Unleashing CGRA Potential for HPC

Boma Adhi[†], Emanuele Del Sozzo[†], Carlos Cortes[†], Xinyuan Wang[§], Tomohiro Ueno[†], Kentaro Sano[†]

[†] *Center for Computational Science (R-CCS), RIKEN*, Japan
{boma.adhi, emanuele.delsozzo, carlos.cortestorres, tomohiro.ueno, kentaro.sano}@riken.jp

[§] *University of Toronto*, Canada
xinyuan.wang@mail.utoronto.ca

***Abstract*—This poster highlights our previous and future design-space exploration effort to optimize our Coarse-Grained Reconfigurable Array (CGRA) architecture for HPC, i.e., intra-CGRA interconnect optimization, FMA and transcendental operation on CGRA, programmable buffer, systolic-array style execution on CGRA, predication support, and FPGA based emulation on actual HPC environment.**

***Index Terms*—CGRA, HPC, Architecture, Dataflow, Systolic Array**

I. Introduction

A CGRA is a reconfigurable computing device traditionally used for accelerators in low-powered embedded devices, akin to FPGA but trades the bit-level programmability with the coarser word-level programmability, offering ASIC-like performance and efficiency. RIKEN CGRA [1] is our CGRA architecture targeted at future HPC systems with OpenMP support [2], and the reminder of this extended abstract summarizes our effort to explore necessary architectural optimizations for future HPC systems.

RIKEN CGRA is developed around these design philosophies: modular and scalable design for high-throughput computation, separation between compute and memory processing tiles, support for (often ignored) floating-point operations. The CGRA mainly comprises of 3 types of tiles, i.e., Processing Element (PE) tiles for computing, Load-Store (LS) tiles for memory access, and Switch Block (SB) tiles for routing, as shown in Figure 1. New type of tiles may be added for future extension. Each tiles is typically connected in a king-style fashion to its nearest-neighbor tiles. A daisy-chained bitstream interface is also employed for configuration. This architecture is defined in C++ with CGRA-ME [3], thus it is sythesizable, highly parameterizable and can be easily extended for design space exploration.

II. CGRA Design-space Optimization for HPC

A. Intra-CGRA Interconnect Optimization

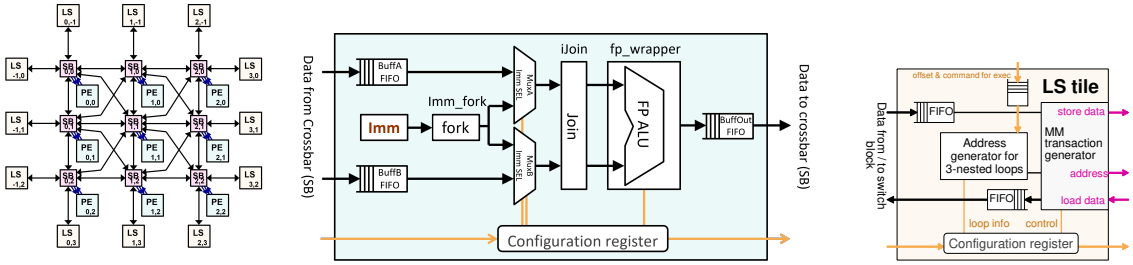
RIKEN CGRA comprises three distinct modules connected by elastic token-based streaming interfaces, namely the Processing Element (PE) tile for computing, the Load-Store (LS) tile to generate memory transactions, and the Switch Block (SB) tile connecting all other tiles forming the whole array. An appropriately designed intra-CGRA interconnect provides enough routability to map common HPC workload kernels while maintaining low hardware costs. We explored ADRES-style integrated mux in each PE vs. HyCube-style interconnect with a discrete router. The latter is far more routable, albeit with 6.3× higher resource usage; however, it is often underutilized at less than 30% connection usage under a typical HPC workload. To find the balance, we evaluate various reduced connection topologies by removing the least used connections and higher compute density topologies [4], [5].

B. Heterogeneous ALUs

Various HPC kernels often require transcendental functions, such as square root and exponential. However, CGRAs usually do not directly implement such operations but instead rely on approximations (e.g., based on lookup tables or Taylor expansion) that can negatively impact the result precision or the number of Arithmetic Logical Units (ALUs)/PEs required. For this reason, we provide our CGRA with a heterogeneous set of ALUs, namely Basic, Complex, and Full. The first ALU implements basic logical, integer, and floating-point operations, while the second features transcendental functions only. Finally, the last ALU groups the previous ones. Table I reports the types of operations available within each ALU. As the number and complexity of operations increase, the required hardware resources also increase. We synthesized three 3×3 CGRAs on Intel's Stratix 10 SX FPGA, each implementing a different type of ALU. The results indicate a slight increase in logic resources as the ALU becomes more complex. On the other hand, the growth of DSPs and RAM blocks is steeper due to the introduction of transcendental functions and the size of internal buffers, which the CGRA employs to guarantee that each operation has the same latency, respectively. Consequently, instead of using just one type of ALU, we plan to combine them within a heterogeneous CGRA and investigate a proper balance to reduce resource usage and, potentially, power consumption.

