Exploiting Contemporary Memory Techniques in Reconfigurable Accelerators

R.W. Hartenstein, M. Herz, T. Hoffmann, U. Nageldinger

University of Kaiserslautern
Erwin-Schrödinger-Straße, D-67663 Kaiserslautern, Germany
Fax: ++49 631 205 2640, email: abakus@informatik.uni-kl.de
www: http://xputers.informatik.uni-kl.de

Abstract. This paper discusses the memory interface of custom computing machines. We present a high speed parallel memory for the MoM-PDA machine, which is based on the Xputer paradigm. The memory employs DRAMs instead of the more expensive SRAMs. To enhance the memory bandwidth, we use a threefold approach: modern memory devices featuring burst mode, an efficient memory architecture with multiple parallel modules, and memory access optimization for single applications. To exploit the features of the memory architecture, we introduce a strategy to determine optimized storage schemes for a class of applications.

1 Introduction

Custom computing machines [1][2] have been a promising research area for the recent years. These systems have shown the potential to achieve high performance for a variety of applications. Combining conventional microprocessors with accelerators featuring field programmable logic, custom computing machines make use of both traditional sequential code for the processor and structural code for the reconfigurable accelerator.

Naturally, the research on reconfigurable computing has focused mainly on the accelerator architectures themselves. However, experiences from traditional computers have shown, that the overall system performance depends also strongly on the peripheral parts. Especially the memory interface has shown to be a potential or existing bottleneck, which can extremely reduce the performance of the system.

The custom computing scene has avoided this problem by building the memory with fast SRAMs. However, SRAMs are still bigger and more expensive than DRAMs. Thus, it is worth thinking about using widespread DRAMs for large memories to be used for reconfigurable computing. As DRAMs are much slower than SRAMs, additional concepts have to be developed to speed the memory up.

For processors based on the so-called von Neumann paradigm, different methods have been developed to increase the memory bandwidth. The approaches can be distinguished into the following groups:

- Memory devices
- Memory architecture
- Compiler techniques

The most direct way to enhance memory access is the development of faster memory devices. This approach aims to reduce the physical data access time. In this field, a large variety of devices, especially in the DRAM sector, have been developed. Often, those devices also offer enhanced data access for consecutive data words (burst mode), with fixed or variable length.

Apart from the development of fast memory devices, special memory architectures have evolved to increase the memory bandwidth. The best known examples are caches [3] and interleaved memories with skewing schemes [4]. Caching relies on the locality principle, which is basically an observation of normal program behavior. Though caches have shown to provide a remarkable speedup to memory accesses, the real efficiency of a cache depends on the application and can hardly be predicted. For interleaved memories, the classic examination by Budnik and Kuck [4] showed, that this architecture provides speedup only for certain access sequences. This is still true, although there has been remarkable research to increase the number of access sequences which take advantage of the interleaved memories, e.g. [5][6].

The third approach for faster memory access uses compiling techniques to adapt applications for utilizing memory architectures. Such techniques have been published for multiprocessors with shared address space [8], where the performance of a shared cache is enhanced by reducing disadvantageous cache-line replacements. Also, loop transformations can be used to increase the data locality in caches [7].

In this paper, we present an approach to improve the memory access for a reconfigurable architecture. Our target hardware is the MoM-PDA (Map-oriented Machine with Parallel Data Access) [9], which is based on the Xputer machine paradigm [10]. In our approach, we combine the three basic techniques mentioned above: We use a special memory architecture with parallel banks, which provides global speedup by supporting the execution model of the architecture. The memory is built of modern Multibank DRAM (MDRAM) devices [11], which feature support for interleaving by their internal structure as well as a fast burst mode for access of consecutive data words. Furthermore, we introduce strategies to utilize the memory architecture by examination of the data access sequences of applications and rearrangement of the application data.

The paper is structured as follows: In the next section, the target hardware MoM-PDA and the basic concepts of Xputers are introduced. Then, the compilation framework for the architecture is sketched briefly. In Section 4, a memory architecture for the MoM-PDA is described. After this, a data rearrangement strategy for this architecture is proposed. In the next section, performance results are given, and finally, the paper is concluded.

