Two-Dimensional Memory Implementation with Multiple Data Patterns

Arseni Vitkovski

ARCES,
University of Bologna,
Viale Pepoli 3/2, 40123 Bologna,
Italy
avitkovski@arces.unibo.it

Georgi Kuzmanov, Georgi Gaydadjiev

Computer Engineering, EEMCS,
Delft University of Technology,
Mekelweg 4, 2600 GA Delft,
the Netherlands
{G.K.Kuzmanov, G.N.Gaydadjiev}@tudelft.nl

Abstract—**In this paper, we propose an implementation of a two-dimensional interleaved memory organization with dynamically reconfigurable access patterns. Our solution targets computing devices with high demands on memory bandwidth, such as multimedia-specific DSPs, scientific vector processors, and SIMD machines. The approach extends prior art by introducing advanced two-dimensional strided accesses augmented with additional parameters for arbitrary rectangular data patterns support. Our scheme provides run-time dynamical reconfiguration and guarantees minimum memory latency and efficient bandwidth utilization for arbitrary chosen pattern parameter combinations. We have implemented in VHDL an architecture independent memory organization and synthesized a set of configurations with 2x2 up to 8x8 memory modules for 90 nm CMOS technology. Synthesis results support our claim on high performance of the memory organization together with high system scalability. More specifically, our estimations suggest that a throughput of up to 1182 Gbit/s can be obtained at the cost of only 212 Kgates.**

Index Terms—**Conflict-free access, high bandwidth, parallel memories.**

I. Introduction

With modern increase of technology development, the performance of memory subsystems lags more and more behind the processing units. This trend becomes increasingly evident for architectures with massively parallel data processing, such as multimedia accelerators, vector processors, SIMD-based machines, etc. There are several techniques developed to reduce the processor versus memory performance gap, including various caching mechanisms, memories advanced with extra wide data word or multiple ports. But most of all, the parallelism phenomenon is utilized in *parallel memory* organizations, where the storage subsystem consists of a set of memory modules working in parallel. The main advantages of this organization are: relatively small overheads, low latency, efficient interconnection usage and possibility of accessing specific *data patterns*. The data patterns depend very much on the target application and might have various shapes, sizes and *strides* (distances between the successive elements).

The design challenge is to ensure conflict-free parallel data access to all (or maximum possible number of) memory modules for a set of different data patterns. This is obtained by means of a *module assignment function*. According to the data pattern format (in other words *template*), various module assignments can be implemented, such as linear functions [1], XOR-schemes [2], rectangular addressable memories [3], periodic schemes [4] and others. *Row address function* specifies physical address inside a memory module. Together, module assignment and row address functions form the class of *skewing schemes*. However, there is no single skewing scheme which would support conflict-free access for all possible data patterns [5]. Two solutions that deal with such limitation are: *Configurable Parallel Memory Architecture* (*CPMA*) [6], [8] and *Dynamic Storage Scheme* (*DSS*) [5], [7], [9], [12]. CPMA provides access to a number of data templates using a single relatively complex hardware when the number of memory modules is arbitrary. A more dedicated DSS unifies multiple storage schemes within one system. The appropriate scheme is chosen dynamically according to the specific data pattern in use. DSS restricts the amount of memory modules to the power of two and considers only interleaved memory system [7], [9].

More precisely, DSS for a strided vector access was presented in [7]. The stride value is detected by the compiler and sent directly to the pipelined address transformation hardware. In such a way, the scheme supports conflict-free accesses for non-restricted vectors with constant arbitrary stride.

In [9] the authors extended their scheme with block, multistride and FFT accesses. To decrease the latency of multistride vector access when conflict-free access is not achieved, it was proposed to use dedicated buffers to smooth out transient non-uniformities in module reference distribution. Block access supported only restricted set of blocks sizes equal to power of two. Finally, in order to improve a radix-2 FFT algorithm, authors proposed non-interleaved storage scheme and a constant geometry algorithm for which they identified three data patterns. For all three types of address transformations, the same hardware was used.

