Visual Data Rectangular Memory

Georgi Kuzmanov, Georgi Gaydadjiev, and Stamatis Vassiliadis

Computer Engineering Lab, Microelectronics and Computer Engineering Dept.,
EEMCS, TU Delft, Mekelweg 4, 2628 CD Delft, The Netherlands,
{G.Kuzmanov, G.N.Gaydadjiev, S.Vassiliadis}@EWI.TUDelft.NL
http://ce.et.tudelft.nl/

Abstract. We focus on the parallel access of randomly aligned rectangular blocks of visual data. As an alternative of traditional linearly addressable memories, we suggest a memory organization based on an array of memory modules. A highly scalable data alignment scheme incorporating module assignment functions and a new generic addressing function are proposed. To enable short critical paths and to save hardware resources, the addressing function implicitly embeds the module assignment functions and it is separable. A corresponding design is evaluated and compared to existing schemes and is found to be cost-effective.[1]

1 Introduction

Vector processor designers have been interested in memory systems that are capable of delivering data at the demanding bandwidths of the increasing number of pipelines, see for example [1,6,9,12]. Different approaches have been proposed for optimal alignment of data in multiple memory modules [1, 3, 9–12]. Module assignment and addressing functions have been utilized in various interleaved memory organizations to improve the performance. In graphical display systems, researchers have been investigating efficient accesses of different data patterns: blocks (rectangles), horizontal and vertical lines, forward and backward diagonals [11]. While all these patterns are of interest in general purpose vector machines and graphical display systems, rectangular blocks are the basic data structures in visual data compression (e.g., MPEG standards). Therefore, to utilize the available bandwidth of a particular machine efficiently, new scalable memory organizations, capable of accessing rectangular pixel patterns are needed.

In this paper, we propose an addressing function for rectangularly addressable systems, with the following characteristics: 1.) Highly scalable accesses of rectangular subarrays out of a two-dimensional data storage. 2.) Separable addressing of the memory modules per rows and columns, which potentially saves hardware. We also introduce implicit module assignment functions to further improve the designs. In addition, we propose a memory organization and its interface, which employs conflict free addressing and data routing circuitry with minimal critical path penalties.

[1] This research is supported by PROGRESS, the embedded systems research program of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs, and the Technology Foundation STW (project AES.5021). The authors of this material express special acknowledgements to Jens Peter Wittenburg for his valuable opinions and expertise.

The remainder of the paper is organized as follows. Section 2 motivates the presented research and introduces the particular addressing problem. In Section 3, the addressing scheme and the corresponding memory organization are described. Related work is compared to ours in Section 4. Finally, the paper is concluded with Section 5.

2 Motivation

Most of the data processing in MPEG is not performed over separate pixels, but over certain regions (blocks of pixels) in a frame. Many computationally and data intensive algorithms access such blocks from an arbitrary position in a virtual two-dimensional storage where frames are stored. This generates problems with data alignment and access in system memory, see [7, 8], described formally in the remainder of the section.

Formal Problem Introduction and Proposed Solution. Consider linearly addressable memories (LAM). Pixel blocks with their *upper-left pixel aligned as a byte at a first (word addressing) position of a LAM word will be referred to as aligned*. All other pixel blocks will be referred to as *non-aligned*. Assume a LAM with word length of *w* bits (*w* = 8, 16, 32, 64, 128) and the time for linear memory access to be T_{LAM}. The time to access a single $a \times b$ sub-array of 8-bit pixels, depending on its alignment is:
1.) Aligned sub-array: $\frac{8 \cdot a \cdot b}{w} \cdot T_{LAM}$; 2.) Not aligned sub-array: $(\frac{8 \cdot a}{w} + 1) \cdot b \cdot T_{LAM}$.
The time, required to access *N* $a \times b$ blocks with respect to their alignment will be:
1.) All *N* blocks aligned: $N \cdot \frac{8 \cdot a \cdot b}{w} \cdot T_{LAM}$;
2.) None of the blocks aligned: $N \cdot (\frac{8 \cdot a}{w} + 1) \cdot b \cdot T_{LAM}$;
3.) Mixed: $N \cdot [\frac{1}{a} \cdot \frac{8 \cdot a}{w} + \frac{a-1}{a}(\frac{8 \cdot a}{w} + 1)] \cdot b \cdot T_{LAM} = N \cdot (\frac{8 \cdot a}{w} + 1 - \frac{1}{a}) \cdot b \cdot T_{LAM}$.
By *mixed* access scenario we mean accessing both aligned and non-aligned blocks. We assume that the probability to access an aligned block is $\frac{1}{a}$, while for a non-aligned block it is $\frac{a-1}{a}$. For simplicity, but without losing generality, assume square blocks of $n \times n$, (i.e., *a*=*b*=*n*). We can estimate the total number of LAM cycles to access N square blocks, again with respect to their alignment:

