

Automated Synthesis of Interleaved Memory Systems for Custom Computing Machines

Adam Postula, Song Chen
Department of Computer Science
and Electrical Engineering
University of Queensland,
St Lucia Campus,
Brisbane, Qld 4072, Australia.
adam@elec.uq.edu.au
chens@elec.uq.edu.au,

Lech Jozwiak
Faculty of Electrical Engineering
Eindhoven University of
Technology, 5600 MB Eindhoven,
The Netherlands
lech@eb.ele.tue.nl

David Abramson
School of Computer Science &
Software Engineering
Monash University
Clayton Campus
Melbourne, Vic 3168,
Australia
davida@csse.monash.edu.au

Abstract

This paper deals with methods of automated memory synthesis for Custom Computing Machines (CCMs). We focus the presentation on the synthesis of interleaved memory systems, which provide the quickest access to data at the cost of more parallel hardware.

The synthesis of an interleaved memory system starts with the analysis of data access patterns in the algorithm, from which a number of possible storage schemes is derived. The storage scheme defines how the array elements can be distributed among different memory banks. The best storage scheme is then chosen according to a complex metric of cost and performance of the required memory system. This metric involves the schedule length of operations, the type and number of required memory chips, the cost and delay of address generation and the interconnection network.

Results of our experiments demonstrate that the developed methods are capable to automatically synthesise storage schemes and interleaved memory systems, which are the same or better than those previously conceived by an experienced hardware designer.

1. Introduction

Custom Computing Machines form an important and very fast growing field in computer architecture [1]. They demonstrate exceptional performance and impressive speedups over conventional computers in solving some very hard computational problems [2][3].

The principle of Custom Computing Machines (CCMs) is that the architecture that matches the algorithm to be executed provides the best performance. With other words a CCM implements an algorithm in hardware and therefore it can be designed to utilise as much parallelism of the algorithm as is required to satisfy the performance goals.

Memory bandwidth is the crucial factor in high performance computing, therefore it is natural to provide a CCM with the parallel storage matching the data access patterns in the algorithm. To be effective, the automatic synthesis of application-specific memory system must tackle both the storage scheme and memory organisation [4]. Our research aims to find the optimal storage schemes and the corresponding interleaved memory structures when mapping array variables from behaviour descriptions onto the physical memory of a CCM.

Most of the reported research [5]-[10] on such storage schemes and interleaved memory originates in research in the area of parallel computing. The design of the storage scheme to optimise address calculations is the most important aspect of this work since it has the largest impact on the overall performance. In CCMs the address generation is implemented in hardware and is only one of the problems to be solved in the memory synthesis.

Much of the research on memory synthesis comes from the field of Digital Signal Processing. The methods developed by the IMEC group in Belgium [11],[12] allow data flow analysis of complex signals. They focus on the parallelisation of memories and optimisation of global communication. That work forms an interesting theoretical foundation for data dependency analysis and also delivers practical results. However, the methods are suitable mainly for DSP problems with rather regular data and the choice of memory configurations is limited. The selection of different types of memory and pipelines for DSP is described by Bakshi and Gajski in [13], but the storage scheme synthesis is not considered. The work reported by Aloqeely and Chen in [14] exploits regularity in the DSP data to replace the random access memory with stacks and queues implemented in hardware. Research on non-DSP memory optimisations is reported by Hemani's group in [15] where an algorithm is given for the selection of variables to be placed in the local or main memories. The

problems of mapping array variables in an algorithm to different memories are discussed in [16],[17],[18]. That research is focused on array grouping and does not address array splitting which is the case of synthesis of the storage scheme for interleaved memories.

In this paper we present the memory synthesis methods that are especially suited for high performance Custom Computing Machines. Our work is based on Periodic Storage Schemes (PSS) [10], which can handle access patterns with data-dependent or randomly generated indices. This suits CCMs, which exhibit complicated data access patterns, varying in different algorithms.

2. Combined synthesis of the storage scheme and interleaved memory

The storage synthesis in our system is carried out before High Level Synthesis, but after the basic functional unit allocation. The number of interleaved memory banks is in the range from 1 to 16, which covers the practical requirements.

