

# Parallel Memory Architecture for Arbitrary Stride Accesses

Eero Aho, Jarno Vanne, and Timo D. Hämäläinen  
Institute of Digital and Computer Systems  
Tampere University of Technology  
Tampere, Finland  
{eero.aho, jarno.vanne, timo.d.hamalainen}@tut.fi

**Abstract**—Parallel memory modules can be used to increase memory bandwidth and feed a processor with only necessary data. Arbitrary stride access capability with interleaved memories is described in previous research where the skewing scheme is changed at run time according to the currently used stride. This paper presents the improved schemes which are adapted to parallel memories. The proposed novel parallel memory architecture allows conflict free accesses with all the constant strides which has not been possible in prior application specific parallel memories. Moreover, the possible access locations are unrestricted and the data patterns have equal amount of accessed data elements as the number of memory modules. The complexity is evaluated with resource counts.

## I. INTRODUCTION

The processor versus memory performance gap has been widening for a number of years. Cache memories can increase system memory bandwidth in general purpose systems, but they are not typically used in digital signal processing applications due to latency and real-time restrictions. Wide memories allow accessing large blocks of data. Unfortunately, they may also produce data that the currently running algorithm does not use. Multiport memories enable several simultaneous memory accesses but they are expensive solutions especially when the number of ports is large.

Parallel memory modules can be used to access special data patterns and feed the processors with only algorithm specific data [1], [2]. In specific data patterns, accessed data elements are separated by a distance called a *stride*. Many applications in the fields of digital signal processing and telecommunications benefit from the use of strides. Vector/matrix computation, fast Fourier transform (FFT), and Viterbi algorithm are some examples [3], [4].

In this study, *interleaved memories* use time-multiplexed parallel memory modules that receive access requests serially one by one. Respectively, *parallel memories* are defined as space-multiplexed memories that are used e.g. in SIMD processing. Parallel memories have wide address and data buses and the memory modules receive access requests in parallel.

Traditionally, the research on stride accesses has concentrated on *module assignment functions* (*skewing schemes*) that try to ensure conflict free parallel data accesses to a set or maximal amount of access strides [3], [5]. Normally, the scheme can be changed only at design time. This paper considers merely constant stride accesses and data patterns having equal amount of accessed data elements as the number of memory modules. Moreover, access locations of the data patterns are supposed to be unrestricted. Unfortunately, no single skewing scheme has been found that allows unrestrictedly placed conflict free accesses for all the constant strides when the number of memory modules is power of two [6]. The generalized proof is given later in this paper.

One solution is to use the *Configurable Parallel Memory Architecture* (CPMA) that allows several data *access formats* (*templates*) and skewing schemes to be used with a single hardware when the number of memory modules is arbitrary [4]. However, a more dedicated system for changeable constant stride access utilizes *dynamic storage schemes* that allow multiple stride specific schemes to be used within a single system [6], [7]. Nevertheless, the memory module count is restricted to a power of two and only interleaved memory system is considered in [6] and [7].

Arbitrary stride accessibility with parallel memories is demanded in real-time image scaling architecture shown in [8]. However, the main aspect in [8] is a new image scaling parallelization method and the implementation utilizing it. The parallel memory theory and implementation details are not discussed.

This paper presents a novel parallel memory architecture allowing conflict free arbitrary constant stride accesses. For hardware simplicity, the number of memory modules is restricted to a power of two ( $N = 2^n$ ). From the hardware point of view, the skewing schemes are simplified from the ones in [7]. A skewing scheme is changed at run time according to the used stride. However, when a skewing scheme is changed, the data locating already in the memory modules is naturally invalidated. This is not a problem, for example, in the image scaling system in [8]. A new stride with a possible new scheme is demanded only when new input image data is written to parallel memories.

Background for the used conflict free skewing schemes is given in Section II. The proposed parallel memory architecture is described in detail in Section III. Complexity is evaluated in Section IV and Section V concludes the paper.

