Compiler Infrastructure for Specializing Domain-Specific Memory Templates

Stephanie Soldavini
Politecnico di Milano
Milan, Italy
stephanie.soldavini@polimi.it

Christian Pilato
Politecnico di Milano
Milan, Italy
christian.pilato@polimi.it

ABSTRACT

Specialized hardware accelerators are becoming important for more and more applications. Thanks to specialization, they can achieve high performance and energy efficiency but their design is complex and time consuming. This problem is exacerbated when large amounts of data must be processed, like in modern big data and machine learning applications. The designer has not only to optimize the accelerator logic but also produce efficient memory architectures. To simplify this process, we propose a multi-level compilation flow that specializes a domain-specific memory template to match data, application, and technology requirements.

1 INTRODUCTION

Domain-specific accelerators are one of the key solutions to continue increasing performance and efficiency beyond the end of Moore's law scaling [2, 3]. These accelerators use only the minimal required resources, consume less power, and compute faster than general purpose hardware [4]. However, the design of such components is complex [2].

Modern big data and machine learning applications need to process huge and potentially distributed data sets with stringent requirements. Managing these data sets requires a combination of different solutions to hide the communication latency and exploit the inherent data parallelism [13]. Researchers proposed accelerators with local caches and private local memories for storing data on chip, while multiple channels help combine classic DRAM with non-volatile memories (NVM) for off-chip data. Memory architectures with **intelligent data transfers** can greatly optimize the systems but require specialization based on the application [10].

On one side, domain-specific languages like Spatial [5] can abstract memory operations while still being hardware-oriented, but they miss a complete tool-flow to port software-oriented algorithms to hardware. High-level synthesis (HLS) is a technology to automatically generate hardware modules starting from high-level descriptions [1, 11] but memory optimization is still an open problem [16]. This line of research proposes a compiler-based approach for optimizing the accelerator memories on top of traditional HLS. The main idea is to use *domain-specific annotations* to pass useful information to the compiler, transform the intermediate representations, and interface directly with modern HLS tools.

LATTE '21, April 15, 2021, Virtual, Earth

2 HIGH-LEVEL SYNTHESIS: THE PRESENT

High-level synthesis helps raise the abstraction level and use high-level, software-like methods for hardware design. Modern high-level synthesis tools are based on state-of-the-art compilers to extract a language-agnostic intermediate representation from common software languages [11]. Using compiler frontends also allows designers to apply common compiler transformations like constant propagation, dead-code elimination, and loop transformations. For example, most HLS tools use the GCC or LLVM compilers to apply state-of-the-art compiler transformations and extract the resulting intermediate representation. In the following phase, the HLS engine determines how to distribute the operations over time (scheduling) and over the hardware resources (allocation and binding). These steps determine the hardware architecture of the *controller*, which determines the evolution of the circuit in each clock cycle, and the *datapath*, which contains the hardware resources and their interconnections.

Current HLS tools have strong focus on the computational aspects, while the surrounding memory architecture is adapted to merely sustain the required data accesses. In case of data-intensive applications, the optimizations should focus more on coordinating memory transfers and accesses, rather than on the actual computation. To do so, compilers need to integrate, propagate, and expose more data-related information. If passed to the HLS engine, this information can help specialize the memory architecture together with the accelerators.

3 DOMAIN-SPECIFIC MEMORY TEMPLATE

Specialized architectures are designed specifically for an accelerator, but the process is time consuming and must be done for each new design. *Domain-specific architectures* are more general since the structure can be reused across multiple applications, sacrificing performance. For the memory aspects of a hardware accelerator, we propose an approach in between, using a **domain-specific template** that allows the specialization of particular components.

The lower part of Figure 1 shows the proposed template. It is composed of existing memory primitives, like caches, DMA engines, prefetchers, and multi-port memories. Based on given area constraints, only part of the data can stay on chip, while the rest is stored in DRAM or non-volatile memories (either on the same device or remotely). On-chip data are stored in different memories based on the application data structures but also the type of accesses that are expected. Irregular accesses can be implemented with custom **latency-insensitive memory architectures** [9]. Data with regular accesses can be stored in fixed-latency **private local memories** (PLMs) and customized with multi-bank configurations to expose a large number of ports to the accelerator logic. Data reuse

buffers can remove unnecessary data transfers. Data accesses with a certain degree of locality can benefit from architectures featuring **caches** that are local or shared with the processor by means of a coherent protocol [7, 18]. We also feature a **direct-memory access** (DMA) engine to make the data transfers more efficient and a **prefetcher** to anticipate known data transfers to hide the communication latency. These IP blocks can be augmented with *special functions*, like data protection (e.g., encryption) or application-specific transformations (e.g., matrix transpose).