The inputs to the synthesis algorithm are :

- The data flow graph $DFG(V,E)$, where: E - directed edges representing data dependencies, V - vertices of operation type including memory access (read, write) operators. All memory access operators are assumed to contain address generation.
- A library of memory components characterised by 7-tuple $\langle WD, BW, Prw, TRC, TWC, C \rangle$, where the symbols refer to the word depth, bit-width, the number of read and write ports, read access time, write access time, and cost of a memory component respectively.
- A library of functional units characterised by $\langle D, N \rangle$, where D - the delay, N - maximum number of one functional unit available.
- The constraints including the expected clock range of the system and the weights reflecting the designer's priorities for performance and cost trade-offs.

The outputs of the algorithm, represented by a group of points in schedule length-cost space, are:

- The number of memory banks in the memory system.
- The memory component type chosen from the given library.
- Optimised storage scheme.
- Optimal clock range.

Given the inputs, the synthesis algorithm produces the outputs by exhaustive search of the limited solution space. This search is effective due to removal of redundant PSSs and a relatively small number of memory banks.

3. Memory system model

3.1 Memory organisation

The memory model used in our system is shown in Figure 1. It differs from the traditional interleaved memory system in that the address generator (AG) is application specific and consists of a lookup table and adders/subtractors. The data alignment is performed implicitly in the AG lookup table. This reduces the delay of address generation since there is no need for interconnection network between address generators and memories. There are M memory banks, each bank may consist of multiple memory components depending on the sizes of arrays to be stored. Without loss of generality, we assume that the arrays are two-dimensional, and the whole memory system is built of one type of memory chips. We also assume use of static RAMs with bidirectional data ports and standard control pins. All other elements are implemented in Field Programmable Gate Arrays (FPGA) XC4000 technology from Xilinx.

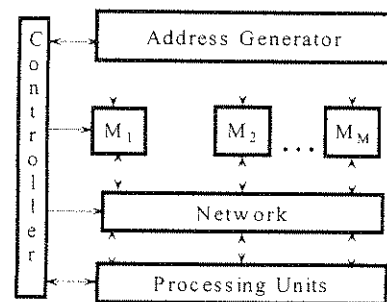


Figure 1. General structure of the interleaved memory system

3.2. Address Generator

The Address Generator (AG) is built of a wide word lookup table connected to M adders/subtractors. Each word of the table consists of M sub-words with each sub-word generating the offset for a separate memory bank. The input to the AG is the base address formed from the word and bank addresses of one element in the access pattern. The base address can be generated in advance in the pipelined design or just stored in a memory as in [19]. All other addresses in the pattern are generated by adding or subtracting the offsets from the base address. The lookup tables in the AG store all the possible memory access sequences and their corresponding offsets. Since the number of those sequences is limited, the lookup tables are small. One memory access sequence means one read or write to all the M banks for some specific

elements in the access pattern. Data alignment is done implicitly in the lookup tables.

A detailed discussion of the address generation in our synthesis system can be found in [20].

3.3. Memory cost and access delay model

Effective read time of memory comprises the address generation delay, the memory read access time (TRC) and the network delay. Effective write time is the address generation delay plus the memory write access time. This is based on the assumption that the delay of the address generator is larger than the combined delays of the network and the controller.

The effective delay of the address generation is the sum of the address generator delay, the chip-selector delay, and the routing and propagation delays. The address generation delays are the same for both read and write cycles.

Memory cost is composed of the number of memory chips c_1 and number of wires from the memory to the data path c_2 :

$$c_1 = M \cdot \lceil S / (WD \cdot M) \rceil$$

$$c_2 = M \cdot (BW + \lceil \log_2(WD) \rceil + 2 + \lceil S / (WD \cdot M) \rceil)$$

Where S is the array size, M the number of memory banks, WD the depth and BW the data width of memory chips. The c_2 comprises the contributions of the number of data lines, address lines, chip selector lines, output enable and write enable lines.

Address generator cost and delay

The cost of the address generator is measured in the number of CLB (basic logic blocks in the Xilinx FPGAs) that implement a basic lookup table of 32 x 1 or 16 x 2 bit words. If WD of the required lookup table is larger than 32, some multiplexers (MUX) are needed. The total cost can be easily computed for a required lookup table.

