AURORA: Automated Refinement of Coarse-Grained Reconfigurable Accelerators

Cheng Tan, Chenhao Xie, Ang Li, Kevin J. Barker, Antonino Tumeo
Pacific Northwest National Laboratory, Richland, WA, USA
{cheng.tan, chenhao.xie, ang.li, kevin.barker, antonino.tumeo}@pnnl.gov

***Abstract*—Coarse-grained reconfigurable arrays (CGRAs), loosely defined as arrays of functional units interconnected through a network-on-chip (NoC), provide higher flexibility than domain-specific ASIC accelerators while offering increased hardware efficiency with respect to fine-grained reconfigurable devices, such as Field Programmable Gate Arrays (FPGAs). Unfortunately, designing a CGRA for a specific application domain involves enormous software/hardware engineering effort (e.g., designing the CGRA, map operations onto the CGRA, etc) and requires the exploration on a large design space (e.g., applying appropriate loop transformation on each application, specializing the reconfigurable processing elements of the CGRA, refining the network topology, deciding the size of the data memory, etc). In this paper, we propose AURORA[*] – a hardware/software co-design framework to automatically synthesize optimal CGRA given a set of applications of interest.**

***Index Terms*—CGRA, domain-specific reconfigurable accelerator, software/hardware codesign**

I. Introduction

With the waning of the benefits of device scaling, domain-specific accelerators have become the only viable approach to keep increasing the performance of computing systems in the same, or stricter, power, and area envelopes [7, 29]. However, even with further increases in the number and types, fixed domain-specific accelerators remain feasible only for the most ubiquitous computational patterns.

Reconfigurable architectures are today experiencing a renewed interest in their ability to provide specialization without sacrificing the capability to adapt to disparate workloads. In particular, coarse-grained reconfigurable arrays (CGRAs), which present a variety of coarse functional units interconnected with a network-on-chip (NoC), achieve higher flexibility than domain-specific accelerators while offering increased hardware efficiency with respect to fine-grained reconfigurable devices, such as Field Programmable Gate Arrays (FPGAs). The dataflow execution models, quicker reconfiguration times than FPGAs, and high-performance specialized functional units make CGRAs an appealing target for many emerging application domains, such as multimedia [24, 26], high-performance computing [6, 22], machine learning [14, 33], in-memory computing [9, 10], etc.

While CGRAs provide enhanced flexibility, the selection of the functional units, memory components, communication architecture of the reconfigurable substrate, and the related operation mapping process, still are critical in meeting performance, power, and area constraints dictated by the application domain. For example, the SambaNova [1] design (derived from Plasticine [28]) employs single-instruction multiple-data (SIMD) like units to better match high-performance computing workloads. Google's TPU1 [14] leverages a systolic array [18] with a large number of Processing Elements (PEs) in a mesh topology (i.e., 128x128) to accelerate deep learning. HyCUBE [16] provides single-cycle data delivery between remote PEs to improve the power-efficiency for embedded applications. Dnestmap [15] trades-off control memory size for throughput to meet the stringent area constraint. UECGRA [8] supports fine-grain DVFS at each PE to efficiently accelerate irregular loops. The identification of suitable hardware and software configurations involves exploring a large design space and solving multiple NP-complete problems (e.g., specializing the functional units of the CGRA, refining the network topology, deciding the size of the data memory/buffer, identify the appropriate compiler transformations, such trading off task-level and instruction level parallelism in loops, etc). Performing such a process manually is difficult and time-consuming, and needs to be effectively automated.

Several recent works have started to investigate the automation of the design space exploration (DSE) and the generation of CGRAs depending on the domain. However, they either concentrate on the compiler optimizations, on the mapping process, on the specialization of the functional units, or on the specialization of the communication architecture. They usually do not combine all the elements of the design together or, if they do, they only define greedy heuristics for the exploration, limiting the search only to promising solutions in the neighborhood of the starting design. In this paper, we propose AURORA – a hardware/software co-design framework that performs *AU*tomated *R*efinement of c*O*arse-grained *R*econfigurable *A*ccelerators, generating optimized CGRAs given a set of applications of interest. AURORA explores a rich, multi-level, design space comprising functional units, memory components, communication elements, and their architecture, and compiler optimizations. The specific contributions of this paper are as follows:

- We define a generic architecture template that can construct any type of specialized CGRA;
- We propose a novel hardware/software co-design framework that automatically explores the CGRA design space and generates the optimal design based on the user-defined target and area/power budget;
- We develop an architecture-aware simulated annealing algorithm that facilitates the exploration of the combined design space.

