

Voltage Aware Approximate CGRA Synthesis for Energy Efficient DNN Inference

Georgios Alexandris¹, Panagiotis Chaidos¹, Alexis Maras¹, Barry de Bruin², Manil Dev Gomony²,
Henk Corporaal², Dimitrios Soudris¹, Sotirios Xydis¹

¹National Technical University of Athens, Greece

²Eindhoven University of Technology, The Netherlands

{galexandris,pchaidos,amaras,dsoudris,sxydis}@microlab.ntua.gr, {e.d.bruin,m.gomony,h.corporaal}@tue.nl

Abstract—The ever-increasing complexity and operational diversity of modern Neural Networks (NNs) have caused the need for low-power and, at the same time, high-performance edge devices for AI applications. Coarse Grained Reconfigurable Architectures (CGRAs) form a promising design paradigm to address these challenges, delivering a close-to-ASIC performance while allowing for hardware programmability. In this paper, we introduce a novel end-to-end exploration and synthesis framework for approximate CGRA processors, enabling the transparent and optimized integration and mapping of approximate multiplication components into CGRAs. Our framework includes an exploration of state-of-the-art approximate multiplication units on the hardware side, along with a software exploration, based on a per-channel model analysis, that maps specific output features onto approximate components based on accuracy degradation constraints, utilizing also SW-based optimization techniques. This enables the optimization of the system’s energy consumption while retaining the accuracy above a certain threshold. At the circuit level, the integration of approximate components enables the creation of voltage islands that operate at reduced voltage levels, which is attributed to their inherently shorter critical paths. This key enabler allows us to effectively reduce the overall power consumption by an average of 30% across our analyzed architectures, compared to their baseline counterparts, while incurring only a minimal 2% area overhead. The proposed methodology was evaluated on the convolutional kernels of a widely used NN model, MobileNetV2, on the ImageNet dataset, demonstrating that the generated architectures can deliver up to 440 GOPS/W with relatively small output error during inference, outperforming several State-of-the-Art CGRA architectures in terms of throughput and energy efficiency.

Index Terms—Approximate CGRA, Heterogeneous Architectures, Low Power CGRA Mapping, Edge AI, Voltage Scaling

I. INTRODUCTION

The rapid advancement of Artificial Intelligence (AI) has driven significant efforts toward deploying AI applications in edge computing environments. A key challenge in edge AI lies in optimizing energy efficiency while sustaining high performance under acceptable rates [1]. Despite the stringent resource constraints of edge devices, significant research efforts have been dedicated to developing high-performance accelerators capable of delivering competitive results. A range of specialized tensor accelerators, such as those targeting matrix multiplication and convolution operations, have been developed for edge

AI applications [2] [3], aiming to achieve balanced trade-offs between computational throughput and energy efficiency.

Coarse-Grained Reconfigurable Processor Architectures (CGRAs) [12] provide a great alternative to specialized tensor accelerators due to their fine-grained configurability and adaptability across diverse neural workloads, offering a balance between ASIC-level efficiency and post-deployment flexibility that fixed-function accelerators lack [2], [3]. Typical CGRA schemes, e.g. CGRA-ME [6], utilize a standardized mesh-based computational grid, with a DFG-based programmability. To improve energy efficiency, X-CGRA [4] introduces approximation on each PE, allowing programmability from a RISC-like ISA with configurable approximation levels, while GREEN [13] also supports precision scale on its tiles, along SIMD support (vector operations). Brandalero et al. [5] explored different approximation levels per tile, while interconnecting each row of PEs with a shared banked memory. Extending on HyCUBE [14], REVAMP [7] provided a heterogeneous PE formation at the CGRA level, while running an automated DSE to determine the best architecture formation per dataflow. On the same track, R-Blocks [11] also provided such a heterogeneous CGRA architecture scheme, making it able to map applications from high-level descriptions (C language). More recently, ICED CGRA [10] introduced and explored Dynamic Voltage and Frequency Scaling (DVFS) support for spatio-temporal CGRA mapping. Table I summarizes the qualitative features of prior work, clearly showing a research gap in CGRA methodologies able to efficiently exploit and analyze approximate computing in conjunction with voltage scaling for Deep Neural Network (DNN) acceleration.