The cost and delay of the second component of the address generator – the N bit adder/subtractor [23] are calculated as:

$$Delay = 9 + 0.75 \cdot N$$

$$Cost = \lceil N / 2 \rceil + 1$$

The routing delay of the address generator is assumed to be equal to the adder/subtractor delay.

Network and chip selector cost and delay

The network connecting the data path and the interleaved memories are built of multiplexers implemented in FPGA. Their cost and delay are estimated in a straightforward way on the basis of the CLB cost and delay, so are the chip selector delay and cost.

4. Periodic storage scheme

For a particular algorithm the data access patterns are known and it is possible to find a storage scheme optimised for those patterns. This is referred to as storage scheme synthesis. The synthesis strategies utilising the high regularity of Periodic Storage Schemes (PSS) and the lookup table address generation lead to relatively simple and very effective interleaved memory systems.

A formal introduction to the synthesis of periodic storage schemes can be found in [21]. In this section we want to introduce the basic concepts less formally but with sufficient precision to explain the generation of PSSs.

We denote Z as integer set, Z^2 as two-dimensional Euclidean space, M as the number of memory banks. The matrix formed by two vectors $\mathbf{a} = (a_1, a_2)$ and $\mathbf{b} = (b_1, b_2)$ in Z^2 is denoted as:

$$[\mathbf{a}, \mathbf{b}]^T = \begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix}$$

Let $\mathbf{x}_1, \mathbf{x}_2 \in Z^2$ and $\det(\mathbf{x}_1, \mathbf{x}_2) \neq 0$. Then the set of points $\mathbf{x} = \lambda_1 \mathbf{x}_1 + \lambda_2 \mathbf{x}_2$ ($\lambda_1, \lambda_2 \in Z$) is called a **lattice** L^2 , and $\mathbf{x}_1, \mathbf{x}_2$ are the basis of the lattice. The fundamental parallelopete of L^2 is the set $\{v_1 \mathbf{x}_1 + v_2 \mathbf{x}_2 \mid v_i \in [0, 1), i = 1, 2\}$. Its volume is denoted as $\Delta(L^2)$ and called the determinant of L^2 . The matrix $[\mathbf{x}_1, \mathbf{x}_2]^T$ is called the basis matrix of L^2 . The lattice mesh is formed by the lines parallel to the two basis vectors with its shape the same as fundamental parallelopete. Fundamental parallelopete is one of the meshes.

A **storage skewing scheme** for an array $A(N_1, N_2)$ is a mapping $s(i, j)$ from $Z_{N_1} \times Z_{N_2} \rightarrow Z_M$, which means each element in A is assigned a memory bank number from Z_M .

A skewing scheme s is **periodic** if and only if there exists a Lattice L^2 , such that

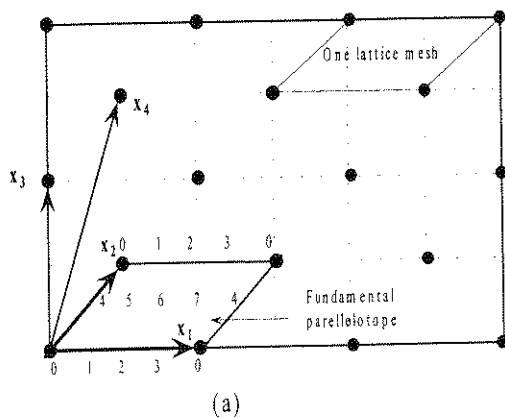
(a) $\Delta(L^2) = M$.

(b) All the overlapped grid points of the array elements with L^2 are mapped onto the same memory bank.

(c) The grid points of the array in one lattice mesh of L^2 or on two edges of the mesh must be mapped to different memory banks. The two mesh edges are those which have the same relative position and direction as that of two basis vectors of L^2 .

(d) If L^2 is shifted at any position in the grid, (b) remains true.