[*]This work was supported by SO(DA)2 project under PNNL's Data-Model Convergence (DMC) Laboratory-Directed RD Initiative.

Prio Art Frameworks	Arbitrary Domain	Factors for Design Space Exploration							Arbitrary Target (Perf/ Power-Efficient/ Area-Efficient)	Constraints (Power/Area)	Hardware Generation
		Loop Tiling	Loop Unrolling	Arbitrary Topology	Control Mem Size	Data Mem Size	FU Type	FU Fusion			
KressArray Xplorer [12]	●	⊖	⊖	◒	⊖	⊖	⊖	⊖	◒	⊖	●
ADRES [23]	⊖	●	⊖	⊖	●	⊖	⊖	⊖	◒	⊖	●
CGRA Express [27]	⊖	⊖	⊖	⊖	●	⊖	⊖	●	⊖	⊖	⊖
EGRA [3]	●	⊖	⊖	⊖	●	●	●	●	◒	⊖	⊖
[30]	●	⊖	⊖	⊖	⊖	⊖	●	⊖	◒	⊖	◒
CGRA-ME [4]	●	⊖	⊖	●	●	⊖	●	⊖	⊖	⊖	●
DSAGEN [35]	●	⊖	●	●	●	●	●	⊖	◒	●	●
RADISH [36]	●	⊖	⊖	⊖	⊖	⊖	⊖	●	◒	⊖	●
Spatial [17]	●	●	●	⊖	⊖	●	⊖	⊖	◒	⊖	●
AURORA	●	●	●	●	●	●	●	●	●	●	●

TABLE I: Comparison with Prior Art – ⊖, ◒, and ● indicate the corresponding feature is not supported, partially supported, and fully supported, respectively. For example, [30] relies on manual effort to design the target CGRA in Verilog/VHDL. In contrast, AURORA can automatically generate the synthesizable Verilog of the target CGRA.

The remainder of this paper is organized as follows. Section II briefly summarizes the related works in literature. Section III introduces the generic architecture template. Section IV details the automated design space exploration. Experimental results are demonstrated in Section V. Finally, Section VI concludes the paper.

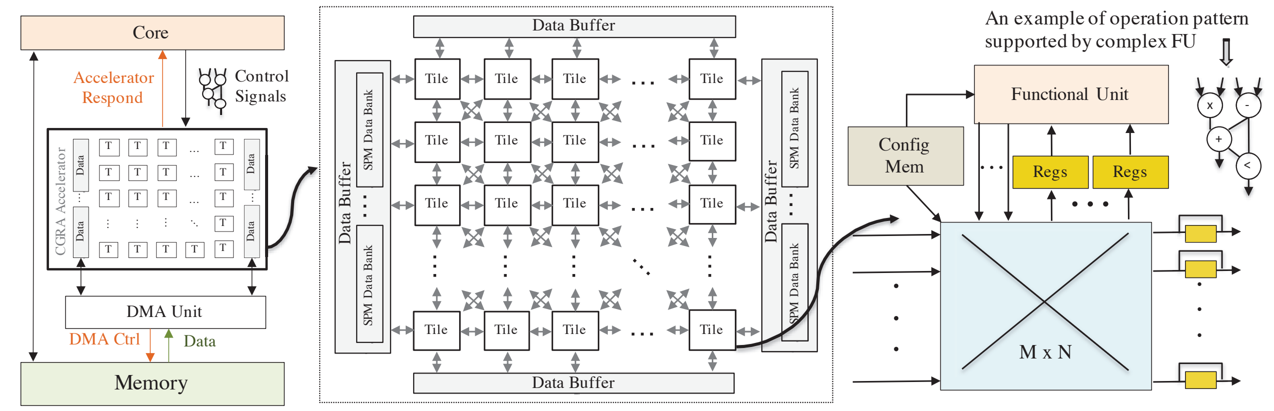
II. Related Work

In Table I, we compare AURORA to the other state-of-the-art coarse-grained reconfigurable accelerator frameworks. All these solutions focus on the offload and acceleration of loops nests in the applications' code.

