Iason Filippopoulos

Exploration of energy efficient memory organizations exploiting data variable based system scenarios

Doctoral thesis
for the degree of philosophiae doctor

October 2015

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Electronics and Telecommunications

Catholic University of Leuven
Department of Electrical Engineering

NTNU

Norwegian University of Science and Technology

Doctoral thesis
for the degree of philosophiae doctor

Faculty of Information Technology, Mathematics and
Electrical Engineering
Department of Electronics and Telecommunications

Catholic University of Leuven
Department of Electrical Engineering

ISBN (printed version)
ISBN (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU,

Printed by NTNU-trykk

Abstract

Modern embedded systems are capable of performing a wide range of tasks and their popularity is increasing in many different application domains. The recent progress in semiconductor processing technology greatly improves the performance of the embedded systems, due to the increased number of transistors on a single chip. The continuous performance improvement, for example higher clock frequencies and lower supply voltage, increases the possibilities for new embedded system designs. However, many embedded systems rely on a battery source, which puts a significant limitation on their lifetime and usage. The management of the energy consumption is a key factor towards increasing the lifetime of an embedded system relying on a battery source. A typical embedded system includes one or more processing elements (CPUs, GPUs, embedded processors etc.), a memory subsystem (cache, scratchpad, flash, etc.) and application specific hardware (antennas, sensors, etc.). The memory subsystem contributes significantly to the overall energy consumption as shown in many studies of embedded system applications and platforms. Especially for applications that are data intensive, the memory architecture may have the highest energy footprint of the whole embedded system.

A hardware/software co-design methodology is proposed for the reduction of the energy consumption in the memory subsystem based on the system scenario methodology. In general, system scenario methodologies propose the use of different platform configurations in order to exploit run-time variations in application needs. The current methodology exploits variations in memory needs during the lifetime of an application in order to optimize energy usage. The different resource requirements, that change dynamically at run-time, are grouped into scenarios to efficiently handle a very large exploration space. Apart from the development of the methodology, an extended memory model is included in this work. The memory models is based on existing state-of-the-art memories, available from industry and academia. In addition, the impact of the technology scaling is studied and the effectiveness of the proposed methodology is analyzed for the future memory architectures. We also investigate the combination of the developed methodology with known code transformation techniques, specifically data interleaving. The proposed design methodology aims at being compatible with the already available code optimization techniques. Actual decrease in memory energy consumption for a selection of studied applications ranges between 30% and 60%.

Preface

This doctoral thesis was submitted to the Norwegian University of Science and Technology (NTNU) in partial fulfillment of the requirements for the degree philosophiae doctor (PhD). The thesis is a part of a dual PhD program between NTNU and the Catholic University of Leuven (K. U. Leuven), in cooperation with Interuniversity Microelectronics Center (IMEC) in Leuven, Belgium, and Eindhoven, The Netherlands.. The work herein was performed at the Department of Electronics and Telecommunications, NTNU and the Department of Electrical Engineering, KU Leuven. The work was performed under the supervision of Professor Per Gunnar Kjeldsberg and Professor Francky Catthoor.

Acknowledgements

I would like to thank my supervisors Prof. Per Gunnar Kjeldsberg and Prof. Francky Catthoor for their support and advise through the long process that eventually became this thesis. I also extend my gratitude to my co-supervisor Prof. Sverre Hendseth for providing encouragement and positive attitude all these years.

I would also like to thank Mladen for sharing both his direct and thought-provoking opinions and office space with me. Furthermore, I thank Elena and Yahya for our great co-operation and their patience on the evaluation meetings. I was lucky to meet great co-workers at NTNU and IMEC that offered their help.

Finally, I would like to thank my family and friends for their support.

Iason Filippopoulos
October 2015

Contents

List of Tables

List of Figures

List of Abbreviations

BER	bit error rate
CAD	computer aided design
CFG	control flow graph
CGRA	coarse-grain reconfigurable array
DCT	discrete cosine transformation
DTSE	data transfer and storage exploration
DVFS	dynamic voltage and frequency scaling
EEG	electroencephalogram
EPIC	efficient pyramid image coder
FFT	fast Fourier transformation
FU	functional unit
GLT	global loop transformations
ITRS	international technology roadmap for semiconductors
MHLA	memory hierarchy layer assignment
MM	memory macros
MUX	multiplexer
RTL	register transfer language
RTS	run-time situation

SCBD	storage cycle budget distribution
SCMEM	standard cell-based memory
SIMD	single instruction multiple data
SNR	signal to noise ratio
SOR	successive over relaxation
TSMC	Taiwan semiconductor manufacturing company
UML	unified modeling language
WCET	worst case execution time

Chapter 1

Introduction

1.1 Embedded Systems and Energy Consumption

Embedded systems are usually designed to perform one or more specific tasks and often consist of domain-specific hardware [30]. For example, typical embedded systems use optimized processing cores to perform signal processing instead of using general purpose CPUs. Some typical examples of embedded systems include TV sets, cellular phones, MP3 players, smart cameras, wireless access points and printers. Health and automotive domains also rely on embedded systems for several different tasks ([54], [103]). An embedded system is designed with strict requirements regarding size, performance and power consumption. The market demand is towards smaller and lighter devices. The ever-progressing semiconductor processing technique has dramatically increased the number of transistors on a single chip, which makes today's hardware increasingly powerful. Embedded systems often rely on a battery source to deliver the desired performance and the energy efficiency is a significant design factor. Assuming that the development in the battery technology will follow the current trend, embedded systems should improve their energy efficiency based on the system design. The slow improvement of the battery technology relative to the growth of power demand from embedded systems is analyzed in [94].

The memory subsystem of an embedded system has to meet the same requirements regarding size, performance and power consumption. Many applications focusing on embedded systems are data intensive and the contribution of the memory to the overall system is significant, e.g. up to 50% in [108]. This work focus on the exploration of energy efficient memory organizations suitable for embedded systems. The goal is to contribute to the development of a systematic way of designing a memory architecture that is energy efficient and meets performance requirements. The current work presents a methodology to exploit variations in memory needs during the lifetime of an application in order to optimize energy usage.

1.2 Dynamic Data Intensive Applications

Data intensive applications perform tasks that involve operations on large sets of data. Thus, the memory requirements of data intensive applications are high and the contribution of the memory to the overall energy consumption significant. The main focus of this thesis is on applications that are both dynamic and data intensive. The dynamism in this context refers to the significant changes in the behavior of the application. In more detail, the studied applications exhibit a dynamic variation in the memory requirements during their lifetime. The dynamic variation in the memory requirements can be input driver, which means that there is a wide variation on the execution of the application based on different inputs. Because of this behavior, a static study of the application code alone is insufficient since the targeted applications have non-deterministic behavior that is driven by input.

1.3 Problem Statement

The general problem is expressed in the following form:

> Assume a given dynamic embedded system application with memory requirements that vary through its lifetime and its range of inputs. Find the most suitable memory architecture and fully exploit its features to fulfill the performance requirements and minimize the energy consumption.

The application is dynamic and the memory requirements vary through its lifetime, so there are opportunities for system optimization based on estimations of the system resources. The system resources include all the different memories available in the system. Different memory platform configurations and data-to-memory assignments provide opportunities for system optimization. In order to provide performance guaranties the estimations should be pessimistic, and not optimistic, as over-estimates are acceptable, but under-estimates are generally not. Currently used design approaches often use worst case estimations, which are obtained by statically analyzing the application. However, these techniques are not efficient when focusing on dynamic and input driven applications. Due to the dynamism in target applications, the ratio of the worst case load versus the average load on the memory is normally high. Hence, if only the worst case estimations are used during design, the resulting system would not be able to exploit this gap.

A way to solve this problem is to design the system to meet the worst case requirements, but add reconfiguration knobs that can exploit the variation in the memory requirements (e.g., by switching off hardware components, which decreases the energy consumption). A run-time mechanism that predicts the current application needs in terms of resources and exploits this information should also be integrated into the system. To enable this exploitation, the possible instances in which the application may run, together with their resource needs, should be known and taken into account during design. The number of different inputs and the variations in the memory requirements for each instance provide a huge exploration space that is difficult to handle, as it is almost impossible to enumerate every possible case. Even if the explosion problem could be solved, it will be very difficult to predict at run-time in which instance the application is running and the platform reconfiguration needed to better exploit it. In addition, the run-time overhead for switching to a different reconfiguration for every instance could not be compensated by the improvements in the energy consumption since the reconfiguration of the platform has an energy penalty.

1.4 Current approaches

The presented problem has been studied before and different ways of tackling it have been proposed. However, there are some aspects that have not been addressed before that are presented in this thesis.

Most of the current approaches rely on a static analysis of the target application and several methodologies have been presented to generate a static application-specific memory hierarchy [10]. Several techniques for designing energy efficient memory architectures for embedded systems are presented in [76]. The main limitation on these methodologies is the fact that they are applicable to applications with very limited dynamism. This work extends the state of the art by proposing a more generic approach, which is also suitable for applications with input driven dynamic behavior. In addition, the current work differentiates by employing a platform that is reconfigurable at run-time.

The approach of a reconfigurable memory platform has also been proposed several times and an extensive overview of current approaches is found in [31]. Most of the proposed solutions are focusing on tackling one specific case-study application, which is divided in a small number of different cases based on observations at the user level. However, our work differentiates by proposing a more generic and application agnostic methodology and analysis at the system level. Thus, it can efficiently handle a wider range of dynamic application characteristics.

Another proposed approach to tackle the problem, is to focus on source code transformations, and especially loop transformations. These methods try to modify the application code and provide an improved version of the application with easier memory management. The main drawback of the code transformation approach is that it is not always possible to achieve the desired behavior, because the applications can be complex. In any case, these methods are fully complementary to the methodology presented in this thesis and should be performed as a prior step to the current work.

More detailed description of previous work can be found in Chapter 2 and in the individual papers presented in the appendixes of this thesis.

1.5 Thesis contributions

In this thesis, we focus on the design and utilization of energy efficient memory architectures for embedded systems. A hardware/software co-design methodology is proposed for the reduction of the energy consumption in the memory subsystem. The methodology exploits variations in memory needs during the lifetime of an application in order to optimize energy usage. The different resource requirements, which change dynamically at run-time, are organized into groups to efficiently handle a very large exploration space. Apart from the development of the methodology, an extended memory model is included in this work. The memory model is based on existing state-of-the-art memories, available from industry and academia.

In addition, the impact of the technology scaling is studied and the effectiveness of the proposed methodology is analyzed for future memory architectures. We also investigate the combination of the developed methodology with known code transformation techniques, specifically data interleaving. The proposed design methodology aims at being compatible with the already available code optimization techniques. We further extend the evaluation of the memory design methodology using a wireless test-case system. The proposed reconfigurable memory subsystem is studied in a dynamic platform with several reconfiguration options that combine the memory and the processing elements. Reduction in energy consumption in the memory subsystem ranges from 30% to 60% for different sets of benchmarks.

1.6 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 contains background information regarding the work performed in the arrays of data transformations and system scenarios. Chapter 3 describes the research process and presents the developed methodology. Each paper on which the thesis is based is described in chapter 4 with a breakdown of the roles of each author. Chapter 5 concludes the thesis with a summary of contributions. The appendixes hold each paper in chronological order. These papers are reproduced faithfully with regard to the published text, but have been reformatted to increase readability.

Chapter 2

Background

2.1 Data and Memory Management Approaches

Several techniques have been proposed to optimize the memory management and an overview of them is presented in this section. Most techniques propose transformations on the (a) loop and control-flow, (b) data reorganization and mapping, and (c) memory platform generation, and aim to tackle the problem of sub-optimal memory organizations. The sub-optimal memory organization problem has been identified both in compiler theory [5] and high-level synthesis [117].

The loop and control flow transformations aim at improving the data access locality and remove any mismatches in production and consumption ordering of data. They can be applied globally across the full code or within specific loops. Several research efforts have provided powerful environments for interactive loop transformations as long as no other transformations are required. In the parallel compiler domain, interactive environments like Tiny [120], Omega at U.Maryland [62], SUIF at Stanford [12], the Paradigm compiler at Univ. of Illinois [4] and the ParaScope Editor [44] at Univ. of Rice have been presented. These works focus on the code and are platform independent. The current work studies the code transformations in the presence of a platform and presents a more complete work-flow.

The data mapping optimizations focus on finding the optimal mapping of the different arrays into the memory. Energy-aware assignment of data to memory banks for several task-sets based on the MediaBench suit of benchmarks is presented in [78]. Design methods with main focus on the traffic and latencies in the memory architecture are presented in [17], [40],

[58] and [95]. Improving memory energy efficiency based on a study of access patterns is discussed in [61]. The main lack in these works is that the mapping is performed on a static memory architecture.

Another approach to solve the problem is the memory platform generation. The authors in [10] present a methodology to generate a static application-specific memory hierarchy. Later, they extend their work in [9] to a reconfigurable platform with multiple memory banks. Several techniques for designing energy efficient memory architectures for embedded systems are presented in [76]. In [93] a large number of data and memory optimization techniques, that could be dependent or independent of a target platform, are discussed. The authors in [1], [55] and [71] present methodologies for designing memory hierarchies. Application specific memory design is a research topic in [105], while memory design for multimedia applications is presented in [87]. The current thesis differentiates mostly by proposing a more dynamic memory platform and run-time reconfiguration.

2.2 Data Transfer and Storage Exploration

A generic approach to tackle the memory access and power bottlenecks is the data transfer and storage exploration methodology (DTSE) presented in [14]. The methodology presents several optimizations and their exploitation in a systematic way. The main idea is to achieve a more optimal design and mapping of data on the memory organization by applying system-level code transformations to the initial application specification. The price paid there will be an increased system design complexity, which can be offset with appropriate design methodology support tools. The global overview of the DTSE methodology is presented in Fig. 2.1.

Platform independent steps of DTSE reduce the number of array accesses and enable later platform dependent optimization steps. They are beneficial for any platform used later in the design-flow and are briefly presented:

1. Pruning.

   This step is necessary to identify and isolate the parts and data structures in the program which are data-dominant and thus relevant for DTSE. The pruning step also presents this code in a way which is optimally suited for transformations. Thus mostly loops with large bounds and exhibiting good reuse of data and data structures such as

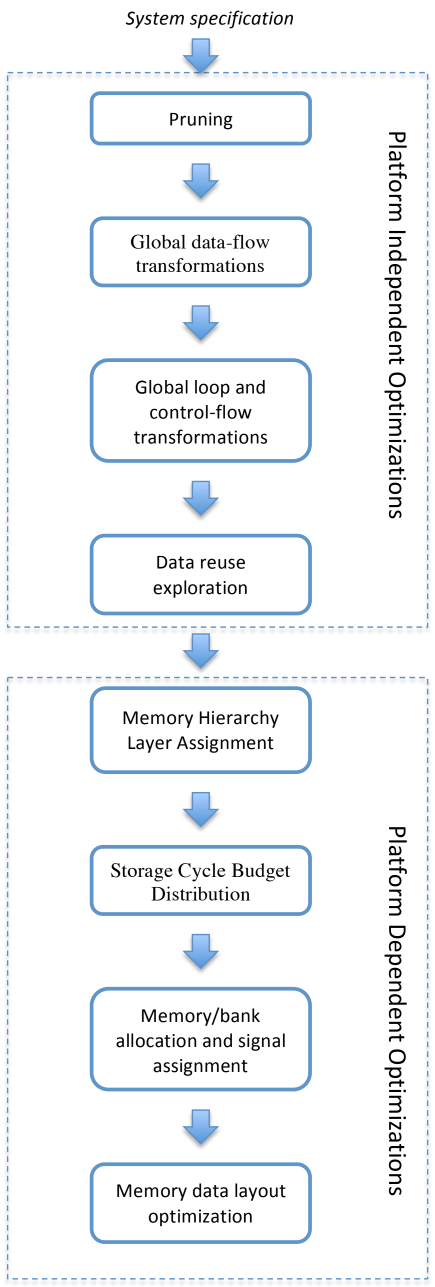

System specification
Pruning
Global data-flow transformations
Global loop and control-flow transformations
Data reuse exploration
Platform Independent Optimizations
Memory Hierarchy Layer Assignment
Storage Cycle Budget Distribution
Memory/bank allocation and signal assignment
Memory data layout optimization
Platform Dependent Optimizations

Figure 2.1: DTSE methodology for data transfer and storage exploration. Source: [89]

array variables are exposed. All the freedom is exposed explicitly, and the complexity of the exploration is reduced by hiding constructs that are not relevant. Apart from areas of power oriented gain, the parts of the program that are bottlenecks for obtaining better performance need to be identified. These are, for example, data structures that have very little locality but are accessed heavily.

2. Global data-flow transformations.

   The goal of the global data-flow optimization step is to reduce the number of bottlenecks in the algorithm that prevent optimizing code restructuring transformations from being applied and to remove access redundancy in the data-flow. The transformations consist mainly of advanced variable substitution avoiding unnecessary copies of data, modifying computation order in associative chains enabling certain loop transformations, shifting of delay lines through the algorithm to reduce the storage requirements, and re-computation issues to reduce the number of transfers and the storage size.

3. Global loop and control-flow transformations.

   The goal of the global loop and control-flow optimization step is to reduce the global lifetimes of the arrays and to increase the locality and regularity of the data accesses. Locality of data accesses means that the accesses to the same memory location have to be close in time while regularity means that the order of consumption should be the same as the order of production. The transformations remove system-level buffers introduced due to mismatches in production and consumption ordering (regularity problems). They allow also the data to be stored later in the design flow in smaller memories closer to the data paths.

4. Data reuse exploration.

   The goal of the data reuse decisions step is to better exploit a hierarchical memory organization to benefit from the available temporal locality in the data accesses. An important consideration here is the distribution of the data over the hierarchy levels such that frequently accessed data can be read from smaller and less power consuming memories. This obviously has a positive effect on the total power consumption of the application because the most frequently accessed data is then read from less power consuming memories. Also the smaller memories can then be closer to the data paths thereby reducing the dissipation in the interconnect, especially if off-chip memory accesses are replaced by on-chip memory accesses.

Platform dependent steps of DTSE uses the information about the predefined memory organization to perform further optimizations. Some substeps only apply for an (embedded) customizable memory organization which is becoming available on several platforms by partly powering down over-dimensioned memory blocks that are not fully needed. The platform dependent steps are briefly presented:

1. Memory Hierarchy Layer Assignment (MHLA) .

   The MHLA step maps the most beneficial candidates from data reuse copy trees to a virtual memory hierarchy subsystem. During MHLA, data reuse copy trees resulting from the platform independent data reuse exploration and the corresponding transfers are partitioned over several hierarchical memory layers, based on the bandwidth and high-level memory size estimation. The high-level memory class of each of the memory layers is determined (e.g., on-chip, off-chip, ROM, SRAM or DRAM and other RAM flavors).

2. Storage Cycle Budget Distribution (SCBD)

   The goal of the SCBD step is to ensure that the (usually stringent) real-time constraints are met with a minimal cost penalty. The major substep involves Storage Budget Optimization (SBO) to determine which data should be made simultaneously accessible in the memory architecture such that the real-time constraints can be met with minimal memory bandwidth related costs. This step mainly determines the bandwidth/latency requirements and the balancing of the available cycle budget over the different memory accesses. Additional loop transformations are performed to meet the real-time constraints, such as merging of loops without dependencies, software pipelining and partial loop unrolling.

3. Memory/bank allocation and signal assignment (MAA).

   The goal of the memory allocation and assignment step is to determine an optimal memory architecture for the background data. The step allocates memory units and ports (including their types) from a memory library and assigns the data to the best suited memory units, given the cycle budget and other timing constraints. The combination of the SCBD and MAA tools allows to derive real Pareto trade-off curves of the background memory related cost (e.g., power) versus the cycle budget.

4. Memory data layout optimization.

   In the memory allocation and data-to-memory assignment step, arrays were assigned to physical memories or to banks within predefined memories. However, the arrays are still multi-dimensional, while the memory itself knows only one dimensional addresses. In other words, the physical address for every array element still has to be determined. This transformation is the data layout decision. Main memory data-layout optimization exploits the memory organization data freedom and thus reduces the conflict misses.

Many steps and techniques of the DTSE methodology are complementary or partly used in the current work. However, the DTSE methodology is static while this thesis focuses on dynamic applications. Another main addition is the dynamic handling of data and the development of a reconfigurable memory architecture.

2.3 System Scenarios

This section describes the basic concepts behind the system scenario methodology and its basic steps. The methodology is applied to a memory specific context in the rest of this thesis.

The goal of the system scenario methodology is, given an application, to exploit at design time its possible operation modes from the resource usage perspective, without getting into an explosion of details. If the environment, the inputs and the hardware architecture status would always be the same, then it would be possible to optimally tune the system to that particular situation. However, since a lot of parameters are changing all the time, the system must be designed for the worst case situation, if a single static solution is employed. Still, it is possible to tune the system at run-time (e.g., change the processor frequency/supply voltage), based on the actual run-time situations (RTS) . An RTS consists of a running instance of a task and its corresponding cost (e.g., energy consumption) and one complete run of the application on the target platform represents a sequence of RTSs [46]. The fine tuning of the system is beneficial, because the system can meet the performance requirements in an energy efficient way by better utilization of its resources. If both the searching for the best solution and the tuning of the system has to happen entirely during run-time, the overhead is most likely too large. So, an optimal set of configurations of the system is selected up front, at design time. However, if a different configuration would be stored for every possible operation mode, a huge database is required. Therefore, the operation modes similar from the resource usage perspective are clustered together into a single scenario, for which we store a tuned configuration for the worst case of all operation modes included in it. The system scenario methodology deals with issues that are common: choosing a good scenario set, deciding which scenario to switch to (or not to switch) and using the scenario to change the system knobs. System knobs are system parameters that are reconfigurable, such as dynamic voltage and frequency scaling (DVFS) or supply voltage on a memory bank. This leads us to the different steps of the methodology:

1. Profiling

   This step combines static analysis and profiling of the application and is done at design time. The studied application is tested exploring the whole range of inputs that is rational for the application. The system resources and the associated cost are found. The costs are the resource usage (e.g., number of processor cycles or memory requirements). If the information about all possible RTSs in which a system may run is known at design time, and the operation modes are considered in different steps of the embedded system design, a more efficient and effective system may be built, as specific and aggressive design decisions can be made for each operation mode [37].

2. Identification

   In this step, the relevant RTSs are selected and clustered into scenarios. This clustering is based on the cost trade-offs of the operation modes, or an estimate thereof. For example, a cost trade-off between an application's execution time and the energy consumption of the CPU is presented in [46]. The identification step should take as much as possible into account the overhead costs introduced in the system by the following steps of the methodology. As this is not easy to achieve, an alternative solution is to refine (i.e., to further cluster) the scenario identification during these steps.

3. Detection/Prediction

At run-time, a scenario has to be selected from the scenario set based on actual parameter values [33]. In general, the parameter values are not known before the operation mode starts. They have to be predicted to enable detection of the corresponding scenario. Detection is not a trivial task: both the number of parameters and the number of scenarios may be considerable, so a simple lookup in a list of scenarios may not be feasible. The detection incurs a certain run-time overhead, which depends on the chosen scenario set. Therefore, the scenario set may be refined based on the detection overhead. The detection mechanism is developed at design-time [75].

4. Switching

   Switching is the act of changing the system from one set of knob positions to another. This implies some overhead (e.g., time and energy), which may be large (e.g., when migrating a task from one processor to another). Therefore, even when a certain scenario (different from the current one) is predicted, it is not always a good idea to switch to it, because the overhead may be larger than the gain. The switching step selects at design time an algorithm, which is used at runtime to decide whether to switch or not. It also introduces in the application the way how to change the knob positions, and refines the scenario set by taking into account switching overhead.

System scenarios have been applied in several cases and for a wide range of applications [75]. The system scenario approach was presented for the first time in [123], where it was used to improve the mapping of dynamic applications onto a multiprocessor platform. In general, previous work on system scenarios has mostly emphasized on exploiting scenarios, and not on identifying and predicting them. In [23], the authors propose the usage of system scenarios for multimedia applications using a processor that supports DVFS. The studied applications are split into parts with large variation in terms of execution time. The proposed approach supplies the information of the execution time variations in run-time, which enables the DVFS reconfiguration of the processor. In [104], authors manually identify system scenarios and define the most energy efficient architecture configuration that meets the timing constraints. Again, a single processor is used and its supply voltage can be changed. In [22], a system scenario approach is used in order to choose the lowest supply voltage for which the timing constraints of an MPEG decoder are met. In all cases, the energy consumption is significantly reduced without violation of the execution time deadlines.

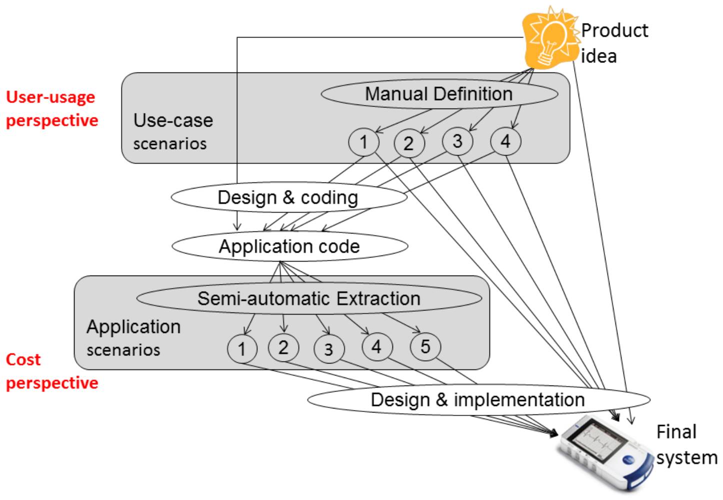


Figure 2.2: A scenario based design flow for embedded systems. Source: [33]

2.3.1 Use-Case vs. System Scenarios

The concept of system scenarios has been presented before in a systematic way on a wide range of applications [33], [75], [37]. Another type of scenario that is used in several published works is the use-case scenario. Use-case scenario approaches generate different scenarios based on a user's behavior. These scenarios concretely describe, in an early phase of the development process, the use of a future system. In case of human-computer interaction, the scenarios appear like narrative descriptions of envisioned usage episodes, and in case of object oriented software engineering like a unified modeling language (UML) use-case diagram which enumerates, from functional and timing point of view, all possible user actions and the system reactions that are required to meet a proposed system function. In the embedded systems domain, use-case scenarios are used in both hardware [52] [97] and software design [27]. In these cases, the scenarios focus on the applications functional and timing behaviors and on its interaction with the users and environment, not on the resources required by a system to meet its constraints. These scenarios are used as an input during system design for user-centered design approaches.

This thesis concentrates on system scenarios, which are derived from the behavior of the application. These scenarios are used to reduce the system cost by exploiting information about what can happen at run-time to make better design decisions. While use-case scenarios classify the application's behavior based on the different ways it can be used, application scenarios classify it from the resource usage perspective, based on the cost trade-off aspects during the mapping to the platform. Fig. 2.2 depicts a design trajectory using use-case and system scenarios. It starts from a product idea, for which the product's functionality is manually defined. These scenarios characterize the system from a user perspective and are used as an input to the design of an embedded system that includes both software and hardware components. In order to optimize the design of the system, the detection and usage of application scenarios augments this trajectory (the bottom gray box in the figure). Once the application is coded, its scenarios related to resource utilization are extracted in an automatic way, and they are considered for the decisions made during the following phases of the system design. Hence, the run-time behavior of the application is classified into several application scenarios, where the cost of the operation modes within a scenario is always fairly similar. For each individual system scenario, more specific and aggressive design decisions can be made.

2.4 Scratch-pad Memory Architectures

Scratch-pad memory architectures are employed in several studies with the objective of improving system performance and energy through means such as reduced memory access count or reduced cache misses. Scratch-pad memories provide a good alternative to cache memories due to their higher flexibility. In cache memory systems, the mapping of data elements is done at run-time, while in scratchpad memory systems this is done by the programmer or the compiler [53]. Unlike the cache memory, the scratchpad memory does not need tag search operations and, as a result, it is more power efficient than the cache memory if programmers or compilers can optimally allocate code and data on the scratchpad memory [109]. On the other hand, caches are used due to the easy integration with the software of the system [110]. The detection of a cache hit or miss is done automatically. If the accessed data is currently not available in the cache, the hardware control automatically copies the data into the cache. Thus, there is no need for the programmer or the compiler to allocate the data, which would be a time consuming task for a general purpose system with several applications. In our case, the application and the memory system are fully analyzed and the allocation of data to a scratchpad memory can be easily done and offer a more energy efficient solution.

An example of partitioning of data variables to scratch-pad memory and DRAM to minimize interference between variables is given in [100]. Several examples of clustered memory architectures have been proposed. In [59] an adaptive scratch-pad memory is successfully used in order to handle the dynamic behavior of multimedia applications. In [119] a clustered memory architecture is employed and an algorithm is developed, which efficiently uses the memory banks to achieve the maximum energy saving while satisfying the given performance constraint. The current approach differentiates by presenting a methodology that combines design-time exploration and a scratchpad memory that is dynamically reconfigurable at run-time.

A network-on-chip approach for a distributed memory system is presented in [114]. The memory architecture is part of a coarse grain reconfigurable architecture (CGRA) and the distributed memories provide data to the processing elements through the network. A network-on-chip is not employed in the current study as it for our applications would introduce unnecessary overhead. Authors in [113] present a framework that extends conventional scratchpad memory to support dynamic application mapping in CGRAs. The dynamic application mapping leads to addressing problems in scratchpad memories, because the data are mapped to a scratchpad memory at compile time. The corresponding addresses are also defined during compile time and there is no run-time mechanism that can search and fetch the data, if there are mapped in a different location on a scratchpad memory. However, dynamic re-mapping of application's data can be useful in some cases. The proposed technique solves the addressing problem and enables the run-time application mapping. The dynamic re-mapping during run-time is beyond the scope of the current work and the typical addressing operation of a scratchpad memory architecture is used.

Chapter 3

Proposed technology platform and design methodology approach

The proposed solution to the problem expressed in Sec.1.3 is discussed in this chapter. The solution approach is split into two parts, namely the platform and the methodology. The first part focuses on the memory architecture in which dynamic applications are executed. The second part focuses on the necessary steps to efficiently couple the applications and the platform.

3.1 Target platform architecture

The platform architecture on which an application is executed is a crucial part of any system scenario methodology. The platform must provide a set of reconfigurable parameters, known as system knobs, which allow different configurations of the platform. The different configurations enable the better adaptation of the platform to the requirements of the application for the specific execution. Many system characteristics can be defined as system knobs and the most common example of such a system knob is the DVFS. DVFS changes the voltage and the frequency of a processor dynamically. A lower frequency and voltage is often sufficient to meet the deadline of an application, while the power consumption of the processor is reduced. Another knob is the supply voltage on the memory system or more specifically the operational mode of a memory bank. A memory bank can have sev-

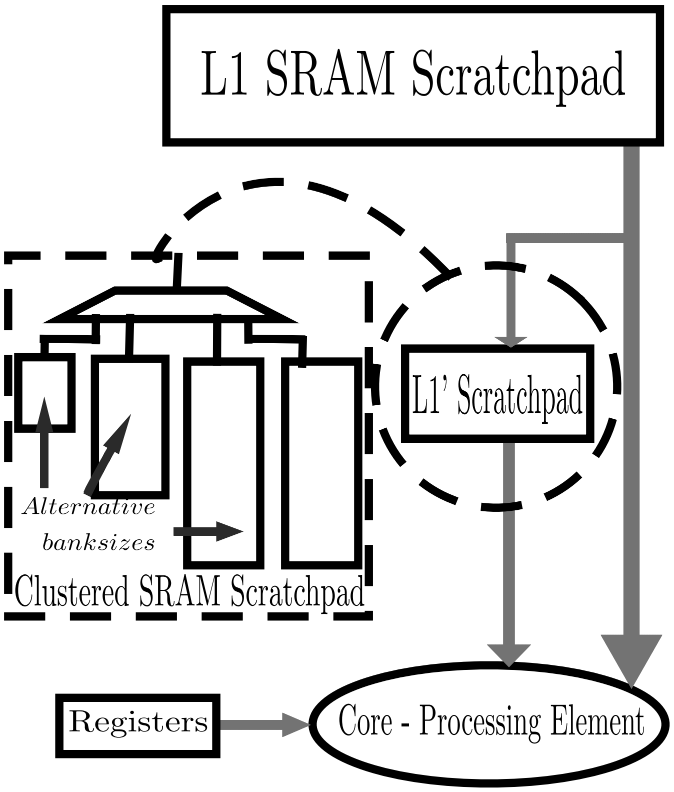


Figure 3.1: Target platform with focus on memory organization.

eral operational modes with different access time and static/dynamic energy consumption characteristics.

A generic reconfigurable architecture template, which is suitable for implementing system scenarios, is shown in Fig. 3.1. This dynamic memory organization consists of two software controlled SRAM scratchpad memories, L1 and L1', and a processing element with its registers. For simplicity it is assumed that all data needed during execution is available in the L1 scratchpad memory without any time penalties even when large background memories are used. This assumption is reasonable for the kind of applications that are generally executed in embedded systems and can, e.g., be achieved through prefetching. Registers are used to save currently used elements, and between the registers and the L1 scratchpad a much smaller L1' scratchpad is introduced. This is a clustered memory that consists of a number of memory banks, which support different operational modes. This work focuses solely on the exploration of the L1' scratchpad memory architecture. The registers and the background memory are only presented in Fig. 3.1 to provide a broader overview of the system architecture.

3.1.1 Target memory platform architecture

A dynamic memory platform is necessary for the memory-aware system scenario methodology. Each configuration of the memory platform corresponds to a different memory energy consumption and available memory space. The dynamic memory platform is achieved by organizing the memory area in a varying number of banks that can be switched between different energy states. The decision to use memory banks with varying sizes on the clustered memory organization increases the reconfiguration options and consequently the potential energy gains. In general, smaller memories are more energy efficient compared to larger memories [109]. However, in some cases large memory banks are needed in order to fit the application data without the need for too many small memories causing complex interconnects. The goal is to use the most energy efficient banks to store the most frequently used data. In order to do that, the DTSE methodology for the analysis and transformation of data provides useful techniques.

A clustered memory organization with a varying number of memory banks of different sizes is explored. In Appendix A such a clustered scratchpad memory with four banks is introduced. It is further expanded to explore memory designs with up to five banks in Appendix B.

3.1.2 Memory models

The memory models that are used to build the clustered memory architecture heavily influence the design decisions. The presented memory model is based on silicon characterization results undertaken by the low-power digital design team at IMEC-nl [32]. For this purpose dedicated memory instances were designed and taped-out in a 40nm low-power process. Measurements covered 50 parts capturing voltage and temperature dependencies of active and passive power as well as limiting operating conditions in terms of speed and minimal data retention voltage. The memory architecture consists of the memory banks and the interconnection between the memory banks, which is necessary to connect the memory to the processor. The most important parameter of the memory models in the current work is the energy profile of the model, although other parameters are also included. The following models are used in this thesis:

- Formula-driven memory model presented in [3]. The memory energy consumption for every access in the clustered scratchpad memory is calculated using a formula, which takes the size, the number of lines and the word length as input parameters. The leakage power is calculated in a similar way. The model supports three different states, namely active, shut-down and retention modes. This model is presented and employed on the work included in Appendix A.

- Commercially available memory macros. For those models delay, access energy and leakage power numbers are derived from a commercial memory compiler. The model supports four different states, namely active, shut-down, light-sleep and deep sleep modes. This model is presented and employed on the work included in Appendix B.

- Experimental standard cell-based memory (SCMEM) [82] . The standard cell-based memories are synthesized using Cadence RTL compiler for TSMC 40nm standard library. Afterwards, place and route of the design is performed and the final IC layout is extracted, which contains the necessary information, such as the capacitance of each component. The layout is used during power simulations, which are carried out using Synopsys PrimeTime, in order to obtain the energy numbers. The model supports four different states, namely active, shut-down, light-sleep and deep sleep modes. This model is presented and employed in the work included in Appendix B and E.

- Interconnection model based on synthesis results. The interconnection model is useful in order to estimate the energy consumption in the peripheral logic, outside the memory banks. This model is developed and presented in the work included in Appendix E.

There are advantages and disadvantages for each of the used models. The analytical model is easy to use, has very low computation time and can calculate energy numbers for any possible size of memory, because the input parameters can have any value. However, the accuracy of the model is lower compared to the other models. The MM model provides accurate energy numbers, but the designer must rely on a library of memory banks. The SCMEM model has also high accuracy, as the energy numbers are based on synthesis and simulation. The designer has the freedom to develop any desirable configuration, although the process of writing, testing and simulating every new memory design is time-consuming. Thus, a library of SCMEM is made and the exploration is limited within the already synthesized models. The differences in the results presented in Appendix A and B is mainly due to the different models and their accuracy. The interconnection model is in both cases omitted, because the contribution of the interconnection to the overall energy consumption is low. The model is based on synthesis, so it is accurate for the specific synthesized design but the generalization to other designs reduces the accuracy.

3.1.3 Technology scaling

The proposed clustered memory architecture is an energy efficient platform for dynamic data-intensive applications. The impact of the technology scaling on the energy efficiency is also explored and presented in Appendix E. The operationally independent memory banks provide an energy efficient platform, but come with an interconnection overhead due to the connections between the memory banks. The energy consumption of the memory banks is reduced due to the scaling that results in smaller memory cells with lower voltage. On the other hand, the energy consumption on the wiring follows a slower reduction curve. Thus, the interconnection energy overhead will increase in the future. A detailed memory and interconnect energy model is developed that includes the scaling impact. Experimental results based on the model suggests that the proposed target memory platform will continue to be energy efficient.

