HASS: Hardware-Aware Sparsity Search for Dataflow DNN Accelerator

Zhewen Yu*, Sudarshan Sreeram*, Krish Agrawal*, Junyi Wu*, Alexander Montgomerie-Corcoran*,
Cheng Zhang*, Jianyi Cheng†, Christos-Savvas Bouganis*, Yiren Zhao*
*Imperial College London, UK, †University of Cambridge, UK
{zhewen.yu18, sudarshan.sreeram19, krish.agrawal20, junyi.wu21, alexander.montgomerie-corcoran15,
cheng.zhang122, christos-savvas.bouganis, a.zhao}@imperial.ac.uk, jianyi.cheng@cl.cam.ac.uk

***Abstract*—Deep Neural Networks (DNNs) excel in learning hierarchical representations from raw data, such as images, audio, and text. To compute these DNN models with high performance and energy efficiency, these models are usually deployed onto customized hardware accelerators. Among various accelerator designs, dataflow architecture has shown promising performance due to its layer-pipelined structure and its scalability in data parallelism.**

Exploiting weights and activations sparsity can further enhance memory storage and computation efficiency. However, existing approaches focus on exploiting sparsity in non-dataflow accelerators, which cannot be applied onto dataflow accelerators because of the large hardware design space introduced. As such, this could miss opportunities to find an optimal combination of sparsity features and hardware designs.

In this paper, we propose a novel approach to exploit unstructured weights and activations sparsity for dataflow accelerators, using software and hardware co-optimization. We propose a Hardware-Aware Sparsity Search (HASS) to systematically determine an efficient sparsity solution for dataflow accelerators. Over a set of models, we achieve an efficiency improvement ranging from 1.3× to 4.2× compared to existing sparse designs, which are either non-dataflow or non-hardware-aware. Particularly, the throughput of MobileNetV3 can be optimized to 4895 images per second. HASS is open-source: https://github.com/Yu-Zhewen/HASS

I. Introduction

Deep Neural Networks (DNNs) models are designed to extract relevant features from raw data, such as images, audio, and text. To improve the performance and energy efficiency when computing these models, the computation is often mapped onto hardware accelerators. Among various accelerator architectures, dataflow accelerators have shown significant performance benefits because of their deeply pipelined computation between layers [1] and scalable data parallelism across devices [2].

In hardware accelerator designs, sparsity has been a popular topic, such that unnecessary computation with zeros can be avoided for better efficiency. There are two types of sparsity:

- **Weight Sparsity**: focusing on the zeros in the weights of a model, whose positions are often available at compile time so that they can be optimized statically.
- **Activation Sparsity**: focusing on the zeros inside the intermediate activation data between layers. The positions of these zeros are only known at run-time, as they depend on the network input.

With the appropriate hardware support, increased sparsity can lead to fewer computations. To maximize the sparsity, pruning is a process that simplifies a model by setting certain weights and activations to zeros, at minimal accuracy loss based on a set of criteria [3].

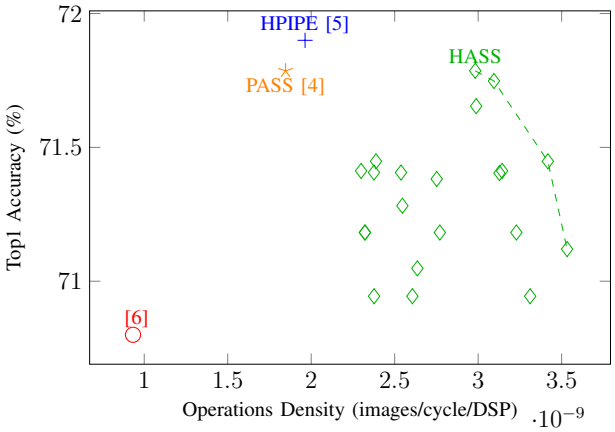


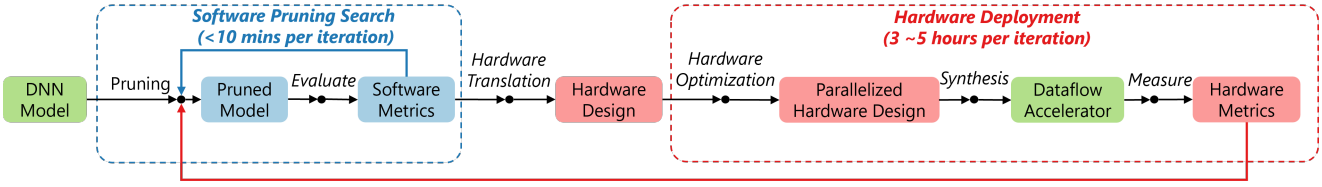
Fig. 1: HASS explores the optimal trade-off between classification accuracy and operation density. We compare HASS with other existing sparse MobileNetV2 implementations.

