LAMBDA: A RECONFIGURABLE ACCELERATOR WITH DATA REORDERING SUPPORT FOR LOW-COST ON-CHIP DATAFLOW SWITCHING

Jianming Tong Anirudh Itagi Tushar Krishna
Georgia Institute of Technology

ABSTRACT

The increasing prevalence of Machine Learning (ML) in various applications has led to the emergence of ML models with diverse structures, types and sizes. The ML model inference boils down to the execution of different dataflows (tiling, ordering, parallelism, and shapes), and using the optimal dataflow can reduce latency by up to two orders of magnitude over an inefficient one. Unfortunately, reconfiguring hardware for different dataflows involves on-chip data reordering and datapath reconfigurations, leading to non-trivial overheads that hinder ML accelerators from exploiting different dataflows, resulting in suboptimal performance. To address this challenge, we propose **LAMBDA**, an innovative accelerator that leverages a novel multi-stage reduction network called Additive Folded Fat Tree (AFFT) for reordering data in data reduction (RIR), enabling seamless switching between optimal dataflows with negligible latency and resources overhead. **LAMBDA** creates an opportunity to change the optimal dataflows at the granularity of layers without incurring additional latency overhead, and to explore the optimal dataflows on the real hardware with faster and more precise evaluation results. **LAMBDA** demonstrates a $0.5 \sim 2\times$ speed up in end-to-end inference latency than the SotA Xilinx DPU on Xilinx ZCU 104 embedded FPGA board.

1 INTRODUCTION

The field of Machine Learning (ML) is expanding rapidly, as ML is successfully applied beyond image classification, object detection/recognition, text summarization, sentiment analysis, and next word prediction. Such a plethora of ML models introduces great diversity in structure (serial or parallel layers connectivity), types (depth-width, point-width, dilation-based convolutions, or even a fusion of them), and sizes (number of channels, kernels, height, and width).

Different ML models have varying preferences for dataflows, which can lead to significant differences in utilization and up to 2-magnitude variance in latency on most ML accelerators (Kao et al., 2022; Kao & Krishna, 2020). However, changing dataflows of accelerators requires modifying data layout in on-chip buffers and reconfiguring datapaths in computation, distribution, and reduction networks.

To minimize the overheads of datapath reconfiguration, prior work has introduced multiple redundant datapaths at design time, with only a subset activated at runtime through configuration (Reuther et al., 2022). These approaches introduce resource overheads but reduce the impact of reconfiguration on latency (Samajdar et al., 2021; Qin et al., 2020; Kwon et al., 2018; Hegde et al., 2019; Zhou et al., 2018; Wang et al., 2021).

However, the overhead of data reordering is a critical and often overlooked problem when dynamically changing dataflows, as the benefits from changing dataflows can be overshadowed by the data reordering overhead. Without a suitable data layout, the required data may be located in the same SRAM banks, resulting in severe underutilization of computation engines due to competence on the same SRAM reading ports. This can lead to stalling and a significant performance gap between theory and practice.

In addition, on-chip data reordering on existing ML accelerators requires additional intermediate storage and back-and-forth data movement between on-chip storages, resulting in significant resource overhead and latency costs. In fact, these costs can outweigh the benefits of switching dataflows, leading existing ML accelerators to compromise settling on a single dataflow that provides good average utilization across all layers, but suboptimal performance.

To unleash the optimal performance, we propose an innovative accelerator **LAMBDA** featured with a novel reconfigurable reduction network called Additive Folded Fat Tree (AFFT), to enable reordering of data in data reduction (RIR). With **LAMBDA**, the latency of data reordering is completely hidden in data reduction, allowing the data layout in on-chip storage to be manipulated for the demand of optimal dataflow, leading to optimal performance.

More generally, there's a significant amount of *dataflows* with various Tiling, Ordering, Parallelism, and Shape (TOPS) (Kwon et al., 2021). **LAMBDA** also opens an opportunity for exploring the impact of various dataflows on

Table 1. Features comparison (prior works v.s. **LAMBDA**).