3.2 Data variable based memory-aware system scenario methodology

3.2.1 Methodology Overview

The memory-aware system scenario methodology is based on the observation that the memory subsystem requirements at run-time vary significantly due to dynamic variations of memory needs in the application code. Most existing design methodologies define the memory requirements as that of the most demanding task and tune the system in order to meet its needs

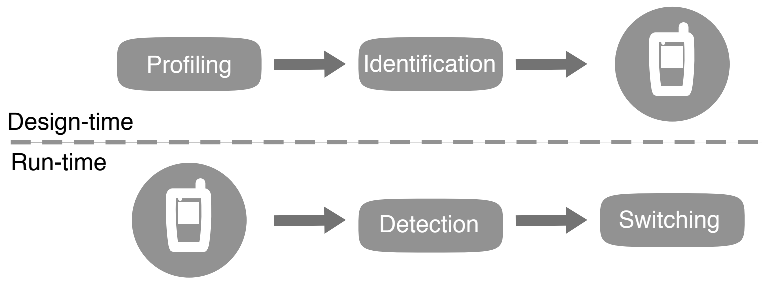


Figure 3.2: System scenario methodology steps. Source: [33]

[75]. Obviously, this approach leads to unused memory area for tasks with lower memory requirements, since those tasks could meet their needs using fewer resources and consequently consuming less energy.

The implementation of the system scenario methodology deals with two main problems:

- The extra overhead introduced by the system scenarios. The usage of system scenarios introduces different types of overheads: from switching between scenarios, from storing code for a set of scenarios instead of a single application instance, from predicting the operation mode, etc.

- The new functionality added to handle the system scenarios at run-time. The usage of system scenarios requires the implementation of extra functionality: deciding which scenario to switch to (or not to switch), using the scenario to change the system configuration, etc.

The decision of what constitutes a scenario has to take into account all these overheads, which leads to a complicated problem. Therefore, the system scenario approach is divided into discrete steps presented in Fig. 3.2. The implementation of new functionality has to exploit all the possible system knobs of the platform. Therefore, a library of reconfigurable memory models is employed in this work.

Designing with system scenarios is workload adaptive and offers different configurations of the platform and the freedom of switching to the most efficient scenario at run-time. A system scenario is a configuration of the system that combines similar RTSs. The system is configured to meet the cost requirements of an RTS by choosing the appropriate system scenario, which is the one that satisfies the requirements using minimal power. The possible RTSs in which the application may run, together with their resource needs should be known and taken into account during design.

3.2.2 Design-time profiling based on data variables

Application profiling is the analysis of the memory requirements for the studied application and is performed at design-time. The analysis focuses on the allocated memory size during execution for a wide range of inputs. The profiling stage consists of running the application code with suitable input data often found in a database, in order to produce profiling results. The profiling reveals parts of the application code with high memory activity and with varying memory access intensity, which possibly depends on input data variables. Because of this behavior, a static study of the application code alone is insufficient since the target applications for this methodology have non-deterministic behavior that is driven by input.

The whole set of possible inputs is studied and analyzed, which is timely possible during design-time. Even a bounded semi-infinite set of input values can be analyzed based on the application's code. For example, assuming that an input value dynamically changes within a range, the corresponding memory requirements are calculated by taking the minimum and maximum limits for that range. The minimum and maximum limits can be found with the help of the application's code. Thus, the profiling in this case should not be perceived as a training series but rather as a full analysis of the whole input space. All possible application inputs that can occur at run-time belong to the input space analyzed at design-time. There are several ways that profiling can be performed. Current program monitoring software cannot offer the needed level of detail. Debuggers and memory tools, such as Valgrind [86], or direct hard-coded profiling are preferable methods, because of their higher accuracy.

Profiling results provided to the designer include complete information about allocated memory size values together with the number of occurrences and duration for each of these memory size values. Moreover, correlation between input data variable values and the resulting memory behavior can possibly be observed. This information is useful for the clustering step that follows. Profiling also reveals the worst case memory usage for a given set of inputs. The memory usage is measured using techniques presented in [66], in which authors compute the minimum amount of memory resources required to store the elements of an application. Appendix B includes a detailed presentation of this step. Appendix C gives an example of how the methodology can be applied to a wireless application.

3.2.3 Design-time system scenario identification based on data variables

The scenario identification is the process of organizing the profiled memory sizes into groups with similar characteristics, which are defined as system scenarios. Grouping or clustering is necessary, because it will be extremely costly to have a different scenario for every possible size, due to the number of memories needed. Clustering neighboring RTSs is a rational choice, because two instances with similar memory needs have similar energy consumption. Independently of the size of the exploration space, i.e. the number of different inputs and RTSs, the number of scenarios is kept low. This is achieved by clustering more RTSs into the same scenario when the number of RTSs is higher. The scenario identification phase always aims at providing a number of scenarios that can be handled by the run-time manager without introducing extreme overheads.

The memory size and the frequency of occurrence of each RTS are not the only two parameters that should be taken into consideration during the system scenario identification. The memory size of each RTS results in a different energy cost depending on the way it is mapped into memory. The impact of the different assignment possibilities is included into the clustering by introduction of energy as a cost metric. The different assignments and their characteristics are explored using the principles of the DTSE methodology. Increasing the number of memory banks results in lower energy per access since the most accessed elements can be assigned to smaller and more energy efficient banks. Unused banks can be switched off. The scenario identification is performed semi-automatically by developing application specific scripts that explore the grouping possibilities for the extracted RTSs.

The system scenario identification step includes the selection of the data variables that determine the active system scenario. This can be achieved by manual inspection of the application code, combined with the application's data input. The variable selection is done before clustering of RTSs into scenarios. For the choice of identification variables, there is a trade-off between the complexity and the accuracy of the scenario detection step. On one hand, if the identification is done using a group of complex variables and their correlation, there is a number of calculations needed in order to predict the active scenario. On the other hand, if the value of a single variable is monitored for scenario identification, the scenario detection is straightforward. Obviously, the accuracy of the scenario detection can be higher in the first case, while the computational needs for scenario detection are lower in the second case. In other words, the more accurate scenario detection, the more resources are used by the run-time manager for detection. In principle, the number of identification variables should be low and in most of the studied applications only a couple of identification variables are enough.

Appendix B includes a detailed presentation of this step. Appendix C gives an example of how the methodology can be applied to a wireless application.

3.2.4 Run-time system scenario detection and switching based on data variables

During run-time, all the switching decisions are taken based on the platform and application information. In this work, we use a simple and straightforward switching approach. Memory models provide the necessary information for the switching decision, namely the energy and the time penalty for switching between stages. The switching step consists of all platform configuration decisions that can be made at run-time, such as the change in the power mode of memory units. Switching takes place when the switching cost is lower than the energy gains achieved by switching. The run-time manager compares the memory energy consumption of executing the next task in the current active system scenario with the energy consumption of execution with the optimal system scenario. If the difference is greater than the switching cost, then scenario switching is performed. Switching costs are defined by the platform and include all memory energy penalties for run-time reconfigurations of the platform, e.g., extra energy needed to change state of a memory unit.

Appendix B includes a detailed presentation of this step. Appendix C gives an example of how the methodology can be applied to a wireless application.

3.2.5 Interleaving exploration based on data variables

Interleaving exploration is a complementary technique applied on the application that can further improve energy gains. Interleaving is a data layout transformation for combining the storage of multiple arrays, so that blocks of data from different arrays are stored contiguously, in order to achieve spatial locality in memory accesses. By interleaving we are able to group the data to be accessed and thus reduce the number of memory accesses for accessing them. The basic principles of the performed interleaving exploration are presented in [106]. Interleaving improves the spatial locality and removes redundant data in order to achieve higher utilization of the hardware resources, especially on single instruction multiple data (SIMD) architectures.

The impact of the data interleaving exploration on the number of memory accesses is significant. When the accesses are irregular and the data are organized in index order, each memory access results in a small amount of useful data due to the presence of holes. In contrast the re-organization of the data provides a sequence of useful data without many holes between them. Thus, a single access to the memory results in a higher number of useful elements. The overall number of memory accesses is reduced, as each access has a higher utilization.

The goal of the interleaving exploration is to select the optimal set of memory banks and to make the optimal decision regarding the mapping of the data to the different memory banks. The parts of the interleaved data that consist mostly of useful elements are mapped into memory banks with low energy per access but at the same time with the necessary access time. The parts of the interleaved data that consist of access holes and rarely accessed elements are optimally mapped into memory banks with energy efficient retention states. In both cases the size of the memory banks should be adequate to fit the stored data but at the same time as small as possible to avoid area and energy penalties.

The interleaving decisions influence the data-to-memory mapping decisions and vice versa. Assuming that the mapping is performed first using the initial data the following interleaving options are reduced. For example, the decision to map two arrays on different memory banks removes the option to interleave them later. Optimizing both the interleaving and the memory mapping at the same time results in a large and inefficient loop of constrain propagation between the two exploration phases. The one way of constraint propagation is from the interleaving exploration to the memory mapping exploration. In more detail, the best interleaving solution, without taking into account the distributed memory organization, would lead only to a local optimum. Applying the local optimum to the memory organization may results in changes due to platform limitations and move to another optimum. The other way of constraint propagation is from the memory mapping exploration to the interleaving exploration. In more detail, the data-to-memory mapping exploration would lead to an optimal solution for the specific memory architecture. Applying that solution to the application data may results in changes due to data dependencies in the application's code. In both cases, new iterations on data interleaving and memory mapping exploration steps are maybe needed. To solve this problem, all the platform parameters are up front defined and taken into consideration for the interleaving exploration, which is performed first. Then, the best data interleaving options are propagated to the data-to-memory mapping step.

Appendix D presents the full interleaving methodology.

Chapter 4

Research Results and Contributions

This thesis is a collection of papers that I have authored or coauthored during my time as a PhD student. Each paper is presented in the appendixes. Rather than including the double-column PDF files, I have opted to reformat each paper to increase readability of the graphs and text. However, I have not altered the text or figures, only the layout and size. The order of the papers presented here is in rough chronological order and corresponds to the different research contributions. In practice, some of the research in these papers have been conducted concurrently.

4.1 Contribution A: Development of the Memory-Aware System Scenario Methodology

The main contribution of this thesis is the development of the system scenario methodology for dynamic data-intensive applications. The methodology provides a systematic way for design-time and run-time handling of the memory subsystem. The work related to the development of the methodology is presented in [29], [28] and [125].

4.1.1 Energy Impact of Memory-Aware System Scenario Approach

Abstract

System scenario methodologies propose the use of different scenarios, e.g., different platform configurations, in order to exploit variations in computational and memory needs during the lifetime of an application. In this paper several extensions are proposed for a system scenario based methodology with a focus on improving memory organization. The conventional methodology targets mostly execution time while this work aims at including memory costs into the exploration. The effectiveness of the proposed extensions is demonstrated and tested using two real applications, which are dynamic and suitable for execution on modern embedded systems. Reductions in memory energy consumption of 40 to 70% is shown.

Main Contributions

This work is a case-study of two applications, which strongly motivates the potential gains from using system scenarios in the memory subsystem. The fact that the chosen applications are from two different domains, supports the effectiveness of the methodology across different domains. The main ideas of the methodology are presented, although the presentation is heavily coupled with the two case-study examples.

Roles of the Authors

I developed the detailed approach, performed the necessary experiments, and wrote the paper, partly based on some initial ideas of Kjeldsberg and Catthoor. Hammari provided one of the applications and Huisken helped in the development of the energy model. I orally presented the paper in the conference. The full paper is presented in Appendix A.

4.1.2 Exploration of energy efficient memory organizations for dynamic multimedia applications using system scenarios

Abstract

We propose a memory-aware system scenario approach that exploits variations in memory needs during the lifetime of an application in order to

optimize energy usage. Different system scenarios capture the application's different resource requirements that change dynamically at run-time. In addition to computational resources, the many possible memory platform configurations and data-to-memory assignments are important system scenario parameters. In this work we focus on clustering of different memory requirements into groups and presenting the system scenario generation in detail. The clustering is a non-trivial problem due to the many different memory requirements, which leads to a very large exploration space. An extended memory model is used as a practical enabler, in order to evaluate the methodology. The memory models include existing state-of-the-art memories, available from industry and academia, and we show how they are employed during the system design exploration phase. Both commercial SRAM and standard cell based memory models are explored in this study. The effectiveness of the proposed methodology is demonstrated and tested using a large set of multimedia benchmarks published in the Polybench, Mibench and Mediabench suites, representative for the domain of multimedia applications. Reduction in energy consumption in the memory subsystem ranges from 35% to 55% for the chosen set of benchmarks.

Main Contributions

This work is a complete and formal presentation of the methodology workflow. The theoretical presentation is accompanied with an extensive number of results for a large set of benchmark applications. The methodology is both extended and better formulated compared to the previous paper.

Roles of the Authors

This work was first presented at a workshop during Embedded Systems Week, as an oral and poster submission. Later, it was extended and published as an invited journal paper. The main idea is based on the previous paper. I performed the necessary simulations and wrote the paper, partly based on suggestions and ideas from Kjeldsberg and Catthoor. The full paper is presented in Appendix B.

4.2 Contribution B: Combined Implementation of the System Scenario Methodology on Memory Subsystem and Processing Elements

The work related to this contribution was published as a conference paper [125].

Abstract

This work explores the power management options for a transmitting wireless system using system scenarios. We exploit the variations in the communication channel and the protocol requirements during the lifetime of a transmission, in order to optimize energy usage. Both the transmission signal power and the memory subsystem are taken into consideration. Different system scenarios and the corresponding configurations capture the different resource requirements, which change dynamically during transmission. Signal power on the antenna and active memory banks are the two main platform parameters explored in this study and sufficiently detailed system models are presented for both. The trade-off between the accuracy of the generated system scenarios and the switching cost between them is analyzed. The exploration is performed for an increasing number of system scenarios, from 1 to 14, and the reported power gains are over 95% and over 25% on the signal power and the memory subsystems respectively.

Main Contributions

This work presents the implementation of the system scenario methodology to a wireless system. The memory-aware methodology developed in this thesis is combined with the general system scenario methodology for processing elements. The target application is a widely used and recent benchmark. The exploration includes all the different subsystems and an interesting analysis of the overhead on the scenario generation is presented. The use of system scenarios in the whole system improves the overall energy efficiency.

Roles of the Authors

My contribution to this work is the experimental and the writing part regarding the memory subsystem. Zompakis did all the experiments and the writing regarding the processing subsystem and the wireless application. The oral presentation was also performed by Zompakis. The rest of the authors contributed to the initial ideas and the improvement of the text.

The full paper is presented in Appendix C.

4.3 Contribution C: Integrated Interleaving and Data-to-Memory Mapping

The work related to this contribution is submitted as a journal paper and is currently in its second round of review.

Abstract

This work presents a methodology for efficient exploration of data interleaving and data-to-memory mapping options for SIMD (Single Instruction Multiple Data) platform architectures. The system architecture consists of a reconfigurable clustered scratch-pad memory and a SIMD functional unit, which performs the same operation on multiple input data in parallel. The memory accesses contribute substantially to the overall energy consumption of an embedded system executing a data intensive task. The scope of this work is the reduction of the overall energy consumption by increasing the utilization of the functional units and decreasing the number of memory accesses. The presented methodology is tested using a number of benchmark applications with irregularities in their access scheme. Potential gains are calculated based on the energy models both for the processing and the memory part of the system. The reduction in energy consumption after efficient interleaving and mapping of data is between 40% and 80% for the complete system and the studied benchmarks.

Main Contributions

This work is also a combination of the developed methodology and another complementary approach. The developed methodology is compatible with code transformation techniques and this work focus on the data interleaving. The integration of the two required significant modification and integration work and the final work-flow was tested using a number of benchmarks. This work shows that the memory-aware system scenario methodology is complementary to other techniques that improve energy efficiency.

Roles of the Authors

The initial idea for this work is partly based on suggestions by Catthoor. The necessary experiments were implemented by me with the help of Sharma. The majority of the paper was written by me and the rest by Sharma, while all the authors contributed on the further improvement of the text with their feedback. The full paper is presented in Appendix D.

4.4 Contribution D: Interconnection Cost Modeling and Scaling

The work related to this contribution is presented as a technical report.

Abstract

Power consumption is a key limiting factor in modern embedded devices. The memory architecture contributes significantly to the overall power consumption of the system. Among many proposed techniques, one effective system design approach to reduce the memory power needs is the design of a dynamically reconfigurable clustered memory architecture. The operationally independent memory banks provide an energy efficient platform, but come with an interconnection overhead due to the connections between the memory banks. Thus, there is a trade-off between the energy gains by increasing the number of memory banks and the increase in interconnection overhead. This work explores the future development of the interconnection overhead, as the interconnection cost is expected to increase while the process technology shrinks to 5nm. The current study employs both CAD tools with simulation results using the current technology and projections provided by institutions. We use predictive technology models supported by information from ITRS and IMEC's interconnect technologists. A model is developed that provide a sufficiently accurate estimate of the interconnection cost overhead for clustered memory architectures consisting of two to five memory banks in technologies ranging from 40nm to 5nm. The model shows that the interconnect overhead is as low as 10.9 % even for the most aggressive technologies. Hence a dynamically reconfigurable clustered memory architecture is a viable solution also for future designs, even if the optimal number of memory banks may be reduces as shown in an experiment with a representative real life application.

Main Contributions

This work investigates the effectiveness of the memory-aware system scenario methodology in the future. A reconfigurable memory architecture is a requirement for the implementation of the methodology, thus it is important to study the development of similar architectures in the future.

Roles of the Authors

I performed the necessary development work and wrote the technical report, partly based on suggestions from Kjeldsberg and Catthoor. The full technical report is presented in Appendix E.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The main scope of this dissertation is to develop a system scenario methodology that focuses on the memory organization of embedded applications and platforms. The developed methodology exploits memory requirement variations and achieves significant reductions in memory energy consumption, which is of great importance in embedded devices. The memory-aware system scenario methodology is suited for applications that experience dynamic behavior with respect to memory organization utilization during their execution. A wide range of application domains, including multimedia, wireless and bio-medical applications, are tested to prove the effectiveness of the methodology and energy reductions of up to 60% have been demonstrated.

An extensive memory energy model is developed in order to have sufficiently accurate simulations and results. A library is built based on commercial and experimental state-of-the-art memory models. The library is employed for the construction of reconfigurable memory architectures, which are suitable for the implementation of system scenarios. Another model is developed to study the impact of the interconnect on the overall energy. The model suggests that overhead will be kept low in the short term and will increase within reasonable levels in the mid-long term. Therefore, the design of energy efficient clustered memory architecture will continue to be a good design choice.

The testing of the methodology on two applications from bioengineering and wireless communications domains justifies its effectiveness in reduction of memory energy consumption. The memory-aware system scenario methodology is also tested using a wide range of multimedia applications, which allow us to draw conclusions about different kinds of dynamic behavior and their effect on the energy gains achieved using the methodology. The results demonstrate the effectiveness of the methodology reducing the memory energy consumption with between 35% and 55%. Since memory size requirements are still met in all situations, performance is not reduced. The presented results show that the memory-aware system scenario methodology is suited for applications that experience dynamic behaviour with respect to memory organisation utilization during their execution.

Apart from justifying the effectiveness of the system scenarios methodology on the memory, this work explores the compatibility of the methodology with other techniques. Firstly, the application of the system scenarios methodology in the complete system, including the memory and the processing subsystem, is studied. The power management options for a wireless system using system scenarios is explored. The dynamic parameters and the variations during the transmission for the targeted wireless protocol are analyzed. Based on the analysis and the system models, system scenarios are generated for the case study, in order to optimize energy usage. Both the signal power and the memory subsystem are taken into consideration. The results demonstrate the effectiveness of the methodology and illustrate the interesting trade-off between the clustering overhead and the switching cost.

Secondly, the proposed methodology is integrated with code and data transformation techniques. This work presents a methodology for efficient exploration of data interleaving and data-to-memory mapping options for SIMD (Single Instruction Multiple Data) platform architectures. The methodology focuses on reducing the overall energy consumption by reducing the number of memory accesses and the energy per access. A wide range of applications is studied that allow us to draw conclusions about different kinds of dynamic behavior and their effect on the energy gains achieved using the methodology. The improvement in the total system energy consumption after efficient interleaving and mapping of data is between 40% and 80% for the studied benchmarks having the type of irregularities in their access scheme that benefit most from the methodology.

The development of an interconnection model for the proposed memory architecture suggests that the interconnection overhead will be kept low in the short term and will increase within reasonable levels in the mid-long term. Therefore, this thesis concludes that the design of energy efficient clustered memory architecture will continue to be a good design choice.

5.2 Future Work

Future research to improve the current work can focus on the system scenario methodology and/or the memory models and architectures.

One improvement is to fully automate the memory-aware system scenario methodology. The optimal implementation should take as an input the application code and the library of memory models and automatically generate the most energy efficient memory architecture. In the current work several scripts were developed to speed up the design space exploration, but the work-flow is not fully automated. The improvement of the prediction and the identification phase of the methodology could be another significant contribution. A multidimensional scenario clustering for the whole system is an interesting improvement. The N-dimensional exploration space will include several parameters, such as memory energy, processing element energy, execution time and reliability, and new methods should be developed to handle the explosion of the exploration space.

The system scenarios currently focus on analyzing one specific application. A future extension would be to study a number of concurrent task executed on a given embedded systems. In that case the system scenarios can overlap between applications, i.e. one scenario may group RTSs from different applications. However, the system requirements of the different applications should be comparable for the methodology to be applicable. Another research direction could be the exploration of the effects that an update version of the studied application has on the system. The system scenario methodology uses a hardware/software co-design approach, so a difference in the application code may have a significant impact on the effectiveness of the designed memory platform. A possible solution could be the addition of a margin to each system scenario by the designer.

The accuracy of the energy models can be significantly increased. More detailed models can be developed based on extensive research and simulation. All the possible clustered memory architectures can be synthesized and tested to get a detailed report on the energy numbers, although it is a very time consuming task. Especially the interconnect model can become more detailed and accurate by taking delay into consideration. The synthesis of more memory designs and configurations can also improve the model. When newer technologies become available in the CAD tools, the designs can be re-synthesized and the model can be calibrated. A manually improved place and route strategy will provide more accurate results regarding the overhead for clustered memory architectures.

An architecture with point-to-point connections between all or some of the memory banks could be explored. Such an architecture allows the direct transfer of data between the memory banks without the intervention of the processor. The cost of moving data between banks should be modeled in order to take thisintocaccount as part of the exploration trade-off. The data transfers between the memory banks may be useful in some cases, although the interconnection overhead is expected to increase significantly.

Bibliography

[1] Santosh G Abraham and Scott A Mahlke. Automatic and efficient evaluation of memory hierarchies for embedded systems. In *Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on*, pages 114–125. IEEE, 1999.

[2] Berkin Akin, Franz Franchetti, and James C Hoe. Data reorganization in memory using 3d-stacked dram. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 131–143. ACM, 2015.

[3] Antonio Artes, Jose L Ayala, Ashoka Visweswara Sathanur, Jos Huisken, and Francky Catthoor. Run-time self-tuning banked loop buffer architecture for power optimization of dynamic workload applications. In *VLSI and System-on-Chip (VLSI-SoC), 2011 IEEE/IFIP 19th International Conference on*, pages 136–141. IEEE, 2011.

[4] Prithviraj Banerjee, John Chandy, Manish Gupta, EW Hodges, John G Holm, Antonio Lain, Daniel J Palermo, Shankar Ramaswamy, Ernesto Su, et al. The paradigm compiler for distributed-memory multicomputers. *Computer*, 28(10):37–47, 1995.

[5] Utpal Banerjee, Rudolf Eigenmann, Alexandru Nicolau, David A Padua, et al. Automatic program parallelization. *Proceedings of the IEEE*, 81(2):211–243, 1993.

[6] Gaurav Bansal, Ziaul Hasan, Md Jahangir Hossain, and Vijay K Bhargava. Subcarrier and power adaptation for multiuser ofdm-based cognitive radio systems. In *Communications (NCC), 2010 National Conference on*, pages 1–5. IEEE, 2010.

[7] Gaurav Bansal, Md Jahangir Hossain, and Vijay K Bhargava. Optimal and suboptimal power allocation schemes for ofdm-based cog-

nitive radio systems. *Wireless Communications, IEEE Transactions on*, 7(11):4710–4718, 2008.

[8] Andrew Bateman. *Digital communications: design for the real world.* Addison-Wesley, 1999.

[9] Luca Benini, Alberto Macii, Enrico Macii, and Massimo Poncino. Increasing energy efficiency of embedded systems by application-specific memory hierarchy generation. *IEEE Design & Test of Computers*, 1(2):74–85, 2000.

[10] Luca Benini, Alberto Macii, and Massimo Poncino. A recursive algorithm for low-power memory partitioning. In *Low Power Electronics and Design, 2000. ISLPED'00. Proceedings of the 2000 International Symposium on*, pages 78–83. IEEE, 2000.

[11] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[12] E Bugnion, Shih-Wei Liao, BR Murphy, SP Amarasinghe, JM Anderson, MW Hall, and Monica S Lam. Maximizing multiprocessor performance with the suif compiler. *Computer*, 4(12):84–85, 1996.

[13] RTL Cadence. Compiler users manual, 2014.

[14] Francky Catthoor, Sven Wuytack, G.E. de Greef, Florin Banica, Lode Nachtergaele, and Arnout Vandecappelle. *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design.* Springer, 1998.

[15] Nanda Kishore Chavali and Venkata Krishna Reddy Pilli. Adaptive time synchronization for vht wireless lan. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, pages 312–317. ACM, 2012.

[16] Shuai Che, Jeremy W Sheaffer, and Kevin Skadron. Dymaxion: optimizing memory access patterns for heterogeneous systems. In *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*, page 13. ACM, 2011.

[17] Fei Chen and Edwin Hsing-Mean Sha. Loop scheduling and partitions for hiding memory latencies. In *Proceedings of the 12th international symposium on System synthesis*, page 64. IEEE Computer Society, 1999.

[18] James Hsueh-Chung Chen, Theodorus E Standaert, Emre Alptekin, Terry Spooner, Vamsi Paruchuri, et al. Interconnect performance and scaling strategy at 7 nm node. In *Interconnect Technology Conference/Advanced Metallization Conference (IITC/AMC), 2014 IEEE International*, pages 93–96. IEEE, 2014.

[19] Wei Chen, Pingyi Fan, and Zhigang Cao. Water filling in cellar: the optimal power allocation policy with channel and buffer state information. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, volume 1, pages 537–541. IEEE, 2005.

[20] Eric Cheung, Harry Hsieh, and Felice Balarin. Memory subsystem simulation in software tlm/t models. In *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*, pages 811–816. IEEE, 2009.

[21] Doosan Cho, Ilya Issenin, Nikil Dutt, Jonghee W Yoon, and Yunheung Paek. Software controlled memory layout reorganization for irregular array access patterns. In *Proceedings of the 2007 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 179–188. ACM, 2007.

[22] Kihwan Choi, Karthik Dantu, Wei-Chung Cheng, and Massoud Pedram. Frame-based dynamic voltage and frequency scaling for a mpeg decoder. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 732–737. ACM, 2002.

[23] Eui-Young Chung, Giovanni De Micheli, and Luca Benini. Contents provider-assisted dynamic voltage scaling for low energy multimedia applications. In *Proceedings of the 2002 international symposium on Low power electronics and design*, ISLPED '02, pages 42–47, 2002.

[24] International Roadmap Committee et al. International technology roadmap for semiconductors: 2013 edition executive summary. *Semiconductor Industry Association, San Francisco, CA, available at: http://www. itrs. net/Links/2013ITRS/2013Chapters/2013ExecutiveSummary. pdf*, 2013.

[25] International Roadmap Committee et al. Process integration, devices, and structures (pids). *Semiconductor Industry Association, San Francisco, CA*, 2013.

[26] Xavier Costa-Pérez, Andreas Festag, Hans-Joerg Kolbe, Juergen Quittek, Stefan Schmid, Martin Stiemerling, Joerg Swetina, and Hans Van Der Veen. Latest trends in telecommunication standards. *ACM SIGCOMM Computer Communication Review*, 43(2):64–71, 2013.

[27] Bruce Powel Douglass. *Real time UML: advances in the UML for real-time systems*. Addison-Wesley Professional, 2004.

[28] Iason Filippopoulos, Francky Catthoor, and Per Gunnar Kjeldsberg. Exploration of energy efficient memory organisations for dynamic multimedia applications using system scenarios. *Design Automation for Embedded Systems*, pages 1–24, 2013.

[29] Iason Filippopoulos, Francky Catthoor, Per Gunnar Kjeldsberg, Elena Hammari, and Jos Huisken. Memory-aware system scenario approach energy impact. In *NORCHIP, 2012*, pages 1 –6, nov. 2012.

[30] Daniel D Gajski, Frank Vahid, Sanjiv Narayan, and Jie Gong. *Specification and design of embedded systems*. PTR Prentice Hall Englewood Cliffs, New Jesey, USA, 1994.

[31] Philip Garcia, Katherine Compton, Michael Schulte, Emily Blem, and Wenyin Fu. An overview of reconfigurable hardware in embedded systems. *EURASIP J. Embedded Syst.*, 2006(1):13–13, January 2006.

[32] Tobias Gemmeke, Mohamed M. Sabry, Jan Stuijt, Pieter Schuddinck, Praveen Raghavan, and Francky Catthoor. *Near Threshold Computing: Technology, Methods and Applications*, chapter Memories for NTC, pages 75–100. Springer International Publishing, Cham, 2016.

[33] Stefan Valentin Gheorghita. *Dealing with dynamism in embedded system design: application scenarios*. PhD thesis, Technische Universiteit Eindhoven, 2007.

[34] Stefan Valentin Gheorghita. *Dealing with dynamism in embedded system design: application scenarios*. PhD thesis, Technische Universiteit Eindhoven, 2007.

[35] Stefan Valentin Gheorghita, Twan Basten, and Henk Corporaal. Intra-task scenario-aware voltage scheduling. In *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 177–184. ACM, 2005.

[36] Stefan Valentin Gheorghita, Twan Basten, and Henk Corporaal. Application scenarios in streaming-oriented embedded system design. In *System-on-Chip, 2006. International Symposium on*, pages 1–4. IEEE, 2006.

[37] Stefan Valentin Gheorghita, Martin Palkovic, Juan Hamers, Arnout Vandecappelle, Stelios Mamagkakis, Twan Basten, Lieven Eeckhout, Henk Corporaal, Francky Catthoor, and Frederik Vandeputte. System-scenario-based design of dynamic embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 14(1):3, 2009.

[38] Stefan Valentin Gheorghita, Martin Palkovic, Juan Hamers, Arnout Vandecappelle, Stelios Mamagkakis, Twan Basten, Lieven Eeckhout, Henk Corporaal, Francky Catthoor, Frederik Vandeputte, et al. System-scenario-based design of dynamic embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 14(1):3, 2009.

[39] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *Solid-State Circuits, IEEE Journal of*, 31(9):1277 –1284, sep 1996.

[40] Peter Grun, Nikil Dutt, and Alex Nicolau. Mist: An algorithm for memory miss traffic management. In *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, pages 431–438. IEEE Press, 2000.

[41] Yibo Guo, Qingfeng Zhuge, Jingtong Hu, Juan Yi, Meikang Qiu, and Edwin HM Sha. Data placement and duplication for embedded multicore systems with scratch pad memory. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(6):809–817, 2013.

[42] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14. IEEE, 2001.

[43] Javier J Gutierrez, Maria J Escalona, Manuel Mejias, Jesus Torres, and Arturo H Centeno. A case study for generating test cases from use cases. In *Research Challenges in Information Science, 2008. RCIS 2008. Second International Conference on*, pages 209–214. IEEE, 2008.

[44] Mary W Hall, Timothy J Harvey, Ken Kennedy, Nathaniel McIntosh, Kathryn S McKinley, Jeffrey D Oldham, Michael H Paleczny, and Gerald Roth. *Experiences using the ParaScope Editor: an interactive parallel programming tool*, volume 28. ACM, 1993.

[45] Juan Hamers and Lieven Eeckhout. Scenario-based resource prediction for qos-aware media processing. *Computer*, (10):56–63, 2010.

[46] E. Hammari, F. Catthoor, J. Huisken, and P G Kjeldsberg. Application of medium-grain multiprocessor mapping methodology to epileptic seizure predictor. In *NORCHIP, 2010*, pages 1 –6, nov. 2010.

[47] Elena Hammari, Francky Catthoor, Per Gunnar Kjeldsberg, Jos Huisken, K Tsakalis, and L Iasemidis. Identifying data-dependent system scenarios in a dynamic embedded system. In *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.

[48] Stefaan Himpe, Francky Catthoor, G De Coninck, and J Van Meerbergen. Mtg and grey-box: modeling dynamic multimedia applications with concurrency and non-determinism. 2002.

[49] J. Hulzink, M. Konijnenburg, M. Ashouei, A. Breeschoten, T. Berset, J. Huisken, J. Stuyt, H. de Groot, F. Barat, J. David, et al. An ultra low energy biomedical signal processing system operating at near-threshold. *Biomedical Circuits and Systems, IEEE Transactions on*, 5(6):546–554, 2011.

[50] L.D. Iasemidis et al. Long-term prospective on-line real-time seizure prediction. *Clinical Neurophysiology*, 116(3):532–544, 2005.

[51] IEEE. Specification frame work for ac: Ieee 802.11-09/0992r21, 2011.

[52] Mugurel Theodor Ionita. *Scenario-based system architecting: a systematic approach to developing future-proof system architectures.* PhD thesis, Technische Universiteit Eindhoven, 2005.

[53] Yuriko Ishitobi, Tohru Ishihara, and Hiroto Yasuura. Code placement for reducing the energy consumption of embedded processors with scratchpad and cache memories. In *Embedded Systems for Real-Time Multimedia, 2007. ESTIMedia 2007. IEEE/ACM/IFIP Workshop on*, pages 13–18. IEEE, 2007.

[54] Robert SH Istepanian, Emil Jovanov, and YT Zhang. Guest editorial introduction to the special section on m-health: Beyond seamless mobility and global wireless health-care connectivity. *Information Technology in Biomedicine, IEEE Transactions on*, 8(4):405–414, 2004.

[55] Bruce L Jacob, Peter M Chen, Seth R Silverman, and Trevor N Mudge. An analytical model for designing memory hierarchies. *Computers, IEEE Transactions on*, 45(10):1180–1194, 1996.

[56] Jiho Jang and Kwang Bok Lee. Transmit power adaptation for multiuser ofdm systems. *Selected Areas in Communications, IEEE Journal on*, 21(2):171–178, 2003.

[57] Jiho Jang, Kwang Bok Lee, and Yong-Hwan Lee. Frequency-time domain transmit power adaptation for a multicarrier system in fading channels. In *Personal, Indoor and Mobile Radio Communications, 2001 12th IEEE International Symposium on*, volume 1, pages D–100. IEEE, 2001.

[58] Axel Jantsch, Peeter Ellervee, Ahmed Hemani, Johnny Öberg, and Hannu Tenhunen. Hardware/software partitioning and minimizing memory interface traffic. In *Proceedings of the conference on European design automation*, pages 226–231. IEEE Computer Society Press, 1994.

[59] Dah-Jing Jwo, Chien-Hao Tseng, Mu-Yen Chen, and Ta-Shun Cho. *Adaptive and nonlinear kalman filtering for GPS navigation processing*. INTECH Open Access Publisher, 2009.

[60] Wang Kai and Xu Zhiwei. Synopsys prime power manual release u-2003.06-qa, 2003.

[61] Mahmut Kandemir, Ugur Sezer, and Victor Delaluz. Improving memory energy using access pattern classification. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 201–206. IEEE Press, 2001.

[62] Mahmut Kandemir, Narayanan Vijaykrishnan, Mary Jane Irwin, and Wu Ye. Influence of compiler optimizations on system power. In *Proceedings of the 37th Annual Design Automation Conference*, pages 304–307. ACM, 2000.

[63] Kyungsu Kang, Luca Benini, and Giovanni De Micheli. A high-throughput and low-latency interconnection network for multi-core clusters with 3-d stacked l2 tightly-coupled data memory. In *VLSI and System-on-Chip (VLSI-SoC), 2012 IEEE/IFIP 20th International Conference on*, pages 283–286. IEEE, 2012.

[64] Xin Kang, Ying-Chang Liang, Arumugam Nallanathan, Krishna Garg, and Rui Zhang. Optimal power allocation for fading channels in cognitive radio networks: Ergodic capacity and outage capacity. *Wireless Communications, IEEE Transactions on*, 8(2):940–950, 2009.

[65] A Kritikakou, F Catthoor, V Kelefouras, and C Goutis. A scalable and near-optimal representation for storage size management. *ACM transaction architecture and code optimization*, 11(1):1–25, 2014.

[66] Angeliki Kritikakou, Francky Catthoor, and Costas Goutis. Intra-signal in-place methodology for non-overlapping scenario. In *Scalable and Near-Optimal Design Space Exploration for Embedded Systems*, pages 97–123. Springer, 2014.

[67] Angeliki Stavros Kritikakou. *Development of methodologies for memory management and design space exploration of SW/HW computer architectures for designing embedded systems.* PhD thesis, Department of Electrical and Computer Engineering School of Engineering, University of Patras, 2013.

[68] Rakesh Kumar, Victor Zyuban, and Dean M Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *Computer Architecture, 2005. ISCA'05. Proceedings. 32nd International Symposium on*, pages 408–419. IEEE, 2005.

[69] C. Lee, M. Potkonjak, and W.H. Mangione-Smith. Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pages 330–335. IEEE Computer Society, 1997.

[70] Jongeun Lee, Kiyoung Choi, and Nikil D Dutt. Compilation approach for coarse-grained reconfigurable architectures. 2003.

[71] Yanbing Li and Wayne H Wolf. Hardware/software co-synthesis with memory hierarchies. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(10):1405–1417, 1999.