TABLE I: Our work is the first attempt to exploring hardware-aware unstructured pruning for dataflow accelerators. Our approach systematically exploits layer-wise sparsity in both weights and activations coupled with hardware resource-constrained analysis.

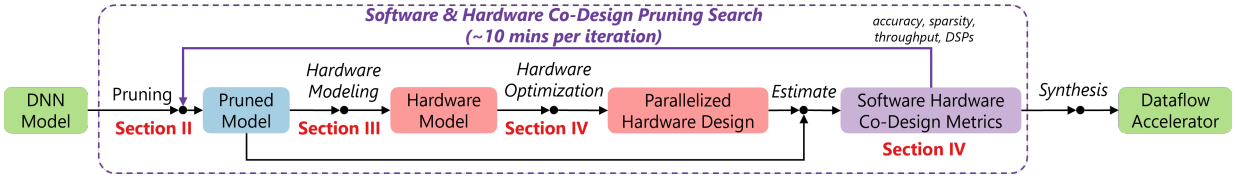
Approaches	[7]	[8]	[9]	[5]	[6]	[10]	[11]	[4]	Ours
Publication Year	2018	2019	2020	2020	2022	2023	2023	2023	2024
Dataflow architecture	✗	✗	✗	✓	✗	✗	✗	✓	✓
Weight sparsity	✓	✓	✓	✓	✓	✓	✓	✗	✓
Activation sparsity	✗	✗	✓	✗	✗	✓	✓	✓	✓
Hardware-aware	✗	✗	✗	✗	✓	✗	✓	✗	✓

Traditional pruning approaches only consider software metrics, such as the overall sparsity in the network, without directly considering the actual impact on hardware performance. Recently, there has been interest in developing hardware-aware, co-design pruning approaches for accelerator designs. These approaches consider hardware performance, such as throughput and energy efficiency, into the pruning criteria. These approaches realize co-optimization of both accuracy and hardware performance.

In non-dataflow hardware accelerators, sparsity is exploited to lift the performance bottleneck in the off-chip memory



(a) Traditional pruning flow for dataflow accelerator design focuses on software metrics only, where a pruned model is treated 'read-only' in the hardware end.



(b) Our flow integrates hardware metrics into the pruning search process. We will explain the details of each step in the highlighted sections.

Fig. 2: An overview of our approach. When designing a dataflow accelerator, traditional pruning steps are separate from the hardware synthesis steps, which restricts software and hardware co-optimization. Our flow explores both software and hardware optimization concurrently, opening up opportunities to find the optimal design.

bandwidth [12]. To efficiently represent sparse data, encoding techniques are often used to reduce both the memory footprints and the required off-chip memory bandwidth [13]. Pruning can be further tailored to optimize individual sparse computation for efficient data access and processing [14].

However, these approaches cannot be applied to the design of dataflow accelerators for two reasons. First, the performance bottleneck for dataflow accelerators is often the computation resources rather than the off-chip memory bandwidth, because most weights and activations reside on-chip. More importantly, in a sparse dataflow accelerator, the overall pipeline performance does not scale proportionally with the total non-zero operations. For example, increasing the sparsity of a non-slowest layer will not change the pipeline performance at all, unless it allows for the reallocation of resources to alleviate the pipeline bottleneck [4].

As such, exploiting the sparsity in a dataflow architecture poses unique challenges in hardware designs, including scheduling and resource allocation within a given hardware resource budget. Existing approaches on sparse dataflow architecture separate the pruning steps and hardware optimization steps, as shown in Fig. 2a. In this paper, we propose a systematic approach to explore unstructured pruning and hardware optimization in a co-design form, as shown in Fig. 2b. To the best of our knowledge, our approach is the first attempt at hardware-aware pruning for dataflow accelerators, as illustrated in Table II. Our main contributions are as follows.

