Power Exploration for Dynamic Data Types through Virtual Memory Management Refinement

Julio L. da Silva Jr, Francky Catthoor, Diederik Verkest, Hugo De Man

IMEC, Kapeldreef 75, B-3001 Leuven, Belgium

Abstract

In this paper we present our novel power exploration methodology for applications with dynamic data types. Our methodology is crucial to obtain effective solutions in an embedded (HW or SW) processor context. The contributions are twofold. First we define the complete search space for Virtual Memory Management (VMM) mechanisms in a structured way with orthogonal decision trees. Secondly we present our systematic methodology for exploration of the maximal power that takes into account characteristics of the application to heavily prune the search space guiding the choices of a VMM mechanism. Finally we demonstrate for two industrial examples that power can vary considerably depending on the VMM chosen. Moreover these experiments show the effectiveness of our exploration methodology.

1 Introduction

We target applications that require manipulation of large amounts of data that are dynamically created and destroyed at run time, such as protocol processing applications. These applications are characterized by tight interaction between control and data-flow behavior, intensive data storage and transfers, and stringent real-time requirements. Due to this, the transport layers of these applications are usually (partly) realised in hardware or embedded software. In the embedded processor design for our target domain, a large part of the area is due to memory units [21]. Also the power for such data-dominated applications is heavily dominated by the storage and transfers (as demonstrated by recent work in the IRAM project at Berkeley, at IMEC [4] and at Princeton [20]).

Given the data storage and transfers importance, we propose a systematic design methodology in which the dynamic storage related issues are globally optimized as a first step, *before* doing the software/hardware or processor partitioning and the detailed compilation on an embedded processor, or the scheduling, data-path and controller synthesis for custom hardware mapping. This preprocessing supports exploration of different data types for Abstract Data Types (ADTs) [24] and exploration of Virtual Memory Management (VMM) mechanisms for these data types, abstracting them from physical memory implementations.

This paper focuses on the exploration of maximal power for different VMM mechanisms in the above target domain. The rest of the paper is organized as follows. First an overview of related work is given. Then we outline our overall design flow and situate the VMM step. After that the full search space for this crucial step is formally defined. Based on this search space and on characteristics of the application, a novel methodology for deciding on power efficient VMMs is described. The large impact on maximal power and performance issues will be illustrated based on an industrial Segment Protocol Processor (SPP) demonstrator for the ATM adaptation layer (from Alcatel [21]), and also on part of an operation and maintenance for ATM protocol (see [8]).

2 Related Work

In the software community much literature is available about possible implementation choices for VMM mechanisms (see [1, 23] and its references) but none of the earlier work provides a complete search space useful for a systematic exploration. Moreover, in the SW community the main criterium used for choosing a VMM mechanism is its speed and secondary its memory usage. Power is never an issue since their target is a computer or workstation. However, our target is an embedded hardware/software realization in which power has a major impact [3].

The memory-oriented synthesis techniques proposed for multimedia signal processing (MSP) applications are not suited as such. In contrast to protocol processing applications that manipulate dynamic data types, MSP applications manipulate (multi-dimensional) array signal streams that can be largely analyzed at compile time.

In [19] synthesis of network components is dealt with, but at a much lower level of abstraction than the one proposed in this paper.

Up to now no systematic power oriented exploration approach has been published to target this important field. Indeed, most effort up to now has been spent, either on data-path oriented work (e.g. [5]), on control-dominated logic or on programmable processors (see [16] for a good overview). Previous algorithm-level low power work [12, 25] has focused on signal processing applications.

3 Dynamic Memory Management Design Flow

Figure 1 gives an overview of the proposed dynamic memory management (DMM) design flow. At the highest level, the application is specified in terms of *abstract data types* (ADTs). The *ADT refinement step* [24] refines these ADTs into concrete data types. These usually consist of a combination of instantiations of concrete data types.