[72] Chien-Ching Lin, Yen-Hsu Shih, Hsie-Chia Chang, and Chen-Yi Lee. Design of a power-reduction viterbi decoder for wlan applications. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 52(6):1148–1156, 2005.

[73] Yuan Lin, Hyunseok Lee, Mark Woh, Yoav Harel, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti, and Krisztian Flautner. Soda: A low-power architecture for software radio. In *ACM SIGARCH Computer Architecture News*, volume 34, pages 89–101. IEEE Computer Society, 2006.

[74] Dake Liu and Christer Svensson. Power consumption estimation in cmos vlsi chips. *Solid-State Circuits, IEEE Journal of*, 29(6):663–670, 1994.

[75] Zhe Ma, Pol Marchal, Daniele Paolo Scarpazza, Peng Yang, Chun Wong, José Ignacio Gómez, Stefaan Himpe, Chantal Ykman-Couvreur, and Francky Catthoor. *Systematic methodology for real-time cost-effective mapping of dynamic concurrent task-based systems on heterogenous platforms.* Springer Science & Business Media, 2007.

[76] A. Macii, L. Benini, and M. Poncino. *Memory Design Techniques for Low-Energy Embedded Systems.* Kluwer Academic Publishers, 2002.

[77] Afzal Malik, Bill Moyer, and Dan Cermak. A low power unified cache architecture providing power and performance flexibility. In *Low Power Electronics and Design, 2000. ISLPED'00. Proceedings of the 2000 International Symposium on*, pages 241–243. IEEE, 2000.

[78] P. Marchal, D. Bruni, J.I. Gomez, L. Benini, L. Pinuel, F. Catthoor, and H. Corporaal. Sdram-energy-aware memory allocation for dynamic multi-media applications on multi-processor platforms. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 516–521, 2003.

[79] Bingfeng Mei, Serge Vernalde, Diederik Verkest, Hugo De Man, and Rudy Lauwereins. Dresc: A retargetable compiler for coarse-grained reconfigurable architectures. In *Field-Programmable Technology, 2002.(FPT). Proceedings. 2002 IEEE International Conference on*, pages 166–173. IEEE, 2002.

[80] P Meinerzhagen, C Roth, and A Burg. Towards generic low-power area-efficient standard cell based memory architectures. In *Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium on*, pages 129–132. IEEE, 2010.

[81] P Meinerzhagen, C Roth, and A Burg. Towards generic low-power area-efficient standard cell based memory architectures. In *Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium on*, pages 129–132. IEEE, 2010.

[82] Pascal Meinerzhagen, SM Yasser Sherazi, Andreas Burg, and Joachim Neves Rodrigues. Benchmarking of standard-cell based memories in the sub-vt domain in 65-nm cmos technology. *IEEE Transactions on Emerging and Selected Topics in Circuits and Systems*, 1(2), 2011.

[83] Narasinga Rao Miniskar. *System Scenario Based Resource Management of Processing Elements on MPSoC*. PhD thesis, Katholieke Universiteit Leuven, 2012.

[84] Sparsh Mittal. A survey of architectural techniques for improving cache power efficiency. *Sustainable Computing: Informatics and Systems*, 4(1):33–43, 2014.

[85] Naveen Muralimanohar and Rajeev Balasubramonian. Interconnect design considerations for large nuca caches. *ACM SIGARCH Computer Architecture News*, 35(2):369–380, 2007.

[86] Nicholas Nethercote and Julian Seward. How to shadow every byte of memory used by a program. In *Proceedings of the Third International ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, 2007.

[87] Yoichi Oshima, Bing J Sheu, and Steve H Jen. High-speed memory architectures for multimedia applications. *Circuits and Devices Magazine, IEEE*, 13(1):8–13, 1997.

[88] M. Palkovic, H. Corporaal, and F. Catthoor. Heuristics for scenario creation to enable general loop transformations. In *System-on-Chip, 2007 International Symposium on*, pages 1 –4, nov. 2007.

[89] Martin Palkovic. Enhanced applicability of loop transformations. *Dissertation Abstracts International*, 68(04), 2007.

[90] Martin Palkovic, Francky Catthoor, and Henk Corporaal. In *Proc. of the 4th Workshop on Optimizations for DSP and Embedded Systems*, pages 21–30. IEEE and ACM SIGMICRO, 2006.

[91] Martin Palkovic et al. Systematic preprocessing of data dependent constructs for embedded systems. *Journal of Low Power Electronics, Volume 2, Number 1*, 2006.

[92] Chenyun Pan and Azad Naeemi. System-level variation analysis for interconnection networks. In *Interconnect Technology Conference/Advanced Metallization Conference (IITC/AMC), 2014 IEEE International*, pages 303–306. IEEE, 2014.

[93] Preeti Ranjan Panda, Francky Catthoor, Nikil D Dutt, Koen Danckaert, Erik Brockmeyer, Chidamber Kulkarni, A Vandercappelle, and Per Gunnar Kjeldsberg. Data and memory optimization techniques for embedded systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 6(2):149–206, 2001.

[94] Sung Park, Andreas Savvides, and Mani Srivastava. Battery capacity measurement and analysis using lithium coin cell battery. In *Proceedings of the 2001 international symposium on Low power electronics and design*, pages 382–387. ACM, 2001.

[95] NL Passes, Edwin HM Sha, and Liang-Fang Chao. Multi-dimensional interleaving for time-and-memory design optimization. In *Computer Design: VLSI in Computers and Processors, 1995. ICCD'95. Proceedings., 1995 IEEE International Conference on*, pages 440–445. IEEE, 1995.

[96] D.A. Patterson and J.L. Hennessy. *Exploiting Memory Hierarchy in Computer Organization and Design the HW/SW Intelface.* Morgan Kaufmann, 1994.

[97] JoAnn M Paul, Donald E Thomas, and Alex Bobrek. Scenario-oriented design for single-chip heterogeneous multiprocessors. *Very*

Large Scale Integration (VLSI) Systems, IEEE Transactions on, 14(8):868–880, 2006.

[98] A. Portero et al. Dynamic voltage scaling for power efficient mpeg4-sp implementation. In *Application-specific Systems, Architectures and Processors, 2006. ASAP '06. International Conference on*, pages 257 –260, sept. 2006.

[99] L.N. Pouchet. Polybench: The polyhedral benchmark suite.

[100] John G Proakis. *Intersymbol Interference in Digital Communication Systems*. Wiley Online Library, 2001.

[101] Abbas Rahimi, Igor Loi, Mohammad Reza Kakoee, and Luca Benini. A fully-synthesizable single-cycle interconnection network for shared-l1 processor clusters. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[102] Herbert John Ryser. *Combinatorial mathematics*. MAA Washington, 1963.

[103] Alberto Sangiovanni-Vincentelli and Marco Di Natale. Embedded system design for automotive applications. *Computer*, 4(10):42–51, 2007.

[104] Ruchira Sasanka, Christopher J Hughes, and Sarita V Adve. Joint local and global hardware adaptations for energy. *ACM SIGARCH Computer Architecture News*, 30(5):144–155, 2002.

[105] Herman Schmit and Donald E Thomas. Synthesis of application-specific memory designs. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 5(1):101–111, 1997.

[106] Namita Sharma, TV Aa, Prashant Agrawal, Praveen Raghavan, Preeti Ranjan Panda, and Francky Catthoor. Data memory optimization in lte downlink. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 2610–2614. IEEE, 2013.

[107] Namita Sharma, Preeti Ranjan Panda, Francky Catthoor, Praveen Raghavan, and Tom Vander Aa. Array interleaving an energy efficient data layout transformation. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 20(3):44, 2015.

[108] Tajana Šimunić, Luca Benini, and Giovanni De Micheli. Cycle-accurate simulation of energy consumption in embedded systems. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 867–872. ACM, 1999.

[109] Stefan Steinke, Lars Wehmeyer, Bo-Sik Lee, and Peter Marwedel. Assigning program and data objects to scratchpad for energy reduction. In *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pages 409–415. IEEE, 2002.

[110] Vivy Suhendra, Chandrashekar Raghavan, and Tulika Mitra. Integrated scratchpad memory optimization and task scheduling for mpsoc architectures. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 401–410. ACM, 2006.

[111] I-Jui Sung, Geng Daniel Liu, and Wen-Mei W Hwu. Dl: A data layout transformation system for heterogeneous computing. In *Innovative Parallel Computing (InPar), 2012*, pages 1–11. IEEE, 2012.

[112] Sriram Swaminathan et al. A dynamically reconfigurable adaptive viterbi decoder. In *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, FPGA '02, pages 227–236, New York, NY, USA, 2002. ACM.

[113] Muhammad Adeel Tajammul, SMA Jafri, Peeter Ellerve, Ahmed Hemani, Hannu Tenhunen, and Juha Plosila. Dymep: An infrastructure to support dynamic memory binding for runtime mapping in cgras. In *VLSI Design (VLSID), 2015 28th International Conference on*, pages 547–552. IEEE, 2015.

[114] Muhammad Adeel Tajammul, Muhammad Ali Shami, Ahmed Hemani, and Sridharan Moorthi. Noc based distributed partitionable memory system for a coarse grain reconfigurable architecture. In *VLSI Design (VLSI Design), 2011 24th International Conference on*, pages 232–237. IEEE, 2011.

[115] Shyamkumar Thoziyoor, Jung Ho Ahn, Matteo Monchiero, Jay B Brockman, and Norman P Jouppi. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, pages 51–62. IEEE, 2008.

[116] Tore Ulversøy. Software defined radio: Challenges and opportunities. *Communications Surveys & Tutorials, IEEE*, 12(4):531–550, 2010.

[117] Ingrid Verbauwhede, Francky Catthoor, Joos Vandewalle, and Hugo De Man. In-place memory management of algebraic algorithms on application specific ics. *Journal of VLSI signal processing systems for signal, image and video technology*, 3(3):193–200, 1991.

[118] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260 –269, april 1967.

[119] Zhong Wang and Xiaobo Sharon Hu. Energy-aware variable partitioning and instruction scheduling for multibank memory architectures. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 10(2):369–388, 2005.

[120] Michael Wolfe. Data dependence and program restructuring. *The Journal of Supercomputing*, 4(4):321–344, 1991.

[121] Cheong Yui Wong, Roger S Cheng, K Ben Lataief, and Ross D Murch. Multiuser ofdm with adaptive subcarrier, bit, and power allocation. *Selected Areas in Communications, IEEE Journal on*, 17(10):1747–1758, 1999.

[122] Shyh-Chyi Wong, Gwo-Yann Lee, and Dye-Jyun Ma. Modeling of interconnect capacitance, delay, and crosstalk in vlsi. *Semiconductor Manufacturing, IEEE Transactions on*, 13(1):108–111, 2000.

[123] Peng Yang, Paul Marchal, Chun Wong, Stefaan Himpe, Francky Catthoor, Patrick David, Johan Vounckx, and Rudy Lauwereins. Managing dynamic concurrent tasks in embedded real-time multimedia systems. In *Proceedings of the 15th international symposium on System Synthesis*, pages 112–119. ACM, 2002.

[124] Ch Ykman-Couvreur, Vincent Nollet, Fr Catthoor, and Henk Corporaal. Fast multi-dimension multi-choice knapsack heuristic for mp-soc run-time management. In *System-on-Chip, 2006. International Symposium on*, pages 1–4. IEEE, 2006.

[125] Nikolaos Zompakis, Iason Filippopoulos, Per Gunnar Kjeldsberg, Francky Catthoor, and Dimitrios Soudris. Systematic exploration of

power-aware scenarios for ieee 802.11 ac wlan systems. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, pages 28–35. IEEE, 2014.

[126] Nikolaos Zompakis, Antonis Papanikolaou, Praveen Raghavan, Dimitrios Soudris, and Francky Catthoor. Enabling efficient system configurations for dynamic wireless applications using system scenarios. *International journal of wireless information networks*, 20(2):140–156, 2013.

Appendix A

Energy Impact of Memory-Aware System Scenario Approach

Iason Filippopoulos, Francky Catthoor, Per Gunnar Kjeldsberg,
Elena Hammari and Jos Huisken

Proceedings of IEEE NORCHIP conference
IEEE, 12-13 Nov. 2012
pages 1-6
DOI: 10.1109/NORCHP.2012.6403111

Abstract

System scenario methodologies propose the use of different scenarios, e.g., different platform configurations, in order to exploit variations in computational and memory needs during the lifetime of an application. In this paper several extensions are proposed for a system scenario based methodology with a focus on improving memory organisation. The conventional methodology targets mostly execution time while this work aims at including memory costs into the exploration. The effectiveness of the proposed extensions is demonstrated and tested using two real applications, which are dynamic and suitable for execution on modern embedded systems. Reductions in memory energy consumption of 40 to 70% is shown.

A.1 Introduction

Modern embedded systems are becoming more and more powerful as the semiconductor processing technique keep increasing the number of transistors on a single chip. Consequentially, demanding applications, such as medical signal processing and streaming applications, can be executed on these devices [83]. On the other hand, the desired performance has to be delivered with the minimum power consumption due to limited amount of power offered in mobile devices [75]. System scenario methodologies propose the use of different platform configurations in order to exploit variations in computational and memory needs often seen during the lifetime of such applications [75]. A platform can, e.g., be configured through frequency/voltage scaling or turning certain processing units on or off. In this work a reconfigurable memory platform is employed in order to study the effectiveness of a memory-aware system scenario methodology.

As shown in [39] memory contributes around 40% to the overall power consumption in general purpose systems. Especially for embedded systems, the memory subsystem accounts for up to 50% of the overall energy consumption [20] and the cycle-accurate simulator presented in [108] estimates that the energy expenditures in the memory subsystem range from 35% up to 65% for different architectures. According to [75], conventional allocation and mapping of data done by regular compilers is suboptimal. Performance loss is caused by stalls for fetching data and data conflicts for different tasks, due to the limited size of memory and the competition between tasks. In addition, modern applications exhibit more and more dynamism. This gives a strong motivation for study and optimization of memory organisation in embedded devices with strongly dynamic application behaviour.

This paper is organized as follows. Section A.2 surveys related work on system level exploration and on system scenario methodologies. Section A.3 presents the chosen methodology and our novel extensions to make it applicable in a memory organisation study. In Section A.4 the target platform is described while the demonstrator applications are presented in Section A.5. Results of applying the described methodology to the targeted applications are shown in Section A.6, while conclusions are drawn in Section A.7. The main contribution of the current work is the proposal of extensions to the existing system scenario methodology, in order to take into account memory costs while performing the design exploration.

A.2 Related Work and Contribution Discussion

Many papers have focused on memory related optimisations, also in the presence of a partitioned and distributed memory organisation with memory blocks of different sizes (e.g. [10], [9], [76], [93]). However, they do not incorporate sufficient support for very dynamically behaving application codes. System scenarios allow to alleviate this bottleneck and to handle such dynamic behaviour. An overview of work on system scenario methodologies and their application are presented in [37]. So far memory organisation is rarely considered and not fully analysed. Furthermore, the majority of the published work focus on control variables for scenario prediction and selection. They can take a relatively small set of different values that can be fully explored. However, the use of data variables [46] is required for these tasks by many dynamic systems with value ranges that make full exploration impossible.

Authors in [90] present a technique to optimise memory accesses for input data dependent applications by duplicating and optimising the code for different execution paths of a control flow graph (CFG) . One path or a group of paths in a CFG form a scenario and its memory accesses are optimized using global loop transformations (GLT) . Apart from if-statement evaluations that define different execution paths, they extend their technique to include while loops with variable trip count in [91]. A heuristic to perform efficient grouping of execution paths for scenario creation is analysed in [88]. However, our work extends the existing solutions towards exploiting the presence of a distributed memory organisation with reconfiguration possibilities.

Reconfigurable hardware for embedded systems, including the memory architecture, is a topic of active research. An extensive overview of current approaches is found in [31]. The approach presented in this paper differentiates by focusing on the data-to-memory partitioning aspects in the presence of a platform with dynamically configurable memory blocks.

A.3 Extended System Scenario Methodology

The system scenario methodology is based on the observation that the workload of most systems varies significantly during their lifetime due to dynamic variation of computational and memory needs in the application code. Most of the existing design methods define the worst case execution time (WCET) of the most demanding task and tune the system in order to meet its needs [75]. Obviously, this approach leads to wasted time and memory area for tasks with lower execution time and memory requirements, since those tasks could meet their needs using fewer resources and consequentially consuming less energy.

In contrast, designing with scenarios is workload adaptive and offers different configurations of the platform and the freedom of switching to the most efficient scenario at run-time. In contrast to use case scenario approaches in which scenarios are generated based on a user's behaviour, the system scenario methodology focuses on behaviour of the system to generate scenarios. A system scenario is a configuration of the system that combines similar run-time situations (RTSs). An RTS consists of a running instance of a task and its corresponding cost (e.g. energy consumption) and one complete run of the application on the target platform represents a sequence of RTSs [46]. The system is configured to meet the cost requirements of an RTS by choosing the appropriate scenario, which is the one that satisfies the requirements using minimal power.

In the following subsections, the different steps of the system scenario methodology is outlined, with emphasis on the new extensions that make it possible to take memory into account.

A.3.1 General description of system scenario methodology

The scenario methodology follows a two stage exploration, namely design-time and run-time stages, as described in [37]. The two stage exploration is chosen because it reduces run-time overhead while preserving an important degree of freedom for run-time configuration [75]. In more detail, the application is analysed at design-time and different execution paths and variations in processing and memory demands are identified. This procedure, which is time consuming and as a result can be performed only during the design phase, will result in a grey-box model representation of the application. The grey-box model hides all static and deterministic parts of the application, by providing only related costs for those, and keeps parts of the application code that are non-deterministic available to the system designer [48]. This way, more focus can be given to parts of the application that have impact on the cost variations so that different execution decisions can be studied. Those different options are made available to the system designer and each decision could be either fixed at design time or forwarded for dynamic evaluation at run-time.

A.3.2 Design-time Profiling

Application profiling is performed at design-time and consists of an analysis of the target application during its lifetime and for a wide range of inputs. The analysis focuses on execution time for a reference processor in the conventional system scenario methodology, but in our case the cost will be memory size. This cost metric is chosen, because several applications allocate and deallocate memory space during their execution. Also, data reuse behavior and decisions made by the system designer about the size of sets of data to be copied to different units in the memory hierarchy (i.e., copy-candidates [14]) strongly influences the memory size. Energy consumption and access time of a memory unit is affected by its size [96]. Together with access pattern information, the memory size is hence an important metric.

The flow of the profiling stage is depicted in Fig. A.1 and consists of running the application code with suitable input data often found in a database, in order to produce profiling results. This reveals parts of the application code with high memory activity and with varying memory access intensity, which possibly depends on input data. Because of this behaviour, a static study of the application code alone is insufficient since the target applications for this methodology have non-deterministic behaviour that is driven by input. Choosing an extensive and accurate database is vital and will heavily influence and steer the designer's decisions in later steps.

Given code and database as inputs, profiling will show memory usage during execution time by running the application using the whole database as an input. Results provided to the designer include complete information about allocated memory size values together with the number of occurrences and duration for each of these memory size values. Moreover, correlation between input data values and the resulting memory behaviour can possibly be observed. This information is forwarded to the next stage.

There are several ways that profiling can be performed. Current program monitoring software cannot offer the needed level of detail. Debuggers and memory tools, such as Valgrind [86], or direct hard-coded profiling are preferable methods, because of their higher accuracy.

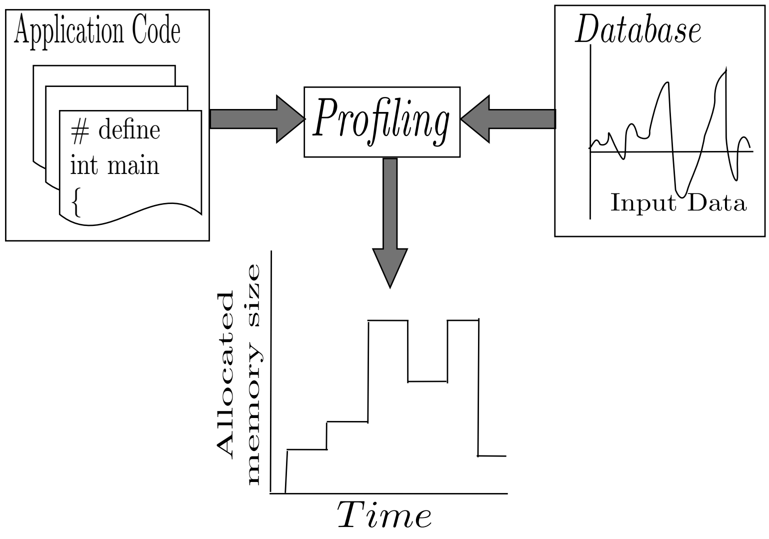


Figure A.1: Profiling results based on application code and input data.

A.3.3 Design-time Scenario Identification and Prediction

Scenario generation is also performed during design time and is the procedure of clustering profiled information into groups with similar characteristics. More precisely, cost points are clustered in groups based on their distance in cost axis and the frequency of their occurrence. Clustering is necessary, because it will be extremely costly to have a different scenario for every possible situation. Switching cost, which is the cost of switching platform to another configuration, for example by turning a memory to retention state, grows with the number of scenarios, because more scenarios result in more frequent switching. In addition, the runtime manager becomes more complex with an increasing number of scenarios.

As shown in Fig. A.2, using the information available after application profiling, situations with similar platform needs will be organised in a common scenario. This is known as clustering of RTSs [37]. This is a rational choice, because two instances with similar platform needs have similar memory energy consumption. The energy gain of having two dedicated scenarios is small and normally outweighed by switching costs if organized in different scenarios. In this example, clustering results in three different scenario areas (Fig. A.2) and the execution scenario sequence is 1(lower), 2(middle), 3(top), 2, 3 and 1. Also, the frequency of occurrence of each instance is an

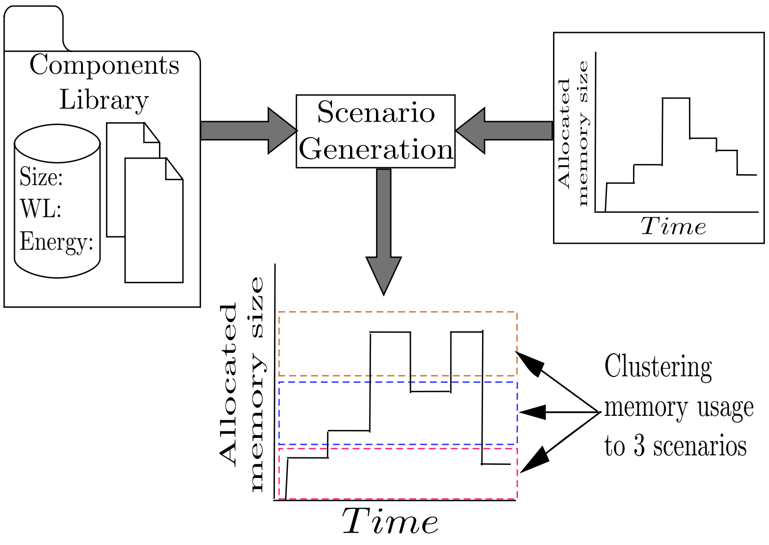


Figure A.2: Scenario generation based on profiling information and memory models.

important factor in scenario generation. Instances with higher frequencies normally have their dedicated scenario, to allow higher optimization, while less common instances can be clustered together in a less optimal scenario.

In order to generate scenarios that can be implemented in realistic memory organisations, a detailed library with memory components is needed. The library should contain a variety of memory models, including different technologies and sizes, and is used to calculate scenario costs. In this work the component library is based on memory models published in [3].

The design-time scenario prediction phase consists of determination of the variables that define the active scenario. This can be achieved by careful study of the application code, combined with the application's data input. In our case the grey-box model reveals only the code parts that will influence memory usage, so that variables deciding memory space changes can be identified. An example of this is a non static variable that influences the number of iterations for a loop that performs one memory allocation at each iteration. Moreover, the designer should look for a correlation between input values and the corresponding cost. This information will be useful in the following steps of the methodology [75].

A.3.4 Run-time Identification, Detection, and Switching

After profiling and scenario generation at design-time, all the necessary information needed for run-time management is available. The run-time manager monitors the current values of prediction variables selected in the previous design time step. Based on this, prediction of the next active scenario is performed [75].

Switching decisions will be taken at run-time by the run-time manager. The switching phase consists of all platform configuration decisions that can be made at run-time, e.g., frequency/voltage scaling, turning on/off a memory unit, and remapping of data on memory units. Switching takes place when the switching cost is lower than the energy gains achieved by switching. In more detail, the run-time manager compares the memory energy consumption of executing the next task in the current active scenario with the energy consumption of execution with the optimal scenario. If the difference is greater than the switching cost, then scenario switching is performed [75]. Switching costs are defined by the platform and include all memory energy penalties for run-time reconfigurations of the platform, e.g., extra energy needed to change state of a memory unit.

A.4 Target Platform

Selection of target platform is an important aspect of the memory-aware system scenario methodology. The key feature needed in the platform architecture is the ability to realize different scenarios generated by the methodology. Execution of different scenarios then leads to different energy costs, as each configuration of the platform results in a specific memory energy consumption. In the conventional system scenario methodology several platform reconfiguration options have been studied [37] with dynamic voltage/frequency scaling (DVFS) most used [98]. By using DVFS techniques, one can change the frequency of the processing element and its supply voltage and energy consumption accordingly. For a memory aware scenario methodology there are multiple ways that memory reconfiguration can be performed. E.g., data can be mapped to different memory units according to size and access frequency, memory units can be turned on or off, or DVFS can be used to allow exactly the access frequency currently needed.

In this paper we have selected a relatively conventional reconfigurable

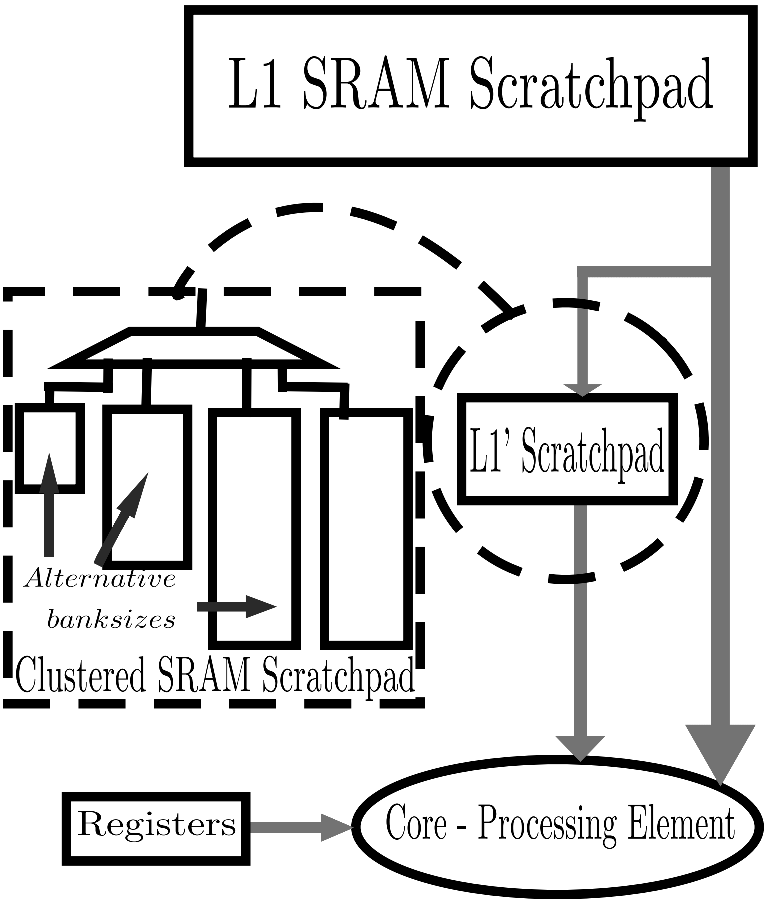


Figure A.3: Target platform with focus on memory organisation.

architecture template that is suitable for implementing system scenarios as shown in Fig. A.3. This dynamic memory organisation is based on memory prototypes presented in [3], and consists of two software controlled SRAM scratchpad memories, L1 and L1', and a processing element with its registers. For simplicity it is assumed that all data needed during execution is available in the L1 scratchpad memory without any time penalties even when large background memories are used. This assumption is reasonable for the kind of applications that are generally executed in embedded systems and can, e.g., be achieved through prefetching. The methodology is in general not restricted to this assumption, however, and can handle more complex hierarchical memory architectures, also including regular caches. Registers are used to save currently used elements, and between the reg-

Table A.1: Energy costs per second and access for a reference memory bank (size: 120 bytes, technology: 90nm) in clustered L1' memory

Modes	Leakage [J]	Wake up [J]	Access [J]
On	$351,86 \times 10^{-6}$	-	-
Off	$4,73 \times 10^{-6}$	$2,77 \times 10^{-12}$	-
Retention	$97,33 \times 10^{-6}$	$2,2 \times 10^{-12}$	-
DFF synch	-	-	0.227×10^{-6}
DFF asynch	-	-	2.18×10^{-6}
Nand2	-	-	0.334×10^{-6}

isters and the L1 scratchpad a much smaller L1' scratchpad is introduced. This is a clustered memory that consists of four memory banks that support three different states (on, off and retention modes) [3].

The memory energy consumption for every access in the clustered scratchpad memory is calculated based on the current situation of the platform as shown in Fig. A.3. The leakage energy estimation is also provided in [3] and is included in our study. In Tab. A.1 the leakage and wake up costs for all different operation modes are presented. The leakage in L1' is small compared to L1 and constant, thus the effect in the results is minimal.

The memory energy consumption per read access is given by Equation A.1, which is provided by the memory models in [3].

$$\begin{aligned} Read\,energy &= (size\,of\,bank) \times (DFF\,sync) + \\ &+ \frac{bank\,lines}{log2} \times DFF\,async + 4 \times size \times Nand2 \end{aligned} \tag{A.1}$$

The cost of accessing the clustered scratchpad organisation is a function of the overall size of the cluster and the size of the specific bank being accessed. In addition, there is a contribution in energy consumption per access added by the flip-flops and the gates needed for the correct operation of the L1' memory organisation.

A.5 Application Benchmarks

The extended methodology will be tested on two representative real life applications that differentiate significantly from each other and cover different domains of applications. The first one is a computational intensive application, in which the code is dominated by loops with dynamic bounds determined by results calculated inside the loop. The second has a more static execution path, but its memory footprint is dynamically defined by the input data. Ideal applications, that can most benefit from memory-aware system scenario methodology, are applications that have dynamic behaviour in memory organisation utilization during their execution. That required dynamism could be produced by several code characteristics, covering a wide range of potential application domains for the proposed methodology.

A.5.1 Epileptic Seizure Predictor

An epileptic seizure predictor algorithm developed at Arizona State University (ASU) [50] is chosen as a benchmark, along with a database with measurements performed on real patients, also provided by ASU. Up to now the algorithm has only been used in clinical environments using PCs, although it would be very helpful for patients as an embedded device, as part of outpatient care. Dynamic input data dependent behaviour is identified in the algorithm and memory traces for execution of three different input samples are presented in Fig. A.4. For each data sample a loop is iterated 168 times. Depending on the input data sample, different memory elements are accessed in each iteration. Note that even though the element accessing for all three samples are drawn together in the figure, only one set of memory elements are accessed for each data sample and only one data sample is processed at a time. Because of the nature of the EEG signal used for the epileptic seizure prediction the benchmark has variables with a very wide range of potential values. The memory usage is heavily influenced by these input values. For demonstration purposes our study focuses on the parts of the prediction algorithm that is most dynamic. In [46] the algorithm is split in thread nodes and dynamic behaviour affecting execution time is identified. The data reuse size for each sample in Fig. A.4 contains all the elements with data reuse factor greater than 1, i.e., are read more than once. They should be saved in L1'. For example, elements in index range from roughly 3300 to 7200 for sample 1 form an approximately 3900×(element size bytes) L1' memory size requirement. These elements are read for the first time between loop iteration 0 and 50 and are also reread later during the sample's lifetime between loop iteration 120 and 170. In Fig. A.4 these memory elements are included in the red rectangle. The exact L1' sizes needed for processing of sample 1, 2, and 3 are 3899, 2646,

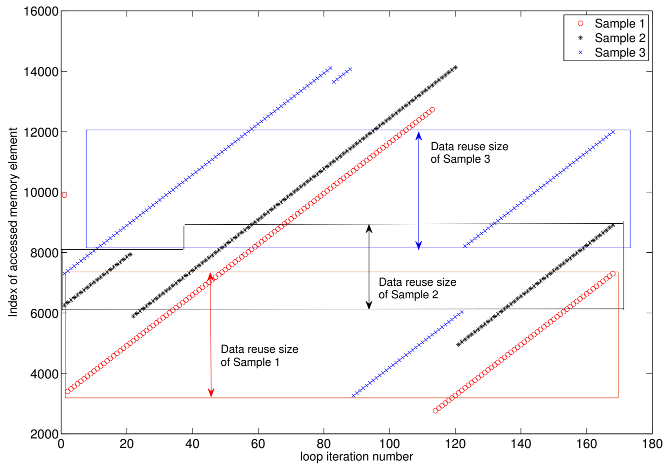


Figure A.4: Memory access pattern of epilepsy predictor. Profiling of data reuse size for 3 samples from a given ([50]) database. The number of elements accessed is input dependent. Only those of the 16K elements accessed multiple times should be saved in L1'. The rest are accessed from L1.

and 3780, respectively.

Based on profiling results, scenario generation can be performed. Two possible clustering solutions are used in this paper, both of them consisting of five scenarios. In the first clustering, the range of memory indexes is split in equally sized partitions. In the second clustering, the range is split based on occurrence frequency, so that each scenario has almost equal number of occurrences. The reconfigurable memory platform is instantiated accordingly in our target template outlined in Section A.4. It contains four equal memory banks of size 975 in the first case and four banks of increasing size (195, 585, 1170, and 1950) in the second case, since smaller sizes are used more often.

A.5.2 Viterbi Algorithm Encoder

The Viterbi algorithm [118] is widely used, both in the industry and academia, as an encoding/decoding algorithm for convolutional codes. As part of the encoding of the transmitted signal, redundant bits are added, which are later used by the decoder for correction of transmission errors. The output

Table A.2: Constraint length for SNR levels on the channel (BER $\leqslant 10^{-5}$)

SNR (dB)	K	L1' size (B)	SNR (dB)	K	L1' size (B)
6,5 - 6,1	5	40	3,9 - 3,1	9	72
6,1 - 5,5	6	48	3,1 - 2,8	12	96
5,5 - 3,9	8	64	2,8 - 2,5	14	112

of an encoder is a function of current input bit(s) and K earlier inputs, where K is the constraint length of the algorithm. Profiling shows that the constraint length dominates the memory cost. In Tab. A.2 [112] constraint length values to achieve 10^{-5} BER are presented for different noise levels in the channel. Due to its limited range of values, the constraint length is classified as a control variable. The scenario generation is based on profiling and aims at achieving a consistent bit error rate with the minimum memory usage. If SNR decreases, another memory bank is activated while reduction in channel noise leads to the opposite effect. The L1' sizes can be found in Tab. A.2. Clustering has been performed for K-values 5 and 6, with a resulting L1' size of 48B, and between K-values 8 and 9, with a resulting L1' size of 72B

A.6 Results

The memory aware system scenario methodology is applied to both of our benchmark applications to study its effectiveness. Memory energy consumption is calculated based on [3] and is the sum of (energy per access) $\times$ (number of accesses) and energy costs for all transitions between memory modes. The energy for each access is defined by the type and the size of the accessed memory. Based on profiling and scenario generation results, four different memory organizations are compared for the epileptic seizure predictor. The first one is a scenario strategy without memory awareness and use a single memory bank large enough to satisfy the most demanding sample. That approach is the worst case for the memory size and statically allocates the highest value of the memory space demanded during the lifetime of the application, i.e., 3900. However, the number of accesses to the memory will be determined by each sample and no worst case assumption is made for number of accesses.

The second approach assumes that L1' has four banks of the same size (975) that can be turned on and off. The third one assumes an L1' with

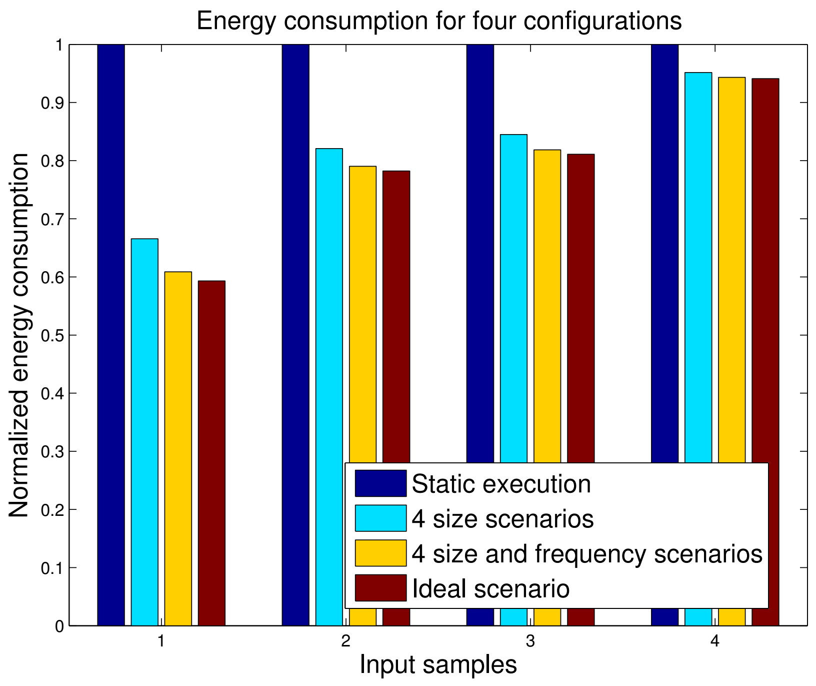


Figure A.5: Memory-aware scenario gains - Epileptic seizure predictor

four banks with different sizes (195, 585, 1170, and 1950), in order to better exploit RTSs with small memory footprints. Finally, a theoretical lower bound assumes an unlimited number of memory banks in all sizes to optimally exploit every situation. Comparison results are shown in Fig. A.5 and memory energy gains up to 40% are achieved with the scenario methodology for dynamic input samples. The high quality of our results is substantiated by the fact that our energy consumption is very close to the theoretical lower bound, even though the latter is impossible due to limited area in embedded devices.