2 Target Architecture MoM-PDA

The MoM-PDA is an architecture based on the Xputer machine paradigm [10]. This non von Neumann paradigm provides high computation power for a large variety of computations. Especially for algorithms, where the same operation is performed on a big amount of data, Xputer-based accelerators achieve high speedup factors [12]. Such algorithms are found in many applications from multimedia, image and digital signal processing. The basic components of an Xputer-based machine and their interconnect are shown in Fig. 1.

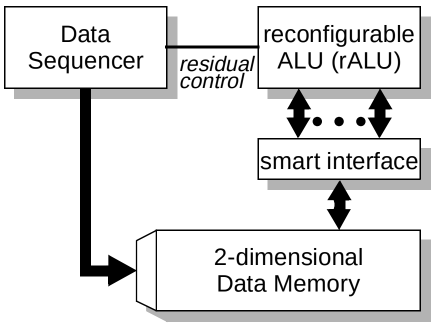


Fig. 1. Basic structure of the MoM-PDA architecture

The main difference between the Xputer and the so-called von Neumann paradigm is, that an Xputer is controlled by a data stream rather than by an instruction stream. Thus, an Xputer features a data sequencer instead of an instruction sequencer. At run time, the data sequencer generates an address stream for the memory according to the data access sequence of the application. The resulting data stream is then fed into a reconfigurable ALU (rALU), which performs the required computation onto the data. The rALU is coupled to the memory by a smart memory interface, which contains a register file to store intermediate results and data, which is needed several times. This way, memory cycles are saved.

To execute an application, both the data sequencer and the reconfigurable ALU need to be configured before the computation. Once this is done, the machine can process a large amount of data without a change in this configuration. Thus, no instructions need to be fetched from memory during execution. The memory is used only for data. To make the mapping of multidimensional data arrays easier, the data memory is organized two-dimensionally. In contrast to traditional processors, which have an instruction set depending on the capabilities of the ALU, the data sequencer of the Xputer is only very loosely coupled to the reconfigurable ALU. So, the rALU and the data sequencer can be exchanged easily.

For the current prototype MoM-PDA, a novel sequencer hardware [9] and a novel memory architecture have been developed. The sequencer can implement a large number of generic address sequences. For the description of one sequence, only a few parameters are necessary. Multiple sequences can be concatenated or nested. The data memory comprises several (two in the prototype) parallel memory modules, which allow parallel access to data words. Each module is connected to the rALU via an own data bus and to the data sequencer via an own address bus. The data sequencer is capable of generating two independent address streams for each module.

The reconfigurable ALU of the MoM-PDA is implemented using the coarse-grained KressArray [13]. The KressArray resembles a mesh-connected array of configurable processing elements. In contrast to FPGAs, which offer only a one bit wide datapath, the processing elements can implement all operators of the language C for 32 bit words. Thus, the KressArray can easily implement a datapath for an application specified in a high level language.

The general execution model of an Xputer, which applies also to the MoM-PDA, is illustrated in Fig. 2. In many applications, the application data is typically arranged in

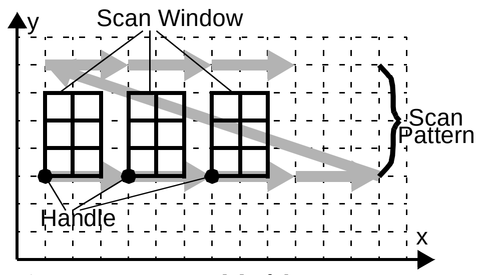


Fig. 2. Execution model of the Xputer

arrays, which may be one- or multidimensional. The data is traversed in nested loops, where a certain computation is performed in the loop body. In order to execute such an application on an Xputer-based machine, first the data arrays are mapped onto the two dimensional data memory. This arrangement of the application data is called data map. The data words, which are consumed or created by one execution of the datapath in the rALU are contained in a rectangular subarea of the data map, which is called scan window. The location of the scan window inside the data memory is determined by a reference position, the so-called handle (see Fig. 2). The traversal of the application data, which is normally done by loops, is performed by moving the scan window over the data map and performing the computation configured in the rALU in each step. The resulting movement pattern of the scan window is called scan pattern.

