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

IASON FILIPPOPOULOS, Norwegian University of Science and Technology
NAMITA SHARMA, Indian Institute of Technology Delhi
FRANCKY CATTHOOR, IMEC & K.U. Leuven
PER GUNNAR KJELDSBERG, Norwegian University of Science and Technology
PREETI RANJAN PANDA, Indian Institute of Technology Delhi

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.

Categories and Subject Descriptors: D.2.2 [**Design Tools and Techniques**]: Design, Methodologies; B.7.1 [**Types and Design Styles**]: Memory Technologies, Design; B.8.2 [**Performance Analysis and Design Aids**]; C.3 [**Special purpose and Application-Based Systems**]: Real-time and Embedded Systems; E.1 [**Data Structures**]: Arrays

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Data Interleaving, Single Instruction Multiple Data (SIMD) Architectures, Design Space Exploration, Memory Reconfiguration, Data Layout, Energy Optimization

ACM Reference Format:
Iason Filippopoulos, Namita Sharma, Francky Catthoor, Per Gunnar Kjeldsberg, Preeti Ranjan Panda, 2015. Integrated Optimization Methodology for Data Interleaving and Memory Mapping Exploration on SIMD architectures. *ACM Trans. Embedd. Comput. Syst.* V, N, Article A (January YYYY), 20 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

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 archi-

Author's addresses: Iason Filippopoulos and Per Gunnar Kjeldsberg, Department of Electronics and Telecommunications, Norwegian University of Science and Technology (NTNU), Hogskoleringen 1, 7491 Trondheim, Norway; Namita Sharma and Preeti Ranjan Panda, Department of Computer Science and Engineering, Indian Institute of Technology Delhi, Hauz Khas, New Delhi - 110016, India; Francky Catthoor, Interuniversity Microelectronics Centre and K. U. Leuven, Kapeldreef 75, B-3001 Leuven, Belgium.

DOI:http://dx.doi.org/10.1145/0000000.0000000

tectures 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 irregular 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 [Gonzalez and Horowitz 1996] 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 [Cheung et al. 2009] and the cycle-accurate simulator presented in [Šimunić et al. 1999] 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 [Hulzink et al. 2011] shows that the memory subsystem consumes more than 40% of the leakage power on the platform. According to [Ma et al. 2007], 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 2 motivates the study of developing a methodology for optimization of the data interleaving exploration and the data-to-memory mapping. Section 3 surveys related work on both the interleaving and memory mapping exploration. Section 5 presents the general work-flow and the sequence of the methodology steps. In Section 4 the target platform is described accompanied by a detailed description of the employed SIMD architecture and the memory models, while the set of benchmarks and their characteristics are analyzed in Section 6. Results of

ALGORITHM 1: Motivational Example Algorithm

```
$N \leftarrow 100$
for ($i = 0; i < N; i + 4$) do
    $result(i) = A(i) + B(i) + C(i) + D(i);$
end
```

applying the described exploration methodology to the targeted applications are shown in Section 7, while conclusions are drawn in Section 8.

2. MOTIVATIONAL EXAMPLE

Many different techniques have been proposed already to deal with the memory accesses and the potential memory bottlenecks. Authors in [Malik et al. 2000] propose a low power cache memory for embedded systems that can optimize memory accesses based on application's requirements. The goal in [Guo et al. 2013] 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 [Mittal 2014]. Reconfigurable hardware for embedded systems, including the memory architecture, is a topic of active research. An extensive overview of current approaches is found in [Garcia et al. 2006]. The current work is complementary to the existing ones and focuses on the areas that has not been explored in detail as discussed in 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. 1 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. 1. 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.