CPMA from [6] supports generate, crumbled rectangle, chessboard, vector, and free data patterns. Virtual address is used to read appropriate row address and access function from the page table which are further transformed into row and module addresses. The authors presented complexity, timing and area evaluations for their architecture.

A buffer memory system for a fast and high-resolution graphical display system was proposed in [8]. It provides parallel access to a block, horizontal, vertical, forward-diagonal, and backward-diagonal data pattern in a two-dimensional image array. All pattern sizes are limited to power of two. The address differences of those patterns are specifically prearranged and saved in two SRAMs so that later they can be added to the base address in order to obtain memory module addresses.

Other researches explore memory scheduling of DRAM chips by addressing locality characteristics within the 3D (bank, row, column) memory structure [11]. The solution consists of reordering memory operations in such a way that allows saving clock cycles on precharging banks and accessing successive rows and columns.

In this paper, we propose a solution, which is a substantial extension of the basic rectangularly addressable memory presented in [3] where a scalable data alignment scheme for parallel access of randomly aligned rectangular blocks of data is described. Hardware and critical path complexity were reduced by implicit integration of the module assignment function into the row address function. We propose implementation of the two-dimensional interleaved parallel memory organization with dynamically configurable regular data patterns. We consider an extended set of pattern definition parameters for *discontinuous rectangular block* access. Our method is to split the problem into six trivial sub-problems, which would require hardware implementation with rather low complexity and short critical path. More specifically, the main contributions of the current proposal are as follows:

- Support for an extended set of two-dimensional pattern parameters, including vertical and horizontal group lengths in addition to the respective strides;
- Efficient design implementation (six trivial subtasks, parallel module address generation, explicit row address, low overheads, design scalability).

Theoretical estimations of the design complexity provide that it is linearly proportional to the number of memory modules along one dimension. This is also confirmed by synthesis results made for ASIC 90 nm CMOS technology. Namely, a 32-bit 2×2 organization requires 26 Kgates (14.5% overheads) and a 64-bit 8×8 one – 212 Kgates (3.8% overheads). The operating frequencies vary from 377 MHz for 2x2 32-bit modules down to 310 MHz for 8x8 64-bit modules. A maximum throughput of 1182 Gbit/s was achieved for 8x8 64-bit design.

The reminder of the paper is organized as follows: in Section II, the target data pattern is described. The proposed design implementation and theoretical complexity evaluation is described in Section III. The synthesis results are reported in Section IV. Finally, Section V concludes the paper.

II. TARGET DATA PATTERN DESCRIBTION

Our research mainly targets highly data-parallel applications with regular data patterns, such as various types of audio/video compression (ADPCM, G721, GSM, H.263, MPEG4, JPEG), image processing, pattern analysis or vector calculations. Basic kernels from the above applications very often require data organized in discontinuous rectangular patterns [13].

Fig. 1 illustrates the data pattern parameters we consider for the implementation, namely: $W\in\mathbb{N}$ - data word width in bytes; $w_A\in\mathbb{N}$ - row address width in bits; $b'(vb,hb)\in[0,\mathbb{N}]$ - linear base address of the accessed block with 2D constituents vb and hb; $VS,HS\in\mathbb{N}$ - vertical and horizontal strides (in words); $VGL,HGL\in\mathbb{N}$ - respective group lengths (in words); $VBL,HBL\in\mathbb{N}$ - respective block lengths (in groups); $M\times N\in\mathbb{N}$ - the memory dimensions; $VD\times HD\in\mathbb{N}$ - size of the matrix of memory modules; (i,j) - vertical/horizontal group indices; (k,l) - respective indices of the elements inside a group. Fig. 1 depicts an example of a data pattern of six groups of size $VGL\times HGL=2\times4$ and strides $(VS,HS)=(4,5)$.

