Self Adaptive Reconfigurable Arrays (SARA): Learning Flexible GEMM Accelerator Configuration and Mapping-space using ML

Ananda Samajdar, Eric Qin
anandsamajdar@gatech.edu,ecqin@gatech.edu
Georgia Tech

Michael Pellauer
mpellauer@nvidia.com
NVIDIA

Tushar Krishna
tushar@ece.gatech.edu
Georgia Tech

ABSTRACT

This work demonstrates a scalable reconfigurable accelerator (RA) architecture designed to extract maximum performance and energy efficiency for GEMM workloads. We also present a self-adaptive (SA) unit, which runs a learnt model for one-shot configuration optimization in hardware offloading the software stack thus easing the deployment of the proposed design. We evaluate an instance of the proposed methodology with a 32.768 TOPS reference implementation called SAGAR, that can provide the same mapping flexibility as a compute equivalent distributed system while achieving 3.5× more power efficiency and 3.2× higher compute density demonstrated via architectural and post-layout simulation.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

datasets, neural networks, gaze detection, text tagging

1 INTRODUCTION

Custom architecture leverage specialized hardware implementation to achieve high performance and efficiency. However, rigidity due to specialization makes designs obsolete when workloads change. To mitigate this limitation, there has been an increasing interest in developing flexible architectures which have additional components (interconnects, buffers, and configuration registers) to support changing workload requirements.

In all of the prior works on flexible DNN accelerators, however, the onus of finding and setting the best configuration lies on the software stack, typically using a compiler/mapper. This dependence causes a few deployment challenges: (i) An optimization loop with custom cost functions for the given accelerator has to be integrated into compilation frameworks to find the best hardware configurations, without which the flexible design loses utility, (ii) an expensive configuration and mapping search has to be performed at compile-time before scheduling any workload. Usually mapping search in software is performed via exhaustive, heuristic or optimization algorithm-based approaches which take about a few seconds to hours even with sophisticated ML assisted frameworks like autoTVM [2]. (iii) the search-time overhead also eliminates opportunities for deploying such flexible accelerators for domains/applications with soft or hard-real time inference targets.

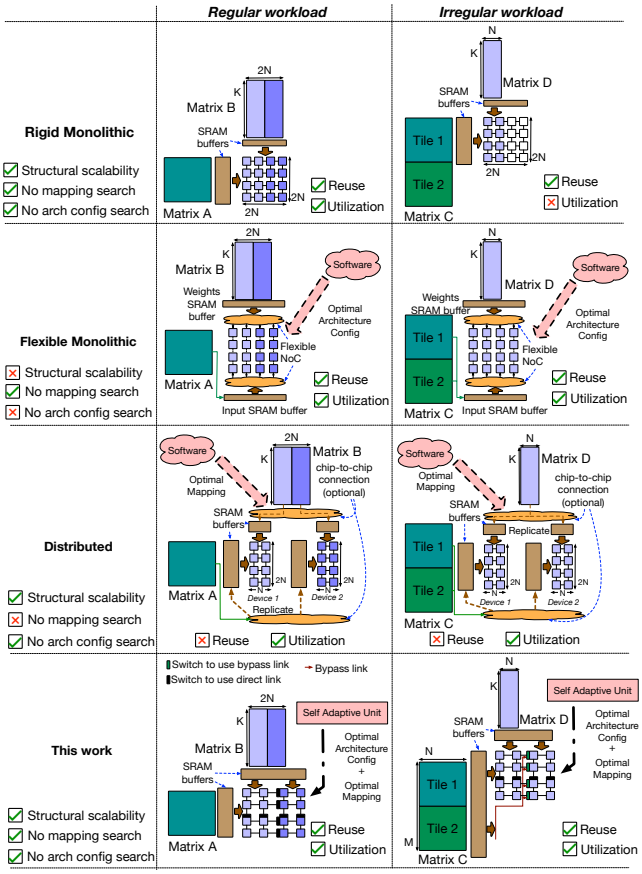

Figure 1: Comparison of scalability, utilization, and operand reuse in traditional monolithic and distributed accelerators, and the position of the proposed architecture

In this work, we demonstrate that the mapping and configuration space of a reconfigurable accelerator can be *learnt* by a machine learning (ML) model, which can then be used to query for optimal parameters for any workload at constant time. Dependence on the software stack can be eliminated by incorporating this learnt model into the hardware itself and querying it in runtime. We illustrate this via two contributions. ***First,*** we design a scalable reconfigurable hardware optimized for GEMM workloads called RECONFIGURABLE SYSTOLIC ARRAY (*RSA*). *RSA* is developed upon the intuition that flexible accelerators often need to trade-off utilization, data reuse, and hardware complexity (i.e., scalability). This is illustrated in Figure 1. *Rigid Monolithic* arraysare simple to construct but offer no flexibility leading to high under-utilization for many workloads.

