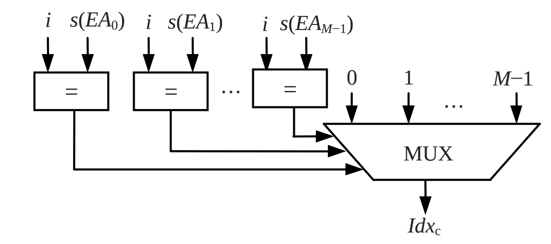


Fig.10 Logic of permutation control unit (PCU) (1/M part)

The parallel memory system employs three full crossbar permutation networks. One of them is presented in Fig.11. Memory module numbers in the original way $\{s(EA_0), s(EA_1), \ldots, s(EA_{M-1})\}$ are used as control signals $\{Ctrl_0, Ctrl_1, \ldots, Ctrl_{M-1}\}$ in the read DPN, while the reordered numbers $\{Idx_0, Idx_1, \ldots, Idx_{M-1}\}$ are utilized as control signals in write DPN and APN.

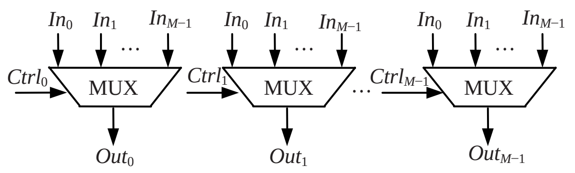


Fig.11 One of the three full crossbar permutation networks used in the parallel memory system

HARDWARE RESULTS AND COMPARISON

The proposed parallel memory architecture has been implemented in a synthesizable register transfer level Verilog-HDL. The area and timing results are obtained by logic synthesis using SYNOPSYS Design Complier. The estimates are performed for architectures with 4, 8 and 16 memory modules. Total memory capacity maintains 8 KB. The memory circuits are generated by Artisan Memory Compiler and the cost is not taken into account in this paper.

Results of hardware implementation

The proposed architecture (excluding memory modules) was synthesized using a 0.18-μm CMOS standard-cell library. Table 1 tabulates the area consumptions in gate count of the proposed parallel memory system. As shown in Table 1, the gate count of EAU increases almost linearly as M increases, while that of PCU and three full crossbar permutation networks increases more significantly. With M=16, the three permutation networks take 55% of the total area. The address widths of each memory module are 11-, 10- and 9-bit for M=4, 8 and 16 respectively, and the data widths are restricted to 8-bit. So the area is different between the APN and two DPNs.

Table 1 Area consumption of the proposed design

Unit		Area consumption (gate)		
		M=4	M=8	M=16
AGU	EAU	1098	2436	5188
	PCU	29	191	1015
	APN	249	1088	3840
Write DPN		203	869	3472
Read DPN		203	869	3472
Total		1782	5453	16987

The delay evaluations of each unit are shown in Table 2. The delay of the proposed parallel memory architecture consists mainly of AGU (especially the EAU) and memory module access. However, the delay of EAU almost maintains constant as M increases. The delay of the PCU and three permutation networks increases remarkably as M increases.

Table 2 Delay estimations of each unit

Unit		Delay estimation (ns)		
		M=4	M=8	M=16
AGU	EAU	2.56	2.65	2.90
	PCU	0.29	0.45	0.60
	APN	0.24	0.38	0.41
Memory module		2.05	2.01	2.00
Write DPN		0.22	0.37	0.41
Read DPN		0.22	0.37	0.41

Comparison

Table 3 lists the implementations and properties of reference parallel memory architectures with the proposed. The area consumptions of these architectures (excluding memory modules) are shown in Table 4. The given transistor counts in (Tanskanen *et al.*, 2004; 2005) are converted into gate counts by supposing four transistors per single 2-input NAND gate. CPMA (Aho *et al.*, 2004) offers very wide configurability and can support a multitude of data patterns demanded in several applications. The results shown in Table 4 contain only the equivalent implementation of (Aho *et al.*, 2007) and our design. Three full crossbars are employed as permutation networks in (Aho *et al.*, 2004; 2007) and the proposed architecture. Their circuits of PCU, APN and two DPNs are similar to those depicted in Section 5.

