Stream-Driven Acceleration for Embedded RISC-V SoCs

João Maia, Ana Silveira, Gonçalo Midões, Nuno Neves, Pedro Tomás, Nuno Roma
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
E-mail: {joaofbmaia, ana.silveira, goncalomidoes}@tecnico.ulisboa.pt, {nuno.neves, pedro.tomas, nuno.roma}@inesc-id.pt

***Abstract*—This paper proposes a stream-driven computational model that expands the recent stream vectorization paradigm into a full dataflow-driven computing model. It exploits spatial computation and time-multiplexing, while relying on streaming engines implementing the RISC-V UVE specification to manage data access patterns, thus streamlining memory operations and reducing latency. By abstracting the kernel loops into stream data-flow graphs and mapping them onto a processing element array, the conceived accelerator architecture exploits both spatial and temporal parallelism across a wide range of computational tasks. Experimental results, conducted on a synthesized 7nm implementation, demonstrate the proposed model's potential to develop high-efficiency accelerators in data-intensive applications, offering performance gains of up to 6× compared with an ARM Cortex-A53 CPU with NEON and 15× compared with a scalar Rocket RISC-V CPU, along with 3.86× energy efficiency improvements.**

***Index Terms*—Data streaming, Streaming engine, Dataflow, PE Array, RISC-V, Accelerator**

I. Introduction

One critical inefficiency that is commonly observed in modern computing arises from the performance gap between processing units and memory systems, often referred to as the memory wall [1, 2]. While caching and prefetching [3–6] are widely deployed across general and special-purpose processors to overcome this problem, their performance gains have plateaued [7]. On the other hand, more radical approaches, such as near and in-memory computing, are still struggling to gain traction in general-purpose computing domains [8].

At the same time, the data-streaming computing paradigm has observed a renewed interest [9–12], especially due to its decoupled access-execute computing model that allows for specialized hardware optimizations. As a result, new streaming engines, which autonomously generate the memory addresses and manage the data transfers, have been integrated into several general-purpose processor (GPP) architectures to reduce load-to-use latency. While some solutions have leveraged this approach to accelerate computation through stream vectorization (e.g., RISC-V Unlimited Vector Extension (UVE) [11]), more advanced structures are still required to fully exploit the benefits offered by stream-based computing.

The presented work expands the GPP stream vectorization model to a full dataflow-driven computing model, exploiting both spatial computation and time-multiplexing, and allowing the deployment of an adaptable stream-based co-acceleration structure [13–18]. Although such techniques have been previously considered to cope with the increasing demand for high-performance computing structures in domains such as signal processing, computer vision, artificial intelligence, and cryptography [19–22], they have been mostly deployed in highly specialized Domain-Specific Accelerators (DSAs), lacking general-purpose computing capabilities. On the contrary, this paper demonstrates the viability of this paradigm in the design of a common tensor-like structure, often used in AI/ML workloads [21, 22], by introducing general-purpose Processing Elements (PEs) alongside reconfigurable spatial and temporal multiplexing. The proposed system integrates an adaptable 2D PE Array (*PE-Array*) powered by a comprehensive data-streaming mechanism, deployed in a host RISC-V GPP within a full System-on-Chip (SoC) infrastructure (see Fig. 1.E). In summary, this paper presents the following contributions:

- A new **Stream-driven Computational Model** that expands the stream vectorization model from UVE [11] with a specialized Data-Flow Graph (DFG) representation (*stream-DFG*), allowing the parallelization and mapping of an arbitrarily kernel loop to a 2D acceleration structure.
- A dedicated UVE-compliant [11] **Streaming Engine (SE)** integrated into the host GPP to autonomously handle: 1) the address generation (based on memory access pattern descriptors obtained at compile-time); 2) the subsequent data fetching; and 3) the assembling of data into stream vectors (mappable to the accelerator);
- A new parameterizable **Stream-based Accelerator Architecture**, capable of spatially and temporally distributing general computational tasks. When integrated within a complete RISC-V SoC, supported by the UVE-compliant [11] SE, the proposed accelerator deploys a fully decoupled access-execute scheme, allowing memory operations to proceed in parallel with computation, reducing memory-access-induced overheads such as load-to-use latency and improving data throughput.

