System-Level Data Format Exploration for Dynamically Allocated Data Structures

Peeter Ellervee[1], Miguel Miranda[2], Francky Catthoor[2,3], Ahmed Hemani[1]
[1]ESD, KTH, Electrum 229, S-16440 Kista, Sweden
[2]IMEC, Kapeldreef 7, B-3001 Leuven, Belgium
[3]Professor at the Katholieke Universiteit Leuven
lrv@ele.kth.se, miranda@imec.be, catthoor@imec.be, ahmed@ele.kth.se

ABSTRACT

Memory bandwidth and power consumption are important design bottlenecks for data dominated applications. We propose a systematic system level exploration approach and formalised techniques to alleviate these bottlenecks based on rearranging the format of the data records that are later stored in memory. The technique exploits parallelism in the data transfer and reduction in bit waste. Using our approach on several real-life ATM processing applications, significant reduction in size, bandwidth and hence power consumption are obtained.

1. INTRODUCTION

Usage of higher abstraction levels allows to increase both productivity of designers and therefore the complexity of systems to be specified. At the same time, performance requirements motivate implementation of such abstract specifications in time-critical embedded software or even in hardware. This twin motivation has been behind the advances in VLSI CAD, which has primarily focussed on raising the abstraction of computational aspects and automating their implementation in hardware. Recently however, the VLSI CAD community has put more emphasis on data storage and communication aspects of specifications since applications are becoming more and more data-dominated (see summary in [1]).

Data-transfer intensive (DTI) applications, which require large storage, such as in protocol processing and multimedia, have been mostly implemented in software. The complexity of protocol processing applications and its history of being implemented in software and demands on productivity, motivate the use of dynamically allocated data structures (DADS). Matisse [8] is a systematic design methodology and supporting prototype tool environment, under development at IMEC, that addresses implementation issues of such structures and in general of telecom network components. It addresses both hardware and software targets.

DADS are defined on the basis of functional and logical coherence for readability. Retaining such coherence while organising the DADS physically in the memory does not optimise the required storage neither the required bandwidth. The latter optimisation goals are better addressed by analysis of the dependencies between memory accesses with the aim of exploring packing alternatives of DADS's elements into single memory words. Heuristic rules have been developed to speed up the exploration phase.

Incorporated into IMEC's Physical Memory Management project (PMM) [11; 9], the data format exploration step enables opportunities for significant reduction of memory bit waste during the later physical memory mapping phase. As as result, it allows to reduce the number of accessed bits and hence the required memory bandwidth and power consumption. Additionally, it also helps in further reducing the addressing and interconnect overhead and it simplifies succeeding synthesis/optimisation steps by properly pruning the search space available at the higher level. However, a difficult trade-off is involved between accesses to the packed words saved due to the simultaneous read/write operations and those lost due to the unpacking needed for non-simultaneous accesses (see Subsection 3.1). Therefore, a formalised exploration methodology supported by automated techniques are crucial to properly explore the search space before actual memory mapping stage.

2. RELATED WORK

Almost all published techniques for dealing with the allocation of storage units are scalar-oriented and employ a *scheduling-directed* view (see e.g. [5]) where the control steps of production/consumption for each individual signal are determined beforehand. This applies also for memory/register estimation techniques (see e.g. [5; 3] and their references). This strategy is mainly due to the fact that applications targeted in conventional hardware synthesis contain a relatively small number of signals. Therefore, as the major goal is typically the minimisation of the number of registers for storing scalars, the scheduling-driven strategy is well-fitted to solve a *register allocation* problem, rather than a *background memory allocation* problem.

The above techniques present serious shortcomings for most DTI applications for several reasons. First, scheduling *must* precede (foreground) memory management in the conventional high-level synthesis systems. However, by switching the order, the dominant background memory cost can be reduced further [12] and the freedom for data-path allocation and scheduling remains almost unaffected. Furthermore, within the scalar view, many examples are intractable

DAC 2000, Los Angeles, California

because of the huge size of the ILP formulations.

Exceptions to these techniques have been initiated by early work at IMEC and Philips. Phideo [10] at Philips is oriented to periodic stream based processing of video signals. At IMEC, since 1988 extensive research has been performed in the direction of this array-oriented custom memory organisation management. For the DTI applications, the most relevant work is the one on automated memory allocation and assignment *before* the scheduling or the procedural ordering are fully fixed [9]. A complete methodology for custom background memory management has been proposed in the ATOMIUM script [1].