- The first dataflow accelerator design that exploits both weight sparsity and activation sparsity coupled with layer-pipelined execution.
- A hardware-aware, unstructured model pruning algorithm that considers both software pruning metrics (accuracy, sparsity ratio) and hardware performance (throughput, resource estimations) for systematic optimization.
- Over a set of DNN models, we show that our approach leads to an improved efficiency ranging from 1.3× to 4.2× compared to existing sparse designs.

II. Related Work

In the existing studies, the sparse DNN-FPGA accelerators are often implemented as a single sparse-sparse matrix multiplication engine, shared by DNN layers in a time-multiplexed manner. As such, the main challenge is irregular memory access patterns within and between layers. As such, sparse data are often encoded to save memory space and simplify the scheduling of non-zero values to processing units [13]. Lu *et al.* [8] explored the weight sparsity only. As the positions of zeros are known at compile-time, a look-up table is built to match the indices of sparse weights and dense activations. To exploit the dynamic, data-dependent activation sparsity, Zhu *et al.* [9] used the activation value to control the clock gating of processing units, which leads to energy saving but the achieved throughput remains the same as the dense computation. As irregular sparsity pattern leads to inefficient hardware, Kong *et al.* [10] proposed the latency-aware pruning that jointly optimizes the accuracy and latency. Similarly, Qu *et al.* [11] modelled energy consumption and processing cycles, and considered them together in the loss function, so that the accelerator architecture can be considered.

Instead of scheduling layers in a time-multiplexed manner, HPIPE [5] and PASS [4] are two works that look in a different direction, building the dataflow architecture that has multiple pipelined sparse matrix multiplication engines. To solve the resource allocation and throughput balancing between multiple sparse engines, they estimated the distribution of sparse data in PyTorch and used that estimation to guide the process of design space exploration. However, HPIPE only exploits the weight sparsity, while PASS only exploits activation sparsity, and neither of them has considered the hardware-aware co-design in their approaches.

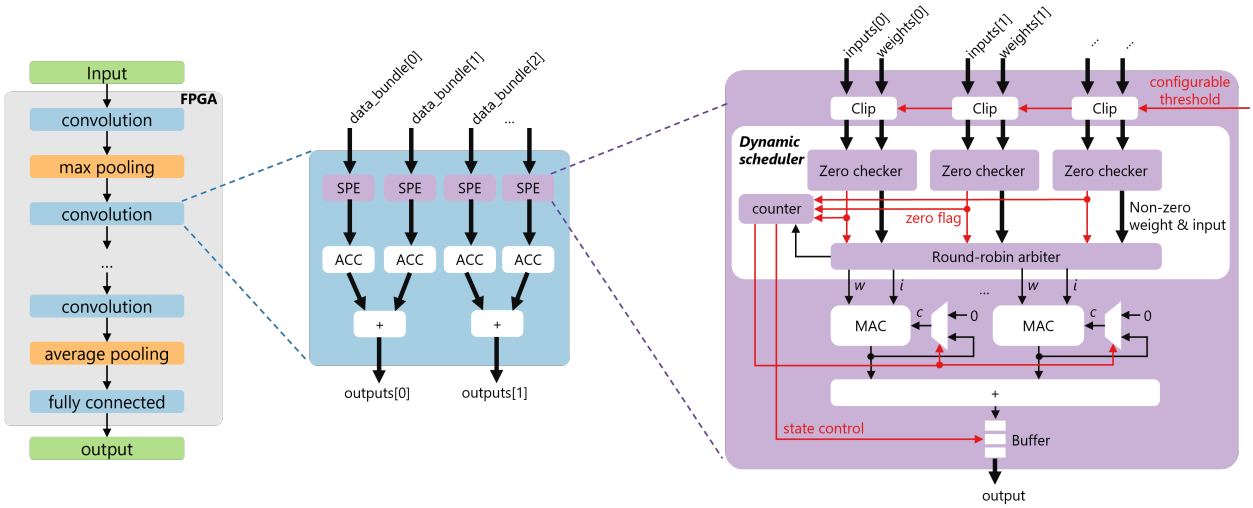


Fig. 3: Architecture of the sparse dataflow accelerator. The computation of layers is pipelined where inside each layer, there are multiple Sparse Vector Dot Product Engines (SPE) operating in parallel. Inside the SPE, clipping modules zero out any weight or activation that falls below a configurable threshold. Zero-filtering then detects these zeros, while the remaining non-zeros are dispatched to the MACs (implemented with DSPs) via an arbiter. We also use a dedicated counter to track the number of skipped zeros and manage accumulation result emissions.

