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 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 function unit, which performs the same operation on multiple input data in parallel. The memory accesses have an important contribution on 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 function units and decreasing the number of memory accesses. The presented methodology is tested using a number of benchmark applications with irregularities on their access scheme. Potential gains are calculated based on the energy models both for the processing and the memory part of the system. The improvement on the energy consumption after efficient interleaving and mapping of data is approximately 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

1. INTRODUCTION

The energy is an important limitation factor on 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 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 accessed always in order. The lack of spatial locality and the presence of redundant data results in lower utilization of the hardware resources, providing 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 on 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 redundant memory accesses. In order to reduce the number of memory accesses and achieve higher utilization of the system architecture a systematic exploration of the interleaving options for application's data is needed.

The energy consumption can be divided into two parts, namely the processing and the memory subsystem. The energy needed for processing depends mainly on the utilization of the FUs and any potential stalls, if the memory cannot provide data on the needed rate. The interleaving exploration can increase the utilization of the processing subsystem and reduce time penalties for data loading. The energy consumption on 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 a 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 4 presents the general work-flow and the sequence of the methodology steps. In Section 5 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 analysed in Section 6. Results of applying the described exploration methodology to the targeted applications are shown in Section 7, while conclusions are drawn in Section 8.

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 [Simunic 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.

2. MOTIVATIONAL EXAMPLE

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 of every fourth line within the combined array $(A|B|C|D)$ result in a sequence of 8 accessed 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 constrain regarding the number of lines or their width on the memory design. The

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**
```

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 coloured elements followed by blocks with 12 redundant elements.