The initial data representation for the arrays *A, B, C and D* is shown in Fig.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.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. 1(b). The data-to-memory mapping for the constructed interleaved array is presented in Fig. 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 $x \bmod 4$. 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. 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. 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 or one larger bank. The overall number of memory accesses is:

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

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:

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

A quick estimation for the difference in the energy consumption between the approaches presented in Fig. 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:

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

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

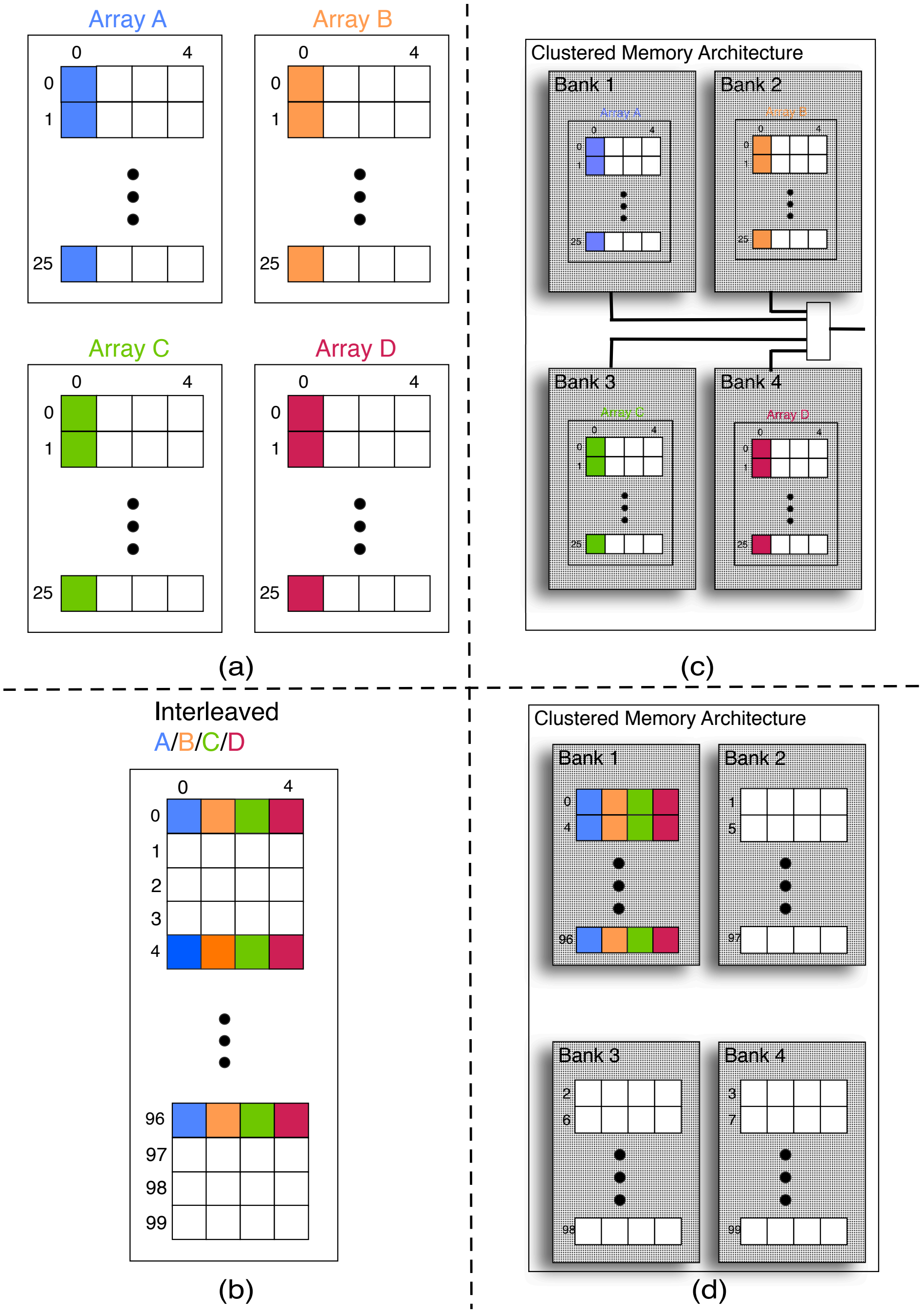


Fig. 1. Motivational Example. Representation of the initial set of data (a) and the interleaved set of data (b). Data-to-memory mapping for the initial (c) and the interleaved data (d) on a clustered scratch-pad memory architecture.

for a four bank memory architecture with non-interleaved data is:

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

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

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

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

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

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.