As an example, assume $M = 8$, the basis vectors are $\mathbf{x}_1$ and $\mathbf{x}_2$, the basis matrix is $[(4,0), (2,2)]^T$. The lattice is shown in Figure 2a, $\Delta(L^2) = 8$. The matrix in Figure 2b shows the PSS corresponding to the lattice in Figure 2a.



0	1	2	3	0	1	2	3	0	1	2	3
5	6	7	4	5	6	7	4	5	6	7	4
2	3	0	1	2	3	0	1	2	3	0	1
7	4	5	6	7	4	5	6	7	4	5	6
0	1	2	3	0	1	2	3	0	1	2	3
5	6	7	4	5	6	7	4	5	6	7	4
2	3	0	1	2	3	0	1	2	3	0	1
7	4	5	6	7	4	5	6	7	4	5	6
0	1	2	3	0	1	2	3	0	1	2	3

(b)

Figure 2. The concept of lattice and Periodic Storage Scheme (a), mapping of array elements to memory banks (b).

Due to one to one mapping from the lattice to a Periodic Storage Scheme the search space for PSS is all the different lattices represented by the basis matrices

Graphically, two lattices are different if they are not completely overlapped.

Finding a lattice is equivalent to finding the appropriate basis matrix. However, for $\Delta(L^2)=M$, the number of basis matrices is infinite. For example, the matrix $[(4,0), (a, 2)]^T$ ($a \in \mathbb{Z}$) has $\Delta(L^2)=8$ but the number of choices is infinite. In Figure 2a, $[x_1, x_2]^T$, $[x_2, x_3]^T$ and $[x_2, x_4]^T$ are all bases of the same lattice.

In order to build effective synthesis we had to find ways to prune the search space. We formulated and proved the theorems [20] that successfully reduced the search space to a small number of choices. Here we present the essence of that development.

Theorem If M is the number of memory banks and all the ordered pairs (α_i, β_i) such that $M = \alpha_i \beta_i$ ($\alpha_i, \beta_i \in \mathbb{Z}$, $i = 0, 1, \dots, k$) are found, then the number of different lattices equals $\alpha_1 + \alpha_2 + \dots + \alpha_k$.

By the above theorem, the infinite search space has been reduced to a limited one. Specifically, the total num-

ber of lattices to be considered for 2D arrays in interleaved memory systems with 1 to 16 banks is 220.

5. Automated synthesis

5.1. System cost

The design metrics is based on system performance and cost. The performance is modelled by the schedule length. The cost includes the number of memory chips required - c_1 , the wiring necessary to connect the memory system to data path - c_2 , the cost of the address generators - c_3 , the cost of interconnection network - c_4 and the cost of chip selectors - c_5 . The total cost is

$$C = \sum_{i=1}^5 w_i c_i$$

Where w_i represents the weights of the corresponding cost c_i . The weight w_i reflects the typical relative market price of a single chip. The values of other weights are chosen by the designer according to the design priorities. These can include technology specific weighting of different costs.

5.2. System performance

List scheduling with chaining of operations and multiple cycling is used to find the system performance. The mobility is used as the priority function calculated from ASAP and ALAP scheduling. Address generation is bound to memory access operations in the DFG. This is to reflect the fact that the memory accesses will be performed immediately once the address generation is finished. In scheduling, the memory write can not be chained with other operators because the write is usually finished at a clock transition.

Since only memory access-related operations need to be considered, the original DFG is first divided into several sub-graphs by profiling. Then each sub-graph is scheduled separately. The total system performance is the weighted sum, or the average, or the largest of all the schedule lengths, depending on the relation of these sub-graphs. For example, if there are two cascaded loops in DFG, the sub-graphs can be the two loop bodies without loop iterators. The performance is the weighted sum of the schedule lengths of the two sub-graphs and the weights are the scaled loop iterators. For conditional paths in the DFG, the probabilities of executing each path are obtained by profiling and used as the weights.

5.3. The synthesis algorithm.

Algorithm: Design space exploration

Input: DFG, memory and functional unit library, design constraints

for all chosen clocks

for all memory types in the library

for memory banks $M = 1$ to 16

for all the storage schemes {

find the schedule lengths of all sub-graphs,
and calculate system performance.

find system cost according to the cost model

draw all the performance- cost curves

and find the Pareto points.