The data-to-memory mapping for the initial set of data is presented in Fig. 1(b). The memory architecture consists of four memory banks and the overall memory size is enough to fit the four arrays. The conventional approach maps each array in a separate bank. The clustered memory architecture is a more energy efficient approach compared to a monolithic approach with one larger memory bank. 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 the modulo of the array line divided by 4. For example, line x is stored in bank $x \bmod 4$. As a result only one forth of the array A can be found on the first bank in contrast to the non-interleaving case, in which the whole array A is mapped on 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. We can assume a system architecture with an SIMD ADD FU that performs operations over 4 words at a time and a memory to processor path that has a width of 4 words. 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 words, 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 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

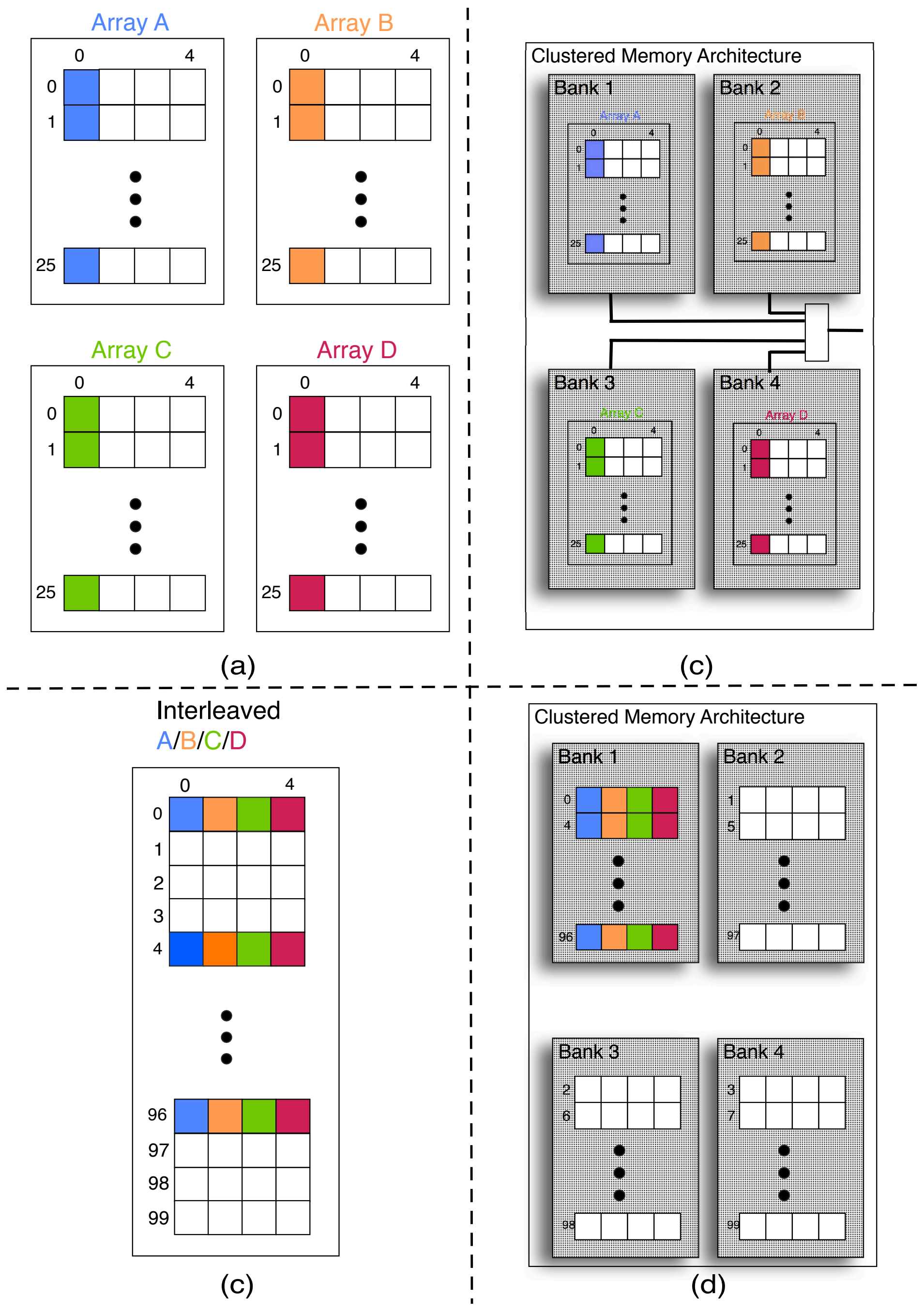


Fig. 1. Motivational Example. Representation of the initial set of data (a) and the interleaved set of data (b). Data-to-memory mapping for the initial (c) and the interleaved data (d) on a clustered scratch-pad memory architecture.

for a clustered memory architecture with non-interleaved data is:

$$100 \times 1E + 4\times 100\times 0.3E = 220E. \quad (4)$$

The energy consumption for a clustered memory architecture with interleaved data is:

$$25 \times 1E + 1\times 25\times 0.3E = 32.5E. \quad (5)$$

is 75% lower when the data are interleaved. 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 is assumed that are turned into retention mode to minimize leakage. The simple estimations presented in equations (1)-(5) shows that an interleaving optimization results in a 75% reduction in the number of accesses. A memory optimization based on an energy efficient clustered memory architecture can achieve an important energy reduction. The energy gains are even greater when interleaved data are mapped into 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 on 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 an independent application of the two 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 more 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 in a way that interference among different variables is minimized. The cache-oriented techniques proposed earlier by researchers in [Panda et al. 2001a] 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. 2001b] 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. 2000b] present a methodology to generate a static application-specific memory hierarchy. Later, they extend their work in [Benini et al. 2000a] to a reconfigurable platform with multiple memory banks.