The KressArray Xplorer [12] is a CAD environment that performs DSE for the KressArray architecture (an array of reconfigurable Data Path Units - rDPUs). While the framework can explore the specific configurations of the rDPUs, it relies on custom high-level language and does not consider loop-level optimizations. [23] proposes a C-based flow that explores the mapping of the digital signal processing kernels on the ADRES architecture (a design where a very long instruction word processor is tightly coupled with a reconfigurable array of functional units). The approach does not deal with hardware generation or optimization. EGRA [3] explores the heterogeneity of processing elements (PEs) to maximize performance on a given benchmark suite. CGRA-ME [4] provides parameterized CGRA models and enables kernel mapping on the custom designs, but it does not automate the design space exploration process. DSAGEN [35] enables compiler-based design space exploration of configurable accelerators for specific application domains. However, the design space exploration approach used only explores hardware modifications in the neighborhood of the original design. CGRA-ME and DSAGEN provide a smaller design space than AURORA: at the software level, they do not consider loop optimizations (e.g, loop tiling), at the hardware level they do not consider operation fusion in PEs (they only add or remove pre-defined basic, ALU-like, PEs). In contrast, RADISH [36] employs a genetic algorithm with a compiler-in-the-loop to iteratively search and evaluate opportunities for combining PEs. The spatial [17] compiler applies several optimizations (including loop optimizations) to efficiently map applications onto FPGAs and the Plasticine [28] CGRA.

III. Generic Architecture Template

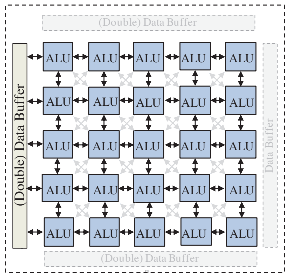
Our approach to DSE of CGRAs starts by defining a generic architecture template that can be configured as any of the common reconfigurable accelerator designs. As shown in Figure 1, the generic CGRA template is composed of a set of modular tiles interconnected via a king mesh network and four scratchpad data buffers. All the basic components in the template are highly modular and parametric. The modular tile is composed of a functional unit (FU), a control memory, a set of registers, and a crossbar switch. A set of tiles is connected to a scratchpad data buffer. A FU in a tile can be basic or complex. The basic FUs support any operation that can be represented in the LLVM IR. The complex FUs support compound operation patterns (i.e., multiple operations chained together) in a single cycle, which may lead to a longer critical path but can significantly shrink the recurrence cycle and improve the overall throughput of the application kernel. The control signals are loaded from the configuration memory in every cycle. The size of the configuration memory is parameterizable and can affect the number of operations that are dynamically supported during execution. For example, the systolic array [18] is a CGRA composed of dedicated PEs (multiplier–accumulators, i.e., MAC units), which only provides multiplication-accumulation operation and can be configured by one or two control signals. At the opposite, the number of control signals in the configuration memory of a conventional CGRA, which activates a signal per cycle, determines the initiation interval. The number of ports of the crossbar switch is a parameter, thus allowing to build arbitrary topologies, as illustrated in Figure 2.



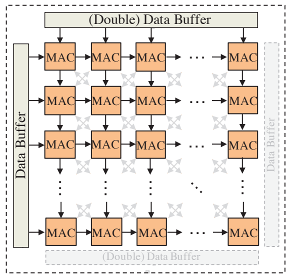
(a) A typical accelerator system. (b) AURORA generic architecture template. (c) Parameterizable tile architecture.

Fig. 1: AURORA generic architecture template – Note that the template is not a target design but providing a design space for AURORA to explore and eventually generate a optimized accelerator based on the given workloads.

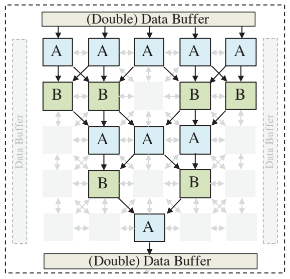
Finally, the size of the data buffer also is a parameter. In particular, it directly depends on the size of the loop tiles, determined by the input data and the tiling factor the offloaded loops of the application kernel. The DSE process needs to explore all these parameters to identify the optimal configuration.