1.) All *N* blocks aligned: $\boxed{\frac{8 \cdot n^2}{w} \cdot N}$; 2.) None of the blocks aligned: $\boxed{(\frac{8 \cdot n^2}{w} + n) \cdot N}$;

3.) Mixed: $\boxed{(\frac{8 \cdot n^2}{w} + n - 1) \cdot N}$. Obviously, the number of cycles to access an $n \times n$ block in a LAM, regardless of its alignment, is a square function of *n*, i.e., $O(n^2)$.

An appropriate memory organization may speed-up the data accesses. Consider the memory hierarchy in Figure 1 and time to access an entire $n \times n$ block from the 2-dimensionally accessible memory (2DAM) to be T_{2DA}. In such a case, the time to access N $n \times n$ sub-blocks in the mixed access scenario will be:

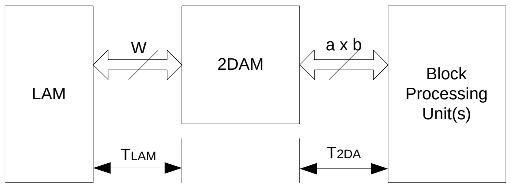


Fig. 1. Memory hierarchy with *2DAM*

$$\frac{N}{n} \cdot \frac{8 \cdot n^2}{w} \cdot T_{LAM} + N \cdot T_{2DA}, [sec] \quad \Leftrightarrow \quad (\frac{8 \cdot n}{w} + \frac{T_{2DA}}{T_{LAM}}) \cdot N, \ [LAMcycles] \ .$$

That is the sum of the time to access the appropriate number of aligned blocks ($\frac{N}{n}$) from LAM plus the time to access all *N* blocks from the 2DAM. It is evident that in a mixed access scenario, the number of cycles to access an $n \times n$ block in the hierarchy from Figure 1 is a linear function of *n*, i.e., *O(n)* and depends on the implementation of the 2D memory array. Table 1 presents access times per single $n \times n$ block. Time is reported in LAM cycles for some typical values of *n* and *w*. There are three cases: 1.) neither of the *N* blocks is aligned - worst case (WC); 2.) mixed block alignment (Mix.); and 3.) all blocks are aligned - best case (BC). The last two columns contain cycle estimations for the organization from Figure 1. In this case, both mixed and best case scenarios assume that aligned blocks are loaded from the LAM to the 2DAM first and then non-aligned blocks are accessed from the 2DAM. The 2DAM worst case (contrary to LAM) assumes that all blocks to be accessed are aligned. Even in this worst case, the 2DAM-enabled hierarchy may perform better than LAM best case if the same aligned block should be accessed more than once. For example, assume accessing *k* times the same aligned block. In LAM, this would take $k \cdot \frac{8 \cdot n^2}{w} = [\frac{8 \cdot n^2}{w} + (k-1) \cdot \frac{8 \cdot n^2}{w}]$, while with 2DAM, it would cost $[\frac{8 \cdot n^2}{w} + (k-1) \cdot \frac{T_{2DA}}{T_{LAM}}]$ LAM cycles per block. Obviously, to have a 2DAM enabled memory hierarchy, faster than pure LAM, it would be enough if $\frac{8 \cdot n^2}{w} > \frac{T_{2DA}}{T_{LAM}}$. All estimations above strongly suggest that *a 2DAM with certain organization may dramatically reduce the number of accesses to the (main) LAM, thus considerably speeding-up related applications*.