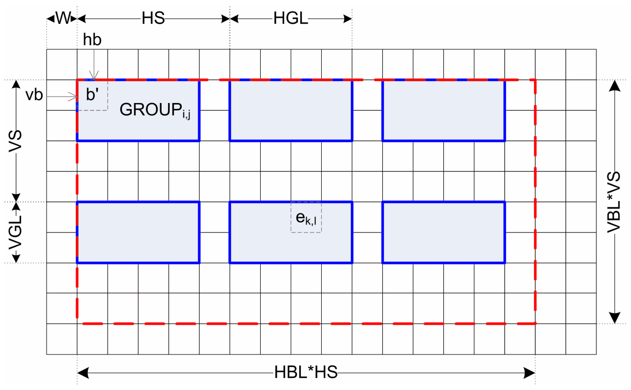


Fig. 1. Considered memory access pattern.

In interleaved memory organizations, the data distribution among the parallel memory modules is determined by a *module assignment function m*. A data element with a linear address *a* is assigned to a memory module according to *m(a)*. A row address function *A* determines the physical address of a data element inside a memory module.

Any data element inside an accessed data block has the following 2D address (va,ha):

$$\begin{aligned} a' &= va\cdot N+ha,\\ va &= vb+i\cdot VS+k=a_{i,k},\\ ha &= hb+j\cdot HS+l=a_{j,l}, \end{aligned} \tag{1}$$

where $i\in[0,VBL-1]$, $k\in[0,VGL-1]$; $j\in[0,HBL-1]$, $l\in[0,HGL-1]$. Equation (1) suggests that the 2D address is completely separable, i.e. its vertical and horizontal constituents are not correlated. Moreover, they are identical. Therefore, in the following discussion we shall skip the direction prefixes 'V' and 'H' in the parameters notations.

The *stride* parameter, being a natural number, can be represented by:

$$S = \sigma \cdot 2^s \in \mathbb{N},$$

where $GCD(\sigma,2)=1$ and $s \in [0,\mathbb{N}]$ (GCD - greatest common divisor). Consequently, S is *odd* when $s=0$, and *even* for $s \in \mathbb{N}$.

III. Proposed Design Implementation

We target an implementation of a memory hierarchy with dynamically adjustable data patterns, to improve the data throughput to the processing units. The considered memory organization contains a memory controller, interleaved memory modules organized in a two-dimensional matrix, special purpose registers (SPRs), and interfaces to the main linearly addressable memory and to the processing units (Fig. 2). The modules are loaded with data from the main memory using conventional mechanisms, such as DMA.

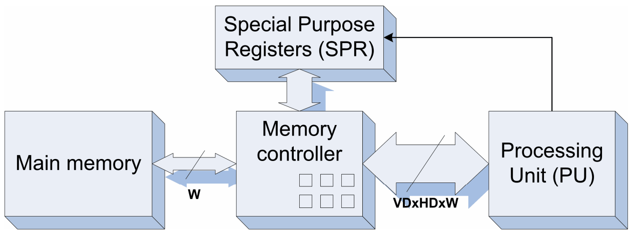


Fig. 2. Interleaved memory organization.

Since no single scheme exists for all strides and group lengths, we propose to partition the problem in a number of cases. This reduces the task to a set of trivial sub-problems (see Fig. 3) that can be implemented with smaller design overheads. The division was performed in a way to ensure minimum memory access latency. The initial problem partitioning is done based on the *stride oddness*. Odd strides can be accessed conflict-free using a basic skewing scheme [1], [3]. Furthermore, all odd and even strides are divided into six subgroups.

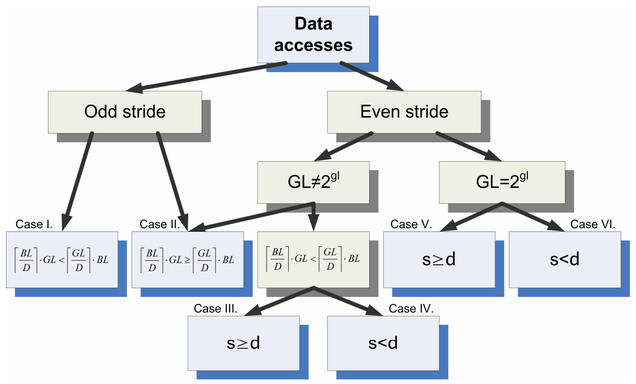


Fig. 3. Problem partition.