Even higher gains are found for the Viterbi encoder (Fig. A.6) compared to a worst-case assumption of the constraint length being equal to 14. That value of length is used to achieve acceptable BER over very noisy channels. Using the extended system scenario methodology, L1' is split into four smaller banks that can be turned off when the noise of the channel is reduced. Again, the theoretical lower bound assumes a memory organisation with bank sizes optimized for each SNR level. Its hardware implementation would need 24 memory banks instead of 4 in our case, which leads to an

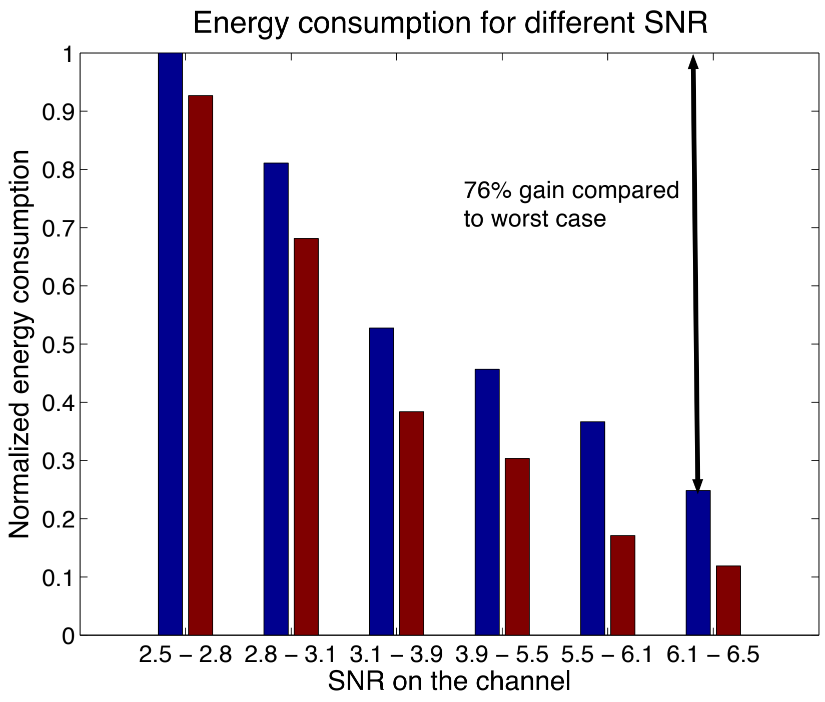


Figure A.6: Memory-aware scenario gains - Viterbi encoder

unacceptable overhead. The proposed reconfigurable memory organisation can lead to more than 70% reduction in memory energy consumption in situations with low noise, compared to a static architecture that is tuned to handle SNR levels down to 2.5 dB.

A.7 Conclusion

The scope of this work is to extend the existing system scenario techniques to better take memory into account. The memory-aware system scenario methodology has been described and tested on two real applications from bioengineering and wireless communications domains. Results justify the effectiveness of the methodology in reduction of memory energy consumption, which is of great importance in embedded devices. Since memory size requirements are still met in all situations, performance is not reduced. The memory-aware system scenario methodology is suited for applications that experience dynamic behaviour with respect to memory organisation utilization during their execution.

Appendix B

Exploration of energy efficient memory organizations for dynamic multimedia applications using system scenarios

Iason Filippopoulos, Francky Catthoor and Per Gunnar Kjeldsberg
Design Automation for Embedded Systems
Springer, 2013
Volume 17, Issue 3-4 , pages 669-692
DOI: 10.1007/s10617-014-9145-6

Abstract

We propose a memory-aware system scenario approach that exploits variations in memory needs during the lifetime of an application in order to optimize energy usage. Different system scenarios capture the application's different resource requirements that change dynamically at run-time. In addition to computational resources, the many possible memory platform configurations and data-to-memory assignments are important system scenario parameters. In this work we focus on clustering of different memory requirements into groups and presenting the system scenario generation in detail. The clustering is a non-trivial problem due to the many different memory requirements, which leads to a very large exploration space. An extended memory model is used as a practical enabler, in order to evaluate the methodology. The memory models include existing state-of-the-art memories, available from industry and academia, and we show how they are employed during the system design exploration phase. Both commercial SRAM and standard cell based memory models are explored in this study. The effectiveness of the proposed methodology is demonstrated and tested using a large set of multimedia benchmarks published in the Polybench, Mibench and Mediabench suites, representative for the domain of multimedia applications. Reduction in energy consumption in the memory subsystem ranges from 35% to 55% for the chosen set of benchmarks.

B.1 Introduction

Modern embedded systems are becoming more and more powerful as the semiconductor processing techniques keep increasing the number of transistors on a single chip. Consequently, demanding applications, e.g., in the signal processing and multimedia domains, can be executed on these devices [83]. On the other hand, the desired performance has to be delivered with minimum power consumption due to the limited energy available in mobile devices [75]. System scenario methodologies propose the use of different platform configurations in order to exploit run-time variations in computational and memory needs often seen in such applications [75].

Platform reconfiguration is performed through tuning of different system parameters, also called system knobs. For the memory-aware system scenario methodology, a platform can be reconfigured through a number of potential knobs, each resulting in different performance and power consumption in the memory subsystem. Foremost, modern memories support different energy states, e.g., through power gating techniques and by switching to lower power modes when not accessed. The second platform knob is the assignment of data to the available memory banks. The data assignment decisions affect both the energy per access for the mapped data, the data conflicts as a result of suboptimal assignment, and the number of active banks. In this work a reconfigurable memory platform is constructed using detailed memory models. This is followed by experiments with dynamic multimedia applications in order to study the effectiveness of the methodology.

The main contribution of the current work is the development of data variable [47] based system scenarios. Previous control variable based system scenarios [37] are unable to handle the fine-grain behavior of the studied multimedia applications due to their significant variation under different execution situations. Furthermore, compared with use case scenario approaches in which scenarios are generated based on a user's behavior [43], the system scenario methodology focuses on the behavior of the system to generate scenarios and can, therefore, fully exploit the detailed platform mapping information. Compared with previous work on system scenarios that has focused on the processing cores, the current work analyses the use of system scenarios on the memory organization. More specifically, this work focuses on the system scenario identification phase of the methodology. The wide range of memory requirements, the amount of different cases, and the different frequency in which each case occurs, results in a very large exploration space. Therefore, there is a need for developing an algorithmic approach that can efficiently tackle this problem.

Another significant contribution is the extensive number of benchmark applications on which the methodology is applied. The chosen set is representative for the domain of multimedia applications. Furthermore, we present a categorisation of applications based on their dynamic characteristics, also applicable to the entire multimedia domain. For the experimental needs of this work we present for the purpose sufficiently detailed and accurate memory models, which are used for the system design exploration. For the multimedia domain, the current work presents a comprehensive methodology for optimising energy consumption in the memory subsystem.

This article is organized as follows. Section B.2 motivates the study of optimization of the memory organization. Section B.3 surveys related work on system level memory exploration and on system scenario methodologies and compares it with the current work. Section B.4 presents the chosen methodology with main focus on the memory organization study. In Section B.5 the target platform is described accompanied by a detailed description of the employed memory models, while the multimedia benchmarks and their characteristics are analysed in Section B.6. Results of applying the described methodology to the targeted applications are shown in Section B.7, while conclusions are drawn in Section B.8.

B.2 Motivational Example

A large number of papers have demonstrated the importance of the memory organization to the overall system energy consumption. As shown in [39] memory contributes around 40% to the overall power consumption in general purpose systems. Especially for embedded systems, the memory subsystem accounts for up to 50% of the overall energy consumption [20] and the cycle-accurate simulator presented in [108] estimates that the energy expenditures in the memory subsystem range from 35% up to 65% for different architectures. The breakdown of power consumption for a recently implemented embedded system presented in [49] shows that the memory subsystem consumes more than 40% of the leakage power on the platform. According to [75], conventional allocation and assignment of data done by regular compilers is suboptimal. Performance loss is caused by stalls for fetching data and data conflicts for different tasks, due to the limited size of memory and the competition between tasks.

Algorithm 1 Motivational example of dynamic memory usage

```
1: while $image \neq EndOfDatabase$ do
2:    $height \leftarrow height(image)$
3:    $width \leftarrow width(image)$
4:    $store(image[height][width])$
5:    for $i = 0 \rightarrow height$ do
6:       for $j = 0 \rightarrow width$ do
7:          $array[i][j] \leftarrow func1(image[i][j])$
8:       end for
9:    end for
10:   $image \leftarrow func2(array)$
11: end while
```

In addition, modern applications exhibit more and more dynamic behavior, which is reflected also in fluctuating memory requirements [75]. Techniques have been developed in order to estimate the memory size requirements of applications in a systematic way [65]. The significant contribution that the memory subsystem has to the overall energy consumption of a system and the dynamic nature of many applications offer a strong motivation for the study and optimization of the memory organization in modern embedded devices.

To illustrate the sub-optimal conventional allocation and assignment of data, the simple example of Alg. 1 is used. The kernel code of an image processing application continuously reads a sequence of images, saves each image in memory and performs function *func1* on each pixel of the image. Arrays are typically used for storing the intermediate calculations in image processing applications, exemplified with the *array* variable in the motivation example. The memory size used for the storage of each initial *image* and the computed *array* are determined by the dimensions of the input image and can be different for a series of input images. In a conventional assignment the highest values of *height* and *width* are identified and a static compiling results in allocation of the worst-case area for the *array* variable. However, only a part of the allocated space is accessed during processing of smaller images.

Assume for instance that we have two different image sizes, ImgA with L=H=1 and ImgB with L=H=2. That is, the size of ImgB is 4 × ImgA. Each pixel in each image is accessed once giving rise to N accesses to each ImgA and 4 × N accesses to each ImgB. Furthermore, in the input stream of images there are four times as many ImgA as ImgB. In our pool of alternative memories we have three memories with Size1 = ImgA, Size2 = 3 × ImgA, and Size 3 = 4 × ImgA (i.e., the size of ImgB). The energy cost of accessing a Size1 memory is 1E, while the average leakage energy in the time between the start of two accesses is 0.3E. The corresponding access/leakage numbers for Size2 and Size3 memories are 1.3E/0.9E and 1.5E/1.2E, respectively. The numbers reflects the fact that as a first order approximation access energy increases sub linearly with increased memory size, while leakage increases linearly with memory size. The total energy (access + leakage) during computations on four ImgA and one ImgB using only one memory of Size3 is:

$$4 \times N \times 1.5E + 4 \times N \times 1.5E + 8 \times N \times 1.2E = 21.6NE$$

The same calculation using one memory of Size1 and one of Size2, in total the size of ImgB, is:

$$\begin{aligned}&4 \times N \times 1.0E + 1 \times N \times 1.0E + 3 \times N \times 1.3E\\&+ 4 \times N \times 0.3E + 4 \times N \times (0.3E + 0.9E) = 14.9NE\end{aligned}$$

giving a reduction in energy consumption of 31%. These calculations are done with simplified assumptions regarding input data and memory models. The results in Section B.7 show even larger gain with realistic dynamic applications, memory models and data.

B.3 Related Work and Contribution Discussion

The memory allocation problem has been studied before. However, we extend state of the art by proposing a more generic approach, which is also suitable for applications with input driven dynamic behavior. The authors in [10] present a methodology to generate a static application-specific memory hierarchy. Later, they extend their work in [9] to a reconfigurable platform with multiple memory banks. However, our work differentiates by proposing a more generic and application agnostic methodology and employing the use of system scenarios, in order to efficiently handle a wider range of dynamic application characteristics.

Several techniques for designing energy efficient memory architectures for embedded systems are presented in [76]. The current work differentiates by employing a platform that is reconfigurable at run-time. In [93] a large number of data and memory optimisation techniques, that could be dependent or independent of a target platform, are discussed. Again, reconfigurable platforms are not considered.

Energy-aware assignment of data to memory banks for several task-sets based on the MediaBench suit of benchmarks is presented in [78]. Low energy multimedia applications are discussed also in [23] with focus on processing rather than the memory platform. Furthermore, both [78] and [23] base their analysis on use case situations and do not incorporate sufficient support for very dynamically behaving application codes. System scenarios alleviate this bottleneck and enable handling of such dynamic behavior. In addition, the current work explores the assignment of data to the memory and the effect of different assignment decisions on the overall energy consumption.

The authors in [1], [55] and [71] present methodologies for designing memory hierarchies. Design methods with main focus on the traffic and latencies in the memory architecture are presented in [17], [40], [58] and [95]. Improving memory energy efficiency based on a study of access patterns is discussed in [61]. Application specific memory design is a research topic in [105], while memory design for multimedia applications is presented in [87]. The current work differentiates by introducing the concept of system scenarios that supports the dynamic handling of application's requirements, although the data mapping is static inside each scenario.

An overview of work on system scenario methodologies and their application are presented in [37]. In [29] extensions towards a memory-aware system scenario methodology are presented and demonstrated using theoretical memory models and two target applications. This work is an extension both in complexity and accuracy of the considered memory library and on the number of target applications.

Furthermore, the majority of the published work focus on control variables for system scenario prediction and selection. Control variables can take a relatively small set of different values and thus can be fully explored. However, the use of data variables [46] is required by many dynamic systems including the majority of multimedia applications. The range of possible values for data variables is wider and makes full exploration impossible.

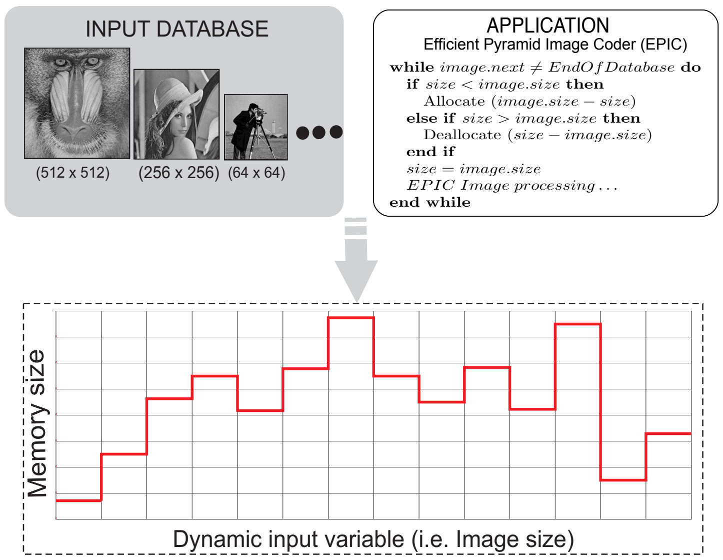
Authors in [90] present a technique to optimise memory accesses for input data dependent applications by duplicating and optimising the code for different execution paths of a control flow graph (CFG). One path or a group of paths in a CFG form a scenario and its memory accesses are optimised using global loop transformations (GLT). Apart from if-statement evaluations that define different execution paths, they extend their technique to include while loops with variable trip count in [91]. A heuristic to perform efficient grouping of execution paths for scenario creation is analysed in [88]. Our work extends the existing solutions towards exploiting the presence of a distributed memory organization with reconfiguration possibilities.

Reconfigurable hardware for embedded systems, including the memory architecture, is a topic of active research. An extensive overview of current approaches is found in [31]. The approach presented in this paper differentiates by focusing on the data-to-memory assignment aspects in the presence of a platform with dynamically configurable memory blocks. Moreover, many methods for source code transformations, and especially loop transformations, have been proposed in the memory management context. These methods are fully complementary to our focus on data-to-memory assignment and should be performed prior to our step.

B.4 Data Variable Based Memory-Aware System Scenario Methodology

The memory-aware system scenario methodology is based on the observation that the memory subsystem requirements at run-time vary significantly due to dynamic variations of memory needs in the application code. Most existing design methodologies define the memory requirements as that of the most demanding task and tune the system in order to meet its needs [75]. Obviously, this approach leads to unused memory area for tasks with lower memory requirements, since those tasks could meet their needs using fewer resources and consequently consuming less energy. Another source of energy waste in the memory system is caused by data conflicts due to misplaced data. Replacement of old data and fetching of new data is both time and energy consuming and should therefore be avoided. Handling of data conflicts is also part of a memory-aware system scenario methodology.

Designing with system scenarios is workload adaptive and offers different configurations of the platform and the freedom of switching to the most efficient scenario at run-time. A system scenario is a configuration of the system that combines similar run-time situations (RTSs). An RTS consists of a running instance of a task and its corresponding cost (e.g., energy consumption) and one complete run of the application on the target platform represents a sequence of RTSs [46]. The system is configured to meet the cost requirements of an RTS by choosing the appropriate system scenario,


INPUT DATABASE
(512 x 512)
(256 x 256)
(64 x 64)
APPLICATION
Efficient Pyramid Image Coder (EPIC)
Memory size
Dynamic input variable (i.e. Image size)

```
while $image.next \neq EndOfDatabase$ do
  if $size < image.size$ then
    Allocate ($image.size - size$)
  else if $size > image.size$ then
    Deallocate ($size - image.size$)
  end if
  $size = image.size$
  $EPIC\ Image\ processing\ldots$
end while
```

Figure B.1: Profiling results based on application code and input data

which is the one that satisfies the requirements using minimal power. In the following subsections, the different steps of the memory-aware system scenario methodology are outlined.

The system scenario methodology follows a two stage exploration, namely design-time and run-time stages, as described in [37]. This splitting is also employed in the memory-aware extension of the methodology. The two stage exploration is chosen because it reduces run-time overhead while preserving an important degree of freedom for run-time configuration [75]. The application is analysed at design-time and different execution paths causing variations in memory demands are identified. This procedure, which is time consuming and as a result can be performed only during the design phase, will result in a grey-box model representation of the application. The grey-box model hides all static and deterministic parts of the application, instead only providing their related memory costs to the system designer. Parts of the application code that are non-deterministic in terms of memory usage are directly available to the system designer [48].

B.4.1 Design-time Profiling Based on Data Variables

Application profiling is performed at design-time for a wide range of inputs. The analysis focuses on the allocated memory size during execution and on access pattern variations. Techniques described in [66] are, e.g., used in order to extract the access scheme through analysis of array iteration spaces.

The profiling stage is depicted in Fig. B.1 and consists of running the application code with suitable input data often found in a database, in order to produce profiling results. The results shown here are limited for demonstrational purposes. A real application would have thousands or millions of profiling samples. The profiling reveals parts of the application code with high memory activity and with varying memory access intensity, which possibly depends on input data variables. Because of this behavior, a static study of the application code alone is insufficient since the target applications for this methodology have non-deterministic behavior that is driven by input.

Profiling results provided to the designer include complete information about allocated memory size values together with the number of occurrences and duration for each of these memory size values. Moreover, correlation between input data variable values and the resulting memory behavior can possibly be observed. This information is useful for the clustering step that follows. Profiling also reveals the worst case memory usage for a given set of inputs. The memory usage is measured using techniques presented in [66], in which authors compute the minimum amount of memory resources required to store the elements of an application.

In Fig. B.1 the profiled applications are two image related multimedia benchmarks and the input database should consist of a variety of images. The memory requirements in each case are driven by the current input image size, which is classified as a data variable due to the wide range of its possible values. Depending on the application the whole image or a region of interest is processed. Other applications have other input variables deciding the memory requirement dynamism, e.g., the channel SNR level in the case of an encoding/decoding application.

The input data used for profiling are generated based on realistic assumptions for each of the chosen benchmarks. Each application is studied and based on its functionality, a range of rational inputs are developed. In addition, example inputs are available for most of the studied benchmark applications. The choice of common and open-source benchmarks provides

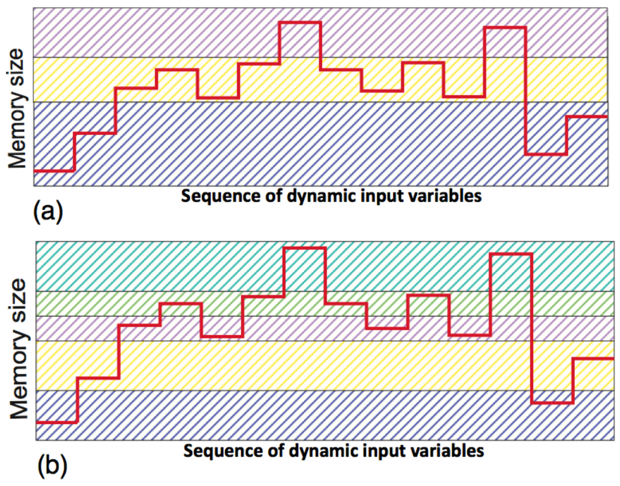


Figure B.2: Clustering of profiling results into three (a) or five (b) system scenarios

us the opportunity to find inputs publicly available. Based on the collected information, we define the range of realistic inputs and we generate a random set on inputs within these limits. For example, the bibliography provides information for the real SNR levels and the constraint length of the Viterbi encoder for each level, in order to achieve a successful communication. For profiling, we generate randomly a set of SNR levels that cover this whole range. Similar technique is used for the applications that use an image as an input. First, we explore the sizes of images commonly used to define the minimum and the maximum size and then we randomly generate a set of image sizes within the size limits. For the random generation of inputs, we assume the same probability for each situation, because there is no information available regarding the frequency of each input. As a result the frequency of each input is random.

B.4.2 Design-time System Scenario Identification Based on Data Variables

The next step is the clustering of the profiled memory sizes into groups with similar characteristics. This is referred to as system scenario identification. Clustering is necessary, because it will be extremely costly to have a different scenario for every possible size, due to the number of memories needed. Clustering neighbouring RTSs is a rational choice, because two instances with similar memory needs have similar energy consumption.

In Fig. B.2 the clustering of the previously profiled information is presented. The clustering of RTSs is based both on their distance on the memory size axis and the frequency of their occurrence. Consequently, the memory size is split unevenly with more frequent RTSs having a shorter memory size range. In the case of a clustering to three system scenarios the space is divided in the three differently coloured hashed areas depicted in Fig. B.2(a). Due to the higher frequency of RTSs in the yellow hashed area, that system scenario has a shorter range compared with its neighbouring scenarios. Such clustering is better than an even splitting because the energy cost of each system scenario is defined by the upper size limit, as each scenario should support all RTSs within its range. Consequently the overhead for the RTSs in the yellow area is lower compared to the overhead in the two other areas.

The same principle applies also when the number of system scenarios is increased to five, as depicted in Fig. B.2(b). The frequency sensitive clustering results in two short system scenarios that contain four RTSs each and three wider system scenarios with lower numbers of RTSs. The number of system scenarios should be limited mainly due to two factors. First, implementation of a high number of system scenarios in a memory platform is more difficult and complex. Second, the switching between the different scenarios involves an energy penalty that could become significant when the switching takes place frequently.

The memory size and the frequency of each RTS are not the only two parameters that should be taken into consideration during the system scenario identification. The memory size of each RTS results in a different energy cost depending on the way it is mapped into memory. The impact of the different assignment possibilities is included into the clustering by introduction of energy as a cost metric. The energy cost for each RTS is calculated using a reference platform with one to N memory banks. Increasing the number of memory banks results in lower energy per access since the most accessed elements can be assigned to smaller and more energy efficient banks. Unused banks can be switched off.

In the system scenario methodology, Pareto curves are used to capture alternative system configurations within a scenario [75]. In our work, a Pareto space is used for clustering that also includes the energy cost metric. For each RTS all different assignment options on alternative platform config-

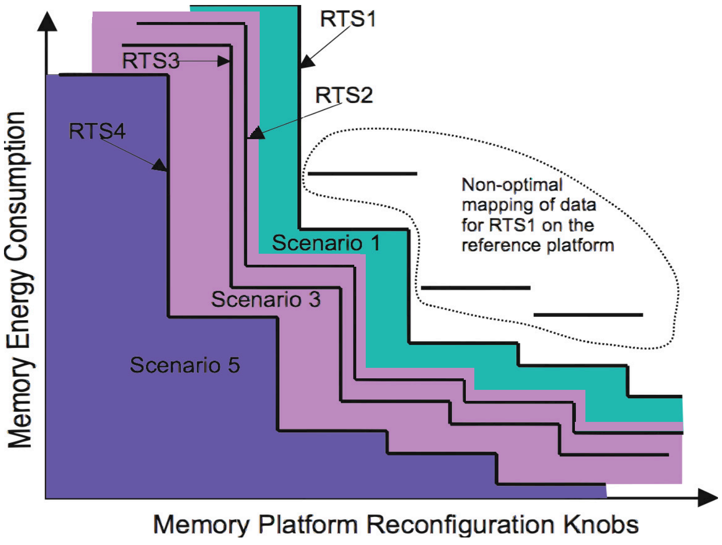


Figure B.3: Clustering of Pareto curves

urations are studied. Memory platform knobs are different sets of memory banks that are turned on and off. A Pareto curve is constructed for each RTS that contains the optimal assignment for each platform configuration. Hence, suboptimal assignments and assignments that result in conflicts are not included in the Pareto curve. In Fig. B.3 four Pareto curves, each corresponding to a different RTS, are shown together with energy cost levels corresponding to different platform configuration and data-to-memory assignment decisions. Three non-optimal mappings are also shown in Fig. B.3 for illustration. They are not part of the Pareto curve and consequently not included in the generation of scenarios. Pareto curves are clustered into three different system scenarios based again both on their memory size differences and frequency of occurrence. Clustering of RTSs using Pareto curves is more accurate compared to the clustering depicted in Fig. B.2, as it includes data-to-memory assignment options in the exploration.

The system scenario identification step includes the selection of the data variables that determine the active system scenario. This can be achieved by careful study of the application code, combined with the application's data input. The variable selection is done before clustering of RTSs into scenarios. For the choice of identification variables, there is a trade-off between the complexity and the accuracy of the scenario detection step. On one hand, if the identification is done using a group of complex variables and their correlation, there is a number of calculations needed in order to predict the active scenario. On the other hand, if the value of a single variable is monitored for scenario identification, the scenario detection is straightforward. Obviously, the accuracy of the scenario detection is higher on the first case, while the computational needs for scenario detection are lower on the second case. In other words, the more accurate scenario detection, the more resources are used by the run-time manager for detection. In our case the grey-box model reveals only the code parts that will influence memory usage, so that data variables deciding memory space changes can be identified. An example of this is a non static variable that influences the number of iterations for a loop that performs one memory allocation at each iteration. In the depicted example the system scenario detection data variable is the input image height and width values. Moreover, the designer should look for a correlation between input values and the corresponding cost. This information will be useful in the following steps of the methodology [75].

B.4.3 Run-time System Scenario Detection and Switching Based on Data Variables

Switching decisions are taken at run-time by the run-time manager. In this work, we use a simple and straightforward switching approach. Memory models provide the necessary information for the switching decision, namely the energy and the time penalty for switching between stages. The switching step consists of all platform configuration decisions that can be made at run-time, e.g., frequency/voltage scaling, changing the power mode of memory units, including turning them off, and reassignment of data to memory units. Switching takes place when the switching cost is lower than the energy gains achieved by switching.

In more detail, the switching mechanism implemented by the run-time manager includes the following actions:

1. Calculation of the energy consumption by processing the next input on the currently active scenario (E1).

2. Calculation of the energy consumption by processing the next input on its most energy efficient scenario (E2).

3. Calculation of the energy penalty for switching the needed memory banks to the configuration of the most efficient scenario (E3).

4. Evaluation of the expression: E1 >E2 + E3. If the energy cost of the current configuration (E1) is greater than the combined cost of the new configuration and the required switching (E2 + E3), then the decision is to switch. Otherwise, the switching decision is negative and the system stays on the currently running configuration

5. Switching of the platform to its new configuration. The memory banks switch to the appropriate state, which is defined by the chosen scenario.

Switching costs are defined by the platform and include all memory energy penalties for run-time reconfigurations of the platform, e.g., extra energy needed to change state of a memory unit.

The run-time manager is minimal, resulting in a very small overhead, and is complementary to an operating system (OS), if such is available on the platform. In the presence of an embedded OS, the OS needs to start the runtime manager and grant it access to system reconfiguration calls. In both cases, when the run-time manager is active, it performs a few simple steps with minimal system performance overhead. In more detail, the run-time manages checks the current state of the monitored identification variable(s), determine the next scenario based on this value and decides whether switching should be performed. The corresponding scenario for a given value of the identification variable is a simple look-up, because all the analysis has been performed at design-time. The hardware configuration for the active scenario is also explored at design-time and the run-time manager is aware of the required changes. In our approach no need is present for modifications of the application code. Instead, we add the above mentioned changes in the middleware layer. That has the additional advantage that the run-time manager is reusable across several tasks/processes running at the application layer on top of this shared middleware. The application data are stored and accessed the same way at the application level. The difference with the proposed approach is the size and the state of the memory bank that the data are stored in. The application is unaware of the exact way the data is stored and accessed and the methodology ensures that the accessed addresses in the application code always corresponds to an active bank.

In Fig. B.4 an example of the run-time phase of the methodology is depicted. The run-time manager identifies the size of the image that will be processed and reconfigures the memory subsystem on the platform, if needed, by increasing or decreasing the available memory size. The reconfiguration options are effected by platform hardware limitations. The image

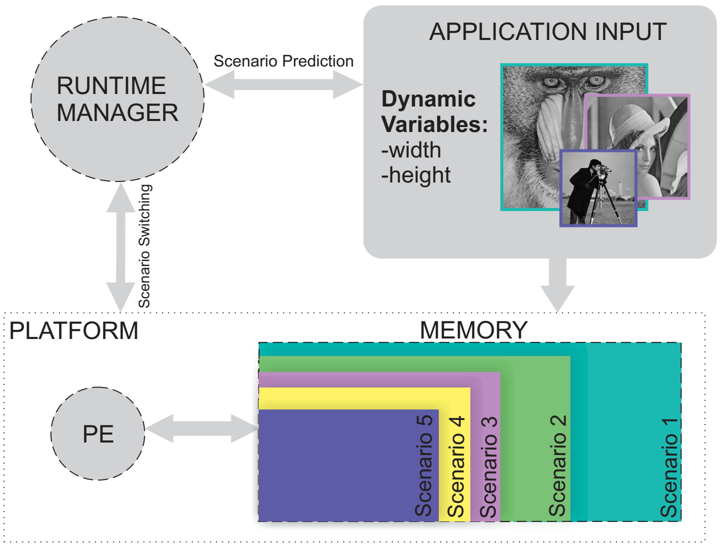


Figure B.4: Run-time system scenario detection and switching based on the current input

size is in this case the data variable monitored in order to detect the system scenario and the need for switching.

B.5 Target Platform and Energy Models

Selection of target platform is an important aspect of the memory-aware system scenario methodology. The key feature needed in the platform architecture is the ability to efficiently support different memory sizes that correspond to the system scenarios generated by the methodology. Execution of different system scenarios then leads to different energy costs, as each configuration of the platform results in a specific memory energy consumption. The dynamic memory platform is achieved by organising the memory area in a varying number of banks that can be switched between different energy states.

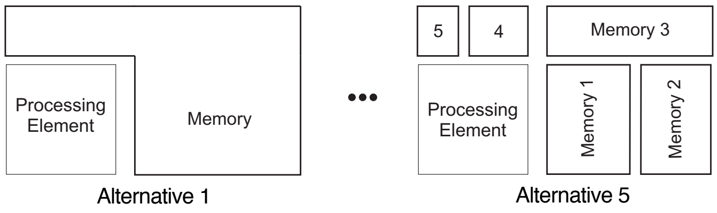


Figure B.5: Alternative memory platforms with varying number of banks

B.5.1 Target Memory Platform Architecture

In this work, a clustered memory organization with up to five memory banks of varying sizes is explored. The limitation in the number of memory banks is necessary in order to keep the interconnection cost between the processing element (PE) and the memories constant through exploration of different architectures.

For more complex architectures the interconnection cost should be considered and analysed separately for accurate results. Although power gating can be applied to the bus when only a part of a longer bus is needed, an accurate model of the memory wrapper and interconnection must developed, which is beyond the scope of the current work.

Some examples of alternative memory platforms that can be used for exploration is shown in Fig. B.5. Point-to-point connections with negligible interconnect costs between elements are assumed for up to five memory banks. The decision to use memory banks with varying sizes on the clustered memory organization increases the reconfiguration options and consequently the potential energy gains. In general, smaller memories are more energy efficient compared to larger memories banks. However, in some cases large memory banks are needed in order to fit the application data without the need for too many small memories causing complex interconnects. The goal is to use the most energy efficient banks to store the most frequently used data. The calculations needed for enabling the minimum number of banks is simple, given the application requirements for the current input and the sizes of the five memory banks.

B.5.2 Models of Different Memory Types

The dynamic memory organization is constructed using commercially available SRAM memory models (MM) . For those models delay and energy numbers are derived from a commercial memory compiler. In addition, experimental standard cell-based memories (SCMEM) [82] are considered for smaller memories due to their energy and area efficiency for reasonably small storage capacities, as argued in [80]. The standard cell-based memories are synthesized using Cadence RTL compiler for TSMC 40nm standard library. Subsequently, power simulations on the synthesized design are carried out using Synopsys PrimeTime, in order to obtain energy numbers. Both MMs and SCMEMs can operate under a wide range of supply voltages, thus support different operating modes that provide an important exploration space.

- Active mode: The normal operation mode, in which the memory can be accessed at the maximum supported speed. The supply voltage is 1.1V. The dynamic and leakage power are higher compared to the other modes. Only on active mode the data are accessible without time penalties, in contrast to light and deep sleep modes. In this work all the memory accesses are performed in active mode.

- Light sleep mode: The supply voltage in this mode is lower than active with values around 0.7V. The access time of the memory is significantly higher than the access time in active mode. Switching to active mode can be performed with a negligible energy penalty and a small time penalty of a few clock cycles (less than 10). Data is retained.

- Deep sleep mode: The supply voltage is set to the lowest possible value that can be used without loss of data. This voltage threshold is expected to be lower for SCMEMs than MM models and can be as low as 0.3V. The number of clock cycles needed for switching to active mode is higher compared to sleep mode, typically in the range of 20 to 50 clock cycles depending on the clock speed. Consequently, the speed of the PE and the real-time constrains of the applications has to be taken into consideration when choosing light or deep sleep mode at a specific time.

- Shut down mode: Power-gating techniques are used to achieve near zero leakage power. Stored data is lost. The switch to active mode requires substantially more energy and time. However, switching unused memories to this mode, providing that their data are not needed in the future, results in substantial energy savings.

The exploration includes memories with 4 energy modes in contrast to a more conservative approach that assumes only on and off states. This is in line with is modern energy efficient memories that tend to support an increasing number of energy modes as a feature. The methodology is still applicable to a memory organization that supports only two modes, but the intention of this work is to explore the state of the art memory technologies. The policy for switching between the modes depends on the reuse of the stored data, which depends on the nature of the target application. The switching policy is determined by examining the following condition for the stored data in a memory bank:

- If the data is currently under processing, then the only acceptable mode is the active mode.
- If the data is not accessed, but will be needed in the near future, then the light or deep sleep mode should be chosen.
- If the processing of the data is completed and the application is not accessing them again, the shut-down mode is the optimal solution.

Applications that perform calculations on a set of input data to produce a result and never re-access the initial data, normally only use two modes. The need for more energy modes arises from the fact that many applications re-access the initial data or some intermediate results. Thus, the runtime manager chooses a sleep mode that reduces the leakage power, but retains the data for future use.

The necessary energy/power information is available to the system designer and relative values for a subset of the used sizes in the current work are presented in Tab. B.1 and in Tab. B.2. It shows that the choice of memory units has an important impact on the energy consumption. Moreover, different decisions have to be made based on the dominance of dynamic or leakage energy in a specific application. In the current work memory architectures with 1 to 5 memory units of different sizes are explored and the optimal configuration is chosen. The methodology is in general not restricted to specific memory types or benchmarks and can handle more complex hierarchical memory architectures and applications. However, in this study the chosen applications have a relatively small memory space requirement limited to around 100KB, which is the case for many applications run on modern embedded systems.