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07.
https://doi.org/10.1145/3489517.3530506

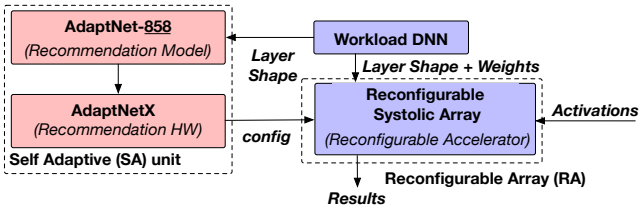

Figure 2: The constitution and interactions of the self adaptive (SA) and reconfigurable array (RA) components to make up the SARA accelerator called *SAGAR* in this work.

Flexible Monolithic arrays provide flexibility via clever use of interconnects and configuration logic, enabling high utilization for a majority of workloads. However, the increased hardware complexity hinders scaling, and the design requires external software support to exploit the benefits of reconfigurability Distributed architectures help address the utilization challenge, since irregular workloads can be tiled on to these smaller arrays. However, this architecture leads to loss in spatial reuse (i.e., direct data-forwarding) that monolithic designs provide, and also requires data replication across the SRAMs of the individual arrays. Data replication leads to a decrease in overall on-chip storage capacity, leading to a loss of temporal reuse due to smaller tiles. Moreover, distributed arrays can exacerbate the mapping search problem *RSA* aims to address the shortcomings of all three design strategies. It is a flexible accelerator capable of supporting mappings that can be realized by monolithic as well as distributed arrays by configuring to variable array dimensions and number of sub-arrays (as depicted later in Figure 3(d)), thereby enhancing both utilization and reuse.

Second, we present a systematic mechanism to cast the architecture configuration as a ML classification problem and discuss considerations for optimal model design, training, and performance of the model at inference. Specifically, we develop a custom ML recommendation system model called ADAPTNET that achieves a recommendation accuracy of 95% on a dataset of 200K GEMM workloads, and on average(GeoMean) 99.93% of the best attainable performance (Oracle). We also design a custom hardware unit to run ADAPTNET called ADAPTNETX. ADAPTNETX enables to get a recommendation response for any query in about 600 cycles which is at least about 6 orders of magnitude faster than software. With ADAPTNETX the configuration lookup using ADAPTNET can be performed at runtime, without involving the software stack.

Together, these two components enable us to develop a new class of accelerators that we call *Self Adaptive Reconfigurable Array (SARA)* (Figure 2). SARA accelerators can self adapt at runtime to optimized configurations for the target workload, without requiring compile-time analysis. We demonstrate an instance of SARA that we name 'Shape Adaptive GEMM AcceleratoR (*SAGAR*) as shown in Figure 2 and evaluate its performance across various configurations. We show that *SAGAR* has 3.2× higher compute density and 3.5× improved power efficiency, over equivalent scaled-out systolic array. The extra flexibility costs <10% in area and 50% in power, compared to an equivalent scaled-up systolic array. Compared to an area normalized state-of-the-art flexible scalable accelerator [13], *SAGAR* incorporates 45% more compute. When comparing compute-equivalent configurations, *SAGAR* consumes 43% less power and 30% less area.

Table 1: Previous accelerator proposals categorized in terms of computation support, and flexibility of hardware and mapping. Accelerators are categorized into various types introduced in Figure 1 viz. Rigid Monolithic (RM), Flexible Monolithic (FM), and Distributed (Dist)

		Mapping Capability		Flexibility			
	Type	Homogenous	Heterogenous	Dataflow	Variable Dimensions	Multi-array Mapping	Self Configurable
Zhang et al. [17]	FM	✓		✓			
Eyeriss [3]	RM	✓					
Alwani et al. [1]	RM		✓				
NeuroCube [9]	Dist		✓			✓	
MAERI [11]	FM	✓	✓		✓		
TPU [8]	RM	✓					
Flexflow [12]	FM	✓		✓			
Tetris [5]	Dist		✓			✓	
Brainwave [4]	Dist		✓			✓	
Simba [16]	Dist		✓			✓	
Tangram [6]	Dist		✓			✓	
Cascades [14]	FM		✓	✓			
Sigma [13]	FM		✓				
Planaria [7]	FM		✓		✓		
***SAGAR* (This work)**		✓	✓	✓	✓	✓	✓

2 MOTIVATION AND RELATED WORK

Table 1 depicts some of the recent architecture proposals and the degree of flexibility in terms mapping various dataflows, capability of reconfiguration in hardware, or performing multi-array mapping. Interestingly when designing an accelerator at scale, we are presented with a contrasting set of requirements. *First*, scalability favours relatively simple layout, which can be easily implemented (eg. systolic arrays). *Second*, achieving high utilization at the face of diverse workloads. Achieving high utilization is a direct consequence of mapping flexibility of the array, which is usually realized by the use of complex interconnect in the traditional proposals. These two requirements contradict each other, as complex interconnects are expensive to implement at scale. An alternative approach adopted in the recent proposals to take a *scaled out* or partitioned approach [15, 16] to attain both scalability and mapping flexibility. However, this approach compromises on energy efficiency by giving up the benefits attainable by exploiting spatio-temporal reuse (Section 1).