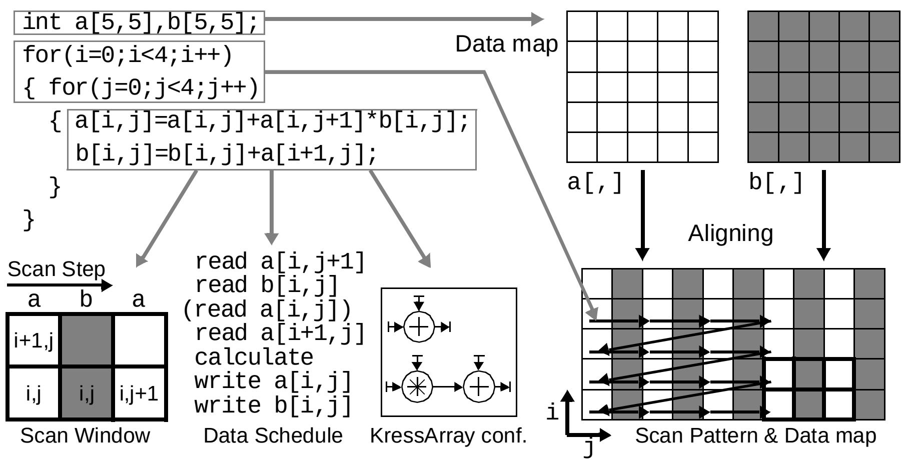
According to the execution model described above, a data map, a scan pattern and a rALU configuration have to be supplied in order to process an application on an Xputer. A compilation environment for this purpose is sketched in the next section.

3 Compilation Framework for the MoM-PDA

For the mapping of applications onto the MoM-PDA and other Xputer-based accelerators, the compilation environment CoDe-X [14] has been developed. CoDe-X is capable to compile applications specified in a high level language onto a system comprising a host and an Xputer-based accelerator.

CoDe-X accepts a description in a superset of the C language. The input program is in the first step partitioned in a set of tasks for the host and a set of tasks for the accelerator. This partitioning is driven by a performance analysis, which minimizes the overall execution time. Also, the first level partitioner performs loop transformations on the code to improve the performance.

The accelerator tasks are then processed by the X-C [15] compiler, which accepts a



```
int a[5,5],b[5,5];
for(i=0;i<4;i++)
{ for(j=0;j<4;j++)
   { a[i,j]=a[i,j]+a[i,j+1]*b[i,j];
     b[i,j]=b[i,j]+a[i+1,j];
   }
}
```

```
 read a[i,j+1]
 read b[i,j]
(read a[i,j])
 read a[i+1,j]
 calculate
 write a[i,j]
 write b[i,j]
```

Fig. 3. Mapping of an application onto an Xputer

subset of X-C as input language. A course sketch of this second compilation step is illustrated by an example in Fig. 3. The X-C compiler performs a second level of partitioning by dividing the input up into code for the data sequencer and code for the rALU comprising the KressArray. First, the data arrays of the application are mapped onto the data memory. Basically, each array becomes a two-dimensional area and needs a scan pattern to access the data. If the loop structure allows it, several arrays can be aligned [15] to form one data area, so only one scan pattern is needed for multiple arrays (see Fig. 3). The scan pattern is generated from the loop structure. In the example, there are two nested loops whereof a so-called video scan is derived.

The loop body holds the information for the calculation to be performed onto the data. The variable references in the loop body combined with the scan pattern determine the scan window for the application. In the example in Fig. 3, there is only one scan window due to the alignment of the two data arrays. The functions in the body itself are then transformed into a KressArray configuration by the Datapath Synthesis System (DPSS) [16]. For the topic of this paper, the configuration and the computational structure is of less importance. However, according to this structure and the ordering of the operators, the DPSS also determines an optimal data schedule. This schedule describes the sequence for reading and writing the data words inside the scan window. In the example in Fig. 3, the value of a[i,j] is the same as a[i,j+1] from the previous iteration. This can be seen by the overlap of two consecutive scan windows by one column. The DPSS recognizes this overlap and generates three schedules for the start of a new scan line, the positions in-between, and the last position of a line. The value a[i,j] has to be read from memory only at the beginning of each scan line. For all other positions, the according value from the last step is taken, which has been stored in the smart memory interface. In Fig. 3, this situation is illustrated by the parentheses around the read operation for a[i,j] in the data schedule.