Recently several other approaches for specific tasks in memory management oriented to non-scalar multi-dimensional signal processing have been published. This includes the MeSA [6] approach at U.C. Irvine focussing on static memory allocation, and at CMU where strategies to mapping arrays in an algorithm on horizontally and vertically partitioned dedicated memory organisations have been proposed too [7]. The latter approach uses horizontal concatenation of arrays, similar to our pre-packing of elements of DADS, but does it at the same time with scheduling thus significantly increasing the complexity of the optimisation task. As a result, the exploration scope of the technique is heavily restricted. Moreover, the focus of our approach includes also dynamically allocated data types with sets of records.

An analysis of how merging data-fields of dynamically allocated data structures affects the number of storage operations is presented in [2]. In this paper a systematic, significantly extended optimisation technique is presented to speed up the exploration of packing alternatives.

3. PRE-PACKING METHODOLOGY

Matisse is a design flow intended for system exploration and synthesis of embedded systems characterised by dynamic data storage and intensive data transfer [8]. To deal with realistic applications, the memory management task of Matisse assigns groups of scalars to memories instead of individual scalars. We call these non-overlapping groups of scalars *basic-groups* (BG). This is done in such a way that for every memory access, it is known at compile time which basic-group is being accessed. The proposed pre-packing of fields of dynamic data structures works also with the basic-groups and is executed before the storage bandwidth optimisation.

3.1 Problem definition and model

The way how the different fields of a data structure are formated onto physical words significantly affects the size of the memories and the number of accesses needed to fetch/store the data. The number of accesses and memory bit-width directly affects the overall performance and power consumption. This is crucial for both custom hardware and embedded software targets.

Let us consider for illustration purposes the example shown in Figure 1.a where the structure `my_type` consists of two fields `field_0` and `field_1` with width 3 and 11 bits respectively. Only two basic-groups can be identified in this example - the arrays corresponding to `field_0` and `field_1`. Software compilers would typically map the data structure array in memory such a way that an instance of the data structure is accessed by a pointer and its different fields are accessed by successive offsets (Figure 1.b). Similar solutions in embedded software or hardware would be undesirable, mostly because of the large bit waste due to bit-width mismatches and the impossibility of performing simultaneous accesses to both fields. Assigning each data-field to different basic-groups and allowing customised memory organisations (see Figure 1.c) would reduce the bit waste and alleviate the speed requirements needed to meet the bandwidth constraints by exploiting parallelism in the data transfer. However, it would also increase the interconnect overhead, the address calculation cost, and the energy required for address decoding. An alternative similar to Case 1 could be to decompose the data structure fields into two independent basic groups and map each of them onto the same memory (Figure 1.d). Although, for this case the drawbacks of Case 2 remain, by combining cases 2 and 3 it would be possible to make trade-offs between bandwidth and bit waste during the basic group to memory assignment phase. However, more interesting trade-offs can be enabled when assigning two or more fields of the same structure to the same basic-group (Figure 1.e). The advantages are significant reduction in the number of memory accesses and a more uniform distribution of data bit width, hence resulting in less bit waste during the subsequent basic-group to memory assignment stage. The downside is that when the data field candidates are not carefully selected, updating a single field requires two accesses to avoid data corruption.

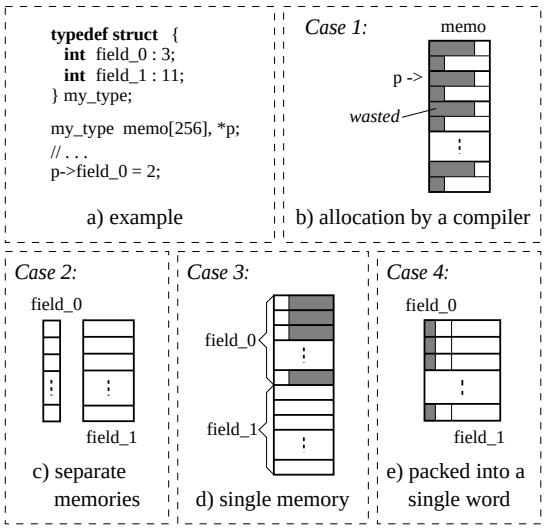


Figure 1: Different data format possibilities

An intuitive approach, based on the experiences of designers, is rather simple in its nature - two similar accesses (read or write) can be merged when: a) the same base address is used; and b) the candidate fields are accessed in a basic block, indicating data transfer locality.