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 [Manjikian and Abdelrahman 1995], cache partitioning, a layout technique for arrays, maps each array to different cache partitions to reduce conflicts. In [Kulkarni et al. 2005] 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 [Brockmeyer et al. 2007], 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 [Panda et al. 1999] partitions the variables to scratch-pad memory and DRAM to minimize interference between variables. The cache-oriented techniques proposed earlier by researchers in [Panda et al. 2001] 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 [Macii et al. 2002]. The current work differentiates by employing a platform that is reconfigurable at run-time. In [Panda et al. 2001] 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 [Benini et al. 2000a] present a methodology to generate a static application-specific memory hierarchy. Later, they extend their work in [Benini et al. 2000b] to a reconfigurable platform with multiple memory banks.

The authors in [Abraham and Mahlke 1999], [Jacob et al. 1996] and [Li and Wolf 1999] present methodologies for designing memory hierarchies. Design methods with main focus on the traffic and latencies in the memory architecture are presented in [Chen and Sha 1999], [Grun et al. 2000], [Jantsch et al. 1994] and [Passes et al. 1995]. Improving memory energy efficiency based on a study of access patterns is discussed in [Kandemir et al. 2001]. Application specific memory design is a research topic in [Schmit and Thomas 1997], while memory design for multimedia applications is presented in [Oshima et al. 1997]. The current work differentiates by presenting both a reconfigurable platform and the necessary data interleaving to better exploit the platform.

The current work combines and expands in a non-trivial way, the interleaving exploration presented in [Sharma et al. 2013] and the data to memory mapping methodology presented in [Filippopoulos et al. 2013]. In [Sharma et al. 2013] 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 [Filippopoulos et al. 2013] 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.

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.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 developed, which is beyond the scope of the current work. The SIMD vector FU is assumed to perform instructions on multiple data.

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.

4.1. Memory Models

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 memo-

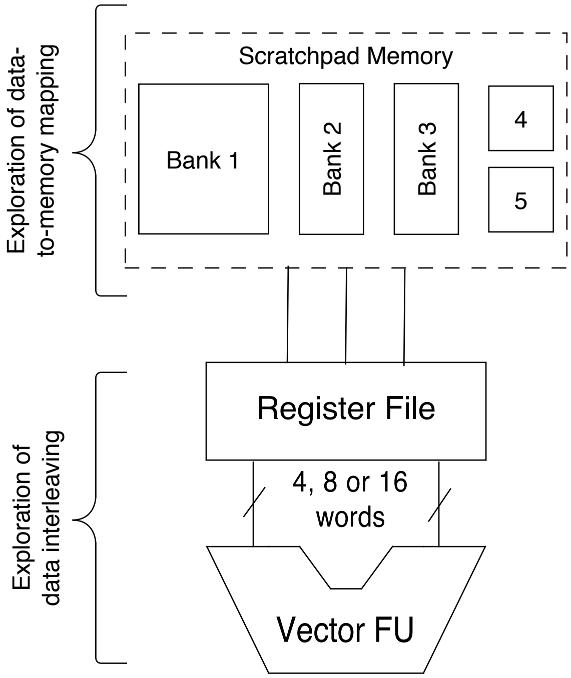


Fig. 2. Exploration options and system knobs depending on a general platform architecture

ries (SCMEM) [Meinerzhagen et al. 2011] are considered for smaller memories due to their energy and area efficiency for reasonably small storage capacities, as argued in [Meinerzhagen et al. 2010]. 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 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 I. 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

Source: [Filippopoulos et al. 2013]

— 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 I. It is clearly shown that the choice of memory units has an important impact on the energy consumption.

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.2. For efficient utilization of the vector FU, the register file has a wide interface (256 bit wide) with the scratchpad memory and allows testing for word length of 4, 8 and 16 elements, assuming that each array element is 16 bit wide. This processor belongs to the class of coarse-grain reconfigurable array (CGRA) processors and is described in more detail in [Lee et al. 2003]. The HDL model of the processor is synthesized using Cadence RTL compiler [Cadence 2014] and the energy numbers are extracted using Synopsys Primepower [Kai and Zhiwei 2003].

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

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 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.4. The exploration space consists

Table II. 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

Note: Models for multiplication are different based on the type of input variables.