3 RECONFIGURABLE ARRAY DESIGN

3.1 Compute Array

We chose systolic-arrays as our base structure due to their simplicity of construction which makes it scalable. Systolic arrays are also highly effective in exploiting the parallelism in GEMM computation and also achieve high energy efficiency by exploiting operand reuse due to inherent nature of data movement. The peer-to-peer links however limit the mapping flexibility of such arrays since any multiply and accumulate (MAC) unit can only work with the operands forwarded by their peers. One possible approach, to improve mapping flexibility, is to augment the MAC units with bypass paths such that operands other than the ones forwarded by its peers can be supplied. Adding such logic to each MAC unit however is impractical due to costs incurred by additional bypass paths and the corresponding reconfiguration overhead, rendering such a design difficult to scale.

Systolic Cells. Our approach to improve the mapping flexibility of a scalable systolic array based design is to construct a structure which we call *systolic-cell*(Figure 3(a)). A *systolic-cell* is constructed by grouping several MAC units and adding multiplexers around the edge of the group such that we can select to accept data either from the peers or some other data from SRAM banks available over bypass links. These can then be arranged together in a 2D grid to create a reconfigurable array as shown in Figure 3(b-c). The dimensions

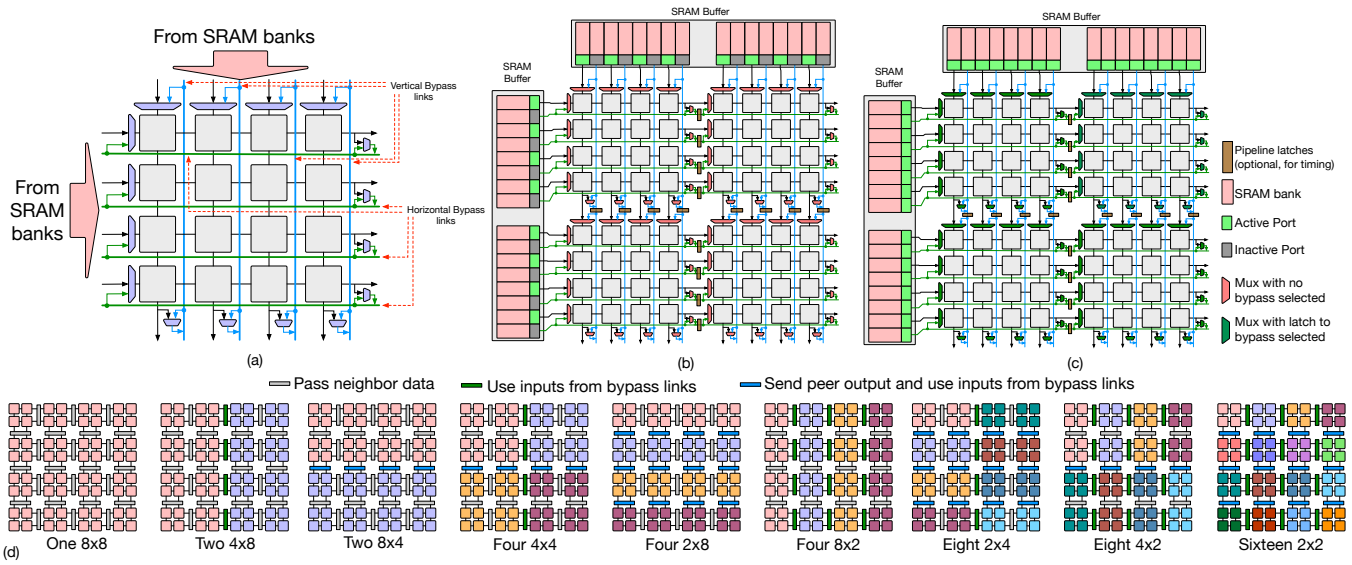

Figure 3: (a) Construction of a 4×4 *systolic-cell* with bypass muxes and bypass links. (b) A 8×8 reconfigurable array operating in scale-up configuration. Each 4×4 *systolic-cell* is connected to its neighbor with the peer-to-peer links as the bypass muxes are turned off. The SRAM ports connected to bypass links are unused. (c) Configuration of bypass muxes to enable the 8×8 reconfigurable array to work as a scaled-out distributed collection of systolic arrays. The bypass muxes are turned on to allow *systolic-cell*s to directly connect to the SRAM ports which are all active. (d) Possible monolithic and distributed configurations possible in the reconfigurable array using 2×2 *systolic-cell*s

of a *systolic-cell* is a design time decision subjected to design budgets. Smaller *systolic-cell*s allow finer grained reconfigurability but increase implementation and configuration cost.