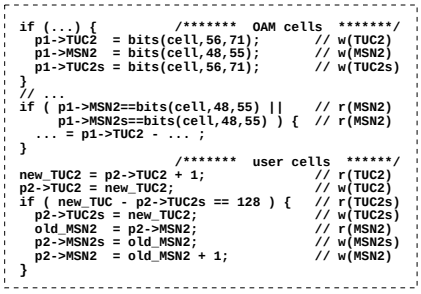
A brief description about analysis of dependency cases is given below. A systematic analysis of dependencies and a detailed description of how the cases are collected, are presented in [2].

A simplified CDFG is used for dependency analysis since only information relevant for the data-transfer is needed. The only nodes of interest of the pruned graph are read and write operations. All dependencies between any pair of memory accesses (read or write) can be classified as one of the two main types - *sequential* or *parallel* dependency. The types of dependencies are *control*, *data* and *address* dependencies. The following well-motivated assumptions are used

in the analysis:

- accesses to the same basic-group are ignored;
- a single execution thread is assumed, i.e. all accesses can be statically ordered on a relative time axis;
- only pairs of accesses to the same data structure and depending on the same pointer (base address) are analyzed;
- only static dependencies are analysed to avoid the need for profiling data (which may be misleading); and
- only access paths with length one are analysed since the long dependency chains are covered by shorter ones.

Only two cases exists which are used to build compatibility graph. The first one, *read-read*, covers relevant sequential and parallel read dependencies; and the second one, *write-write*, covers relevant parallel write dependencies.



```
if (...) {          /*******  OAM cells  *******/
  p1->TUC2  = bits(cell,56,71);       // w(TUC2)
  p1->MSN2  = bits(cell,48,55);       // w(MSN2)
  p1->TUC2s = bits(cell,56,71);       // w(TUC2s)
}
// ...
if ( p1->MSN2==bits(cell,48,55) ||    // r(MSN2)
     p1->MSN2s==bits(cell,48,55) ) {  // r(MSN2)
  ... = p1->TUC2 - ... ;
}
                    /*******  user cells  ******/
new_TUC2 = p2->TUC2 + 1;              // r(TUC2)
p2->TUC2 = new_TUC2;                  // w(TUC2)
if ( new_TUC - p2->TUC2s == 128 ) {   // r(TUC2s)
  p2->TUC2s = new_TUC2;               // w(TUC2s)
  old_MSN2  = p2->MSN2;               // r(MSN2)
  p2->MSN2s = old_MSN2;               // w(MSN2s)
  p2->MSN2  = old_MSN2 + 1;           // w(MSN2)
}
```

Dependency cases

Case	Fields	No.
read-read	TUC2-TUC2s	1
read-read	TUC2-MSN2	2
read-read	TUC2-MSN2s	1
read-read	TUC2s-MSN2	1
read-read	MSN2-MSN2s	1
write-write	TUC2-TUC2s	2
write-write	TUC2-MSN2	1
write-write	TUC2s-MSN2	1
write-write	MSN2-MSN2s	1

Figure 2: Example of a compatibility graph

3.2 Compatibility graph construction

To illustrate how the dependency cases are collected and how the corresponding compatibility graph is built, an example consisting of a simplified CDFG is shown in Figure 2. The CDFG is a part of the Operation and Maintenance (OAM) handler of an ATM switch [4]. The exploration results of the full design are described in the results section. The data structure consists of four fields. '$r()$' and '$w()$' mark reading and writing a field. Edges with circles denote control dependencies in the CDFG. Column "No." in the table shows the total number of corresponding dependency cases.

The compatibility graph (G=(N,E)) of basic-groups is built based on the collected dependency cases. The nodes of the graph, each node corresponding to a BG, are grouped into clusters using a partitioning/clustering method commonly referred as "hierarchical clustering" [3].

Every node, $n_i \in N$, of the compatibility graph has two weights associated with it - one to represent writings (w_i) and another to represent readings (r_i). The value of a weight is equal to the number of corresponding accesses. For clarity, the nodes of the compatibility graph in the figure are placed in such a way that their "read-sides" are directed to the left. Every node is marked with its field name, bit-width and with the number of read and write accesses.

Any relevant pair of memory accesses forms a weighted edge in the graph. The edges, $<r_i,r_j> \in E$ or $<w_i,w_j> \in E$, correspond to dependency cases, i.e. *read-read* or *write-write*. The weight of an edge, $r_{i,j}$ or $w_{i,j}$, is equal to the number of corresponding dependency cases. It should be noted that $r_i \geq r_{i,j}$ and $w_i \geq w_{i,j}$ because the number of cases related to two accesses can never be larger than the smallest of these two accesses. Every edge is marked with its weight.