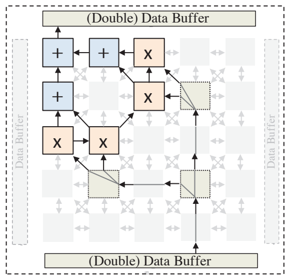
(a) A traditional CGRA.



(b) A systolic-array accelerator.



(c) A CCA-like accelerator.



(d) A MAERI-like accelerator.

Fig. 2: Examples of different CGRA-like accelerators that can be generated based on the AURORA architecture template – A and B indicate two different types of functional units embedded inside the tiles. We show two typical and small designs for CCA-like and MAERI-like accelerators, respectively. Both of them can involve more tiles if there is no stringent area or power budget. Note that the gray components will not be synthesized/generated as a part of the final design/Verilog.

Figure 2 illustrates state-of-the-art spatial reconfigurable accelerators that can be constructed from our generic CGRA architecture template. Specifically, we can specialize the template with simple ALUs and a mesh network to construct a traditional CGRA. By implementing FUs with MAC units and by limiting the number of ports of the crossbar switch, we can describe a systolic array. We can also build a design like the configurable compute accelerator (CCA) [5], which consists of arrays of functional units that implement many common dataflow subgraphs, or MAERI [19], which is designed as a collection of multiply and adder engines, each augmented with tiny configurable switches that can be configured to implement rich communication networks (distribution and reduction trees) to support different kinds of dataflows. We can build CCA-like and MAERI-like accelerators by selecting specialized FUs and implementing the specific network topology. We can build a specific network topology by eliminating certain tiles (i.e., the inport directly connects to the outport) or FUs (i.e., the tiles act like switches).

During DSE, AURORA first tries to identify the tile allocation that maximizes the kernel parallelism, and then iteratively refines the configuration of each component (e.g., tiles, FUs, functionalities of FUs, links between switches, etc) to achieve the most optimized (e.g., in terms of performance, power-efficiency, area-efficiency, etc) design given a set of applications of interest.

IV. AURORA Framework

Figure 3 shows an overview of AURORA. The framework contains three main components – high-level loop transformation, low-level DFG (data-flow graph) manipulation, and accelerator-related mutation. Each component explores different optimization factors and design features. Correspondingly, Figure 4 demonstrates a simplified walk-through example about how a coarse-grained reconfigurable accelerator is synthesized and optimized by AURORA given a specific workload.

A. High-Level Loop Transformation

Given a set of application kernels of interest, typical approaches to enable acceleration on CGRAs rely on the appli-

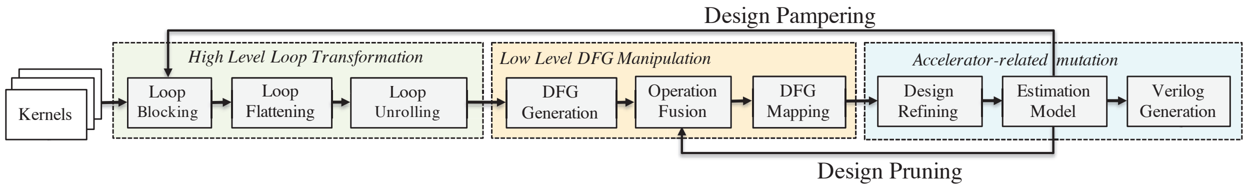
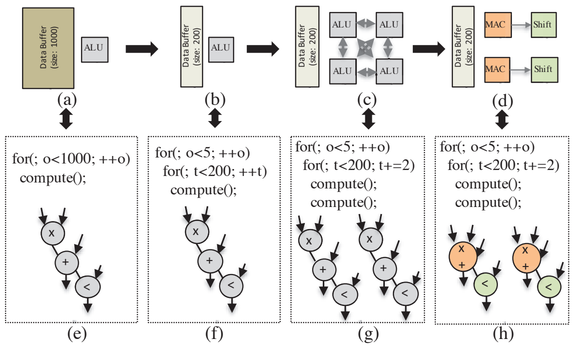


Fig. 3: Overview of AURORA framework – The framework is implemented in LLVM infrastructure [21].