## II. BACKGROUND AND THEORY

The address space in parallel memories cannot be assigned arbitrarily but the data has to be referenced using predetermined patterns, called access formats or templates. The distribution of data to the memory modules is called a module assignment function  $S$  that is also known as a skewing scheme. The used module assignment function determines access formats that can be used conflict free. A data element at location  $i$  is assigned to a memory module according to  $S(i)$ . An address function  $a(i)$  (also called row number) determines the physical address in a memory module for a data element.

*Theorem 1:* No single skewing scheme can be found that allows conflict free accesses for all the constant strides when the access format can be unrestrictedly placed. This theorem is valid with arbitrary number of memory modules and when at least two data elements are accessed in parallel.

*Proof:* Let  $S(i)$  be the used module assignment function and  $i_1$  the first accessed data location. Therefore, the second accessed data location  $i_2 = i_1 + \text{stride}$ . The two elements  $i_1$  and  $i_2$  cannot be accessed conflict free with a stride that is selected such that  $S(i_1) = S(i_2)$ .

*Theorem 2:* Let  $\sigma$  be relatively prime to 2 (i.e.  $\sigma$  is an odd positive integer),  $s$  a nonnegative integer, and the number of memory modules  $N = 2^n$ . Any skewing scheme that allows a conflict free access for a stride  $2^s$  also provides a conflict free access for any stride defined as

$$\text{stride} = \sigma \cdot 2^s. \quad (1)$$

Theorem 2 is proved in [6]. All the strides can be formed by changing  $\sigma$  and  $s$ . According to Theorem 2, a stride with a particular  $s$  and any  $\sigma$  values can be used conflict free in a single skewing scheme. Therefore, by finding conflict free storage schemes for each of the values of  $s$  (i.e. single scheme for a single  $s$ ) enables conflict free access for arbitrary strides.

The conventional low-order interleaved scheme utilizes the module assignment function

$$S(i) = i \bmod N, \quad (2)$$

and the address function

$$a(i) = \lfloor i / N \rfloor. \quad (3)$$

The  $n$  least significant bits (LSBs) of the data location  $i$  are used as module number and the rest of the bits as an address

in a module. All the odd strides can be accessed with the low-order interleaved scheme [5]. The same can be proved with Theorem 2. It is trivial to see that stride  $2^0 = 1$  is always conflict free in the low-order interleaved scheme. Therefore, according to Theorem 2, also all the other odd strides are conflict free.

Dynamic schemes for arbitrary stride accesses were first introduced by Harper *et al.* in [6]. Row rotation schemes [9] with  $N = 2^n$  was used. XOR-schemes allow simpler computation since merely logic gates are demanded and no carry bits are used for computation [1], [10]. XOR-schemes always require power of two memory module count. Another Harper's approach concerning arbitrary stride accesses utilizes XOR-schemes [7].

In this study, XOR-schemes and  $N = 2^n$  memory modules are used. The definitions here differ a slightly from that in [7]. In a parallel memory architecture with  $2^l$  data locations,  $i$  and  $z$  are  $l$ -bit vectors and  $T$  is an  $l \times l$  transformation matrix containing binary numbers. Vector  $z$  is partitioned into two fields: the module number  $S(i)$  is defined with  $n$  least significant bits and the address in the memory module  $a(i)$  with the remaining bits. The vector  $z$  is defined as follows:

$$z = i \cdot T. \quad (4)$$

The elements of the matrix  $T$  are marked in a custom fashion. The top left element of the matrix is  $t_{l-1,l-1}$  and the bottom right is  $t_{0,0}$ . The vector addition and multiplication in (4) are achieved by bit-wise logical operations XOR and AND, respectively. The matrix  $T$  is identity matrix having also some off-diagonal 1-elements. Let  $T_{y,x}$  represent an *elementary transformation* with an additional 1-element in a matrix position  $(y, x)$ . Several off-diagonal 1-elements can be represented by a product of elementary transformations. For example,

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = T_{1,0} \cdot T_{2,1}. \quad (5)$$

In [7],  $T$  is defined as

$$T = \prod_{k=0}^{\min(n,s)-1} T_{\max(n,s)+k,k}, \quad (6)$$

which ensures conflict free access for any stride. The proposed transformation simplifies (6) by defining

$$T = \prod_{k=0}^{n-1} T_{s+k,k}. \quad (7)$$

The proposed transformation (7) equals to (6) when  $s \geq n$ .