Work	Dataflow Switching	Load Balancing	Data Layout Consideration
MAERI (Kwon et al., 2018)	✗	✓	✗
SIGMA (Qin et al., 2020)	✗	✓	✗
NVDLA (NVIDIA, 2016)	✗	✓	✗
Eyeriss v2 (Chen et al., 2018)	✗	✗	✓
Xilinx DPU (Xilinx, 2022)	✗	✗	✓
Gemmini (Genc et al., 2019)	✗	✗	✓
SARA (Samajdar et al., 2021)	✗	✗	✗
LAMBDA	✓	✓	✓

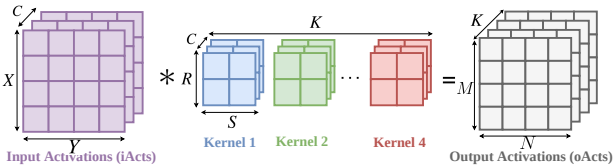


Figure 1. Terminology of convolution workload

the real hardware with the consideration of data layout in on-chip storage, which is not considered by any existing accelerator (to the best of our knowledge).

In addition to the hardware, we devise the end-to-end deployment framework that (a) compiles ML workloads with intra-layer TOPS optimizations in real time. (b) generates configurations for **LAMBDA** to manipulate the AFFT. (c) schedule workloads automatically between CPU and **LAMBDA**. We demonstrate the benefits of optimal dataflow as well as the exploration by implementing **LAMBDA** on Xilinx FPGA.

Our key contributions can be summarized as follows:

- a flexible reconfigurable end-to-end inference accelerator, **LAMBDA** which enables real-time optimal dataflows switching for every single convolution layer with negligible latency overhead.
- a multi-stage reduction network, Additive Folded Fat Tree (AFFT), together with its routing algorithm to enable flexible data reordering (including arbitrary unicasting and multicasting) in data reduction.
- an end-to-end inference framework and compiler to deploy arbitrary ML models down to **LAMBDA** running on Xilinx FPGAs. On ResNet50, **LAMBDA** achieves $0.5 \sim 2\times$ speedup over Xilinx DPU with a similar number of PE running at the same clock frequency.

2 Background and Motivation

2.1 Convolution Layers and Dataflow Space

Fig. 1 depicts the general convolution with seven dimensions with various shapes. Existing dataflows could be represented as a nested loop with four types of optimization.

- **Tiling** breaks down dimensions K, C, X, Y into smaller chunks, and enables executing workloads in tile granularity as on-chip storage is limited.
- **Ordering** allows arbitrary loop reordering to reuse more data since dimensions K, C, X, Y, R, S don't come with loop-carry dependency.
- **Parallelism** allows for arbitrary parallelism over any dimensions as all dependencies are loop-independent, leading to different reuse opportunities.
- **Shaping** permits variant sizes of workload tiles.

All four above types of optimizations create an extremely large dataflow design space with a complexity of $O(10^{36})$ for a single convolution layer (Kao & Krishna, 2020). Surprisingly, none of these dataflows is generally optimal for all types of workloads (Kwon et al., 2020).

2.2 Reconfigurable Accelerator and Challenges

The current state-of-the-art (SotA) publicly available FPGA accelerator is the Xilinx DPU. We evaluate the performance of the Xilinx DPU B2304 running at 200 MHz on the ZCU 104 evaluation board under two typical workloads, ResNet50 and MobileNetV3. The results are shown in Fig. 2, where the vertical axis encodes the ratio of real latency (measured in us) to the theoretical number of computations. A higher value on y-axis indicates a higher inefficiency under a specific workload because it consumes more latency for a given number of demanded computations. Both figures reveal two inefficiencies of the Xilinx DPU

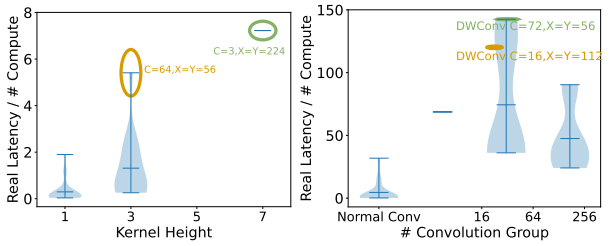
- Fig. 2a shows a general inefficiency of Xilinx DPU across various workloads, especially on workloads with large iAct height/width and large kernel height/width as illustrated by the pointed circles.
- Fig. 2b depicts that depth-wise convolution costs about $50\times$ latency/compute than the average ratio of normal convolution layers on Xilinx DPU.

The inefficiency of Xilinx DPU stems from its rigid homogeneous scaling up systolic array design, which only supports a single dataflow with parallelism $(2, 12, 12)$ in $(X/Y, C, K)$ for Xilinx DPU B2304. Under such a fixed dataflow, Xilinx DPU hardly achieves good utilization among all different types of workloads. This limitation highlights the potential for performance improvement through switching dataflows. By adopting the optimal dataflow for each individual layer, we can achieve better performance and efficiency.