It can be seen from the description above, that the smart memory interface performs a similar task as a cache in a conventional computer. The difference is, that this caching is not indeterministic. Instead, the reusable data is identified by the DPSS and only this data is stored. Thus, the smart memory interface needs much less capacity than a normal cache.

4 Memory Architecture

The data memory of the MoM-PDA uses a novel architecture, which supports the execution model and enables high-speed access. As described above, the logical organization of the memory is two-dimensional. This suits the execution model of the Xputer paradigm with two-dimensional scan patterns. It has shown, that this concept supports the mapping of many algorithms, e.g. from image processing, which are performed on multidimensional arrays.

The architecture of the data memory should provide a global means of improvement, like a cache or interleaved memory, so that one can expect a higher bandwidth for common applications. Furthermore, the architecture should support the enhancement of memory access for single applications by data rearrangement strategies. The following speedup features are to be considered:

- Multiple parallel memory modules (two in the prototype) with each using its own bus. This allows concurrent access to two data words.
- Interleaved memory banks, supporting access to sequential data words.
- More support for sequential data words by using modern high-speed memory devices, which offer a burst mode.

For the implementation of the memory architecture, we selected MDRAMs [11] from Siemens. An MDRAM features internally up to 36 independent banks. For our architecture, we have chosen devices with 32 banks. Reading and writing to and from a bank is done in bursts with variable length. An access to a bank requires an activation command, followed by the desired operation (read or write) and concluded by a precharge. The 32 banks can be activated independently from each other.

The resulting memory architecture using two parallel modules consisting of MDRAMs with 32 banks is shown in Fig. 4. The rows of the two-dimensional memory are alternately mapped onto the two memory modules, allowing parallel data access for two adjacent rows. Rows, which are in the same module are assigned to different banks of the MDRAM, allowing interleaved memory access within a module. Within a row, which resembles an MDRAM bank, the burst mode can be used. Thus, the memory architecture offers different speedup features in both dimensions. To address the memory modules, the two parrallel address streams from the data sequencer pass a burst control unit, which

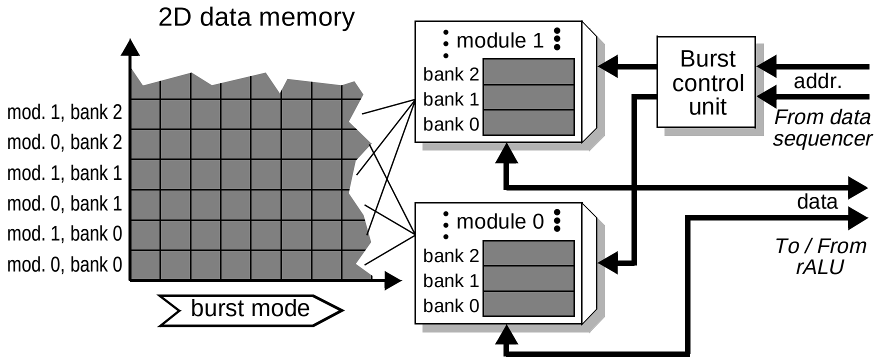


Fig. 4. Two-dimensional memory architecture

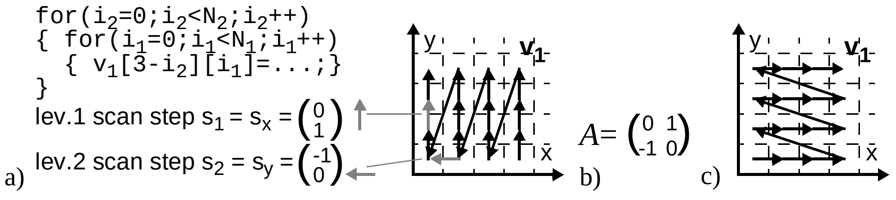
handles memory refresh and bursts. As the burst length is limited to 32 words, the burst control unit has to split bursts, if they reach over two horizontally neighbored banks.

5 Data Rearrangement Strategy

The memory architecture described above provides a basic means of enhancing the memory access. However, the memory bandwidth depends on the arrangement of the data according to the corresponding scan pattern. For many applications, this arrangement can be changed in a way to exploit the memory architecture, resulting in a higher bandwidth.

