Summary of changes

We would first like to thank the reviewers for very useful comments to improve the paper. We have addressed them as explained below.

Referee 1

- *The term "irregular" in the paper used to describe the access patterns is not accurate and needs to be better defined. Even though there are access holes in the given example, the access patterns can still be well defined using simple mathematical formula.*

  The term irregular has been replaced with more descriptive terms, i.e. application with holes in their access pattern.

- *In the experimental evaluation, have you also compared the proposed methodology with the state-of-art energy optimization techniques for embedded systems? If so, what is improvement of your approach over theirs?*

  Several state-of-the-art techniques are presented as related work. Those approaches are not directly applicable for comparison, as it is discussed on Section 3. The related work is not directly applicable primarily due to differences in the memory architectures and the studied sets of benchmarks. The current work presents a reconfigurable memory architecture that exploits the dynamism in memory requirements during application's lifetime. The discussed related work is not designed to handle dynamic variations on memory requirements due to employment of a static memory platform. Thus, it is expected to provide poor results when applied to the dynamic applications targeted in the current work. The paper hence compares the current approach with previously published work from our groups.

  This has been clarified on Section 3

- *Interleaving is only one type of data layout optimization. Have you also tried other more complex optimizations? What is the main reason that you think interleaving is the most suitable/efficient one?*

  Array Interleaving is a data layout transformation for combining the storage of multiple arrays, so that blocks of data from different arrays are stored contiguously, with the objective of reducing the number of memory accesses through better spatial locality. [9]. The target applications on the current work benefit most from the proposed methodology, because they are characterized by having access patterns with

holes. Interleaving is a widely used technique that fits the goal of generating more compact sets of data. An important contribution of the work is the combination of the interleaving optimization with data to memory mapping. Another important advantage of interleaving transformations is the low addressing overhead. In more detail, interleaving results in a more regular global access by reducing the number of holes without increasing significantly the addressing for accessing the individual arrays afterwards. Although more complex data layout transformations may reduce the holes in the access patterns even more, they have a negative impact on the complexity of addressing and access overhead. Having shown the benefits of this, future work can include extending the methodology to be compatible with additional layout optimization techniques.

This has been clarified on Section 6.1

- *Do you think improving temporal locality of data accesses in cache is also important, if the cache is available? In embedded systems, there should also be a wide range of applications in which the same data need to be accessed multiple times? Thus if temporal locality is exploited, both stalls and number of accesses to main memory can be reduced.*

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

  This has been clarified on Section 4

- *Can you add references to the following: 1) simple energy model for evaluating the energy consumption for the motivation example; 2) the commercial memory compiler for SRAM memory models; 3) the XML based language used to describe the architecture.*

1. A quick estimate of the difference in the energy consumption between the approaches presented in Fig. 1 can be found using a

realistic energy model for the memory banks. The energy model, which has been also presented in [9], is derived from synthesized memory models presented in Sec. 4. The model is based on realistic estimates of the dynamic and static energy in the memory banks. The scope is to illustrate the motivation for this work, without extensively describing the detailed memory model characteristics. The target architecture and the energy models are described in depth in Sec. 4. For the simple model in this chapter, we assume that the static energy is 30% of the dynamic energy, which is a rational approximation for small memory banks with sizes between 1KB and 16KB. The static energy increases linearly with the memory size, which is a good approximation as also demonstrated by the detailed models in Sec. 4. Similarly the approximation for the correlation between the dynamic energy and the memory size is an approximation, which is simple but sufficiently accurate for the motivational example.

This has been clarified on Section 2

2. The commercial memory compiler is part of DesignWare Memory Compilers provided by Synopsys. The logic libraries support a wide range of foundries and process technologies from 250nm to 28nm. The memories are optimized for low power, high performance and high density. A 40nm library was chosen for the current work and a presentation of the memory cells can be found in [5]. Of confidentiality reasons we are not allowed to reveal details of the compiler or process and the presented numbers are relative.