The module assignment function for each of the six different cases is presented in Table I. The row address function determines the linear address inside a memory module. Instead of having six different representations of the module assignment function, there is only one. This is another feature helping to reduce complexity of our design.

The row address function is separable and it is described as follows:

$$A(va, ha) = \left\lfloor \frac{va}{VD} \right\rfloor \cdot \left(\frac{N}{HD} \right) + \left\lfloor \frac{ha}{HD} \right\rfloor. \qquad (2)$$

Table I. Module assignment function.

Case #	*Module assignment function*
Cases I-II.	$m_{1,2}(a) = a \bmod D$
Case III.	$m_3(a) = \left(a + \left\lfloor \frac{\lfloor a/D \rfloor}{2^{s-d}} \right\rfloor \right) \bmod D$
Case IV.	$m_4(a) = \left(a + \left\lfloor \frac{a}{D} \right\rfloor \bmod S' \right) \bmod D$
Case V.	$m_5(a) = \left(a + GL \cdot \left\lfloor \frac{\lfloor a/D \rfloor}{2^{s-d}} \right\rfloor \right) \bmod D$
Case VI.	$m_6(a) = (a + \left(GL \cdot \left\lfloor \frac{a}{D} \right\rfloor \right) \bmod S') \bmod D$

Structurally, the interleaved memory consists of an address generation part, a data routing part, and a matrix of memory modules (see Fig. 4). The pattern parameters are stored in Special Purpose Registers (SPRs). The address part is split in identical vertical and horizontal sides and includes the following blocks: mode select unit, address generator, set of row address generators, module assignment unit and address shuffle units. The data routing part consists of a number of shuffle and de-shuffle units. The critical path (highlighted on Fig. 4) comprises the address generator, the module assignment unit, and the decoding part of the shuffle block.

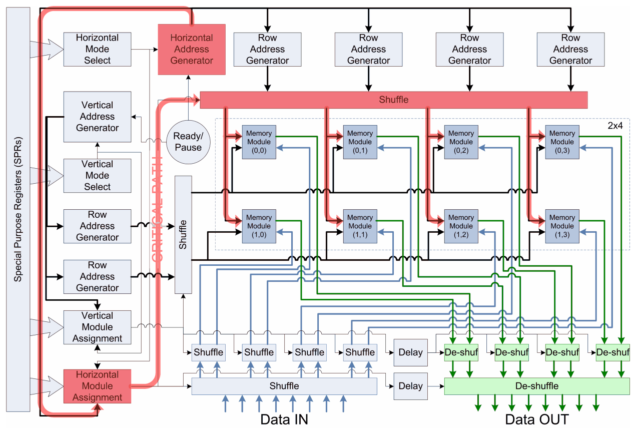


Fig. 4. Parallel memory controller block diagram.

A. *Mode select unit*

The mode select unit sets the address generation logic to a mode, corresponding to the six cases of the problem partitioning (Fig. 3). The pattern parameters S, GL and BL are loaded from the programmable SPRs. The block implements stride oddness check and resolving of two inequalities:

$\left\lceil \frac{BL}{D} \right\rceil \cdot GL \ge \left\lceil \frac{GL}{D} \right\rceil \cdot BL$ and $s \ge d$ (see Fig. 5). The *Counter** includes logic for power of two equality check, i.e. it counts number of logic '1' in the input signal: if there is only single '1', then the input is equal to power of two and the output is set to '1', otherwise the output is set to '0'. The *Coder* block performs coding of four 1-bit signals according to the problem partition diagram (Fig. 3) in order to create 3-bit *Mode* signal.