Data Buffer (size: 1000)
ALU
Data Buffer (size: 200)
ALU
MAC
Shift
(a)
(b)
(c)
(d)
for(; o<1000; ++o) compute();
for(; o<5; ++o) for(; t<200; ++t) compute();
for(; o<5; ++o) for(; t<200; t+=2) compute(); compute();
(e)
(f)
(g)
(h)

Fig. 4: Simplified walk-through example of AURORA workflow – For clarity, we simplify the DFG by eliminating the control-flow. Loop flattening is not applied, because there is no nested loop in the original loop.

cation of high-level loop transformations to extract parallelism and data locality. Typical transformations include:

Loop Blocking – Loop blocking (i.e., loop tiling) is widely used to improve the performance by overlapping computation and communication (e.g., adapt to the size of the data buffer), recognizing data locality (e.g., cache-friendly), etc. Loop blocking indicates the amount of data need to be transferred before the accelerator is invoked. Therefore, appropriate loop tiling factor (Figure 4e-4f) should adapt to the memory bandwidth and the data buffer size of the accelerator (Figure 4a-4b).

Loop Flattening – To avoid multiple invocations of the acceleration for the innermost loop, loop flattening flattens the nested loops into a single-nested loop to enable acceleration on the entire tiled loop body by a single invocation [20].

Loop Unrolling – Loop unrolling determines the instruction level parallelism inside each iteration of the loop (Figure 4c), which should match the computing resource capability of the accelerator (Figure 4g).

B. Low-Level DFG Manipulation

DFG Generation – After applying the high-level loop transformation, we can generate the data-flow graph (DFG). Note that in AURORA control dependency is represented as data dependency through partial predication [11].

Operation Fusion – The speedup of an application kernel accelerated by a reconfigurable accelerator with sufficient computing resources is limited by the inter-iteration data dependency (i.e., recurrence cycle) in the DFG. The low-level DFG manipulation attempts to fuse operations [3, 32] on the recurrence cycle based on the functionality provided by the accelerator's functional units to maximize the acceleration throughput. Specifically, AURORA explores the operations that each functional unit should provide. If the target accelerator is equipped with a complex functional unit(Figure 4d), the corresponding operations in the DFG can be fused into a single one (Figure 4h) and mapped onto it for acceleration.

DFG Mapping – A heuristic mapping algorithm [31] maps the fused DFG onto a given CGRA design. Basically, the algorithm maps each operation in the sorted DFG onto an available functional unit in the Modulo Routing Resource Graph (MRRG) with minimum initiation interval (II). The II value is incrementally increased until a valid mapping between the DFG and the MRRG is found. The data dependency represented as data communication between functional units is routed using Dijkstra's algorithm.

C. Accelerator-related Mutation

Automated Design Refinement – We developed an optimization algorithm based on simulated annealing to perform the design space exploration along all the previously listed dimensions. Simulated annealing is a metaheuristic that takes its name from the annealing (heating and controlled cooling) in metallurgy. The approach involves accepting worse solutions with respect to the optimization objectives with a probability proportional to the progressively lowering temperature. Accepting worse solutions allows a better exploration of the design space, potentially overcoming local optima. The design space exploration starts from the most basic accelerator design (i.e., single tile design as shown in Figure 4a) and can progressively add tiles, change the hardware parameters (Figure 4d) and modify the compiler optimization factors (i.e., loop tiling in Figure 4f and unrolling factors in Figure 4g) on the input applications, until the objective converges. The initial candidates for the functional units are identified from the analysis of the input applications: only the operations in the input applications are available, and each functional unit can, at maximum, support these as operation chains on the recurrence cycles. The mutation operator in AURORA can then probabilistically change the types of functional units on each tile, the number of inports and outports of each crossbar switch, and the size of the data buffer and the configuration memory) without violating the given power/area budget.

