

# CUT-MC: Optimizing the Relationship Between Context Count, Unrolling and Throughput in Multi-Context CGRAs

Stephen Wicklund<sup>†,‡</sup> and Jason Anderson<sup>†</sup>

<sup>†</sup>University of Toronto, <sup>‡</sup>Fujitsu Consulting (Canada) Inc.  
Toronto, Ontario, Canada

stephen.wicklund@fujitsu.com, janders@ece.utoronto.ca

**Abstract**—We propose techniques to raise throughput for compute-intensive loops when implemented in multi-context CGRAs via judicious choices for context count and loop-unroll factor. We show that using a non-minimal initiation interval ( $II$ ) can yield higher throughput in many applications (vs. using the minimum  $II$ ), provided there is sufficient loop unrolling. Average throughput improvements of 36% are achieved across a range of applications, CGRA sizes and context counts.

## I. INTRODUCTION

Modern computing has entered the accelerator era where general-purpose processor performance plateaus due to the slowing of Moore’s Law [1] and the physical limits of traditional scaling [2]. This trend has led to an increased reliance on domain-specific accelerators, which offer higher efficiency and performance than general-purpose CPUs for specific workloads. Coarse-Grained Reconfigurable Arrays (CGRAs) present a middle ground between FPGAs and ASICs by offering higher efficiency than FPGAs while retaining adaptability. A challenge for EDA tools is to produce implementations that maximize computational throughput by leveraging the underlying capabilities of the target IC media. We propose compilation techniques for raising the throughput of CGRAs.

Many CGRAs offer a feature called *multi-context* [3], [4], where the CGRA is loaded with multiple configuration bitstreams: each referred to as a *context*. The CGRA switches contexts on a cycle-by-cycle basis, allowing the CGRA hardware to be time multiplexed. Typically, context changes are done in a round-robin manner, e.g., with four contexts, the CGRA iterates through contexts 0, 1, 2, 3, 0, 1, . . . . Round-robin context switching is common as CGRAs are often used to accelerate compute-intensive loops. The multi-context notion is closely related to the loop’s initiation interval ( $II$ ). In multi-context CGRAs, the initiation interval ( $II$ ) equals the number of contexts required for the loop body. If a loop body is implemented in a 2-context CGRA, a new loop iteration will commence every 2 clock cycles, implying  $II = 2$ . This work explores how Context count and loop Unrolling affect Throughput in Multi-Context CGRAs (CUT-MC).

Conventional wisdom is that implementations with the lowest  $II$  offer the best throughput. We challenge this wisdom by showing that in certain scenarios, a higher non-minimal  $II$  offers better throughput, when combined with loop unrolling.

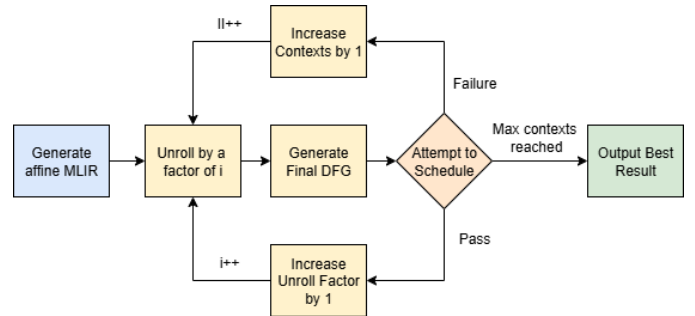


Fig. 1: Flowchart for multi-context unrolling exploration.

If an unrolled-by-3 loop body can fit into a two-context CGRA ( $II = 2$ ), the throughput is  $3/2 = 1.5$ , which is 50% higher than the throughput associated with  $II = 1$ . Counterintuitively, a higher  $II$  leads to higher throughput. This occurs when the loop unroll factor increases faster than the  $II$  (number of contexts) needed for successful mapping. It is precisely this phenomenon that we leverage in this work.

Our work is implemented within MLIR-to-CGRA [5], an MLIR-based front-end compiler that translates high-level C/C++ code into optimized Data Flow Graphs (DFGs) mappable to CGRAs through the CGRA-ME framework [6]. We introduce new compiler passes that systematically evaluate loop unrolling factors while dynamically adjusting the  $II$  to identify throughput-optimal configurations. Experimental results demonstrate that the proposed techniques significantly raise throughput, by 36% on average, across a variety of benchmarks and CGRA sizes.

## II. UNROLLING & MULTI-CONTEXT EXPLORATION

Fig. 1 shows the flow of the multi-context unrolling exploration approach. The process begins with generating the initial affine dialect MLIR, following the procedure of MLIR-to-CGRA [5]. At this stage, a copy of the compiler’s intermediate representation (IR) is saved, as the copy will be used as the starting point for generating multiple unrolled DFGs.