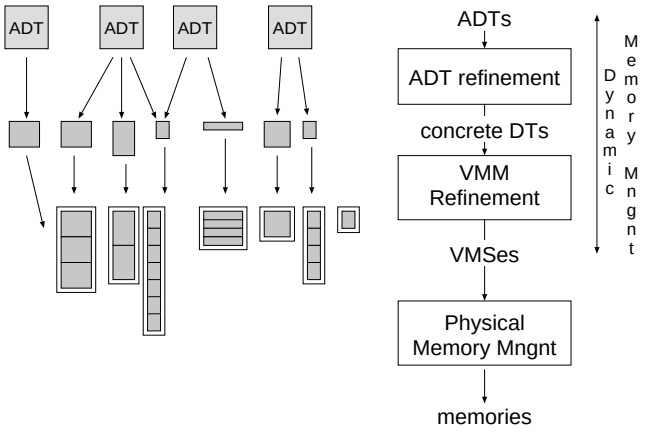


Figure 1: Dynamic memory management design flow

Next, the *Virtual Memory Management (VMM) step* defines a number of *virtual memory segments* (VMSes) and their corresponding custom memory managers. Each VMS consists of a reserved amount of memory to store all instances of one or more concrete data types. To this end, the VMS is divided into a number of blocks, each capable of storing a single instance of the data type. The VMM step determines, via analysis or simulation of a number of scenarios, the amount of blocks that is required to store all instances of that data type. If instances of the data type have to be dynamically created and destroyed, the VMM step also determines a custom memory manager for that VMS. Its task is to return a handle to a free block in the VMS when requested during creation of an instance (allocation), and to mark a block as free again after destruction of an instance (recycling). The ADT and VMM refinement are combined in the DMM stage.

During the subsequent *Physical Memory Management (PMM) stage*, in our overall memory management flow, the VMSes will be assigned to a number of allocated memories. This stage determines the size of the memories in terms of bit width and word depth as well as the number and type (read, write, or read/write) of ports on each memory. The final result is a custom memory architecture, heavily optimized for power and/or area, for the given application. An environment which is suitable for this purpose has been proposed in [15] but also other similar approaches could be used as backend for our DMM stage.

In the next section the VMM refinement step will be detailed. ADT refinement [24] and the PMM stage [15] are out of the scope of this paper.

4 Virtual Memory Management Search Space

VMM consists of two tasks: allocation and recycling. Allocation is the mechanism that searches the pool of free blocks and returns a free block large enough in order to satisfy a request of a given application. Recycling is the mechanism that returns a block which is not used anymore to the pool of free blocks enabling its reuse.

The application may free blocks in any order, creating holes (free blocks) among alive blocks. If these holes are small and numerous they cannot be used to satisfy future requests for larger blocks. This problem is known as fragmentation. Fragmentation is divided in two categories: external and internal. External fragmentation happens when the total memory space is available to satisfy a request, but it is not contiguous. Internal fragmentation happens when the free block chosen to satisfy a request is slightly larger than the necessary one, resulting in a segment of memory internal to a block not being used. Allocation mechanisms make use of splitting and merging of free blocks to keep memory fragmentation under control.

In the software community the programmer either uses the default VMM mechanism provided in a library or implements his/her own mechanism when not satisfied with performance (speed or memory usage) of the default one. An alternative is to use for instance the approach of [1] which provides a set of VMM mechanisms. The programmer may experiment with these mechanisms and then choose the most convenient one. Yet another possibility, is to use a methodology [7] that evaluates characteristics of the application and provides the best VMM mechanism for that application. We follow the latter approach. However, in order to derive such a method it is essential to understand all the relevant implementation possibilities for VMM mechanisms.

A systematic exploration is only feasible in practice by identifying the orthogonal decision trees in the available search space from which all VMM schemes can then be derived by combination. Based on the search space a methodology (manually or automatically steered) that chooses the best VMM mechanisms for a given application can be derived.

In our search space (see Figure 2), any combination of a leaf from each of the decision trees represents a valid VMM mechanism[1]. A solid arc between two or more choices represents that all possible combinations between these branches of the decision tree are feasible. A dashed arc between two choices represents a set of possibilities that can vary from one extreme to another in the decision tree over an enumerated axis. This search space representation is our first contribution in this paper.