Scale-up and Scale-out using systolic-cells. The reconfigurable array obtained above is capable of operating as a single monolithic array or as a collection of arrays. For example, Figure 3(b) depicts the 8×8 reconfigurable array constructed from 4 4×4 *systolic-cell*s is used as a single array by configuring the muxes to allow data only from the peers. In Figure 3(c) however, the muxes are configured to choose data only from bypass links, making the same array to operate as a scaled out collection of arrays. Several other configurations are also possible to realize using different mux setting. For instance, Figure 3(d) depicts the many configurations a 8×8 reconfigurable array constructed using 2×2 *systolic-cell*s can be realized, which allows efficient mapping of various workloads.

3.2 Other components

Bypass Links. Bypass links are allocated from SRAM banks to the muxes at the edge of each systolic cells to allow for mapping flexibility. Table 2 shows the bandwidth requirements for each GEMM operand for the three possible dataflows in the systolic array and their implication on the bandwidth requirements for the bypass paths. We provide dedicated links for each systolic cells in both directions to eradicate throughput loss from bandwidth limitations.

Scalability via pipelining. Previous studies on wire scalability [10] has shown that a signal can traverse 9mm-11mm in a nanosecond. To further improve scalability and allow higher clock speeds, we add latches in the bypass links. Our evaluations show that 8 4×4 *systolic-cell*s can be bypassed at 1GHz in 28nm (Figure 8(h)). These additional latches only add a few cycles to data fill time and do not affect the computation time of the systolic arrays.

Scratchpad Memory. The scratchpad memory arrangement is similar to the ones in monolithic accelerators with three distinct buffers to store the two operands and one buffer to store the output. The buffers however are multi-ported with each bypass link allocated separate port to minimize bandwidth bottleneck.

Table 2: Bandwidth requirements for the bypass links for various dataflows, contrasted to the requirements of operands (names in parenthesis reflects the corresponding operands in 2D convolutions)

	Operands			Links	
	Input Mat1 (Activations)	Input Mat2 (Filters)	Outputs	Hor. Bypass	Ver. Bypass
Output Stationary	High	High	Low	High (Inputs)	High (Filters)
Weight Stationary	High	Low	High	High (Inputs)	High (Outputs)
Input Stationary	Low	High	High	High (Filters)	High (Outputs)

4 RECOMMENDATION MODEL

Motivation. The reconfigurable array described above can morph to various hardware configurations optimal for each GEMM operation. However, determining the best configuration per layer is costly to be performed at runtime. Conventional reconfigurable arrays rely on compilers for optimization. However, this means that building a robust cost model and optimization pass into existing compilers is critical to enable deployment of the architecture. This requirement often becomes prohibitive for widespread adoption of configurable custom accelerators. Moreover, efficient optimization in software is also an open problem and requires search based approaches which take significant energy and time. To combat this challenge we propose a learning based approach for on-device optimization, which offloads the compiler framework which now only has to perform code translation without concerning about optimization passes.

Architecture Optimization as ML problem. Our solution is to use a ML model to predict the optimal configuration in hardware as workloads arrive. We found that a classification or recommendation