The next step is to unroll the main loop body. The unroll factor starts at 1 (i.e., no unrolling). For a given unroll factor, the compiler attempts to generate and evaluate an unrolled DFG. The unroll factor is increased only if the generated DFG is accepted, up to a configurable maximum threshold. Next, the

TABLE I: Base and unrolling exploration optimized throughput for benchmarks and different CGRA sizes.

Benchmark	8 × 8		10 × 10		12 × 12		14 × 14		16 × 16		Mean Speedup
	Base	Opt	Base	Opt	Base	Opt	Base	Opt	Base	Opt	
<b>fir</b>	2.0	2.29	2.0	2.88	3.0	3.5	3.0	4.13	4.0	4.71	1.26
<b>mvt</b>	0.5	0.60	0.5	0.75	0.5	0.88	1.0	1.00	1.0	1.20	1.33
<b>gesummv</b>	0.33	0.33	0.33	0.43	0.5	0.5	0.5	0.63	0.5	0.71	1.19
<b>relu</b>	1.0	1.60	2.0	2.0	2.0	2.4	2.0	2.8	3.0	3.2	1.25
<b>seidel-2d</b>	0.33	0.43	0.5	0.6	0.5	0.71	0.5	0.83	0.5	0.88	1.47
<b>trisolv</b>	0.5	0.5	0.5	0.67	0.5	0.8	0.5	0.88	1.0	1.0	1.34
<b>floyd-warshall</b>	0.5	0.88	1.0	1.2	1.0	1.43	1.0	1.67	1.0	1.88	1.58
<b>spmv</b>	0.5	0.8	1.0	1.0	1.0	1.2	1.0	1.4	1.0	1.6	1.36
<b>bigc</b>	0.33	0.43	0.5	0.6	0.5	0.71	0.5	0.83	0.5	0.88	1.47
<b>Size Mean Speedup</b>	1.32		1.24		1.36		1.46		1.43		<b>1.36</b>

final passes from the MLIR-to-CGRA tool are run to ensure the DFG is correctly generated after unrolling. Afterward, an attempt is made to schedule the generated DFG. For this work, a heuristic resource-constrained scheduler was developed based on the modulo System of Difference Constraints (SDC) scheduling of the LegUp high-level synthesis tool [7]. A final check determines whether the generated DFG with the current unrolling factor is accepted. If the CGRA utilization exceeds configured thresholds, the DFG is rejected.

If the DFG is accepted, the unrolling factor increases. If the DFG is not accepted, the unrolling factor remains unchanged, and the *II* (number of contexts) is increased by 1. If the *II* reaches a configured limit, the pass concludes. The successfully generated DFGs are saved for later analysis, from which the best-performing configuration can be selected.

### III. EXPERIMENTAL RESULTS

We evaluate the multi-context unrolling exploration by comparing the throughput vs. values acquired using the baseline MLIR-to-CGRA method. For the multi-context unrolling, we considered *II* values from 1 to 8, reflecting an architecture with 8 contexts. This represents a reasonable # of configurations, similar to other multi-context CGRA works [3], [8], [9]. The resource utilization threshold is set to 60% of memory and ALU resources. A legal mapping was possible for all application DFGs. CLUMAP [10], integrated into CGRA-ME 2.0 [6], [11], is used for mapping. CLUMAP uses simulated annealing-based placement, followed by negotiated congestion routing [12]. A collection of applications from Polybench [13], Machsuite [14], and CGRA-Bench [15] was used to evaluate the proposed methods. A modified HyCUBE [5], [16] supporting predication is the target CGRA.

Fig. 2 shows the multi-context unrolling exploration throughput compared with the baseline throughput for each benchmark and architecture size explored. The vertical axis reflects throughput. The blue portion of each bar is the minimum-*II* baseline throughput; the orange portion of each bar indicates higher throughput from the use of non-minimal *II* values. The chart illustrates that multi-context unrolling exploration yields gains in throughput with even modest increases in CGRA size, allowing performance to scale roughly linearly with the increase of CGRA resources.