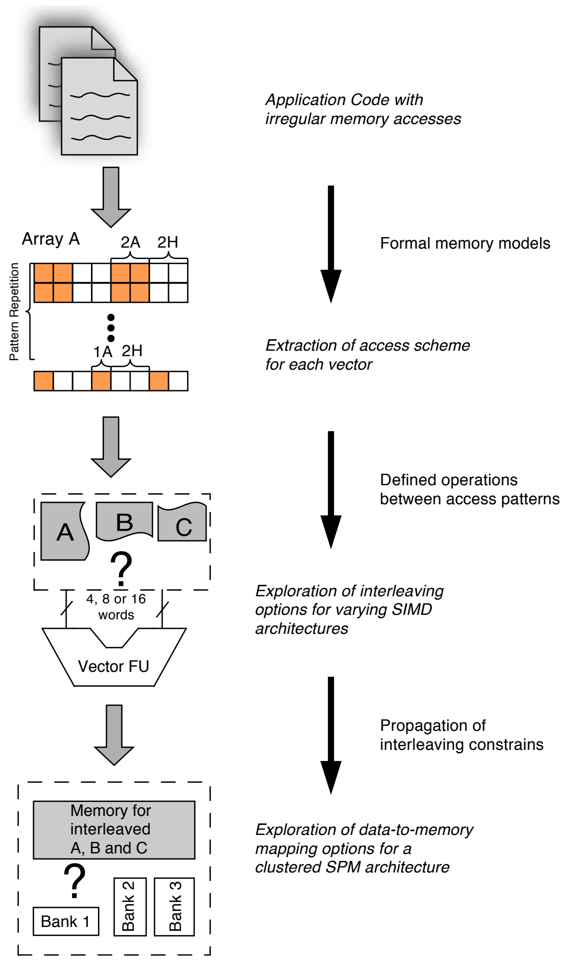
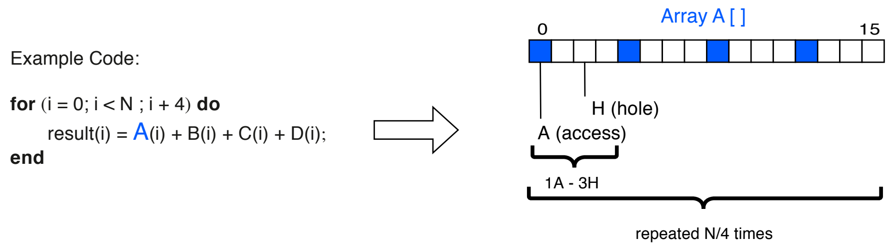


Fig. 2. Methodology steps

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 the interleaving exploration presented in [Sharma et al. 2013] and the data to memory mapping methodology presented in [Filippopoulos et al. 2013].


Example Code:

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

Fig. 3. Extraction of access pattern from application code

4. SYSTEM DESIGN EXPLORATION WORK-FLOW

The systematic exploration of interleaving options and data to memory mapping possibilities is necessary. 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 of this work is presented in Fig.2. 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 interleaving exploration, which explores all different option for re-arrangement of the data. The result of the interleaving is a set of data with a reduced number of holes. The next step is the mapping of the interleaved data set to the clustered memory architecture.

4.1. Formal Model Representation of Access Pattern

A representation model for the access pattern is employed in order to formally present each step of the methodology. The model presented in [Kritikakou et al. 2013] 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, is represented with an A(Access). Otherwise, there is a hole in the access pattern represented with an H(Hole) as shown in Fig.3. 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 on the work-flow.

4.2. 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 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 a more efficient way that increases the number of consequential accesses. A simple example is presented in Fig.4. 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 consequentially. The use of access patterns and the theory for calculating the access

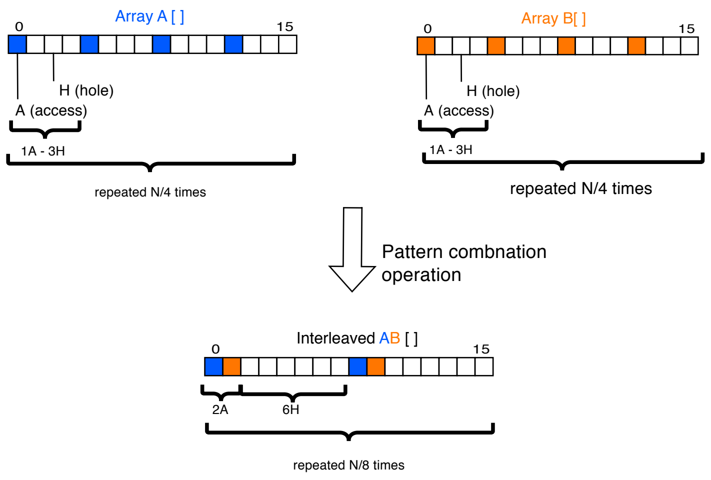


Fig. 4. Example of combination between two arrays and their access patterns

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.4. 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. On the case of the interleaved array, each memory access returns four elements, from which two are useful and two redundant. Thus, the number of overall memory access in the second case is reduced in half compared to the first case.