TABLE I  
EXAMPLE OF THE PROPOSED SKEWING SCHEME ( $n = 2, s = 1$ )

$i$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(i)$	0	1	3	2	2	3	1	0	0	1	3	2	2	3	1	0
$a(i)$	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3

An example of the proposed skewing scheme is tabulated in Table I where  $n = 2$  and  $s = 1$ . The memory module count  $N = 2^n = 4$  and the address depth in a memory module is restricted to four in Table I. Sixteen data elements are displayed. For example, a data element at location  $i = 5$  is stored in memory module  $S(i) = 3$  under address  $a(i) = 1$ . The proposed transformation matrix  $T$  (7) in Table I is defined as (5). With  $s = 1$  in (1), strides of 2, 6, 10, 14, ... are conflict free in arbitrary location. For example, an access format with stride = 2 at location  $i = 1$  (painted gray) is conflict free since all the four accessed data elements are located in different memory modules.

#### A. Hardware Implementation

All the off-diagonal 1-elements of the matrix  $T$  (6) locate in LSB side. Therefore, the address in a memory module  $a(i)$  is received straight from the  $l-n$  MSBs of the data element location  $i$  and no additional hardware is demanded [7]. This address function is defined in (3). The same situation holds for the proposed transformation (7).

The module number  $S(i)$  calculation is pretty simple because just at most two elements of any column of  $T$  (6) are nonzero [7]. Therefore, the module number can be calculated as follows:

$$S(i) = i[n + s - 1 : \max(n, s)] \oplus i[n - 1 : 0], \quad (8)$$

where a data element location  $i$  is stated in binary form [MSB : LSB] and  $\oplus$  denotes bit-wise logical XOR operation.

The proposed transformation matrix  $T$  (7) also has at most two nonzero elements in a single column. This leads the following simple implementation:

$$S(i) = i[n + s - 1 : s] \oplus i[n - 1 : 0]. \quad (9)$$

The implementation of (8) and (9) demands defining  $n$  and  $s$ . In a particular implementation, the number of memory modules  $N = 2^n$  is a constant. Respectively,  $s$  is determined by a stride (1). Therefore,  $n$  is defined at design time and  $s$  at run time.

With  $n = 2$  and  $s = 1$  (Table I), the proposed module assignment function  $S(i)$  (9) and address function  $a(i)$  are defined as

$$S(i) = i[2:1] \oplus i[1:0], \\ a(i) = i[3:2].$$

The hardware implementations of [6], [7], and [11] as well as (9) do not function properly when *odd* strides are used ( $s = 0$ ). As mentioned above, all the odd strides can be accessed with a conventional low-order interleaved scheme (2). In the proposed implementation, the odd strides are separately taken into account. When  $s = 0$ , low-order interleaved scheme is formed as follows:

$$S(i) = 0 \oplus i[n - 1 : 0] = i[n - 1 : 0]. \quad (10)$$

#### B. Discussion

As stated above, the proposed transformation  $T$  (7) equals to (6) when  $s \geq n$ . In the other even stride lengths, the proposed skewing schemes (9) have been verified with Matlab software simulations. The memory organizations under simulation had power of two number of memory modules  $N = 2, 4, 8, \dots, 1024 \mid n \in \{1, \dots, 10\}$ . All the possible lengths of strides were verified to be conflict free when  $s \in [1, n-1]$ . Respectively, all the odd strides ( $s = 0$ ) are managed with (10). Therefore, it is stated that the proposed skewing schemes (9) and (10) provide conflict free access for all the constant strides with practical number of memory modules.