III. Unstructured Pruning Algorithm

In this paper, we choose to apply one-shot pruning without fine-tuning, where a significant portion of the network is pruned in a single step, to reduce the deployment effort. This is in contrast to iterative pruning approaches, where pruning is performed gradually over multiple iterations, accompanied by fine-tuning stages [15].

In terms of the pruning criterion, we use the magnitude-based evaluation (L1 norm), where weights and activations having smaller magnitudes below the threshold will be forced to zeros. The pruning threshold can either be uniform or unique across layers. Adopting a uniform threshold is straightforward to explore the trade-off between accuracy and sparsity. However, setting unique thresholds instead better preserve the network accuracy, as many researchers have demonstrated the diversity of statistics per layer [14], [16].

Therefore, for each individual layer in the network, our pruning algorithm requires the identification of unique thresholds τ_w and τ_a for weight and activation pruning respectively, and the introduced sparsity is denoted as S_w and S_a, with a range between 0 and 1, indicating the fraction of zeros in the data. As introduced before, once the pruning threshold is given, the weight sparsity S_w is a fixed value since the positions of zeros are already available at compile time. However, the activation sparsity S_a depends on the network's input, making it dynamic at run-time.

IV. Sparse Dataflow Architecture

Traditional sparse architectures use two approaches to address the challenges in memory storage and computational efficiency. For efficient memory storage, they statically encode weights, trading off between efficient memory storage and runtime overhead caused by decoding [13]. For efficient computation, they skip zeros during computation, but leaving the corresponding hardware operator idle [9]. To fully utilize the available computation resources, we statically analyze the run-time sparsity and pre-fetch data in a buffer to keep the hardware operators busy at each cycle. This leads to better performance than the zero-skipping approach.

Fig. 3 illustrates our target sparse dataflow accelerator architecture. On the left, the model is presented in a dataflow graph, where each node represents a dataflow component in hardware and each edge represents their data interface between every two nodes. The blue nodes are often resource intensive, such as DSP blocks on an FPGA, and can be significantly optimized by sparse computation. In each blue node, data parallelism is applied to improve performance. For example, the middle of the figure represents a hardware convolutional layer. The hardware optimization steps consider two metrics: 1) spatial data parallelism by duplicating hardware processing elements, shown as multiple Sparse Vector Dot Product Engine (SPE) in the figure, and 2) time-wise folding by iterating using the same SPE and accumulating the result, shown as ACC and "+" in the figure. Detailed exploration on these metrics are explained in Section V-A.

The hardware implementation of sparse computation is in SPE, shown in the right side of Fig. 3. The engine takes a set of inputs from the preceding layer and weights from the on-chip memory, which are passed through the clip modules which would zero out any value below the configurable threshold. Afterwards, zero values are directly forwarded to the counter on the left to keep track of the iteration count, while non-zero pairs are buffered in a round-robin arbiter for computation. The arbiter dispatches multiple pairs to the available Multi-

ply–Accumulate (MAC) units concurrently such that they are busy in each clock cycle. Once the counter reaches full, the output data will be released.

Note that the accumulation of the vector product can take place both inside each SPE and also between multiple SPEs. This strategy can constrain the fan-in and fan-out of the arbiter, to reduce the area overhead and the degradation of clock frequency.

The number of the MACs in each SPE, denoted as N, is statically determined by estimating the overall sparsity of the input activations and weights on a calibration dataset. Consider the scenario where a SPE takes M input weight pairs. In traditional hardware architecture for dense computation, the computation would take M/N cycles to complete. Each MAC unit accumulates for M/N times before emitting partial sums to the N-input adder tree for the final output calculation. In sparse hardware architecture, let $\overline{S}$ be the average sparsity of the input activation and weight pair, where the computations can be skipped if any s any of them is equal to zero. In this case, the arbiter will schedule the computation to complete with t cycles, where:

$$t(\overline{S}) = \lceil \frac{(1-\overline{S}) \times M}{N} \rceil \tag{1}$$

This is also known as initial interval of the SPE. Considering both weight and activation sparsity, our focus is on the probability of either weight or activation becoming zero. In our implementation, N and M are customized for each layer according to its $\overline{S}$.