In this section, we will show for a specific application class, how the memory bandwidth can be enhanced. For simplicity, the algorithm is shown for max. two-dimensional data arrays, but the methods can be adapted to higher dimensions. We will utilize the compiler environment, which provides the data arrays and the according access sequences of an application in explicit form (see Fig. 3). Hereby, we assume, that the data arrays are not yet aligned.

Before the optimization strategies are discussed, we will introduce some naming conventions. Although all the information for the algorithm are provided as scan pattern parameters, we will mainly use terminology from the original loop structure to improve understandability. A two-dimensional loop nest L with index variables $i_1,...,i_n$, i_1 being the index of the innermost loop, can be rewritten so that each loop starts at 0 and ends at its limit N_l-1, l=1,...,n (we use C notation). The loop body contains variables $v_1,...,v_m$, which are generated or consumed, depending if they appear on the left or right side of an assignment. The set of v_k in L, which are generated is called GV(L), the set of consumed v_k is called CV(L). Each v_k is addressed by an index function $f_k(i_1,...,i_n)$: $Z^n \rightarrow Z^2$ with $f_k(i_1,...,i_n) \rightarrow (x,y)$. An example loop nest is shown in Fig. 5a), with one variable v_1. The index function for v_1 is $f_1(i_1,i_2) = (3-i_2,i_1)$.


```
for(i2=0;i2<N2;i2++)
{ for(i1=0;i1<N1;i1++)
  { v1[3-i2][i1]=...;}
}
```
lev.1 scan step $s_1 = s_x = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

a) lev.2 scan step $s_2 = s_y = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$

$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ b) c)

Fig. 5. Example application and scan pattern (a), transformation matrix (b), opimized scan pattern (c)

The index function $f_k(i_1,...,i_n)$ defines vectors determining the step direction of the scan pattern for v_k. There is one vector for each index variable i_j , j=1,...,n. The vector for the innermost loop is called level 1 scan step or s_1, the vector for the next loop level 2 scan step or s_2 and so on (see also Fig. 5a). These vectors can directly been derived from the scan pattern description generated by the compiler environment. The vectors could also be derived from the loop representation for a given f_k: $s_j = f_k(i_1,...,i_j+1,...,i_n) - f_k(i_1,...,i_j+1,...,i_n)$.

We will now introduce an application class, which allows a quite intuitive optimization strategy. Consider a loop nest L, which satisfies the following conditions:

1. $\forall\, v_k, v_l \in \mathrm{CV}(L) \cup \mathrm{GV}(L)$: $v_k = v_l \rightarrow f_k = f_l$;
2. $\forall\, v_k \in \mathrm{CV}(L)$: $f_k(i_1,...,i_n) = (c_0+c_1i_1+...+c_ni_n,\ d_0+d_1i_1+...+d_ni_n)$; $c_0,...,c_n \in$ Z; $d_0,...,d_n \in$ Z;
3. $\forall\, v_k \in \mathrm{GV}(L)$: $f_k(i_1,...,i_n) = (c_0+c_1i_1+...+c_ni_n,\ d_0+d_1i_1+...+d_ni_n)$; $c_0,...,c_n \in \{-1, 0, 1\}$; $d_0,...,d_n \in \{-1, 0, 1\}$;
4. $\forall\, v_k \in \mathrm{CV}(L) \cap \mathrm{GV}(L)$: $f_k(i_1,...,i_n) = (c_0+c_1i_1+...+c_ni_n,\ d_0+d_1i_1+...+d_ni_n)$; $c_0,c_2,...,c_n \in \{-1, 0, 1\}$; $d_0,d_2,...,d_n \in \{-1, 0, 1\}$; $c_1 = 0$; $d_1 = 0$;

The first condition demands, that there is only one access pattern for each variable. The next two conditions are due to the capabilities of the current compilation environment, which can handle only linear combinations of index variables for index functions. The third condition additionally requires from variables to be written, that the step width of their access sequence is at most one. The last condition means, that variables, which are written are not addressed using the innermost scan step s_1. Such variables can be buffered in the smart memory interface. This situation appears often in applications with multiply-accumulate operations.