AURORA can target different optimization objectives, such as performance, area-efficiency, power-efficiency, etc. If, after applying the low-level DFG manipulation, the quality of the solution improves, AURORA adds tiles to the target accelerator design and selects the best combination (i.e., the one that provides the highest quality) for the loop optimization

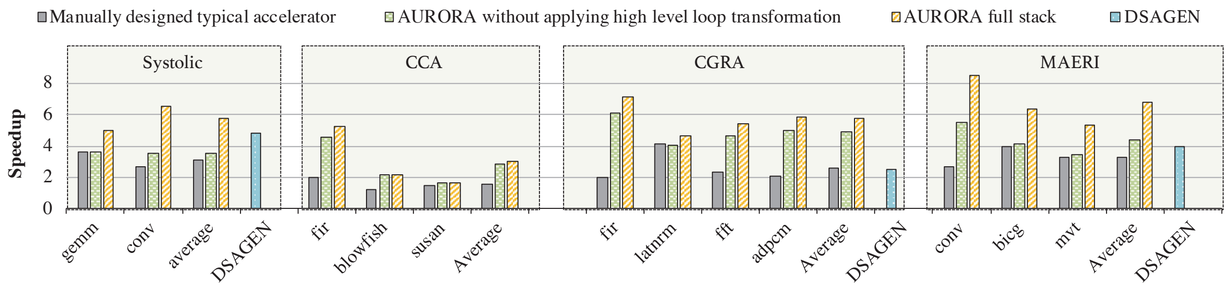


Fig. 5: Performance improvement of the optimized accelerators generated from AURORA compared against the manually designed baselines across different collections of workloads – The speedup of DSAGEN is derived from [35] and the softbrain [25] accelerator is treated as CGRA in our evaluation.

parameters. Because the exploration is performed through simulated annealing, there is some probability that a worse solution is accepted. It then repeats this procedure until the quality of the solution does not improve, and the temperature reaches the threshold.

The execution time of the AURORA DSE heavily depends on the efficiency of the mapping algorithm, which is related to the size of both the DFG and the reconfigurable accelerator. Our evaluation shows that the entire procedure can complete within a few minutes when the size of the target accelerator does not exceed 6×6 and the DFG contains less than 200 operations. AURORA can also significantly shrink the design space and quickly generate an optimal design by providing a nontrivial initial design point (e.g., start from 4×4 rather than the most basic 1×1 accelerator design) with stringent power/area budget.

Estimation Model – To quickly evaluate a design point for the target reconfigurable accelerator during the DSE, we build an analytical regression model for the performance, power, area, and timing evaluation. We synthesize all the basic components of the architecture template with OpenRoad [2] flow (leveraging 45nm NanGate standard-cell library) and collect the corresponding statistics. The functional unit, configuration memory, and crossbar contribute to the power/area overhead of a tile. The performance speedup is calculated as follows:

$$\begin{aligned} speedup &= T_{base} \div T_{acc} \\ T_{acc} &= \begin{cases} T_{comm}, & if\ T_{comm} > T_{comp} \\ T_{comp}, & otherwise \end{cases} \\ T_{comp} &= (II \times \#iter) \div timing \end{aligned} \tag{1}$$

T_{base} is evaluated by running the original C codes (without applying optimizations) on Intel Xeon Gold 6126 @2.60GHz. T_{comm} indicates the time cost for data delivery required by each invocation of the acceleration, which depends on the data size and the time to transfer each data tile. Note that in our architecture template we employ a Direct Memory Access (DMA) unit, and we model the use of double buffering in AURORA to improve throughput. II is the initiation interval achieved by DFG mapping. $iter$ is the number of outermost loop iterations executed at each acceleration invocation. $timing$ is the longest critical path in the mutated architecture template after applying the design refinement.

We have empirically validated the regression model for area and power, which shows negligible error when comparing metrics obtained by synthesizing hand-designed systolic arrays with metrics computed with the model. Metrics for the data memory are derived from Cacti6.5 [34].

Verilog Generation – Once the DSE is completed, the architectural description of the refined reconfigurable accelerator is generated. AURORA employs OpenCGRA [13, 31], which takes the generated architectural description as input, to generate the corresponding synthesizable Verilog.