The proposed sparse computation engine can be aggregated to enhance performance, both within a layer (intra-layer) and between layers (inter-layer). Within each convolutional layer, concurrent vector dot products can occur, allowing parallel computation across the input-channel (I) and output-filter (O) dimensions. We designate the levels of this parallelism as $i \in [1, I]$ and $o \in [1, O]$, respectively. In the case of unstructured pruning, the sparsity pattern is not uniform within a convolutional layer. Consequently, the processing rates of $i \times o$ SPEs are dynamic and may be imbalanced at run-time, potentially causing pipeline stalls. To mitigate this, we employ the following strategies:

- **Balancing Strategy**: During compile-time, we estimate the weight and activation sparsity in each input-channel and output-filter. We then utilize simulated annealing to solve an allocation problem. This assigns the computation of I input-channels and O output-filters to $i \times o$ sparse computation engines, minimizing the difference in their processing rates.
- **Buffering Strategy**: Buffering is employed to absorb the instantaneous variance of dynamic processing rates. The selection of buffer size involves a trade-off between resource usage and throughput, and we determine the buffer size following a heuristic approach similar to [4] which is based on the observation of moving window statistics.

In the sparse dataflow architecture, computation is pipelined on a layer-by-layer basis using FIFOs and handshake signals. In this architecture, activation data is not encoded despite its sparsity. This is because most intermediate data remains on-chip, and frequent encoding and decoding between layers would incur significant computational costs.

V. Hardware-aware Workflow

A. Accelerator Design Space Exploration

This section focuses on the Design Space Exploration (DSE) problem of the sparse DNN-FPGA accelerator, and it can be formalized using the following terms:

- $L : \{l_0, l_1, l_2...\}$ denotes the layers in a network;
- $S \subseteq [0, 1)$ denotes the sparsity search space;
- $D : \{d_0, d_1, d_2...\}$ denotes all possible hardware design points of a layer;
- $g \subseteq L \times D \times S$ denotes a design point of the network;

Given a budget for the hardware resources R, we search for an efficient g to maximize network throughput in a greedy form.

1) Performance modeling: Let C_l denote the number of operations (including the zeros) in layer l. The throughput of the same layer is then:

$$\theta(l, d, \overline{S}) = \frac{i \times o \times M}{C_l \times t(\overline{S})} \tag{2}$$

Since t is dynamic, depending on the average sparsity $\overline{S}$, so as the throughput of the layer. The sparse dataflow architecture is layer-wise pipelined, therefore, the network throughput θ is restricted by the slowest layer:

$$\forall g.\, \theta \leq \min_{l \in L} \theta(l, d, \overline{S}) \tag{3}$$

The aim of the DSE process is to determine an efficient g with a large θ.

.

2) Resource-constrained rate balancing: Apart from the slowest layer, the remaining layers in the pipeline may underperform, meaning that their actual throughput is significantly lower than the maximum achievable throughput using the allocated resources. These hardware resources could be unused at run-time, leading to inefficiency. To address this, we can reduce the parallelism of these layers without affecting overall throughput to enable more parallelism at the performance bottleneck. This is also known as rate balancing.

Let $\theta_r(l, d, \overline{S})$ represent the actual throughput of a layer instance $(l, d, \overline{S})$. After rate balancing, each layer in the balanced design g' is configured with a parallelism level close to its actual throughput.

$$\theta'_l = \min\{\theta(l, d', \overline{S}) | \theta(l, d', \overline{S}) \geq \theta_r(l, d, \overline{S})\} \tag{4}$$

$$\forall l \in L \wedge (l, d, \overline{S}) \in g \Rightarrow \theta(l, d', \overline{S}) = \theta'_l \wedge (l, d', \overline{S}) \in g' \tag{5}$$

Constraint (5) ensures that all layers in the design are computing efficiently in a pipeline, optimizing the overall resource utilization.