In the following subsections we present the decision trees for allocation and recycling mechanisms. The definition of the search space has enabled us to classify several classical allocation mechanism, such as, ordered binary trees [17], segregated lists [14], and buddy systems [13].

4.1 Keeping track of free blocks

The possibilities for keeping track of free blocks are depicted in Figure 2a. Free blocks may have fixed allowable sizes (such as powers of two in [10]) or not. The allocation mechanism keeps track of free blocks using either link fields within free blocks or lookup tables. Lookup tables may use state variables indicating free or used blocks [2], or tables containing ranges of sizes [14]. The free blocks may be indexed by size, or address, or both [18], etc. The decision trees for ADT refinement are described in [24]. Free blocks may be organized in data types such as: linked lists, trees, pointer arrays or arrays.

Using link fields within free blocks avoids overhead in terms of memory usage as long as a minimum block size is respected. On the other hand, lookup tables always incur an overhead in terms of memory usage.

4.2 Recording information about the block

The possibilities for recording information about the block are depicted in Figure 2b. A block may optionally contain information about itself, such as size and/or whether it

[1]We do not consider implicit recycling mechanisms, known as garbage collectors, in our search space.

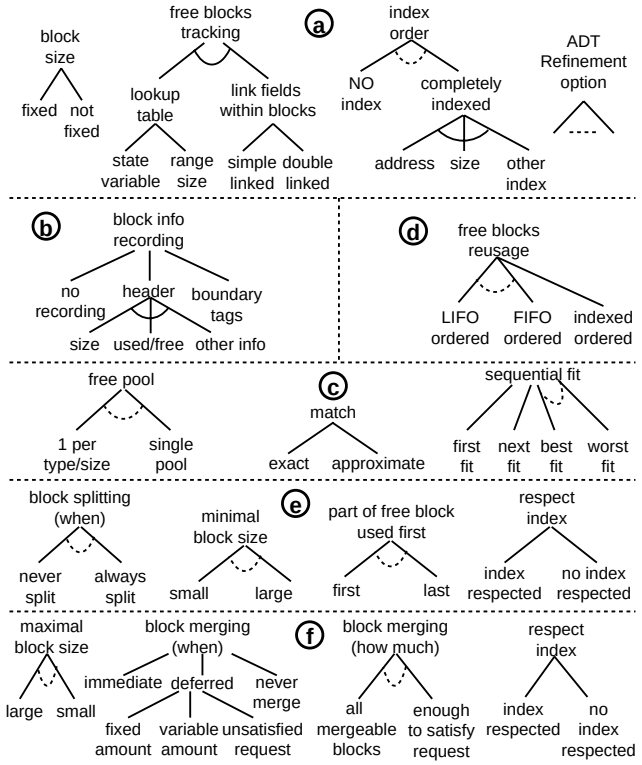


Figure 2: Decision trees for VMM mechanisms

is used or free, and/or information about relationship with neighbors, and/or any other useful information.

Recording information about the block implies an overhead in terms of memory usage. The smaller the block size allocated, the more significant the overhead is. However, it is useful to record information about the block when implementing policies for block merging and splitting. For instance, boundary tags are used for general merging of free blocks. They consist of a header and a footer, both of which record the block size and if the block is in use or free. When a block is freed the footer of the preceding block of memory is examined. Adjacent free areas may then be merged to form larger free blocks.

4.3 Choosing a free block

The possibilities for choosing a block from the free block pool, in order to satisfy a given request, are depicted in Figure 2c. Free blocks may be either in a single pool for all blocks or grouped in sectors [14]. These sectors may group blocks per size or type. The sectors may use either an exact match policy, in which a sector groups blocks of a specific size, or an approximate match policy, in which a sector group blocks of a set of sizes.

In a sequential way the allocation mechanism tries to satisfy a given request by finding either the first free block large enough (*first fit*) or the best match fit (*best fit*). A variation of first fit (*next fit*) [10] keeps a pointer to the free block after the previous allocated block. This pointer, is used as a starting point for searching the next free block. Opposed to best fit there is a policy known as *worst fit*.