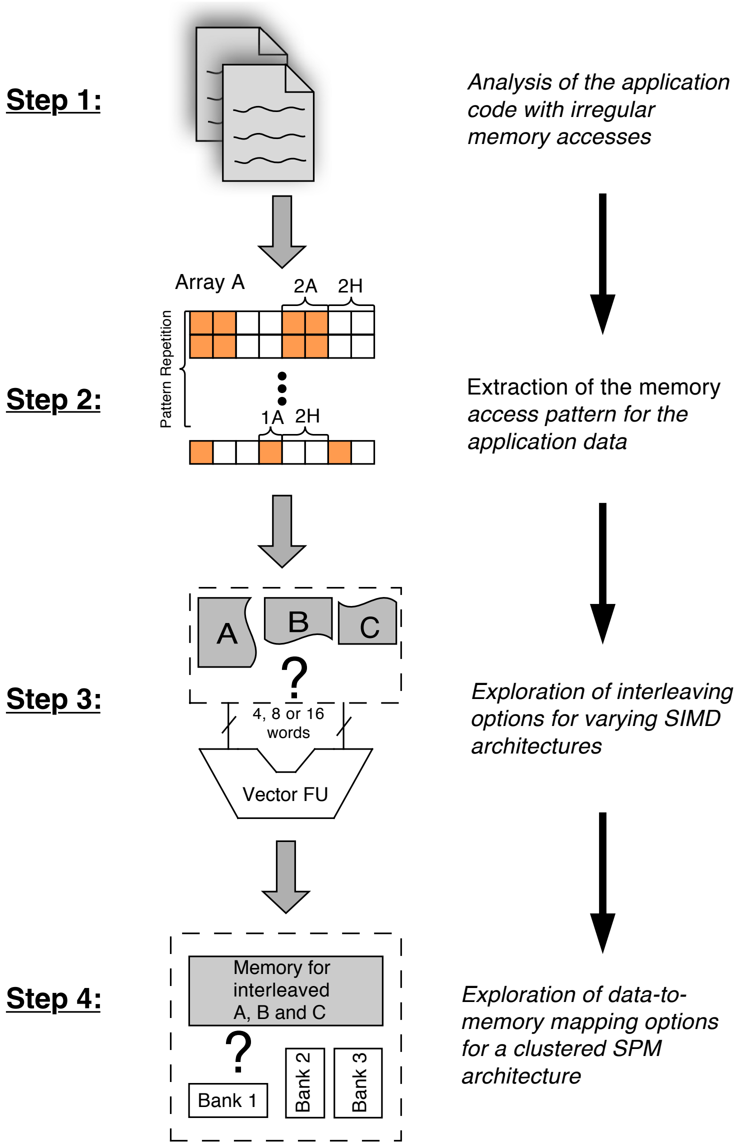
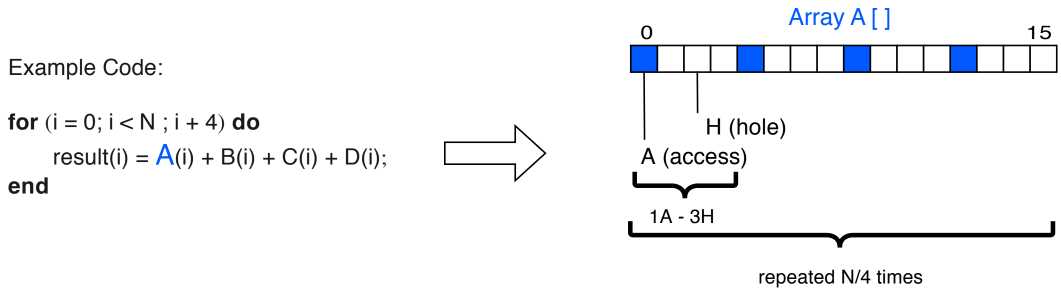

Step 1:
Analysis of the application code with irregular memory accesses
Array A
2A
2H
Pattern Repetition
1A
2H
Step 2:
Extraction of the memory access pattern for the application data
A
B
C
?
4, 8 or 16 words
Vector FU
Step 3:
Exploration of interleaving options for varying SIMD architectures
Memory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

Fig. 3. Methodology steps



Example Code:

```
for (i = 0; i < N ; i + 4) do
    result(i) = A(i) + B(i) + C(i) + D(i);
end
```

Fig. 4. Extraction of access pattern from application code

of the potential combinations between the different arrays and the different scratchpad memory architectures, which combine memories of different types and sizes.