Table 3 Comparison with existing parallel memory architectures

Source	Technology	Capacity	Skewing scheme	Data-type	Access format
Li *et al.*, 2005	0.35 μm	80 B/module	Linear	Byte	Row, column
Tanskanen *et al.*, 2004	0.35 μm	Total 8 KB	Linear	Byte	Row
Tanskanen *et al.*, 2005	0.35 μm	Total 8 KB	XOR	Byte	Row, column, rectangle
Aho *et al.*, 2004	0.25 μm	2 KB/module	Linear, XOR, diamond	Byte	Nearly arbitrary
Aho *et al.*, 2007	0.18 μm	512 B/module	XOR	Byte	1D arbitrary
This work	0.18 μm	Total 8 KB	Linear shift/rotate	Byte Half-word	Row, column, power-of-two strides in two directions

Table 4 Comparison of gate counts with reference architectures

Source	Area consumption (gate)		
	M=4	*M*=8	*M*=16
Li *et al.*, 2005		1350	
Tanskanen *et al.*, 2004	862	1753	
Tanskanen *et al.*, 2005	883	3286	13535
Aho *et al.*, 2004	11169	29952	79639
Aho *et al.*, 2007	1042	4069	14882
This work	1782	5453	16987

As shown in Table 4, the parallel memory architectures with fewer access formats and fixed memory modules, like (Li *et al.*, 2005) and (Tanskanen *et al.*, 2004), require noticeable less area than configurable architectures. For example, Li *et al.*(2005)'s system consumes only 20% gate counts of the proposed architecture with 8 memory modules. However, the architecture of (Aho *et al.*, 2007) and the proposed architecture demand 18% and 23% respectively of the CPMA gate count with *M* from 4 to 16. Although architectures in (Aho *et al.*, 2007) and (Tanskanen *et al.*, 2005) have a little advantage in area consumption over the presented architecture, they do not support stride access of 16-bit element and 2D stride access patterns. Therefore, the proposed system makes a tradeoff between flexibility and area-efficiency for video coding.

For further comparison with Aho *et al.*(2007)'s architecture, the proposed architecture is also divided into 3 and 2 pipeline stages for read and write respectively, depicted as Fig.12. Table 5 shows the critical path timing of Aho's and the proposed architecture with pipelining. The critical path of the proposed system is from EAU to APN, which has a longer delay than Aho's by a factor of 1.17, when *M*=4. But when *M*=8 and 16, the delay of Aho's exceeds our design, because the delay of EAU in the proposed architecture almost maintains constant as *M* increases. Hence, the proposed architecture is more extensible for a wider bus width. The synthesis results show that the proposed system can achieve 318, 265, and 230 MHz clock frequency with 4, 8, and 16 memory modules, respectively.

Table 5 Critical path time of (Aho *et al.*, 2007) and this work

Source	Critical path time (ns)		
	M=4	*M*=8	*M*=16
Aho *et al.*, 2007	2.68	3.91	5.02
This work	3.14	3.77	4.33

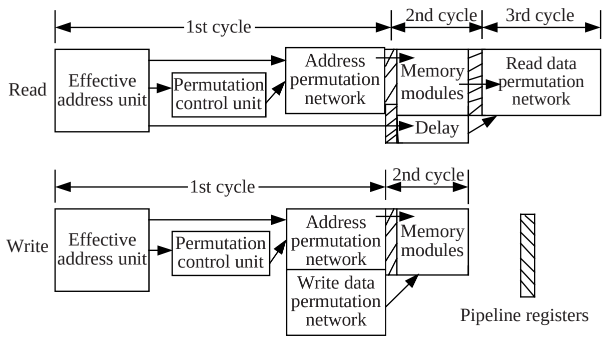


Fig.12 Pipeline architecture for read and write operations

PERFORMANCE COMPARISONS

According to the 2D access pattern in Fig.4, the vector data load and store instructions are defined as

vLD Base, *Ox*, *Oy*, *Mode*, *vR*
//load *vR* from memory
vST vR, *Base*, *Ox*, *Oy*, *Mode*
//store *vR* into memory

where *vR* is an *M*-byte vector register. *Base*, *Ox*, *Oy* are 16-bit registers, which contain the base address, the horizontal offset and the vertical offset, respectively. *Mode* is the 16-bit configuration register, in which parameters, including *xstride*, *ystride*, *ES*, *A/S* and *column*, are specified before memory access.

