

A Cluster-Based Distributed Memory Architecture for CGRAs

Shangkun Li[†], Cheng Tan[‡], Zeyu Li[†], Jinming Ge[†], Jiawei Liang[†], Hao Yang[§], Linfeng Du[†], Jiang Xu[¶], Wei Zhang[†]

[†]The Hong Kong University of Science and Technology [‡]Google

[§]The George Washington University [¶]The Hong Kong University of Science and Technology (Guangzhou)

shangkun.li@connect.ust.hk, wei.zhang@ust.hk

Abstract—Coarse-Grained Reconfigurable Arrays (CGRAs) are a promising solution for achieving high energy efficiency and reconfigurability across various application domains. However, their practical performance is frequently impeded by centralized memory architectures. Conventional CGRA architectures restrict direct memory access to specific tile locations, forcing extensive data routing that competes with computation resources. To mitigate this routing overhead, this work introduces a cluster-based distributed memory architecture. By partitioning the array into clusters that share localized memory units and utilizing a lightweight coherence mechanism, the proposed design enhances data accessibility. Evaluation demonstrates that this distributed memory architecture for CGRAs yields an average speedup of $1.34\times$ compared to traditional global-memory baseline CGRAs.

Index Terms—CGRAs, Memory Architecture, Reconfigurable Computing

I. INTRODUCTION

Coarse-Grained Reconfigurable Arrays (CGRAs) offer a compelling balance between high flexibility and high energy efficiency [1], making them increasingly popular for accelerating applications in domains like high-performance computing (HPC) [2] and machine learning (ML) [3]–[5]. Architecturally, typical CGRAs are composed of an array of computing tiles linked via reconfigurable on-chip networks. To accelerate kernels, the compiler transforms loop kernels into Data Flow Graphs (DFGs) [6]–[8] and maps them onto the Modulo Routing Resource Graph (MRRG) [9] of the CGRA. The primary objective of this mapping is to minimize the initial interval (II) — the number of cycles between the start of successive loop iterations — thereby maximizing the overall throughput.

Despite the computational prowess of CGRAs, the efficacy of conventional CGRAs is frequently compromised by data provisioning constraints inherent in centralized memory designs. Contemporary CGRA architectures typically couple the tile array with a monolithic global memory, employing one of two restrictive memory access methods. The **tile-based method** [10], [11] confines memory interfaces to specific boundary tiles, necessitating multi-hop data relaying to reach inner computing tiles. This excessive data routing consumes valuable interconnect resources and significantly degrades performance. Alternatively, the **HW/SW arbitration-based method** [12]–[14] either uses a hardware arbiter to select one tile per row or column to access the memory each cycle, or relies on the compiler to

statically schedule non-conflicting memory accesses. While this method attempts to broaden access, but introduces serialization latencies when multiple tiles within a row or column contend for limited bandwidth. In scenarios of high contention, these memory access conflicts force memory request deferrals and complex rerouting. Consequently, both memory access methods fail to deliver the parallel, low-latency data access required by many workloads, necessitating a paradigm shift toward distributed memory hierarchies to resolve these fundamental bottlenecks.

II. PROPOSED METHODOLOGY

Fig. 1 gives an overview of our proposed novel CGRA architecture. The fundamental innovation of the design lies in its memory subsystem organization, which departs from monolithic shared memory in favor of a distributed, cluster-based approach. As shown in Fig. 1(a), tiles are grouped into clusters of four tiles, with each cluster sharing a local memory unit. A memory coherence controller maintains the coherence across these distributed memory units. Each tile contains function units that support LLVM [15] IR operations, control memory, a 6×12 crossbar, eight register sets, and four bypass buffers (Fig. 1(c)). The tiles are interconnected in a mesh-style topology [16].