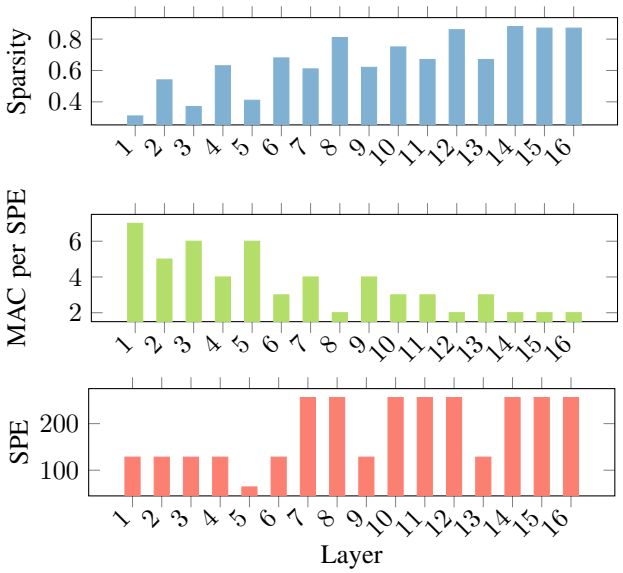


Fig. 4: DSE results of a specific sparse ResNet-18 workload with 16 3×3 convolutional layers. The allocation of MAC per SPE mainly depends on the per-layer sparsity statistic. A higher sparsity leads to a smaller MAC per SPE. However, the increase of the layer index leads to an increasing number of convolutional filters, so as the number parallel SPEs to match the rate between layers.

3) Resource-constrained incrementing: The DSE starts with the resource-minimal design, where the computation within each layer is fully sequential. At each iteration, the DSE increases the parallelism of the slowest layer by a small step followed by the rate balancing process described in step 2). This incremental process repeats until the resource budget R satisfies. The resource utilization of each sparse computation engine is modeled on the basis of the regression model.

4) Partitioning and reconfiguration: In practice, the restriction caused by finite hardware resources could fail to map an entire DNN to a single dataflow accelerator device. In this work, we fold the dataflow pipeline at the block level and iteratively compute them on the available hardware resources using full reconfiguration on an FPGA device. This allows changing hardware architecture at run-time at the price of additional reconfiguration time. To reduce such overhead caused by the reconfiguration time, the data is processed in a large batch size [1]. The decisions of where to split the partition and the number of partitions are given by a simulated annealing solver that trade-off the reconfiguration time and data parallelism gained.

B. Multi-objective Search

Let's denote the per-layer weight and activation pruning threshold as τ_w and τ_a. To identify the optimal values of them, we construct the following optimization problem for the network L:

- f_{acc}: network accuracy, measured on validation data;
- f_{spa}: average sparsity of the network, including both weights and activations;

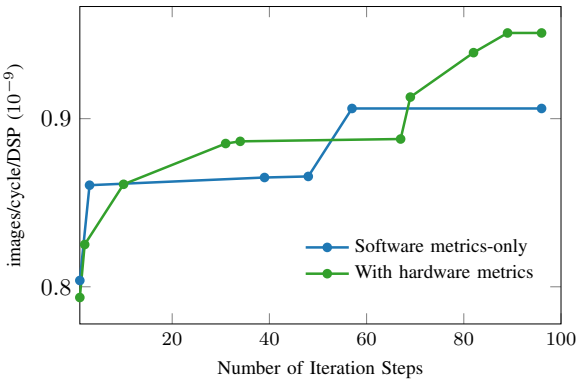


Fig. 5: Comparison between software metrics-only sparsity search and the proposed hardware-aware sparsity search for ResNet-18. We use 96 iteration steps for both approaches.

- f_{thr}: network throughput for a given pruned network searched by DSE;
- f_{dsp}: resource utilization of the accelerator, represented by the number of DSPs used, which is usually the bottleneck in dataflow accelerators.

The objective for the hardware-aware pruning is to maximize accuracy, sparsity, and throughput while minimizing resource utilization:

$$\max_{\{\tau_w,\tau_a\},l\in L} f_{acc}(L)+\lambda_1 f_{spa}(L)+\lambda_2 f_{thr}(L)-\lambda_3 f_{dsp}(L) \quad (6)$$

λ_1, λ_2 and λ_3 are hyperparameters that normalize the values of these metrics, determined by heuristics. An efficient search algorithm for multi-objective search is required, and we use Tree-structured Parzen Estimator (TPE) [17], a Bayesian optimization algorithm that uses a tree structure to model the probability density of the objective function and iteratively constructs this tree based on observed evaluations of the objective function.

The proposed hardware-aware pruning approach could achieve better computation efficiency (throughput per area) than the same pruning approach with software metrics only (i.e. accuracy and sparsity). Fig. 5 shows a comparison of these two approaches for ResNet-18. The proposed hardware-aware search shown as the green curve is guided by the objective function in Equation 6. The software metrics-only search shown as the blue curve only uses accuracy and sparsity as the objective function. Both approaches ran 96 iteration steps, taking about 3 hours to complete. At the initial stage, the change in green curve in computation efficiency is slow because the objective function is complex and contains more hardware metrics. Then it reaches a better computation efficiency because the hardware metrics guided the search towards efficient hardware implementation. This is more helpful if more iteration steps are performed for large networks.