Table 1. Access time per $n \times n$ block in LAM cycles. $t = \frac{T_{2DA}}{T_{LAM}}$

n	w	LAM			2DAM	
		WC	Mix.	BC	Mix./BC	WC
8	8	72	71	64	8+t	64+t
	16	40	39	32	4+t	32+t
	32	24	23	16	2+t	16+t
16	8	272	271	256	32+t	256+t
	16	144	143	128	16+t	128+t
	32	80	79	64	8+t	64+t

3 Block Addressable Memory

In this Section, we present the proposed mechanism by describing its addressing scheme, the corresponding memory organization and a potential implementation.

Addressing Scheme. Assume $M \times N$ image data stored in $k = a \times b$ memory modules ($1 \leq a \leq M; 1 \leq b \leq N$). Furthermore, assume that each module is linearly addressable. We are interested in parallel, conflict-free access of $a \times b$ blocks (B) at any (i, j) location, defined as: $B(i,j) = \{(i+p, j+q) | 0 \leq p < a, 0 \leq q < b\}$,$0 \leq i \leq M - a, 0 \leq j \leq N - b$. To align data in *k* modules without data replication, we organize these modules in a two-dimensional $a \times b$ matrix. A module assignment function, which maps a piece of data with 2D coordinates *(i,j)* in memory module $(p,q) : 0 \leq p < a, 0 \leq q < b$, is required. We separate the function denoted as $m_{p,q}(i,j)$, into two mutually orthogonal assignment functions $m_p(i)$ and $m_q(j)$. We define the following module assignment functions for each module at position *(p,q)*:

$$m_p(i) = (i - p) \, mod \, a, \quad m_q(j) = (j - q) \, mod \, b \, . \tag{1}$$

The addressing function for module *(p,q)* with respect to coordinates *(i,j)* is defined as:

$$A_{p,q}(i,j) = (i\ div\ a + c_i) \cdot \frac{N}{b} + j\ div\ b + c_j\ , \tag{2}$$

$$c_i = \begin{cases} 1, i\ mod\ a > p \\ 0, otherwise\ ; \end{cases} \quad c_j = \begin{cases} 1, j\ mod\ b > q \\ 0, otherwise\ . \end{cases}$$

Obviously, if $p = a - 1 \Rightarrow c_i = 0$ for $\forall i$; if $q = b - 1 \Rightarrow c_j = 0$ for $\forall j$, respectively. In essence, c_i and c_j are the module assignment functions, implicitly embedded into the linear address $A_{p,q}(i,j)$. The proof of all properties of the proposed addressing scheme can be found in [7].

Memory Organization. The key purpose of the proposed addressing scheme is to enable performance-effective memory implementations optimized for algorithms requiring the access of rectangular blocks. Designs with shortest critical paths are to be considered with the highest priority, as they dictate machine performance. Equations (1)-(2) are generally valid for any natural values of parameters *a*, *b* and *N* (i.e., $for\ \forall\ a, b, N\ \in \mathbb{N}$). To implement the proposed addressing and module assignment functions, however, we will consider practical values of these parameters. Since pixel blocks processed in MPEG algorithms have dimensions up to 16×16, values of practical significance for parameters *a* and *b* are the powers of two up to 16 (i.e., 1, 2, 4, 8, 16). Figure 2 illustrates an example for a block size of $a \times b = 2 \times 4$.

Module Addressing. An important property of the proposed module addressing function is its *separability*. It means that the function can be represented as a sum of two functions of a *single* and *unique* variable each (i.e., variables *i* and *j*). The separability of $A_{p,q}(i,j) = Ai_p(i) + Aj_q(j)$ allows the address generators to be implemented per column and per row (see Figure 2) instead of implemented as individual addressing circuits for each of the memory modules.