The overall work-flow is presented in Fig.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 irregular 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 re-arrangement 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.

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 [Kritikakou et al. 2014] 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.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.

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 number of memory accesses for accessing them. The basic principles of the performed interleaving exploration are presented in [Sharma et al. 2013].

The employed model of access pattern representation is accompanied with a suitable algebra presented in [Kritikakou 2013]. 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.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 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.

This idea is illustrated in Fig.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 second case is reduced by half compared to the first case.

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. 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

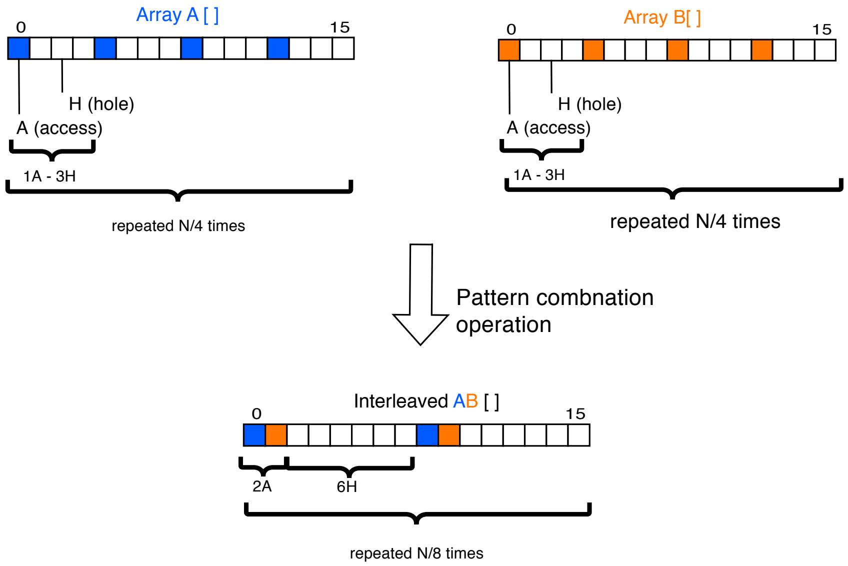


Fig. 5. Example of combination between two arrays and their access patterns

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.

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 [Catthoor et al. 1998]. 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.

6. APPLICATIONS

The applications that benefit most from the proposed methodology are characterized by having irregular access patterns with holes. Applications with irregular accesses 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. 2. It uses four arrays that all have the same irregular 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 a more irregular access pattern. 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 an representative example from the equation solver domain.
(3) FFT benchmark has an irregular pattern during the access of the pilot matrices. The interleaving exploration for FFT is presented in [Sharma et al. 2013]. 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.

7. 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.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.

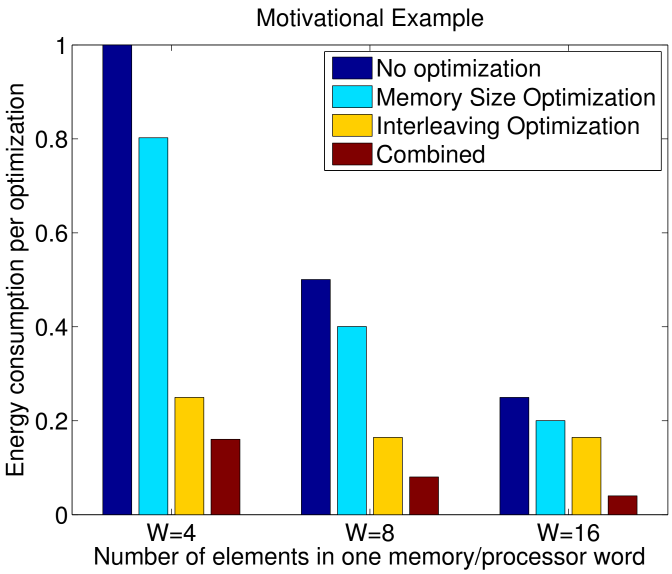


Fig. 6. Motivational Example

7.1. Motivational Example

The normalized energy consumption for the motivational example is presented in Fig.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.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 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.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