In this paper, we propose a seamless end-to-end synthesis framework for CGRAs, designed to efficiently map high-level DNN kernels onto approximate-aware reconfigurable hardware. The proposed flow systematically analyzes each DNN layer to assess its sensitivity to approximation and selectively assigns computations either to approximate or accurate computational units, ensuring controlled DNN accuracy degradation. Additionally, the framework incorporates voltage islanding techniques, allowing several CGRA parts to operate at reduced voltage levels, taking advantage of the diverse timing paths distribution across CGRA tiles, thereby maximizing energy savings. The key contributions of our methodology are:

- We implement and analyze a novel end-to-end C-to-RTL

This work is funded in part by the Convolv project evaluated by the EU Horizon Europe research and innovation program under grant agreement No. 101070374

CGRA Architecture	Approximate Computing	Voltage Scaling	Flexible PEs	High Level Programmability	DNN Support
X-CGRA [4]	✓	Static	✗	DFG-Based	✗
Brandalero et al. [5]	✓	✗	✓	Behavioural Model	✗
CGRA-ME [6]	✗	✗	✗	DFG-Based	✗
REVAMP [7]	✗	✗	✓	DFG-Based	✗
CGRA4ML [8]	✗	✗	✗	Custom Python Flow	✓
ML-CGRA [9]	✗	✗	✗	MLIR from Python	✓
ICED [10]	✗	Dynamic	✗	DFG-Based	✗
R-Blocks [11]	✗	✗	✓	C compiler	✗
Our Work	✓	Static	✓	C compiler	✓

TABLE I: Qualitative comparison among the state-of-the-art CGRAs

mapping framework for approximate CGRAs, which co-explores resource allocation and QoS applied on modern DNN models. By systematically analyzing the importance of each DNN layer’s output features, our flow dynamically assigns computations to either accurate or approximate functional units, ensuring an optimal trade-off between computational accuracy and energy efficiency under user-defined constraints.

- We provide an arithmetic approximation to our CGRAs multiplication processing elements. The unit selection comes as an extensive analysis among state-of-the-art solutions [15]–[17], providing solutions based on the energy consumption, area utilization, and performance optimization among the implementations given.
- A voltage island-based power optimization strategy that enables groups of CGRA’s processing elements (PEs) to operate at different supply voltages. By strategically assigning voltage levels to PEs executing approximate and accurate operations, we enable effective cooperative optimization between approximate computing and voltage islanding, thereby aggressively minimizing energy consumption while maintaining computational integrity.
- Through post-synthesis experimental evaluation, utilizing MobileNetV2 on ImageNet [18] dataset as our driver edge AI application, we show that the proposed CGRA optimizations implemented in a 22nm technology node achieve energy improvements of up to 30%, when compared to several iso-resource CGRA designs, while achieving 378 to 440 GOPS/W, significantly outperforming several state-of-the-art CGRA architectures and frameworks.

II. VOLTAGE-AWARE APPROXIMATE CGRA

A. Exploration of Approximate Units

For an arithmetic approximation to be considered, we need to be as energy-efficient and area-constrained as possible w.r.t. the error introduced by the approximation approach.

For this reason, we utilized an exploration of three SotA architectures. The first one is ROUP [16], which is a rounding and perforation-based approximate unit that also includes error correction on the LSBs of the produced result. The ROUP multiplier performs rounding up to the r -bit, cutting the p last partial products generated (ROUP $_p$ _ r configurations). The second architecture is DRUM [15], which incorporates a leading one detector, and then utilizes a $k \times k$ bit accurate multiplication (k is user defined), truncating the remaining LSBs (DRUM $_k$

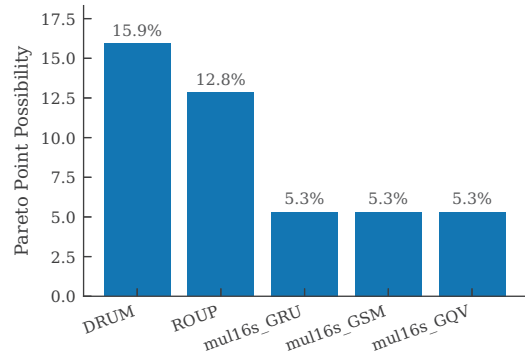


Fig. 1: Possibility of the Approximation Technologies of Being in a Pareto Front

configurations). DRUM also implements unbiasing at the final LSB for error correction. The last architecture to be considered is the library of EvoApproxLib [17], which uses genetic algorithms to find optimal approximate components, letting the evolution algorithm define the pruning of certain hardware parts during the execution. All the configurations coming from EvoApproxLib are custom.