}}}}

The search space of the algorithm comprises the number of memory banks, the type of memory chips from a library, the storage scheme and the clock period of the system. Pruning of the search space presented in Section 4 allows us to use the exhaustive search and find the optimal solutions in relatively short time.

The design objectives are performance and cost as defined by the cost weights given by the designer. Pareto points [22] are used to represent the optimised results, which is a set of non-dominated points in the design space.

6. Experimental Results

Our algorithms are implemented in Visual C++ on a Pentium 200 PC with 9000 lines of code. The design of sintering simulation machine [19], whose memory system was previously designed manually serves as an experimental design case.

The data structure in the sintering machine is a large 2D array A[1304,1304]. Each element in the array represents an atom or a hole and is three bits. In processing, as shown in Figure 3, after choosing an element (a) randomly in the array grid, one of its six immediate neighbours, say element (b), will also be read. Depending on some conditions, the process will either go to next iteration or read the rest of immediate neighbours and the three potential neighbours (c), (d) and (e). Then after some calculation and judgement, (a) and (b) may be swapped and written back to memory. For every element (a) there are six possible directions, so there are six access patterns. Figure 3 shows just one of them.

In order to simplify the presentation of the results we assume that the delays of all functional units are 30ns, and the clock ranges from 30ns to 70ns. The weight of chip counts is chosen as the cost index in table 1, other weights are all chosen as 1.

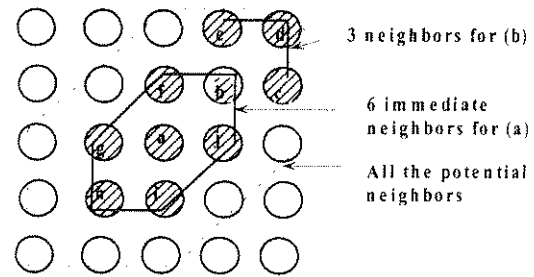


Figure 3. The access pattern when element (b) is chosen as direction

The search is performed in the four-dimensional space and results in finding the performance (weighted schedule length) and cost. The search function can be expressed as $(l, c) = f(ck, T, M, P)$, where the left side denotes the weighted schedule length and system cost, the right side the clock, the chosen memory type, the number of memory banks, and the specific storage scheme respectively. Three kinds of problems can be solved:

Table 1 Memory library in the example (BW for all memories are 4 bits)

Type	A	B	C	D
WD(k)	128	128	128	256
Access Time (ns)	10	40	80	10
Cost index	40	15	10	80

Type	E	F	G	H	I
WD(k)	256	256	512	512	512
Access Time (ns)	40	80	10	40	80
Cost index	30	20	130	50	35

Problem A.

For a specific clock ck_0 and memory type T_0 , find all the points $(l, c) = f(ck=ck_0, T=T_0, M, P)$. The results are in Figure 4 and in Table 2.

Figure 4 shows one of these results, with $ck = 50ns$, memory type = B. For the sake of clarity only some solutions are printed. Pareto points are from $M = 1, 3, 4, 5, 6, 12$.

The basic trend is that performance is more than doubled when M increases but cost also increases. The performance increase is not linear since the influence of memory operators on performance is not linear and the delay of AG and functional units become critical when $M > 6$. Memory type B from table 1 is relatively cheap and constitutes about 50% of system cost used as the off-chip main memory when $M \leq 4$. However, when $M > 4$, the costs of address generation and interconnection become comparable to the cost of the main memory. The system cost varies little with different storage scheme for the

same number of memory banks, but the differences on performance are dramatic. Different PSSs have small influence on system cost, since storage schemes affect word addresses, resulting in different offsets and lookup table in AG. Table 2 shows more detailed results for chosen solutions. The item in the PSS field is the two basis vectors for the storage scheme. All the PSSs are shown for $M = 4$, for other number of memory banks only the best points are listed. Two solution points found by the algorithm are of special interest. The first is (50, B, 3, P3), which is the same as the manual design in [19]. The second is (50, B, 4, P46), which offers a better performance than the manual design.

