Towards Coarse-Grained Reconfigurable Approximate Computing with CGRAgen

Hans Jakob Damsgaard, Aleksandr Ometov, Jari Nurmi
Electrical Engineering Unit, Tampere University, Tampere, Finland
{hans.damsgaard, aleksandr.ometov, jari.nurmi}@tuni.fi

***Abstract*—Modern Edge Computing devices execute applications that must meet strict latency requirements as per traditional standardization activities. Achieving the needed performance implies a need for efficiency in all aspects, thus, flexible solutions are needed. In this Ph.D. project, we address this issue for error-tolerant applications by using Coarse-Grained Reconfigurable Arrays (CGRAs) enriched with Approximate Computing (AxC) features. To do so, we aim to develop a CGRA architecture modeling, mapping, and hardware generation flow complete with AxC hardware primitives and significance analysis.**

***Index Terms*—approximate computing, coarse-grained reconfigurable array, computation offloading, mapping**

I. Introduction

Effectively tackling the communication demands of the increasingly popular Internet of Things domain requires powerful, distributed computing at the network Edge [1]. Related applications are diverse, and, fortunately, many show resilience to constrained computational errors [1]. The latter permits energy and latency savings through Approximate Computing (AxC), while reconfigurability can address the former. In this Ph.D. project, we will explore the combination of these features in Coarse-Grained Reconfigurable Arrays (CGRAs). These architectures comprise mesh arrays of Processing Elements (PEs) that integrate routing, arithmetic, logic, and buffering hardware, see Fig. 1. CGRAs shine in scenarios where the high-overhead, bit-level reconfigurability of Field-Programmable Gate Arrays (FPGAs) is unnecessary. Integrating multiple AxC techniques into CGRAs remains largely unexplored with initial results being promising [2].

Working with CGRAs requires a powerful yet flexible tool flow. Therefore, we are developing *CGRAgen*, aiming to: 1) mitigate a lack of well-engineered open-source flows for mapping Data Flow Graphs (DFGs) to diverse CGRAs, and 2) enable design space exploration on CGRAs integrating AxC features. CGRAgen is developed in Scala and comprises modules designed to be self-contained and, as far as possible, non-reliant on third-party libraries. Initially, it was closely modeled after CGRA-ME [3] but has since been upgraded substantially. This paper introduces CGRAgen, provides preliminary mapping and hardware generation results, and discusses prospective directions for future work.

The authors gratefully acknowledge funding from European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska Curie grant agreement No. 956090 (APROPOS: Approximate Computing for Power and Energy Optimisation, http://www.apropos-itn.eu/).

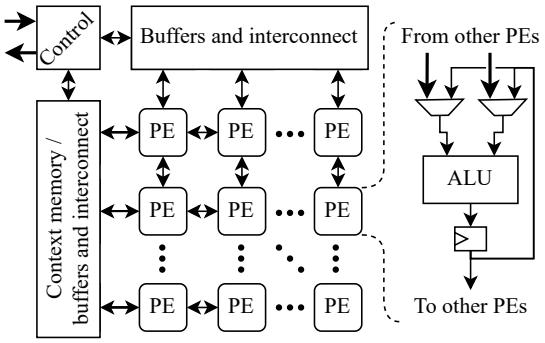

Fig. 1: Overview of a generic CGRA architecture with external control, interconnect, and buffer logic [4].

II. The CGRAgen Flow

Like other open-source CGRA flows [3], [5], CGRAgen models applications as DFGs whose nodes represent logic or arithmetic operations and edges the data dependencies between these. We describe DFGs in a subset of the DOT language, requiring that 1) nodes define the opcode attribute that determines which PEs they can be mapped to, and 2) edges define the operand attribute that indicates their arithmetic position in the sink operation, in addition to their source and sink nodes. Internal to CGRAgen, this representation is transformed into an actual graph before mapping. So far, we have resorted to describing DFGs manually but aim at automating this with a traditional compiler flow as in other open-source tools [3], [5]. We also intend to add a significance analysis pass that assigns approximation modes to operations given quality constraints of individual outputs to simplify using AxC techniques.

As part of our aim to simplify design space exploration, we adopt and reduce the XML-based Architecture Description Language (ADL) of CGRA-ME [3]. It stands out greatly compared with the involved languages used in other flows [6], [7] but suffers from *syntactic sugar* extensions that complicate parsing without adding major value. Our reduced ADL enables describing heterogeneous CGRAs with templated, hierarchical PEs based on a set of pre-defined primitives. Within CGRAgen, architectures are represented in two formats: 1) as abstract modules with ports, sub-modules, and connections between these, and 2) as Modulo Resource Routing Graphs (MRRGs) commonly used for mapping [3], [8]. We have extended the formats to support parameterizable latencies and initiation intervals for operations in Arithmetic Logic Unit

TABLE I: Mapping results in order of increasing runtime.

Application	# operations I/O	const.	logic	# values	Min. *II*	Runtime
2×2 **convolution**	5	4	7	15	1	5 *s*
2nd order poly.	2	2	3	7	1	17 *s*
4th order poly.	2	4	7	15	1	193 *s*
4-point DCT	8	2	10	24	1	6821 *s*
3×3 **convolution**	10	9	17	35	2	33009 *s*
Sobel operator	9	3	16	33	1	60691 *s*

(ALU) primitives. We also aim to integrate approximation modes with verified, guaranteed quality levels to support AxC.