This template is general enough to be reused across multiple applications but it can also be specialized based on the accelerator characteristics. For instance, we can vary the number of ports on a multi-bank memory based on the specific access patterns of the application. Also, components can be removed if they are unnecessary for the application. For example, if the data resides entirely on-chip, the prefetcher can be removed or if there is only a single memory, the multi-channel controller can be simplified. We propose to use a compiler-based approach to progressively refine such template.

4 SPECIALIZATION OF THE MEMORY TEMPLATE

To achieve better performance and reduce costs, designers can specialize the memory template based on the given accelerator. For this, our approach is based on the idea of *platform-based design* [17], where the memory template is refined in different stages, starting from the general organization of the data in memory to the actual interaction with the actual accelerator. The upper part of Figure 1 shows our compiler-based customization flow.

Intermediate Representation. The compiler infrastructure will need to include more hardware-related information. We target novel multi-level representations, like MLIR [6], to include more hardware-related information early in the compilation flow to make progressive refinements of the architecture at proper levels of abstraction. A novel flow is required because existing approaches are not fully compatible with HLS. CIRCT [19] proposes MLIR extensions for low-level hardware synthesis (below the HLS level). Calyx [12] follows, instead, a different approach with a novel IR and associated compiler. SODA [8] proposes a MLIR-based synthesis framework for machine learning accelerators with more focus on the computational aspects.

Compilation Flow. We extend the LLVM-MLIR compilation flow with additional passes to include memory-related information and transform the IR accordingly. Our passes include solutions to define the data layout, size the physical memories (both caches and PLMs), optimize the access patterns, and create multi-port PLMs for fast access. Currently, we use custom generators like Mnemosyne[1] to derive the HDL descriptions from such information. We will also investigate the possibility to interface directly with MLIR formats for hardware, like CIRCT.

The customization flow shown at the top of Figure 1 would proceed as follows: At the highest abstraction level, the *data organization* phase analyzes the data representations to determine the coarse memory structure, i.e. deciding which data are stored off-chip or on-chip. The next step, the *layout* phase, reorganizes

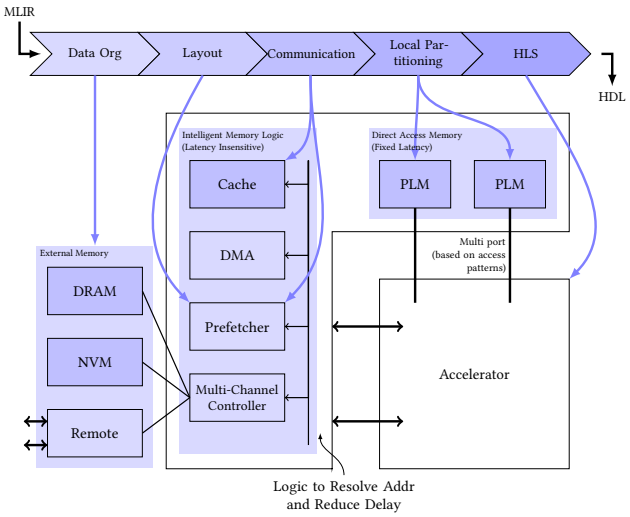

Figure 1: Multi-level compilation flow for the specialization of domain-specific memory architectures.

the computation to better exploit local memories (either caches or PLMs). Then, in the *communication* phase, the prefetcher is configured to hide transfer latency based on the data access patterns. After this, the *local partitioning* phase determines the multi-bank PLM architecture, also sharing physical memories for data with disjoint lifetimes [16]. Finally, the *HLS* phase generates the computation part of the component with traditional HLS, producing the complete syntesizable description of the accelerator.

Accelerator Logic HLS. With our approach, the accelerator is designed only at the end of the flow according to the resulting memory organization. The accelerator features state-of-the-art solutions for memory management (e.g., dynamic address resolution [14, 15]). The accelerator is mostly unaware of the data organization and layout since the IR has been already updated based on the memory transformations. It is only optimized to efficiently access the data with fixed or unbounded latency. This part can leverage existing HLS tools that start from low-level intermediate representations. For example, the final LLVM IR representation can be directly interfaced with the Xilinx Vitis HLS front-end[2].