VI. Evaluation

In terms of the experiment set-up, the FPGA device we used for the result measurements is AMD Xilinx Alveo U250, and the version of software employed is Vitis 2023.1. The clock

TABLE II: A comparison with state-of-the-art sparse DNN-FPGA accelerators.

Models	ResNet-18		ResNet-50			MobileNetV2				MobileNetV3S		MobileNetV3L	
Works	[4]	Ours	[6]	[4]	Ours	[6]	[5]	[4]	Ours	Dense	Ours	Dense	Ours
Accuracy	69.75	69.59	N/A	76.13	75.58	70.80	71.90	71.79	71.45	67.42	67.28	74.04	73.76
Platform	U250	U250	7V690T	U250	U250	7V690T	Stratix10	U250	U250	U250	U250	U250	U250
Freq (MHz)	250	250	150	250	250	150	390	250	250	250	250	250	250
DSPs	10974	12234	2160	11952	7434	2160	5928	3596	5261	4282	1796	6577	4324
kLUTs	1659	1679	308	1721	1724	308	523	1552	1720	971	507	1535	1728
BRAM18k	4554	4817	1883	4262	4178	1883	4512	1774	1902	2278	1779	3706	5376
images/s	1904	2819	33	330	776	302	4539	1660	4495	4890	4895	1897	1898
images/cycle/DSP (10^{-9})	0.69	**0.92**	0.10	0.11	**0.42**	0.93	1.96	1.84	**3.42**	4.57	**10.90**	1.15	**1.76**

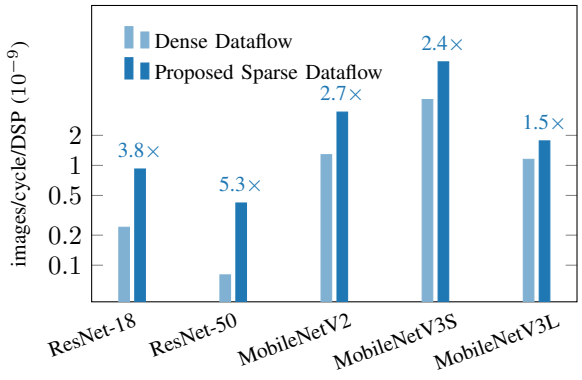


Fig. 6: By exploiting sparsity in both weights and activations, our approach achieves significant speedup compared to the dense implementation.

frequency of our accelerators is 250MHz. We examine our approach on a set of mainstream DNN benchmarks including ResNet-18, ResNet-50, MobileNetV2 and MobileNetV3, taken from pyTorch torchvision and trained on the ImageNet dataset. All the networks quantized to 16-bit fixed point for both weights and activations. Our pruning algorithm is one-shot and post-training, without applying any fine-tuning to the networks.

The tool flow is presented in Fig. 2b. We implemented a fully-automated flow that translates DNN models to Torch FX graph. In an FX graph, the model is represented as a dataflow graph similar to the one shown on the left of Fig. 3. Our pruning search engine extracts the layer-wise pruning space using the TPE algorithm. With the proposed hardware model presented in Section V-A, the hardware cost is evaluated as well as the accuracy loss. This guides the next iteration step of the pruning search process. The final pruned model and its hardware parameters are sent to a hardware synthesis tool for hardware implementation. Our approach is general and can be extended to support other hardware synthesis tools. For this work, we used a DNN-FPGA synthesis tool named fpgaConvNet [1] for prototyping.