Table B.1: Relative dynamic energy for a range of memories with varying capacity and type

Type	**Lines x wordlength**	**Dynamic Energy [J]**		Switching to Active from	
		Read	Write	Deep[uJ]	Light[uJ]
MM	32 x 8	4.18×10^{-8}	3.24×10^{-8}	0.223	0.031
MM	32 x 16	6.79×10^{-8}	5.89×10^{-8}	0.223	0.031
MM	32 x 128	4.33×10^{-7}	4.31×10^{-7}	1.42	0.168
MM	256 x 128	4.48×10^{-7}	4.60×10^{-7}	1.70	0.171
MM	1024 x 128	5.11×10^{-7}	5.75×10^{-7}	2.81	0.179
MM	4096 x 128	9.60×10^{-7}	4.57×10^{-7}	9.01	0.457
SCMEM	128 x 128	2.5×10^{-7}	0.8×10^{-8}	1.51	0.045
SCMEM	1024 x 8	1.7×10^{-8}	0.6×10^{-8}	0.325	0.021

Table B.2: Relative static power for a range of memories with varying capacity and type

Type	**Lines x wordlength**	**Static Leakage Power per Mode[W]**			
		Active	Light-sleep	Deep-sleep	Shut-down
MM	32 x 8	0.132	0.125	0.063	0.0016
MM	32 x 16	0.134	0.127	0.064	0.0022
MM	32 x 128	0.171	0.160	0.083	0.0112
MM	256 x 128	0.207	0.184	0.104	0.0293
MM	1024 x 128	0.349	0.283	0.189	0.102
MM	4096 x 128	0.95	0.708	0.544	0.396
SCMEM	128 x 128	0.083	0.057	0.027	0.0022
SCMEM	1024 x 8	0.042	0.028	0.014	0.0011

B.5.3 Total Energy Consumption Calculation

Both the dynamic and the static energy consumed in the memory subsystem is included in the calculations. The overall energy consumption for each configuration is calculated using a detailed formula, as can be seen in Eq.B.1. All the important transactions on the platform that contribute to the overall energy are included, in order to achieve as accurate results as possible. In particular:

- N_{rd} is the number of read accesses
- E_{Read} is the energy per read
- N_{wr} is the number of write accesses
- E_{Write} is the energy per write
- T is the execution time of the application
- $T_{LightSleep}$, $T_{DeepSleep}$ and $T_{ShutDown}$ are the times spent in light sleep, deep sleep and shut down states respectively
- $P_{leak_{Active}}$ is the leakage power in active mode
- $P_{leak_{LightSleep}}$, $P_{leak_{DeepSleep}}$ and $P_{leak_{Shutdown}}$ are the leakage power values in light sleep, deep sleep and shut down modes with different values corresponding to each mode
- $N_{SW\,Light}$, $N_{SW\,Deep}$ and $N_{SW\,ShutDown}$ are the number of transitions from each retention state to active state
- $E_{LightSleep\;to\;Active}$, $E_{DeepSleep\;to\;Active}$ and $E_{ShutDown\;to\;Active}$ are the energy penalties for each transition respectively.

$$\begin{aligned}
E = \sum_{memories}^{all} & (N_{rd} \times E_{Read} \\
& + N_{wr} \times E_{Write} \\
& + (T - T_{LightSleep} - T_{DeepSleep} - T_{ShutDown}) \times P_{leak_{Active}} \\
& + T_{LightSleep} \times P_{leak_{LightSleep}} \\
& + T_{DeepSleep} \times P_{leak_{DeepSleep}} \\
& + T_{ShutDown} \times P_{leak_{ShutDown}} \\
& + N_{SW\,Light} \times E_{LightSleep\,to\,Active} \\
& + N_{SW\,Deep} \times E_{DeepSleep\,to\,Active} \\
& + N_{SW\,ShutDown} \times E_{ShutDown\,to\,Active})
\end{aligned} \tag{B.1}$$

The overall energy consumption is given after calculating the energy for each memory bank. The execution time of the application is needed to calculate the leakage time. It can be found by executing the application on a reference embedded processor. The simulator described in [11] is chosen to calculate execution time for the chosen applications in this work. The processor is assumed to be running continuously, accepting new input data as soon as computations on the previous data set has been finished. Memory sleep times are hence only caused by data dependent dynamic behavior.

B.5.4 Memory Architecture Exploration

The exploration of alternative memory platforms is performed using the steps described in Alg. 2. The exploration is performed at system scenario identification phase, after the profiling of RTSs. All potentially energy efficient configurations are tested for a given number of scenarios and the sequence of RTSs of the application. First, a database with all the memory models that are available to the system designer is imported. The memory database can afterwards be pruned to reduce the complexity of the exploration, as explained in the following paragraph. After the optional pruning, all possible configurations for a given number of memory banks are constructed. The only requirement in order to keep a configuration for further investigation is that the combined size of all banks should satisfy the storage requirements of the most demanding RTS. Then, each configuration is tested for the sequence of RTSs and the one that minimizes Eq.B.1 is chosen as the most energy efficient for this number of scenarios (i.e., number of banks).

Algorithm 2 Memory organization exploration steps

```
1:  $RTSset \leftarrow$ storage requirement for each RTS
2:  $Database \leftarrow$ extensive memory database
3:  //Database pruning:
4:  for all relevant memory sizes do
5:      pick memory models from $Database$ according to application characteristics
6:  end for
7:  $m \leftarrow$ number of memory models in pruned $Database$
8:  $N \leftarrow$ number of scenarios (up to 5 in this work)
9:  //Exploration of memory organizations:
10: for $n = 1 \rightarrow N$ do
11:     $k \leftarrow$ combination of $n$ banks out of a set of $m$ memories
12:     for all generated k combinations do
13:         if $\sum_1^n size(bank) \geq size(max(RTS))$ then
14:             //Select configuration that minimizes Eq.B.1
15:             $Ecurrent \leftarrow$ Energy given by Eq.B.1 for current combination
16:             if $Eminimum > Ecurrent$ then
17:                 $Eminimum \leftarrow Ecurrent$
18:             end if
19:         end if
20:     end for
21: end for
```

The exploration for the most energy efficient memory organization is a computational intensive task and can only be performed at design-time. At run-time the search for the optimal configuration is very simple, because the set of the few possible configurations is available. The exhaustive search for the most energy efficient configuration for up to five banks is performed in a reasonable time at design-time. The number of possible configurations is given by the number of the memory banks and the number of memories in the database, as shown in Alg. 2. In general, if n is the number of banks and m is the number of memory models, there are m^n possible combinations.

Assuming a range of sizes from 1KB to 64KB, there are 7 different sizes (1KB, 2KB, 4KB, 8KB, 16KB, 32KB, 64KB). In every size, there is at most one memory model with the minimum energy per read access, one with the minimum energy per write access and one with the minimum leakage power. Depending on the worst-case memory size given by application profiling, only a number of combinations between the 7 memory sizes fulfils the requirements. For example, 5 memory banks of 1KB are not sufficient for any of the studied applications. The code of the application may reveal if there is dominance of read/write accesses for an array or it is not accessed frequently so that leakage will dominate. This way some of the memory models can be eliminated for each size. Let us assume an application with a worst case memory requirement of 100KB that is dominated by sequences of read accesses. In this case, we have only 21 possible combinations (all combinations of 5 out of a set of 7 with repetitions and unimportant order [102]). Then, we have to explore only the combinations that are greater than 100KB using bank models with the minimum energy per read.

More formally, at design-time we generate all the combinations with repetitions from a database of m memory models taken out n at a time. We are looking for sets with repetitions, which means that we can choose the same memory model more than once. This is important, in order to also include more homogeneous organizations into the exploration. However, the order of memories has no effect on our exploration. This means that the combination of memories S1 = (8KB, 8KB, 16KB, 32KB) is equivalent with S2 = (8KB, 16KB, 8KB, 32KB) and only one of them is included in our exploration. The number of possible configurations is given by the following general formula:

$$k = \binom{m}{n} = \frac{m(m-1)...(m-n+1)}{n(n-1)...1}$$

In our case, the typical values for m and n are 15 and 5 respectively. The complexity of the exploration algorithm is $\mathcal{O}(k)$, where k is the number of possible memory combinations. Therefore, the whole exploration for up to 5 memory banks can be performed during the design phase. At run-time the system can chose the appropriate configuration, without the need for exploration.

B.6 Application Benchmarks

The applications that benefit most from the memory-aware system scenario methodology are characterised by having dynamic utilization of the memory organization during their execution. Multimedia applications often exhibit such a dynamic variation in memory requirements during their lifetime and consequently are suitable candidates for the presented methodology. The

Table B.3: Benchmark applications overview

Name	Source	Scenario detection variable
Epic image compression	MediaBench	Image size
Motion Estimation	MediaBench	Image size
Blowfish decoder	MiBench	Input file size
Jacobi 1D Decomposition	Polybench	Number of steps
Mesa 3D	MediaBench	Loop bound
JPEG DCT	MediaBench	Block size
PGP encryption	MediaBench	Encryption length
Viterbi encoder	Open	Constraint length

effectiveness is demonstrated and tested using a variety of open multimedia benchmarks, which can be found in the Polybench [99], Mibench [42] and Mediabench [69] benchmark suites. The broad set of multimedia benchmarks under exploration is representative for the entire domain of multimedia applications.

B.6.1 Benchmark Applications and Corresponding Input Databases

An overview of the benchmark applications that were tested is presented in Tab. B.3.

Two key parameters under consideration are the dynamic data variable of each application and the variation in the memory requirement it causes. The dynamic data variable is the variable that results in different system scenarios due to its range of values. Examples of such a variable are an input image of varying size or data dependent loop bound values. For each application an appropriate set of realistic RTS cases is constructed. The memory size limits are defined as the minimum and maximum storage requirement occurring during the profiling of an application.

EPIC (Efficient Pyramid Image Coder) image compression can compress all possible sizes of images. The size of the input image has an effect on memory requirements during compression and several images were given as inputs. *Motion estimation* is another media application in which image size is the dynamic data variable. In this case the image defines the area that has to be explored to determine the motion vectors and different images are tested. The set of input data is constructed using publicly available images commonly used for testing this algorithm.

The dynamism in the *blowfish decoder* benchmark is a result of variations in the input file that is decoded. Again, the methodology explores the behavior for several input files in order to identify system scenarios. The *Jacobi 1D decomposition* algorithm can be executed using a varying number of steps with a direct effect on memory usage and is hence another suitable benchmark for the system scenario methodology. The number of steps is increased on every next iteration, in order to generate a set of different memory requirements. The set of input data is a random sequence of input signals and each of them corresponds to a different number of steps. *Mesa 3D* is an open graphics library with a dynamic loop bound in its kernel that provides the desired dynamic behavior.

The discrete cosine transformation (DCT) block used in the *JPEG compression* algorithm has a memory footprint that is heavily influenced by the block size. The input database consists of an ascending size sequence of blocks. For the *PGP encryption* algorithm the encryption length parameter has an important impact on memory size, which can be exploited using system scenarios. Thus, we create a database starting from the lowest encryption length value of 384 and gradually increasing it up to 2048. The effect of the channel SNR level on the constraint length value of the *Viterbi encoder* algorithm is discussed in [29]. Increasing noise on the channel demands a more complex encoding in order to maintain a constant bit error rate (BER), which consequently increases the memory requirements during execution. The memory size variation is given for execution under different SNR levels.

B.6.2 Classification of Applications Based on Dynamic Characteristics

The required dynamism for applying the memory-aware system scenario methodology can be produced by several code characteristics, covering a wide range of potential applications, as discussed in the previous subsection. In this subsection dynamic characteristics are outlined that can assist the system designer in the employment of the methodology and reveal the expected behavior prior to experimentation with an application. The dynamic characteristics that are used to categorize the applications are the dynamism in the memory size bounds and the variance of cases within the memory size limits. The characterization of the benchmark applications

Table B.4: Characterization of benchmark applications (See Tab. B.3 for index

Name	**Dynamic Characteristics**	
	Memory Variation(B)	Shape of histogram
Epic image compression	4257 - 34609	Right skewed
Motion Estimation	4800 - 52800	Gaussian-like
Blowfish decoder	256 - 5120	Left skewed
Jacobi 1D Decomposition	502 - 32002	Right skewed
Mesa 3D	5 - 50000	Gaussian-like
JPEG DCT	10239 - 61439	Gaussian-like
PGP encryption	3073 - 49153	Gaussian-like
Viterbi encoder	5121 - 14337	Right skewed

based on those key parameters is presented in Tab. B.4.

The profiling information can be organised into a histogram in order to be easily comprehensive. The horizontal axis depicts the memory requirements for the RTSs and the vertical axis the number of occurrences for its RTS. In this way the system designer can quickly identify the expected gains of the system scenario methodology by identifying the width and the shape of the histogram. The width of the histogram gives an overview of the memory size bounds, while the shape reveals the variation of the RTSs for the studied application.

The memory size bounds correspond to the minimum and maximum memory size values profiled over all possible cases. In general, larger distances between upper and lower bounds increase the possibilities for energy gains. This is a result of using larger and more energy hungry memories in order to support the memory requirements for the worst case even when only small memories are required. Large energy gain is expected when large parts of the memory subsystem can be switched into retention for a long time. For several of the benchmarks the difference between maximum and minimum memory size is close to 50KB. This includes JPEG, motion estimator, mesa 3D, and PGP, where large gains can be expected. On the other hand, the system designer should expect lower energy gains for applications that show a relatively less dynamic behavior with regard to their memory size limits. Examples here are the blowfish and viterbi algorithms.

The variation takes into consideration both the number of different cases that are present within the memory requirement limits and the distribution of those cases between minimum and maximum memory size. This variation corresponds to the shape of the histogram of the application. Applications with a limited number of different cases are expected to have most of its possible gain obtained with a few platform supported system scenarios and much smaller energy gains from additional system scenarios. After this point most of the cases are already fitting one of the platform configurations and adding new configurations have a minimal impact. The opposite is seen for applications that feature a wide range of well distributed cases.

Based on the analysis above, applications can be classified into four main categories. The first category has a histogram with a wide range of RTS memory sizes and most RTSs placed to the left. This category is defined as a right skewed distribution, according to the direction of the tail. In this category the RTSs corresponding to large memories are rarely used, so high gains are expected by applying the methodology. The second case is when there is a close to Gaussian RTS distribution in the histogram. The expected gains are here lower than for the first category. The opposite of the first category is when the histogram is left skewed, meaning that the RTSs with the higher memory requirements are dominant. In this category, system designer should expect the smallest gains. The forth case is when all the RTSs have the same memory requirements and there is no distribution on the histogram, which results in no energy gains for the methodology.

B.7 Results

The memory aware system scenario methodology is applied to all the presented benchmark applications to study its effectiveness. The profiling phase is based on different input for the data variables shown in Tab. B.3 and is followed by the clustering phase. Different stimuli are used for the profiling phase and the run-time calculations. The two different sets of input are generated using the same random distribution function. The two input sets have different sequences of inputs and the occurrence of each RTS is random. The probability of each RTS is kept the same in the two input sets. The execution and sleep times needed in Eq.B.1 are found through the profiling but are also reflected by the dynamic characteristics in Tab. B.4. Data variables are the variables used by the run-time manager in order to predict the next active scenario. The clustering is performed with one to five system scenarios. All potentially energy efficient configurations are tested for a given number of scenarios using the steps described in Alg. 2. For example, in the case of 2 scenarios all possible memory platforms with 2 memory banks that fulfil the memory size requirement of the worst case are generated and tested. The same procedure is performed for 3, 4 and 5 scenarios. The exploration includes memories of different sizes, technologies and varying word lengths.

The proposed methodology provides the same performance on the memory subsystem and only reduces energy, if possible. The access time is the same both in the static and the dynamic memory architecture. The goal of the methodology is to always meet the memory requirements of the applications, using the minimum amount of active memory banks. A worst case scenario is activated any time that the input is outside of the range of inputs. The worst case configuration activates all the available memory banks. In this case the system achieves its highest capability, but the energy consumption is the highest. The active scenario always provides enough space to fit the active data for an input within the profiled range of inputs. The energy gains are a result of switching unnecessary banks to a low-power mode when it is possible. The active banks have the same performance and are independent of any inactive banks on the clustered architecture.

The energy gain percentages are presented in Fig. B.6. Energy gains are compared to the case of a fixed non-re-configurable platform, i.e., a static platform configuration with only 1 scenario. This corresponds to zero percentage gain in Fig. B.6. The most efficient of the tested organizations for each benchmark are presented in Fig. B.7, where each memory bank is depicted with a different colour and each length is proportional to the memory bank size. The blowfish decoder is the only benchmark that has only 3 banks in its most efficient memory organization. In Tab. B.5 the minimum and maximum energy gains for each benchmark application are shown.

B.7.1 Classification of the Applications

The introduction of a second system scenario results in energy gains between 15% and 40% for the tested applications. Depending on the application's dynamism the maximum reported energy gains range from around 35% to 55%. As expected according to the categorisation presented in subsection B.6.2, higher energy gains are achieved for applications with more dynamic memory requirements, i.e., bigger difference between the minimum and maximum allocated size. The maximum gains for JPEG, motion estimator, mesa 3D and PGP are around 50% while blowfish, jacobi, and Viterbi decoders are around 40%.

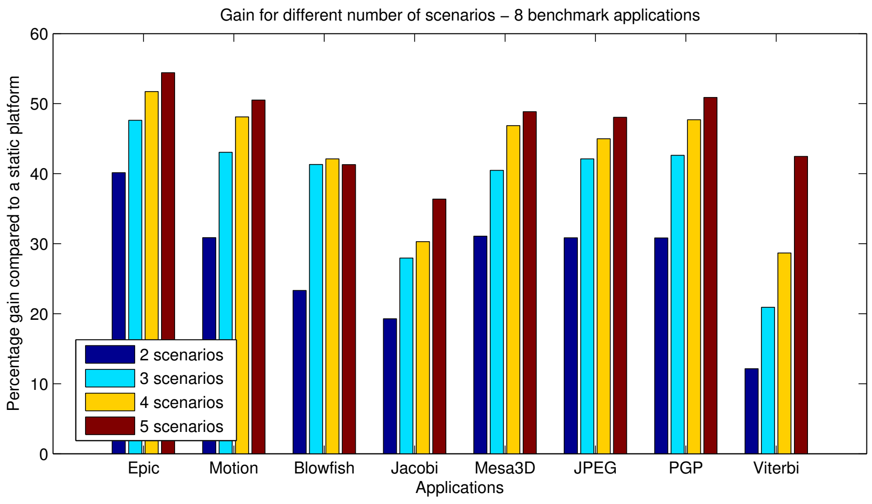


Figure B.6: Energy gain for increasing number of system scenarios - Static platform corresponds to 0%

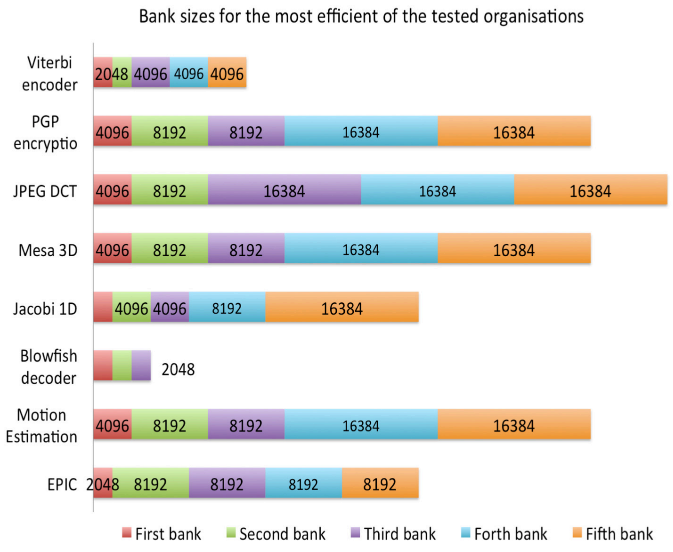


Figure B.7: Bank sizes for the most efficient of the tested organizations for each benchmark

Table B.5: Range of energy gains on the memory subsystem

EPIC		**Motion**		**Blowfish**		**Jacobi**	
Min	Max	Min	Max	Min	Max	Min	Max
40.6%	55.1%	31.2%	50.1%	23.5%	42.0%	19.8%	35.9%
Mesa3D		**JPEG**		**PGP**		**Viterbi**	
Min	Max	Min	Max	Min	Max	Min	Max
31.3%	49.2%	31.1%	48.9%	30.6%	51.2%	12.5%	42.4%

As the number of system scenarios that are implemented on the memory subsystem increases, the energy gains improve since variations in memory requirements can be better exploited with more configurations. However, the improvement with increasing numbers of system scenarios differ depending on the kind of dynamism present in each application. The application with the highest variation in distribution of memory requirements is the Viterbi encoder/decoder and gains around 10% is seen for every new memory bank added, even for a platform growing from four to five banks. In contrast, the application with the lowest number of different cases, blowfish, cannot further exploit a platform with more than three banks. Another case in which smaller energy gains are achieved, after a certain number of platform supported system scenarios have been reached, is the PGP encryption algorithm. In this benchmark the introduction of more scenarios has an energy impact of less than 5% after the limit of three system scenarios has been reached.

B.7.2 Switching Overhead

The switching cost increases for an increasing number of system scenarios due to the increasing frequency of platform reconfiguration. This overhead reduces the achieved gain, but for up to 5 scenarios we still see improvements for all but one of our benchmarks. The switching cost is below 2% even for a platform with 5 memory banks in all cases. Apart from the number of scenarios, the switching cost depends on the sensitivity of the variable used for scenario identification. A change of value on the identification variable indicates potentially a new scenario. For an increasing frequency of changes, the switching cost increases.

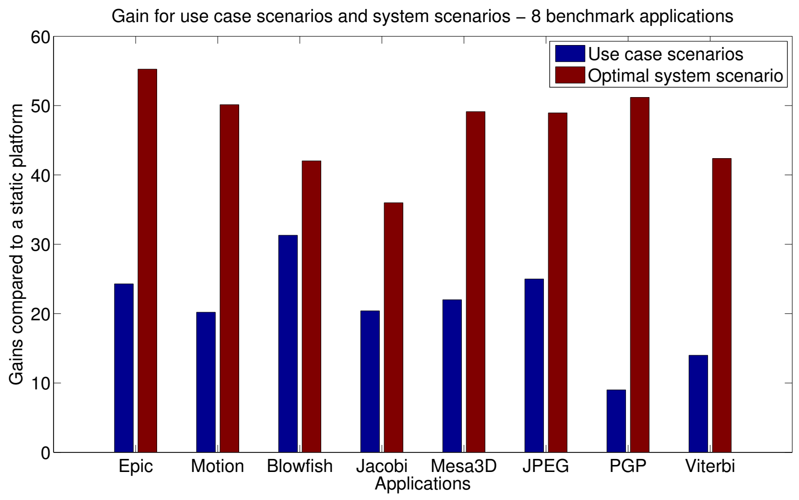


Figure B.8: Energy gain for use case scenarios and system scenarios

B.7.3 Comparison with Use Case Scenario

Comparative results from applying a use case scenario approach as a reference are presented in Fig. B.8. Reported energy gains for both use case scenarios and the most efficient case of the system scenarios are given assuming a static platform as a base (0%). Use case scenarios are generated based on a higher abstraction level that is visible as a user's behavior. For example, use case scenarios for image processing applications generate three scenarios, if large, medium and small are the image sizes identified by the user. Similarly, use case scenarios for JPEG compression identify only low and high compression as options and motion estimation is performed on I, P and B video frames, without exploring fine grain differences inside a frame. In general, use case scenario identification can be seen as more coarse compared to identification on the detailed system implementation level. As seen in Fig. B.8 the use case gains are superior only to a static platform.

B.7.4 Run-Time Overhead

The reported energy gains are for the memory subsystem. As motivated in Section B.2 this has previously been shown to be a major contributor to the total energy consumption. An additional energy overhead from the system scenario approach can be found in the processor performing the run-time system scenario detection and switching. This overhead is partly incorporated in $E_{SleepActive}$, in particular if traditional system scenarios are already implemented so that the only overhead is the addition of memory-awareness. The run-time overhead is kept low, because the run-time manager is active for less than 1% of the time needed for the execution of the application.

B.8 Conclusions

The scope of this work is to apply the memory-aware system scenario methodology to a wide range of multimedia application and test its effectiveness based on an extensive memory energy model. A wide range of applications is studied that allow us to draw conclusions about different kinds of dynamic behavior and their effect on the energy gains achieved using the methodology. The results demonstrate the effectiveness of the methodology reducing the memory energy consumption with between 35% and 55%. Since memory size requirements are still met in all situations, performance is not reduced. The memory-aware system scenario methodology is suited for applications that experience dynamic behavior with respect to memory organization utilization during their execution.

Appendix C

Systematic Exploration of Power-Aware Scenarios for IEEE 802.11ac WLAN Systems

Nikolaos Zompakis, Iason Filippopoulos, Per Gunnar Kjeldsberg, Francky Catthoor and Dimitrios Soudris

17th Euromicro Conference on Digital System Design (DSD)
IEEE, 27-29 Aug. 2014
pages 28 - 35
DOI: 10.1109/DSD.2014.76

Abstract

This work explores the power management options for a transmitting wireless system using system scenarios. We exploit the variations in the communication channel and the protocol requirements during the lifetime of a transmission, in order to optimize energy usage. Both the transmission signal power and the memory subsystem are taken into consideration. Different system scenarios and the corresponding configurations capture the different resource requirements, which change dynamically during transmission. Signal power on the antenna and active memory banks are the two main platform parameters explored in this study and sufficiently detailed system models are presented for both. The trade-off between the accuracy of the generated system scenarios and the switching cost between them is analyzed. The exploration is performed for an increasing number of system scenarios, from 1 to 14, and the reported power gains are over 95% and over 25% on the signal power and the memory subsystems respectively.

C.1 Introduction

Modern wireless technology [26] has opened new horizons in the means and ways that users communicate. Radio devices exist in a multitude of items such as cell phones, tablet computers and digital TVs. The different types of applications demand different types of communication standards. Modern 4G networks provide high quality of services (QoS) exploiting new innovative techniques, which combine smart transceivers and high performance receivers. This trend creates new challenges that the conventional radio equipment cannot cope with. Promising technologies, like Software Defined Radio (SDR) [116], attempt to integrate the rapidly changing wireless networks, merging existing and new communication standards into one platform. In this context, IEEE 802.11 wireless local area networks (WLANs) play an important role in fourth-generation wireless mobile communication systems [15]. The development of WLANs has primarily been guided by legacy IEEE 802.11a/b/g devices. With the recent emergence of the IEEE 802.11n and the upcoming 802.11ac standards, WLANs are given the option to operate over wider channels that achieve higher transmission rates. IEEE 802.11ac supports very high bandwidth communication with a targeted data rate greater than 1 Gbps in the band below 6 GHz [51].

Meanwhile, wireless applications become more and more resource demanding, intending to provide better QoS. However, the majority of these applications are operating under the limitations of the mobile handsets. Thus, the design challenges include both the high performance requirements and the low power consumption constraints. New techniques that can efficiently exploit the energy resources are undoubtedly an imperative need, in order to extend the battery life. In this direction, the signal power and the memory system represent the main sources of power consumption on a mobile device. Signal power account at least for 50% of the power consumption when a mobile device is on a standby mode, and even more during calls. The memory subsystem has also an important contribution due to the data intensive nature of the wireless applications. Both the transmit power requirements on the antennas and the data requirements on the memory change dynamically. Smart antennas optimize automatically their transmission and their reception pattern responding to the changes of the external environment. Flexible OFDM systems exploit the channel state information adjusting the transmit power and increasing the achieved data rate [57][56]. At the same time the chosen coding and modulation scheme for data transmission affects the memory requirements.

The dynamic behavior outlined above results in unnecessarily high energy consumption if the system is statically designed to accommodate the worst-case behavior. This is avoided if a system scenario methodology is applied [38]. At design time a number of cost-optimized scenarios are selected so that the system can be reconfigured at run-time according to the current dynamic situation. In the current study, we consider a wireless system where the transmit power and the activity of the memory is adapted to the running situation based on application constraints and channel conditions. The scope is to classify from a cost perspective the functionality of a transmission process exploiting the system scenario methodology. The first main contribution is the specification of the energy optimal configurations of a potential wireless system focusing on the energy consumption of both transmission signals and memories. The second key contribution is that we examine the trade-off between the accuracy of the extracted scenarios and their switching overhead, which is correlated with the system configuration cost. For our purpose, a state of the art communication protocol (802.11ac) is chosen with a large fluctuation on the supported communication characteristics.

This paper is organized as follows. Section II surveys related work and differentiation of the current work. Section III presents an overview of the system scenario principles. In Section IV the system models for the signal power and the memory are described, while the case study of this work is analyzed in Section V. Results for the case study are shown in Section VI. Finally, conclusions are drawn in Section VII.

C.2 Related Work

Transmission signal power adaptation is a key issue for energy-aware wireless devices. Several algorithms have been proposed for dynamic power management of the antenna signal. The critical point in all cases is the interference of the neighbor users. In a wireless system, the power allocation problem can be efficiently addressed by the water-filling algorithm [100]. The scope of this algorithm is to manage, in an optimal way, the antenna signal transmission power of every user in a wireless system network. The target is to maximize the available capacity of the transmission channel. The main drawback of this approach is that the provided QoS is not taken into consideration. QoS has been successfully considered in a number of other studies. In [121] authors attempt to minimize the whole system transmission power under fixed performance requirements for a given sets of user data rates. In [19] an extension of the traditional water-filling technique, considering users queue-length constraints, is presented. In the context of a cognitive radio OFDM system, authors in [7] propose optimal power loading schemes for a single user and extend their proposal for multiple users in [6]. In [64] optimal power control policies are presented focusing on fading channel at cognitive radio networks. These policies pay special attention to the interference influence. All the above solutions have the drawback that they presuppose a centralized control, which implies significant changes to the network infrastructure. A second important drawback is that they are characterized by high implementation complexity.

In the context of the current study we work towards overcoming these drawbacks. Regarding the first, we propose a distributed power adjustment approach, which can be applied at client-platform level. This approach exploits the signal power scaling range between the minimum required SNR (Signal to Noise Ratio) (based on the noise conditions and the required data rate) and the permissible signal power radiation (EMC, FCC Rules Dictate Antenna Use). The main contribution is focused on the second design obstacle, which is related with the implementation complexity cost. In respect to that we apply a scenario-based methodology which provides the required flexibility to cluster and classify the multi-operation conditions into system scenarios which are effectively detected and exploited at run-time. The key issue is that we examine the trade-offs between the switching and clustering overhead. A somewhat similar trade-off was presented in [124], but this is the first time that it is applied in a systematic way for a wireless application and especially for the specific characteristics of WLAN IEEE 802.11ac systems.

A comprehensive study on a low-power architecture specifically designed for software radio is presented in [73]. The target wireless protocols explored are W-CDMA and 802.11a and the power consumption breakdown is analyzed for the whole system. In both cases the memory architecture are an important contributor to the overall energy consumption, exceeding 30% for the worst case. In [72] a power saving mechanism for high speed WLAN applications is presented. The authors propose a hierarchical memory design to reduce memory access operations and report a 30 to 40% power dissipation reduction. This method is complementary to ours, because it reduces the number of memory accesses while we focus on optimizing the energy per access. Again, the main differentiation with the current work is the scenario exploration, which is viable by employing a platform that is reconfigurable during run-time.

The system scenario methodology has been described in [34] and is presented in Section III. The system scenario concept was also outlined in [35], where the tasks are written using a combination of a hierarchical finite state machine (FSM) and a synchronous dataflow model (SDF). The disadvantage of this method is that the applications must be written using a limited model, which is a time consuming and error-prone operation. In [45] authors propose a scenario-based framework for managing processor resources while achieving QoS on contemporary multi-core processors running media-streaming applications. In [126] the efficiency of system scenarios for dynamic wireless applications is shown. The usage of system scenarios on the memory is presented in [29]. In this study, the main focus is on the dynamic management of both the signal power and the memory activity. Apart from the application of the system scenario methodology on an emerging wireless protocol, we perform a design exploration trade-off analysis. To be more specific, we examine the main tradeoff between clustering overhead, which is correlated with how representative are the scenarios, and switching overhead, which represents the tuning cost of the platform.

C.3 System Scenario Principles

The system scenario methodology is a design approach for handling the complexity analysis of applications with multi-dimensional costs and strict constraints. The main challenges are the optimal application mapping on the platform and the efficient management of the platform resources. In particular, by classifying and clustering the possible system executions into system scenarios, a run-time resource manager can substantially reduce the average cost resulting from this execution compared to the conventional worst-case bounding approach, while still meeting all constraints.

The aim of system scenarios is to capture the data dependent dynamic behavior inside a thread in order to better schedule a multi-thread application on a heterogeneous multi-processor architecture. [36] presents a design methodology that provides a systematic way of detecting and exploiting system scenarios for streaming applications. A scenario is defined as the application behavior for a specific type of input data, i.e. a group of execution paths for that particular group of input data.

An important term in the scenario methodology is the Run-Time Situation (RTS). An RTS is a deterministic thread execution path with specific and fixed cost dimensions. RTSs are treated as execution units. The system scenario methodology comprises 5 individual steps as defined by [38]:

1. Identification. The first step is to identify all possible RTSs. To achieve this, we identify and classify all RTS parameters. As a parameter, we can assume every variable affecting the state of the system from a functionality point of view.

2. Characterization. Each RTS can be characterized by a number of cost factors obtained from profiling the application on a platform or by using high-level cost estimators. The placement of the possible cost points of each RTS on the cost space leads to a Pareto curve. The Pareto curve contains all the potential exploitation points in the multi-dimensional exploration space.

3. Clustering. It defines the grouping of the individual execution situations into system scenarios. The clustering process introduces inevitably an overestimation, caused by the deviation between the real cost of the RTS and the estimated cost which is the representative cost for the system scenario of the RTS. An efficient clustering is very important and the aim is to keep run-time overhead low without high overestimation.

4. Detection. This step implements the mechanism, which detects in which of the several system scenarios the current RTS belongs. System scenario detection can be performed by monitoring the changes of the application parameters.

5. Switching. Switching includes both the switching decision, which is whether a system reconfiguration will be applied or not, and the switching mechanism, which apply the chosen system reconfiguration to the platform. A scenario switch is performed if the gain of switching is greater than the switching cost.

More details about the methodology steps can be found in [38]. An aim of this study is to exploit the system scenario classification decreasing the design complexity, as the high number of RTSs is grouped in a small number of scenarios.

C.4 System Model

C.4.1 Antennas Signal Power

We consider an uplink Wireless transmission channel of a MIMO-OFDM system based on the IEEE 802.11ac communication protocol [51]. The transmission data rate, for which we can achieve a successful transmission, is defined by the bandwidth, the capacity and the noise on the channel. A fundamental trade-off exist between Bit-Error-Rate (BER), which is correlated with the provided QoS, and antenna signal power. A potential run-time reconfiguration manager can adjust the signal power and the memory subsystem to the running situation. The scheduler selects the energy optimal configuration scheme (number of spatial streams, bandwidth, modulation and coding (MC) schemes) which respect the running constrains, based on the targeted communication standard (WLAN 802.11ac) characterization [51]. More precisely, the scheduler chooses the communication scheme, which requires the minimum SNR for the current data rate requirements under given conditions of external distortion. This presupposes that the scheduler has perfect updated knowledge of the channel condition and the application deadlines. The antenna signal power is adjusted to give the required data rate.

The aforementioned fundamental bound between signal power and data rate under specific noise conditions is mathematically expressed by the ShannonHartley theorem [8]:

$$C = B \times log_2(1 + S/N) \tag{C.1}$$

C is the channel capacity in bits per second; B is the bandwidth of the channel in hertz; S is the average received signal power over the bandwidth, measured in Watt; N is the average noise or interference power over the bandwidth, measured in Watt; and S/N is the signal-to-noise ratio (SNR).

This equation shows that a theoretical minimum SNR exists for achieving a target capacity with specific available channel bandwidth. The minimum SNR for a specific level of noise defines the minimum required signal power for an error-free transmission. For example, if the available bandwidth is Bw the theoretical minimum SNR for a transmission with bit-rate Cb without errors is:

$$SNR \leq 2^{C_b/B_w} - 1 \tag{C.2}$$

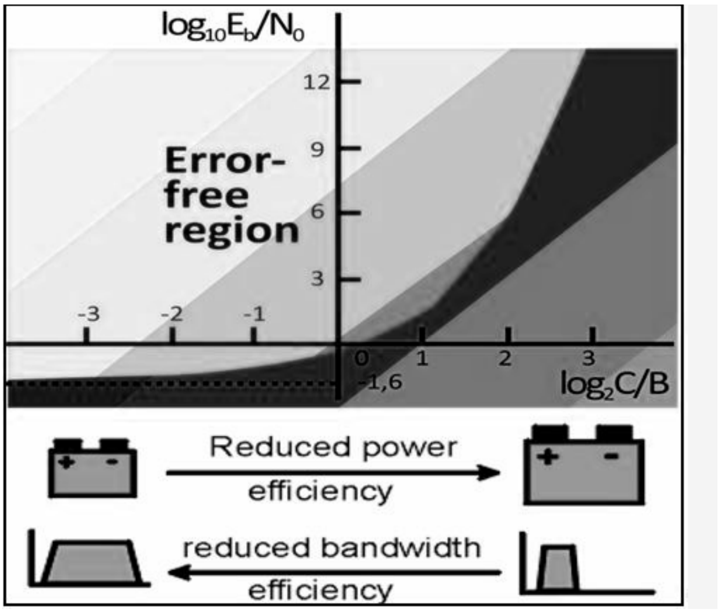


Figure C.1: ShannonHartley theorem

The average signal power, S, can be written as S=EbC, where Eb is the average energy per bit. The average noise power, N, can also be redefined as, N=N0B, where N0 is the noise power (Watts/Hz). The ShannonHartley theorem [8] can be written in the form:

$$\frac{C}{B} = log_2(1 + \frac{E_b C}{N_0 B}) \tag{C.3}$$

The ratio C/B represents the bandwidth efficiency of the system in bits/second/Hz. The graphical representation of the ShannonHartley theorem presented in Fig.C.1 shows that bandwidth efficiency can be traded for power efficiency and vice-versa. The light gray area represents the area free from errors. Knowing the SNR levels, we can characterize the total signal power efficiency of every configuration (minimum Signal Power) to achieve the targeted capacity. If the configuration supports multiple antennas (multiple spatial streams) the total signal power is estimated as the sum of the signal of each antenna.