A. Cluster-Based Distributed Memory Architecture

To mitigate the interconnect congestion inherent in centralized memory architectures, we restructure the tile array into independent clusters, each equipped with a dedicated, multi-banked local memory unit. The number of banks in each local memory unit matches the cluster size to enable simultaneous memory access from all tiles within the cluster. Within a cluster, tiles are tightly coupled to this local memory, enabling simultaneous parallel memory access without contending for global routing resources. This design choice effectively localizes the majority of memory traffic, drastically reducing the demand on the inter-tile routing resources. Our evaluation explores cluster sizes of $\{1, 2, 4, 6\}$ for a 6×6 CGRA. Other cluster sizes can also be supported based on user configurations.

B. Lightweight Memory Coherence Controller

Since a variable may be needed by tiles in multiple clusters, our proposed architecture permits variable replication across multiple memory units. To maintain data integrity amidst this

Corresponding author: Wei Zhang (wei.zhang@ust.hk).

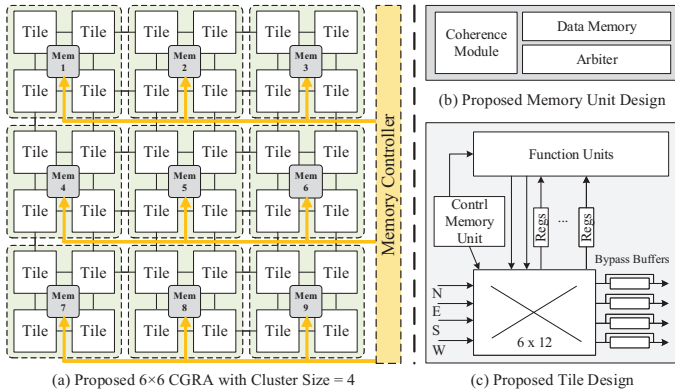


Fig. 1. Proposed Architecture Overview. (a) The 6×6 CGRA is partitioned into clusters (i.e., cluster size of 4), where tiles within each cluster share a local memory unit managed by a coherence controller. (b) The memory unit integrates a multi-bank data memory, a hardware arbiter for local access, and a coherence module to track data modifications. (c) Tile micro-architecture features standard function units and control memory.

replication, we introduce a centralized hardware controller that enforces a MESI-like coherence protocol [17]. This controller manages a global state table tracking the residency and status (*modified*, *exclusive*, *shared*, and *invalid*) of all active variables. The coherence maintenance is executed through a streamlined hardware pipeline. Upon receiving a write notification from the coherence module in a cluster’s memory unit, the controller immediately updates the global state to reflect the modification and broadcasts invalidation signals to all other clusters holding copies of that variable. Subsequently, it generates synchronization requests to update the stale copies with the new value. This hardware-automated synchronization, operating in tandem with compiler-enforced access schedules, ensures that the distributed memory view remains consistent across the entire array with minimal latency overhead.

Crucially, to prevent the coherence mechanism from becoming a performance bottleneck, we optimized the controller logic into a deterministic two-cycle pipeline. The first cycle handles state lookup and invalidation broadcasting, while the second cycle resolves data synchronization. This low-latency design ensures that memory consistency is maintained without degrading the operating frequency of the compute tiles.

III. EXPERIMENTAL RESULTS

We evaluate the proposed architecture using an integrated framework comprising an LLVM-based compiler and a cycle-accurate simulator. To obtain precise physical metrics, we implemented the design in RTL [8], [11] and synthesized a 6×6 prototype using the TSMC 22nm ULL library with Synopsys Design Compiler and CACTI 7.0 [18] (for SRAM macros). The evaluation employs a diverse benchmark suite collected from [8], [19]–[22], including linear algebra kernels and complex applications from diverse domains. Performance is compared against a baseline conventional 6×6 mesh CGRA configured with a 64 KB global SRAM (a minimum size sufficient to execute all benchmarks), utilizing an identical mapper without cluster constraints to ensure fairness.

We begin by aligning the total memory capacity of a proposed 6×6 CGRA (cluster size 4) with the baseline’s 64 KB.