A comparison between **LAMBDA** and other prior works is illustrated in Tab. 1, where none of prior arts supports dataflows switching based on the workloads.

3 LAMBDA Overview

In this section, we will unveil the mechanism of **LAMBDA** to change dataflows with negligible overheads by introducing separate components and corresponding scheduling.



(a) ResNet50 Profiling. (b) MobileNetV3 Profiling.

Figure 2. Performance profiling of Xilinx DPU (B2304).

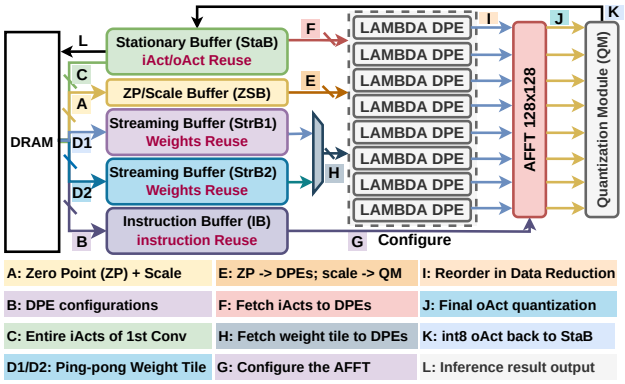


Figure 3. The overview of the **LAMBDA** architecture.

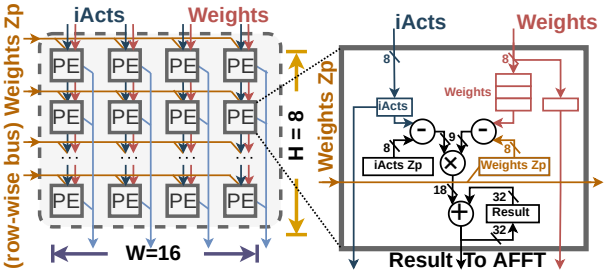


Figure 4. The micro-architecture of **LAMBDA** DPE and PE.

3.1 LAMBDA Dot Product Engine (DPE) Design

Typical accelerators feature thousands of MAC arrays to deliver sufficient throughput e.g. Google TPUv4, IBM's AIU, and Nvidia's Simba (Dey et al., 2019), and so on. However, a homogeneous rigid arrangement of thousands of MAC arrays could lead to low utilization on depth-wise and point-wise workloads. To address this, **LAMBDA** implements a scale-out design of **LAMBDA** DPEs, each with independent input ports that fetch data from exclusive on-chip storage. The output ports of all DPEs connect to the same AFFT reduction network, allowing fine-grain data reduction and data reordering among all on-chip storage. We implement separate computation engines for ReLU and BatchNorm. Pooling layers could be converted to convolution and get executed on the DPEs with a comparator add-on. To minimize data movement costs, all computation engines share the same on-chip storage.

The microarchitecture of DPE is illustrated in Fig. 4. Each DPE contains a 8×16 2D PE array. Infrequently changing data like zero points of weights will be fed through a row-wise bus to each PE. Frequently changing data such as input activations (iActs) and weights move inside the 2D PE array in a column-wise store-and-forward manner, allowing for both reuse across different rows. Additionally, each PE has an internal 8-depth buffer to hold stationary data inside PE and generate results every 8 cycles. The combination of store-and-forward logic and internal buffers enables only a single PE row to generate valid output every cycle, such that all rows of PEs can share the same reduction network with a column-wise bus to deliver output to it.

3.2 On-chip storage

The on-chip storage has been physically divided into separate buffers with different layouts for sufficient bandwidth.

3.2.1 Stationary (StaB) and Streaming Buffer (StrB)

The typical paradigm of processing convolution or General Matrix Multiply (GEMM) will keep one type of data stationary, termed a stationary tensor, and stream the other type of data, termed as a streaming Tensor. As for the streaming tensor, **LAMBDA** fetches and processes in the tiling granularity and we implement a ping-pong buffer to enable the latency hiding of fetching the next tile from off-chip DRAM. Fig. 3 shows the case of buffer-level activation stationary, where iActs and oActs will be kept stationary inside StaB while weights will be streamed through StrB1/StrB2. We could swap the storage of iActs/oActs and weights for buffer-level weights stationary based on the demand of dataflows.