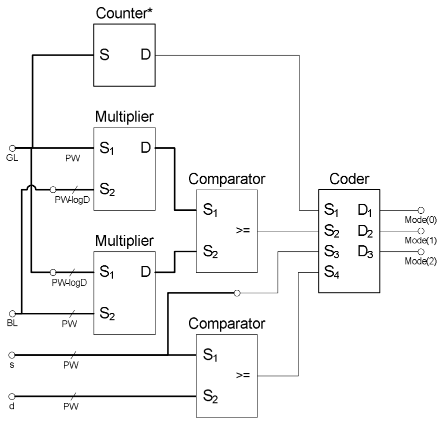


Fig. 5. Mode select unit.

The hardware complexity does not depend neither on the number of memory modules, nor on the data word length.

B. *Address generator*

The address generator produces the vertical and horizontal constituents of the 2D addresses according to (1). Data pattern parameters are loaded from the SPRs and an address mode is loaded from the mode select unit. The address generator consists of two double parallel counters and one simple parallel counter (Fig. 6), that generate sequences of paired indices *(i, k)* or *(j, l)*. The double counters generate group and element indices separately (for cases I-IV), and the single counter generates group and element indices on the base of a common index by implementing respectively division and modulo by group length (for cases V-VI). Note that the group length is equal to power of two for cases V-VI therefore division and modulo operations do not require complex hardware. One side of a parallel double counter is shown on Fig. 7.

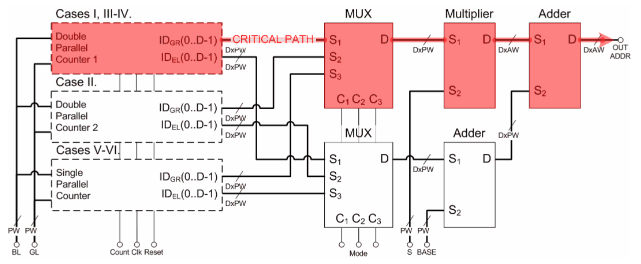


Fig. 6. Address generator.

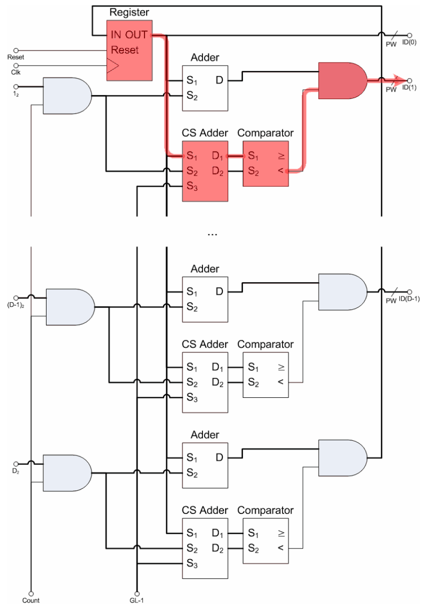


Fig. 7. Parallel counter.

The complexity of this block is *O(*w_A*·D)*. The critical path comprises a double counter, a multiplexer, a multiplier and an adder. The critical path complexity is *O(log D)*.

C. *Row address generator*

The row address generator translates vertical/horizontal constituents of 2D addresses into physical linear addresses inside the memory modules (called row address) according to (2). Since (2) is separable, vertical blocks are connected to upper address bits, and horizontal blocks - to lower bits. No additional logic is needed to implement this block.

D. *Module assignment unit*

The module assignment unit translates vertical/horizontal constituents of 2D addresses into memory module addresses inside the memory modules matrix according to equations from Table I. Data pattern parameters are read from SPRs and an address mode is loaded from the mode select block. The equations are implemented in parallel and their outputs are multiplexed according to the address mode as depicted on Fig. 8.

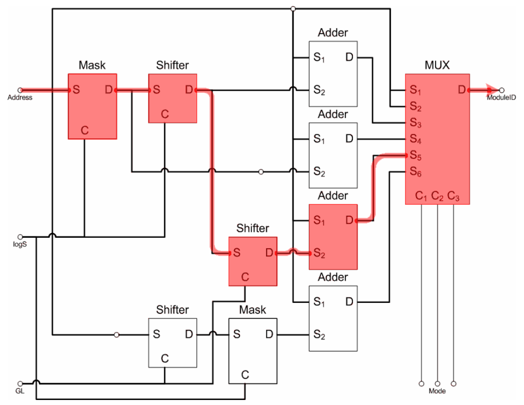


Fig. 8. Module assignment unit.