When sizes or types to be allocated are known at compile time, keeping different sectors per size improves allocation speed and eliminates internal fragmentation, but it may increase external fragmentation because blocks of one sector can not be reused in another sector. So a trade off is involved.

4.4 Freeing used blocks

The possibilities for returning a recently freed block to the pool of free blocks are depicted in Figure 2d. The indexed ordered option [18] is usually slower than the FIFO and LIFO [22] ordered options. It returns a block to the pool of free blocks respecting an index order instead of simply returning it to the top or bottom of the pool. However, it may avoid wasted memory when combined with merging and splitting techniques. The performance of an indexed ordered scheme may be improved by using hashing, but it does not work well for all ADT choices. There is a clear trade off between speed and area in this choice.

4.5 Splitting blocks being allocated

When the free block chosen to satisfy a request is larger than the necessary one, a policy for splitting the block being allocated should be implemented. The possibilities for splitting are depicted in Figure 2e.

The splitting of a block may be done: never, sometimes or always. The splitting may be done only if the block has a minimum size [10]. Which part of the free block is used first should be chosen. The splitting may have to respect some index, such as size. The remainder of the split returns to the pool of free blocks obeying a decision tree equivalent to the one presented in Section 4.4.

4.6 Merging free blocks

When adjacent blocks are free they may be merged following one of the possibilities depicted in Figure 2f. For instance, for already large blocks it may be uninteresting to merge them, because the result will be an even larger block for which a suitable allocation request may never be issued.

In general it is interesting to defer the merging in order to avoid subsequent splitting operations. Deferred merging [11] may be implemented in different ways: wait for a fixed or variable amount of allocation requests before merging or wait for an unsatisfied allocation request before merging. The amount of blocks to be merged should be chosen from merging all mergeable blocks to merging only enough blocks to satisfy the last request. When the free blocks are kept in an indexed ordered way, the merging mechanism must also respect the index used.

5 Methodology for Maximal Power Exploration

We have identified a set of orthogonal decisions that should be taken when implementing a VMM mechanism. The decisions that should be taken in these trees are not totally independent however. A decision taken in one tree may affect the cost characteristics of other decision trees. A systematic exploration methodology consists of: identifying how much each decision influences a given parameter, identifying the dependencies among the decisions, and making proper use of the knowledge about the applications.

Given a parameter (e.g. maximal power, average power or area) to be minimized and given design constraints (e.g. timing), the decisions should be taken starting with the ones that have a major influence on the chosen parameter, but also taking into account the effects on other trees.

In embedded implementations we must take advantage of any information available at compile time. Taking into account the knowledge of the applications that are using the VMM mechanisms to be implemented allows to heavily prune our search space. This knowledge enables us to find a power efficient VMM mechanism while avoiding unallowable memory overhead and fragmentation. Three important factors help in guiding the choices in the decision trees previously described: the knowledge about all possible types or sizes used in the application, the knowledge about the maximum amount of instances of each type requested by the application, and the maximum total amount of data alive at any time.

In a memory oriented power model, the access count (number of accesses to memory) has the major (linear) influence on power, because the memory size only has (strongly) sublinear effect [9]. For a given record the access count is the sum of the number of accesses during its allocation plus the accesses during its lifetime plus the ones during deallocation. Hence, when the worst case number of accesses during allocation and deallocation of a block is dominant over the number of accesses during its lifetime, the maximal power exploration can be based on the decision ordering presented below. However, for other applications, the number of accesses to some VMSes during its lifetime may be dominant over the number of accesses during allocation and deallocation. In this case it does make sense to try to minimize memory size which still influences power in a less direct way. The methodology for size dominated applications is out of the scope of this paper. Also other parameters such as area overhead in logic, code size or complexity may be taken into account.