The graphical representation of the ShannonHartley theorem, see Fig.C.1, clearly shows that throughput can be traded for power efficiency, and vice-versa. The light gray area represents the area free from errors. The theoretical minimum SNR for an error-free transmission is impossible to reach in

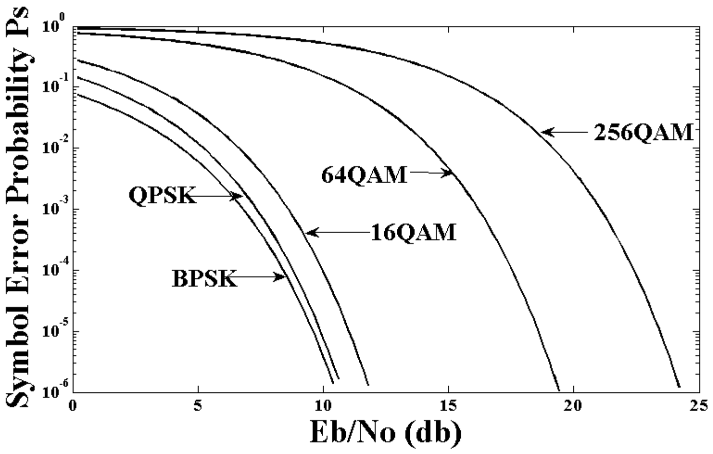


Figure C.2: Symbol error probability for 802.11ac Modulation schemes

practice. The modulation schemes define how close to this theoretical SNR-min the transmission can be. Every modulation scheme is characterized by a minimum SNR that allows the demodulation of the transmitted symbols without errors. Knowing the minimum SNR for every modulation scheme (MS), we can define the minimum Signal Power for every MS for specific levels of noise. The equation that defines the symbol error probability (Ps) for every MS, with respect to SNR is the following [100]:

$$P_{s,M-ary} = (\frac{M-1}{M}) \times erfc(\sqrt{\frac{3}{M^2-1} \times \frac{E_{Saver}}{N_0}}) \tag{C.4}$$

M is the number of symbols used, Es the average received signal power, N0 the average noise signal power and erfc is the complementary error function. The graphical expression of this equation for the modulation schemes of the 802.11ac is presented in Fig.C.2. Channel coding improves the SNR by a factor R [8]. So the curves can be normalized for equal energy per information bit (pre-coding) bearing in mind that the energy per transmitted bit is less than the energy per information bit by a factor equal to the code rate R. The graphical expression of the symbol error probability for the modulation and coding (MC) schemes of the 802.11ac can be found in Fig.C.3.

In this context, every system scenario RTS is characterized by a two-dimensional cost 1) the total signal power and 2) the bit error rate (BER). The signal power is inversely proportional to the symbol error probability and correspondingly to bit error probability as shown in Fig.C.2 and Fig.C.3.

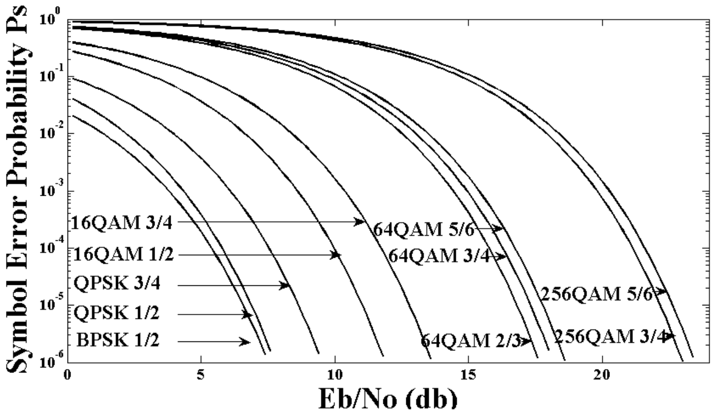


Figure C.3: Symbol error probability for 802.11ac Modulation and Coding schemes

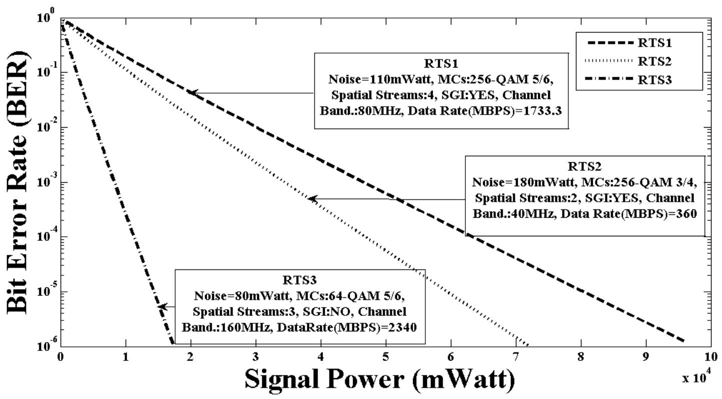


Figure C.4: RTS Characterization

Each RTS is characterized by a curve in the two-dimensional space of total signal power. This curve is derived by the respective curve at Fig.C.3 that corresponds at the MCs of the RTS. Based on the bits-per-symbol of MCs (BPSK: 1bps, QPSK: 2bps, etc.), the short guard interval (SGI) and the noise level of the RTS, the Ps (symbol error probability) to SNR curve can be transformed to BER to Signal Power curve. In Fig.C.4, due to limited space, we present 3 representative examples of these RTS curves. For a given BER, the minimum signal power to achieve successful communication is chosen and this point represents the system configuration for this RTS.

Besides the above-mentioned technical analysis the most unstable parameter for a transmission is the user profile, e.g., the distance between receiver and transmitter, the existence of other communication channels or others sources of distortion. These are factors that influence the channel transmission and are directly influenced by the user behavior. For example, if the user moves in a saturated spectrum area or in a noisy environment high communication channel interference is expected. Correspondingly, if the user changes position very rapidly, (for example, driving a car) this has impact on the normal demodulation of the transmitted signal (Doppler Effect).

C.4.2 Memory Banks

The SNR level and the changing environment on the wireless channel also affects the memory requirements. In more detail, the conditions of the channel determine the coding and modulation scheme needed for a successful communication and, consequently the required data rate. The coding phase transforms an m-bit data string into an n-bit string in order to be encoded, when the given coding rate is m/n. The modulation phase conveys a varying number of bit streams together, based on the chosen modulation. The data rate constraint defines the storage and transmission requirements for the data. As a result, the memory footprint depends on the data rate of the channel and is dynamic for a changing environment. Energy consumption on the memory subsystem depends on the number of accesses and the energy per access, which are different based on the size and the type of memory. The observation that the memory requirements at run-time vary significantly due to dynamic variations on the transmission channel and the protocol, is exploited through use of system scenarios. Instead of defining the memory requirements for the worst-case data rate and tuning the system according to this, system scenarios are generated for different situations. The combination of the coding and the modulation parameters define the data rate for each RTS. The data rate is the identification variable and the cost factor is its memory footprint. Based on the cost factor, the different memory footprints are clustered into scenarios. The clustering of RTSs is based both on their distance on the memory size axis and the frequency of their occurrence.

The key feature needed in the platform architecture is the ability to efficiently support different memory sizes that correspond to the system scenarios generated by the methodology. Execution of different system scenarios then leads to different energy costs, as each configuration of the platform results in a specific memory energy consumption. The dynamic memory platform is achieved by organizing the memory area in a varying number of banks that can be switched between different energy states.

In this work, a clustered memory organization with up to five memory banks of varying sizes is explored. The dynamic memory organization is constructed using commercially available SRAM memory models (MM). For those models delay and energy numbers are derived from a commercial memory compiler. In addition, experimental standard cell-based memories (SCMEM) [81] are considered for smaller memories due to their energy and area efficiency for reasonably small storage capacities. The standard cell-based memories are synthesized using Cadence RTL compiler for TSMC 40nm standard library.

Both MMs and SCMEMs can operate under a wide range of supply voltages, thus support different operating modes that provide an important exploration space.

- Active mode: The normal operation mode, in which the memory can be accessed at the maximum supported speed. The supply voltage is 1.1V. The dynamic and leakage power are higher compared to the other modes. In this work all the memory accesses are performed on the active mode.

- Light/Deep sleep mode: The supply voltage in this mode is lower than active with values around 0.7V and 0.3V respectively. The access time of the memory is significantly higher than the access time in active mode. Switching to active mode can be performed with an energy penalty (switching cost) and a small time penalty of a few clock cycles. Data is retained.

- Shut down mode: Power-gating techniques are used to achieve near zero leakage power. Stored data is lost. The switch to active mode requires substantially more energy and time. However, switching unused memories to this mode results in substantial energy savings.

The goal is to design a suitable clustered scratchpad memory architecture that can serve the generated scenarios. When the maximum data rate is required by the protocol, all the memory banks should be active and accessed. When the needed data rate is lower, the size and the rate required by the memory system are lower and one or more memory banks can be switched off. As a result, the amount of energy consumed during a system scenario with lower requirements is reduced.

C.4.3 Combined Model

The optimal energy configuration both on the signal power and the memory subsystem takes into consideration six parameters, namely (a) the number of spatial streams (streams), (b) the MC schemes, (c) the channel bandwidth, (d) the SGI, (e) the data rate, and (f) the noise level. Based on their values the run-time manager choses the active scenario for the signal power and the memory subsystem and performs the appropriate reconfigurations. The key point is that this decision is not taken independently. For the first time in the context of a wireless system, two system scenario sets adjusted to achieve an overall optimization. This represents a new methodology extension because the clustering step has to take into consideration not only the overhead impact at the targeted subsystem, but also the interaction with the other subsystem scenario sets. Thus, it is provided an optimal configuration for the whole system and not sub-optimal for individual parts.

C.5 Case Study

In the context of the current case study, we examine the potential power gain exploiting two scenario-based schedulers. The first adjusts dynamically the antenna signal power and the second manages the memory banks activation. For the needs of our study, we consider an artificial application that generates data bursts that have to be transmitted within specific deadlines. The distribution of the generated data bursts is presented at Fig.C.5. For every data burst there is a corresponding transmission time deadline, which defines a minimum data rate. The distribution of the time deadlines is presented at Fig.C.6.

Every combination of six parameters discussed in Section IV.C represents a different communication scheme. The memory adaptation scheduling takes into consideration only the data rate. The first four parameters (streams, MCs, BW, SGI) is defined by the targeted protocol (802.11ac) [51] creating an exploration space of 240 RTSs. Given values for these four parameters, the maximum value for the fifth parameter, data rate, can be calculated. Thus, data rate does not contribute to the exploration space, but it is used as an identification variable of the suitable communication scheme. More precisely, the desired data rate is extracted dynamically by the running application requirements (Data Bursts size and Deadline) and

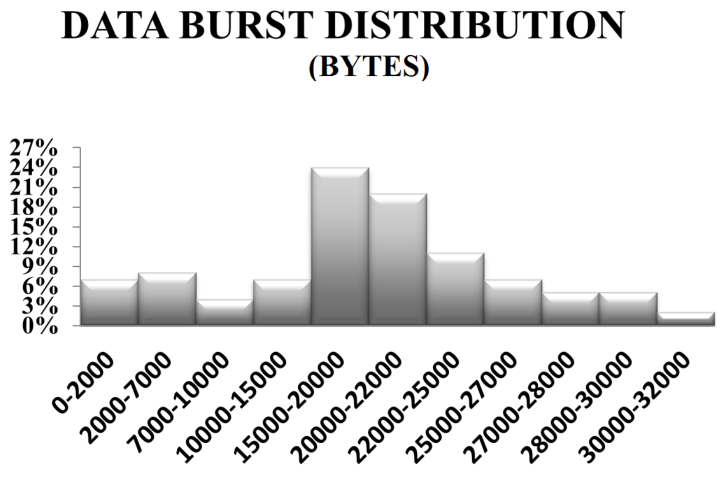


Figure C.5: Data Burst Distribution

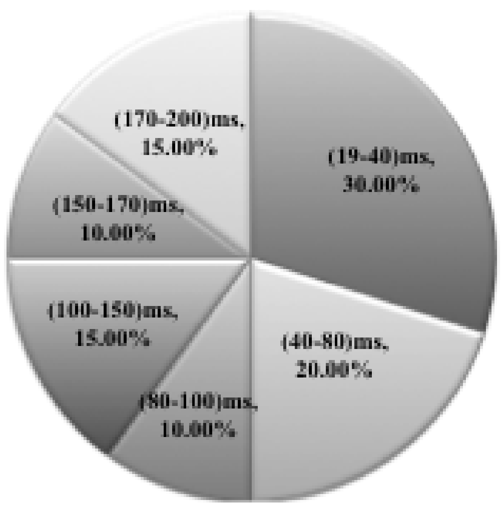


Figure C.6: Application Deadline Distribution

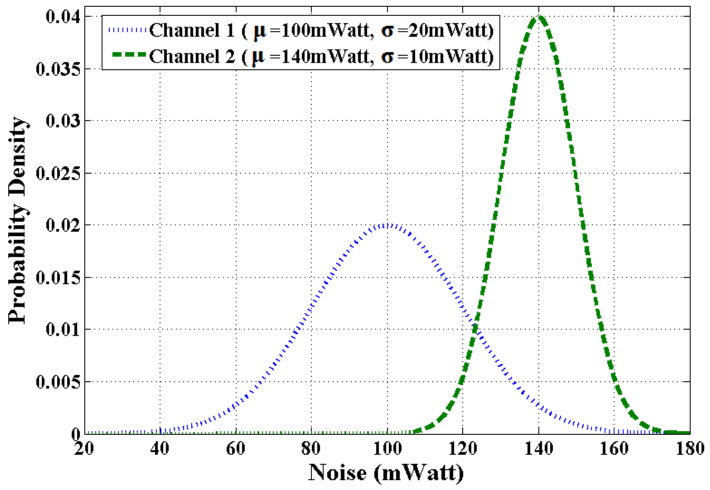


Figure C.7: Noise Distribution

the scheduler identifies which communication schemes can satisfy this data rate. The last and the most important variable is the external noise. In order to be more representative, we take noise distributions and not discrete values of noise. For the needs of our measurements we consider two channels with different Gaussian noise distributions as shown in Fig.C.7 having different average values (μ) and standard deviations (σ) ($channel_1 : \mu = 100mWatt, \sigma = 20mWatt, channel_2 : \mu = 140mWatt, \sigma = 10mWatt$). The second channel represents a noisier and more rapidly changing environment compared to the first. The noise distribution defines the appearance probability of the running situations (RTSs). A different noise distribution creates a different probability density for the RTSs. For example in our case the probability density of the RTSs, with noise range ($\mu, \mu + \sigma$) and (μ and $\mu - \sigma$), is 34.13% (because of the Gaussian distribution). Similarly, the RTSs probability for ($\mu, \mu \pm 2\sigma$), ($\mu, \mu \pm 3\sigma$), ($\mu, \mu \pm 4\sigma$) and ($\mu, \mu \pm 5\sigma$) are calculated. Thus, for a Gaussian distribution, we have in total 10 bounding levels of noise, which in combination with the values of the previous parameters creates an exploration space of (240×10) 2400 RTSs. The clustering of the RTSs into system scenarios varies based on the noise distributions. Thus, for every distribution we have a different optimal set of system scenarios.

The scheduling decision about the suitable communication scheme follows three main priorities: 1) the required data rate, 2) the noise tolerance and 3) the power saving. The first criterion fulfills the current application deadlines while the second ensures the signal power for an acceptable BER.

Thus, the signal power scheduler first finds the communication schemes that respect the running time constrain, and in a second stage chooses these, that permit the lowest signal power. The final decision is based on the overall system power saving.

In our case study, we have in total three different sets of system scenarios; two for the signal power scaling (one for each of the two channels presented in Fig.C.7) and a third set for the memory banks power management (based on the data size distribution presented in Fig.C.5). To minimize memory bank power, the memory scheduler at run-time activates a suitable number of scratch pad memory banks based on the selected coding scheme. Data are transferred to the memories as part of the transmission chain during the modulation phase.

For the extraction of the system scenarios, we take into consideration the trade-off between the run-time implementation overhead and the accuracy of the extracted scenarios. The first is related to the switching overhead while the second is related to the number of the scenarios and the clustering overhead. In comparison with the approach presented in the second case study of publication [126] (related to signal power adaptation), the presented study has the following differentiations. Except from the different characteristics of protocol 802.11ac (compared with 802.11n), in our case the proposed scheduler both adjusts signal power and selects the optimal power communication scheme (MCs, streams, BW etc.) based on the current noise conditions and data rate constrains. This increases the design space complexity from 80 RTSs [126] to 2400 RTSs. Additionally, the current approach takes into consideration two channel noise distributions (one for every channel) which differentiates the RTS appearance frequency, changing dramatically the clustering of RTSs into system scenarios (as the RTS frequency affects significantly the clustering overhead [126]). Furthermore, in the context of the current study we differentiate by presenting the trade-offs between switching and clustering overhead for different numbers of extracted system scenarios and by including memory into exploration. In respect with the last, we insert a new methodology extension adjusting the clustering of two system scenario sets (Signal Power and memory) to achieve an overall optimization.

C.6 Results

Tab.C.1, Tab.C.2 and Tab.C.3 present the variation of the clustering and switching overhead. The first column (number of scenarios) gives the number of system scenarios for each row. The second column (clustering overhead compared to WCE) and third column (clustering overhead compared to previous step) presents normalized the percentage variation of the clustering overhead compared with the worst case (monolithic approach) and that of the previous row, respectively. When RTSs are clustered into scenarios, the cost for the most energy demanding RTS is used as the representative cost for all the RTSs within the scenario. The system is configured to serve the most demanding RTS of the scenario and consequently all the less demanding RTSs, which belong to the scenario. The clustering overhead represents the difference between the RTSs within a scenario. As the number of scenarios increases the percentage approaches 100%, which would be the value if there was a dedicated scenario for each RTS. Similarly, the fourth column (switching overhead) and fifth column (switching var) represents the switching overhead expressed as the scenario switching probability and the percentage change of the switching overhead compared with that of the previous row, respectively.

In Tab.C.1, for example, for a set of 8 system scenarios we have a clustering overhead of 7% and a decrease in clustering overhead compared with the previous row (7 system scenarios) of 19%. The switching overhead is 85% and it is increased by 4% compared with the previous row. For an increasing number of scenarios, the clustering overhead is reduced, because the difference between the RTSs within the scenarios decreases. Respectively, the switching overhead is increased, because there are more scenarios to choose from and thus the system switches between scenarios more often. Depending on the design objectives, it is possible to choose an optimized number of scenarios. A larger number of scenarios increases the implementation overhead (more switching overhead) but it reduces the clustering overhead and vice versa.

The power gain percentages for each of the channels and the memory are presented in Fig.C.8. Results are given for 10 executions of the case study with the noise distribution given in Fig.C.7. The baseline is a system designed based on the worst-case requirements. The system is assumed monolithic without any reconfiguration based on the noise level and the MC scheme. The gain for the first antenna is around 97% for all the cases, while for the second antenna it is a little lower. The worst case has a noise

Table C.1: Memory Banks Scenario Overhead

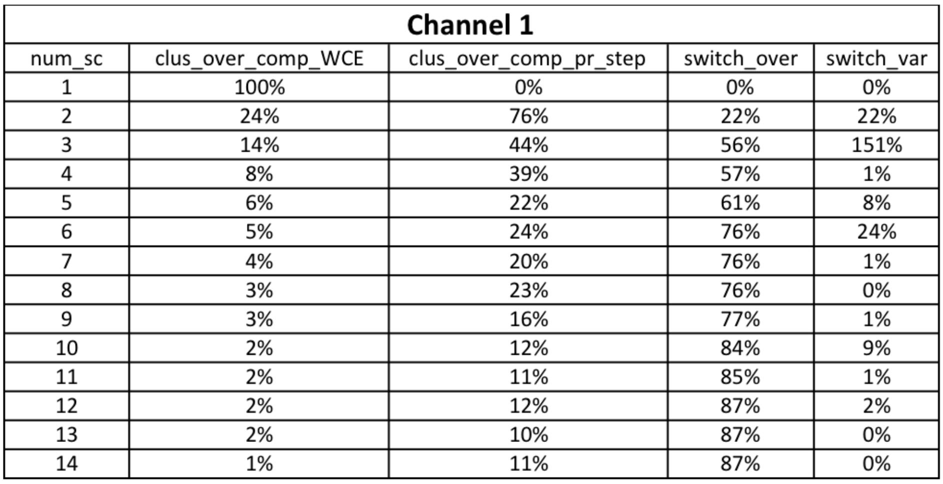
Memory Banks				
num_sc	clus_over_comp_WCE	clus_over_comp_pr_step	switch_over	switch_var
1	100%	0%	0%	0%
2	41%	59%	44%	44%
3	27%	33%	48%	9%
4	17%	36%	65%	36%
5	13%	27%	78%	19%
6	11%	17%	79%	2%
7	8%	20%	82%	3%
8	7%	19%	85%	4%
9	5%	21%	86%	0%
10	4%	18%	88%	3%
11	4%	18%	89%	1%
12	3%	17%	90%	1%
13	2%	19%	90%	0%
14	2%	22%	91%	1%

level of 180 mWatt, MC scheme 5/6 256QAM, SGI and 4 MIMO, while best case has a noise level of 20 mWatt, MC scheme 1/2 BPSK, no SGI and 1 MIMO. The reported gains for the memory are in the range from 27.8% to 29%. The power savings are significantly higher on the antennas due to two main reasons. Firstly, the variation between the best and the worst case on the antennas is wider and thus offers a greater opportunity for energy savings. Secondly, the reconfiguration option for the antennas offer more flexibility and all possible signal power levels can be exploited. The memory architecture has a physical limitation in the number of memories that can be used and only a range of sizes is supported by the system models.

C.7 Conclusion

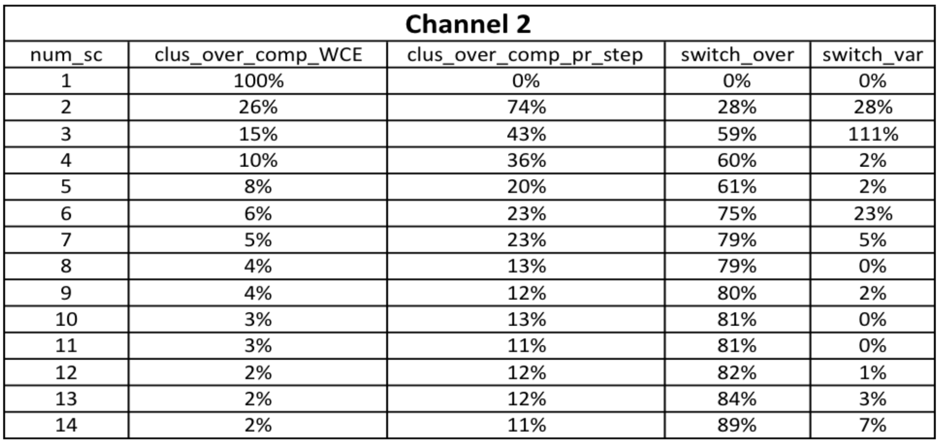
The scope of this work is to explore the power management options for a transmitting wireless system using system scenarios. The dynamic parameters and the variations during the transmission for the targeted wireless protocol are analyzed. Based on the analysis and the system models, system scenarios are generated for the case study, in order to optimize energy usage. Both the signal power and the memory subsystem are taken into consideration. The results demonstrate the effectiveness of the methodology and illustrate the interesting trade-off between the clustering overhead

Table C.2: Signal powerChannel 1 Scenario Overhead



Channel 1				
num_sc	clus_over_comp_WCE	clus_over_comp_pr_step	switch_over	switch_var
1	100%	0%	0%	0%
2	24%	76%	22%	22%
3	14%	44%	56%	151%
4	8%	39%	57%	1%
5	6%	22%	61%	8%
6	5%	24%	76%	24%
7	4%	20%	76%	1%
8	3%	23%	76%	0%
9	3%	16%	77%	1%
10	2%	12%	84%	9%
11	2%	11%	85%	1%
12	2%	12%	87%	2%
13	2%	10%	87%	0%
14	1%	11%	87%	0%

Table C.3: Signal powerChannel 2 Scenario Overhead



Channel 2				
num_sc	clus_over_comp_WCE	clus_over_comp_pr_step	switch_over	switch_var
1	100%	0%	0%	0%
2	26%	74%	28%	28%
3	15%	43%	59%	111%
4	10%	36%	60%	2%
5	8%	20%	61%	2%
6	6%	23%	75%	23%
7	5%	23%	79%	5%
8	4%	13%	79%	0%
9	4%	12%	80%	2%
10	3%	13%	81%	0%
11	3%	11%	81%	0%
12	2%	12%	82%	1%
13	2%	12%	84%	3%
14	2%	11%	89%	7%

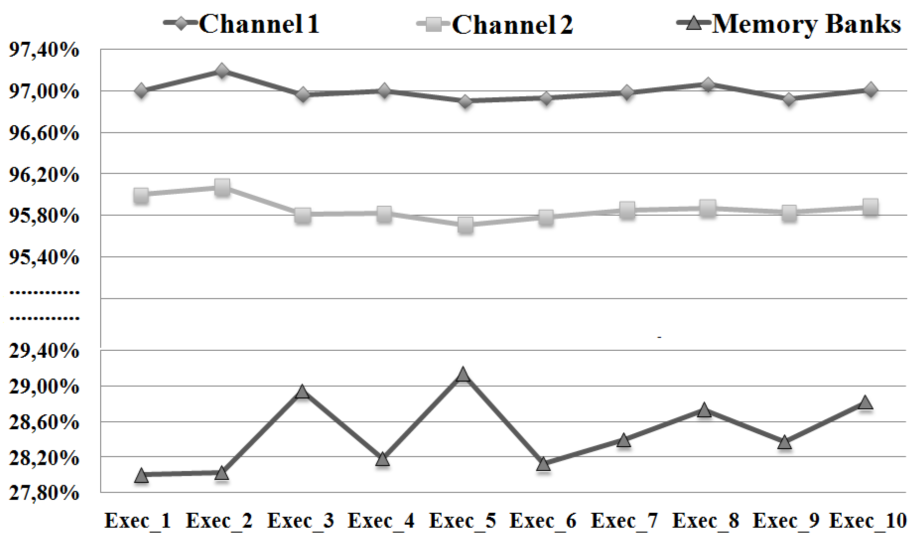


Figure C.8: Power Gain

and the switching cost.

Appendix D

Integrated Exploration Methodology for Data Interleaving and Data-to-Memory Mapping on SIMD architectures

Iason Filippopoulos, Namita Sharma, Francky Catthoor,
Per Gunnar Kjeldsberg and Preeti Ranjan Panda
ACM Transactions on Embedded Computing Systems
Submitted in 2015
(currently under second round of review)

Abstract

This work presents a methodology for efficient exploration of data interleaving and data-to-memory mapping options for SIMD (Single Instruction Multiple Data) platform architectures. The system architecture consists of a reconfigurable clustered scratch-pad memory and a SIMD functional unit, which performs the same operation on multiple input data in parallel. The memory accesses contribute substantially to the overall energy consumption of an embedded system executing a data intensive task. The scope of this work is the reduction of the overall energy consumption by increasing the utilization of the functional units and decreasing the number of memory accesses. The presented methodology is tested using a number of benchmark applications with irregularities in their access scheme. Potential gains are calculated based on the energy models both for the processing and the memory part of the system. The reduction in energy consumption after efficient interleaving and mapping of data is between 40% and 80% for the complete system and the studied benchmarks.

D.1 Introduction

Energy is an important limiting factor for modern embedded systems. New novel hardware solutions have been proposed to reduce the energy consumption. On the processing side, SIMD architectures offer new possibilities for potential improvements on the performance and the energy consumption. On the memory side, modern memory architectures provide different energy modes and clustered scratch-pad memory banks that can operate independently, offering more options for reducing energy consumption. Dynamic and data intensive algorithms are implemented on embedded systems. Data intensive applications have often holes in their memory accesses, meaning that the data in memory are not always accessed in order. The lack of spatial locality and the presence of redundant data results in lower utilization of the hardware resources, given that a conventional approach is employed for the handling of data. This problem motivates the development of a methodology to improve the management of the data.

In order to tackle this problem we propose an interleaving exploration that aims to increase spatial locality and reduce the memory accesses on redundant data. The goal of this work is to improve the energy consumption for data intensive applications without any loss on the application's performance. We focus on single instruction multiple data (SIMD) architectures and explore applications that have irregularities in their access scheme. SIMD architectures can potentially increase the performance of an application, providing that the utilization of them is high. However, applications with irregular access patterns do not provide compact sequences of data that are suitable for high utilization. Hence the performance is lower than expected and the number of memory accesses is high due to the accessing of redundant data. In order to reduce the number of memory accesses and achieve higher utilization of the system architecture a systematic exploration of the interleaving options and memory mapping for an application's data is needed.

A large number of papers have demonstrated the importance of the memory organization to the overall system energy consumption. As shown in [39] memory contributes around 40% to the overall power consumption in general purpose systems. Especially for embedded systems, the memory subsystem accounts for up to 50% of the overall energy consumption [20] and the cycle-accurate simulator presented in [108] estimates that the energy expenditures in the memory subsystem range from 35% up to 65% for different architectures. The breakdown of power consumption for a recently implemented embedded system presented in [49] shows that the memory subsystem consumes more than 40% of the leakage power on the platform. According to [75], conventional allocation and assignment of data done by regular compilers is suboptimal. Performance loss is caused by stalls for fetching data and data conflicts for different tasks, due to the limited size of memory and the competition between tasks.

The energy consumption can be divided into two parts, namely the processing and the memory subsystem consumption. The energy needed for processing depends mainly on the utilization of the FUs and any potential stalls, if the memory cannot provide data at the needed rate. The interleaving exploration can increase the utilization of the processing subsystem and reduce time penalties for data loading. The energy consumption in the memory subsystem is affected by the number of memory accesses and the energy per access. Again, the memory architecture and the data-to-memory mapping decisions have great impact on both the number of accesses and the energy per access.

This article is organized as follows. Section D.2 motivates the study of developing a methodology for optimization of the data interleaving exploration and the data-to-memory mapping. Section D.3 surveys related work on both the interleaving and memory mapping exploration. Section D.5 presents the general work-flow and the sequence of the methodology steps. In Section D.4 the target platform is described accompanied by a detailed description of the employed SIMD architecture and the memory models. The set of benchmarks is analyzed in Section D.6 and results of applying the described exploration methodology to the targeted applications is also presented there. Finally, conclusions are drawn in Section D.7.

D.2 Motivational Example

Many different techniques have been proposed already to deal with the memory accesses and the potential memory bottlenecks. Authors in [77] propose a low power cache memory for embedded systems that can optimize memory accesses based on application's requirements. The goal in [41] is to effectively utilize the scratch-pad memory of an embedded system in order to improve the memory accesses. A recent survey of the developed techniques for improving cache power efficiency is presented in [84]. Reconfigurable hardware for embedded systems, including the memory architecture, is a topic of active research. An extensive overview of current approaches is found in [31]. The current work is complementary to the existing ones and focuses on the areas that has not been explored in detail as discussed in D.3.

The approach presented in this paper differentiates by exploring both the data transformations and data-to-memory assignment aspects in the presence of a platform with dynamically configurable memory blocks and bus widths. Moreover, many methods for source code transformations, and especially loop transformations, have been proposed in the memory management context. These methods are fully complementary to our focus on data-to-memory assignment and should be performed prior to our step. The contribution of the proposed work is the presentation of a combined approach that investigates the interleaving and memory mapping options for a reconfigurable SIMD architecture.

To illustrate the sub-optimal utilization of SIMD architectures using conventional allocation and assignment of data, the simple example of Alg. 3 is used. In this example, we assume that the desired result is always the sum of 4 elements from arrays *A, B, C and D*. The access pattern shows an irregularity, as a result of the iteration index. For every group of four sequential array elements, there is only one used for the calculation of the result and the other three are skipped. An intuitive interleaving optimization is the interleaving of the arrays *A, B, C and D*, in order to generate sequences of elements that are all useful on the calculation of the *result* variable. A full interleaving exploration could reveal several options to produce larger sequences of array elements that are needed during the execution of Alg. 3. For example, the interleaving within the combined array $(A|B|C|D)$ can result in a sequence of 8 accessed elements. The sequence of 8 elements is achieved by combining every forth line of the combined array, i.e. line 0 and line 4 are placed next to each other resulting in a sequence of 8 useful elements.

Algorithm 3 Motivational Example Algorithm

```
1: $N \leftarrow 100$
2: **for** $i = 0 \rightarrow N$ **do**
3:     $result(i) \leftarrow A(i) + B(i) + C(i) + D(i)$
4:     $i \leftarrow i + 4$
5: **end for**
```

The initial data representation for the arrays *A, B, C and D* is shown in Fig.D.1(a). The array elements that are accessed during the execution of the motivational example are marked with colors, in contrast to the elements that are not accessed. The array elements are drawn in 25 lines with 4 elements on each line only for a better visual representation. At this point there is no mapping to the memory architecture and no constraint regarding the number of lines or their width in the memory design. The interleaving of the arrays *A, B, C and D* is shown in Fig.D.1(c). Each line of the new interleaved array contains one elements from the initials arrays. For example, the first line consists of the elements *A(0), B(0), C(0) and D(0).* The result of the interleaving is the construction of blocks consisting of 4 colored elements followed by blocks with 12 redundant elements.

Let us now consider possible data-to-memory mappings for the initial and interleaved array elements. We assume here that he memory architecture either consists of one single memory large enough to fit all four arrays or of four memory banks with a combined memory size large enough to fit the four arrays. The conventional approach with the initial set of data maps each array after each other in the single memory or maps each array in a separate bank. A possible data-to-memory mapping for the initial set of data using four banks is presented in Fig. D.1(b). The data-to-memory mapping for the constructed interleaved array is presented in Fig. D.1(d). The array is split into four parts and stored in the four different banks using the same memory architecture. Each of the lines of the interleaved array is stored in the bank corresponding to modulo 4 of the array line. For example, line x is stored in bank x mod 4. As a result only one forth of array A can be found in the first bank in contrast to the non-interleaving case, in which the whole array A is mapped in the first bank.

A quick estimation for the difference in the number of memory accesses between the two approaches presented in Fig. D.1 can be made. The target architecture for the current work consists of a reconfigurable clustered memory architecture, a processing element that supports SIMD FUs and a wide bus between them. The architecture is presented in detail in Sec. D.4. The motivational example uses a system architecture with an SIMD ADD FU that performs operations over 4 array elements at a time and a memory to processor path that has a width of 4 elements. Each array element is assumed to have the size of one word. The register file at the processor level can only store the iteration variable i and 5 elements, which are the variables *result, A, B, C and D*. Without interleaving of data the number of memory accesses for each loop iteration is 4, because the elements *A(i), B(i), C(i) and D(i)* are stored in different lines and different memory banks

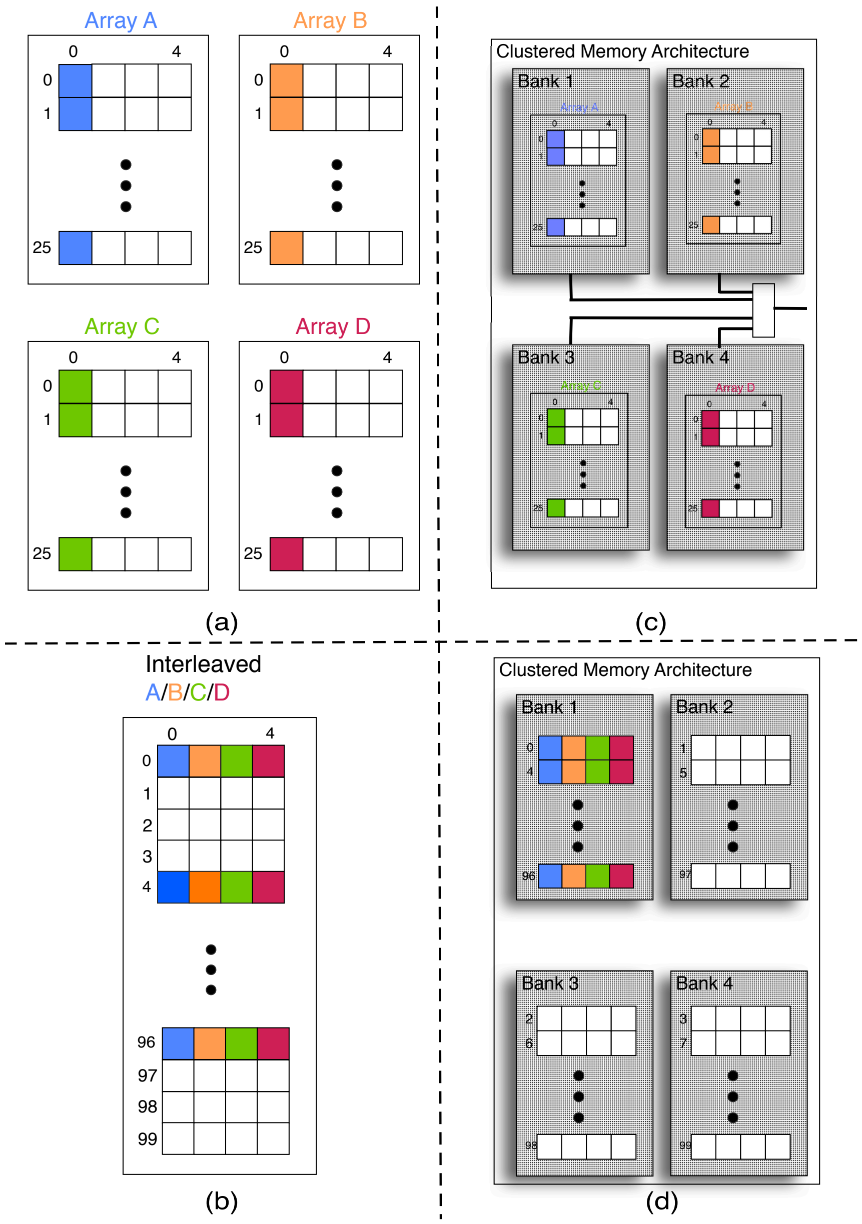


Figure D.1: Motivational Example. Representation of the initial set of data (a) and the interleaved set of data (b). Data-to-memory mapping for the initial (c) and the interleaved data (d) on a clustered scratch-pad memory architecture.

or one larger bank. The overall number of memory accesses is:

$$\mathit{25\ loop\ iterations} \times \mathit{4\ accesses\ per\ iterations} = \mathit{100\ memory\ accesses.} \tag{D.1}$$

After performing the data interleaving exploration there is only one memory access for each loop iteration, because the elements *A(i), B(i), C(i) and D(i)* are stored in one memory line in the same bank. The overall number of memory accesses is:

$$\mathit{25\ loop\ iterations} \times \mathit{1\ accesses\ per\ iteration} = \mathit{25\ memory\ accesses.} \tag{D.2}$$

A quick estimation for the difference in the energy consumption between the approaches presented in Fig. D.1 can be calculated using a simple energy model for the memory banks. We assume that the energy cost of accessing a memory bank the size of one array is 1E, while the average leakage energy in the time needed for the execution of one loop iteration is 0.3E. The corresponding access and leakage numbers for a four times larger memory that can fit all the data together is 3E and 1.2E, respectively. The numbers reflects the fact that as a first order approximation access energy increases sub-linearly with increased memory size, while leakage increases linearly with memory size. The energy consumption for a large memory with non-interleaved data is:

$$100 \times 3E + 100\times\ 1.2E = 420E. \tag{D.3}$$

In this case there is a performance loss and an increased leakage energy, because all the accesses has to be done sequentially on the same memory. The energy consumption for a four bank memory architecture with non-interleaved data is:

$$100 \times 1E + 4\times\ 100\times\ 0.3E = 220E. \tag{D.4}$$

The energy consumption for a large memory with interleaved data is:

$$25 \times 3E + 25\times\ 1.2E = 105E. \tag{D.5}$$

Finally, the energy consumption for a four bank memory architecture with interleaved data is:

$$25 \times 1E + 1\times\ 25\times\ 0.3E = 32.5E. \tag{D.6}$$

It should be noted that the leakage energy in the last case is considered only for the one bank that is always accesses. The memory banks that store redundant data are assumed switched to retention mode to minimize leakage. The simple estimations presented in equations (1)-(6) shows that a pure interleaving optimization results in a 75% reduction in the number of accesses. A memory optimization based on an energy efficient four bank clustered memory architecture can also give an important energy reduction. The energy gains are even greater when interleaved data are mapped to an efficient memory architecture. The observations for this simple example motivates the further exploration of the data interleaving and the data-to-memory mapping for applications with irregularities in their accesses.