   This has been clarified on Section 4.1

3. An XML based language is used to describe the architecture, and a cycle-accurate simulator of the processor is used to simulate the generated code on the architecture. The XML provides a structural way of describing the architecture presented in Fig. 2 including the different components, the parameters of each component, and the relationship between them. The XML description generates a graphical representation of the architecture and is the input for the simulator as presented in [7]. The chosen simulator is developed for coarse-grained reconfigurable architectures and is suitable in our case, because of the dynamic parameters of our architecture.

   This has been clarified on Section 4.2

- *At the end of page 3, there is such description "The interleaving of the arrays A, B, C and D is shown in Fig.1(c)"? Do you actually mean Fig.1(b)?*

  Yes. The error has been fixed.

- *In Page 9, please rephrase the redundant description, "For efficient utilization of the vector FU, the register file ..."*

  The appropriate changes have been made.

- *In the experimental part, can you comment on why you choose 5 banks? And what will be effect with different number of banks?*

  There are two main reasons for exploring architectures up to five memory banks. Firstly, the energy gains achieved by increasing the number of memory banks in the memory architecture are nearly saturated even for five banks. In [4] a group of different applications were studied with regard to their energy consumption on a clustered memory architecture consisting of up to five memory banks. The results show that depending on the application, the energy gains start to saturate after adding a third or a fourth bank and become insignificant when adding a fifth bank. Thus, for most applications a memory architecture with five memory banks already provides the necessary reconfiguration options. Secondly, the overhead increases exponentially with the number of memory banks, due to the increased complexity of the memory architecture. Therefore, memory architectures with six or more banks are typically not efficient options due to the high overhead and the low energy gain.

  This has been clarified in the first paragraph on Section 5.3.

- *Section 6 and Section 7 can be combined as single Section for experimental evaluation.*

  The appropriate changes have been made.

- *Typos:*
  Page 4: we assume here that he memory →we assume here that the memory
  Page 4: using four banks is presented in Fig.1(b) →using four banks is presented in Fig.1(c)

Page 4: The data-to-memory mapping for the constructed →The optimized data-to memory mapping for the constructed
Page 6: that is always accesses →that is always accessed
Page 7: must developed →must be developed
Page 8: is higher compared to sleep mode →is higher compared to light sleep mode
Page 14: This application is an representative →This application is a representative

The typos have been fixed.

Referee 2

- *However, the paper does not clearly distinguish the main contribution and the difference from the previous work. Original contribution needs to stand out clearly.*

  The contribution of the proposed work is the development of a combined approach that investigates the interleaving and memory mapping options for a reconfigurable SIMD architecture. The current work combines and expands in a non-trivial way, the interleaving exploration presented in [8] and the data to memory mapping methodology presented in [4]. The current work is more than a simple application of the two approaches, one after the other. Such an approach cannot always guarantee a viable solution. The reason is that for each different interleaving solution, there are a number of constraints on the placement of the data into the memory due to hardware limitations. If the constraints are not propagated into the data-to-memory mapping step, the final solution suffers from data conflicts. Different interleaving solutions introduce different constraints. Therefore, there is a need to develop an integrated methodology to achieve the improvements of both the approaches.

  This has been clarified at the end of Section 3.

- *It would be interesting to see more workloads analyzed in the framework.*

  Unfortunately, it was impossible to analyze more benchmarks for the current revision. The chosen applications are representative candidates for the multimedia and the wireless domains. We believe that the results are also representative for other applications with the same characteristics.

- *The paper focuses on SIMD architectures but it does not make any reference memory mapping optimization on GPUs. Also it would be interesting to see if the framework can capture more irregular access patterns and data mapping schemes (e.g. permutations, indirect addresses, etc.).*

  The presented methodology should indeed be applicable in these cases as well. Some modifications would be needed, however. For example, GPUs can exploit computation parallelism, so data should be mapped in different memory banks to be accessed in parallel.