Three typical kernels of video coding, including 16×16 SAD (full pixel), 8×8 OMBC (overlapped block motion compensation) and 8×8 IDCT, are implemented for performance comparisons. Table 6 shows the memory instruction counts (permutation overheads are included) of different memory architectures. In the 16×16 SAD, current block data are correctly word-aligned, but the matched reference data in the search area are unaligned. The word-addressable memory needs extra memory access and "align" operations for reference data access. In the 8×8 OMBC, accessing every other pixel is required in the interpolated area. One extra memory access and one "mix" operation are needed per row for word- and byte-addressable memories. The word- and byte-addressable memories need matrix transposition between the row and column transforms in the 8×8 IDCT. According to (Lee, 2000), 32, 32 and 24 "mix" instructions are used for the 8×8 transposition, in the cases *M*=4, 8 and 16, respectively. Tanskanen *et al.*(2005) do not support parallel access to coefficients with power-of-two strides for 2D orthogonal transforms. Then it requires extra "mix" instructions to permute subwords for in-place computation of the 8×8 IDCT. As can be seen from Table 6, the proposed architecture obtains 1.95× speedups in the 8×8 IDCT function compared to (Tanskanen *et al.*, 2005).

Furthermore, the proposed architecture has been implemented on VSP (Liu *et al.*, 2006), which has a dual-pipeline 64-bit SIMD architecture with 8 byte-addressable memory modules. We develop a cycle-accurate simulator for performance evaluation. Fig.13 shows the cycle counts of H.264 decoding (excluding entropy decoding) at 30 fps (QCIF, *QP*=28), for both VSP with and without the proposed architecture. It can be seen that VSP enhanced with the proposed architecture achieves 1.28× speedups for H.264 real-time decoding, compared to VSP in (Liu *et al.*, 2006).

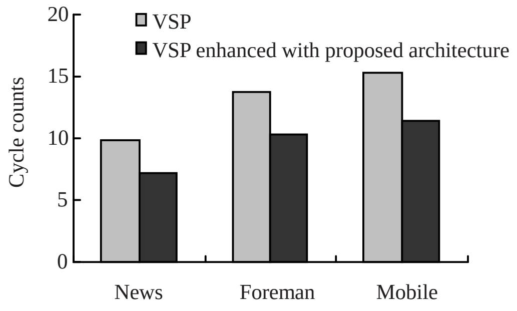


Fig.13 Performance comparison on VSP with and without proposed architecture for H.264 decoding

Table 6 Instruction counts of memory access with *M*=4, 8, and 16

Memory architecture	16×16 SAD			8×8 OMBC			8×8 IDCT		
	4	8	16	4	8	16	4	8	16
Word-addressable	208	112	64	256	168	84	292	162	90
Byte-addressable	128	64	32	256	168	84	292	162	90
Tanskanen *et al.*, 2005	128	64	32	176	88	44	260	130	66
Proposed	128	64	32	176	88	44	132	66	34

CONCLUSION

SIMD architectures provide a cost-effective solution to exploit inherent data-level parallelism of video coding. But the performance is quite limited by the overheads of providing operands in a format amenable to SIMD processing. Therefore, an optimal parallel memory architecture is required for accessing subwords simultaneously, so as to eliminate the excessive memory operations and explicit data rearrangements. In this paper, we propose a parallel memory architecture for video coding with power-of-two memory modules. It employs two novel skewing schemes to provide conflict-free parallel access to adjacent elements or with power-of-two intervals in both horizontal and vertical directions. They support stride data access of both 8-bit and 16-bit data types in video processing. The proposed system with 4, 8 and 16 memory modules have been implemented under a 0.18-μm COMS technology, and the costs vary from 2.4k to 19k gates excluding memory cells. Furthermore, the results show that the proposed parallel memory architecture can efficiently reduce the memory instructions and improve the performance of video coding.

References

Aho, E., Vanne, J., Kuusilinna, K., Hamalainen, T.D., 2004. Address computation in configurable parallel memory architecture. *IEICE Trans. on Inf. Syst.*, **87**(7):1674-1681.