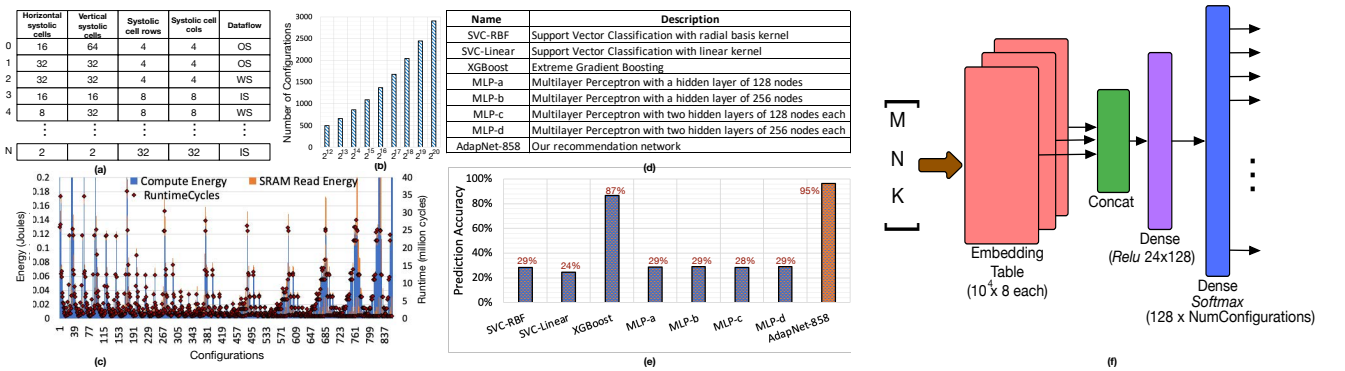

Figure 4: (a) Size of the configuration space wrt number of MAC units for a *systolic-cell* based flexible array (b) Example of configurations predicted by ADAPTNET indexed by category ID (c) Performance and energy consumption by various configurations when running layer 19 of FasterRCNN (d) Chart showing the name and description and name of the various classifiers used in this work (e) The accuracies obtained by classifiers on predicting the architecture parameters for out dataset of RSA configurations with 2^{14} MAC units (f) Architecture of our proposed recommendation network

setting fits well for this task. In this setting the input space to the model comprises of the GEMM operand dimensions M, N, and K. The output space of the model is the set configurations the array can take. Once trained the learnt model is expected to generate the ID of the optimal configuration for the queries workload dimensions. Figure 4(a)shows the configuration space of the proposed reconfigurable array. In Figure 4(b) we show the size of configuration space as the compute capacity of the reconfigurable array is scaled. These number although small relative to the size of design space in accelerators, is quite large such that it takes from a few seconds to a minute on software. Figure 4(c) shows the cost landscape of the various configurations for 2^{14} MAC array for a representative workload. As the figure depicts, the cost space is highly irregular, and the penalty of a sub-optimal configuration is steep. The objectives of the on device optimizer model therefore are (i) Low implementation overhead, (ii) High accuracy, and (iii) Negligible latency.

Dataset Construction and Model Search. For dataset generation, each point is created by sampling M, N, and K values from a uniform distribution of integers in $(0, 10^4]$. For each such workload we searched for the optimal configuration using SCALE-Sim as the cost model. The ID of the optimal configuration is then used as the label. We generated 2.5×10^6 data points to complete the dataset. Next, we used this dataset to train several off-the-shelf classifier models from support vector classifiers (SVC), tree based classifiers (XGBoost), to several multi-layer perceptrons (MLP) listed in Figure 4(d). The test performance of these models is shown in Figure 4(e), where we notice that XGBoost attains a reasonable prediction accuracy depicting that architecture optimization can be learnt. **Recommendation Model**. To further improve the prediction accuracy, we create a custom model called ADAPTNET as shown in Figure 4(f). At inference the model looks up the trained embeddings for each feature and then passes it into a simple two layer MLP classifier. Different variations of ADAPTNET can be constructed for various arrays as the reconfigurable array is scaled. Since these variations only differ in the number of output nodes, we distinguish the variations by adding the number of output nodes as suffix (eg. ADAPTNET-858 is specific for array with 858 configurations).

Model training and Performance. We train the model on the dataset described above, with categorical cross entropy loss and

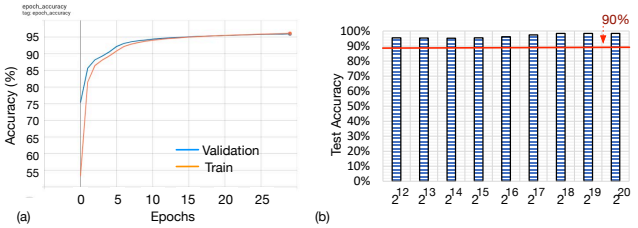

Figure 5: (a) The training and validation accuracies obtained during the training step in ADAPTNET-858 (b) Test accuracies obtained on test sets for ADAPTNETs trained on *systolic-cell* based flexible array with various MAC units

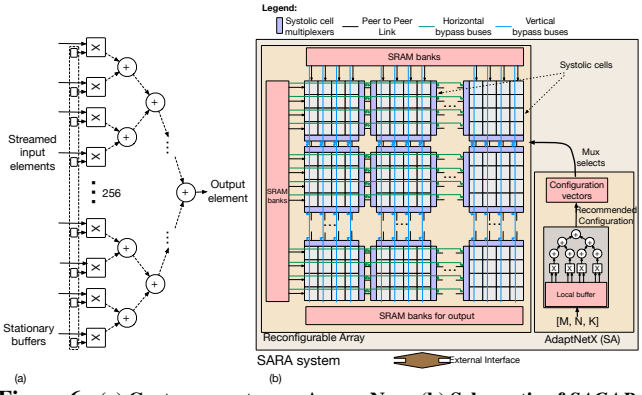

Figure 6: (a) Custom core to run ADAPTNET, (b) Schematic of *SAGAR*, an instance of a SARA accelerator.