4.3. 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 make the optimal decision regarding the mapping of the data to the dif-

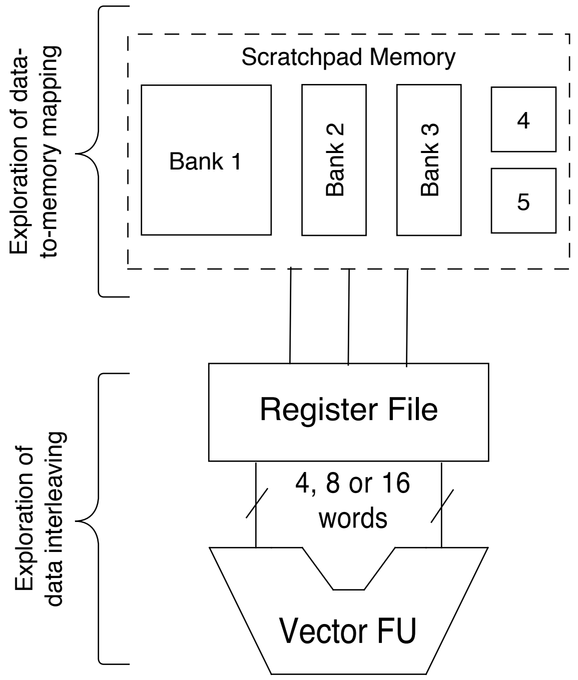


Fig. 5. Exploration options and system knobs depending on a general platform architecture

ferent memory banks. The parts of the interleaved data that consist mostly of useful elements are mapped into memory banks with low energy per access. 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 size as possible to avoid area and energy penalties.

4.4. One way constraint propagation

The decisions taken on the interleaving step affect the mapping options. If the interleaving decisions lead to small compact sets of data, the mapping can be done on small energy efficient memory banks. On the other hand, if the mapping exploration is performed first, the freedom for interleaving is reduced. We split the decisions in two steps and the interleaving decisions are propagated as constraints on the mapping exploration phase.

5. TARGET ARCHITECTURE

A generic architecture is presented in Fig.5. The methodology explores different interleaving and data to memory mapping options for this reconfigurable architecture. The scratchpad memory consists of up to five memory banks. The optimal sizes are found based on the sizes of the data after the interleaving exploration. The vector FU is assumed to perform instructions on multiple data. The explored data lengths are 4, 8 and 16 words. Each word is an array elements in our case.

5.1. Memory Models

The dynamic memory organisation 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) [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. Afterwords, 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.

Should I add tables as well?

5.2. Function Unit Models

We have used an extension of the DRESC compiler framework [10] for mapping the reference application to a CGRA architecture. The CGRA part comprises one SIMD enabled functional units (FU) and a central vector register file. 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 a 16 bit wide.

6. APPLICATIONS

The applications that benefit most from the proposed methodology are characterised by having irregular access patterns with holes. The multimedia domain offers some suitable candidates for the presented methodology. 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 has the same pattern. Further interleaving within the array can result into even longer sequences of use-

ful 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) 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.

(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.

(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.

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.5. Four approaches are explored and the corresponding energy consumption of each of them is calculated.

— *No optimization.* In this case there is no interleaving exploration and the memory architecture consists of a 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 four 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 there is 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 are performed. Both he number of memory accesses and the energy per access are reduced.

The normalized energy consumption for the motivational example is presented in Fig.6. The four different approaches are normalized using the conventional case without any optimization. Energy results for this benchmark are presented for a word length of 4, 8 and 16 elements, which means that every memory access loads/stores 4,8 or 16 elements. The application code is perfectly suitable for interleaving as discussed in Sec.2. Thus the interleaving exploration has a greater impact that the memory size optimization. However, the integrated approach optimizes the energy consumption even further. The application is suitable for higher word lengths and there are important gains while moving from 4 to 8 and 16. This is explained by the nature of the application that offers good interleaving options for larger words.