TABLE I
BENCHMARKS AND EVALUATION RESULTS

Domains	Kernels/ Applications	EDFG Characteristics ^a	Best ^b Perf/Area
Linear Algebra	gemm	33, 4, 3, 21168	$1.56 \times / 1.20 \times$
	2mm	52, 6, 5, 35280	$1.42 \times / 1.20 \times$
	trVecAccum ^c	18, 1, 1, 2048	$1.40 \times / 1.20 \times$
	gemver	168, 28, 5, 6144	$1.39 \times / 1.06 \times$
	reduce-sum	73, 8, 2, 4352	$1.30 \times / 1.20 \times$
Signal Process	fft	96, 24, 6, 20480	$1.26 \times / 1.20 \times$
Communication	viterbi	97, 8, 4, 940	$1.49 \times / 1.20 \times$
Graph	floyd	20, 4, 1, 7056	$1.21 \times / 1.20 \times$
Optimization	levmarq	52, 4, 2, 84	$1.40 \times / 1.20 \times$
ML	gcn	106, 19, 11, 55720	$1.25 \times / 1.20 \times$

a. Characteristics contain {#OpNds, #VarNds, #Vars, Total Size of Vars in bytes}.
b. **Best** refers to the cluster size from {1, 2, 4, 6} that yields the highest speedup for that kernel. All Speedup and Area values are normalized to the **baseline** CGRA’s speedup and area (with SRAM).
c. trVecAccum transforms a triangular matrix.

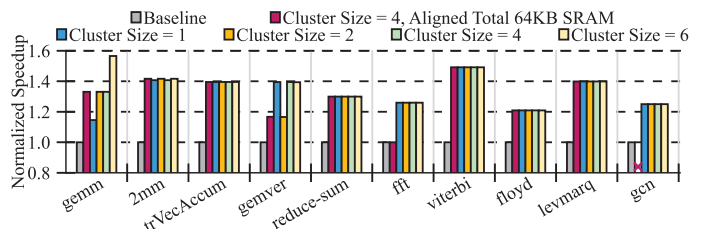


Fig. 2. Performance evaluation. Normalized speedup of the proposed cluster-based architecture with varying cluster sizes ($\{1, 2, 4, 6\}$) compared to the conventional baseline. The “Cluster Size = 4, Aligned Total 64KB SRAM” bar indicates the performance of an initial configuration constrained to match the baseline’s total memory capacity.

However, this configuration is insufficient for the gcn kernel. Consequently, to enable a comprehensive evaluation across the full benchmark suite, we adopt a 16KB memory unit per cluster (the minimum size dictated by the most demanding kernel gcn). Fig. 2 presents the performance results across 6×6 CGRAs with cluster sizes of $\{1, 2, 4, 6\}$. The proposed architecture consistently outperforms the baseline, achieving average speedups of $\{1.33 \times, 1.32 \times, 1.34 \times, 1.37 \times\}$, respectively, with an overall average of $1.34 \times$. These gains stem from the architecture’s ability to localize memory traffic, effectively alleviating global routing pressure. Regarding physical implementation, the area of the synthesized 6×6 prototype CGRA with cluster size of 4 is 0.38mm^2 with SRAM macros, operating at $800 \text{MHz} @ 22 \text{nm}$. The memory controller accounts for a negligible 0.50% of the total area. As shown in Table I, our performance per area consistently surpasses the baseline, yielding an average improvement of 16% .

IV. CONCLUSION

This work addresses the critical memory bottleneck in conventional CGRAs caused by centralized storage and rigid access topologies. We propose a cluster-based distributed memory architecture that partitions the computing array into local clusters, each served by a dedicated memory unit and managed by a lightweight coherence controller. By localizing memory traffic and enabling parallel access, this design effectively mitigates global routing congestion. Experimental results demonstrate that the proposed architecture achieves an average speedup of $1.34 \times$ and a 16% improvement in performance-per-area compared to a traditional baseline, confirming its efficacy in delivering efficient data provisioning for diverse workloads.