II. Stream-driven Computational Model

The proposed architecture deploys a decoupled stream-based computational model, which expands from the UVE [11] stream vectorization model to support 2D acceleration structures. It abstracts each kernel loop into two separate parts:

Work supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under project 2022.06780.PTDC (DOI: 10.54499/2022.06780.PTDC). We also acknowledge the contributions from project UIDB/50021/2020 (DOI: 10.54499/UIDB/50021/2020).

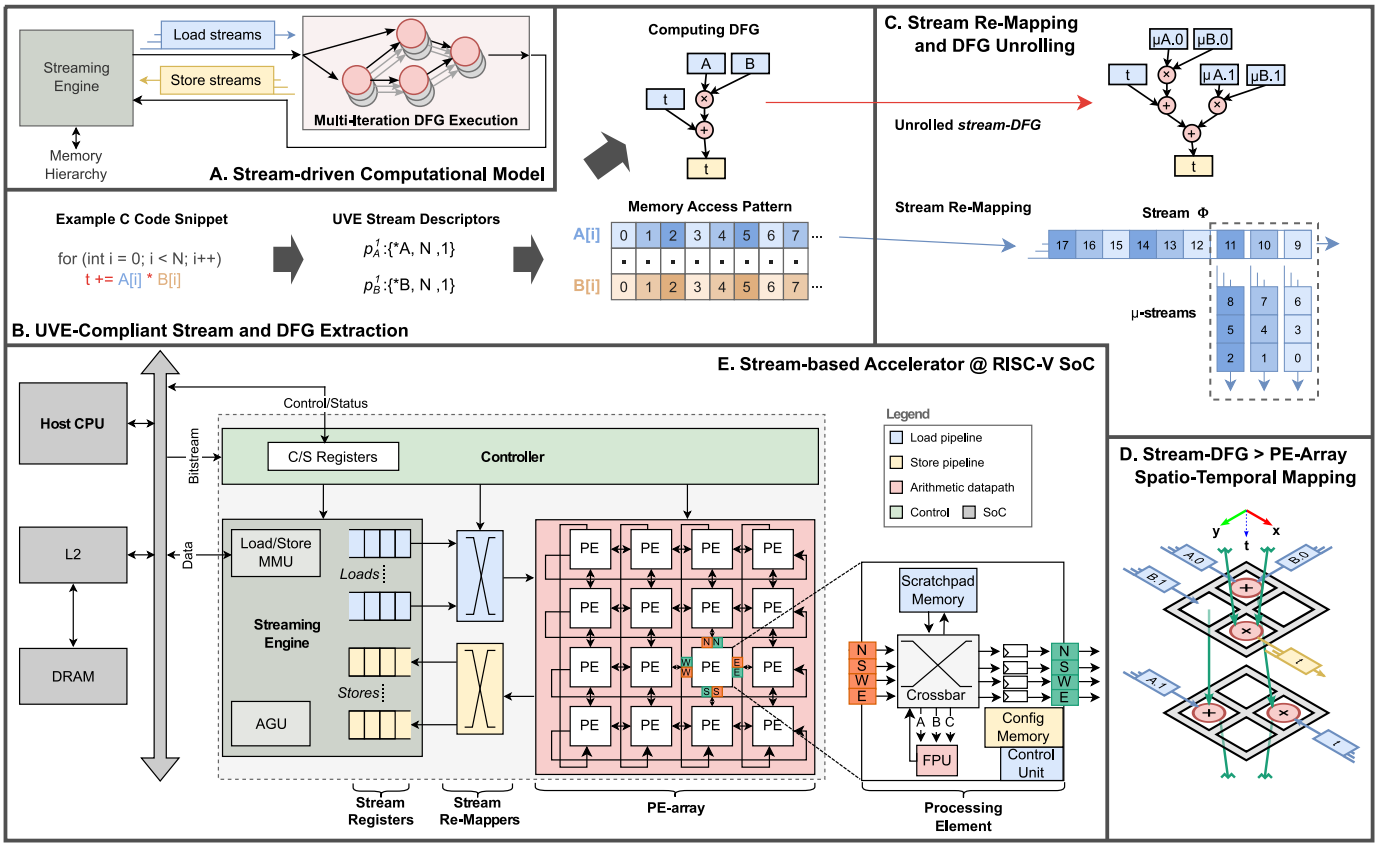

Fig. 1. Proposed system overview: A) Stream-driven computational abstraction model; B) Example dot-product UVE-compliant [11] stream (S_ϕ) and DFG representations; C) DFG unrolling (2×) into the stream-DFG (top) and corresponding S_ϕ mapping to μ-streams (bottom); D) Example of a spatio-temporal mapping of the stream-DFG from C) into the PE-Array in E); E) Overall hardware architecture (with PE zoom-in).

i) **data streaming**, encompassing the decoding of the memory access patterns to be used by the SE to automatically handle the data transfers; and *ii)* **dataflow computing**, including the representation of the computation kernel with a dedicated stream-DFG and its deployment on the PE-Array. Figs. 1.A-C depict the proposed model expansion and its components.

A. Data Streaming Scheme

The data streaming mechanism is centered on the SE, which manages all load/store operations for the accelerator. It receives a set of UVE-like [11] descriptors (or streams) $(S_1, \cdots, S_K)$ from the host GPP, where each stream S_ϕ represents an access pattern to a distinct data structure. Each stream is defined as a chain of 1-D memory address patterns (p^i_ϕ) (see Fig. 1.B). Thus, each stream identifier ϕ characterizes a multi-dimensional, regular, or indirect memory access pattern over a unique set of addresses, with the UVE [11] specification.

