Leveraging MLIR for Efficient Irregular-Shaped CGRA Overlay Design

(PhD Forum Paper)

Mohamed Bouaziz, Suhaib A. Fahmy
King Abdullah University of Science and Technology (KAUST)
Thuwal, Saudi Arabia
Email: mohamed.bouaziz@kaust.edu.sa

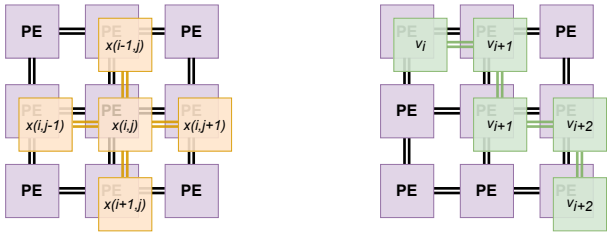
***Abstract*—Coarse-grained reconfigurable arrays (CGRAs) are reconfigurable architectures that combine the efficiency of custom datapaths with the flexibility of FPGAs. Optimized word-level functional units with reconfigurable interconnect enable a wide range of applications. Due to the irregular data movement patterns and the diversity of real-world HPC applications, a traditional homogeneous regular-shaped CGRA architecture can be sub-optimal. A more application-specific arrangement of resources could, however, result in poor hardware reuse and increased deployment effort for a CGRA. Using CGRA overlays can increase hardware reuse while maintaining high performance of a more optimized architecture. This proposal uses and builds upon the built-in tools of the MLIR infrastructure to target a class of applications that can be analyzed, rewritten, and optimized based on the available resources of an underlying CGRA architecture, thereby better exploiting hardware resources.**

I. Problem and Motivation

Coarse-Grained-Reconfigurable-Arrays (CGRAs) are reconfigurable architectures comprising an array of interconnected processing elements (PEs) capable of performing different computational functions. Recently, the interest in CGRAs has renewed thanks to their capability in compute acceleration, fast compilation, and portability to various applications. However, traditional regular-shaped rectangular architectures differ from the irregular-shaped data movement patterns in many real-world applications, thereby missing opportunities for efficient compilation and hardware utilization.

Fig. 1 illustrates example application kernels mapped onto a 3×3 CGRA where Fig. 1a overlays the data access pattern of a 2D Jacobi stencil kernel and Fig. 1b overlays the data access pattern of an FIR filter kernel . In both examples, the top right PE and the bottom left PE are not utilized. However, as the compiler is oblivious to the data access patterns, it includes those PEs in the search space when it searches for a feasible mapping of both applications on the underlying architecture. More generally, the problem concerns all the hardware resources, including the width of interconnect, the structure of the memories and the registers, and the supported operations by the processing elements. As more complex architectures include more hardware components, the search space grows accordingly, increasing compilation effort.

In this work, we propose generating CGRA overlays based on a target class of applications. Analyzing and optimizing the application kernels results in transforming the kernels for better-shaped data access patterns at the application level, reducing the compilation search space at the compiler level, and increasing the reusability of the hardware by accommodating more overlays on the same architecture at the hardware level.



(a) 2D Jacobi (b) FIR Filter

Fig. 1: Mapping examples

II. Background and Related Work

Several works have targeted optimizing acceleration of HPC applications on CGRAs.

REVAMP [1] proposed an approach for optimizing the hardware architecture by analyzing a set of input applications targeting the reduction of area and power and maintaining the same throughput. However, it does not explore the opportunities of transforming the application kernels and targets generating heterogeneous CGRA architectures rather than overlays that reuse the same hardware. FlexC [2], contrary to REVAMP, explored rewriting application kernels to reduce hardware requirements. It explored opportunities for a set of input applications to use fewer hardware components by rewriting them into fewer operations, which increases hardware reuse. However, it does not perform optimizations on the hardware itself. OverGen [3] proposed a framework for domain-specific overlay generation targeting FPGAs for ease of compilation of applications, as compiling to FPGAs is highly time-consuming. However, OverGen does not exploit existing CGRA architectures and focuses mainly on reducing the compilation time to FPGAs. There is also work demonstrating that such coarse grained overlays can be deployed on FPGAs in an architecture-centric manner, thereby extracting much higher performance than is possible through architecture agnostic design [4].

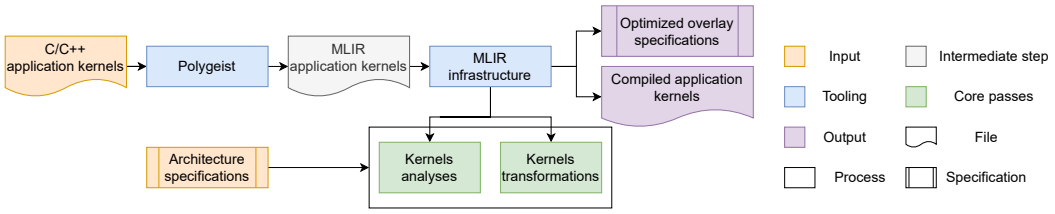


Fig. 2: Optimized overlay generation process.