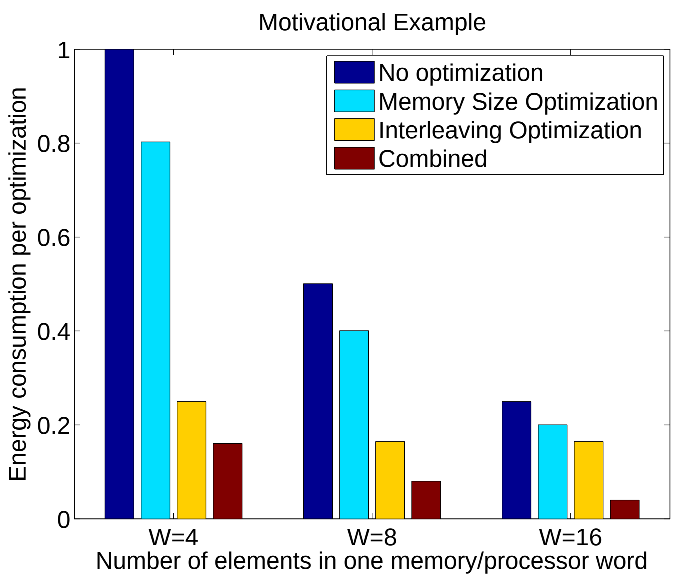


Fig. 6. Motivational Example

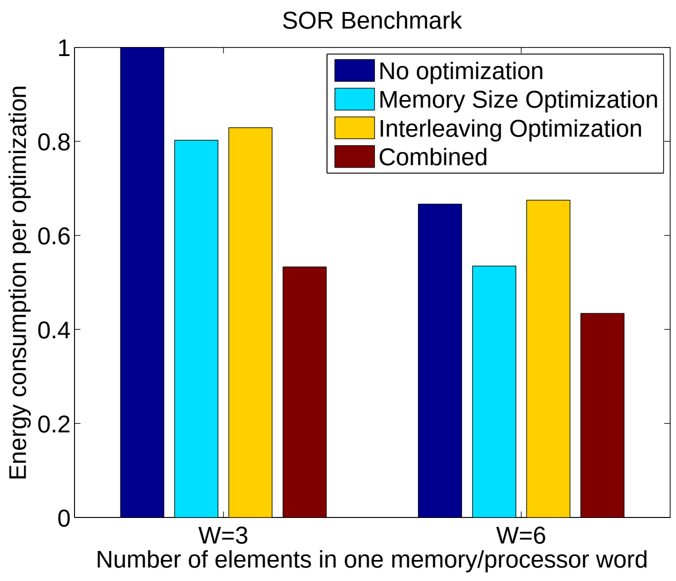


Fig. 7. SOR Benchmark

The normalized energy consumption for the SOR benchmark is presented in Fig.7. The four different approaches are normalized using the conventional case without any optimization. Energy results for this benchmark are presented for a word length of 3 and 6, which means that every memory access loads/stores 4 or 8 elements, because the architecture supports word lengths in the power of 2. 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 of the application. Thus the interleaving

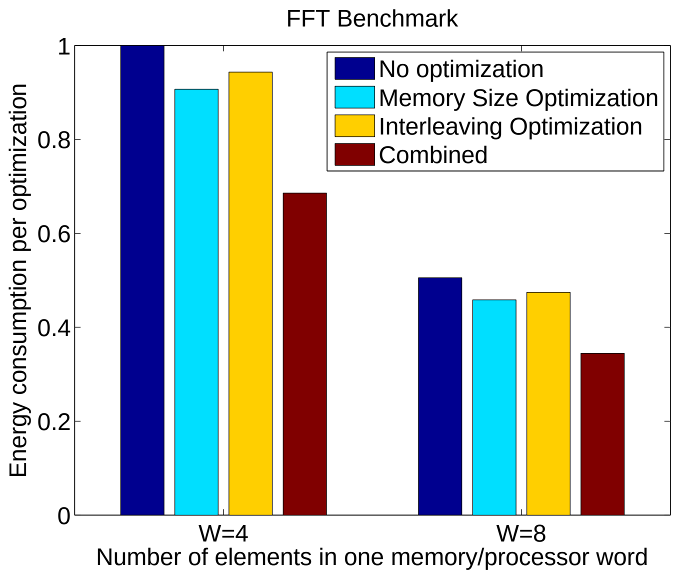


Fig. 8. FFT Benchmark

exploration alone has a small impact on the reduction of energy consumption for word length of 3 elements. The results are even worst for a word length of 6 elements, because the interleaved data have greater size and the conventional mapping of them to the memory is poor. The memory size optimization provides more important reduction. The integrated approach exploit on a better way the few interleaving options by providing a better mapping of the interleaved data to the memory architecture.