At the SE, the stream descriptors are processed by an Address Generation Unit (AGU), which generates an address sequence and forwards it to the Load/Store Memory Management Unit (MMU) (see Fig. 1.E). The MMU handles the corresponding requests, reordering them (as needed) to manage potential out-of-order memory responses. Once reordered, load requests are buffered in corresponding ϕ *Stream Registers*, implemented as First-In, First-Out (FIFO) queues. Store streams sent by the PE-Array are similarly queued in the appropriate Stream Registers and issued to memory.

Configurable *Stream Re-Mappers* connect the stream registers to the PE-Array input/output buffers. These structures decompose each stream S_ϕ into a set of μ-streams, each matching the input sequence for an input/output PE (see Fig. 1.C). Hence, they effectively scatter stream data into PE array inputs and gather PE array outputs into store streams.

With such an approach, the SE abstracts all memory accesses from the PE-Array, providing efficient data prefetching that mitigates memory access latency and linearizes the memory access sequences. As a result, from the perspective of the PEs, the memory access pattern appears as a simple coalesced sequence of elements, simplifying the computational model.

B. Stream-driven Dataflow Computing

The computation is abstracted as an acyclic DFG, where each node represents an operation and each edge denotes a data dependency (see Figs. 1.B-C). Source nodes represent the input (load) data streams, and sink nodes represent the output (store) streams. Cyclic dependencies are implicitly represented by load streams that are subsequently read by store streams.