ADAM as the optimizer. The training converges in about 10 epochs at about 94% validation accuracy and about 95% test accuracy on a dataset with 200K points (Figure 5(a)). In Figure 5(b) we plot the test performance of various ADAPTNETs as the size of the reconfigurable array is scaled, trained on appropriate datasets. The figure demonstrates that each instance of ADAPTNET achieves >90% test accuracy when trained on datasets for each reconfigurable array.

5 SARA DESIGN

The reconfigurable array described in Section 3 is capable of morphing into scaled-up or scaled-out configurations to simultaneously

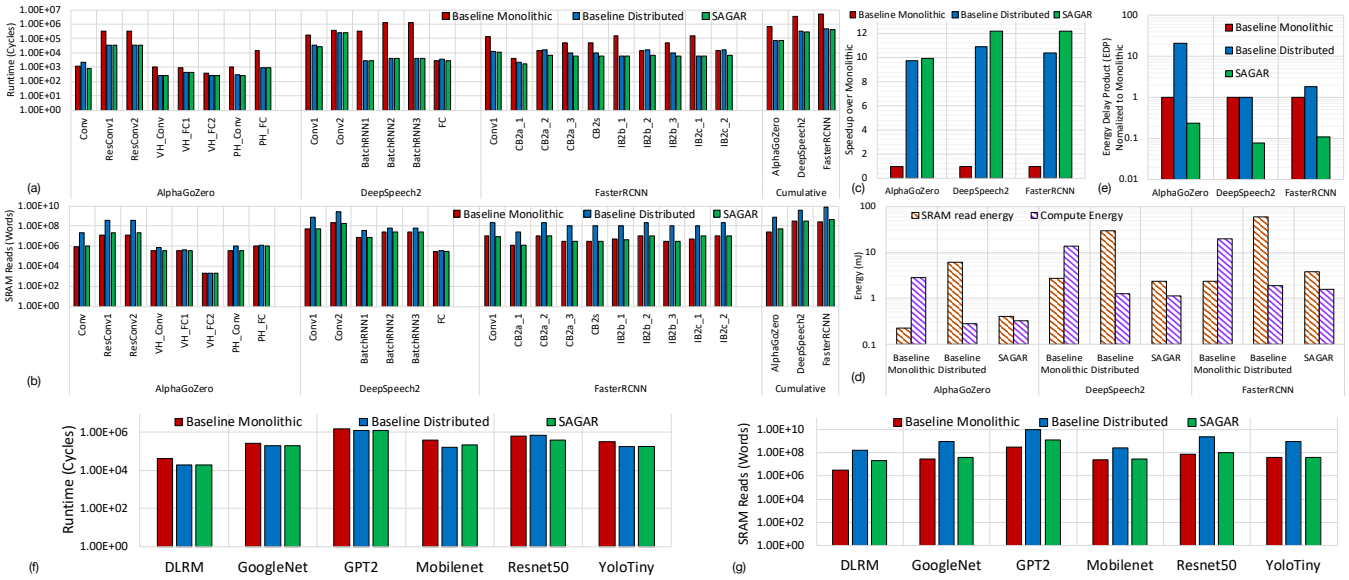

Figure 7: (a) Simulated runtimes for monolithic 128×128 baseline, distributed 1024 4×4 baseline, and *SAGAR* for layers in AlphaGoZero, DeepSpeech2, and first 10 layers of FasterRCNN (b) SRAM reads for the same workloads for *SAGAR* and baseline configurations (c) Speedup of *SAGAR* and distributed baseline as compared to the monolithic baseline (d) Energy consumption breakdown for our workloads in *SAGAR* and baselines (e) Energy delay product(EDP) of *SAGAR* and baselines, normalized to EDP for monolithic baseline (f-g) Sensitivity analysis on various networks for runtime and SRAM reads