For the exploration of the above designs, we implemented a customized testbench as a Python library, replacing certain multiplications of a convolution kernel with each approximate unit. The library was built upon PyTorch firmware to provide direct access to a prebuilt NN model to evaluate. The user could configure the convolutional layers of a selected model to schedule all of their multiplication operations through our customized library.

The backend of our implemented flow runs a behavioral simulation using Verilator firmware first to determine the Mean Squared Error (MSE) metric of each layer when approximation is introduced, and secondly, generates a switching activity file (SAIF) of each multiplication operation. The generated SAIF file is then processed through Synopsys Power Compiler along with our approximate units’ RTL representation, to provide as accurate a power estimation as possible. Synopsys Design Compiler is additionally used to determine the area of each approximate component using GlobalFoundries 22nm FD-SOI technology library.

Figure 1 presents the possibility of each approximation architecture being on the Pareto front among our analysis results. In our evaluation, we tested designs among different

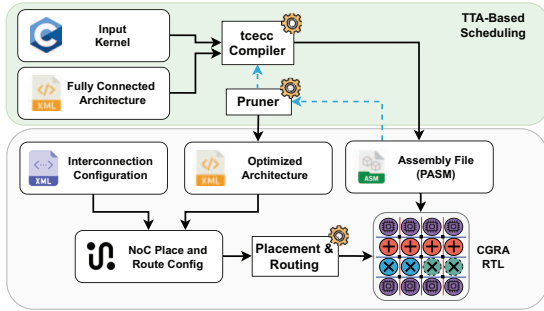


Fig. 2: Code-to-RTL Mapping Flow Overview

convolutional units, using both different convolution parameters and different data inputs. Targeting Energy per operation and MSE as the main exploration paths, the total Pareto points given by our analysis were around 220, from which the possibilities were calculated.

From the results produced, we chose to proceed with DRUM [15] as our approximate architecture, choosing a k configuration of 5, which provides great energy savings (near $\times 2$ compared to an accurate multiplier) with minimal error introduction.

B. CGRA Architecture and Mapping Overview

As already mentioned, CGRAs try to balance the trade-off between performance and reconfigurability. For this purpose, the general CGRA scheme consists of interconnected processing elements (PEs), with each one of them having a certain ISA (it can be the same or different). This interconnected grid, along with certain circuitry for proper memory fetching and instruction decoding, functions as a complete reprogrammable accelerator, with each of the PEs running different instructions per clock cycle, increasing the parallelism as well as the overall efficiency of the given accelerator. This provides a very flexible and ready-for-the-modern-era accelerator, adapting their mapping needs with modern AI computational dataflows (e.g., heterogeneous convolution kernels).

We adopt a modern approach to the classic CGRA scheme, thus basing our analysis on the heterogeneous CGRA, i.e. different ISA per PE [11] which can be configured both in scalar and in vectorized mode, driving the same instruction with different data inputs to one or more PEs, accordingly. The approximate multiplication PEs are based on the DRUM [15] multiplier after the extensive analysis presented on the previous section, while implementing the same ISA as the accurate multiplication (MUL) PE. The programmability of the CGRA is achieved through custom Instruction Decode (ID) PEs, driving the requested instruction either to one or more PEs per clock cycle. The architecture utilizes two NoCs, one for the data and one for the configuration word transmission among PEs (with blue and red in Fig.2), with Wilton-type Switchboxes [19] controlling the information flow. Register files (RFs) are utilized for quick data transferring as well as local memories as PEs (LM) to enhance the overall throughput of the proposed accelerator.

CGRA Application Mapping: An overview of our proposed compilation and mapping flow is presented in Figure 2. Our

compilation flow utilizes the OpenASIP toolchain, which, by applying scheduling schemes to Transport Triggered Architecture (TTA) processors [20], optimizes data and instruction communication based on the PE interconnection and the DFG of the application. The applications are described in a high-level C language, with the support of macros and intrinsics, also allowing the user to control the mapping process (e.g., guiding operations to be mapped to a specific PE). Through specific datatypes, users map operations to vectorized PEs, enabling the SIMD functionality of the CGRA. Bypassing is also supported, i.e., buffering the output of some PEs within them, rather than performing the round trip between memories (Local or External) [21]. After the compilation succeeds, an automated analysis optimizes the code (pruner), and the final machine code binary is available. The final CGRA RTL is generated by combining the bitstream file with appropriate NoC and PE configuration files, producing the final HW design.

Leveraging our compilation framework, we extend the existing intrinsic support to incorporate the usage of approximate multipliers, enabling fine-grained control over their integration during the mapping process. Through the mapping methodology, presented in Section III, the compiler is automatically driven to direct computations to accurate multiplier processing elements (PEs) when the associated output error exceeds a predefined threshold, while routing less critical operations to approximate multiplier PEs to optimize energy efficiency without violating application-level accuracy constraints.