TABLE I: Utilized resources per AI Engine.

Kernel	Tile Size	Vector Update ops.	Multiply ops.	MAC ops.
2D Convolution	16×16	6	1	8
2D Jacobi	16×16	6	1	4

III. Approach and Uniqueness

In this work, we propose leveraging the Multi-Level Intermediate Representation (MLIR) compiler infrastructure [5] to analyze and transform the input application kernels and generate the appropriate overlay based on an input architecture specification. This approach differs from the previous work by applying the analyses and the transformations on multiple levels, using MLIR, instead of on the bitcode level, using LLVM, thereby capturing more information about the data access patterns and requirements of the operations upwards from an abstract view close to the kernel code and downwards from a lower view close to the architecture.

As shown in Fig. 2, the input application kernels are parsed using the Polygeist [6] C/C++ frontend to generate MLIR files. Then, the MLIR infrastructure is leveraged to perform several passes on the kernels to analyze them and apply relevant transformations, considering the input architecture specifications. The process results in outputting the specifications of an optimized overlay and the bitcode of the application kernels.

Notably, one of the key features of MLIR is the ability to target different architectures at different levels of abstraction, making porting the overlay and the compiled kernels flexible to targeting different hardware. One possible target is FPGAs, which is feasible using the CIRCT framework [7] for hardware compilation using MLIR, in particular by using Calyx [8].

IV. Preliminary Results

We target the commercial CGRA architecture of the AMD Versal ACAP VCK5000, which comprises 400 (8×50) PEs, called AI Engine. Each AI Engine is optimized for vector operations with a scratchpad memory of 32 KB, shared with neighbouring PEs in the cardinal directions.

We target two application kernels, 2D Convolution and 2D Jacobi, with a 3×3 sliding window. We employ MLIR's `affine-loop-tile` pass with a `cache-size` of 32KB to fit within the AI Engine memory limits. Subsequently, we use `affine-loop-unroll` with `unroll-full` and `unroll-full-threshold=3` to fully unroll loops with trip counts up to 3, unrolling the element-wise operations in the sliding window. We then apply `affine-super-vectorize` with `virtual-vector-size=8` for 8-lane SIMD vectorization. Lastly, the `aie-vectorize` pass in the MLIR-AIE framework lowers MLIR code to `aievec` dialect for the AMD Versal ACAP architecture.

Table I shows the resulting overlay configuration. The 8×50 AI Engine PEs are allocated 16×16 single precision float elements of the 2D input grid. We notice that both kernels require an exact number of vector updates, translating into similar data access patterns. For the innermost loop, the number of the multiply operations is also similar, while the number of the multiply-accumulate (MAC) operations differs slightly, translating into slightly different computation behavior. This results in using the same overlay architecture with a mere change in the innermost loop computation operations that run on the vector RISC-V cores.

V. Expected Results and Contributions

Extending this work to incorporate more analysis and optimization passes will contribute to better designing overlays for a wide range of architectures as the MLIR ecosystem is bridging the gap between the hardware and the software through compilation support. Additionally, not only does it ease the reachability of the variety of architectures, but it also optimizes the implementation further, reducing the development cost.

References

[1] T. K. Bandara, D. Wijerathne, T. Mitra, and L.-S. Peh, "REVAMP: a systematic framework for heterogeneous CGRA realization," in *Proceedings of ASPLOS*, 2022.

[2] J. Woodruff, T. Koehler, A. Brauckmann, C. Cummins, S. Ainsworth, and M. F. P. O'Boyle, "Rewriting history: Repurposing domain-specific CGRAs," in *arXiv*, 2023.

[3] S. Liu, J. Weng, D. Kupsh, A. Sohrabizadeh, Z. Wang, L. Guo, J. Liu, M. Zhulin, R. Mani, L. Zhang, J. Cong, and T. Nowatzki, "OverGen: Improving FPGA usability through domain-specific overlay generation," in *Proceedings of MICRO*, 2022.

[4] A. K. Jain, D. L. Maskell, and S. A. Fahmy, "Coarse grained FPGA overlay for rapid just-in-time accelerator compilation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1478–1490, 2021.

[5] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "MLIR: scaling compiler infrastructure for domain specific computation," in *Proceedings of CGO*, 2021.

[6] W. S. Moses, L. Chelini, R. Zhao, and O. Zinenko, "Polygeist: Raising C to polyhedral MLIR," in *Proceedings of PACT*, 2021.

[7] S. Eldridge, P. Barua, A. Chapyzhenka, A. Izraelevitz, J. Koenig, C. Lattner, A. Lenharth, G. Leontiev, F. Schuiki, R. Sunder *et al.*, "MLIR as hardware compiler infrastructure," in *Proceedings of WOSET*, 2021.

[8] R. Nigam, S. Thomas, Z. Li, and A. Sampson, "A compiler infrastructure for accelerator generators," in *Proceedings of ASPLOS*, 2021.