For applications satisfying the above conditions, the access sequence of the data words is directly given by the scan pattern (i.e. the scan window has the size 1 by 1). The strategy to enhance the memory performance for such applications tries to exploit the burst mode of the memory for the innermost, time critical loop. This is done by rearranging the data map and the scan pattern in using a transformation matrix, so that the most relevant scan step points into the burst direction. After the rearrangement, the data is distributed onto parallel modules.

The transformation matrix is derived from the both most relevant scan steps. A scan step is considered not relevant, if it is the null vector or if it has the same direction as a scan step of a lower level. The algorithm for calculation of the transformation matrix A works as follows:

- Determine the most relevant scan step s_x:
  $s_x = s_j$, $j = \min(1,...,n)$ and $s_j \neq 0$
- Determine the second most relevant scan step s_y:
  $s_y = s_k$,$k = \min(j+1,...,n)$ and $s_k \neq 0$ and $\neg\, \exists\, l \in Z$: $(ls_k = s_x \vee ls_k = \begin{pmatrix}1\\0\end{pmatrix})$
  If there is no such s_k, then set:
  $$s_y = \begin{pmatrix}0\\1\end{pmatrix}$$
- Calculate the transformation matrix A by solving the following equations:
  $$A \times s_x = \begin{pmatrix}1\\0\end{pmatrix} \qquad A \times s_y = s_y$$

A data array v_k can then be transformed to v_k' using $v_k(x,y)' = A \times v_k(x,y)$, plus an offset vector, which is determined by the transformed array limits. Like the data, also the scan pattern is rearranged by calculating new scan steps $s_k' = A \times s_k$.

For the example in Fig. 5a, the transformation matrix and the result of the transformation are shown in Fig. 5b and Fig. 5c respectively. It can be seen, that A performs a clockwise turn by 90 degrees.

After the rearrangement of the arrays and the scan patterns, the parallel modules of the memory architecture are exploited. This is done by trying to align multiple arrays (see also Fig. 3). The alignment algorithm runs as follows:

- If CV(L) = ∅ or GV(L) = ∅ then distribute the arrays alternately onto adjacent rows of the memory.
- If there are arrays in both CV(L) and GV(L) then assign the variables in CV(L) to one module and the variables in GV(L) to the other one.
- In case there are more than one variable in one module, use vertical alignment of the arrays.
- Create a scan pattern and a scan window for each memory module and the aligned arrays.

To illustrate the algorithm, some examples are given in Fig. 6. The matrix-matrix

a) Matrix-Matrix multiplication

$$C = A \cdot B$$

$$c_{ij} = \sum_{s=0}^{n-1} a_{is} \cdot b_{sj}$$

b) Convolution

$$c = a \bullet b$$

$$c_i = \sum_{s=0}^{n-1} a_s \cdot b_{(i-s)}$$

c) Generic Application

$$c_{i,j} = a_{i,j} \cdot b_{i,j}$$

$$d_{i,j} = a_{i,j} + b_{i,j}$$

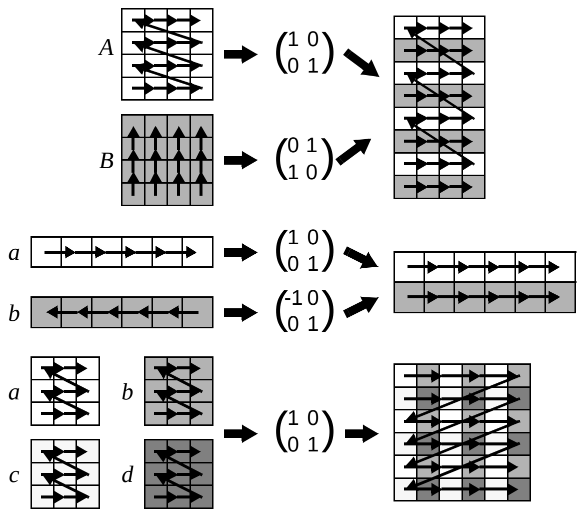

A
B
$\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$
$\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$
a
b
$\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$
$\begin{pmatrix}-1 & 0\\0 & 1\end{pmatrix}$
a
b
c
d
$\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$

Fig. 6. Examples illustrating the transformation (a),(b) and aligning (c) of data

multiplication and the convolution (Fig. 6a,b) can be optimized, because the write operation to the current data word c_i or c_{ij} respectively does not take place in the innermost loop. Thus, the current value of c_i or c_{ij} can be stored in the smart memory interface, without needing any memory access. The example in Fig. 6c illustrates the aligning of more than two arrays.