C. Voltage Island Formation

Taking advantage of the heterogeneous CGRA micro-architecture, it is safe to say that different PEs can achieve different timing slacks, meaning that we can explore different voltages applied on each PE, trying to approach maximum energy gains without introducing errors to the produced circuits. The low variance in the PE-level, which resulted from DNN workload testing, led us to use static voltage scaling instead of implementing more dynamic solutions. This strategy reduces the total power consumption w.r.t. timing violations on the circuit, as voltage reduction is applied only to tiles with sufficient slack.

Figure 3 illustrates the timing slacks for the most computationally intensive and power-consuming tiles when integrated into the CGRA. The remaining tiles contribute significantly less to the overall power consumption and are, therefore, excluded from this analysis. Components such as the approximate multipliers and the ALUs exhibit increased timing slack. This indicates that their operating voltage can be safely reduced without violating timing constraints, whereas the accurate multipliers should remain at higher voltages to ensure stability and correct functionality.

Considering the analysis presented, we chose to implement two voltage domains on our proposed CGRA, i.e. one voltage domain at 0.6V incorporating the approximate multiplication tiles, the ALUs, the Register Files, and the switchboxes that are connected to these tiles, and a second one at 0.8V for the remaining tiles. While more fine-grain voltage islands are theoretically possible, it can significantly increase design

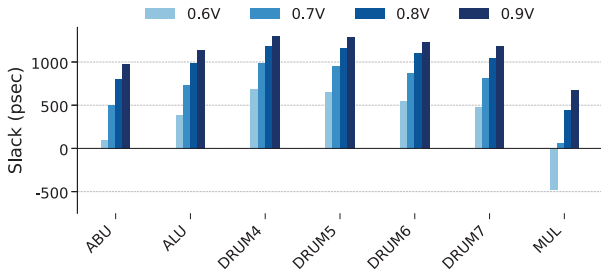


Fig. 3: Timing slack comparison at different voltage levels for the most power-hungry components in the proposed CGRA. Results are based on post-synthesis measurements, considering a clock frequency of 400 MHz at 22nm node.

complexity, as it strains the physical design constraints, leading to challenges in the power delivery network design, routing congestion, and area overhead due to the required placement of level shifters between different voltage regions [22].

The formation of the defined voltage regions was implemented with the use of Unified Power Format (UPF) descriptions to enable proper power-aware synthesis. Since the goal is to better balance the critical paths of the individual components (and tighten them whenever possible) by reducing the operating voltage of the DRUM multipliers and the ALUs, while maintaining the voltage of the accurate ones at the nominal value, we effectively decrease the timing slack deviation among the CGRA tiles from 300 psec to 104 psec.

III. PROPOSED MAPPING FRAMEWORK

This section describes the proposed design flow, detailing how DNN models are analyzed, optimized, and mapped onto the proposed approximate CGRA architecture. We formed a step-by-step methodology, shown in Figure 4, starting from a high-level DNN model down to an optimized CGRA mapping that balances energy efficiency, computational accuracy, and hardware utilization.

A. Approximation-Aware DNN Model

The process begins with a high-level DNN model, specifically implemented using the PyTorch framework and extended via the Brevitas library [23] ①. To incorporate approximate computing into our framework, we extend Brevitas to simulate the behavior of approximate multipliers, ensuring an accurate representation of their impact on neural network computations by enabling the approximation of the features of a layer using dedicated approximate computing functional units. The behavior of the approximate functional units is described in the form of Look Up Tables (LUTs) that store the outcomes of all possible $N \times N$ -bit approximate multiplications, allowing efficient value retrieval during inference.

B. Importance Factors Calculation

After extracting all the necessary information about our model’s layers, we proceed with the calculation of the Importance Factors ②. The concept of importance estimation was originally introduced in the work of Molchanov et al. [24] as a fundamental principle for layer pruning in deep neural

networks (DNNs). The authors calculate the importance of neurons (filters) by assessing the squared change in loss that results from their removal from the network. They approximate this calculation using Taylor expansions to ensure efficiency, especially in larger networks. The primary focus is on its first-order approximation. More specifically, the importance score for a filter is defined as $I_m(W) = (g_m w_m)^2$, where g_m represents the gradient available from backpropagation and w_m is the corresponding weight of the filter.