We will now detail our methodology for access count dominated applications. In Figure 3, the arrows show which are the most significant inter-dependencies among the decision trees in the VMM search space. The most important decision tree involves choosing between using one sector per type or using an entire pool for all types since this has a global effect. It affects all data types and all the other decisions. Next, the choice about the use of an index to order the free blocks and the merging /splitting choices have also an impact in several trees. However, this impact is now per data type or set of data types that are grouped together, according to the first decision.

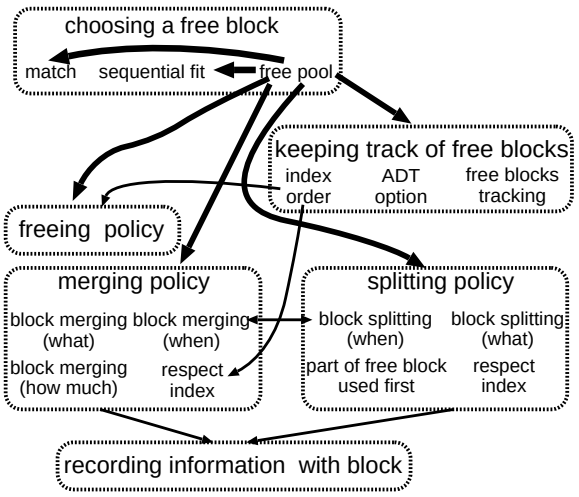
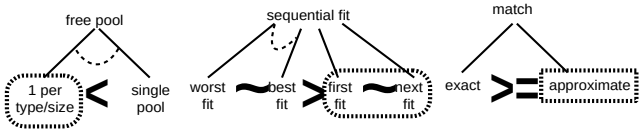


Figure 3: Dependencies among decision trees

All other choices are done for each sector independently. The second decision that has influence on several other decision trees is about the use of an index ordering. Apart from these major inter-dependencies, the other decisions are independent in terms of influence on the number of accesses. Thus we can choose in each decision tree the leaf that minimizes the number of accesses locally.

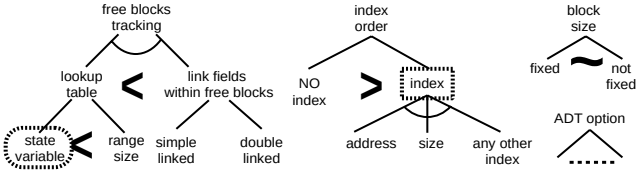
Next, we describe for each decision tree the local effect on power by comparing the impact of each leaf of these trees on the worst case number of accesses (NA)[2]. We also indicate in the accompanying figures each best local choice surrounded by a dashed line.

Local influence on NA: choosing a free block



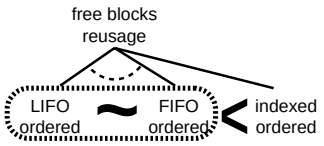
• NA is smaller when having one sector per type or size instead of one entire pool for all blocks. Sequential fits do not scale well and depending on the size of the pool of free blocks they are unusable due to the high NA necessary for allocation. Having one sector per type or size decreases the scalability problems inherent to sequential fit policies.
• NA is usually the same for next fit and first fit policies and both are usually smaller than best fit. This is due to the behavior of the best fit policy that not only requires a block large enough for satisfying a request but also the best fit for the given request.
• NA for an approximate match is smaller than finding an exact match.

Local influence on NA: keeping track of free blocks



• NA is usually the same for fixed or not fixed allowable sizes.
• NA for tracking blocks with lookup tables is smaller than using link field within free blocks. Using a state variable scheme requires fewer accesses than range size scheme.
• NA for indexed ordered structures is usually smaller than for non-indexed ordered ones.
• NA for different ADTs choices varies according to [24].

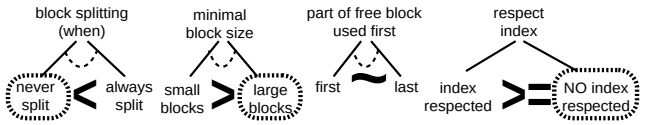
Local influence on NA: freeing used blocks



• NA for FIFO and LIFO ordered is the same. However, when an index ordered is used, returning free blocks must obey this index and a larger NA is necessary to find out the right location to return the free block.