CGRAgen uses an Integer Linear Programming (ILP)-based mapper, which assumes that the PEs operate in lockstep. While this approach is functional, it is not flawless; ILP solvers are known for being slow, and the model size using the constraints from [3] quickly grows, even for small CGRAs. As a result, in anticipation of wanting to explore alternative mapping algorithms, like in [5], CGRAgen is designed to be extensible and its mapping representation has full support for multi-cycle contexts. To automate the use of AxC, we will extend the mapper to consider the approximation modes supported by the application and architecture too.

Recently, we have extended CGRAgen with a hardware backend capable of generating Verilog descriptions of arbitrary module templates. Like Pillars [6], we base this extension on the Chisel [9] Hardware Description Language (HDL) as it integrates well with the existing flow and permits the generation of dynamically defined hardware modules, including their IO. We find that these facets greatly cut down design times while likely reducing the error proneness of hand-written HDL generators and templates common to existing flows [3], [7]. This hardware backend is the main focus of our current efforts as we aim to 1) optimize code generation to avoid unnecessary redundancies; 2) extend support to generate entire CGRAs; and 3) integrate support for AxC techniques, starting with approximate arithmetic units [2], [10].

III. Preliminary Results

CGRAgen can currently map simple DFGs to complex CGRAs and generate hardware descriptions of any template module. To illustrate this, we include mapping results of a handful of DFGs, inspired by that of [2, Table VI], targeting a 4×4 CGRA with HyCUBE-inspired [11] PEs equipped with crossbar-style routing. Four of the PEs share a register file. The results, included in Table I, are well in line with prior work [3] and highlight that the ILP-based mapper suffers from very long execution times, underlining the need for integrating alternative, likely heuristic-based algorithms to speed up mapping [5].

To demonstrate the hardware generation capabilities, we generate Verilog descriptions of the simple PE shown in Fig. 1. For this experiment, we assume neighbor-only connections and a fully combinational ALU that supports addition and subtraction operations. The corresponding module template comprises only 16 ADL statements, of which eight define its IO. We synthesize and implement the design for a Xilinx Zynq UltraScale+ FPGA using Vivado 2022.2 and report the resulting utilization and maximum clock frequency numbers in Fig. 2. The design does not use any DSP primitives.

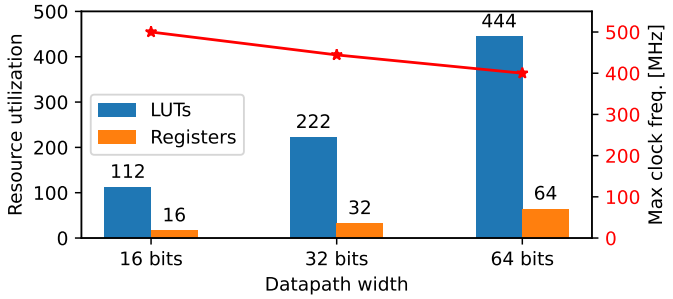

Fig. 2: Resource utilization and maximum clock frequencies for a PE targeting a Xilinx Zynq UltraScale+ FPGA.

IV. Conclusion

In this paper, we introduced CGRAgen, a CGRA architecture modeling, mapping, and hardware generation flow. CGRAgen aims to extend upon existing work with AxC features, including hardware primitives and significance analysis. In its current state, the flow's ILP-based mapper can map simple DFGs to complex architectures, and its hardware backend can generate Verilog descriptions of individual PEs. Once complete, the flexibility of CGRAgen will permit fast and easy design space exploration of, potentially heterogeneous, CGRA architectures and the effects of AxC.

References

[1] H. J. Damsgaard *et al.*, "Approximation Opportunities in Edge Computing Hardware: A Systematic Literature Review," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–49, 2022.

[2] O. Akbari *et al.*, "X-CGRA: An Energy-Efficient Approximate Coarse-Grained Reconfigurable Architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2558–2571, 2019.

[3] S. A. Chin *et al.*, "CGRA-ME: A Unified Framework for CGRA Modelling and Exploration," in *Proceedings of 28th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*. IEEE, 2017, pp. 184–189.

[4] A. Podobas *et al.*, "A Survey on Coarse-Grained Reconfigurable Architectures From a Performance Perspective," *IEEE Access*, vol. 8, pp. 146 719–146 743, 2020.

[5] S. Dave and A. Shrivastava, "CCF: A CGRA Compilation Framework," 2017.

[6] Y. Guo and G. Luo, "Pillars: An Integrated CGRA Design Framework," in *Workshop on Open-Source EDA Technology (WOSET)*, 2020.

[7] T. K. Bandara *et al.*, "REVAMP: A Systematic Framework for Heterogeneous CGRA Realization," in *Proceedings of 27th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2022, pp. 918–932.

[8] B. Mei *et al.*, "DRESC: A Retargetable Compiler for Coarse-Grained Reconfigurable Architectures," in *Proceedings of International Conference on Field-Programmable Technology*. IEEE, 2002, pp. 166–173.

[9] J. Bachrach *et al.*, "Chisel: Constructing Hardware in a SCALA Embedded Language," in *Proceedings of 49th Design Automation Conference (DAC)*. ACM, 2012, pp. 1212–1221.

[10] H. J. Damsgaard, "approx: A Library of Approximate Arithmetic Units in Chisel," https://github.com/aproposorg/approx, 2022.

[11] M. Karunaratne *et al.*, "HyCUBE: A CGRA with Reconfigurable Single-Cycle Multi-Hop Interconnect," in *Proceedings of 54th Design Automation Conference (DAC)*. ACM, 2017, pp. 1–6.