Although finding efficient DFG mappings to 2D structures can be a highly complex problem [23, 24], the linearization of the memory access pattern attained by the SE, together with the adopted dataflow computing model, greatly simplifies the

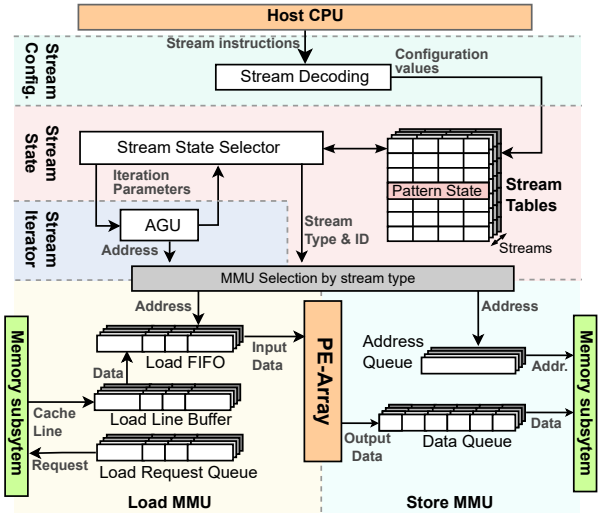

Fig. 2. Streaming engine architecture overview.

process. Furthermore, to maximize the throughput, multiple iterations of the loop are executed in parallel, by considering an unrolled DFG scheme, *stream-DFG* (see Fig. 1.A and Fig. 1.C-top). They are scheduled by assigning each node of the stream-DFG to a specific PE at a specific execution cycle while respecting data dependencies. Routing between DFG nodes is achieved by mapping the communication of resulting values to the wires (spatial connections) or by using registers to implement delays (temporal connections). Mapping and scheduling are then optimized by minimizing the number of cycles between consecutive iterations (see Fig. 1.D).

III. System Architecture

The proposed accelerator architecture is integrated on a SoC infrastructure, managed by a RISC-V host GPP, and integrating a cache-based memory subsystem (see Figure 1.E). It comprises three modules: *i*) SE and Stream Re-Mapper, *ii*) PE-Array, and *iii*) a dedicated controller to interface with the host GPP and manage the execution of stream-DFGs.

A. Streaming Engine and Re-Mapper

The SE comprises five modules: Stream Configurator; Stream State; Stream Iterator; and two MMUs (Load and Store) (see Fig. 2). The stream life cycle begins with the decoding of the stream configuration instructions and filling the Stream Tables, which hold the status of all active streams.

The Stream State Selector arbitrates the access to the single AGU (saving HW resources), responsible for iterating over the stream. At each cycle, the Stream State Selector picks the parameters of a given stream from the Stream Tables and sends them to the AGU. With these parameters, the AGU generates a new address, passing it to the Load MMU. Finally, the current stream state is recorded back in the Stream Tables, except if the same stream is again selected for iteration.

The Load MMU manages the data loads through its sub-modules: the Load FIFO, the Load Line Buffer, and the Load Request Queue. When a new address is generated, it is tagged and saved on a Requests Queue entry, coalescing with a previous cache line request (whenever possible - to minimize memory bandwidth) or creating a new entry (otherwise). At the same time, load requests are recorded in the Load FIFO, along with the request ID. This way, when a cache line is received from memory, it is temporarily stored in a Line Buffer (acting as a small fully associative L0 cache, to avoid duplicating cache line requests), and its contents are processed to answer all pending Load FIFO entries that match the request ID.

The Store MMU works similarly: generated store requests are saved on the address queue, where they wait for the incoming data from the PE-Array, and are also aligned with cache lines to minimize store bandwidth to memory.

Finally, the Stream Re-Mappers act like crossbars between the elements of the stream registers and the array I/O. However, since elements do not require multicasting, these modules are implemented using permutation networks instead of full crossbars, resulting in substantial area savings.

B. PE-Array Architecture

The PE-Array consists of a highly flexible and parameterizable systolic 2-D grid of interconnected PEs, allowing for easy scaling of its characteristics according to the requisites of different use cases. Each PE (see Fig. 1.E-right) is designed for efficient data processing and can execute several operations depending on its configuration. It has four cardinal inputs and outputs (North, South, East, West), and contains the following components: Configuration Memory, Data Memory, Control Unit, and a Floating-Point Unit (FPU). The Control Unit stores the PE configuration that defines the sequence of operations, implements hardware loops, manages data forwarding to/from other PEs, and oversees the FPU execution. The FPU itself was adapted from [25] and supports 16 single-precision floating point operations, including fused multiply-accumulate (FMA).