We compare our approach with other sparse DNN-FPGA accelerators from related works. Our evaluation encompasses the trade-off among accuracy, resource, throughput (images/s) and hardware efficiency (images/cycle/DSP). Fig. 6 shows the improvements from the dense dataflow architecture to the proposed sparse dataflow architecture, and the detailed results are shown in Table II. We made the following observations:

- Compared with related work using sparse computation, the dataflow architectures, including HPIPE [5], PASS [4] and Ours, have significant performance improvements over the non-dataflow design [6] because of its deeply pipelined computation. For example, when targeting ResNet-50 and MobileNetV2, the advantage in terms of throughput per DSP can be up to 4.2 × and 3.7×, respectively.
- The advantage of sparse dataflow architecture is not always free. Compared to the non-dataflow sparse accelerator design [6], the dataflow approach requires more resources (upto 3× DSPs and 5× LUTs). This is because the hardware dataflow pipeline requires extra logic and buffers in exchange for pipeline performance and data parallelism scalability.
- For MobileNetV3, we also include the dense results in the table. Compared with the dense results, our sparse implementations achieve the same throughput with a reduced number of DSP utilized. The throughput remains similar because the designs are either LUT or BRAM bounded.
- PASS [4] is the work related most to our solution, and it optimizes hardware for the activation sparsity. However, they do not explore weight sparsity or consider to drive the pruning strategy with hardware information. Compared with them, our approach achieves improved efficiency with 1.3×, 3.8× and 1.9× on ResNet-18, ResNet-50 and MobileNetV2. Meanwhile, our post-training accuracy degradation is less than 0.6 percentage points.
- The variance in the results depends on the sensitivity of the models to data sparsity. The accuracy loss may be reduced with the help of fine-tuning, at the price of training time.

VII. Conclusion

In this paper, we propose a novel approach to exploit software and hardware co-optimization for sparsity, targeting the dataflow DNN-FPGA accelerator. We implement a hardware-aware sparsity search that considers both software pruning metrics and hardware performance for systematic optimization. We also exploit both static weight sparsity and dynamic activation sparsity for dataflow accelerator design. Over a set of DNN models, we achieve 1.3× to 4.2× efficiency compared to existing sparse designs. In terms of future work, we will delve into the integration of a more diverse range of pruning algorithms to further enhance applicability.

REFERENCES

[1] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 2, 2019.

[2] Y. Zhang, N. Zhang, T. Zhao, M. Vilim, M. Shahbaz, and K. Olukotun, "Sara: Scaling a reconfigurable dataflow accelerator," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 1041–1054.

[3] A. K. Jain, H. Omidian, H. Fraisse, M. Benipal, L. Liu, and D. Gaitonde, "A domain-specific architecture for accelerating sparse matrix vector multiplication on fpgas," in *2020 30th International conference on field-programmable logic and applications (FPL)*. IEEE, 2020, pp. 127–132.

[4] A. Montgomerie-Corcoran, Z. Yu, J. Cheng, and C.-S. Bouganis, "Pass: Exploiting post-activation sparsity in streaming architectures for cnn acceleration," in *2023 33rd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2023.

[5] M. Hall and V. Betz, "Hpipe: Heterogeneous layer-pipelined and sparse-aware cnn inference for fpgas," *arXiv preprint arXiv:2007.10451*, 2020.

[6] Z. Liu, Q. Liu, S. Yan, and R. C. Cheung, "An efficient fpga-based depth-wise separable convolutional neural network accelerator with hardware pruning," *ACM Transactions on Reconfigurable Technology and Systems*, 2023.

[7] L. Lu and Y. Liang, "Spwa: An efficient sparse winograd convolutional neural networks accelerator on fpgas," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[8] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on fpgas," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 17–25.

[9] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse convolutional neural networks on fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 9, pp. 1953–1965, 2020.

[10] Z. Kong, P. Dong, X. Ma, X. Meng, W. Niu, M. Sun, X. Shen, G. Yuan, B. Ren, H. Tang *et al.*, "Spvit: Enabling faster vision transformers via latency-aware soft token pruning," in *European Conference on Computer Vision*. Springer, 2022, pp. 620–640.

[11] Y. Qu, Y. Ma, and P. Zhou, "A speed-and energy-driven holistic training framework for sparse cnn accelerators," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[12] A. K. Jain, C. Ravishankar, H. Omidian, S. Kumar, M. Kulkarni, A. Tripathi, and D. Gaitonde, "Modular and lean architecture with elasticity for sparse matrix vector multiplication on fpgas," in *2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2023, pp. 133–143.

[13] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH computer architecture news*, vol. 45, no. 2, pp. 27–40, 2017.

[14] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, and B. Li, "Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights," *Proceedings of the IEEE*, vol. 109, no. 10, pp. 1706–1752, 2021.

[15] C. M. J. Tan and M. Motani, "Dropnet: Reducing neural network complexity via iterative pruning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9356–9366.

[16] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[17] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.