5 CONCLUSION

We described a novel approach for specializing domain-specific memory templates during the compilation flow and *before* high-level synthesis of the accelerator logic. Starting from a high-level memory template, we apply a multi-level compilation flow based on MLIR that progressively refines the memory architecture and then interfaces with commercial HLS tools. Our approach borrows idea from platform-based design, trading off flexibility and specialization based on specific needs of the designers.

ACKNOLEDGEMENTS

This project is partially funded by the EU Horizon 2020 Programme under grant agreement No 957269 (EVEREST).

[1] http://github.com/chrpilat/mnemosyne

[2] https://github.com/Xilinx/HLS

REFERENCES

[1] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang. 2011. High-Level Synthesis for FPGAs: From Prototyping to Deployment. *IEEE Transactions on CAD of Integrated Circuits and Systems* 30, 4 (2011), 473–491.

[2] W. J. Dally, Y. Turakhia, and S. Han. 2020. Domain-Specific Hardware Accelerators. *Comm. of the ACM* 63, 7 (July 2020), 48–57.

[3] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. 2012. Dark Silicon and the End of Multicore Scaling. *IEEE Micro* 32 (2012), 122–134. Issue 3.

[4] M. Horowitz. 2014. 1.1 Computing's energy problem (and what we can do about it). *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 10–14.

[5] D. Koeplinger, M. Feldman, R. Prabhakar, Y. Zhang, S. Hadjis, R. Fiszel, T. Zhao, L. Nardi, A. Pedram, C. Kozyrakis, and K. Olukotun. 2018. Spatial: A Language and Compiler for Application Accelerators. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*.

[6] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko. 2020. MLIR: A Compiler Infrastructure for the End of Moore's Law. arXiv:2002.11054

[7] P. Mantovani, D. Giri, G. Di Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni. 2020. Agile SoC Development with Open ESP. In *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design (ICCAD)*.

[8] M. Minutoli, V. G. Castellana, C. Tan, J. Manzano, V. Amatya, A. Tumeo, D. Brooks, and G. Y. Wei. 2020. SODA: a New Synthesis Infrastructure for Agile Hardware Design of Machine Learning Accelerators. In *Proceedings of the IEEE/ACM International Conference On Computer-Aided Design (ICCAD)*.

[9] M. Minutoli, V. G. Castellana, A. Tumeo, M. Lattuada, and F. Ferrandi. 2016. Enabling the High Level Synthesis of Data Analytics Accelerators. In *Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*.

[10] O. Mutlu. 2020. Intelligent Architectures for Intelligent Machines. *Proceedings of the IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*.

[11] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels. 2016. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 10 (Oct. 2016), 1591–1604.

[12] R. Nigam, S. Thomas, Z. Li, and A. Sampson. 2021. A Compiler Infrastructure for Accelerator Generators. In *Proceedings of ACM SIGPLAN Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[13] C. Pilato, S. Bohm, F. Brocheton, J. Castrillon, R. Cevasco, V. Cima, R. Cmar, D. Diamantopoulos, F. Ferrandi, J. Martinovic, G. Palermo, M. Paolino, A. Parodi, L. Pittaluga, D. Raho, F. Regazzoni, K. Slaninova, and C. Hagleitner. 2021. EVEREST: A design environment for extreme-scale big data analytics on heterogeneous platforms. In *Proceedings of the ACM/IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[14] C. Pilato and F. Ferrandi. 2013. Bambu: A modular framework for the high level synthesis of memory-intensive applications. *Proceedings of the IEEE International Conference on Field programmable Logic and Applications (FPL)*.

[15] C. Pilato, F. Ferrandi, and D. Sciuto. 2011. A Design Methodology to Implement Memory Accesses in High-level Synthesis. *Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*.

[16] C. Pilato, P. Mantovani, G. Di Guglielmo, and L. P. Carloni. 2017. System-Level Optimization of Accelerator Local Memory for Heterogeneous Systems-on-Chip. *IEEE Transactions on CAD of Integrated Circuits and Systems* 36, 3 (2017), 435–448.

[17] A. Sangiovanni-Vincentelli and G. Martin. 2001. Platform-Based Design and Software Design Methodology for Embedded Systems. *IEEE Design & Test* 18, 6 (Nov. 2001), 23–33.

[18] Y. S. Shao, S. L. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks. 2016. Co-Designing Accelerators and SoC Interfaces Using Gem5-Aladdin. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[19] A. Wilson, S. Neuendorffer, and C. Lattner. 2020. CIRCT: Circuit IR Compilers and Tools. https://github.com/llvm/circt.