Complexity of the module assignment function for all cases is presented in Table II. The notation $x_{ms:ls}$ represents the bit interval from the least significant bit ls to the most significant bit ms. The complexity of the complete block is proportional to *O(logD)* because of the adders with input signals of the maximum width equal to *d=logD*.

Table II. Complexity of the module assignment function.

Case #	*Complexity*
Cases I-II.	$m(a) = a_{d-1:0}$
Case III.	$m(a) = \left(a_{d-1:0} + a_{s+d-1:s}\right)_{d-1:0}$
Case IV.	$m(a) = \left(a_{d-1:0} + a_{s+d-1:d}\right)_{d-1:0}$
Case V.	$m(a) = \left(a_{d-1:0} + a_{s+d-1:s} \cdot 2^{gl}\right)_{d-1:0}$
Case VI.	$m(a) = \left(a_{d-1:0} + \left(a_{8 \cdot W-1:d} \cdot 2^{gl}\right)_{s-1:0}\right)_{d-1:0}$

The critical path comprises a masking unit, two shifters, an adder, and a multiplexer. It is mostly influenced by the adder of width *logD* and its complexity is *O(log(logD))*.

E. Shuffle unit

The shuffle unit is used to reorder data according to the module assignment function. It consists of a parallel set of de-multiplexers and output OR-gates.

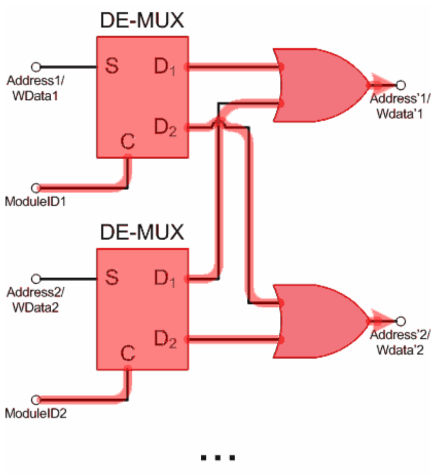


Fig. 9. Shuffle unit.

The complexity of the shuffle unit is *O(w_A·D)* and the critical path does not depend on D or w_A.

F. De-shuffle unit

The de-shuffle unit reorders data back to the initial sequence. It consists of a set of multiplexers and its complexity is *O(w_A·D)*.

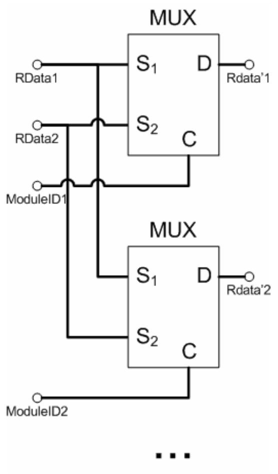


Fig. 10. De-shuffle unit.

As a result, the total complexity of the design forms from the mode select units, address generators, row address generators, module assignment units, shuffles and de-shuffles complexities:

$$\begin{aligned} &O(const) + O(w_A \cdot D) + 0 + O(\log D) + O(w_A \cdot D) + \\ &+ O(w_A \cdot D) = O(w_A \cdot D). \end{aligned} \quad \textbf{(3)}$$