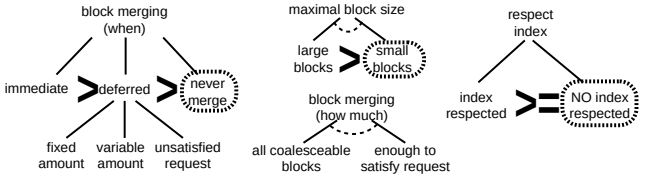
[2]The NA is always the same for all leaves in the decision tree that concerns recording information about the block (Figure 2b).

Local influence on NA: splitting blocks



• NA is smaller when splitting is not used (it may be harder to find desired block).
• NA is smaller when a minimum block size for splitting is imposed instead of splitting already small blocks.
• NA is the same for using the last or first part of a free block.
• NA is larger when the splitting of free blocks must respect some index because after splitting the remainder must be placed in the indexed structure in the right position.

Local influence on NA: merging free blocks



• NA is smaller when merging is not used (it may be harder to find the desired block). Immediate merging requires a larger NA than deferred merging.
• NA is smaller when a maximum block size for merging is imposed instead of merging already large blocks.
• NA concerning the amount of blocks to be merged (all or enough to satisfy request) depends on the application behavior.
• NA is larger when the merging of free blocks must respect some index because after the merging the block must be placed in the indexed structure in the right position.

6 Experimental Results on Industrial Drivers

We present results for two applications: SPP [21], an implementation of ATM Adaptation Layer 3/4, and F4 [8], an operation and maintenance part for the ATM protocol. Using our systematic methodology we can efficiently traverse the search space of Section 4 for the best solutions in terms of access count, power or area. In this paper we look only at worst case access count and maximal power consumption.

In the SPP, the largest block size is 7 words and the smallest is 2 words. The total memory size is targeted to be 128 Kwords (each word 64 bits) for the main off-chip memory. In table 1, the three data types in the SPP that correspond to the the largest amount of data in the main off-chip memory are presented. Also their relative area contribution is shown. Cell stands for ATM cells, IPI stands for Internal Packet Identifier, and RR stands for Routing Record.

	$Area$ (%)	$Width$ (bits)	$VMSSize$ (No. of blocks)
$Cell$	62	448	11200
IPI	16	384	3350
RR	22	128	13400

Table 1: SPP Data Types.

In table 2 we show figures for each data type using three VMM schemes selected from our decision trees. All three schemes use LIFO single linked lists for tracking free blocks. Scheme 1 uses three different VMSes, one for each data type. Scheme 2 uses one VMS for all data types. Scheme 3 uses two different VMSes, one for Cell and another for IPI and RR. Scheme 2 and 3 use merging/splitting to avoid internal fragmentation since they allow different data types to share the same storage area.

	$Scheme1$		$Scheme2$		$Scheme3$	
	$Power$ LT (%)	$Power$ VMM (%)	$Power$ LT (%)	$Power$ VMM (%)	$Power$ LT (%)	$Power$ VMM (%)
$Cell$	62	28	22	74	61	28
IPI	5.98	0.02	2.2	0.1	6.18	0.02
RR	3.98	0.02	1.55	0.15	4.78	0.02

Table 2: SPP - Power distribution using 3 VMM mechanisms.

Note that power is given in relative numbers instead of absolute numbers due to the confidentiality of the memory power model being used. Although we have shown only three possible VMM schemes, a large range of power budgets exists and it is crucial to identify a (nearly) optimal solution in the many decision trees of Section 4. Our methodology does allow to identify these optimal choices.

Cell is an access count dominated data type, while IPI and RR are size dominated data types, concerning number of accesses during lifetime compared to number of accesses during (de)allocation. The power consumed during (de)allocation (VMM) for IPI and RR is much smaller compared to the power consumed during their lifetime (LT). The power consumed during (de)allocation (VMM) for Cell is in the same order of magnitude as the power consumed during its lifetime (LT). This is valid independent of the VMM scheme being used.