Table I shows the throughput achieved in the baseline and with the unrolling/multi-context exploration, for five different CGRA sizes. Observe that all benchmarks achieved a speedup

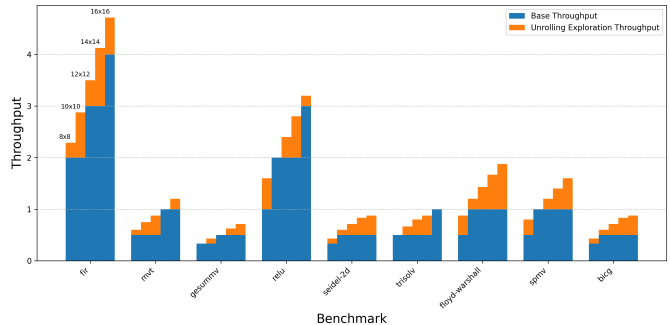


Fig. 2: Stacked bar chart showing base throughput (blue) and multi-context unrolling exploration throughput (orange) for each benchmark at CGRA sizes: 8 × 8, 10 × 10, 12 × 12, 14 × 14 and 16 × 16.

greater than the baseline min-*II* method for at least one architecture size. Depending on CGRA size, average throughput improvements range from 24% to 46%. For individual benchmarks, throughput improvements up to 75% are observed. It is expected that the proposed method does not provide a performance gain for some benchmarks and CGRA sizes: The best throughput is achieved when the utilization is at the utilization limit. So, if the baseline min-*II* case can achieve this through unrolling at the lowest-possible *II*, then the proposed multi-context unrolling approach will not find any better results. Overall, the results clearly demonstrate that use of non-minimal *II* values, combined with unrolling and targeting multi-context CGRAs, can yield significant throughput improvements over a baseline min-*II* approach.

### IV. CONCLUSIONS

We proposed CGRA compilation techniques to raise computational throughput by using loop unrolling and taking advantage of multiple CGRA contexts (configurations). Conventional wisdom suggests that the highest throughput is associated with the lowest *II*. We demonstrated that higher throughput can be achieved using a non-minimal *II*, provided that the loop unroll factor exceeds the new *II* and that the application can be mapped into the target CGRA size. We incorporated resource-constrained scheduling and a multi-context unrolling exploration pass into the MLIR-to-CGRA [5] framework for such throughput optimization. Across all benchmarks and CGRA sizes considered, and considering 8 available CGRA contexts, average throughput improvements are ~36%.

## REFERENCES

- [1] G. E. Moore, "Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff.," *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, 2006.
- [2] H. Esmailzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *IEEE/ACM ISCA*, pp. 365–376, 2011.
- [3] V. Chacko and J. Anderson, "Power, performance and area consequences of multi-context support in CGRAs," in *IEEE ASAP*, pp. 49–52, 2021.
- [4] T. Toi, N. Nakamura, T. Fujii, T. Kitaoka, K. Togawa, K. Furuta, and T. Awashima, "Optimizing time and space multiplexed computation in a dynamically reconfigurable processor," in *IEEE FPT*, pp. 106–111, 2013.
- [5] T. Yu, O. Ragheb, S. Wicklund, and J. Anderson, "MLIR-to-CGRA: A versatile MLIR-based compiler framework for CGRAs," in *IEEE ASAP*, pp. 184–192, 2024.
- [6] O. Ragheb, S. Wicklund, M. Walker, R. Beidas, A. Ragab, T. Yu, and J. Anderson, "CGRA-ME 2.0: A research framework for next-generation CGRA architectures and CAD," in *CGRA4HPC*, pp. 642–649, 2024.
- [7] A. Canis, S. D. Brown, and J. H. Anderson, "Modulo SDC scheduling with recurrence minimization in high-level synthesis," in *FPL*, pp. 1–8, 2014.
- [8] R. Beidas and J. H. Anderson, "Mapping enumeration for multi-context cgras using zero-suppressed binary decision diagrams," in *IEEE FCCM*, pp. 151–161, 2024.
- [9] O. Ragheb, T. Yu, R. Beidas, and J. Anderson, "Elastic multi-context CGRAs," in *CGRA4HPC*, pp. 655–662, 2022.
- [10] O. Ragheb and J. H. Anderson, "CLUMAP: Clustered mapper for CGRAs with predication," in *ACM/IEEE DAC*, 2024.
- [11] 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 *IEEE ASAP*, pp. 184–189, 2017.
- [12] L. McMurchie and C. Ebeling, "PathFinder: A negotiation-based performance-driven router for FPGAs," in *ACM FPGA*, pp. 111–117, 1995.
- [13] T. Yuki, "Understanding polybench/C 3.2 kernels," in *Int'l Workshop on Polyhedral Compilation Techniques (IMPACT)*, 2014.
- [14] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "MachSuite: Benchmarks for accelerator design and customized architectures," in *IEEE International Symposium on Workload Characterization*, October 2014.
- [15] C. Tan, "CGRA-bench," 2020.
- [16] M. Karunaratne, A. K. Mohite, T. Mitra, and L.-S. Peh, "HyCUBE: A CGRA with reconfigurable single-cycle multi-hop interconnect," in *IEEE/ACM DAC*, 2017.