The inner PEs are connected bidirectionally to their neighbors, allowing data transfers in all four cardinal directions. In addition, the interconnect allows the definition of ring structures across rows and columns (e.g., the North output of each PE in the first row is connected to the South input of the PE in the last row, forming a ring structure along each column). Also, the outer PEs, located in the first column and first row, receive inputs from the Stream Re-Mapper through the West and North inputs, respectively, while the outer PEs in the last column and last row handle the output of data to the outside of the array, through their East and South outputs.

IV. Experimental Evaluation

The functional validation of the proposed accelerator was conducted by integrating it into a Chipyard's Rocketchip SoC [26, 27] including a Rocket Core RISC-V CPU, a 4-bank 8-way set-associative L2 cache totaling 512 KiB, quad-channel DRAM, and a 256-bit system bus, ensuring data bandwidth suited for high-throughput data transfer. The accelerator's throughput was evaluated for a 4×4 PE-array, with a SE configuration of 32-entry Load Request Queue and a 4-row Load Line Buffer. The 4×4 PE-array was chosen because it strikes a good balance by providing enough processing elements to exploit the parallelism of the workloads while keeping the mapping time reasonable. The specific SE parameters were chosen based on a design space exploration

as a balanced configuration that optimizes performance while managing area and complexity.

A. Performance Evaluation

The proposed architecture was validated with RTL simulations using Synopsys VCS 2022.06, by considering a cycle-accurate DRAM model provided by DRAMSim2 [28].

A diverse set of benchmarks (GEMM, Blackscholes, Jacobi-1D, Jacobi-2D, Heat-3D, FIR) was selected to evaluate different accelerator properties, such as memory access pattern and usage of the PE-array. DFG extraction, unrolling, mapping, and bitstream generation were performed with a custom modified Morpher [29] tool.

Fig. 3 presents the obtained speedups when the proposed accelerating structure was compared with a scalar Rocket Core RISC-V CPU and a vector Arm Cortex-A53 NEON CPU. The obtained results, which consider a strict clock cycle comparison to normalize for different operating frequencies, denote a clear advantage of the proposed structure, providing performance gains as high as 6x (when compared with the Arm A53 CPU with NEON) and 15x (when compared with the scalar Rocket RISC-V CPU). Further work is being conducted to improve the throughput of the SE, which is currently limited by its address generation rate.

B. Hardware Resource Analysis

The accelerator setup was synthesized using Cadence Genus 21.15, targeting the 7nm ASAP7 [30] technology process. The hardware resources of the accelerator are presented in Table I. This table also presents the silicon area occupied by the Rocket CPU, when implemented with the same technology process using the available RTL description [26, 27].

As it can be observed, most of the area footprint is related to the SE, particularly due to the buffers used to hide memory access latency and handle the out-of-order nature of the memory responses to the accelerator. Naturally, the second largest element is related to the PE-array, which occupies $\approx 22\%$ of the accelerator area. Nevertheless, the overall silicon area ($0.122\ mm^2$) is considerably lower than the area required by the Rocket CPU + L2 Cache ($0.265\ mm^2$).

C. Power Consumption and Energy Efficiency

When considering the power consumption of the proposed accelerator and the Rocket RISC-V GPP implemented in the same 7nm technology depicted in Table I, it is highlighted the low power consumption overhead imposed by the proposed accelerator when integrated into the SoC. Both the Rocket core and the proposed accelerator were synthesized targeting a clock frequency of 500 MHz; this frequency was used consistently in the energy analysis. This conclusion is further emphasized when the two structures are compared in terms of energy consumption (see Fig.4). As it can be observed, the proposed acceleration structure provides energy gains as high as 3.86× when compared with the Rocket RISC-V GPP, as a direct result of the combination of efficient data streaming with an adaptable acceleration structure. Note that these power figures reflect only static power consumption. Additionally, while Fig. 4 also includes a comparison with an ARM Cortex-A53 (since performance was also compared with the ARM A53), some of the energy gains observed for the ARM CPU are attributable to technology differences, as the ARM core (clocked at 1.2 GHz) is implemented in 28nm technology.