A row access format ( $N$  adjacent data elements with stride 1) is commonly used. Word addressability means that the memory can be addressed only at word boundaries when the word is defined as  $N$  data elements. The proposed module assignment function  $S(i)$  (9) allows conflict free word addressable row accesses with all the values of  $n$  and  $s$ . This property is also available with (8) unless not considered in [7]. For example with  $N = 4$  (Table I), data locations  $i = 8, 9, 10, 11$  or  $i = 12, 13, 14, 15$  can be accessed conflict free. However, locations  $i = 9, 10, 11, 12$  are not conflict free because data elements at  $i = 11$  and  $i = 12$  are located in the same memory module 2. Therefore, when a row access format is demanded to reference arbitrarily in the memory, not only according to word boundaries, the skewing scheme needs to be modified accordingly.

The differences between the proposed implementation and the Harper's implementations [6], [7], and [11] are discussed in the following. The proposed implementation use parallel memories instead of interleaved memory. In parallel memories, each of the  $N$  addresses  $a(i)$  of the data locations  $i$  is assigned to the correct memory module  $S(i)$  simultaneously. Therefore, a lot of parallel logic is demanded. Moreover, address and data permutation networks are required around the memory modules.

The proposed schemes are optimized from Harper's ones [7], [11] in terms of hardware complexity. The hardware in (8) includes  $n$  two-input XOR gates, a comparator (max determination), two multiplexers, a mask generator and a

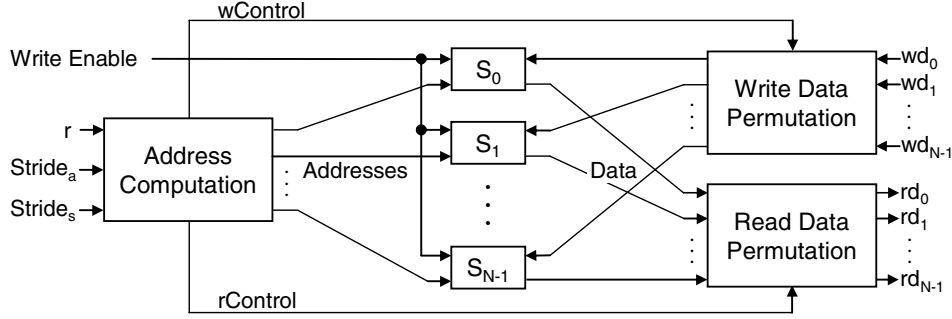


Fig. 1. Proposed parallel memory architecture.

shifter with a capability of bit masking [7], [11]. Respectively, the proposed implementation (9) demands only  $n$  two-input XOR gates and a shifter. Moreover, the proposed hardware implementation determining  $s$  according to a stride differs from Harper's one.

### III. PROPOSED PARALLEL MEMORY ARCHITECTURE

A block diagram of the proposed parallel memory architecture is shown in Fig. 1. The referenced data locations are defined with the *first element location (scanning point)*  $r$  and the stride length  $Stride_a$ . The skewing scheme is formed according to the  $Stride_s$  input. In write operation, *Write Enable* input is asserted simultaneously with the data to be written  $wd_0, wd_1, \dots, wd_{N-1}$ . The two *Data Permutation* units permute the written or read data according to the control signals from the *Address Computation* unit.

As mentioned above, a word addressable row access format (stride = 1) is conflict free in all the used skewing schemes but the scheme can only be changed before writing a new data to memory. Therefore, the access stride ( $Stride_a$ ) and the stride defining the used skewing scheme ( $Stride_s$ ) are

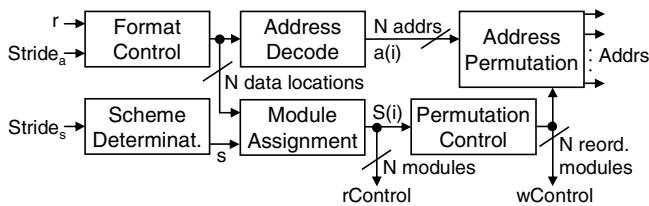


Fig. 2. Address Computation unit.

