SHADE: A Software and Hardware Co-design Infrastructure for EDDO Architectures

Jingqun Zhang*
Georgia Tech
USA

Devin Pohl*
Georgia Tech
USA

Prasanth Chatarasi
IBM Research
USA

Jun Shirako
Georgia Tech
USA

Vivek Sarkar
Georgia Tech
USA

Cong Hao
Georgia Tech
USA

1 INTRODUCTION

The Explicit Decoupled Data Orchestration (EDDO) paradigm is gaining popularity in machine learning accelerators thanks to its superior efficiency compared to conventional cache-based (Implicit Coupled) architectures. This paradigm, as illustrated in Figure 1, decouples data movement from execution, allowing for improved dataflow control and optimization [1–8, 10]. Notable EDDO accelerator designs include Eyeriss [1, 2], IBM AIU [11], NVIDIA SIMBA [10], Morph [5], MAERI [7], and Extensor [6]. These architectures typically feature distributed scratchpads and programmable units specialized for memory management, address generation, computation, networking, etc. Systems employing these specialized units are proven more efficient compared to those with generalized homogeneous engines. While early instances of EDDO systems were fixed-function, recent trends prioritize flexibility, supporting *similar* workloads without hardware reconfiguration.

Existing EDDO software compilers are majorly limited, typically targeting *specific* EDDO architecture implementations. Broadening support requires time-consuming modifications, prohibiting design exploration for different workloads. Our work aims to address this challenge by: (1) formulating a hardware specification language capable of accommodating *various* EDDO designs (2) developing a software compiler tailored to the abstract hardware specification, and (3) developing a hardware compiler to implement the provided specification. To ensure the feasibility of this ambitious project, we constrain our initial scope in input to (1) affine dense programs with static workloads and (2) programmer-specified data movement parallelism/synchronization. Hardware is similarly constrained to (1) one-dimensional vector engines with simple FMA operations (primarily HPC/ML-oriented PolyBench kernels) and (2) Load/Store units with simple data and synchronization operations.

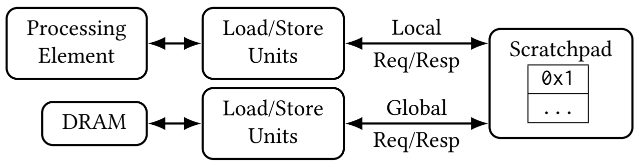

Figure 1: Explicit Decoupled Data Orchestration Paradigm

2 FULL-STACK SOLUTION

As depicted in Figure 2, our comprehensive full-stack architecture is segmented into two principal components: the Software Compiler and the Hardware Compiler. The Hardware Compiler takes the high-level hardware design expressed in the hardware specification language, translating it into programmable hardware and ensuring that the specified designs are accurately rendered into physical components. In parallel, the Hardware Intermediate Representation (IR) is passed to the Software Compiler, which is responsible for mapping algorithms and code generation onto the generated hardware. Subsequently, a set of decoupled heterogeneous executables are produced, ready to run on the generated physical components.

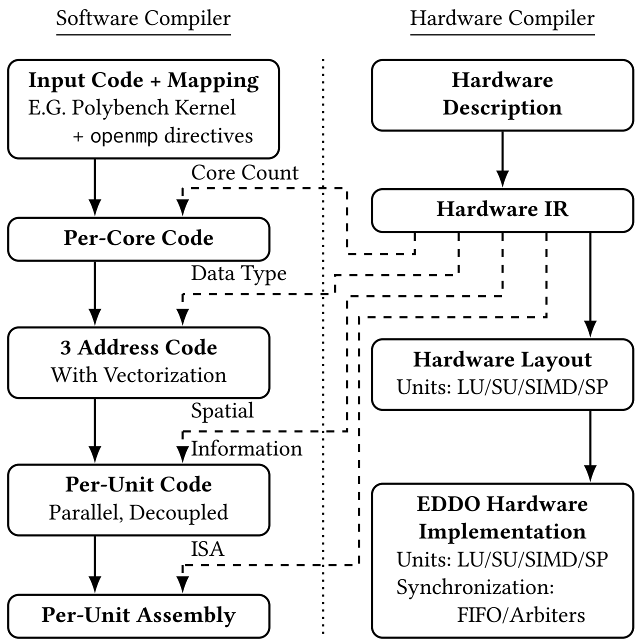

Figure 2: Co-Compilation Process

2.1 Hardware Compiler

EDDO architectures are characterized by a distributed arrangement of scratchpads, programmable compute engines, programmable load/store units, and an interconnected network linking these components. We present a **Hardware Specification Language** enabling users to define EDDO designs using a directed graph methodology, where nodes represent components (e.g. scratchpads, compute engines, load/store units) and edges denote data transfer interconnections. Each node contains component specification including vector register file size, precision, or supported compute operations. Previous research by Parashar et al. [8] presented a polyhedral-based HST abstraction for EDDO designs. However, this method employs piece-wise affine expressions when dealing with interconnection networks that cannot be adequately represented using regular affine expressions, resulting in increased complexities.