ALGORITHM 2: Code snippet from the SOR benchmark

```
...
for $(j = 0; j < N; j + 2)$ do
    ...
    resid = ...
    $+c(i)(j) \times u(i)(j-1)$
    $+d(i)(j) \times u(i)(j)$
    $+e(i)(j) \times u(i)(j+1)$
    + ...
end
...
```

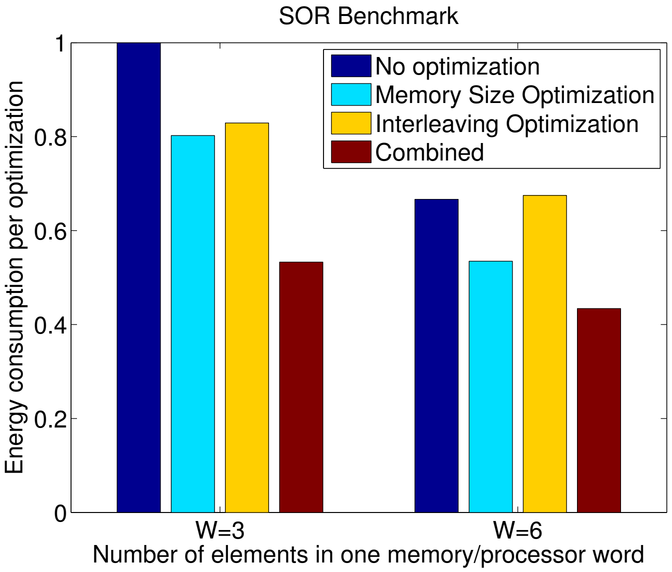


Fig. 7. SOR Benchmark

perfectly compact sets of data as shown in Fig.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 case for width of 16. The combined approach results in an energy gain of 84% on the same case.

7.2. SOR Benchmark

The normalized energy consumption for the SOR benchmark is presented in Fig.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. 2 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

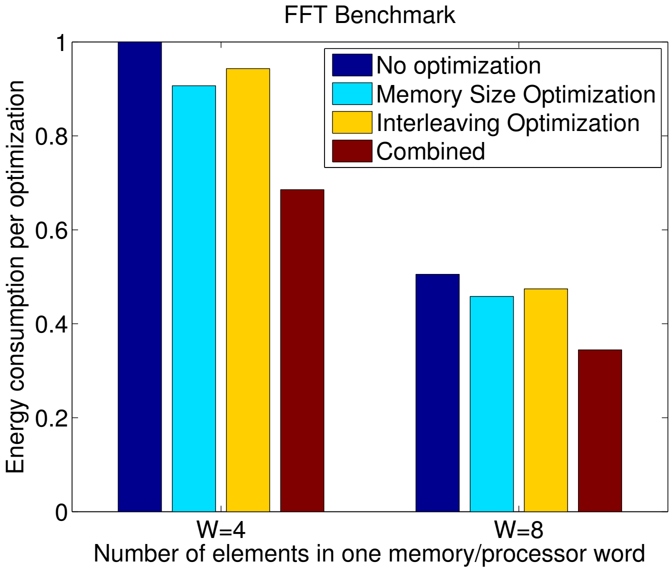


Fig. 8. FFT Benchmark

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.

7.3. FFT Benchmark

The normalized energy consumption for the FFT benchmark is presented in Fig.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.