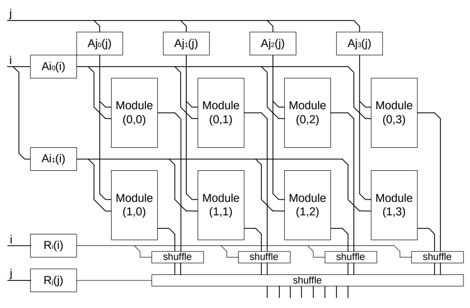
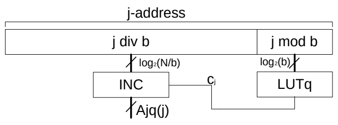


Fig. 2. 2DAM for *a=2, b=4* and $N = 2^n \geq 16$



(a) Generation Circuit of q-addresses for $1 \leq q < b$

j mod b		c_j			i mod a	c_i
		q=0	q=1	q=2		p=0
0	0	0	0	0	0	0
0	1	1	0	0	1	1
1	0	1	1	0	-	-
1	1	1	1	1	-	-

(b) LUTs contents for *a=2, b=4*

Fig. 3. Module address generation

The requirements for the frame sizes of all MPEG standards and for Video Object Planes (VOPs) [2] in MPEG-4 are constituted to be multiples of 16, thus, *N* is a multiple of 2^4. For the assumed values of *N* and *b*, further analysis of Equation (2) suggests that $j\ div\ b + c_j < \frac{N}{b}$ and $(j\ div\ b + c_j)_{max} = \frac{N}{b} - 1$, i.e., no carry can be ever generated between $Ai_p(i)$ and $Aj_q(j)$. Therefore, we can implement $A_{p,q}(i,j)$ for every module *(p,q)* by simply routing signals to the corresponding address generation blocks without actually summing $Ai_p(i) + Aj_q(j)$. Figure 3(a) illustrates address generation circuitry of *q*-addresses ($Aj_q(j)$) for all modules except the first ($1 \leq q < b$). With respect to (2), if c_j is 1 the quotient *j div b* should be incremented by one, otherwise it should not be changed. To determine the value of c_j, a Look-Up-Table (LUT) with *j mod b* inputs can be used. For the assumed practical values of *a* and *b* (≤ 16), such a LUT would have at most 4 inputs, i.e., c_j *is a binary function of at most 4 binary digits*. Row *p*-addresses are generated identically. For *p=1* or *q=3*, $c_i = 0$, $c_j = 0$ respectively. Therefore, address generation in these cases does not require a LUT and an incrementor. Instead, it is just routing *i div a* and *j div b* to the corresponding memory ports, i.e., blocks $Ai_1(i)$ and $Aj_3(j)$ in Figure 2 are empty. Figure 3(b) depicts all 4 LUTs for the case $a \times b = 2 \times 4$. The usage of LUTs to determine c_i and c_j is not mandatory, fast pure logic can be utilized instead.

Data Routing Circuitry. In Figure 2, the shuffle blocks, together with blocks $R_p(i)$ and $R_q(j)$, illustrate the data routing circuitry. The shuffle blocks are in essence circular barrel shifters, i.e. having the complexity of a network of multiplexors. An $n \times n$ shuffle is actually an $n \rightarrow 1$ n-way multiplexor. In the example from Figure 2, the *i*-level shuffle blocks are four ($2 \rightarrow 1$) 16-bit multiplexors and the *j*-level one is ($4 \rightarrow 1$) 64-bit. To control the shuffle blocks, we can use the module assignment functions for $p = q = 0$, i.e., $R_i(i) = i\ mod\ a$ and $R_j(j) = j\ mod\ b$. These functions calculate the *(p,q)*-coordinates of the "upper-left" pixel of the desired block, i.e., pixel *(i,j)*. For the assumed practical values of *a* and *b* being powers of two, the implementation of $R_i(i)$ and $R_j(j)$ is simple routing of the least-significant $log_2(a)$ -bits (resp. $log_2(b)$) to the corresponding shuffle level.