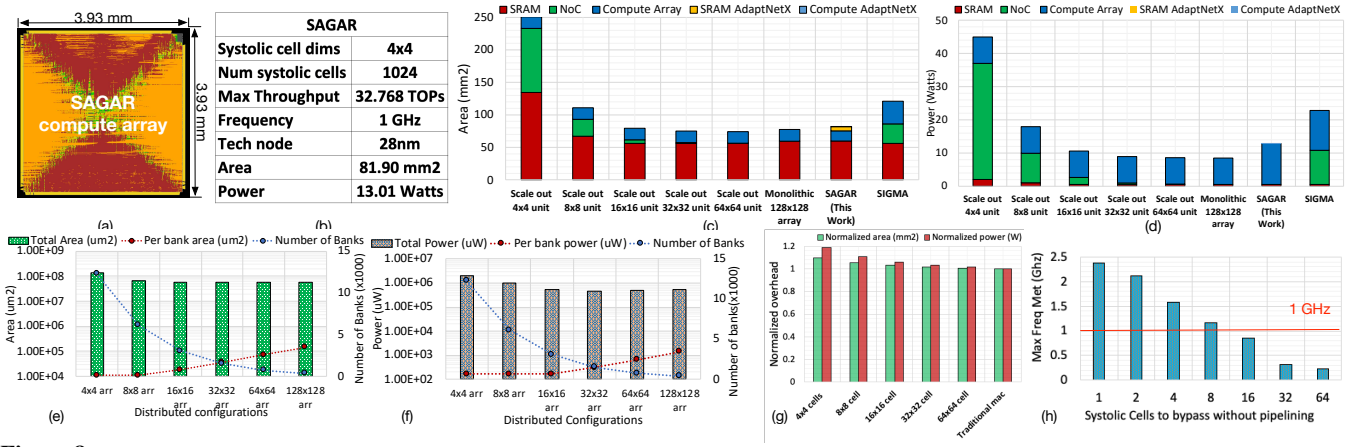

Figure 8: (a) The post PnR floor-plan diagram of *SAGAR*'s compute array, (b) Implementation details of *SAGAR* (c) Post synthesis area for systolic arrays, *SAGAR*, and SIGMA (d) Power consumed by various components in distributed systolic array based designs, the monolithic systolic baseline, *SAGAR*, and SIGMA (e) Total area footprint of SRAM banks in various distributed systolic array and monolithic configuration vs variation in bank sizes and the number of banks required, (f) Power consumption by the SRAM banks in distributed systolic array and monolithic configurations, and (g) the area and power of a 128×128 array vs size of "*systolic-cells*" normalized to the area and power of an array constructed with traditional MAC units. (h) The maximum attainable frequency vs the number of 4×4 *systolic-cell*s to bypass at 28nm.

achieve high performance, and energy efficiency of diverse workloads. The ADAPTNET described in Section 4 provides optimal configurations in a single inference, attains high prediction accuracy, directly from workload parameters, independent of modifications in the software stack. Together these components create a self sufficient unit, which allows users to run GEMM workloads at best possible performance and efficiency. Similar concept can be extended to other reconfigurable architectures (RA) by augmenting it with appropriate Self-Adaptive (SA) component like ADAPTNET. We believe this results in a new class of designs which we name *Self Adaptive Reconfigurable Array (SARA)*.

Custom hardware for ADAPTNET. To completely decouple the architecture optimization from the software stack we run the ADAPTNET on hardware. Given the RA is a GEMM accelerator, it is natural to think running ADAPTNET directly on the RA. However, our experiments depict that running ADAPTNET on a custom hardware core achieves 2× faster inference while incurring minimal hardware overheads when compared to running the network on the

RA itself. Figure 6(a) shows the architecture of the custom core so designed. We term this unit ADAPTNETX.

SAGAR Accelerator. We implement a reference design called SAGAR (shown in Figure 6(b)) by augmenting a reconfigurable array with 2^{14} MAC units (equivalent to TPUv2) and ADAPTNETX for running ADAPTNET-858. SAGAR uses 4×4 *systolic-cell*s and which work on 8bit operands. Since each row and column in this configuration has 31 bypass links and one link to MAC, each buffer is constructed as a collection of 1024 1KB banks.

Enhancements. The design can be further enhanced by enabling caching the optimal configurations for frequently encountered workloads such that ADAPTNET is not invoked for redundant queries. This optimization however trades-off chip area to improve the dynamic power and thus should be considered according to the implementation budgets.

6 EVALUATIONS

We present simulation and implementation evaluations of SAGAR below to distinguish between architecture and implementation aspects. In each case we use several popular DNN models as our workloads, and compare SAGAR against two baselines, a monolithic 128 MAC systolic array and a distributed 1024 4×4 array.