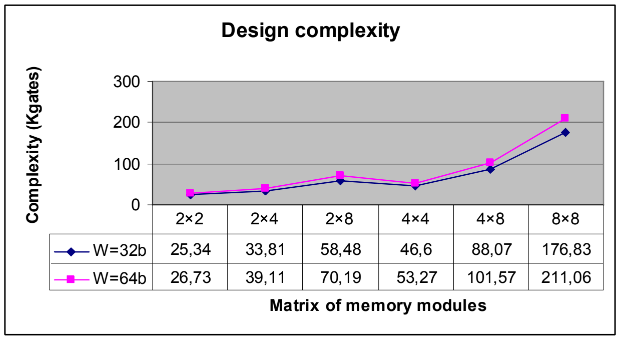
The complete critical path (highlighted on Fig. 4) consists of the address generator, the module assignment unit, and the decoding part of the shuffle block. Thus, it is equal to:

$$O(\log D) + O(\log(\log D)) + O(const) = O(\log D)\,. \quad \textbf{(4)}$$

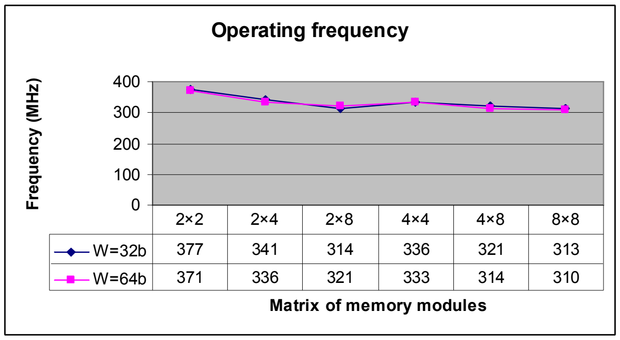
IV. Synthesis Results and Analysis

Our technology independent complexity estimations from (3) and (4) indicate that the critical path complexity is weakly sensitive to the size of the memory matrix and thus the design is well scalable to any matrix size. In fact, the throughput is directly proportional to the matrix size *VD×HD*, and inversely proportional to the critical path, i.e. $throughput \propto W \cdot D^2 / \log D$.

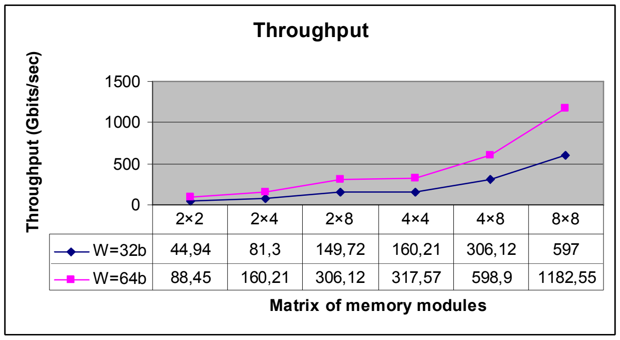
The synthesis was performed for an ASIC 90 nm CMOS technology. The results for six different matrix sizes, word widths *W* to 32 and 64 bits, and 12-bit addresses are presented in Fig. 11. In fact, data word width of 8 Bytes corresponds to utilization of two concurrently coupled 32-bit wide memory modules. Generally speaking, the address width ranging from 8 till 16 bits is enough for the most of practical applications, which would give the complexity range of 45.5-53.9 Kgates for 4×4 matrix with *W* = 32 bits. Considering that the memory modules used in the design have 4096×32bits size and occupy 43.8 Kgates, the logic complexity overheads vary from 14.5% for 2×2 32-bits matrix to 3.8% for 8×8 64-bits matrix with respect to the total hardware complexity. The presented synthesis results confirm the linear increase of the design complexity and the quadratic increase of the throughput, derived from our theoretical estimations. As it was expected, the critical path is proportional to the logarithm of the matrix size along one dimension and the design complexity depends linearly on it.



	2×2	2×4	2×8	4×4	4×8	8×8
◆ W=32b	25,34	33,81	58,48	46,6	88,07	176,83
■ W=64b	26,73	39,11	70,19	53,27	101,57	211,06



	2×2	2×4	2×8	4×4	4×8	8×8
◆ W=32b	377	341	314	336	321	313
■ W=64b	371	336	321	333	314	310



	2×2	2×4	2×8	4×4	4×8	8×8
◆ W=32b	44,94	81,3	149,72	160,21	306,12	597
■ W=64b	88,45	160,21	306,12	317,57	598,9	1182,55

Fig. 11. Synthesis results: design complexity, frequency and throughput.