LAM Interface. Figure 4 depicts the organization of the interface between LAM and 2DAM (recall Figure 1) for the modules considered in Figure 2. The data bus width of the LAM is denoted by W (in number of bytes). In this particular example, W is assumed to be 2, therefore modules have coupled data busses. For each *(i,j)* address, the AGEN block sequentially generates addresses to the LAM and distributes write enable (WE) signals to a corresponding module couple. Two module *WE* signals (WE_i, WE_j) are assumed for easier row and column selection. In the general case, the AGEN block should sequentially generate $\frac{a \cdot b}{W}$ LAM addresses for each *(i,j)* address. Provided that pixel data is stored into LAM in scan-line manner and assuming that only aligned blocks will be accessed from the LAM (i.e., *(i,j)* are aligned), the set of LAM addresses to be generated is defined as follows:
$A_{LAM}(i,j) = (i+k) \cdot N + j + l \cdot W, \quad k = 0,1,...,a-1; l = 0,1,...,\frac{b}{W} - 1$.
In the 2DAM, the data words should be simultaneously written in modules:
$(p,q) = (k, l \cdot W), (k, l \cdot W + 1), ..., (k, l \cdot W + W - 1)$ at local module address:
$A^{LAM}_{p,q}(i,j) = (i\ div\ a) \cdot \frac{N}{b} + j\ div\ b$. Note, that accessing only aligned blocks from the LAM enables thorough bandwidth utilization. When only aligned blocks are addressed,

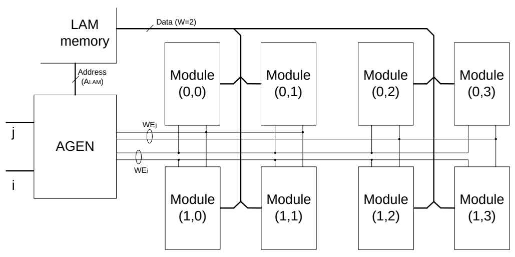


Fig. 4. LAM interface for *W=2, a=2, b=4*

all address generators issue the same address, due to (2). Therefore, during write operations into 2DAM, the same addressing circuitry can be used as for reading. If the modules are true dual port, the write port addressing can be simplified to just proper wiring of both *i* and *j* address lines because the incrementor and the LUTs from Figure 3(a) are not required. Therefore, module addressing circuitry is not depicted in Figure 4.

Critical Paths. Regarding the performance of the proposed design, we should consider the created critical path penalty. Assuming generic synchronous memories where addresses are generated in one cycle and data are available in another, we separate the critical paths into two: address generation and data routing. For the proposed circuit implementation, the address generation critical paths are the critical path of either a $log_2(\frac{M}{a})$-bit or a $log_2(\frac{N}{b})$-bit adder, whichever is longer, and the critical path of one (max. 4-input) LUT. The data routing critical path is the sum of the critical paths of one $a \longrightarrow 1$ multiplexor and one $b \longrightarrow 1$ multiplexor. More details regarding the implementation of the memory organization and a case study design can be found in [7].