In our work, we adopt the same fundamental principle, but instead of performing kernel pruning, we extend the importance analysis to all convolutional kernels comprising each DNN filter. We propose calculating the Importance Factors for all kernels to determine which one should be mapped on the approximate components of CGRA, rather than eliminating them from the network. This allows us to examine fine-grained allocation strategies to effectively assign different output channels within the same layer to either accurate or approximate multipliers. Implementing the above allocation strategy on top of a DNN dataflow that exploits output channel parallelism offers the given CGRA increased capabilities/opportunities for parallel execution without sacrificing the DNN’s accuracy. For each layer, the Importance can be calculated as follows:

$$I_{oc,l} = MSE(Q_{out}(D, W), Q_{ax}(D, W, oc, l)) \quad (1)$$

where $I_{oc,l}$ denotes the Importance of the output channel oc in layer l , Q_{out} refers to the output feature map that is produced when a set of input data D is provided to a model with a weight set of W , and $Q_{ax}(D, W, oc, l)$ is the approximate output feature map generated, when the same input data D and weights W are given, but with approximate multiplications applied only on the oc channel of layer l .

C. Approximate vs. Accurate Kernel Mapping Strategy

Once the Importance Factors ($I_{oc,l}$) have been calculated, the next step ③ determines how each layer will be mapped. For our analysis, we consider a CGRA architecture comprising both an approximate and an accurate vector computation region (see also Section IV-C). This architectural configuration, in conjunction with the output channel parallel DNN dataflow mentioned above, enables the concurrent execution of both accurate and approximate multiplications within a single clock cycle, thereby maximizing computational parallelism.

The mapping process is structured into two distinct stages, as shown in Figure 4. i) Sorting by Importance Factor: For each DNN layer, the output channels (OCs) are ranked in descending order based on their Importance Factors, prioritizing the most significant channels. ii) Approximate-Aware Mapping: Given a user-defined Quality-of-Service (QoS) constraint α , the system assigns output channels to approximate multipliers starting from the least significant, progressively mapping additional channels until the QoS threshold is reached. This structured mapping strategy ensures that computational resources are efficiently allocated, allowing approximate multipliers to be fully utilized while maintaining accuracy constraints dictated by the QoS parameter.

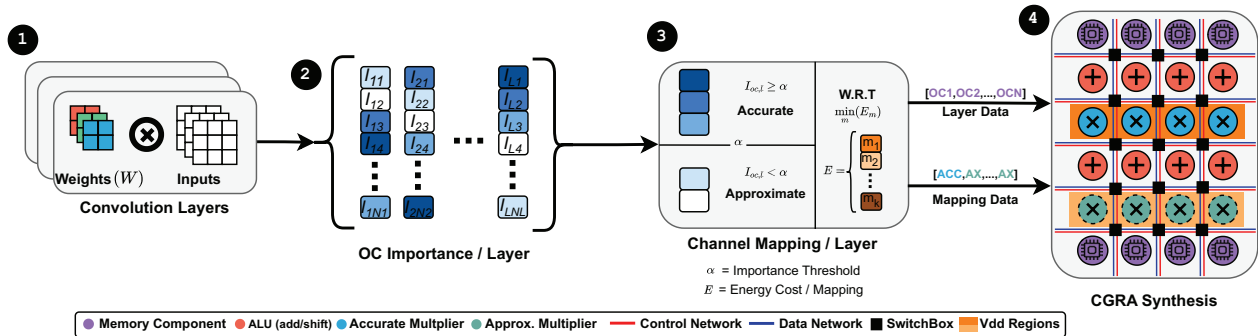


Fig. 4: Overview of the proposed DNN mapping flow. For each conv layer, the process begins by extracting the importance score for each output channel. Based on these scores, each channel is assigned to either accurate or approximate units.

The energy cost of each mapping is taken into account and is estimated using the formula described in Equation 2:

$$E_l = \sum_{i=0}^{OC} (x_i E_{acc}(V_{acc}) + (1 - x_i) E_{ax}(V_{ax})) \quad (2)$$

where E_l represents the total energy consumed during the inference of a single layer l , x_i is a Boolean variable indicating the mapping type of an OC, where 1 refers to accurate mapping and 0 to approximate. $E_{acc}(V_{acc})$ and $E_{ax}(V_{ax})$ correspond to the energy costs of a single multiplication performed on Acc and Ax PEs respectively, when operating under the specified voltage conditions V_{acc} and V_{ax} .