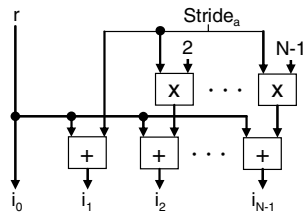


Fig. 3. Format Control unit.

separated to allow controlling the access stride and skewing scheme separately. For example, the data storage in Table I can be referenced with a row access format without changing the skewing scheme by assigning  $Stride_a = 1$  and  $Stride_s = 4$ . However, the row access format can be addressed only at word boundaries.

The proposed parallel memory system is implemented in VHDL. The system is designed to be configurable both at design time and at run time. In the hardware implementation, most parameters are VHDL generics, which means that the memory module count ( $N = 2^n$ ), size of a memory module, and width of buses are adjustable. The run-time configurability refers to the ability to change the skewing scheme at run time.

Fig. 2 depicts a block diagram of the Address Computation unit. The outputs of the unit are the memory module addresses for each of the memory module (*Adrs*). Moreover, control signals ( $rControl, wControl$ ) are formed for the Data Permutation units.

Inside the unit, the *Format Control* unit specifies the accessed data locations. The *Scheme Determination* unit defines the used skewing scheme. The *Module Assignment* unit computes the specific memory modules that contain the accessed data and the *Address Decode* unit, in turn, computes the physical addresses for each memory module. The *Permutation Control* unit reorders the memory module numbers. According to these values, the *Address Permutation* unit directs the physical addresses to the proper memory modules.

A detailed block diagram of the Format Control unit is shown in Fig. 3. The stride is multiplied with constant numbers  $2, \dots, N-1$  and the referenced data locations  $i_0, i_1, \dots, i_{N-1}$  are formed by additions.

Fig. 4 depicts the Scheme Determination unit. The input to the unit is the stride defining the used skewing scheme  $Stride_s = Str[d-1 : 0]$ . The relation between a stride and  $s$  is given in (1). The output  $s$  is constructed by determining the count of adjacent zeros before first '1' by starting from LSB. For example, with  $Stride_s = 12_{10} = 01100_2$ , the output  $s = 2_{10}$ . Another input of the equality comparators is a constant zero but the width of the other input is different in each of

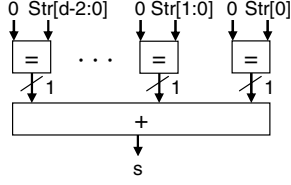


Fig. 4. Scheme Determination unit.

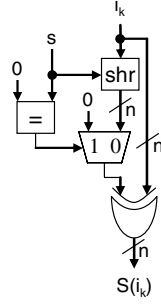


Fig. 5. Module Assignment unit (1/N part of the unit).

the comparators. The output of the equality comparators is one bit wide. Those one bit wide results are summed up to get the output  $s$ . Since the  $\text{Stride}_s \geq 1$  (i.e. stride cannot be 0), the most significant bit  $\text{Str}[d-1]$  is not actually demanded.

Fig. 5 shows a block diagram of the Module Assignment unit. In fact, there are  $N$  parallel similar logic blocks in the Module Assignment unit. The input  $s$  from the Scheme Determination unit defines the used scheme. When  $s = 0$ , low-order interleaved scheme is used as shown in (10). Zero value is selected with a multiplexer and the  $n$  LSBs of the data location  $i_k$  is used to form  $S(i_k)$  when  $k \in [0, N-1]$ . Otherwise, the data location  $i_k$  is shifted right (shr) according to  $s$  and, after that,  $n$  parallel bit-wise XOR operations are performed as shown in (9).

The Address Decode unit is presented in Fig. 6. The unit implements the address function (3) and contains no logic, only hardwired shifting. The MSBs of a data location  $i_k$  are selected as address  $a(i_k)$  output.

The Permutation Control unit is shown in Fig. 7. The unit changes the order of the module numbers received from the Module Assignment unit. The comparators are used to find the memory module number  $S(i_b)$  ( $b \in [0, N-1]$ ) that equals the constant  $k$ ,  $k \in [0, N-1]$ . The respective index  $b$  of the data element  $i_b$  is forwarded to output  $S'(i_k)$ . As an example with  $N = 4$  and with the memory module numbers  $\{ S(i_0), S(i_1), S(i_2), S(i_3) \} = \{ 1, 2, 3, 0 \}$ , the reordered module number are  $\{ S'(i_0), S'(i_1), S'(i_2), S'(i_3) \} = \{ 3, 0, 1, 2 \}$ .