3.2.2 ZP/Scale Buffer (ZSB)

We adopt PyTorch FBGEMM and QNNPACK quantization schemes. Both convert 32-bit floating point data into 8-bit integers together with 8-bit zero points (ZP) and 32-bit floating point scales (Krishnamoorthi, 2018). Further, the scale is quantized in a 32-bit fixed point format with a 30-bit fraction to save computation overhead. Both ZP and scale are shared among a group of data to save storage.

3.2.3 Instruction Buffer (IB)

The **LAMBDA** is featured with a reconfigurable flexible reduction network, the configurations of which are generated offline and get fetched into IB to configure the reduction network in the run-time.

3.3 Additive Folded Fat Tree (AFFT)

The Additive Folded Fat Tree (AFFT) is a multi-stage network that performs data reordering in data reduction. It takes computation results from **LAMBDA** DPE array and routes them to new locations of buffer while performing data reduction, lining it up for the layout required by the next

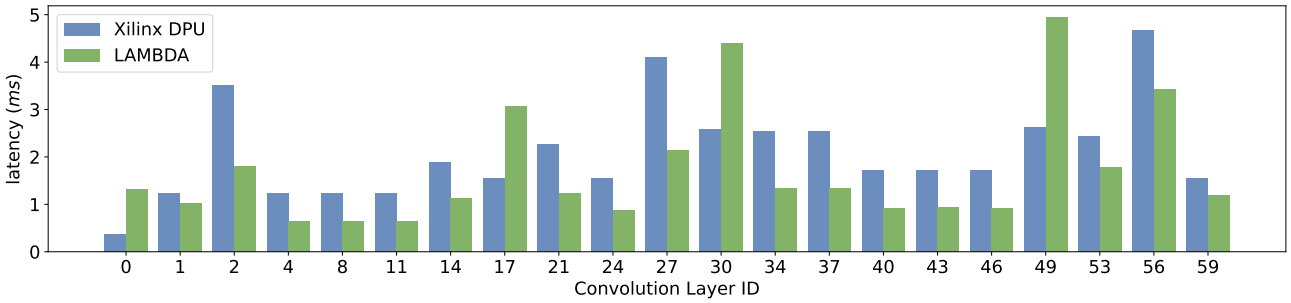


Figure 5. The latency comparison between **LAMBDA** and Xilinx DPU on ResNet 50.

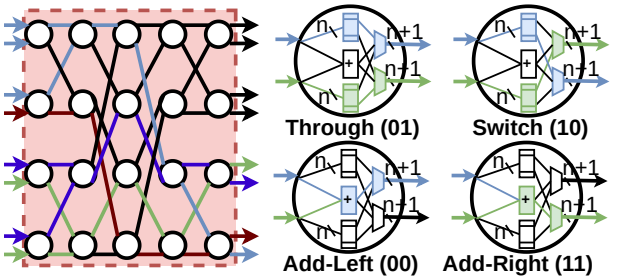


Figure 6. Functionality of AFFT 8x8 switch and control.

dataflow. Such low-cost data reordering enables **LAMBDA** to change optimal dataflows for different workloads, thus further reducing latency.

3.3.1 AFFT Switch

The AFFT utilizes a 2x2 AFFT switch as its basic building block, which is depicted in Fig. 6. The switch is controlled by a 2-bit configuration word to manipulate 4 reordering in reduction (RIR) patterns as illustrated in Fig. 6.

- **Through/Switch**: directly connect or swap two input data to output ports.
- **Add-Left/Right**: add two input data and send to the left or right output port.

3.3.2 AFFT Topology

For **LAMBDA** DPEs with N columns in total (N must be a power of 2), the AFFT has $2 \times log(N) - 1$ level with $N/2$ AFFT switches located at every level. In essence, the AFFT topology is a folded version of the fat tree topology, which makes it a symmetric topology with respect to the middle level, allowing recursive constructions.

3.3.3 AFFT Routing

Previous arts have proven the following two routing capabilities of the fat tree (Leiserson, 1985; Bogdanski, 2014).

- unicasting on arbitrary source-destination pair using lowest common ancestor routing algorithm. Specifically, the lowest common ancestor (one switch in the middle stage in Fig. 6) is specified first. Then one path is set up by tracing the path from the lowest common ancestor to both the source and destination.