Once the mapping is determined, the input features and weights are organized in the external memory of our system as follows: bottom-up for the approximate channels and top-down for the accurate ones, grouped by the vectorization supported by the accelerator configuration (e.g., vectors of size 4, 8, etc.). Channels are scheduled for execution in pairs, ensuring that the maximum utilization of approximate and accurate components is achieved. In cases where the distribution of channels across the two multiplier types is unbalanced, one of the two computation regions of the CGRA remains idle and waits for the processing of this layer to end.

Having generated the mappings and with the operating conditions given, meaning the voltage scales of the different computation regions of the hardware (orange regions on the CGRA fabric presented in Figure 4), the CGRA is programmed ④ using the sorted weight data per layer, as explained on step ③. The mapping constraints, which guide the selection of which channels will be mapped as approximate and which as accurate, are also propagated. After programming the synthesized CGRA, we evaluate its performance, power, and area (PPA) w.r.t. the specific mapping and voltage allocation configuration.

IV. RESULTS & DISCUSSION

A. Experimental Setup

Since we mainly target Edge AI applications, for our analysis, we consider models that are commonly utilized in the computer vision model of MobileNetV2, evaluating the whole execution of the model as well as individual layer configuration, having multiple convolution layers with various dimensions

for sufficient benchmarking. We perform inference on the ImageNet dataset [18], to bring our benchmarking closer to real-world scenarios. We construct the proposed approximate CGRA mapping flow by extending both architecture and compilation flow of the R-Blocks CGRA [11] to accommodate our approximation components as well as the corresponding compiler intrinsics.

Quantile	Clock Cycles	RMSE	Energy Cost	OC map (%)	
				Acc	Ax
1	52.7 M	5.90	8528	0.0	100.0
0.875	49.7 M	6.23	9508	9.0	91.0
0.75	46.1 M	6.00	10488	19.1	80.9
0.5	40.7 M	5.46	12164	47.9	52.1
0.25	46.1 M	5.41	14408	69.0	31.0
0.125	49.6 M	5.62	15388	83.8	16.2
0	52.7 M	0.00	17056	100.0	0.0

TABLE II: MobileNetV2 Performance-Error Analysis

To obtain measurements regarding area, power, and throughput, we implement our CGRA in Verilog HDL. For this experimental evaluation, we have built three different designs: The first is a relatively small scalar architecture that features 4 multipliers (1 accurate, 1 approximate, and 2 for address generation and constant propagation) and 4 ALUs (Scalar). The remaining two are vector architectures featuring two vector lanes with widths of 4 (Vector-4) and 8 (Vector-8), respectively, meaning that 4 and 8 operations can be executed on the same clock cycle on each architecture. More specifically, Vector-4 architecture includes 19 ALUs and multipliers in total, while Vector-8 doubles this number.

The designs were synthesized using Synopsys Design Compiler, in 22nm technology, operating at 400 MHz. The Memories were implemented using real SRAM Macros generated by the GlobalFoundries Memory Compiler. The synthesized netlists underwent functional validation by executing the aforementioned DNN benchmarks in Siemens Questasim. To estimate the overall power consumption, the switching activities of the netlists were extracted during the post-synthesis simulations, and subsequently, the switching activity was parsed through Synopsys PrimeTime.

B. Approximation-Aware Analysis of MobileNetV2

We continue our exploration by identifying how the user-defined QoS constraint α affects the output error; we evaluate

		REVAMP [7]	CGRA4ML ¹ [8]	X-CGRA ¹ [4]	ICED [10]	Our Work Vec-4 / Vec-8
Arch	Technology	28nm	7nm	45nm	7nm	22nm
	Clock Freq. (MHz)	100	1000	300	108–434	400
	# of PEs	36	192	16	36	19 / 38
	Area (mm ²)	0.125	0.049	0.061	7.18	0.51 / 1.13
Perf	Power (mW)	5	127	3.2	121.3	11.6 / 29.6
	GOPs	0.9	191.6	1.72	8	5.1 / 11.2
	GOPs/W	180	1509	537	66	440 / 378

TABLE III: Comparison with State-of-the-Art Designs. Performance metrics are reported from the respective publications.

¹ All works measure GOPs and GOPs/W w.r.t. their applications, except CGRA4ML, which reports peak GOPs.