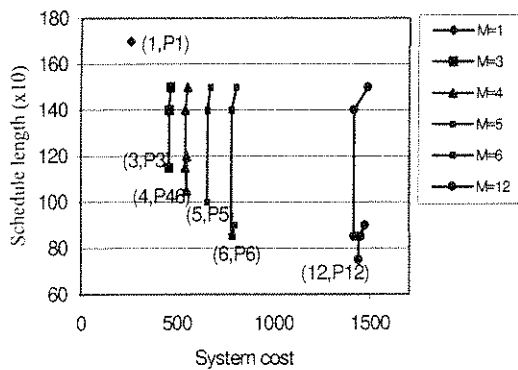


Figure 4 The search points for function $(l, c) = f(ck = 50, \text{type} = B, M, P)$. Pareto points are marked with (M, P) .

Table 2 Design metrics with $ck = 50\text{ns}$, memory type B

M	1	2	3	5	6	12	
Length	17	18	11.5	10	8.5	7.5	
System cost	257	355	458	658	783	1440	
PSS	P1 (0,1) (1,0)	P2 (2,0) (0,1)	P3 (3,0) (2,1)	P5 (5,0) (2,1)	P6 (6,0) (2,1)	P12 (6,0) (4,2)	
M	4						
Length	15	14	14	11.5	11.5	10.5	12
System cost	556	544				552	552
PSS	P41 (1,0) (0,4)	P42 (4,0) (0,1)	P43 (4,0) (1,1)	P44 (4,0) (2,1)	P45 (4,0) (3,1)	P46 (2,0) (0,2)	P47 (2,0) (1,2)

Table 3 The formation of Pareto points for some memory types (the index is in the order of cost increase, the matrix of PSSs are listed in Table 2)

Pareto points	(M, P) changes with memory types				
	A	B	C	E	H
1	(1, P1)				
2	(2, P2)	(3, P3)	(2, P2)	(3, P3)	(4, P46)
3	(3, P3)	(4, P46)	(3, P3)	(4, P4)	(6, P6)
4	(4, P46)	(5, P5)	(4, P46)	(5, P5)	(12, P12)
5	(6, P6)	(6, P6)	(12, P12)	(6, P6)	\
6	(12, P12)	(12, P12)	\	(12, P12)	\

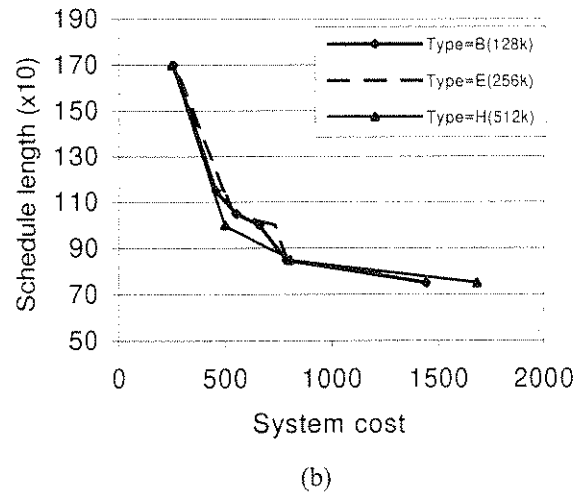
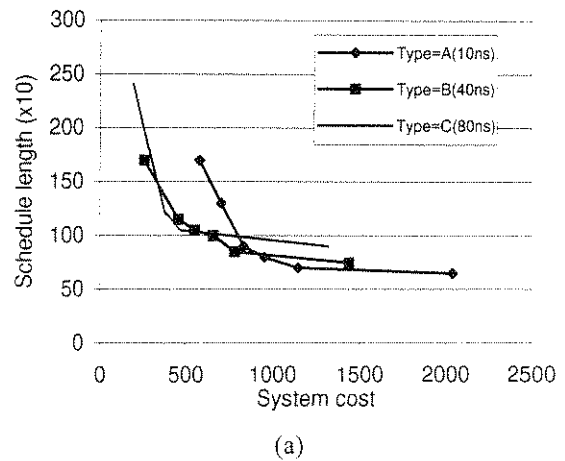


Figure 5 Pareto point curves for different memory types, $(l, c) = f(ck = 50, T, M, P)$.