V. Experimental Evaluation

A. Environment Setup

AURORA is implemented in the LLVM infrastructure [21]. We select a set of workloads from different benchmark suites (Table II) to drive our evaluation. We identify four different baseline domain-specific coarse-grained reconfigurable accelerators (Figure 2), each targeting a specific workload collection. Specifically, we select a systolic array to accelerate deep learning kernels (i.e., `gemm` and `conv`). CCA targets the embedded domain, represented by the `fir`, `blowfish`, and `susan` kernels. Similarly, other representative embedded workloads (i.e., `fir`, `latnrm`, `fft`, and `adpcm`) are typically used to evaluate CGRAs [16]. The work in [35] evaluates linear-algebra workloads (i.e., `conv`, `bicg`, and `mvt`) on MAERI. AURORA generates the optimized accelerators considering each collection of workloads. These are then compared against the baseline accelerators in terms of performance, area-efficiency, and power-efficiency. We also compare AURORA with DSAGEN [35], since it covers more DSE factors than the other state-of-the-art works.

B. Evaluation Results

Figure 5 shows the speedup of the optimized accelerators generated by AURORA and the manually designed baseline accelerators across the different sets of workloads. The speedup is calculated as described in Section IV-C. Comparing with the baseline accelerators, the optimized accelerators generated by AURORA achieve higher speedup in most of the cases even without applying the high-level loop transformation. This demonstrates the benefits provided by operation fusion. In the case of the systolic array baseline design, AURORA's low-level DFG manipulation does not provide

Application	Benchmark	#Opt	Data	Domain	Target Accelerator
gemm	polybench	28	64^3	DL	systolic array
conv	polybench	30	32^2	DL	systolic&MAERI
fir	UTDSP	12	64	DSP	CCA&CGRA
blowfish	CBench	33	256	security	CCA
susan	CBench	28	600×450	auto	CCA
latnrm	UTDSP	25	32	DSP	CGRA
fft	UTDSP	31	1024	DSP	CGRA
adpcm	Mibench	55	1000	telecom	CGRA
bicg	polybench	50	32^2	LA	MAERI
mvt	polybench	28	32^2	LA	MAERI

TABLE II: Target workloads – DL and LA indicate deep learning and linear-algebra, respectively. #Opt shows the original number of operations without applying high-level loop transformation and low-level DFG manipulation. This number can easily surpass hundred when the loops are flattened and unrolled.

significant improvements for the automatically generated solution. In fact, the FUs of the baseline systolic array are multiply-accumulate (MAC) units, corresponding to complex FUs with chained operations in the AURORA flow. AURORA's high-level loop transformations, instead, do not impact much on the `susan` benchmark due to the loop-carried dependency. The designs identified by AURORA also provide higher speedups than the solutions found by DSAGEN when optimizing a systolic array, a CGRA, and a MAERI-like design. The reason is that AURORA also considers loop blocking and operation fusion as DSE parameters. We highlight that the speedups provided by DSAGEN generated designs are derived from [35] (softbrain [25] is considered as a CGRA).

AURORA can optimize for different objective functions. Figure 6 shows the normalized area-/power-efficiency improvement of the optimized accelerators generated from AURORA with respect to the manually designed baseline accelerators across the different sets of workloads. The accelerators automatically generated by AURORA achieve an area-efficiency up to 2.5× and a power-efficiency up 2.7× higher than the baseline MAERI and CGRA designs, respectively.

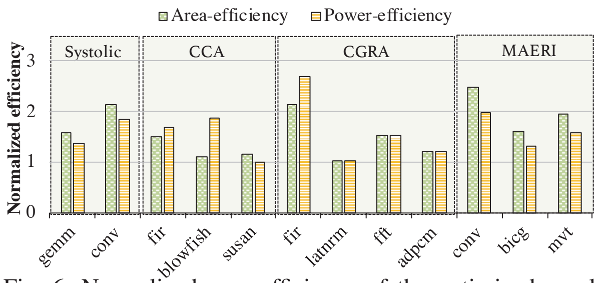


Fig. 6: Normalized area–efficiency of the optimized accelerators generated from AURORA with respect to the manually designed baselines across different collections of workloads.

VI. Conclusion

This paper proposes AURORA, a hardware/software co-design framework for the automatic synthesis of optimized coarse-grained reconfigurable accelerators starting from a set of kernels of interest. AURORA employs a generic architecture template that allows building any type of specialized CGRA and performs the DSE through an architecture-aware simulated annealing optimization algorithm.

References

[1] SambaNova Systems. https://sambanova.ai.