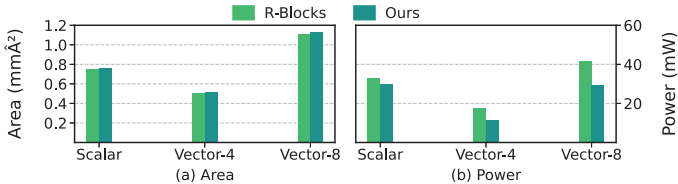


Fig. 5: Area and power comparison of the examined architectures before (R-Blocks) and after DRUM Multiplier Integration and Voltage Scaling.

different quantile thresholds of the Importance Factors for each output channel. This approach allows us to evaluate various partition strategies between accurate and approximate components, assessing their effect on computational accuracy and efficiency. Table II shows a complete analysis of the given architecture, using the Vec-8 configuration and DRUM7 as the selected approximate multiplier. As shown, the error propagation varies between different quantiles of importance where each quantile represents a statistical breakpoint in the distribution of layer importance (e.g., 0.25 = lower-importance thresholds, 0.75 = higher-importance thresholds), also providing energy trade-offs between mapping configurations. Finally, we observe a linear relationship between the Acc-Ax mapping distribution and model performance, with the 0.5 quantile (nearly perfect split) representing the median value, indicating better performance.

C. Comparison with Baseline

To assess the effectiveness of our methodology in terms of area and power efficiency, we perform a comparative study between scalar & vectorized produced CGRA and its iso-resource R-Blocks architecture. Figure 5 demonstrates the area measurements of the same scalar, Vector-4, and Vector-8 architectures tested on both CGRAs. As expected, the scalar architecture exhibits the largest area overhead relative to the number of processing elements (PEs) included in each design. This is attributed to the increased number of Instruction Decode PEs needed to fetch scalar instructions to individual PEs. We can see that by utilizing a vectorized configuration, the hardware cost of multiple ID PEs wears off. It is worth noting that the insertion of level shifters between different voltage domains has a very small impact on area, resulting in less than a 2% increase. This tiny increase shows the impact of voltage scale controllers over the approximate multiplier units, justifying the choice of only choosing two voltage regions for our implementation. Furthermore, between the two vectorized

architectures, we observe an almost linear increment in area, which aligns with the scaling of the computational resources.

Power results follow a similar trend. Despite having fewer computational resources, the scalar architecture is the least power-efficient, reinforcing the advantages of vectorized CGRAs. As shown in Fig. 5b, incorporating DRUM multipliers and voltage islands reduces power by 32.6% in Vector-4 and 29.3% in Vector-8. In contrast, the smallest design gains only a 6% reduction because it includes just one approximate multiplier, limiting the achievable savings.

D. Comparison with State-of-the-Art

Table III offers a high-level comparison with data reported in the papers of state-of-the-art CGRA platforms. For our approach, we include both vectorized architectures to better emphasize the trade-offs between throughput and energy efficiency. We note that X-CGRA [4] and CGRA4ML [8] do not consider the impact of the memory components in their measurements, which can significantly affect both area and power consumption. If we perform the same analysis and exclude memory impact in the total CGRA power (which is about the 25% in both Vec4 and Vec8 architectures) the reported GOPs/W will be 629 and 554 for Vec4 and Vec8 architectures, respectively.

In contrast to this analysis, our evaluation includes memory components, which, on average, account for 35% of the total cell area and contribute approximately 30% to the overall power consumption. As shown, we generate competitive CGRA architectures w.r.t. GOPs and GOPs/W metrics while offering design freedom at the operational level, allowing for the final CGRA to have a different number of heterogeneous PEs.

V. CONCLUSIONS

In this work, an end-to-end synthesis and mapping framework for approximate CGRA Edge AI processors, that selectively assigns computations to approximate multipliers, optimizing energy efficiency while maintaining accuracy. We provided a DSE targeting SotA approximate units, cherry picking optimal approximate units for our CGRA DNN-based accelerator, and analyzed our proposed framework on MobileNetV2 model, showcasing that it can deliver up to 440 GOPs/W with minimal output error, outperforming several state-of-the-art CGRA architectures and frameworks both in terms of throughput and energy efficiency.