V. Conclusions

We implemented an interleaved two-dimensional memory organization with dynamically reconfigurable data pattern. Our solution targets highly parallel computing systems such as SIMD machines, vector processors and media accelerators. Our proposal supports an extended set of data pattern parameters enabling parallel access to discontinuous rectangular and, as a particular case, vector data patterns. We made additional efforts at relaxation of all restrictions of pattern parameters even though it led to some design complexity increase comparing to other related solutions with limited applications. More specifically, compared to related art, our design proposal introduced the following new features: support of an extended set of 2D access pattern parameters; programmability of the access patterns via SPRs; arbitrary values of the pattern parameters; minimal memory latency; as well as design scalability. We proved theoretically that the design complexity scales proportionally to $O(w_A \cdot D)$ and the critical path is proportional to $O(\log D)$. Our theoretical conclusions were confirmed by hardware synthesis for 90 nm CMOS technology.

Acknowledgements

This research was partially supported by the Dutch Technology Foundation STW; applied science division of NWO and the Technology Program of the Ministry of Economic Affairs (project DCS.7533); European Commission FP6 (projects SARC and MORPHEUS); and European Doctorate in Information Technology (EDITH).

References

[1] P. Budnik and P.J. Kuck, “The organization and use of parallel memories,” IEEE Trans. on Comp., vol. 20, no. 12, pp. 1566-1569, 1971.

[2] J. Frailong, W. Jalby, and J. Lenfant, “XOR-schemes: A flexible memory organization in parallel memories,” Int. Conf. Parallel Processing, pp. 276-283, 1985.

[3] G. Kuzmanov, G. Gaydadjiev, and S. Vassiliadis, “Multimedia rectangular addressable memory,” IEEE Trans. on Multimedia, vol. 8, no. 2, pp. 315-322, 2006.

[4] H. Shapiro, “Theoretical limitations on the efficient use of parallel memories,” IEEE Trans. Comput., vol. C-27, no. 5, pp. 421-428, 1978.

[5] E. Aho, J. Vanne, and T.D. Hamalainen, “Parallel Memory Architecture for Arbitrary Stride Accesses,” Design and Diagnostics of Elect. Circ. and Syst., pp. 63 – 68, April 18-21, 2006.

[6] K. Kuusilinna, J. Tanskanen, T. Hamalainen, J. Niittylahty, and J. Saarinen, “Configurable parallel memory architecture for multimedia computers,” Journal of Systems Architecture, vol. 47, no. 14-15, pp. 1089-1115, August 2002.

[7] D.T. Harper III, D.A. Linebarger, “Conflict-Free Vector Access Using a Dynamic Storage Scheme,” IEEE Trans. on Comp., vol. 40, no. 3, pp. 276-283, March 1991.

[8] J.W. Park, “An efficient buffer memory system for subarray access,” IEEE Trans. on Parallel and Distrib. Syst., vol. 12, no. 3, pp. 316-335, 2001.

[9] D.T. Harper III, “Block, multistride vector, and FFT access in parallel memory systems,” IEEE Trans. on Parallel and Distrib. Syst., vol. 2, no. 1, pp. 43-51, 1991.

[10] J. Lee, C. Park, and S. Ha, “Memory access pattern analysis and stream cache design for multimedia applications,” Proc. of the ASP-DAC 2003 – Design Automation Conf., pp. 22-27, 2003.

[11] S. Rixner, W.J. Dally, U.J. Kapasi, P. Mattson, and J.D. Owens, “Memory access scheduling,” Proc. of the 27th Int. Symp. on Comp. Arch., pp. 128-138, 2000.

[12] M. Valero, T. Lang, M. Peiron, E. Ayguade, “Conflict-free access for streams in multimodule memories,” IEEE Trans on Comp., vol. 44, no. 5, pp. 634-646, 1995.

[13] E. Aho, J. Vanne, K. Kuusilinna, and T. Hamalainen, “Address computation in configurable parallel memory architecture,” IEICE Trans. Inf. and Syst., vol. E87-D, no. 7, July 2004.