D.3 Related work

Many studies have been published in the area of data layout transformations and memory management. This work differentiates by presenting an integrated approach that combines the two areas. The combination is performed in a systematic way and the final result is better than the sum of the two individual optimizations. The data transformations on the application level are studied in combination with the data-to-memory mapping on the hardware level. An overview of the related work on those two areas is presented.

Data layout optimizations aim to arrange data in memory with the objective of improving system performance and energy through means such as reduced memory access count or reduced cache misses. Several generic data layout techniques have been explored by researchers at various levels in the memory hierarchy. In [51], cache partitioning, a layout technique for arrays, maps each array to different cache partitions to reduce conflicts. In [57] authors address the problem for cache miss reduction by evaluating the tiling size for arrays and merging the arrays appropriately for each loop nest. Memory Hierarchy Layer Assignment (MHLA), an optimization methodology for data memory hierarchy presented in [56], determines for each data set an appropriate layer in the hierarchy and type of memory (single/dual port) taking data re-use into account. The strategy in [100] partitions the variables to scratch-pad memory and DRAM to minimize interference between variables. The cache-oriented techniques proposed earlier by researchers in [121] do not find a straightforward application in our context, where the target system hardware consists of a scratch-pad memory and a SIMD functional unit. The current work explores the assignment of data to the memory and the effect of different interleaving decisions on the overall energy consumption.

Several techniques for designing energy efficient memory architectures for embedded systems are presented in [76]. The current work differentiates by employing a platform that is reconfigurable at run-time. In [93] a large number of data and memory optimisation techniques, that could be dependent or independent of a target platform, are discussed. Again, reconfigurable platforms are not considered. The authors in [10] present a methodology to generate a static application-specific memory hierarchy. Later, they extend their work in [9] to a reconfigurable platform with multiple memory banks.

In [16] the authors tackle the problem of sub-optimal data structure layouts in GPUs with a large number of parallel cores, especially for programs that are designed with a CPU memory interface in mind. An API is presented, that allows programmers to improve CUDA programs by optimizing memory mappings in order to increase the efficiency of memory accesses. The main differences of the current work are the platform and the types of code transformations. We focus on an SIMD CPU and a dynamic clustered scratchpad memory compared to multicore GPUs and a static memory. We differentiate by focusing on interleaving as the preferred code transformation, being suitable for the target applications. Another work that discusses memory layout for GPUs is presented in [111]. The authors focus on off-chip DRAM memory optimization using a number of data layout transformations. We differentiate by focusing on the memory closest to the CPU. We also study data interleaving while the main focus in [111] is the transformations that increase data parallelism, which is more important for their multicore GPU architecture. In [2] a number of common data reorganization operations such as shuffle, pack/unpack, swap, transpose, and layout transformations are presented. The goal is to study the cost of applying these operations in the memory at run-time. The target memory is 3D-stacked DRAM and additional hardware is employed in order to efficiently perform the reorganization operations with a low overhead. Apart from the different type of platform, the current work differentiates in the type of data reorganizations and the mapping of the reorganized data to the scratchpad memory at compilation time.

The authors in [1], [55] and [71] present methodologies for designing memory hierarchies. Design methods with main focus on the traffic and latencies in the memory architecture are presented in [17], [40], [58] and [95]. Improving memory energy efficiency based on a study of access patterns is discussed in [61]. Application specific memory design is a research topic in [105], while memory design for multimedia applications is presented in [87]. The current work differentiates by presenting both a reconfigurable platform and the necessary data interleaving to better exploit the platform. The discussed related work is not designed to handle dynamic variations on memory requirements due to employment of a static memory platform. Thus, it is expected to provide poor results when applied to the dynamic applications targeted in the current work.

The current work combines and expands in a non-trivial way, the interleaving exploration presented in [106] and the data to memory mapping methodology presented in [28]. In [106] a memory dominant application is chosen, which normally suffers from data memory organization bottlenecks while implemented on conventional architectures. A data interleaving approach is employed in order to improve memory energy by reducing memory accesses. Significant reduction in data memory energy consumption can be achieved if array data are interleaved, with no performance overhead. In [28] a memory-aware system scenario methodology is presented. The variations in memory needs during the lifetime of an application are studied in order to optimize energy usage. Different system scenarios capture the application's different resource requirements that change dynamically at run-time. Many possible memory platform configurations and data-to-memory assignments are studied as system scenario parameters, which lead to a large exploration space.

The contribution of the proposed work is the development of a combined approach that investigates the interleaving and memory mapping options for a reconfigurable SIMD architecture. The current work combines and expands in a non-trivial way, the interleaving exploration presented in [106] and the data to memory mapping methodology presented in [28]. The current work is more than a simple application of the two approaches, one after the other. Such an approach cannot always guarantee a viable solution. The reason is that for each different interleaving solution, there are a number of constraints on the placement of the data into the memory due to hardware limitations. If the constraints are not propagated into the data-to-memory mapping step, the final solution suffers from data conflicts. Different interleaving solutions introduce different constraints. Therefore, there is a need to develop an integrated methodology to achieve the improvements of both the approaches.

D.4 Target Architecture and Energy Models

Selection of target platform is an important aspect of the exploration. The motivational example in the previous section shows that the available choices on the memory models have a great impact on the overall energy consumption. The key feature needed in the platform architecture is the ability to efficiently support different memory sizes that are suitable for different data structures. A generic architecture is presented in Fig.D.2. The scratch-pad memory consists of up to four memory banks. For more complex architectures the interconnection cost should be considered and analyzed separately for accurate results. Although power gating can be applied to the bus when only a part of a longer bus is needed, an accurate model of the memory wrapper and interconnection must be developed, which is beyond the scope of the current work. The SIMD vector FU is assumed to perform instructions on multiple data.

Improving temporal locality of data accesses in cache is indeed important. The proposed architecture uses scratch-pad memories, however, and no cache memory is included in the current study. Software controlled allocation is a significant feature for the current methodology, as the allocation of data can be fully determined by the designer at design-time. The basic principles of the methodology are still applicable for hardware controlled caches, but some modifications would be needed to deal with the automatic resolution of hits and misses. Temporal locality and data reuse are taken into consideration during the interleaving exploration.

The methodology explores different interleaving and data to memory mapping options for the reconfigurable architecture. The optimal sizes are found based on the sizes of the data after the interleaving exploration. The memory banks can operate independently and have different sizes. The explored data lengths for the vector FU and the connections are 4, 8 and 16 elements. The exploration is based on the available system models and results in the final system architecture. Thus, the accuracy of the models influence the decisions taken during the design phase.

D.4.1 Memory Models

The dynamic memory organization is constructed using commercially available SRAM memory models (MM). For those models delay and energy num-

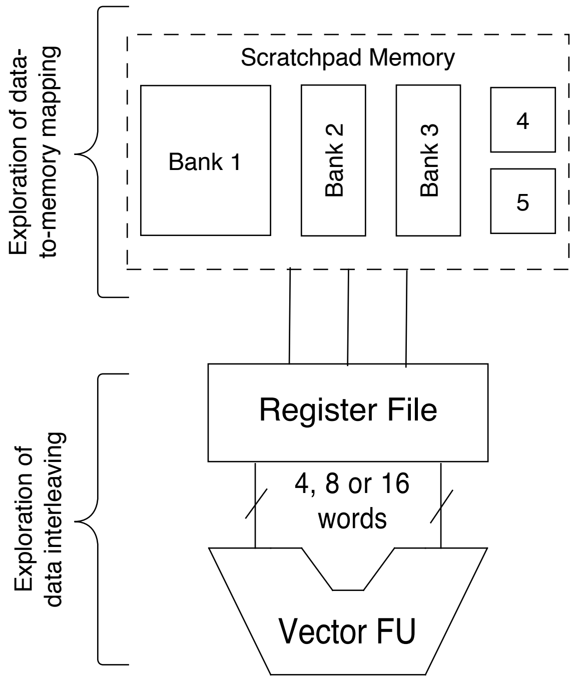


Figure D.2: Exploration options and system knobs depending on a general platform architecture

bers are derived from a commercial memory compiler. The commercial memory compiler is part of DesignWare Memory Compilers provided by Synopsys. The logic libraries support a wide range of foundries and process technologies from 250nm to 28nm. The memories are optimized for low power, high performance and high density. A 40nm library was chosen for the current work. Of confidentiality reasons we are not allowed to reveal details of the compiler or process and the presented numbers are relative. In addition, experimental standard cell-based memories (SCMEM) [82] are considered for smaller memories due to their energy and area efficiency for reasonably small storage capacities, as argued in [80]. The standard cell-based memories are synthesized using Cadence RTL compiler for TSMC 40nm standard library. Afterwards, power simulations on the synthesized design are carried out using Synopsys PrimeTime, in order to obtain energy numbers. Both MMs and SCMEMs can operate under a wide range of supply voltages, thus support different operating modes that provide an important exploration space.

- Active mode: The normal operation mode, in which the memory can be accessed at the maximum supported speed. The supply voltage is 1.1V. The dynamic and leakage power are higher compared to the other modes. Only in active mode the data are accessible without time penalties, in contrast to light and deep sleep modes. In this work all the memory accesses are performed in active mode.

- Light sleep mode: The supply voltage in this mode is lower than active with values around 0.7V. The access time of the memory is significantly higher than the access time in active mode. Switching to active mode can be performed with a negligible energy penalty and a small time penalty of a few clock cycles (less than 10). Data is retained.

- Deep sleep mode: The supply voltage is set to the lowest possible value that can be used without loss of data. This voltage threshold is expected to be lower for SCMEMs than MM models and can be as low as 0.3V. The number of clock cycles needed for switching to active mode is higher compared to light sleep mode, typically in the range of 20 to 50 clock cycles depending on the clock speed. Consequently, the speed of the PE and the real-time constraints of the applications have to be taken into consideration when choosing light or deep sleep mode at a specific time.

Table D.1: Relative dynamic energy for a range of memories with varying capacity and type

Type	**Lines x wordlength**	**Dynamic Energy**		Active from	
		Read[nJ]	Write[nJ]	Deep[uJ]	Light[uJ]
MM	32 x 8	41.8	32.4	0.223	0.031
MM	32 x 16	67.9	58.9	0.223	0.031
MM	32 x 128	433	431	1.42	0.168
MM	256 x 128	448	460	1.70	0.171
MM	1024 x 128	511	575	2.81	0.179
MM	4096 x 128	960	457	9.01	0.457
SCMEM	128 x 128	250	8	1.51	0.045
SCMEM	1024 x 8	17	6	0.325	0.021

- Shut down mode: Power-gating techniques are used to achieve near zero leakage power. Stored data is lost. The switch to active mode requires substantially more energy and time. However, switching unused memories to this mode, providing that their data are not needed in the future, results in substantial energy savings.

The necessary energy and power information is available for the memory models. Relative values for a subset of them is presented in Table D.1. It is clearly shown that the choice of memory units has an important impact on the energy consumption.

D.4.2 Functional Unit Models

The processing of the interleaved data is performed in the part of the architecture consisting of the SIMD FU and the central vector register file, as shown in Fig.D.2. The bus between the memory part and the processing part is 256 bit wide. The wide bus allows testing for word length of 4, 8 and 16 elements, assuming that each array element is 16 bit wide. The different word lengths provide different design options for the vector FU. This processor belongs to the class of coarse-grain reconfigurable array (CGRA) processors and is described in more detail in [70]. The HDL model of the processor is synthesized using Cadence RTL compiler [13] and the energy numbers are extracted using Synopsys Primepower [60].

For efficient utilization of the vector FU, the register file has a wide

Table D.2: Relative dynamic energy for different FU models

Type of FU instruction	Width	Dynamic Energy [J]
ADD	1	5.70E-08
ADD	4	2.28E-07
MULT	1	2.48E-07
MULT complex	1	5.33E-07
MULT	4	1.03E-06
MULT complex	4	1.95E-06

interface with the clustered scratch-pad memory. Since the target architecture is a configurable processor that can be customized in various ways, the standard evaluation and execution mechanism is to run the programs on a processor simulator. An XML based language is used to describe the architecture, and a cycle-accurate simulator of the processor is used to simulate the generated code on the architecture. The XML provides a structural way of describing the architecture presented in Fig. 2 including the different components, the parameters of each component, and the relationship between them. The XML description generates a graphical representation of the architecture and is the input for the simulator as presented in [79]. The chosen simulator is developed for coarse-grained reconfigurable architectures and is suitable in our case, because of the dynamic parameters of our architecture.

The energy consumption information for different types of instructions and different widths is available for the FU models. The width of the FU corresponds to the number of pairs of array elements on which the specific instruction is performed. Relative values for a subset of them is presented in Table D.2. It is clearly shown that the wider FUs have a significantly higher energy consumption compared to FUs that operate on only two element. Thus, it is important to achieve high utilization of wider FUs to achieve both performance and energy gains.

D.5 System Design Exploration Work-flow

As motivated in the previous sections a systematic exploration of the interleaving options and the data-to-memory mapping possibilities is necessary. The input to the work-flow is the application code and the output of the

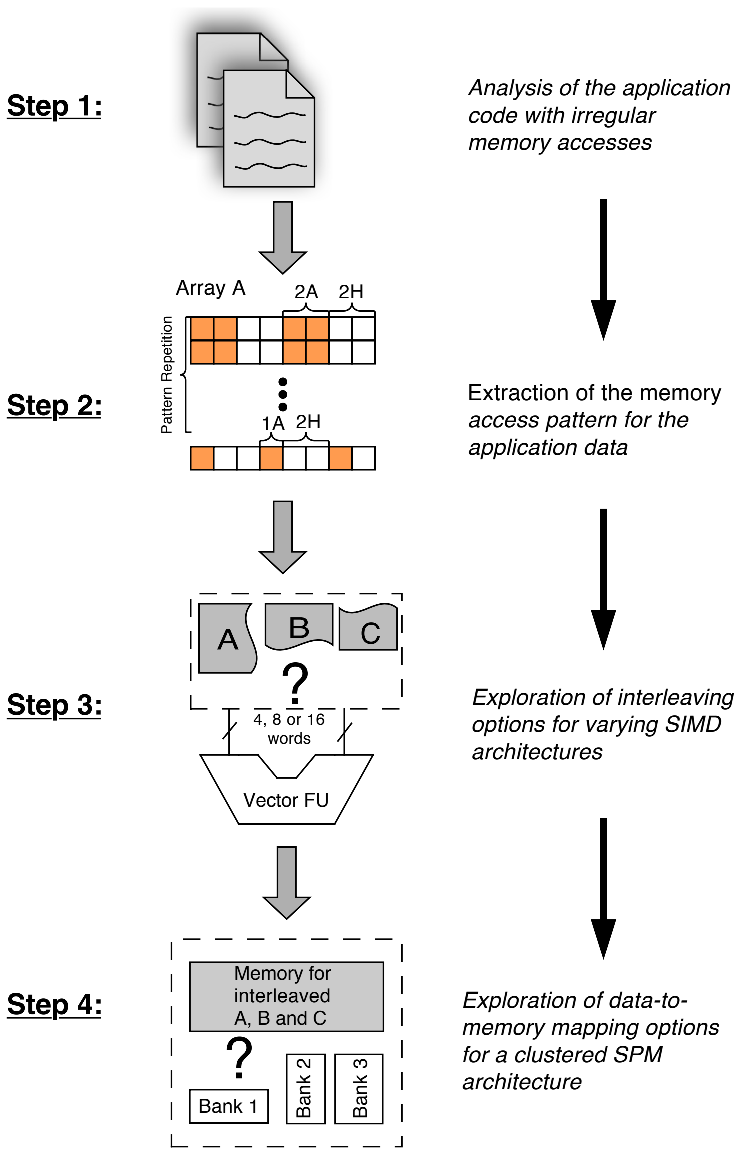

Step 1:
Analysis of the application code with irregular memory accesses
Array A
2A
2H
Pattern Repetition
1A
2H
Step 2:
Extraction of the memory access pattern for the application data
A
B
C
?
4, 8 or 16 words
Vector FU
Step 3:
Exploration of interleaving options for varying SIMD architectures
Memory for interleaved A, B and C
?
Bank 1
Bank 2
Bank 3
Step 4:
Exploration of data-to-memory mapping options for a clustered SPM architecture

Figure D.3: Methodology steps

exploration is an efficient solution for the interleaving of the data and a mapping to a reconfigurable system architecture based on the available models presented in Sec.D.4. The exploration space consists of the potential combinations between the different arrays and the different scratch-pad memory architectures, which combine memories of different types and sizes.

The overall work-flow is presented in Fig.D.3. The first step of the methodology is the analysis of the application code. The methodology is applicable to any application. However the applications that can benefit more from the proposed methodology are the ones with holes in their access patterns. The second step is the extraction of the access pattern for each data structure present on the application code. The third step is the interleaving exploration, which explores all the different options for the rearrangement of the data. The aim of this step is the construction of compact sets of data by using defined operations between the available access patterns. The goal is to better utilize the different width of the vector FUs. The result of the interleaving is a set of data with a reduced number of holes. The forth step is the mapping of the interleaved data set to the clustered memory architecture.

Some significant changes are needed in order to combine the different techniques in a functional work-flow. The interleaving and the data-to-memory mapping exploration steps have to be modified and a simple concatenation is not sufficient. The results of the interleaving exploration are given as constraints to the mapping exploration. The constraints stir the exploration in a potentially different solution compared to the result of the data-to-memory mapping exploration without the interleaving constraints. The interleaving exploration finds solutions for the different bus widths and then all the different solutions are propagated to the mapping exploration. The mapping exploration takes all the interleaving constraints into consideration. Thus, the data-to-memory mapping solutions using the constraints are in general different from the solutions provided in a conventional approach. The propagation of the constraints is explained in more detail later.

The application is fully analyzed at design-time, because of the time consuming nature of the task. The access patterns are extracted in software and the possible interleaving options are explored. The mapping to the specific target architecture is also fully decided at design-time. The data-to-memory mapping exploration takes into consideration the platform and code limitation and proposes a mapping without data and bank conflicts. The employment of a scratchpad memory architecture is crucial for this. In contrast to cache memory systems, in which the mapping of data elements is done at run-time, in scratchpad memory systems the mapping is performed by the programmer or the compiler [53]. The address mapping follows the principles presented in [21] and [107]. Unlike the cache memory, the scratchpad memory does not need tag search operations and it is the responsibility of the programmers or compilers to correctly allocate code and data on the scratchpad memory [109]. This is possible for small embedded systems designed to run one or a limited set of specific applications. For other types of systems and applications, a cache memory can be the preferred solution as the detection of a cache hit or miss is done automatically and any conflicts can also be resolved by the platform at run-time. In our case, the application and the memory system are fully analyzed and the allocation of data to a scratchpad memory can be easily done and offers an energy efficient solution.

D.5.1 Formal Model Representation of Access Patterns

A representation model for the memory access patterns is employed in order to formally present each step of the methodology. The model presented in [65] is a generic model suitable for irregular iteration spaces on arrays. The irregularities are created by the application code access statements in a conditional loop structure.

When an array element is accessed during the code execution, it is represented with an A(Access). Otherwise, there is a hole in the access pattern represented with an H(Hole) as shown in Fig.D.4. The sequence of accesses and holes is usually repeated periodically, because normally the loop conditions are not totally random. Thus we can define the frequency of each access pattern. The analysis of the application code results in the access patterns and their corresponding frequencies, which is the necessary input for the next step in the work-flow.

D.5.2 Data Interleaving Exploration

Interleaving is a data layout transformation for combining the storage of multiple arrays, so that blocks of data from different arrays are stored contiguously, in order to achieve spatial locality in memory accesses. By interleaving we are able to group the data to be accessed and thus reduce the

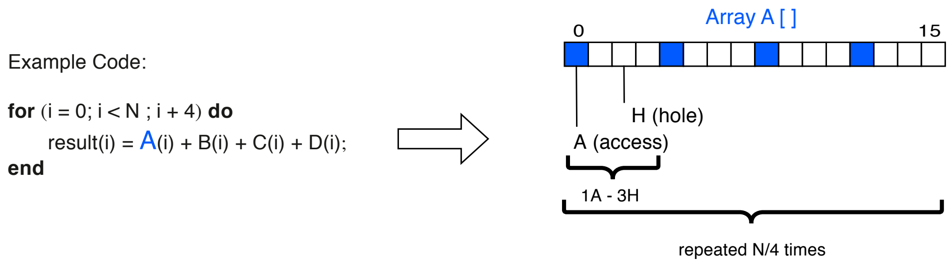


Figure D.4: Extraction of access pattern from application code

number of memory accesses for accessing them. The basic principles of the performed interleaving exploration are presented in [106].

The employed model of access pattern representation is accompanied with a suitable algebra presented in [67]. The operations defined help the designer explore different interleaving options. The goal of the process is to rearrange the data in an efficient way to increase the number of sequential accesses. A simple example is presented in Fig.D.5. Arrays A and B are interleaved by interchangeably storing an element of each array in the new array. The access patterns of arrays A and B are combined in a new access pattern, which has two accessed elements placed consecutively. The use of access patterns and the theory for calculating the access patterns of different combinations enables an extensive exploration of the possible data interleaving options.

The impact of the data interleaving exploration on the number of memory accesses is significant. When there are holes in the access pattern and the data are organized in index order, each memory access results in a small amount of useful data due to the presence of holes. In contrast the re-organization of the data provides a sequence of useful data without many holes between them. Thus, a single access to the memory results in a higher number of useful elements. The overall number of memory accesses is reduced, as each access has a higher utilization.

This idea is illustrated in Fig.D.5. The number of memory accesses for each case can be calculated by assuming that each memory access loads four array elements, i.e. the word length for the memory and bus architecture is four elements. Each time an element from the array A or B is needed, the memory access returns four elements, from which one is the useful and the other three are not used by the running code. In the case of the interleaved array, each memory access returns four elements, from which two are useful and two redundant. Thus, the overall number of memory access in the

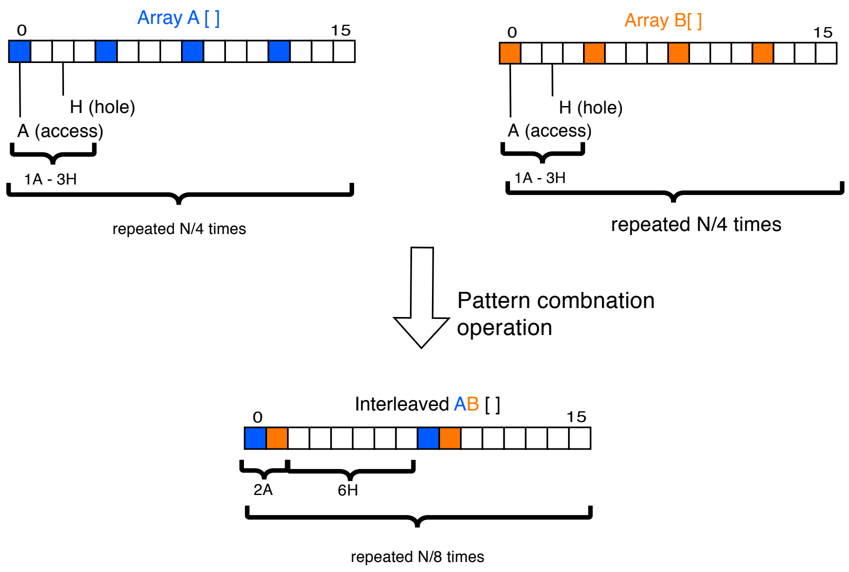


Figure D.5: Example of combination between two arrays and their access patterns

second case is reduced by half compared to the first case.

D.5.3 Data-to-Memory Mapping Exploration

Given the input from the previous step, we explore the mapping of the interleaved data to the memory architecture. A clustered memory organisation with up to five memory banks of varying sizes is explored. The limitation in the number of memory banks is necessary in order to keep the interconnection cost between the processing element (PE) and the memories constant through exploration of different architectures. There are two main reasons for exploring architectures up to five memory banks. Firstly, the energy gains achieved by increasing the number of memory banks in the memory architecture are nearly saturated even for five banks. In [28] a group of different applications were studied with regard to their energy consumption on a clustered memory architecture consisting of up to five memory banks. The results show that depending on the application, the energy gains start to saturate after adding a third or a fourth bank and become insignificant when adding a fifth bank. Thus, for most applications a memory architecture with five memory banks already provides the necessary reconfiguration options. Secondly, the overhead increases exponentially with the number of memory banks, due to the increased complexity of the memory architecture. Therefore, memory architectures with six or more banks are typically not efficient options due to the high overhead and the low energy gain.

The decision to use memory banks with varying sizes on the clustered memory organization increases the reconfiguration options and consequently the potential energy gains. In general, smaller memories are more energy efficient compared to larger memories banks. However, in some cases large memory banks are needed in order to fit the application data without the need for too many small memories causing complex interconnects. The goal is to use the most energy efficient banks to store the interleaved data.

The exploration space consists of different sizes and types of memory banks. The goal is to select the optimal set of memory banks and to make the optimal decision regarding the mapping of the data to the different memory banks. The parts of the interleaved data that consist mostly of useful elements are mapped into memory banks with low energy per access but at the same time with the necessary access time. The parts of the interleaved data that consist of access holes and rarely accessed elements are optimally mapped into memory banks with energy efficient retention states. In both cases the size of the memory banks should be adequate to fit the stored data but at the same time as small as possible to avoid area and energy penalties.

D.5.4 One way constraint propagation

The interleaving decisions influence the data-to-memory mapping decisions and vice versa. For example, the decision to interleave two arrays $A\ and\ B$ have an impact on the freedom of mapping the interleaved array $A|B$ on the memory banks, because of the difference in size and structure. The data-to-memory mapping decisions affect the interleaving decisions in a similar way. Assuming that the mapping is performed first using the initial data the following interleaving options are reduced. For example, the decision to map two arrays on different memory banks removes the option to interleave them later. Optimizing both the interleaving and the memory mapping at the same time results in a large and inefficient loop of constrain propagations between the two exploration phases.

We choose to perform the interleaving exploration step first and then propagate the most efficient interleaving options to the data-to-memory mapping step. The code and data transformations should be performed earlier to avoid omitting possible solutions as justified in [14]. Thus, the interleaving decisions are propagated as constraints on the mapping exploration phase. The constrains consist of the arrays that are interleaved, the new access pattern and the width of SIMD architecture for which the specific interleaving solution is optimal. The mapping step is performed based on these constrains.

D.6 Applications and Experimental Evaluation

D.6.1 Benchmark Applications

Array Interleaving is a data layout transformation for combining the storage of multiple arrays, so that blocks of data from different arrays are stored contiguously, with the objective of reducing the number of memory accesses through better spatial locality. [107]. The target applications on the current work benefit most from the proposed methodology, because they are characterized by having access patterns with holes. Interleaving is a widely used technique that fits the goal of generating more compact sets of data. An important contribution of the work is the combination of the interleaving optimization with data to memory mapping. Another important advantage of interleaving transformations is the low addressing overhead. In more detail, interleaving results in a more regular global access by reducing the number of holes without increasing significantly the addressing for accessing the individual arrays afterwards. Although more complex data layout transformations may reduce the holes in the access patterns even more, they have a negative impact on the complexity of addressing and access overhead. Having shown the benefits of this, future work can include extending the methodology to be compatible with additional layout optimization techniques.

Applications with holes in their access pattern can be found on several application domains. The current study includes representative candidates from three different domains, namely the multimedia, the wireless and the equation solver domains. Suitable applications can also be found on other areas. An overview of the tested benchmark applications is presented below:

1. Motivational example is the very simple example presented in Sec. D.2. It uses four arrays that all have holes in their access pattern, namely

1A-3H. One element from each array is used to calculate an intermediate result on every loop iteration. The interleaving is easy because the four arrays have the same pattern. Further interleaving within the array can result in even longer sequences of useful elements. The access pattern is repeated for every four elements (1A-3H), so the scaling for the different word lengths (4, 8 and 16) is expected to be good.

2. Successive Over Relaxation (SOR) is a method for evaluating partial differential equations or solving a linear system of equations. The SOR benchmark has an access pattern with more holes and different distribution of them. The interleaving exploration provides sequences of three or six sequential useful elements. Thus, the utilization on the SIMD architecture is expected to be lower. This application is a representative example from the equation solver domain.

3. FFT benchmark has holes in the access pattern during the access of the pilot matrices. The interleaving exploration for FFT is presented in [106]. The number of matrices is higher and more interleaving options are present. However, the interleaving cannot provide an acceptable solution for 16 sequential useful elements. This application is an representative example from the wireless domain, because FFT is widely used on wireless applications.

4. Motion estimation benchmark is a dynamic algorithm that results in different access patterns based on the identification of the moving objects. The static parts are not accessed resulting in holes in the accesses and the interleaving aims to minimize those parts. The interleaving exploration provides alternatives for all the tested word lengths. This application is an representative example from the multimedia domain.

D.6.2 Results

The design exploration is applied to the chosen application benchmarks and energy numbers are derived based on the described target platform. The energy numbers are calculated both for the memory and the SIMD architecture presented in Sec.D.4. Four approaches are explored and the corresponding energy consumption for each of them is calculated.

- *No optimization.* In this case there is no interleaving exploration and the memory architecture consists of one large memory bank. All the data are mapped on the memory bank without any optimization.
- *Memory Size Optimization.* In this case there is no interleaving exploration and the memory architecture consists of five memory banks. The optimal size for the memory banks and the optimal mapping of the non-interleaved data on them is explored. The number of memory accesses is the same as in the previous approach. However, the data is mapped on an efficient clustered memory architecture.
- *Interleaving optimization.* In this case the interleaving exploration step is performed and the memory architecture consists of one large memory bank. The optimal interleaving decision is found and applied to the data, so the locality of useful data is increased. However, all the data are mapped on one large memory bank. The number of accesses is significantly reduced by the interleaving step, but the energy per access is kept high due to lack of data mapping to an efficient clustered architecture.
- *Integrated co-exploration.* In this case the co-exploration of both the interleaving and the data-to-memory mapping optimizations is performed. Both he number of memory accesses and the energy per access are reduced.

Motivational Example

The normalized energy consumption for the motivational example is presented in Fig.D.6. The four different approaches are normalized using the monolithic approach without any optimization. Energy results for this benchmark are presented for a bus width of 4, 8 and 16 elements, which means that every memory access loads/stores 4,8 or 16 elements. The interleaving exploration proposes three different solutions for an SIMD architecture width of 4, 8 and 16 elements respectively. The data access patterns of the three interleaving decisions are propagated as a set of constrains to the mapping step. The mapping step explores all the different data-to-memory mapping options and proposes the most energy efficient memory organization for each of the three interleaved data solutions.

The application code is perfectly suited for interleaving as discussed in Sec.D.2. Thus the interleaving exploration has a greater impact than the memory size optimization. However, the integrated approach optimizes the energy consumption even further. The application is suitable for higher

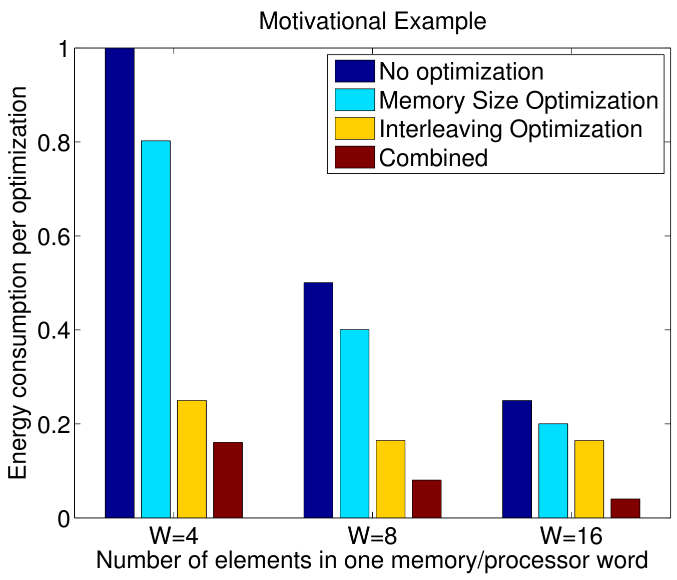


Figure D.6: Motivational Example

bus widths and there are important gains while moving from 4 to 8 and 16. The increase in the architecture width result in significant gains even without any optimization. The gains are approximately 50 and 70% for a width of 8 and 16 compared to a width of 4. This is explained by the fact that the accessed array elements are close enough even without data transformations and the existence of holes in the access pattern. By fetching 8 and 16 elements from the memory, the number of useful elements is two and four times more. Studying the motivational example in Fig.D.6(a) for a width of 16, each access results in 4 lines with one colored element each.

The interleaving optimization results in greater improvements compared to the memory optimization. This is explained by the nature of the application that offers good interleaving options for larger words, i.e. higher values of bus width. The memory optimization is around 20% lower than the monolithic approach for any width. The interleaving optimization results in more than 70% lower energy for a width of 4 and more than 80% for a width of 8 and 16. The interleaving of the arrays provides perfectly compact sets of data as shown in Fig.D.6 and the interleaving cannot improve more for higher values of width. The great impact of the combined approach is better illustrated for a width of 16. In this case the two optimizations alone report an energy gain around 20% and 35% compared to the non optimized

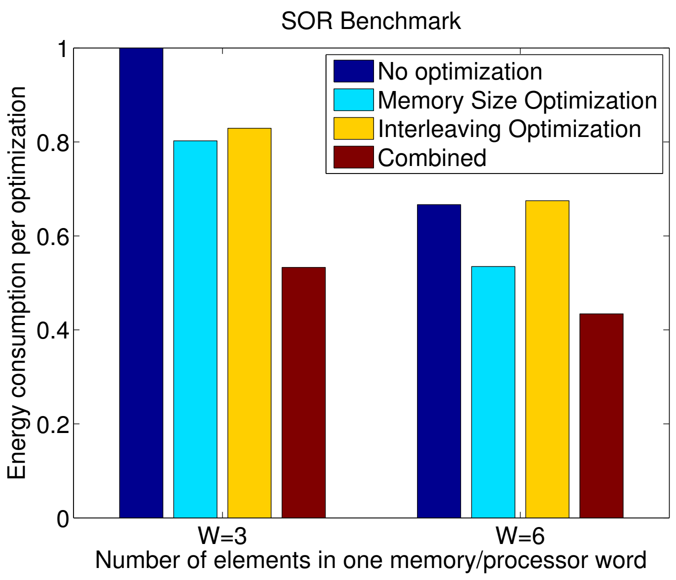


Figure D.7: SOR Benchmark

case for width of 16. The combined approach results in an energy gain of 84% on the same case.

SOR Benchmark

Algorithm 4 Code snippet from the SOR benchmark

```
1: for $j = 0 \to N$ do
2:     ...
3:     $resid \leftarrow ... + c(i)(j) \times u(i)(j-1) + d(i)(j) \times u(i)(j) + e(i)(j) \times u(i)(j+1) + ...$
4:     ...
5:     $j \leftarrow j + 2$
6: end for
```

The normalized energy consumption for the SOR benchmark is presented in Fig.D.7. The four different approaches are normalized using the energy consumption of a system without any optimization as the base. Energy results for this benchmark are presented for a bus width of 3 and 6, although the architecture supports bus widths of the power of 2. This means that every memory access loads or stores 4 or 8 elements, but only up to 3 or 6 array elements are used in the algorithm. This limitation is due to the nature of the application code that adds elements from 3 arrays to 3 sequential elements of another array. Alg. 4 is a code snippet that demonstrates the considered data structures. The arrays *c, d and e* are potential candidates for the interleaving exploration. The interleaving exploration on the application code concludes that it is only possible to make sequential sets of 3 and 6 elements by interleaving the arrays *c, d and e.* The elements that are multiplied with them are already sequential.