C. Programmable Buffer

Achievable throughput of CGRAs IO bounded applications is often limited by the number of LS tiles connecting the CGRA to main memory. Ideally, memory accesses must be sequential and non-redundant which is often unachievable. Therefore, we propose a Programmable Buffer (PB) tile, which act as a buffer and masks non-linear and redundant memory



a. RIKEN CGRA baseline overview.

b. Processing Element Tile.

c. Load-Store Tile.

Fig. 1. RIKEN CGRA Architecture

access to the memory controller. The buffer can be adapted to user algorithm requirements, for example, a single PB provides one main output port with a configurable offset in memory, and secondary outputs with consecutive outputs. Multiple PBs tiles can be chained to implement line buffers with more output ports. In addition, the buffer uses configurable forwarding and backwarding to introduce dummy samples on output ports. These are required at start and end of data streams, when real delayed samples are not yet available for some ports. PBs are structured as CGRA columns. In a typical usage scenarion one column of PB is placed next to LS tiles for memory access buffering and another column in the PE area to cache PE outputs. PBs are inserted between SB and connect to existing SB.When a PB is unused, it can be configured to transparently forward data on the connected ports, replicating the behavior of the CGRA without PB.

D. Systolic-Array Style Execution on CGRA

Compared to Systolic Arrays, CGRA offers higher flexibility and programability as users can map arbitrary DFG into the CGRA. However, the flexibility costs a higher logic complexity, thus resulting in more area usage in silicon. Moreover, mapping an arbitrary DFG does not guarantee 100% utilization of the available PE, reducing the efficiency further compared to a simple systolic array. In our lates effort, we proposed several changes to allow systolic array style execution on CGRA, i.e., extra systolic style registers for local data storage on each PE, which is commonly found in systolic array style computation, and defining a DSL to efficiently contruct the mapping primitives that can be placed easily into the CGRA utilizing maximum number of available PEs.

TABLE I
LIST OF OPERATIONS FOR EACH ALU TYPE

ALU Type	Operations
BASIC	Logical: `NOP, AND, XOR, OR, NOT` Integer: `ADD, SUB, MUL, DIV` Floating-point: `ADD, SUB, MUL`
COMPLEX	Floating-point: `EXP, LOG, INV, SQRT`
FULL	Logical: `NOP, AND, XOR, OR, NOT` Integer: `ADD, SUB, MUL, DIV` Floating-point: `ADD, SUB, MUL, EXP, LOG, INV, SQRT`

E. Predication Support

Predication support is necessary to improves the programability of the CGRA, allowing the CGRA to do conditional branching. Several design decisions must be made, for example the type of predication, either data-based predication or a true DFG branching. The former is much simpler to implement but might waste a lot of PEs as both side of the branch are always executed. In addition, the type of predication network, either a dedicated predication network or sharing the datapath. Also, we must consider the compiler support. A further study on the trade-offs between all these factors is necessary.

F. FPGA-Based Emulation

For a large-scale evaluation in an HPC environment, we are developing FPGA-based emulation of the CGRA. The CGRA emulator will be deployed on the ESSPER FPGA cluster connected to Supercomputer Fugaku [6].

REFERENCES

[1] B. Adhi, C. Cortes, Y. Tan, T. Kojima, A. Podobas, and K. Sano, "Exploration framework for synthesizable cgras targeting hpc: Initial design and evaluations," in *The First International Workshop on Coarse-Grained Reconfigurable Architectures for High-Performance Computing (CGRA4HPC)*. IEEE, 2022.

[2] T. Kojima, B. Adhi, C. Cortes, Y. Tan, and K. Sano, "An architecture-independent cgra compiler enabling openmp applications," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2022, pp. 631–638.

[3] 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. [Online]. Available: https://doi.org/10.1145/3597031.3597055

[4] B. Adhi, C. Cortes, E. D. Sozzo, T. Ueno, Y. Tan, T. Kojima, A. Podobas, and K. Sano, "Less for more: Reducing intra-cgra connectivity for higher performance and efficiency in hpc," in *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2023, pp. 452–459.

[5] B. Adhi, C. Cortes, Y. Tan, T. Kojima, A. Podobas, and K. Sano, "The cost of flexibility: Embedded versus discrete routers in CGRAs for HPC," in *Proceedings of the IEEE Cluster conference*, 2022, accepted (10 pages).

[6] K. Sano, A. Koshiba, T. Miyajima, and T. Ueno, "Essper: Elastic and scalable fpga-cluster system for high-performance reconfigurable computing with supercomputer fugaku," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, ser. HPC Asia '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 140–150. [Online]. Available: https://doi.org/10.1145/3578178.3579341