In table 3 we show the relative power consumption of the different VMM schemes. We can see that Scheme 2 consumes 2.89 times as much power as Scheme 1, while Scheme 3 consumes 1.01 times as much. By using Scheme 3, power increases only 1% and it allows IPI and RR to share the same memory (obeying exclusion of lifetimes), resulting in a decreased area.

	$Scheme1$	$Scheme2$	$Scheme3$
$Cell$	1	3.1	1
IPI	1	1.05	1.05
RR	1	1.09	1.09
$Total$	1	2.89	1.01

Table 3: SPP - Power comparison of 3 VMM mechanisms.

In the F4 application, only three data types are shown in table 4, which correspond to the the largest amount of data. Cell stands for ATM cells, table1 is used for dynamic process management, and table2 is used for fault management.

	$Area$ (%)	$Width$ (bits)	$VMSSize$ (No. of blocks)
$Cell$	71	448	256
$table1$	27	172	256
$table2$	2	13	256

Table 4: F4 Data Types.

In table 5 we show figures for each data type using two VMM schemes. Scheme 1 uses three different VMSes, one

for each data type. Scheme 2 uses one VMS for all data types. Scheme 2 uses merging/splitting to avoid internal fragmentation since it allows different data types to share the same storage area.

	*Scheme*1		*Scheme*2	
	Power LT (%)	*Power VMM* (%)	*Power LT* (%)	*Power VMM* (%)
Cell	51.7	26.9	5.5	84.4
*table*1	18.2	2.0	1.9	6.4
*table*2	0.7	0.5	0.1	1.7

Table 5: F4 - Power distribution using 2 VMM mechanisms.

Note that all three data types in the F4 are access count dominated data types. Scheme 2 allows these data types to share the same storage space but it is nearly 10 times worse than Scheme 1 in terms of power consumption.

7 Conclusion

The maximal power exploration methodology proposed in this paper is in use in the Matisse design flow [6]. It helps to obtain effective solutions in an embedded (HW or SW) processor context by exploring different VMM mechanisms for applications with dynamic data types.

We have defined the search space for VMM mechanisms in a structured way with orthogonal decision trees. Based on this search space we have presented a systematic power exploration methodology that takes into account characteristics of the applications to prune the search space and guide the choices of a VMM.

We have shown preliminary results for maximum power consumption of different VMM schemes on the SPP demonstrator and on the F4 module. These results are quite promising and show the effectiveness of our exploration methodology.

In the future we intend to do experiments for average power consumption. It is still an open research topic to extend the search space to include implicit recycling mechanisms (known as garbage collectors) and the subsequent definition of a power exploration methodology that supports such mechanisms.

Acknowledgements: We gratefully acknowledge the discussions with our colleagues at IMEC and Alcatel and especially the contributions of M. Miranda, C. Ykman, A. Vandecappelle, S. Wuytack and G. de Jong. This research has been partly funded by the Flemish IWT and Alcatel in the HASTEC project, by the Esprit project MEDIA (No.21929), and by a Brazilian Government Fellowship (CAPES).

References

[1] G.Attardi, T.Flagea, "A Customisable Memory Management Framework", *Proc. of the USENIX C++ Conference*, Cambridge, MA, 1994.

[2] H.Boehm, M.Weiser, "Garbage collection in an uncooperative environment", *Software Practice and Experience*, Vol.18, pp.807-820, Sep. 1988.

[3] R.W.Brodersen, "The network computer and its future", *Proc. IEEE Int. Solid-State Circuits Conf.*,San Francisco, CA, pp.32-36, Feb. 1997.

[4] F.Catthoor, et al., "Global communication and memory optimizing transformations for low power signal processing systems", *IEEE workshop on VLSI signal processing*, La Jolla CA, Oct. 1994. Also in *VLSI Signal Processing VII*, J.Rabaey, P.Chau, J.Eldon (eds.), IEEE Press, New York, pp.178-187, 1994.