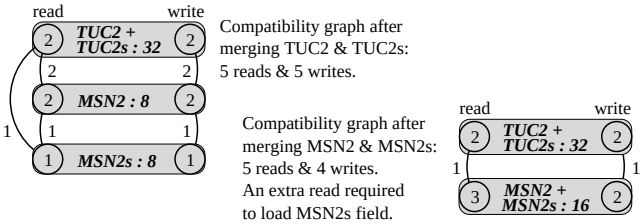


Figure 3: Clustering example

3.3 Clustering

The clustering algorithm considers pairs of objects (nodes) and groups them according to their closeness. The two closest objects are considered to be a single object in the next iterations of the clustering. The algorithm stops when a single cluster is generated and a hierarchical cluster tree has been formed. The cost of the compatibility graph for the memory field pre-packing is equal to the sum of weights of nodes - the total number of accesses. The "closest pair" is defined as the pair of nodes which gives the best improvement of the cost function. A set of heuristic re-evaluation rules of weights, described below, have been developed to avoid rebuilding the CDFG and compatibility graph after every clustering iteration.

Modification rules: Let the nodes to be merged are n_i and n_j, and the new node is n_k. The weights of the node n_k and the edges connected to it have the following values:

- read-weight - $r_k = max(r_i, r_j) + (w_i - w_{i,j}) + (w_j - w_{i,j})$ - takes into account that all fields are loaded anyway; $w_i - w_{i,j}$ and $w_j - w_{i,j}$ take into account the number of extra reads needed before writings;
- read-edge-weight - $r_{k,l} = max(r_{i,l}, r_{j,l})$ - the new number of cases *read-read*;
- write-weight - $w_k = max(w_i, w_j)$ - the new number of write accesses;
- write-edge-weight - $w_{k,l} = max(w_{i,l}, w_{j,l})$ - the new number of cases *write-write*.

Figure 3 demonstrates the clustering of the example graph from Figure 2. It should be noted that merging fields MSN2 and MSN2s has introduced an extra read to avoid corruption of field MSN2 when writing MSN2s. At the same time the total number of accesses is reduced.

Additionally, the total width of fields in bits can be used as an extra cost or as a constraint. The cost function can be modified to evaluate also the uniformity of the field widths. Field splitting, another possibility to make the word width uniform, can be performed using the same clustering principles described above.

4. EXPLORATION RESULTS

Four subsets from real-life ATM cell processing applications [4; 9; 8] have been used as test drivers for pre-packing. The functionality of the drivers is dominated by intensive manipulation of dynamically allocated data structures. Exploration results after the memory allocation and assignment phase are shown in Table 1. "Critical path" is the number of memory accesses in the critical path. Relative memory

Table 1: Results after memory allocation and assignment

Design		Number of accesses	Number of BG-s	Crtical path	Number of memories	Relative size	Relative power
#1	without pre-packing	18	18	12	4	1.0	1.0
	with pre-packing	6	4	4	4	0.798	0.397
#2	without pre-packing	85	44	50	8	1.0	1.0
	with pre-packing #1	38	17	30	8	0.907	0.552
#3	without pre-packing	234	49	-	-	-	-
	intuitive packing	149	22	29	8	1.0	1.0
	with pre-packing #1	156	19	34	7	1.011	0.787
#4	without pre-packing	219	23	-	-	-	-
	intuitive packing	76	16	-	10	1.0	1.0
	with pre-packing #1	36	8	-	6	0.970	0.592
	with pre-packing #2	35	7	-	6	0.970	0.586
Average						0.931	0.583

size and relative power consumption are measured using accurate memory models against the solutions without pre-packing (if available). The basic groups are counted after pre-packing and the number of memories shows the number of physical memories after the memory allocation phase.

Design #3 represents a network component of an ATM Segment Protocol Processor [8]. The first (intuitive) packing alternative is based on manually packing data together to greedily reduce the number of accesses without involving any trade-off. By formally exploring the search space, it is possible to get another combination of fields that increases slightly the area size and the number of accesses but finally leads to a power improvement of more than 20% (*pre-packing #1*). Design #4 illustrates the Operation and Maintenance (OAM) handler of an ATM switch. The *intuitive packing* case corresponds to a process partitioned description that is synthesised using conventional high-level synthesis tools[4]. However, cases *pre-packing #1* and *pre-packing #2* corresponds to the approaches illustrated in Figures 2 and 3 while keeping the total number of memories constant.