TABLE I
HARDWARE RESOURCES BREAKDOWN.

	Component	Area (mm^2)	Power (mW)
Accelerator	PE-Array (4×4)	0.027	76.7
	SE	0.079	115.4
	ReMapper	0.004	4.4
	Controller	0.006	2.1
	TOTAL	0.122	192.6
SoC	Rocket Tile	0.033	22.3
	L2 Cache (512 KiB)	0.232	24.2
TOTAL		0.433	259.3

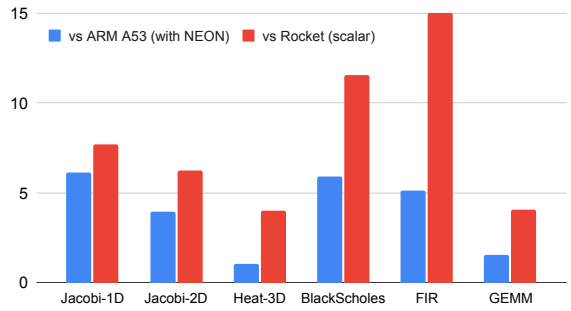

Fig. 3. Clock cycle improvement provided by the proposed accelerator.

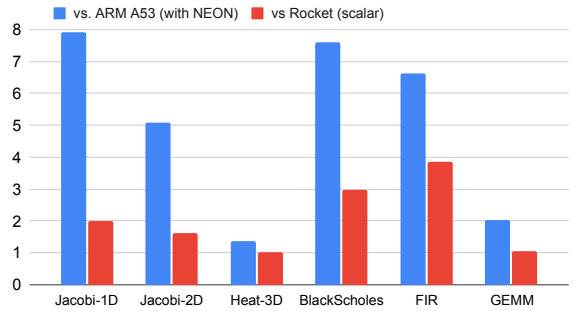

Fig. 4. Energy improvement provided by the proposed accelerator.

V. Conclusion

This paper presents a stream-driven computational model that extends recent stream vectorization techniques into a fully dataflow-driven architecture. By utilizing a UVE-compliant [11] SE to autonomously manage data accesses, the model minimizes memory latency and load-to-use delays, enabling high efficiency in data-intensive applications. The architecture's design, which maps kernel loops as DFGs onto a versatile 2-D PE array, effectively exploits both spatial and temporal parallelism across general-purpose RISC-V systems. Experimental results from a synthesized 7nm implementation in RISC-V SoC show significant performance and energy efficiency gains compared to conventional GPP architectures, validating the stream-driven model capabilities and adaptability for high-demanding application domains.

REFERENCES

[1] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[2] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *Proceedings of the 37th annual international symposium on Computer architecture*, 2010, pp. 37–47.

[3] M. Shevgoor, S. Koladiya, R. Balasubramonian, C. Wilkerson, S. H. Pugsley, and Z. Chishti, "Efficiently prefetching complex address patterns," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 141–152.

[4] I. Hadade, T. M. Jones, F. Wang, and L. d. Mare, "Software prefetching for unstructured mesh applications," *ACM Transactions on Parallel Computing (TOPC)*, vol. 7, no. 1, pp. 1–23, 2020.

[5] S. Mostofi, H. Falahati, N. Mahani, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Snake: A variable-length chain-based prefetching for GPUs," in *56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 728–741.

[6] Y. Zhang, N. Sobotka, S. Park, S. Jamilan, T. A. Khan, B. Kasikci, G. A. Pokam, H. Litz, and J. Devietti, "Rpg2: Robust profile-guided runtime prefetch generation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, vol. 2, 2024, pp. 999–1013.