Aho, E., Vanne, J., HÄmÄlÄinen, T.D., 2007. Configurable data memory for multimedia processing. *J. VLSI Signal Processing*, **50**(2):231-249. [doi:10.1007/s11265-007-0126-x]

Budnik, P., Kuck, D.J., 1971. The organization and use of parallel memories. *IEEE Trans. on Comput.*, **C-20**(12): 1566-1569. [doi:10.1109/T-C.1971.223171]

Cheresiz, D., Juurlink, B., Vassiliadis, S., Wijshoff, H.A.G., 2005. The CSI multimedia architecture. *IEEE Trans. on VLSI Syst.*, **13**(1):1-13. [doi:10.1109/TVLSI.2004.840415]

Corbal, J., Valero, M., Espasa, R., 1999. Exploiting a New Level of DLP in Multimedia Applications. Proc. Int. Symp. on Microarchitecture, p.72-79. [doi:10.1109/MICRO.1999.809445]

Deb, A., 1996. Multiskewing—a novel technique for optimal parallel memory access. *IEEE Trans. on Parall. Distrib. Syst.*, **7**(6):595-604. [doi:10.1109/71.506698]

Frailong, J.M., Jalby, W., Lenfant, J., 1985. XOR-Schemes: A Flexible Data Organization in Parallel Memories. Proc. Int. Conf. on Parallel Processing, p.276-283.

Gossel, M., Rebel, B., Creutzburg, R., 1994. Memory Architecture & Parallel Access. Elsevier Science Inc., New York, USA, p.250.

Khailany, B., Dally, W.J., Kapasi, U.J., Mattson, P., Namkoong, J., Owens, J.D., Towles, B., Chang, A., Rixner, S., 2001. Imagine: media processing with streams. *IEEE Micro.*, **21**(2):35-46. [doi:10.1109/40.918001]

Kozyrakis, C., Patterson, D., 2002. Vector vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks. Proc. Int. Symp. on Microarchitecture, p.283-293. [doi:10.1109/MICRO.2002.1176257]

Kozyrakis, C.E., Patterson, D.A., 2003. Scalable vector processors for embedded systems. *IEEE Micro.*, **23**(6):36-45. [doi:10.1109/MM.2003.1261385]

Lee, R.B., 2000. Subword Permutation Instructions for Two-dimensional Multimedia Processing in MicroSIMD Architectures. Proc. IEEE Int. Conf. on Application—Specific Systems, Architectures and Processors, p.3-14. [doi:10.1109/ASAP.2000.862373]

Li, L., Goto, S., Ikenaga, T., 2005. An Efficient Deblocking Filter Architecture with 2-Dimensional Parallel Memory for H.264/AVC. Proc. Asia and South Pacific Design Automation Conf., p.623-626. [doi:10.1145/1120725.1120978]

Liu, K.J., Qin, X., Yan, X.L., Quan, Li, 2006. A SIMD Video Signal Processor with Efficient Data Organization. Proc. IEEE Asia Solid-State Circuits Conf., p.115-118. [doi:10.1109/ASSCC.2006.357865]

Park, J.K., 2004. Multiaccess memory system for attached SIMD computer. *IEEE Trans. on Comput.*, **53**(4):439-452. [doi:10.1109/TC.2004.1268401]

Sohi, G.S., 1993. High-bandwidth interleaved memories for vector processors—a simulation study. *IEEE Trans. on Comput.*, **42**(1):34-44. [doi:10.1109/12.192212]

Talla, D., John, L.K., Burger, D., 2003. Bottlenecks in multimedia processing with SIMD style extensions and architectural enhancements. *IEEE Trans. on Comput.*, **52**(8):1015-1031. [doi:10.1109/TC.2003.1223637]

Tanskanen, J., Sihvo, T., Niittylahti, J., Takala, J., Creutzburg, R., 2000. Parallel Memory Access Schemes for H.263 Encoder. Proc. IEEE Int. Symp. on Circuits and Systems, p.691-694. [doi:10.1109/ISCAS.2000.857189]

Tanskanen, J.K., Sihvo, T., Niittylahti, J.T., 2004. Byte and modulo addressable parallel memory architecture for video coding. *IEEE Trans. on Circuits Syst. Video Technol.*, **14**(11):1270-1276. [doi:10.1109/TCSVT.2004.835148]

Tanskanen, J.K., Creutzburg, R., Niittylahti, J.T., 2005. On design of parallel memory access schemes for video coding. *J. VLSI Signal Processing*, **40**(2):215-237. [doi:10.1007/s11265-005-4962-2]

Trenas, M.A., Opez, J., Arguello, F., Zapata, E.L., 1998. A Memory System Supporting the Efficient SIMD Computation of the Two Dimensional DWT. Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, p.1521-1524.