[2] T. Ajayi et al. Openroad: Toward a self-driving, open-source digital layout implementation tool chain. *Proc. GOMACTECH*, 2019.

[3] G. Ansaloni et al. EGRA: A Coarse Grained Reconfigurable Architectural Template. *TVLSI'10*.

[4] S. A. Chin et al. Cgra-me: A unified framework for cgra modelling and exploration. In *ASAP'17*.

[5] N. Clark et al. Application-specific processing on a general-purpose core via transparent instruction set customization. In *MICRO'04*.

[6] S. Das et al. Rhyme: Redefine hyper cell multicore for accelerating hpc kernels. In *VLSID'16*.

[7] J. Dongarra et al. Numerical algorithms for high-performance computational science. *Philosophical Transactions of the Royal Society A*, 2020.

[8] C. Torng et al. Ultra-elastic cgras for irregular loop specialization. In *HPCA'21*.

[9] A. Farmahini-Farahani et al. Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules. In *HPCA'15*.

[10] M. Gao et al. Hrl: Efficient and flexible reconfigurable logic for near-data processing. In *HPCA'16*.

[11] M. Hamzeh et al. Branch-aware loop mapping on cgras. In *DAC'14*.

[12] R. Hartenstein et al. Kressarray xplorer: A new cad environment to optimize reconfigurable datapath array architectures. In *DAC'20*.

[13] S. Jiang et al. Pymtl3: A python framework for open-source hardware modeling, generation, simulation, and verification. *IEEE Micro'20*.

[14] N. P. Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA'17*.

[15] M. Karunaratne et al. Dnestmap: mapping deeply-nested loops on ultra-low power cgras. In *DAC'18*.

[16] M. Karunaratne et al. Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect. In *DAC'17*.

[17] D. Koeplinger et al. Spatial: A language and compiler for application accelerators. In *PLDI'18*.

[18] H. T. Kung. Why systolic architectures? *Computer*, 1982.

[19] H. Kwon et al. Maeri: Enabling flexible dataflow mapping over dnn accelerators via programmable interconnects. *ASPLOS'18*.

[20] Jongeun et al L. Flattening-based mapping of imperfect loop nests for cgras. In *CODES'14*.

[21] C. Lattner et al. Llvm: A compilation framework for lifelong program analysis & transformation. In *CGO'04*.

[22] K. T. Madhu et al. Compiling hpc kernels for the redefine cgra. In *ICESS'15*.

[23] B. Mei et al. Adres: An architecture with tightly coupled vliw processor and coarse-grained reconfigurable matrix. Springer, 2003.

[24] B. Mei et al. Implementation of a coarse-grained reconfigurable media processor for avc decoder. *Journal of signal processing systems*, 2008.

[25] T. Nowatzki et al. Stream-dataflow acceleration. In *ISCA'17*.

[26] H. Park et al. Polymorphic pipeline array: a flexible multicore accelerator with virtualized execution for mobile multimedia applications. In *MICRO'09*.

[27] Y. Park et al. Cgra express: accelerating execution using dynamic operation fusion. In *CASES'09*.

[28] R. Prabhakar et al. Plasticine: A reconfigurable architecture for parallel patterns. In *ISCA'17*.

[29] J. Shalf et al. The future of computing beyond moore's law. *Philosophical Transactions of the Royal Society A*, 2020.

[30] D. Suh et al. Design space exploration and implementation of a high performance and low area coarse grained reconfigurable processor. In *FPT'12*.

[31] C. Tan et al. OpenCGRA: An Open-Source Unified Framework for Modeling, Testing, and Evaluating CGRAs. In *ICCD'20*.

[32] C. Tan et al. Stitch: Fusible heterogeneous accelerators enmeshed with many-core architecture for wearables. In *ISCA'18*.

[33] M. Tanomoto et al. A cgra-based approach for accelerating convolutional neural networks. In *IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, 2015.

[34] S. Thoziyoor et al. CACTI 6.5. https://www.hpl.hp.com/research/cacti/.

[35] J. Weng et al. Dsagen: Synthesizing programmable spatial accelerators. In *ISCA'20*.

[36] M. Willsey et al. Iterative search for reconfigurable accelerator blocks with a compiler in the loop. *TCAD'18*.