The parallel memory system includes three Permutation units. One of the units is depicted in Fig. 8. Full crossbar permutation networks are used. Memory module numbers in original or reordered way are used as control signals ( $\text{Ctrl}_0, \text{Ctrl}_1, \dots, \text{Ctrl}_{N-1}$ ). In the Write and Read Data Permutation



Fig. 6. Address Decode unit (1/N part of the unit).

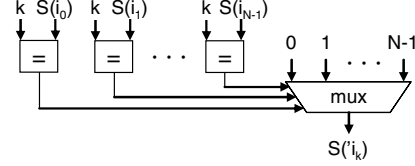


Fig. 7. Permutation Control unit (1/N part of the unit).

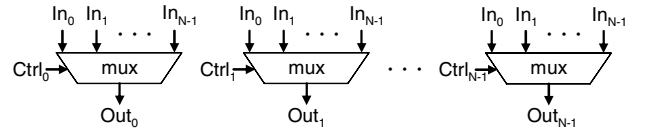


Fig. 8. Permutation unit.

units, the inputs ( $\text{In}_0, \text{In}_1, \dots, \text{In}_{N-1}$ ) and outputs ( $\text{Out}_0, \text{Out}_1, \dots, \text{Out}_{N-1}$ ) are data buses. Respectively, addresses are permuted in the Address Permutation unit.

#### IV. COMPLEXITY

All of the resources required in the sub-blocks of the proposed parallel memory system are tabulated in Table II. The resources are separated into Format Control (FC), Scheme Determination (SD), Address Decode (AD), Module Assignment (MA), Permutation Control (PC), and three Permutation units (3xP). The number of memory modules is denoted by  $N = 2^n$  ( $N \geq 2$ ) and the width of the  $\text{Stride}_s$  input by  $d$  (Fig. 4).

Except for multiplexers, just two inputs are supposed in all the resources. Multiplexers use an additional control signal to select one of the two inputs. The bit widths of the inputs depend on the design-time configurations, for example, the used memory module width and depth. In the Scheme Determination unit (Fig. 4), the utilized adder is multi-input but the multiple inputs are just one bit wide. Therefore, it is marked as a single adder.

One of the inputs in some of the two-input resources can be a constant and those units are separately marked 'const'. With multiplexers, one or two of the three inputs can be constant.

The Format Control unit utilizes  $N-1$  adders and  $N-2$  multipliers with the other input defined as a constant. Respectively, the Module Assignment unit demands  $N$  shifters and  $N \times n$  2-bit XOR gates. In addition,  $N$  multiplexers and  $N$  equality comparators with some constant inputs are required.

TABLE II  
UTILIZED RESOURCES IN THE PARALLEL MEMORY SYSTEM

	+/-	shift	mux	x	const mux	=	gates XOR
FC	$N-1$			$N-2$			
SD	1					$d-1$	
AD							
MA		$N$			$N$	$N$	$N \times n$
PC					$N(N-1)$	$N \times N$	
3xP			$3N(N-1)$				
All	$N$	$N$	$3N(N-1)$	$N-2$	$N \times N$	$N(N+1)+d-1$	$N \times n$

Adder, shifter, and multiplier amounts are linearly proportional to  $N$ . The number of multiplexers and equality comparators are proportional to  $N^2$  in the Permutation Control unit. However, while the other comparator input is a constant, the comparators can be implemented in hardware using  $N^2$  AND ports with some inverted input bits. Moreover, two of the three inputs are constants in the multiplexers of the Permutation Control unit (Fig. 7). When increasing the number of memory modules, the most significant area increase is with the Permutation units, where the multiplexer amount is proportional to  $N^2$ . As a case study, full crossbar permutation networks dominate the logic area in a video coding specific parallel memory system with memory module count four or more [12].