6.1 Architectural evaluations

We simulate the baseline and SAGAR by modifying SCALE-Sim to obtain runtime and energy estimates. Figure 7(a) depicts the layer wise and cumulative runtimes for running three representative workloads on baselines and SAGAR. We observe that SAGAR either matches the performance of the better baseline or outperforms both baselines for each layer, owing to its mapping flexibility. The same observation is corroborated in Figure 7(f) which shows the cumulative runtime for a few other popular networks. Effective exploitation of operand reuse reduces SRAM accesses and thus improve energy efficiency. Figure 7(b) depicts the layer wise and cumulative SRAM reads encountered by the baselines and SAGAR on representative networks. We observe that SAGAR, in spite of emulating the same distributed configurations as the distributed baseline has SRAM accesses closer to the monolithic array. Figure 7(g) depicts the continuation of this property for other networks as well. If we consider the overall energy consumption, Figure 7(d) shows that SAGAR's compute energy is closer to distributed configuration as it does not waste compute cycles due to under utilization, while its SRAM read energy is closer to the monolithic configuration as it exploits operand reuse owing to the bypass logic. This results in lower total energy consumption than either of the baselines. The overall benefit of improved performance and energy efficiency is best captured using Energy-Delay Product (EDP). As Figure 7(e) shows SAGAR achieves orders of magnitude lower EDP than either of the baselines.

6.2 Implementation evaluations

We implement SAGAR and the baselines in RTL and run the ASIC flow till place-and-route (PnR) using TSMC 28nm. The SRAM memories are synthesized using SAED32 and then scaled down using Dennard's scaling. Figure 8(a) shows the post PnR layout of SAGAR's compute array. We observe that SAGAR closes at 1GHz with an area footprint of $81.90mm^2$ and consumes 13.01W (see Figure 8(b)). In Figure 8(c) we provide the area breakdown for SAGAR, the two baselines, a few distributed configurations and a state of the art flexible accelerator SIGMA. We observe that SAGAR takes about 20% of the area of to distributed 4×4 array and 8% more area than the monolithic baseline. SRAM is the predominant contributor for area footprint, however it remains fairly constant for equal capacity, except when very small bank sizes are used as shown in Figure 8(e). We examine the power consumption in Figure 8(d) for SAGAR, our baselines, and SIGMA. We observe that the 4×4 distributed array takes $5.3\times$ more power than SAGAR, while both have same mapping flexibility. The NoC power dominates in fine grained distributed setting, which contributes 78% to the total of the 4×4 distributed array baseline. In SAGAR, the array along with the bypass links consume 50% more power than the monolithic systolic baseline. However, SAGAR takes only about 60% of the power for a fully flexible accelerator like SIGMA. The SRAM power is fairly insignificant and it reduces with the decrease in bank sizes for configurations with equal capacity as shown in Figure 8(f). We study the overheads of the granularity of *systolic-cell*s in Figure 8(g), and observe that we incur about 10% area and 20% power penalty as compared to an monolithic array. Figure 8(h) depicts that we can bypass 8 systolic cells without latching to meet our 1GHz target.

7 CONCLUSIONS

This paper describes SAGAR, a scalable, reconfigurable, self adaptive accelerator for GEMM workloads aimed to extract both performance and energy efficiency. Our evaluations show that owing to the mapping flexibility as well as ability to exploit operand reuse due to bypass wires, SAGAR achieves several order of magnitude better EDP than both baselines while consuming a fraction of area and power of an equivalent distributed systolic array setting.

ACKNOWLEDGMENTS

This work was supported by NSF Award 1909900.

REFERENCES

[1] Manoj Alwani et al. 2016. Fused-layer CNN accelerators. In *MICRO*.
[2] Tianqi Chen et al. 2018. Learning to optimize tensor programs. In *NeurIPS*.
[3] Yu-Hsin Chen et al. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ISCA* (2016).
[4] Jeremy Fowers et al. 2018. A configurable cloud-scale DNN processor for real-time AI. In *ISCA*. IEEE, 1–14.
[5] Mingyu Gao et al. 2017. Tetris: Scalable and efficient neural network acceleration with 3d memory. In *ASPLOS*.
[6] Mingyu Gao et al. 2019. Tangram: Optimized coarse-grained dataflow for scalable NN accelerators. In *ASPLOS*.
[7] Soroush Ghodrati et al. 2020. Planaria: Dynamic Architecture Fission for Spatial Multi-Tenant Acceleration of Deep Neural Networks. In *MICRO*. IEEE.
[8] Norman P Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *ISCA*. 1–12.
[9] Duckhwan Kim et al. 2016. Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory. *ISCA* (2016).
[10] Tushar Krishna et al. 2014. SMART: single-cycle multihop traversals over a shared network on chip. *IEEE micro* (2014).
[11] Hyoukjun Kwon et al. 2018. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. *ASPLOS* (2018).
[12] Wenyan Lu et al. 2017. Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks. In *HPCA*.
[13] Eric Qin et al. 2020. SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training. In *HPCA*.
[14] Ananda Samajdar et al. 2019. Scaling the Cascades: Interconnect-aware FPGA implementation of Machine Learning problems. In *FPL*.
[15] Ananda Samajdar et al. 2020. A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim. In *ISPASS*.
[16] Yakun Sophia Shao et al. 2019. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *MICRO*.
[17] Chen Zhang et al. 2015. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *FPGA*.