[5] "Low power CMOS design", (eds. A.Chandrakasan, R.Brodersen), *IEEE Press*, 1998.

[6] J.L.da Silva Jr, et al., "Efficient System Exploration and Synthesis of Applications with Dynamic Data Storage and Intensive Data Transfer", accepted for *Proc. 35th ACM/IEEE Design Automation Conf.*, San Francisco CA, June 1998.

[7] D.Grunwald, B.Zorn, "CustoMalloc: Efficient Synthsized Memory Allocators", Software: Practice & Experience, Vol.23, No.8, pp.851-869, August 1993.

[8] A.Hemani, et al. "Design of Operation and Maintenance Part of the ATM Protocol", *Journal on Communications*, Hungarian Scientific Society for Telecommunications, special issue on ATM networks, 1995.

[9] K.Itoh, et al., "Trends in low-power RAM circuit technologies", special issue on "Low power electronics" of the *Proceedings of the IEEE*, Vol.83, No.4, pp.524-543, April 1995.

[10] D.Knuth, "The Art of Computer Programming, volume 1: Fundamental Algorithms", Addison-Wesley, Reading, Massachusetts, 1973.

[11] B.Margolin, et al., "Analysis of free-storage algorithms", *IBM Systems Journal*, Vol.10, pp.283-304, April 1971.

[12] P.Panda, N.Dutt, "Low power mapping of behavioral arrays to multiple memories", *Proc. IEEE Intnl. Symp. on Low Power Design*, Monterey CA, pp.289-292, Aug. 1996.

[13] J.Peterson, T.Norman, "Buddy Systems", *Communications of the ACM*, Vol.20, pp.421-431, June 1977.

[14] P.Purdom, et al., "Statistical investigation of three storage allocation algorithms", *BIT*, Vol.11, pp.187-195, Nov. 1971.

[15] P.Slock, et al., "Fast and Extensive System-Level Memory Exploration for ATM Applications", *Proceedings 10th ACM/IEEE International Symposium on System-Level Synthesis*, Antwerp, Belgium, Sep. 1997.

[16] D.Singh, J.Rabaey, M.Pedram, F.Catthoor, S.Rajgopal, N.Sehgal, T.Mozdzen, "Power conscious CAD tools and methodologies: a perspective", special issue on "Low power design" of the *Proceedings of the IEEE*, Vol.83, No.4, pp.570-594, April 1995.

[17] T.Standish, "Data Structure Techniques", Addison-Wesley, Reading, Massachusetts, 1980.

[18] C.Stephenson, "Fast fits: New methods for dynamic storage allocation", *Proc. 9th Symposium on Operating Systems Principles*, pp.30-32, Oct. 1983.

[19] B.Svantesson, et al., "Modeling and synthesis of operational and management system (OAM) of ATM switch fabrics", *Proc. 13th Norchip Conf.*, pp.115-122, Nov. 1995.

[20] V.Tiwari, et al., "Instruction-level power analysis and optimization of software", *Journal of VLSI Signal Processing*, No.13, Kluwer, Boston, pp.223-238, 1996.

[21] Y. Therasse, G. Petit, and M. Delvaux. "VLSI architecture of a SDMS/ATM router", *Annales des Telecommunications*, 48(3-4), 1993.

[22] C.Weinstock, "Dynamic Storage Allocation Techniques", PhD. Thesis, Carnegie-Mellon University, Pittsburgh, April 1976.

[23] P.R.Wilson, et al., "Dynamic Storage Allocation: A Survey and Critical Review", *Proc. Intnl. Wsh. on Memory Management*, Kinross, Scotland, UK, Sep. 1995.

[24] S. Wuytack, F. Catthoor, H. De Man, "Transforming Set Data Types to Power Optimal Data Structures", *IEEE Transactions on Computer-aided Design*, Vol.CAD-15, No.6, pp.619-629, June 1996.

[25] S.Wuytack, et al., "Power Exploration for Data Dominated Video Applications", *Proc. IEEE Intnl. Symp. on Low Power Design*, Monterey CA, pp.359-364, Aug. 1996.