The two Data Permutation units could be combined in the proposed system. The inputs of the combined Data Permutation unit would be selected with "Write Enable" signal between the current inputs of the Read or Write Data Permutation units (Fig. 1). The output of the combined unit would be directed to the inputs of the memory modules and the output of the whole parallel memory system. The number of additional multiplexers would be  $2N$  instead of another Data Permutation amount  $N \times (N-1)$ . This would decrease the resource count when  $N > 2$ . As a drawback, the delay of the combined Data Permutation unit would increase due to additional multiplexers. Moreover, write after read turn-around time of the proposed parallel memory system would also increase. Another or additional possibility to decrease the demanded logic is to utilize multistage interconnection networks like Benes [13].

## V. CONCLUSION

Stride accesses are frequently required in the fields of digital signal processing and telecommunications. In this paper, we have presented a novel parallel memory architecture allowing all the constant stride accesses which has not been possible in prior application specific parallel memory systems. The proposed architecture is mainly useful in SIMD type of processing.

## REFERENCES

- [1] S. Chen, A. Postula, and L. Jozwiak, "Synthesis of XOR storage schemes with different cost for minimization of memory contention," in *Proc. Euromicro Conf.*, Milan, Italy, pp. 170–177, Sep. 1999.
- [2] R. Hartenstein, J. Becker, M. Herz, and U. Nageldinger, "An embedded accelerator for real world computing," in *Proc. IFIP Int'l Conf. Very Large Scale Integration*, Gramado, Brazil, Aug. 1997.
- [3] J. Takala and T. Järvinen, "Stride permutation access in interleaved memory systems," in *Domain-Specific Multiprocessors – Systems, Architectures, Modeling, and Simulation*, ed. S. S. Bhattacharyya, E. F. Deprettere, and J. Teich, pp. 63–84, Marcel Dekker, New York, NY, USA, 2004.
- [4] E. Aho, J. Vanne, K. Kuusilinna, and T. D. Hämäläinen, "Address computation in configurable parallel memory architecture," *IEICE Trans. Inf. & Syst.*, vol. E87-D, no. 7, pp. 1674–1681, July 2004.
- [5] M. Valero, T. Lang, M. Peiron, and E. Ayguadé, "Conflict-free access for streams in multimodule memories," *IEEE Trans. Computers*, vol. 44, no. 5, pp. 634–646, May 1995.
- [6] D. T. Harper III and D. A. Linebarger, "Conflict-free vector access using a dynamic storage scheme," *IEEE Trans. Comput.*, vol. 40, no. 3, pp. 276–283, Mar. 1991.
- [7] D. T. Harper III, "Increased memory performance during vector accesses through the use of linear address transformations," *IEEE Trans. Comput.*, vol. 41, no. 2, pp. 227–230, Feb. 1992.
- [8] E. Aho, J. Vanne, T. D. Hämäläinen, and K. Kuusilinna, "Block-level parallel processing for scaling evenly divisible images," *IEEE Trans. Circuits Syst. I*, vol. 52, no. 12, pp. 2717–2725, Dec. 2005.
- [9] P. Budnik and D. J. Kuck, "The organization and use of parallel memories," *IEEE Trans. Comp.*, vol. C-20, no. 12, pp. 1566–1569, Dec. 1971.
- [10] J. M. Frailong, W. Jalby, and J. Lenfant, "XOR-schemes: A flexible data organization in parallel memories," in *Proc. Int'l Conf. Parallel Processing*, Washington, DC, USA, pp. 276–283, Aug. 1985.
- [11] D. T. Harper III and D. A. Linebarger, "Dynamic address mapping for conflict-free vector access," U.S. Patent 4 918 600, Apr. 17, 1990.
- [12] J. K. Tanskanen and J. T. Niittylahti, "Scalable parallel memory architectures for video coding," *Journal of VLSI Signal Processing*, vol. 38, no. 2, pp. 173–199, Sep. 2004.
- [13] S. Dutta, W. Wolf, and A. Wolfe, "A methodology to evaluate memory architecture design tradeoffs for video signal processors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, no. 1, pp. 36–53, Feb. 1998.