Under certain conditions, both the rearrangement and the aligning may result in a higher usage of memory. E.g. If s_1 is not parallel to either the x or the y coordinate, the transformation will shear the data array, resulting in a bigger area. In order to make the class of transformable applications as big as possible, we did not forbid such situations by adding more conditions. The trade-off between the memory usage and the speed has to be considered by a post processor, which is not covered in this paper.

Furthermore, we did not consider any time for the actual computation to be performed on the data. Considering the generic example in Fig. 6c, it is obvious, that the results would have to be written in the same speed, as the input is read. In reality, this can hardly be reached. For the example, both read and write burst would probably have to be interrupted to allow the computation to take place. Also, one would expect a delay

Reiner W. Hartenstein, Michael Herz, Thomas Hoffmann, Ulrich Nageldinger: Exploiting Contemporary Techniques in Reconfigurable Accelerators;
Proceedings of 8th International Workshop on Field-Programmable Logic and Applications, FPL'98, Tallinn, Estonia, August 31- September 3, 1998.
Proceedings: Reiner W. Hartenstein, Adres Keevallik (Ed.), Lecture Notes in Computer Science 1482, Springer-Verlag, Germany, 1998.

between the start of the read burst and the writing back of the results. However, both a delay and an interrupt do not influence the usability of the resulting storage scheme.

6 Memory Performance

Some estimated access times for our approach are given in Table 1. The times are given in microseconds, assuming a cycle time of 15 ns, and calculated for two of the sample applications in Fig. 6 with different problem sizes. The applications are the matrix-matrix multiplication (MAT, Fig. 6a) and the convolution (CON, Fig. 6b). The problem size is specified by the dimensions of one input array, i.e. not the resulting aligned array. We consider three different types of devices: Fast Page Mode DRAM (FPM), Burst Enhanced Data Out DRAM (BEDO) [17], and MDRAM (MDR) [11]. The first situation in the left major column represents the worst case where all data words are accessed without exploiting any burst or fast page modes and without parallel access. The other two columns assume the proposed parallel memory architecture, built from the different device types. The right column contains access times with the rearrangement strategy applied. The middle column contains theoretical values, assuming that the existing data ordering is exploited for burst mode and access to parallel banks is possible. These figures are only for examining the efficiency of the algorithm in optimizing the burst mode, as in reality, parallel access would not be possible without a proper data arrangement.

Table 1. Performance of the presented memory architecture

Application	Size of one array	Single word access			Without transformation			With transformation		
		FPM	BEDO	MDR	FPM	BEDO	MDR	FPM	BEDO	MDR
MAT	4 x 4	10.8	10.8	12.72	5.28	4.56	5.16	4.56	3.12	3.36
	40 x 40	9720	9720	11640	3984	3480	3600	3048	2040	1320
CON	20 x 1	18	18	19.8	7.05	9.75	11.4	7.05	5.31	4.65
	200 x 1	1530	1530	1833	475.5	772.5	924	475.5	322.5	197.7

For the calculation, we assume, that the FPM DRAMs can read or write data with 5 cycles for the first word and 3 cycles for any other word in the same page. Words in the same row of the two dimensional data memory are supposed to be also on the same page. For the BEDO DRAMs, we assume, that they work exclusively in burst mode with a maximum burst length of 4. The first word is accessed in 5 cycles, the next three in 1 cycle each, for reading and writing. The MDRAMs also use burst mode exclusively. The maximum burst length is 32 words, a read burst needs 5+*n* cycles, a write burst 4+*n* cycles, where *n* is the burst length.

Though the figures in Table 1 describe the theoretical peak performance, they show the efficiency of the proposed architecture and the transformation strategy. Comparing different devices, the MDRAMs have a poor performance for single word access, as MDRAMs use exclusively bursts, which involves a rather long initialization time for read accesses. This disadvantage can also reduce the performance for small data sizes, as the application MAT with 4 x 4 arrays shows. Looking at the second column, it can be seen, that BEDO devices with shorter bursts are superior, if the data is not properly arranged. However, the combination with the proposed algorithm provides dramatic speedups for large array sizes, which justifies the use of MDRAMs.