[7] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture." *Comms. of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[8] J. Dean *et al.*, "A new golden age in computer architecture: Empowering the machine-learning revolution," *IEEE Micro*, vol. 38, no. 2, pp. 21–29, 2018.

[9] Z. Wang and T. Nowatzki, "Stream-based Memory Access Specialization for General Purpose Processors," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019, pp. 736–749.

[10] F. Schuiki, F. Zaruba, T. Hoefler, and L. Benini, "Stream Semantic Registers: A Lightweight RISC-V ISA Extension Achieving Full Compute Utilization in Single-Issue Cores," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 212–227, 2021.

[11] J. M. Domingos, N. Neves, N. Roma, and P. Tomás, "Unlimited Vector Extension with Data Streaming Support," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 209–222.

[12] P. Scheffler, F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, "Sparse stream semantic registers: A lightweight isa extension accelerating general sparse linear algebra," *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[13] S. Chen, C. Cai, S. Zheng, J. Li, G. Zhu, J. Li, Y. Yan, Y. Dai, W. Yin, and L. Wang, "HierCGRA: A Novel Framework for Large-scale CGRA with Hierarchical Modeling and Automated Design Space Exploration," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 17, no. 2, pp. 1–31, 2024.

[14] D. Liu, Y. Xia, J. Shang, J. Zhong, P. Ouyang, and S. Yin, "E2EMap: End-to-End Reinforcement Learning for CGRA Compilation via Reverse Mapping," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 46–60.

[15] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[16] R. Prabhakar, Y. Zhang, D. Koeplinger, M. Feldman, T. Zhao, S. Hadjis, A. Pedram, C. Kozyrakis, and K. Olukotun, "Plasticine: A reconfigurable architecture for parallel paterns," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 389–402, 2017.

[17] T. Nowatzki, V. Gangadhar, N. Ardalani, and K. Sankaralingam, "Stream-dataflow acceleration," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 416–429.

[18] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[19] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Communications of the ACM*, vol. 63, no. 7, pp. 48–57, 2020.

[20] B. Boroujerdian, Y. Jing, D. Tripathy, A. Kumar, L. Subramanian, L. Yen, V. Lee, V. Venkatesan, A. Jindal, R. Shearer *et al.*, "7farsi: An early-stage design space exploration framework to tame the domain-specific system-on-chip complexity," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 2, pp. 1–35, 2023.

[21] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[22] NVIDIA, "NVIDIA Deep Learning Accelerator," Tech. Rep., 2018. [Online]. Available: http://nvdla.org/primer.html%7D

[23] S. Das, K. Martin, T. Peyret, and P. Coussy, "An Efficient and Flexible Stochastic CGRA Mapping Approach," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 1, pp. 8:1–8:24, Oct. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3550071

[24] B. R. Rau, "Iterative Modulo Scheduling," *International Journal of Parallel Programming*, vol. 24, no. 1, pp. 3–64, Feb. 1996. [Online]. Available: https://doi.org/10.1007/BF03356742

[25] L. Crespo, P. Tomás, N. Roma, and N. Neves, "Unified Posit/IEEE-754 Vector MAC Unit for Transprecision Computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 5, pp. 2478–2482, May 2022, conference Name: IEEE Transactions on Circuits and Systems II: Express Briefs. [Online]. Available: https://ieeexplore.ieee.org/document/9737228/?arnumber=9737228

[26] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[27] Berkeley Architecture Research, *Chipyard Documentation, Release 1.8.1*, Oct. 2022. [Online]. Available: https://chipyard.readthedocs.io/_/downloads/en/1.8.1/pdf/

[28] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, 2011.

[29] D. Wijerathne, Z. Li, M. Karunaratne, L.-S. Peh, and T. Mitra, "Morpher: An open-source integrated compilation and simulation framework for cgra," in *Fifth Workshop on Open-Source EDA Technology (WOSET)*, 2022.

[30] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.