The hardware compiler empowers users to tailor the data path according to the spatial structure they wish to investigate. It achieves this by sharing the abstract EDDO design with the compiler via translation of the provided user specification to a Hardware IR. This IR incorporates spatial information, including EDDO layout (for correctness) and cost models (for optimization). The IR is then transformed into High-Level Synthesis (HLS) code. Ultimately, the HLS toolchain generates the implementation hardware.

2.2 Software Compiler

The software compiler will leverage the Hardware IR constructed by the hardware compiler to produce decoupled per-unit executables including data movement, computation, and synchronization instructions. Summarized in Figure 2, the software compiler will:

(1) Leverage parallelism in the source program – we currently target only programs with affine loopness (via C + OpenMP) – to generate code for a multi-core decupled architecture
(2) Apply distribution and vectorization to the source program according to the capabilities and layout of the SIMD units
(3) Separate each per-core program into per-unit programs by expanding and identifying load, store, and SIMD instructions while inserting required synchronization instructions
(4) Optimize speed and synchronization overhead according to the connection cost model supplied within the Hardware IR
(5) Assemble according to the HW compiler's generated ISA

Through this approach, source programs may be flexibly mapped to user-defined hardware without the need to consider the target EDDO architecture from the perspective of the source program. Pending data on optimal hardware/application matching, future work may include the software compiler advising the hardware compiler of generalized EDDO topology families it believes to be optimal based on input program characteristics.

3 HARDWARE IMPLEMENTATION

Data transfer between units is implemented with FIFOs, which conveniently also manage the bulk of unit-level synchronization while also simplifying inter-unit connections and control. More detailed explanations of each unit, as explified in Figure 3, include:

- Single Instruction Multiple Data Unit (SIMD): In-order execution without branch prediction, prefetching, or caching.
- Load/Store Unit (LU/SU): Fine-grained memory accessors with synchronization instructions generated during static analysis routed through arbiters.
- Load/Store Arbiter: Effective memory synchronization is provided with straightforward control logic and a low synchronization connection overhead of only $2N$ (where N is the number of scratchpads) [9], facilitating multithreaded operations for user-defined design space exploration.

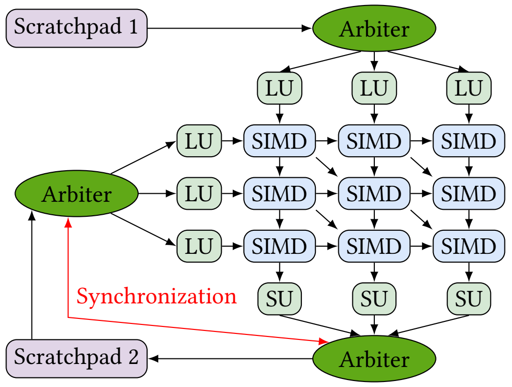

Figure 3: Example of Generated Hardware Implementation

4 MEMORY SYNCHRONIZATION

Although forward synchronization is implicitly handled via FIFO connections, anti- and output-dependencies on scratchpads require explicit synchronization signals. These signals are implemented as "lock" and "unlock" instructions, inserted during static analysis.

To reduce synchronization connection overhead, we implement synchronization logic at arbiters – the local convergence points between load/store units. Our proposed scheme decreases the synchronization interconnection complexity by a factor of $LUs \times SUs$ [9] over a naive N-N scheme. We plan to extend this approach to the entire memory hierarchy (e.g. between scratchpads and DRAM).

5 INITIAL RESULTS AND FURTHER WORK

EDDO architectures can achieve better performance and energy efficiency at the cost of program/compiler complexity. We present SHADE, a full-stack solution that mitigates this complexity increase, enabling efficient design space exploration of EDDO architectures. Our key contributions include open-source tools targeted towards user exploration, insights for optimal hardware and application matching, and EDDO-friendly memory management techniques.