Problem B.

For each type of memory find all the Pareto points for a fixed clock, i.e. $(l, c) = f(ck=ck_0, T, M, P)$. The results are in Table 3 and Figure 5.

From Figure 5a we can see that type A (10ns) has a little higher performance but much higher cost than that of B (40ns) and C (80ns). The slow memory C has the largest performance improvement with the increasing M. In Figure 5b the performance is similar for memories with the same access times but different memory capacities. System cost using type H (521k) is slightly lower since the memory price does not change linearly with capacity.

Problem C

For all the curves $(l, c) = f(ck_0, T, M, P)$ as drawn in Figure 6 we can find their Pareto point curve corresponding to optimal solution for ck_0 . Each ck corresponds to one optimal curve. In Figure 6 three curves correspond to three clocks. When ck changes from 70ns to 30ns, the improvement of performance is about 20%. Table 4 lists the optimal points, related to the optimal curve in Figure 6.

From Table 4 we can see M=3 and M=4 are attractive trade-off points. M=3 is the manual design, and M=4 with P46 has better performance and a little higher cost. M=4 is potentially better because M is a power of two and it is easier to generate word addresses. The bank-mapping matrix for this scheme is presented in Figure 7.

Table 4 Global optimal points in form of index number : (T,M,P), where $ck = 30ns$, index number is in the order of cost increase.

1 : (1, 1, P1)	2 : (H, 1, P1)	3 : (1, 2, P2)
4 : (C, 3, P3)	5 : (G, 4, P46)	6 : (B, 3, P3)
7 : (H, 4, P46)	8 : (B, 6, P6)	9 : (G, 4, P46)
10 : (D, 7, P7)	11 : (D, 11, P11)	12 : (D, 12, P12)

7. Conclusions

We have presented efficient synthesis of interleaved memory systems for Custom Computing Machines. Our methods combine storage scheme synthesis with interleaved memory system synthesis and provide optimal solutions for the design objectives set by the designer. The effectiveness of the presented methods has been checked in an experiment that produced a series of solutions to the design problem previously solved by an experienced hardware designer. Our automatic synthesis not only found the solution that had been conceived by the designer but also produced another design with better performance and simpler address generation. We are in a

process of extensive testing of the developed methods on a benchmark of memory intensive designs. Our future work will cover synthesis of multi-dimensional arrays for interleaved CCM memory systems.

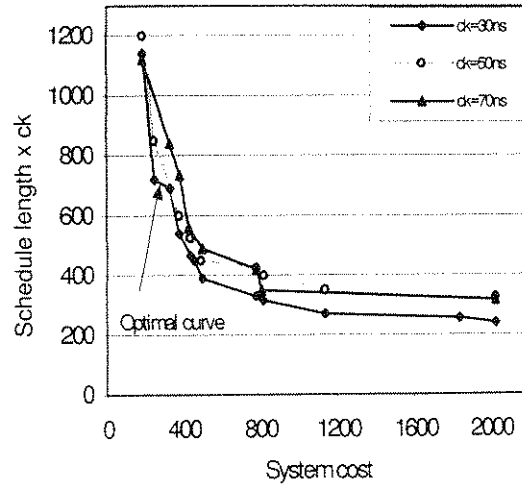


Figure 6 Pareto point curves with function $(l, c) = f(ck, T, M, P)$ for different clocks.

0	1	0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1	0	1

Figure 7 Bank mapping matrix for the solution (4, P46)

8. Acknowledgement

The authors would like to thank Paul Logothetis in the School of Computer Science and Software Engineering at Monash University for very helpful discussions and contribution of the design case. Logothetis is supported by a related ARC Large grant.