The normalized energy consumption for the FFT benchmark is presented in Fig.8. The four different approaches are normalized using the conventional case without any optimization. Energy results for this benchmark are presented for a word length 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 word length of 8 are significantly high, because blocks of 8 words can be constructed by interleaving of application's arrays.

The normalized energy consumption for the motion estimation benchmark is presented in Fig.9. The four different approaches are normalized using the conventional case without any optimization. Energy results for this benchmark are presented for a word length of 4, 8 and 16 elements. The application code provides good possibilities for interleaving and consequentially the interleaving exploration has a greater impact that the memory size optimization. However, the integrated approach optimizes the energy consumption even further. In this case the energy gains for increasing size of word length are minimal. This is explained by the nature of the application...

8. CONCLUSION

Justify why the integrated approach gives better results than just combining the two methods...

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.

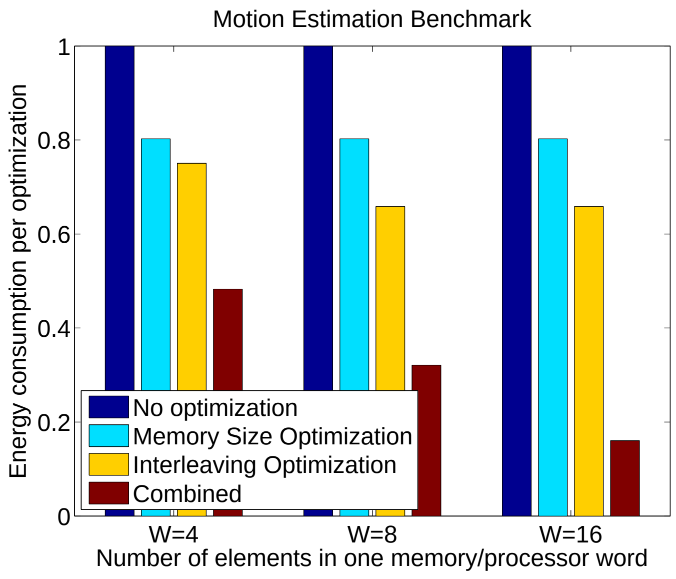


Fig. 9. Motion Estimation Benchmark

L. Benini and others. 2000a. Increasing energy efficiency of embedded systems by application-specific memory hierarchy generation. *Design Test of Computers, IEEE* 17, 2 (apr-jun 2000), 74 –85. DOI:http://dx.doi.org/10.1109/54.844336

L. Benini, A. Macii, and M. Poncino. 2000b. A recursive algorithm for low-power memory partitioning. In *Low Power Electronics and Design, 2000. ISLPED '00. Proceedings of the 2000 International Symposium on*. 78 – 83. DOI:http://dx.doi.org/10.1109/LPE.2000.155257

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.

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.

E. Cheung, H. Hsieh, and F. Balarin. 2009. Memory subsystem simulation in software TLM/T models. In *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*. 811 –816. DOI:http://dx.doi.org/10.1109/ASPDAC.2009.4796580

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.

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.

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.

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. 2013. A Scalable and Near-optimal Representation for Storage Size Management. *ACM Trans. Architecture and Code Optimization* conditionally accepted (2013).

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.

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 and others. 2007. *Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogenous Platforms* (1st ed.). Springer Publishing Company, Incorporated.

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

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).

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.

P. R. Panda and others. 2001b. Data and memory optimization techniques for embedded systems. *ACM Trans. Des. Autom. Electron. Syst.* 6, 2 (April 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.

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

NL Passes, EH-M 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.

T. Simunic, L. Benini, and G. De Micheli. 1999. Cycle-accurate simulation of energy consumption in embedded systems. In *Design Automation Conference, 1999. Proceedings. 36th*. 867 –872. DOI:http://dx.doi.org/10.1109/DAC.1999.782199