Future work will include improving SHADE's ISA-based SIMD implementation, driven by unbalanced FPGA resource allocation shown in Figure 4's preliminary results. Therefore, we intend to introduce SIMD unit variants specialized for specific workloads. Leveraging the maturity this will bring to our flexible ISA scheme, we will also explore low design-overhead methods of supporting user-defined custom engines (e.g. Vision Transformer or Dynamic GNN). In the software compiler, we will investigate alternative methods of dependency expression; while C + OpenMP excels in dense applications, it falls short in sparse applications. Finally, dynamic control flow is an intriguing challenge we want to address.

xc7z020	DSP	FF	LUT
SIMD (10 Cores)	10(4%)	2033(1%)	7114(13%)
SU	—	16(0%)	306(0%)
LU	—	15(0%)	207(0%)

(a) Synthesis Resource Estimates

Frequency (Hz)	100M
Cycles #	403800
CPI	2

(b) Performance

Figure 4: Preliminary Results based on One SIMD Core, Executing the First Loop of PolyBench GemVer

REFERENCES

[1] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH computer architecture news* 44, 3 (2016), 367–379.

[2] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308.

[3] Bruce Fleischer, Sunil Shukla, Matthew Ziegler, Joel Silberman, Jinwook Oh, Vijavalakshmi Srinivasan, Jungwook Choi, Silvia Mueller, Ankur Agrawal, Tina Babinsky, et al. 2018. A scalable multi-TeraOPS deep learning processor core for AI trainina and inference. In *2018 IEEE symposium on VLSI circuits*. IEEE, 35–36.

[4] Brian Gaide, Dinesh Gaitonde, Chirag Ravishankar, and Trevor Bauer. 2019. Xilinx adaptive compute acceleration platform: VersalTM architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 84–93.

[5] Kartik Hegde, Rohit Agrawal, Yulun Yao, and Christopher W Fletcher. 2018. Morph: Flexible acceleration for 3d cnn-based video understanding. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 933–946.

[6] Kartik Hegde, Hadi Asghari-Moghaddam, Michael Pellauer, Neal Crago, Aamer Jaleel, Edgar Solomonik, Joel Emer, and Christopher W Fletcher. 2019. Extensor: An accelerator for sparse tensor algebra. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 319–333.

[7] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. *ACM SIGPLAN Notices* 53, 2 (2018), 461–475.

[8] Angshuman Parashar, Prasanth Chatarasi, and Po-An Tsai. 2021. Hardware abstractions for targeting EDDO Architectures with the Polyhedral Model. In *11th International Workshop on Polyhedral Compilation Techniques (Vitural Event)(IMPACT 2021)*.

[9] V. Sarkar. 1988. Synchronization using counting semaphores. In *Proceedings of the 2nd International Conference on Supercomputing* (St. Malo, France) *(ICS '88)*. Association for Computing Machinery, New York, NY, USA, 627–637. https://doi.org/10.1145/55364.55426

[10] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. 2019. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 14–27.

[11] Swagath Venkataramani, Vijayalakshmi Srinivasan, Wei Wang, Sanchari Sen, Jintao Zhang, Ankur Agrawal, Monodeep Kar, Shubham Jain, Alberto Mannari, Hoang Tran, Yulong Li, Eri Ogawa, Kazuaki Ishizaki, Hiroshi Inoue, Marcel Schaal, Mauricio Serrano, Jungwook Choi, Xiao Sun, Naigang Wang, Chia-Yu Chen, Allison Allain, James Bonano, Nianzheng Cao, Robert Casatuta, Matthew Cohen, Bruce Fleischer, Michael Guillorn, Howard Haynie, Jinwook Jung, Mingu Kang, Kyu-hyoun Kim, Siyu Koswatta, Saekyu Lee, Martin Lutz, Silvia Mueller, Jinwook Oh, Ashish Ranjan, Zhibin Ren, Scot Rider, Kerstin Schelm, Michael Scheuermann, Joel Silberman, Jie Yang, Vidhi Zalani, Xin Zhang, Ching Zhou, Matt Ziegler, Vinay Shah, Moriyoshi Ohara, Pong-Fei Lu, Brian Curran, Sunil Shukla, Leland Chang, and Kailash Gopalakrishnan. 2021. RaPiD: AI Accelerator for Ultra-low Precision Training and Inference. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 153–166. https://doi.org/10.1109/ISCA52012.2021.00021