References

- [1] R.W. Hartenstein, Custom Computing Machines - an Overview, Proc DMM'95 - In'l Symp. On Design Methodologies in Microelectronics, Smolenice Castle, Slovakia, September 1995.
- [2] J. Arnold et al., Splash2 - FPGAs in a Custom Computing Machine, IEEE Computer Society Press, 1996.
- [3] J.E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H.H. Touati and P. Boucard, "Programmable Active Memo-

- ries: Reconfigurable Systems Come of Age, IEEE Trans on VLSI Systems, Vol.4, No.1, March 1996, pp 56-68.
- [4] A.Postula, D.Abramson, Z.Fang, P.Logothetis, A Comparison of High Level Synthesis and Register Transfer Level Design Techniques for Custom Computing Machines, HICSS'98, Hawaii, January 1998.
- [5] Deb-A., "Multiskewing-A Novel Technique For Optimal Parallel Memory Access," IEEE Trans. Parallel-and-Distributed-Systems. Vol.7, No.6, June 1996, pp.595-604.
- [6] Norton-A; Melton-E, "A class of Boolean linear transformations for conflict-free power-of-two stride access," Proceedings of the 1987 International Conference on Parallel Processing, 1987, pp.247-254.
- [7] De-Lei-Lee, "On access and alignment of data in a parallel processor," Information-Processing-Letters., Vol. 33, No.1; 27 Oct. 1989; pp.11-14.
- [8] Sohi-G.S., "High-bandwidth interleaved memories for vector processors-a simulation study," IEEE-Trans.-on-Computers. Vol.42, No.1; Jan. 1993; pp.34-44.
- [9] Harper, D.T., Linebarger, D.A., "Conflict-Free Vector Access Using a Dynamic Storage Scheme," IEEE Trans. on Computers, Vol. 40, No. 3, March 1991, pp. 276-283.
- [10] Wijshoff-HAG; van-Leeuwen-J, "The Structure Of Periodic Storage Schemes For Parallel Memories," IEEE-Trans.-on-Computers. Vol.C-34, No.6; June 1985; pp.501-505.
- [11] F.Balasa , F.Cathoor and H.De Man , Dataflow-driven memory allocation for multi-dimensional signal processing systems, Proceedings of the IEEE International Conference on CAD, 1994, pp 31-34.
- [12] F.Balasa , F.Cathoor and H.De Man, Background Memory Area Estimation for Multi-dimensional Signal Processing Systems, IEEE Trans. on CAD, , Vol 14 , 1995.
- [13] Bakshi-S; Gajski-DD S. , "A Memory Selection Algorithm for High-Performance Pipelines", Proceedings EURO-DAC '95, 1995, pp. 124-129.
- [14] M.Aloqeely, C.Y.R.Chen, "A new Technique for Exploiting Regularity in Data Path Synthesis", Proc. of EURO-DAC -94, 1994, pp394-399.
- [15] A. Jantsch, P. Ellervee, J. Oberg, A. Hemani, H. Tenhunen, "Hardware/Software Partitioning and Minimizing Memory Interface Traffic", Proc. of EURO-DAC - 94, pp226-231, 1994.
- [16] L.Ramachandran, D.D. Gajski and V. Chaiyakul, "An Algorithm for Array Variable Clustering", Proceedings of European Design and Test Conference (EDAC), 1994.
- [17] D.Karchmer, and J.Rose, "Definition and Solution of the Memory Packing Problem," Proc. of ICCAD, pp. 53-58, Nov. 1994.
- [18] Herman Schmit and Donald E. Thomas, "Synthesis of Applications-Specific Memory Designs," IEEE Trans. on VLSI Systems, Vol. 5, No. 1, pp. 101-111, March, 1997.
- [19] Postula-A; Abramson-D; Logothetis-P, "The Design Of A Specialised Processor For The Simulation Of Sintering," Proceedings of the 22nd EUROMICRO Conference. 1995, pp.501-508.
- [20] S. Chen, A. Postula, Interleaved Memory Optimization with Periodic Storage Scheme Synthesis, Internal Report, Dept. of Computer Science and Electrical Engineering, The University of Queensland, Australia, 1998.
- [21] S.Chen, A.Postula, A.Hemani, Interleaved Memory Optimization with Periodic Storage Scheme Synthesis, APCHDL'98, Korea, 1998.
- [22] G. De Micheli, Synthesis And Optimization Of Digital Circuits, McGraw-Hill, Inc., 1994.
- [23] The Programmable Logic Data Book, Xilinx, 1994, <http://www.xilinx.com>.