For small size drivers (e.g., drivers #1 and #2), the proposed methodology gives very similar results when compared to the ones obtained without actual exploration (e.g., steered by a greedy reduction in the number of accesses). However, for larger ones (e.g., drivers #3 and #4) it clearly outperformed the intuitive approaches. This substantiates the necessity and validity of formalised techniques for data format exploration. Some other points of interests, suggested by the clustering, were explored for further optimisation, but the final area/power figures did not change much. Very small differences in area and power consumption point out the need for accurate estimates for finer trade-offs [11].

5. CONCLUSIONS

In this paper we have shown that for data dominated applications, exploring data format alternatives at the system level is crucial to enable significant reduction in all three memory cost aspects: size, bandwidth and power. We propose a formalise approach to efficiently explore the search space available before the actual memory mapping stage. This exploration results in significant reduction in the number of accessed bits. We have validated our techniques using several real-life ATM cell processing applications where we have obtained significant reductions in memory size (up to 20%), power (up to a 60%) and bandwidth.

6. ACKNOWLEDGEMENTS

We gratefully acknowledge the discussions with our colleagues at IMEC especially to Arnout Vandecappelle and Sven Wuytack and at University of Queensland to Adam Postula.

7. REFERENCES

[1] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, A. Vandecappelle, "Custom Memory Management Methodology – Exploration of Memory Organisation for Embedded Multimedia System Design", Kluwer Academic Publishers, 1998.

[2] P. Ellervee, M. Miranda, F. Catthoor, A. Hemani, "Exploiting Data Transfer Locality in Memory Mapping", *Proc. 25th Euromicro Conf.*, pp. 14-21, Sept. 1999.

[3] D. Gajski, N. Dutt, A. Wu, "High-level synthesis: introduction to chip and system design", Kluwer Academic Publishers, 1992.

[4] A. Hemani, B. Svantesson, P. Ellervee, A. Postula, J. Öberg, A. Jantsch, H. Tenhunen, "High-Level Synthesis of Control and Memory Intensive Communications System", *Proc. 8th Annual IEEE Intnl. ASIC Conference and Exhibit*, pp. 185-191, Sept. 1995.

[5] F. J. Kurdahi, A. C. Parker, "REAL: a program for register allocation", *Proc. 24th ACM/IEEE Design Automation Conf.*, pp. 210-215, June 1987.

[6] L. Ramachandran, D. Gajski, V. Chaiyakul, "An algorithm for array variable clustering", *Proc. 5th ACM/IEEE Europ. Design and Test Conf.*, pp. 262-266, Feb. 1994.

[7] H. Schmit, D. Thomas, "Synthesis of Application-Specific Memory Designs", *IEEE Trans. on VLSI Systems*, Vol.5, No.1, pp. 101-111, Mar. 1997.

[8] J. L. da Silva Jr., C. Ykman-Couvreur, M. Miranda, K. Croes, S. Wuytack, G. de Jong, F. Catthoor, D. Verkest, P. Six, H. De Man, "Efficient System Exploration and Synthesis of Applications with Dynamic Data Storage and Intensive Data Transfer", *Proc. 35th ACM/IEEE Design Automation Conf.*, pp. 76-81, June 1998.

[9] P. Slock, S. Wuytack, F. Catthoor, G. de Jong, "Fast and extensive system-level memory exploration for ATM applications", *Proc. 10th ACM/IEEE Intnl. Symp. on System-Level Synthesis*, pp. 74-81, Sep. 1997.

[10] J. Van Meerbergen, P. Lippens, W. Verhaegh, A. van der Werf, "PHIDEO: high-level synthesis for high throughput applications", *Journal of VLSI signal processing*, Vol.9, No.1/2, Kluwer, pp. 89-104, Jan. 1995.

[11] A. Vandecappelle, M. Miranda, E. Brockmeyer, F. Catthoor, D. Verkest, "Global Multimedia System Design Exploration using Accurate Memory Organization Feedback", *Proc. 36th ACM/IEEE Design Automation Conf.*, pp. 327-332, June 1999.

[12] I. Verbauwhede, F. Catthoor, J. Vandewalle, H. De Man, "In-place memory management of algebraic algorithms on application-specific IC's", *Journal of VLSI signal processing*, Vol.3, Kluwer, pp. 193-200, 1991.