7 Conclusions and Future Work

We have shown a memory architecture for the MoM-PDA, which uses high-speed DRAMs. The memory supports the two-dimensional memory model of the machine paradigm and offers high bandwidth due to the use of two parallel modules. We have presented a strategy to exploit the burst mode of the memory and the parallel modules for a class of applications. The algorithm makes use of the compilation framework for the MoM-PDA, which provides the explicit data access sequence. The next aim is to extend the group of optimizable algorithms. A promising approach tries to rearrange the data words inside the scan window. This kind of optimization is quite complex, as there are many constraints. With a combination of several techniques we are confident, that a high memory bandwidth can be achieved for many applications.

References

1. R.W. Hartenstein, J. Becker, R. Kress: Custom Computing Machines vs. Hardware/Software Co-Design: From a Globalized point of view; Proc. of the 6th Intl. Workshop of Field Programmable Logic and Applications FPL'96, pp. 65 - 76, Springer LNCS, 1996.
2. W.H. Mangione-Smith, et. al.: Seeking Solutions in Configurable Computing; IEEE Computer, Dec. 1997.
3. D.A. Patterson, J.L. Hennessy: Computer Architecture, A Quantitative Approach; Morgan Kaufmann Publishers, 1990.
4. P. Budnik, D.J. Kuck: The Organization and Use of parallel Memories; IEEE Transactions on Computers, Dec. 1971.
5. K. Kim, V.K. Prasanna: Perfect Latin Squares and Parallel Array Access; Proc. Int. Symposium on Computer Architecture, ACM Press, 1989.
6. A. Deb: Multiskewing - A Novel Technique for Optimal Parallel Memory Access; IEEE Transactions on Parallel and Distributed Systems; Vol. 7, No. 6, June 1996.
7. M.E. Wolf, M.S. Lam: A Data Locality Optimizing Algorithm; Proc. of the SIGPLAN'91 Conf. on Programming Language Design and Implementation, pp. 30 - 44, Toronto, Canada, June 1991.
8. J. M. Anderson, S. P. Amarasinghe and M. S. Lam: Data and computation transformations for multiprocessors; Proceedings of the Fifth ACM SIGPLAN Symp. on Principles and Practice of Parallel Processing, July 1995.
9. R.W. Hartenstein, J. Becker, M. Herz, U. Nageldinger: A Novel Universal Sequencer Hardware; Proceedings of Fachtagung Architekturen von Rechensystemen ARCS'97, Rostock, Germany, September 8-11, 1997.
10. R.W. Hartenstein,A. Hirschbiel,K. Schmidt,M. Weber: A Novel Paradigm of Parallel Computation and its Use to Implement Simple High-Performance-HW; Future Generation Computer Systems 7 91/92, p. 181-198, North Holland.
11. N.N.: Siemens Multibank DRAM; Ultra-high performance for graphic applications; Siemens Semiconductor Group, Oct. 1996.
12. R.W. Hartenstein, J. Becker, R. Kress, H. Reinig: High-Performance Computing Using a Reconfigurable Accelerator; CPE Journal, Special Issue of Concurrency: Practice and Experience, John Wiley & Sons Ltd., 1996.
13. R. Kress: A Fast Reconfigurable ALU for Xputers; Ph.D. dissertation, University of Kaiserslautern, 1996.
14. J. Becker: A Partitioning Compiler for Computers with Xputer-based Accelerators; Ph.D. dissertation, University of Kaiserslautern, 1997.
15. K. Schmidt: A Program Partitioning, Restructuring, and Mapping Method for Xputers; Ph.D. dissertation, University of Kaiserslautern, 1994.
16. R. W. Hartenstein, R. Kress: A Datapath Synthesis System for the Reconfigurable Datapath Architecture; Asia and South Pacific Design Automation Conference, ASP-DAC'95, Nippon Convention Center, Makuhari, Chiba, Japan, Aug. 29 - Sept. 1, 1995.
17. N.N: The Burst EDO DRAM Advantage, Technical Note TN-04-41, Micron Technology Inc., 1995