- *Some references worth discussing: Dymaxion: optimizing memory access patterns for heterogeneous systems, SC'11 Data reorganization in memory using 3D-stacked DRAM, ISCA'15 DL: A data layout transformation system for heterogeneous computing, InPar'12*

  In [2] the authors tackle the problem of sub-optimal data structure layouts in GPUs with a large number of parallel cores, especially for programs that are designed with a CPU memory interface in mind. An API is presented, that allows programmers to improve CUDA programs by optimizing memory mappings in order to increase the efficiency of memory accesses. The main differences of the current work are the platform and the types of code transformations. We focus on an SIMD CPU and a dynamic clustered scratchpad memory compared to multicore GPUs and a static memory. We differentiate by focusing on interleaving as the preferred code transformation, being suitable for the target applications. Another work that discusses memory layout for GPUs is presented in [11]. The authors focus on off-chip DRAM memory optimization using a number of data layout transformations. We differentiate by focusing on the memory closest to the CPU. We also study data interleaving while the main focus in [11] is the transformations that increase data parallelism, which is more important for their multicore GPU architecture. In [1] a number of common data reorganization operations such as shuffle, pack/unpack, swap, transpose, and layout transformations are presented. The goal is to study the cost of applying these operations in the memory at run-time. The target memory is 3D-stacked DRAM and additional hardware is employed in order to efficiently perform the reorganization operations with a low overhead. Apart from the different type of platform, the current work differentiates in the type of data reorganizations and the mapping of the reorganized data to the scratchpad memory at compilation time.

  We have added these references to our related work overview

in Section 3.

Referee 3

- *In particular, what microarchitecture enhancement is included and how access patterns are extracted (in SW/HW) ? How address mapping is implemented and its policy? How interleaving is handled when there are bank conflicts and imperfect coalescing for SIMDs. In general, many details are missing. Fig 3 is not very helpful.*

  The application is fully analyzed at design-time, because of the time consuming nature of the task. The access patterns are extracted in software and the possible interleaving options are explored. The mapping to the specific target architecture is also fully decided at design-time. The data-to-memory mapping exploration takes into consideration the platform and code limitation and proposes a mapping without data and bank conflicts. The employment of a scratchpad memory architecture is crucial for this. In contrast to cache memory systems, in which the mapping of data elements is done at run-time, in scratchpad memory systems the mapping is performed by the programmer or the compiler [6]. The address mapping follows the principles presented in [3] and [9]. Unlike the cache memory, the scratchpad memory does not need tag search operations and it is the responsibility of the programmers or compilers to correctly allocate code and data on the scratchpad memory [10]. This is possible for small embedded systems designed to run one or a limited set of specific applications. For other types of systems and applications, a cache memory can be the preferred solution as the detection of a cache hit or miss is done automatically and any conflicts can also be resolved by the platform at run-time. In our case, the application and the memory system are fully analyzed and the allocation of data to a scratchpad memory can be easily done and offers an energy efficient solution.

  This has been clarified on Section 5. Changes have been made to better explain the platform and the flow presented in Fig. 3

- *Also, the authors may look into the recent work by Akin, et al. "Data Reorganization in Memory Using 3D-stacked DRAM", ISCA 2015.*

  In [1] a number of common data reorganization operations such as shuffle, pack/unpack, swap, transpose, and layout transformations are presented. The goal is to study the cost of applying these operations

in the memory at run-time. The target memory is 3D-stacked DRAM and additional hardware is employed in order to efficiently perform the reorganization operations with a low overhead. Apart from the different type of platform, the current work differentiates in the type of data reorganizations and the mapping of the reorganized data to the scratchpad memory at compilation time.

We have added this reference to our related work overview in Section 3.

- *A minor point is the chosen benchmarks are not very irregular if appropriate data layouts are chosen. This is fine, but the authors may consider other possibilities such as graphs and sparse matrices.*

  The term irregular has been replaced with more descriptive terms, i.e. application with holes in their access pattern, as also noted by Referee 1. Having shown the effectiveness of the technique for the chosen benchmarks, future work can extend the irregularity to include graphs and sparse matrices.

References

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

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

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

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

[5] Tobias Gemmeke, Mohamed M Sabry, Jan Stuijt, Praveen Raghavan, Francky Catthoor, and David Atienza. Resolving the memory bottleneck for single supply near-threshold computing. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE, 2014.

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

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

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

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

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

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