4 Related Work and Comparisons

Two major groups of memory organizations for parallel data access have been reported in literature - organizations with and without data replication (redundancy). We are interested only in those without data replication. Another division is made with respect to the number of memory modules - equal to the number of accessed data points and exceeding this number. Organizations with a prime number of memory modules can be considered as a subset of the latter. An essential implementation drawback of such organizations is that their addressing functions are non-separable and complex, thus slower and costly to implement. We have organized our comparison with respect to block accesses, discarding other data patterns, due to the specific requirements of visual data compression. To compare designs, two basic criteria have been established: scalability and implementation drawbacks in terms of speed and/or complexity. Comparison results are reported in Table 2. Budnik and Kuck [1] described a scheme for conflict free access of $\sqrt{N}\times\sqrt{N}$ square blocks out of $N\times N$ arrays, utilizing $m > N = 2^n$ memory modules, where *m* is a prime number. Their scheme allows the complicated full crossbar switch as the only possibility for data alignment circuitry and many costly *modulo(m)* operations with *m* not a power of two. In a publication, related to the development of the Burroughs Scientific Processor, Lawrie [9] proposes an alignment scheme with data switching, simpler than a crossbar switch, but still capable to handle only $\sqrt{N} \times \sqrt{N}$ square blocks out of *m=2N* modules, where $N = 2^{2n+1}$. Both schemes in [1] and [9] require a larger number of modules than the number of simultaneously accessed (image) points (*N*). Voorhis and Morin [12] suggest various addressing functions considering $p \times q$ subarray accesses and different number of memory modules m: both $m = p \times q$ and $m > p \times q$. Neither of the functions proposed in [12] is separable, which leads to an extensive number of address generation and module assignment logic blocks. In [3] the authors propose a module assignment scheme based on Latin squares, which is capable of accessing $\sqrt{N} \times \sqrt{N}$ square blocks out of $N \times N$ arrays, but not from random positions. Similar drawbacks has the scheme proposed in [10]. A display system memory, capable of simultaneous access of $p \times q$ rectangular subarrays is described in [11]. The design, proposed there, utilizes a prime number of memory modules, which enables accesses to numerous data patterns, but disallows separable addressing functions. Therefore, regarding block accesses, it is slower and requires more memory modules than our proposal. Large LUTs (in size and number) and a yet longer critical path with consecutive additions can be considered as other drawbacks of [11]. A memory organization, capable of accessing $N \times N$ square blocks, aligned into $(1 + N)^2$ memory modules was described in [5]. The same scheme was used for the implementation of the matrix memory of the first version of HiPAR-DSP [13]. Besides the restriction to square accesses only, that memory system uses a redundant number of modules, due to additional DSP-specific access patterns considered. A definition of a rectangular $p \times q$ block random addressing scheme from the architectural point of view dedicated for multimedia systems was introduced in [8], but no particular organization was presented there. In the latest version of HiPAR16 [4], the matrix memory was improved so that a restricted number of rectangular patterns could also be accessed. This design, however, still utilizes an excessive number of memory modules as *p* and *M* respectively *q* and *N* should not have common divisors. E.g., to access a 2×4 pattern, the HiPAR16 memory requires $3 \times 5 = 15$ memory modules, instead of only 8 for ours. The memory of [4] requires a complicated circuitry. Both [4] and [13] assume separability, however, the number of utilized modules is even higher than the closest prime number to $p \times q$. Compared to [1, 3–5, 9–11, 13], our scheme enables a higher scalability and a lower number of memory modules. This reflects to the design complexity, which has been proven to be very low in our case. Address function separability reduces the number of address generation logic and critical path penalties, thus enables faster implementations. Regarding address separability, we differentiate from [1, 3, 9–12], where address separability is not supported. As a result, *our memory organization is envisioned to have the shortest critical path penalties among all referenced works*.

Table 2. Comparison to other proposed schemes

Related Work	scalability	# modules (m)	implementation drawbacks or limitations
Budnik, Kuck [1]	$\sqrt{N} \times \sqrt{N}$ from $N \times N$	prime $m > N = 2^n$	*mod(m)*, crossbar, no addressing
Lawrie [9]	$\sqrt{N} \times \sqrt{N}$	$m = 2.N; N = 2^{2n+1}$	*mod(m)*, no addressing
Voorhis, Morin [12]	$p \times q$ from $M \times N$	$m \geq p \times q$	not separable,*mod(pq)*,*mod(pq+1)*,
Kim, Prasanna [3]	$\sqrt{N} \times \sqrt{N}$ from $N \times N$	*m* = *N*	certain blocks are inaccessible
De-lei Lee [10]	$\sqrt{N} \times \sqrt{N}$ from $N \times N$	*m* = *N*	many modules for higher *N*
Park [11]	$p \times q$ from $M \times N$	prime $m > p \times q$	not separable, many adders, big LUTs
HiPAR-DSP [5, 13]	$N \times N$	$m = (1 + N)^2$	$2 \times N + 1$ additional modules, *mod(m)*
HiPAR-DSP16 [4]	$p \times q$ from $M \times N$	$m >> p \times q$	big number of modules, *mod(m)*
This proposal	$p \times q$ from $M \times N$	$m = p \times q$	none of the above, rectangular patterns only