The interleaving exploration has a small impact on the reduction of energy consumption for a bus width of 3 elements. This is due to the addition of a redundant element, in order to have a width of four that is supported by the architecture. The results are even worse for a bus width of 6 elements, because there is a need for two redundant elements to comply with a set width of eight. The mapping of the initial data to a clustered memory results in a reduction of around 20%, which is slightly better than the interleaving optimization. The reason is that the mapping of the initial data demands all the memory banks active and the memory is heavily accessed during the execution of the benchmark. The integrated approach exploit on a better way the few interleaving options and provides a better mapping of the interleaved data to the memory architecture. The final gain for a width of three is 47%, compared to 17% and 20% for the interleaving and the memory optimization respectively.

FFT Benchmark

The normalized energy consumption for the FFT benchmark is presented in Fig.D.8. The four different approaches are normalized using the monolithic approach without any optimization. Energy results for this benchmark are presented for a bus width of 4 and 8 elements, which are the two viable options provided by the interleaving exploration. Again, the integrated approach results in the lower energy consumption. The energy gains for a bus width of 8 are significant, because blocks of 8 elements can be constructed by interleaving of application's arrays.

Motion Estimation Benchmark

The normalized energy consumption for the motion estimation benchmark is presented in Fig.D.9. The four different approaches are normalized using the monolithic approach without any optimization. Energy results for this benchmark are presented for a bus width of 4, 8 and 16 elements. The application code provides good possibilities for interleaving and consequently the interleaving exploration has a greater impact than the memory size optimization. However, the integrated approach optimizes the energy con-

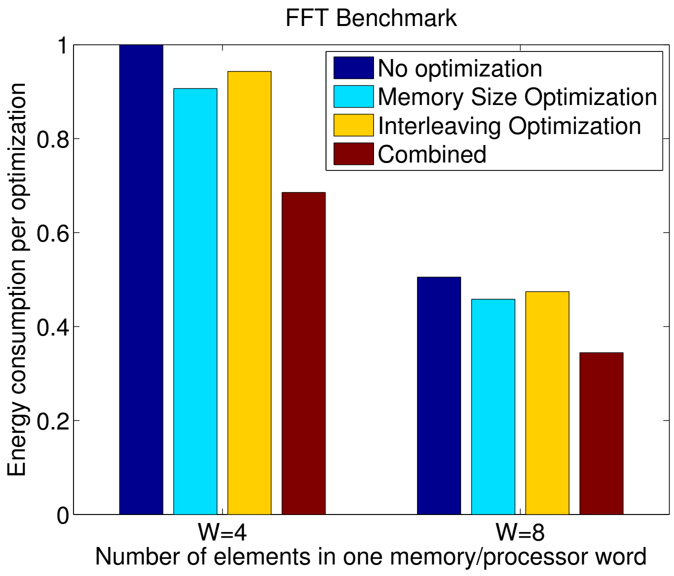


Figure D.8: FFT Benchmark

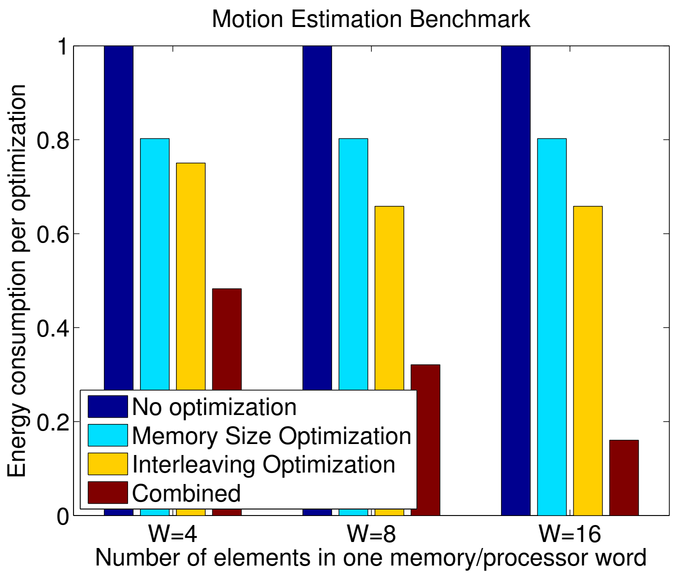


Figure D.9: Motion Estimation Benchmark

sumption even further. In this case the energy gains for increasing size of bus width are minimal when only one optimization is applied. This is explained by the nature of the application, which has large parts of redundant data spread out through the memory. Thus, the whole memory is active when the mapping is not performed on the compact interleaved data. The combined approach offers significant energy gains even for the highest bus width.

D.7 Conclusion

The scope of this work is to presents a methodology for efficient exploration of data interleaving and data-to-memory mapping options for SIMD (Single Instruction Multiple Data) platform architectures. Detailed energy models are presented for the studied architecture. The methodology focuses on reducing the overall energy consumption by reducing the number of memory accesses and the energy per access. The number of memory accesses is reduced by interleaving the application data to construct compact sets of sequential data. The energy per memory access is reduced by employing a reconfigurable scratch-pad memory architecture with multiple banks that can operate independently. A systematic way is presented in order to explore the different options that lead to the interleaving and data-to-memory mapping decisions. A wide range of applications is studied that allow us to draw conclusions about different kinds of dynamic behavior and their effect on the energy gains achieved using the methodology. The improvement in the total system energy consumption after efficient interleaving and mapping of data is between 40% and 80% for the studied benchmarks having the type of holes in their access scheme that benefit most from the methodology.

Appendix E

Technology scaling impact on the interconnection of clustered scratchpad memory architectures

Iason Filippopoulos, Francky Catthoor, Per Gunnar Kjeldsberg

Technical report

2015

Abstract

Power consumption is a key limiting factor in modern embedded devices. The memory architecture contributes significantly to the overall power consumption of the system. Among many proposed techniques, one effective system design approach to reduce the memory power needs is the design of a dynamically reconfigurable clustered memory architecture. The operationally independent memory banks provide an energy efficient platform, but come with an interconnection overhead due to the connections between the memory banks. Thus, there is a trade-off between the energy gains by increasing the number of memory banks and the increase in interconnection overhead. This work explores the future development of the interconnection overhead, as the interconnection cost is expected to increase while the process technology shrinks to 5nm. The current study employs both CAD tools with simulation results using the current technology and projections provided by institutions. We use predictive technology models supported by information from ITRS and IMEC's interconnect technologists. A model is developed that provide a sufficiently accurate estimate of the interconnection cost overhead for clustered memory architectures consisting of two to five memory banks in technologies ranging from 40nm to 5nm. The model shows that the interconnect overhead is as low as 10.9 % even for the most aggressive technologies. Hence a dynamically reconfigurable clustered memory architecture is a viable solution also for future designs, even if the optimal number of memory banks may be reduces as shown in an experiment with a representative real life application.

E.1 Introduction

Embedded systems normally rely on battery and their lifetime between charging is limited and dependent on the energy usage of the system. The power consumption can be divided between the processing elements and the memory subsystem. One efficient way to reduce the power consumed in the memory, when executing an application with dynamic memory needs, is to design a clustered memory architecture. A study of the efficiency and the potential gains of this approach is presented in [28]. In Fig. E.1 the two different approaches are presented. Alternative 1 to the left has a large static memory while Alternative 5 to the right has five smaller memory banks. The designer can choose other alternatives in between. The different memory banks can operate independently and adjust to the application requirements. When the memory requirements of the application are small, unused memory banks can be switched off to reduce the energy consumption, while the application still runs successfully on the system.

The main drawback of the clustered memory architecture approach is the need for extra interconnection circuitry. Unfortunately, it is expected that the interconnection networks will take up an increasingly significant portion of system power in the future. Thus, there is a need to obtain a detailed memory and interconnect energy model including the scaling impact. That will allow us to accurately incorporate the interconnection cost overhead in our studies to decide whether and when the power gains justify the use of a clustered memory architecture instead of a monolithic one. It especially also allows to identify the best trade-off working points between using more or less memory banks within the clustered approach. Without this, an overly optimistic distribution across too many banks would seem to provide the best energy for a given application requirement.

The scope of this work is to explore the future development of the interconnection overhead and develop a model that can provide a sufficiently accurate estimate of that overhead. System designers could use such a model to find the preferred memory architecture and the number of banks given the nature of the executed application and the target technology. The model is based on the current technology and available projections about the future technology. The current technology is available as libraries in CAD tools. The synthesis and the simulation of memory architectures similar to the one in Fig. E.1 provide useful results. The study of the future technology is based on the reports released by the International Technology Roadmap for Semiconductors (ITRS) [24] . The goal of the developed model is to

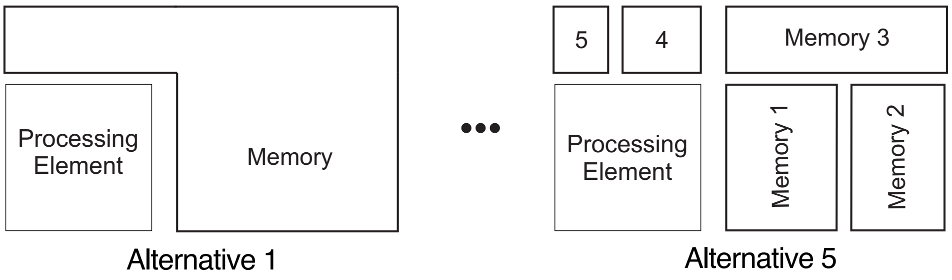


Figure E.1: The alternative clustered memory architectures ranging from one to five memory banks

provide a sufficiently accurate estimate of the interconnection cost overhead for clustered memory architectures consisting of two to five memory banks and a range of technologies from 40nm to 5nm.

The studied memory architecture is targeted to applications that significantly increase their energy efficiency when they are mapped on clustered memory architectures. These applications are characterized by having dynamic utilization of the memory organization during their execution. Suitable dynamic applications are available on several different domains. For example, a set of multimedia applications, which exhibit such a dynamic variation in memory requirements during their lifetime, is presented in [28]. A similar clustered architecture is employed in [29] for a bio-medical application as a suitable example. In general, smaller memories are more energy efficient compared to larger memory banks [109]. The distribution of data into the memory banks should allocate the most frequently accessed data to the most energy efficient memory banks, in order to maximize the potential gains.

The paper is organized as follows. Section E.2 surveys related work on on-chip interconnection with a focus on energy consumption and presents research work on technology scaling. Section E.3 presents the chosen synthesis work-flow and results for the target architecture using the current technology. In Section E.4 the scaling projections for the different parts of the target architecture are presented. The development of a model that can provide a sufficiently accurate estimate of the power overhead for the interconnection in clustered memory architectures is presented in Section E.5. Finally, conclusions are drawn in Section E.6.

E.2 Related Work

A comprehensive view of a class of interconnect architectures is presented in [68]. The authors examine the area, power, performance, and design issues for the on-chip interconnects. The main finding is that the interconnect should not be independently optimized but co-designed with the other components, i.e., memories and cores, in order to arrive to the best platform design. In [101] authors propose an interconnection network with high-performance for the communication between processors and memories in a cluster of processors. Another approach that focuses on the interconnection network between the memory and the processing cores is presented in [63]. The majority of the published work focuses on the interconnection between the processing cores in a system-on-chip. The current work differentiates by focusing on the interconnection between the memory banks.

Several examples of clustered memory architectures have been proposed. In [59] an adaptive scratch-pad memory is successfully used in order to handle the dynamic behavior of multimedia applications. In [119] a clustered memory architecture is employed and an algorithm is developed, which efficiently uses the memory banks to achieve the maximum energy saving while satisfying the given performance constraint. Our approach explores the effectiveness of similar architectures into the future, which is not studied before to the best of our knowledge.

A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies is developed in [115]. This tool uses the CACTI model and targets large general purpose DRAM and SRAM memory designs. However, the current work focuses on small scratch-pad memory designs, which cannot be successfully covered by the CACTI model. A circuit level analysis of the interconnect delay from the 10 nm node to the 7 nm node is presented in [92]. The authors in [18] analyzes the impact of interconnect variation at the system-level in terms of clock frequency based on a fast and efficient system-level design methodology. The current work differentiates by focusing on the energy impact and providing a high-level prediction model.

E.3 Current technology

E.3.1 Generic Work-flow

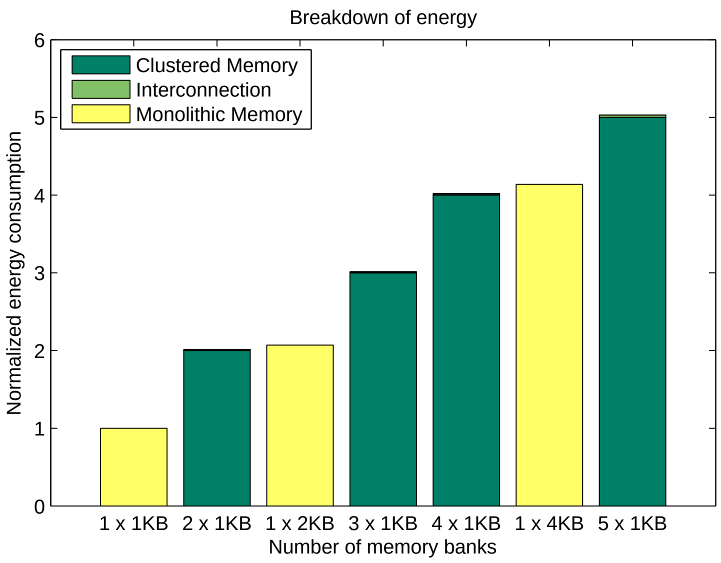
The current technology is studied as a first step towards the development of an interconnection cost model. CAD tools allow the design of the described clustered memory architectures. The synthesis and the simulation provide reliable data for the area and the power consumption of the different parts of the memory architecture. The goal is to synthesize a clustered memory architecture and extract power data for the memory banks and the interconnection logic separately. The work-flow is divided in several sub-steps:

- A number of memory banks is chosen from a library, which contains several state-of-the-art designs.
- An RTL description for connecting the memory banks into a full memory architecture design is written.
- A simulation is performed to verify the correct functionality of the memory architecture.
- A target technology is chosen and the logic synthesis of the memory architecture is performed.
- Floor-planning and the place & rout of the memory design is performed.
- Dynamic timing and the power simulation are performed and results are provided.

The memory models applied in the first step are presented in [28]. They are part of a library of standard cell-based memories (SCMEM) designed at IMEC. As shown in [82] such memories are power efficient when the size requirements are small. The SCMEM used in the current work are developed in IMEC and have similar characteristics with the ones described in [82]. The energy numbers for SCMEM are derived from synthesis and simulation results. The presented work-flow is a typical procedure followed by an industry designer except for the usage of SCMEM instead of commercially available memory macros. SCMEM are preferred for their characteristics, especially their energy and area efficiency for reasonably small storage capacities, as argued in [80]. Thus, the choice of using SCMEM is mostly related to the heavily distributed memory organization, which we target. The sizes of the memory banks in the target memory organization are too small to motivate a solution based on macro memory blocks, mainly because macro memories are dominated by the periphery. Another factor for this decision is that a rectangular cell array is not necessarily the optimal solution in terms of energy consumption. The usage of the custom memories provides the freedom to explore different memory organizations by combining banks of different sizes and structures.

For the second step, the RTL description connects the memories using MUX , signals, and other components into a functional clustered memory architecture. In the third step, verification is performed through simulation where a flash-write followed by a read on the whole memory architecture is performed. The read and write test-bench is representative for the types of data-intensive applications that benefit the most from clustered memory architecture. The target technology depends on the available libraries. In this work, a TSMC 45 nm library is used. Place and route can either be performed automatically through the CAD tool or manually by the designer. In our case, the layout is automatically generated to reduce the needed time and effort. However, the automatically generated layout is not ideal and a manual layout can be more efficient. The performance of the layout generation deteriorates for larger memory designs, so in the current work the clustered memory architectures are based on small memory banks. Although the layout could be improved further manually by the designer, all the designs are generated in the same way and the comparison between them, which is the main focus of this work, can give reliable results.

The final step includes extraction of parasitic, static timing analysis, and annotation of the timing to the netlist. Afterwards, power simulations on the synthesized design are carried out using Synopsys PrimeTime, in order to obtain energy numbers. The access pattern for the simulation consists of one full write of the memory architecture, followed by one full read and the comparison of the written and read data to verify that the memory operates correctly. Although both the dynamic and static power are reported, the focus of this work is on the dynamic part. The memory architecture is heavily accessed for the studied data-intensive applications, thus the dynamic power is dominant. The leakage is higher inside the memory banks than the interconnection, but in both cases significantly lower than the corresponding dynamic part.



Note: The small interconnection overhead is not easily visible in the figure without zooming in the top of the bars.

Figure E.2: Normalized energy breakdown between the memory banks and the interconnection.

E.3.2 Example design: synthesis and simulation

A group of clustered memory architectures is designed and synthesized following the presented work-flow. The simulation provides results for the current technology and the contribution of the interconnection to the overall energy consumption. The study includes clustered memories with an increasing number of memory banks, beginning with only one memory bank and having five memory banks as the maximum. There are two main reasons for exploring architectures up to five memory banks. The energy gains achieved by increasing the number of memory banks in the memory architecture are nearly saturated even for five banks. In [28] a group of different applications were studied with regard to their energy consumption on a clustered memory architecture consisting of up to five memory banks. The results shows that depending on the application, the energy gains start to saturate after adding a third or a fourth bank and become far smaller when adding a fifth bank. Thus, for most applications a memory architecture with five memory banks already provides more than necessary reconfigura-

Table E.1: Normalized energy breakdown between the memory banks and the interconnection

Memory Configuration	Energy on Memory Banks	Energy on Interconnection	Interconnection Overhead
1 x 1KB	1	-	0%
2 x 1KB	2	0.03	1.26%
1 x 2KB	2.07	-	0%
3 x 1KB	3	0.04	1.37%
4 x 1KB	4	0.07	1.77%
1 x 4KB	4.14	-	0%
5 x 1KB	5	0.15	3.01%
The energy is normalized to a memory bank of 1KB			

tion options. Secondly, the interconnect overhead increases exponentially with the number of memory banks, due to the increased complexity of the memory architecture. The significant increase in the overhead is indicated by the synthesis results, especially while comparing the overhead between four and five memory banks. Therefore, a memory architecture with six banks seems to be a less efficient option due to the high overhead and the very low energy gain. However, further investigation is proposed as future work to provide accurate numbers for architectures with six memory banks.

The breakdown of energy consumption in the memory architecture is split into two parts. The first part is the energy consumption internally in each memory bank, which includes the memory cells and the necessary logic to connect the cells. The second part is the interconnection cost between the different memory banks, which includes the necessary logic to locate and transfer the data outside of the banks. In other words, the interconnection outside the memory banks is given separately. The interconnection between the different memory cells inside one bank is included in the energy of the memory bank.

The example design is built using memory banks of 1KB and a bus width of 32bits. Each bank has one read and one write port and can operate independently. The memory banks are not directly connected to each other, but connect to a multiplexer that can read and write data to the appropriate bank based on the data address. The energy breakdown between the first part of the memory banks and the second part of the interconnection (part2) is presented in Fig. E.2. The energy cost of the interconnection

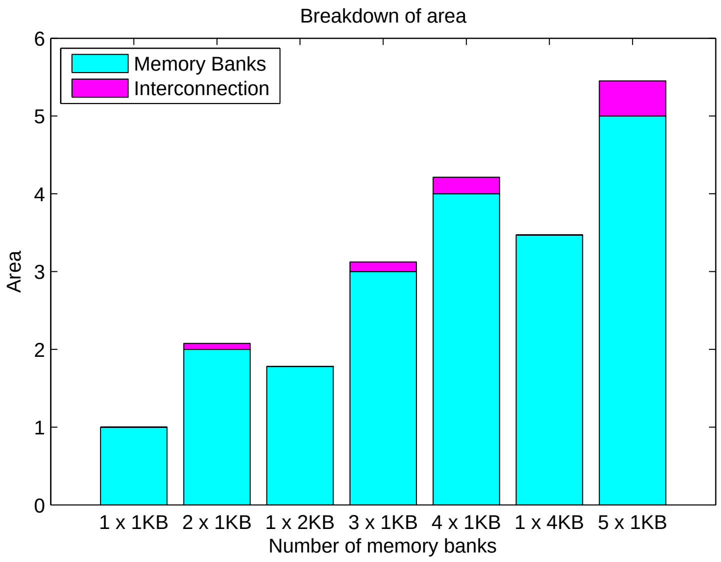


Figure E.3: Normalized area breakdown between the memory banks and the interconnection

logic is very small compared to the energy cost for the write and read operations on the memory banks. Tab. E.1 contains the exact percentages of the energy overhead of the interconnection. Energy consumption for a monolithic design with only one large 4KB memory is included in Fig. E.2. The comparison shows that the energy consumption for this design is higher than in a clustered solution with four 1KB memories, although there is no need for interconnection logic. This is due to the fact that the energy consumption for each access operation increases with the size of the memory [109].

The changes in memory area for the designs are presented in Fig. E.3. The area used for the placement of the memory banks is separated from the area occupied by the interconnection. The overhead calculation for additional banks takes into consideration the increase in the interconnect and the increase in address decoding and all other necessary design modifications. The last column corresponds to the area footprint for a monolithic 4KB memory. The area footprint for placing 4 banks of 1KB is larger than the area for one bank of 4KB. The higher area is needed for the interconnect. Furthermore, the layout is easier and more compact when there is only one bank of 4KB. A manual layout can potentially reduce the area overhead for the clustered memory, but this is beyond the scope of this work.

In addition, the synthesis and simulation of the memory designs provide useful information about the dimensions of the memory banks, the length and the capacitance of the wires. Several interesting observations are possible based on the study of the current technology. The energy overhead caused by the interconnection of the banks grows when there are more memory banks connected, as expected. However, the overhead is only slightly over 3% even for a clustered memory with five banks. The area overhead is significantly higher and grows exponentially for an increasing number of memory banks. The maximum area overhead is still below 10%. This is explained by the need for extra wiring to connect the different memory banks. Again, it should be noted that the results are for the dynamic power. Static power is omitted due its low contribution in the interconnection and the data intensive behavior of the target applications.

E.4 Technology Scaling

The technology scaling projections are based on the reports released by the International Technology Roadmap for Semiconductors (ITRS) [24]. The clustered memory architecture can be divided into two parts, the memory banks and the interconnection.

E.4.1 Memory Banks

The memory banks consist of the memory cells, which in our case are built using gates. Thus, the predictions about the future behavior of the memory banks are based on the ITRS reports for logic. The reports for logic are chosen, because the clustered memory architectures that are studied in this work are synthesized using standard cells. Different predictions and reports are provided by ITRS for other types of memories, such as DRAM. However, the proposed scratchpad memory architectures are the preferred choice for our scope. Typically the memories used in embedded systems are smaller and more energy efficient than DRAM.

In the coming years, the transistor gate length is expected to be reduced as shown in Fig. E.4 generated from data found in [25]. The values are approaching 5nm around 2028, which is potentially the limit using the current manufacturing process. The reduction in gate lengths leads to a reduced size

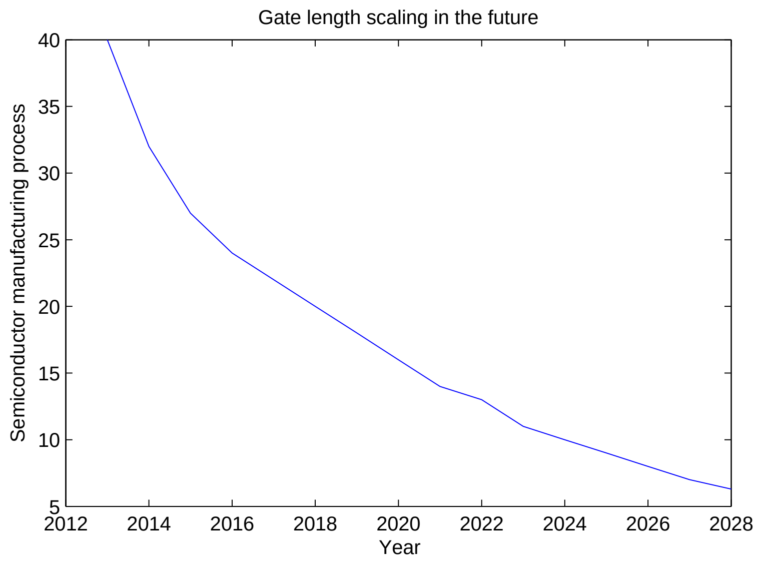


Figure E.4: Impact of technology scaling into gate length

of the memory cell and consequently smaller memory banks. The smaller memory banks affect both the area of the design and the power consumption. The projections provided by ITRS regarding the power are presented in Fig. E.5 generated from data found in [25]. There is a significant reduction in power consumption in the short term and slighter reduction towards the end of the projections.

E.4.2 Interconnection

The interconnection cost is based on the projections about wiring, which are different from the projections for logic gates. The current study is restricted to the reports about the lower and intermediate metal layers, because these are the ones that are relevant for the interconnection of our memory organizations. The reports about the global interconnection scaling is not our focus, as it is used for the power and clock routing and between computing clusters on large SoC platforms. However, the changes in size of the memory banks affects the length of the needed wires. The most important parameters for the current study are the capacitance and the power consumption of the interconnection. Curves for these two parameters are presented in

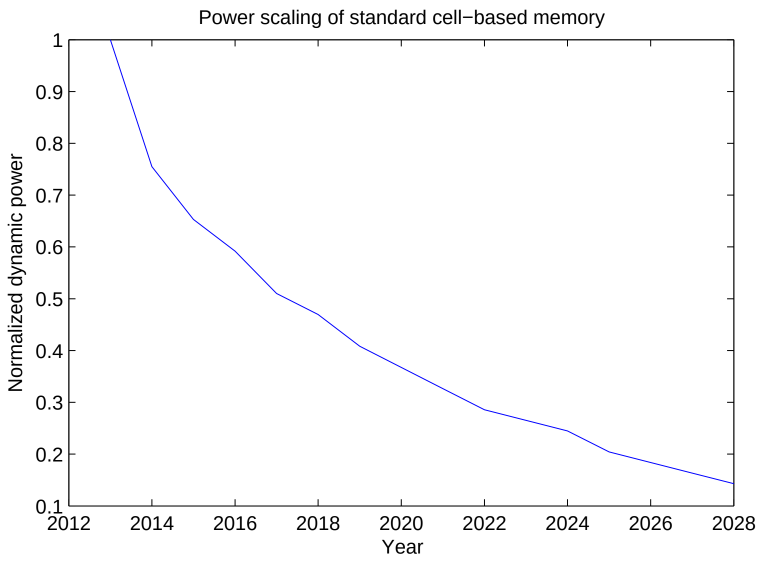


Figure E.5: Memory Banks: Normalized dynamic power and technology scaling

Fig. E.6 generated from data provided by ITRS [25]. The capacitance is expected to be reduced in the following year. Although the capacitance is reduced, the rate of this reduction is lower compared to the expected reduction for the memory cells. The power values are per length unit and they are expected to rise due to several challenges that the interconnection technology will face in the future.

The significant parameter in this work is the capacitance. Resistance is also generally studied in the interconnection because of its important impact, especially for signal delay. The current work focus on energy rather than delay and this is a rational choice for embedded systems. In general, the clock speeds are relatively low for a typical embedded system in contrast to high performance computing. In our target domain, a system architecture includes several processing cores that can be of different types to fulfill different application requirements. When higher performance is needed in an embedded system, it is usually achieved by adding another processing core or a special hardware unit rather than having an extremely high clock speed. Thus, it is expected that the critical path delay is not the main worry for a system designer. If the aim is high-performance designs, the current work has to be extended in the future. A different approach where delay-energy trade-offs are incorporated from the start should be developed in such a case. The interconnect resistance would then be important in

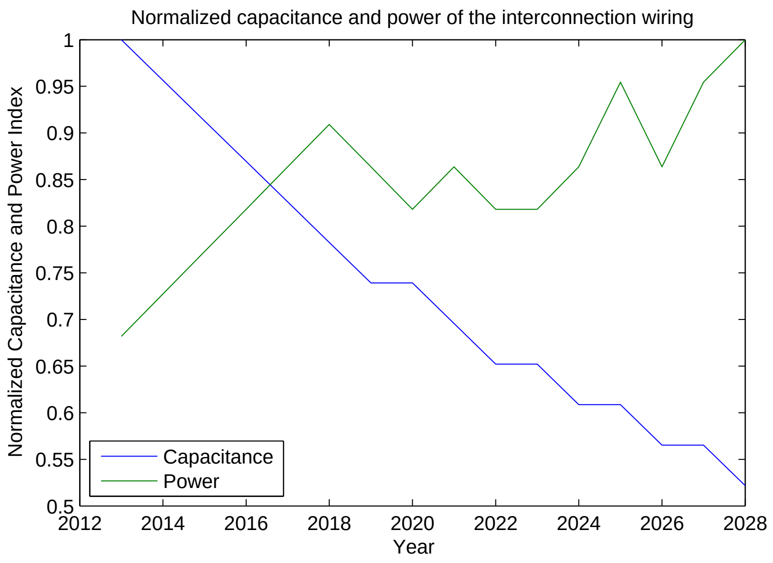


Figure E.6: Interconnection: Impact of technology scaling on the capacitance and the power consumption of the interconnection part

addition to the capacitance.

E.5 Model Construction and Projection Results

E.5.1 Model Construction

The scope is to develop a model that can provide a sufficiently accurate estimate of the power overhead for the interconnection in clustered memory architectures. The model is based on synthesis results of feasible designs in current technologies and study of the projections provided by ITRS. The model focuses on the dynamic power, which is the dominant factor for data intensive applications. Although leakage power is increasing for smaller technology nodes, the effect on the proposed clustered architecture is low. The main reason is that the part of the memory that is active at a given time is heavily accessed for the target applications, thus the static power is negligible compared to the static power. The part of the memory that is not accessed at a given time is normally switched off, thus the static power is heavily reduced. The study of the dynamic power is hence sufficient within our scope. The interconnect static power is generally low compared to the memory banks as discussed in [74].

The input for the model is the process technology and the configuration of the clustered memory architecture. The process technology leads to different points in Fig. E.5 and Fig. E.6 for the memory banks and the interconnection respectively. For the ITRS predicted year of a given technology on the x-axis chosen, the corresponding normalized values for dynamic memory power, interconnect capacitance and power can be found. The configuration of the architecture reveals the number of memory banks and all widths and lengths of the memory configuration. The power consumed by the memory banks is calculated using the total number of memory cells:

$$Banks_{power} \propto \sum_{Banks}^{forall} W \times L \times Cell_{power}$$

where W and L is the width and the length of each memory bank measured in memory cells. The power per cell is extracted using Fig. E.5. Each technology node year on the x-axis, leads to the corresponding cell power on the y-axis.

The prediction of the power consumption on the interconnection logic is based in Fig. E.6 in a similar way. Based on the technology and the number of memory banks, the length of the wires is calculated, which again gives the power and the capacitance. The general formula for the power consumption on a wire is:

$$Power = \frac{1}{2} \times f \times C \times V^2$$

where f is the activity factor, C the capacitance and V the supply voltage. However, the simulation results and the projections provided by ITRS suggest that a more detailed model is needed. The basic principle for the model is that the overall interconnection power is proportional to the power of the wires and their capacitance, as explained in the model presented in [122] and justified on the interconnection study in [85]. Thus, the general formula for the interconnection power is:

$$Interconnection_{power} \propto Area(W, L) \times Wire_{power} \times Capacitance$$

where $Area$ is calculated based on the total width and length of the memory banks according to the selected configuration. In more detail, the different memory banks are placed in a structure similarly to the configuration presented in Fig. E.1. The library of memory banks provides all the necessary information regarding their sizes. The simulation results for the synthesized configurations provide a sufficiently accurate estimate of the area needed for the interconnection, given the number and the sizes of the memory banks. The model for the interconnection power cost overhead for a given target technology t is given by the $Banks_{power}$ and the $Interconnection_{power}$:

$$Overhead = \frac{Area(W,L) \times Wire_{power}(t) \times Capacitance(t)}{\sum_{Banks}^{forall} W \times L \times Cell_{power}(t)}$$

where

- $Wire_{power}(t)$ is the wiring efficiency factor based on the power curve in Fig. E.6
- $Capacitance(t)$ is the capacitance of the interconnection wires for the given technology and the length of wiring based on the area, which is calculated for the different number of banks
- $Cell_{power}$ is the power consumption of a memory bank for the given technology

The model is an approximation of the interconnection overhead and the goal is to produce results for relative comparison.

The development of the interconnection cost overhead using the proposed model is presented in Fig. E.7. The interconnection overhead is kept below 10% for most of the cases. It exceeds this limit in designs with 5 memory banks around a decade from now. As expected, the overhead increases as we move from 2 towards 5 memory banks. However, the increase is much higher between a design of 4 and 5 memory banks compared to the smaller configurations. As motivated before, a memory architecture with six banks is not presented due to the expected high overhead that cannot be justified with the reconfiguration energy gains. The interconnection overhead predicted by the model is compared with the synthesis results for the current technology. In Tab. E.2 the first column is the overhead percentage presented in Tab. E.1 and the second column the predicted overhead percentage presented in Fig. E.7. The agreement between the two suggests that the model is sufficiently accurate for the current technology.

Table E.2: Comparison between predicted and simulated overhead

Number of Banks	Simulated Overhead	Predicted Overhead
2	1.26%	1.16%
3	1.37%	1.26%
4	1.77%	1.63%
5	3.0%	2.78%

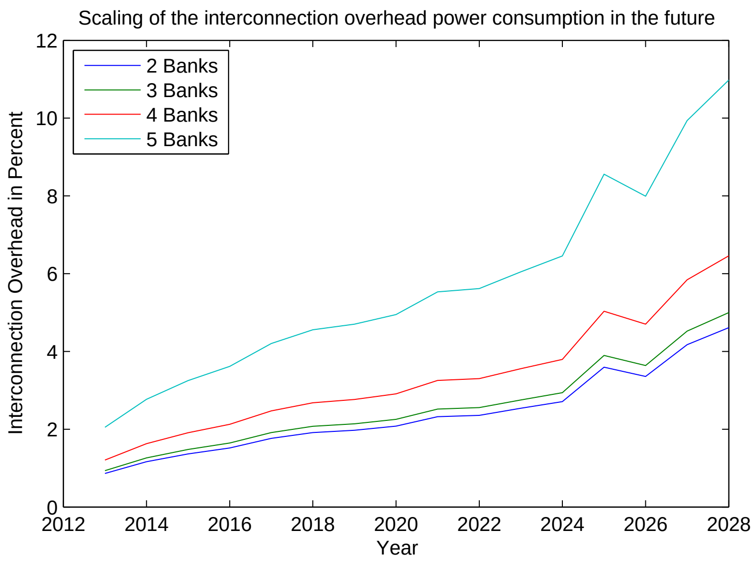


Figure E.7: Projections of the interconnection cost power overhead for different numbers of memory banks

E.5.2 Results

The model is useful during exploration of efficient clustered memory organization for a given application and technology. Based on Fig. E.7 it is likely that the optimum memory organization for a 45 nm design will include more memory banks compared to a 5 nm design. The overall results also indicates that since the overhead is small, the design approach using clustered memory architectures will be relevant in the future. The model provides

Table E.3: Energy gains vs. interconnection overhead

Banks	Sizes [kB]	Gains	Overhead(45nm)	Overhead(5nm)
2	8/32	40.1%	0.8%	4.6%
3	8/8/16	47.6%	0.9%	4.9%
4	8/8/8/8	51.7%	1.2%	6.4%
5	2/8/8/8/8	54.4%	2.0%	10.9%

necessary information to the system designer and steers the decision about the number of banks in the memory architecture for a given technology. The improvement in the designer's decision is a strong motivation for the development of the model.

To better illustrate the change in the optimum memory architecture for future technologies, an example application is chosen. The EPIC (Efficient Pyramid Image Coder) is an image compression algorithm and its behavior on a clustered image architecture is presented in [28]. The energy gains for the most energy efficient memory organizations are presented in Tab. E.3. In these numbers the cost of the added interconnect is not included. The energy gain percentage is constant and independent of the technology, because the comparison is always between a monolithic and a clustered memory architecture of the same technology. The sizes of the memory banks for each configuration are used to calculate the interconnection overhead for synthesizing in 45nm and 5nm. The percentage of the energy overhead due to the interconnection cost is also presented in Tab. E.3 both for the current and the most advanced technology. Based on the improvements and the overheads the most energy efficient design for a specific technology can be defined for the EPIC benchmark application. For the 45nm technology, a clustered memory architecture with five memory banks is the optimum, while for the 5nm technology four memory banks give the optimal solution. The addition of a fifth bank reduces the energy consumption by 2.7%, as a result of the lower energy consumption inside the memory banks, but at the same time increases the energy consumption by 4.5%, as a result of the higher energy consumption on the interconnection between the banks.

E.6 Conclusion

We have proposed a model that can provide estimates of the interconnection overhead in a clustered memory architecture in future technology nodes. The model suggests that overhead will be kept low in the short term and will increase within reasonable levels in the mid-long term. Therefore, the design of energy efficient clustered memory architecture will continue to be a good design choice. The estimations for the future can be useful for system designers that try to design power efficient architectures for applications with dynamic memory requirements throughout their lifetime. Another contribution of the model is that it provides an early indication about the optimal number of banks for a given technology and design. This information reduces the design space exploration and the design time. When newer technologies become available in the CAD tools, the designs can be re-synthesized and the model can be calibrated.