REFERENCES

- [1] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin, and S. Wei, "A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications," *ACM Comput. Surv.*, vol. 52, no. 6, Oct. 2019.
- [2] J. Anderson, B. Adhi, C. Cortes, E. D. Sozzo, O. Ragheb, and K. Sano, "Exploration of compute vs. interconnect tradeoffs in cgras for hpc," in *Proceedings of the 13th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, ser. HEART '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 59–68.
- [3] R. Prabhakar, "Sambanova sn401 rdu: Breaking the barrier of trillion+ parameter scale gen ai computing," in *2024 IEEE Hot Chips 36 Symposium (HCS)*, Aug 2024, pp. 1–24.
- [4] K. Koul, M. Strange, J. Melchert, A. Carsello, Y. Mei, O. Hsu, T. Kong, P.-H. Chen, H. Ke, K. Zhang, Q. Liu, G. Nyengele, A. Balasingam, J. Adivarahan, R. Sharma, Z. Xie, C. Tornig, J. Emer, F. Kjolstad, M. Horowitz, and P. Raina, "Onyx: A programmable accelerator for sparse tensor algebra," in *2024 IEEE Hot Chips 36 Symposium (HCS)*, 2024, pp. 1–91.
- [5] J. Qin, T. Xia, C. Tan, J. Zhang, and S. Q. Zhang, "Picachu: Plug-in cgra handling upcoming nonlinear operations in llms," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 845–861.
- [6] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling," in *2003 Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp. 296–301.
- [7] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Hara-Azumi, and J. Anderson, "Cgra-me: A unified framework for cgra modelling and exploration," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2017, pp. 184–189.
- [8] C. Tan, C. Xie, A. Li, K. J. Barker, and A. Tumeo, "Opencgra: An open-source unified framework for modeling, testing, and evaluating cgras," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, 2020, pp. 381–388.
- [9] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Dresc: a retargetable compiler for coarse-grained reconfigurable architectures," in *2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings.*, 2002, pp. 166–173.
- [10] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.
- [11] C. Tan, D. Patil, A. Tumeo, G. Weisz, S. Reinhardt, and J. Zhang, "Vecpac: A vectorizable and precision-aware cgra," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9.
- [12] L. Huang and D. Liu, "Optimizing data reuse for cgra mapping using polyhedral-based loop transformations," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.
- [13] C. Tan, N. B. Agostini, T. Geng, C. Xie, J. Li, A. Li, K. J. Barker, and A. Tumeo, "Drips: Dynamic rebalancing of pipelined streaming applications on cgras," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 304–316.
- [14] Z. Li, D. Wu, D. Wijerathne, and T. Mitra, "Lisa: Graph neural network based portable mapping on spatial accelerators," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 444–459.
- [15] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*, ser. CGO '04. USA: IEEE Computer Society, 2004, p. 75.
- [16] H. Park, Y. Park, and S. Mahlke, "Polymorphic pipeline array: A flexible multicore accelerator with virtualized execution for mobile multimedia applications," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 370–380.
- [17] M. S. Papamarcos and J. H. Patel, "A low-overhead coherence solution for multiprocessors with private cache memories," *SIGARCH Comput. Archit. News*, vol. 12, no. 3, p. 348–354, Jan. 1984.
- [18] S. Thoziyoor, N. Muralimanohar, J. Ahn, and N. Jouppi, "Cacti 6.5," *hpl. hp. com*, 2009.
- [19] L.-N. Pouchet *et al.*, "Polybench: The polyhedral benchmark suite," *URL: <http://www.cs.ucla.edu/pouchet/software/polybench>*, vol. 437, pp. 1–1, 2012.
- [20] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "MachSuite: Benchmarks for accelerator design and customized architectures," in *Proceedings of the IEEE International Symposium on Workload Characterization*, Raleigh, North Carolina, October 2014.
- [21] J. Cheng, L. Josipović, G. A. Constantinides, P. Ienne, and J. Wickerson, "Dass: Combining dynamic static scheduling in high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 3, pp. 628–641, 2022.
- [22] J. Wang, L. Guo, and J. Cong, "Autosa: A polyhedral compiler for high-performance systolic arrays on fpga," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 93–104.