5 Conclusions

We presented a scalable memory organization capable of addressing randomly aligned rectangular data patterns in a 2D data storage. High performance is achieved by a reduced number of data transfers between memory hierarchy levels, efficient bandwidth utilization, and short hardware critical paths. In the proposed design, data are located in an array of byte addressable memory modules by an addressing function, implicitly containing module assignment functions. An interface to a linearly addressable memory has been provided to load the array of modules. Theoretical analysis proving the efficiency of the linear and the two-dimensional addressing schemes was also presented. The design is envisioned to be more cost-effective compared to related works reported in the literature. The proposed organization is intended for specific data intensive algorithms in visual data processing, but can also be adopted by other general purpose applications with high data throughput requirements including vector processing.

References

1. P. Budnik and D. J. Kuck. The organization and use of parallel memories. *IEEE Transactions on Computers*, 20(12):1566–1569, December 1971.
2. ISO/IEC JTC11/SC29/WG11, N3312. MPEG-4 video verification model version 16.0.
3. K. Kim and V. K. Prasanna. Latin squares for parallel array access. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):361–370, 1993.
4. H. Kloos, J. Wittenburg, W. Hinrichs, H. Lieske, L. Friebe, C. Klar, and P. Pirsch. HiPAR-DSP 16, a scalable highly parallel DSP core for system on a chip: video and image processing applications. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 3112–3115, Orlando, Florida, USA, May 2002. IEEE.
5. J. Kneip, K. Ronner, and P. Pirsch. A data path array with shared memory as core of a high performance DSP. In *Proceedings of the International Conference on Application Specific Array Processors*, pages 271–282, San Francisco, CA, USA, August 1994.
6. P. M. Kogge. *The Architecture of Pipelined Computers*. McGraw-Hill, 1981.
7. G. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis. Multimedia rectangularly and separably addressable memory. Technical Report CE-TR-2004-01, TU Delft, Delft, January 2004. http://ce.et.tudelft.nl/publications.php.
8. G. Kuzmanov, S. Vassiliadis, and J. van Eijndhoven. A 2D Addressing Mode for Multimedia Applications. In *Workshop on System Architecture Modeling and Simulation (SAMOS 2001)*, volume 2268 of *Lecture Notes in Computer Science*, pages 291–306. Springer-Verlag, 2001.
9. D. H. Lawrie. Access and alignment of data in an array processor. *IEEE Transactions on Computers*, C-24(12):1145–1155, December 1975.
10. D. Lee. Scrambled Storage for Parallel Memory Systems. In *Proc.IEEE International Symposium on Computer Architecture*, pages 232–239, Honolulu, HI, USA, May 1988.
11. J. W. Park. An efficient buffer memory system for subarray access. *IEEE Transactions on Parallel and Distributed Systems*, 12(3):316–335, March 2001.
12. D. C. van Voorhis and T. H. Morrin. Memory systems for image processing. *IEEE Transactions on Computers*, C-27(2):113–125, February 1978.
13. J. P. Wittenburg, M. Ohmacht, J. Kneip, W. Hinrichs, and P. Pirsh. HiPAR-DSP: a parallel VLIW RISC processor for real time image processing applications. In *3rd International Conference on Algorithms and Architectures for Parallel Processing, 1997. ICAPP 97.*, pages 155–162, Melbourne, Vic. , Australia, December 1997.