7.4. Motion Estimation Benchmark

The normalized energy consumption for the motion estimation benchmark is presented in Fig.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

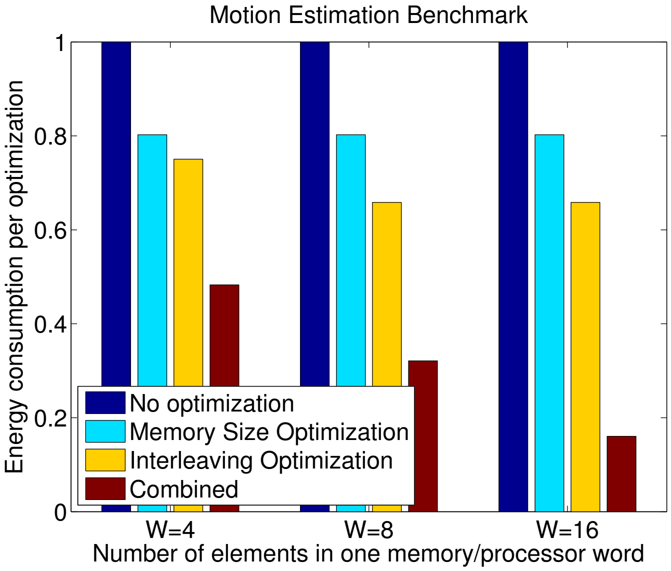


Fig. 9. Motion Estimation Benchmark

interleaving and consequently the interleaving exploration has a greater impact than the memory size optimization. However, the integrated approach optimizes the energy consumption 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.

8. 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 irregularities in their access scheme that benefit most from the methodology.

REFERENCES

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

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

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

Erik Brockmeyer, Bart Durinck, Henk Corporaal, and Francky Catthoor. 2007. Layer assignment techniques for low energy in multi-layered memory organizations. In *Designing Embedded Processors*. Springer, 157–190.

RTL Cadence. 2014. Compiler Users Manual. (2014).

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

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

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

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

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

R. Gonzalez and M. Horowitz. 1996. Energy dissipation in general purpose microprocessors. *Solid-State Circuits, IEEE Journal of* 31, 9 (sep 1996), 1277 –1284. DOI:http://dx.doi.org/10.1109/4.535411

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

Yibo Guo, Qingfeng Zhuge, Jingtong Hu, Juan Yi, Meikang Qiu, and Edwin HM Sha. 2013. 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 (2013), 809–817.

J. Hulzink, M. Konijnenburg, M. Ashouei, A. Breeschoten, T. Berset, J. Huisken, J. Stuyt, H. de Groot, F. Barat, J. David, and others. 2011. An Ultra Low Energy Biomedical Signal Processing System Operating at Near-Threshold. *Biomedical Circuits and Systems, IEEE Transactions on* 5, 6 (2011), 546–554.

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

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

Wang Kai and Xu Zhiwei. 2003. Synopsys Prime Power Manual Release U-2003.06-QA. (2003).

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

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

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

Chidamber Kulkarni, C Ghez, Miguel Miranda, Francky Catthoor, and Hugo De Man. 2005. Cache conscious data layout organization for conflict miss reduction in embedded multimedia applications. *Computers, IEEE Transactions on* 54, 1 (2005), 76–81.

Jongeun Lee, Kiyoung Choi, and Nikil D Dutt. 2003. Compilation approach for coarse-grained reconfigurable architectures. (2003).

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

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

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

Afzal Malik, Bill Moyer, and Dan Cermak. 2000. 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*. IEEE, 241–243.

Naraig Manjikian and Tarek Abdelrahman. 1995. Array data layout for the reduction of cache conflicts. In *Proceedings of the 8th International Conference on Parallel and Distributed Computing Systems*. Citeseer, 1–8.

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

Pascal Meinerzhagen, SM Yasser Sherazi, Andreas Burg, and Joachim Neves Rodrigues. 2011. 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).

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

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

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

Preeti Ranjan Panda, Nikil D Dutt, and Alexandru Nicolau. 1999. *Memory issues in embedded systems-on-chip: optimizations and exploration*. Springer Science & Business Media.

Preeti Ranjan Panda, Nikil D Dutt, Alexandru Nicolau, Francky Catthoor, Arnout Vandecappelle, Erik Brockmeyer, Chidamber Kulkarni, and Eddy De Greef. 2001. Data memory organization and optimizations in application-specific systems. *IEEE Design & Test of Computers* 3 (2001), 56–57.

NL Passes, Edwin Hsing-Mean Sha, and Liang-Fang Chao. 1995. 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*. IEEE, 440–445.

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

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

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