REFERENCES

- [1] X. Wang, Z. Tang, J. Guo, T. Meng, C. Wang, T. Wang, and W. Jia, "Empowering edge intelligence: A comprehensive survey on on-device ai models," *ACM Comput. Surv.*, Mar. 2025, just Accepted. [Online]. Available: <https://doi.org/10.1145/3724420>
- [2] K. Seshadri, B. Akin, J. Laudon, R. Narayanaswami, and A. Yazdanbakhsh, "An evaluation of edge tpu accelerators for convolutional neural networks," 2022. [Online]. Available: <https://arxiv.org/abs/2102.10423>
- [3] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE Press, 2021, p. 769–774. [Online]. Available: <https://doi.org/10.1109/DAC18074.2021.9586216>
- [4] O. Akbari, M. Kamal, A. Afzali-Kusha, M. Pedram, and M. Shafique, "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, Oct. 2020. [Online]. Available: <https://ieeexplore.ieee.org/document/8815818/>
- [5] M. Brandalero, L. Carro, A. C. S. Beck, and M. Shafique, "Approximate on-the-fly coarse-grained reconfigurable acceleration for general-purpose applications," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: <https://doi.org/10.1145/3195970.3195993>
- [6] 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.
- [7] T. K. Bandara, D. Wijerathne, T. Mitra, and L.-S. Peh, "Revamp: a systematic framework for heterogeneous cgra realization," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 918–932. [Online]. Available: <https://doi.org/10.1145/3503222.3507772>
- [8] G. Abarajithan, Z. Ma, Z. Li, S. Koparkar, R. Munasinghe, F. Restuccia, and R. Kastner, "Cgra4ml: A framework to implement modern neural networks for scientific edge computing," 2024. [Online]. Available: <https://arxiv.org/abs/2408.15561>
- [9] Y. Luo, C. Tan, N. B. Agostini, A. Li, A. Tumeo, N. Dave, and T. Geng, "MI-cgra: An integrated compilation framework to enable efficient machine learning acceleration on cgras," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.
- [10] C. Tan, M. Jiang, D. Patil, Y. Ou, Z. Li, L. Ju, T. Mitra, H. Park, A. Tumeo, and J. Zhang, "Iced: An integrated cgra framework enabling dvfs-aware acceleration," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 1338–1352.
- [11] B. De Bruin, K. Vadivel, M. Wijtvliet, P. Jääskeläinen, and H. Corporaal, "R-Blocks: an Energy-Efficient, Flexible, and Programmable CGRA," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 17, no. 2, pp. 1–34, Jun. 2024. [Online]. Available: <https://dl.acm.org/doi/10.1145/3656642>
- [12] M. Wijtvliet, L. Waeijen, and H. Corporaal, "Coarse grained reconfigurable architectures in the past 25 years: Overview and classification," in *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, 2016, pp. 235–244.
- [13] Z. Ebrahimi and A. Kumar, "GREEN: An Approximate SIMD/MIMD CGRA for Energy-Efficient Processing at the Edge," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 10, pp. 2874–2887, Oct. 2024. [Online]. Available: <https://ieeexplore.ieee.org/document/10488043/>
- [14] 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.
- [15] S. Hashemi, R. I. Bahar, and S. Reda, "DRUM: A Dynamic Range Unbiased Multiplier for approximate applications," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Austin, TX, USA: IEEE, Nov. 2015, pp. 418–425. [Online]. Available: <http://ieeexplore.ieee.org/document/7372600/>
- [16] V. Leon, K. Asimakopoulos, S. Xydis, D. Soudris, and K. Pekmestzi, "Cooperative arithmetic-aware approximation techniques for energy-efficient multipliers," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: <https://doi.org/10.1145/3316781.3317793>
- [17] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, pp. 258–261.
- [18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.
- [19] S. A. Razavi, M. S. Zamani, and K. Bazargan, "A tileable switch module architecture for homogeneous 3d fpgas," in *2009 IEEE International Conference on 3D System Integration*, 2009, pp. 1–4.
- [20] H. Corporaal and R. Lamberts, "Tta processor synthesis," in *First Annual Conf. of ASCI*. Citeseer, 1995, pp. 18–27.
- [21] K. Vadivel, B. De Bruin, R. Jordans, H. Corporaal, and P. Jaaskelainen, "Prebypass: Software Register File Bypassing for Reduced Interconnection Architectures," in *2022 25th Euromicro Conference on Digital System Design (DSD)*. Maspalomas, Spain: IEEE, Aug. 2022, pp. 157–164. [Online]. Available: <https://ieeexplore.ieee.org/document/9996848/>
- [22] M. R. Kakoei and L. Benini, "Fine-grained power and body-bias control for near-threshold deep sub-micron cmos circuits," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 2, pp. 131–140, 2011.
- [23] AMD-Xilinx, "Brevitas," <https://github.com/Xilinx/brevitas>.
- [24] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance Estimation for Neural Network Pruning," Jun. 2019, arXiv:1906.10771 [cs]. [Online]. Available: <http://arxiv.org/abs/1906.10771>