Table 2. Comparison with State-of-the-art Implementation

	LAMBDA	DPUCZDX8G (B1024)	ratio
LUT	25406	33796	0.75
Register	46508	48144	0.97
BRAM	320	104	3.07
URAM	8	0	
DSP	576	230	2.5
PeakOps/cycle	1024	1024	1
GFlops(100MHz)	102.4	102.4	1

- conjectured to allow rearrangeable multicasting from the pair of multiple sources and a specific destination, and vice versa. However, there are no efficient algorithms to determine the path for multicasting.

Therefore, we adopt an oblivious routing algorithm to obliviously select the middle switch for unicasting, and an exhaustive sweeping of middle switches to find out the feasible routing for required multicasting. Fig. 6 illustrates one multicasting case, where the input data sharing the same color get reduced during traverse inside AFFT and get routed to the specific output port connected to the target buffer.

4 Preliminary Evaluation

We compare **LAMBDA** against Xilinx DPU using end-to-end latency on the same Xilinx ZCU 104 (5W) with resources shown in Tab. 2, where high DSP consumption could be saved through multiplication width tuning.

We configure **LAMBDA** to run two selected dataflows and it achieves 0.5∼1.95× faster than Xilinx DPU (Fig. 5). The performance benefits come from the changing of dataflows across different layers, while the inefficiency comes from the extra padding required for layers with small sizes, which could be improved by picking better dataflows.

5 Conclusion

In this paper, we propose **LAMBDA**, the first accelerator design that enables switching optimal dataflows for different workloads with negligible costs. **LAMBDA** features a novel reduction network called Additive Folded Fat Tree (AFFT) that enables data reordering in data reduction (RIR). This allows each result to be directly routed to its new location in the layout required by the next dataflow, without incurring explicit latency overhead. We believe it could achieve optimal theoretical performance through executing purely optimal dataflows as well as enable dataflow explorations.

REFERENCES

Bogdanski, B. Optimized routing for fat-tree topologies. *Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo, Norway*, 2014.

Chen, Y.-H., Yang, T.-J., Emer, J., and Sze, V. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *arXiv preprint arXiv:1807.07928*, 2018.

Dey, T. K., Shi, D., and Wang, Y. Simba: An efficient tool for approximating rips-filtration persistence via simplicial batch collapse. *ACM J. Exp. Algorithmics*, 24, jan 2019. ISSN 1084-6654. doi: 10.1145/3284360. URL https://doi.org/10.1145/3284360.

Genc, H., Haj-Ali, A., Iyer, V., Amid, A., Mao, H., Wright, J. C., Schmidt, C., Zhao, J., Ou, A. J., Banister, M., Shao, Y. S., Nikolic, B., Stoica, I., and Asanovic, K. Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures. *CoRR*, abs/1911.09925, 2019. URL http://arxiv.org/abs/1911.09925.

Hegde, K., Asghari-Moghaddam, H., Pellauer, M., Crago, N., Jaleel, A., Solomonik, E., Emer, J., and Fletcher, C. W. Extensor: An accelerator for sparse tensor algebra. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, pp. 319–333, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369381. doi: 10.1145/3352460.3358275. URL https://doi.org/10.1145/3352460.3358275.

Kao, S.-C. and Krishna, T. Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm. In *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380263. doi: 10.1145/3400302.3415639. URL https://doi.org/10.1145/3400302.3415639.

Kao, S.-C., Pellauer, M., Parashar, A., and Krishna, T. Digamma: Domain-aware genetic algorithm for hw-mapping co-optimization for dnn accelerators. In *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 232–237, 2022. doi: 10.23919/DATE54114.2022.9774568.

Krishnamoorthi, R. Quantizing deep convolutional networks for efficient inference: A whitepaper, 2018. URL https://arxiv.org/abs/1806.08342.

Kwon, H., Samajdar, A., and Krishna, T. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

Kwon, H., Chatarasi, P., Sarkar, V., Krishna, T., Pellauer, M., and Parashar, A. Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings. *IEEE Micro*, 40(3):20–29, 2020. doi: 10.1109/MM.2020.2985963.

Kwon, H., Pellauer, M., Parashar, A., and Krishna, T. Flexion: A quantitative metric for flexibility in dnn accelerators. *IEEE Computer Architecture Letters*, 20(1):1–4, 2021. doi: 10.1109/LCA.2020.3044607.

Leiserson, C. E. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE transactions on Computers*, 100(10):892–901, 1985.

NVIDIA. NVIDIA Deep Learning Accelerator (NVDLA), 2016. URL http://nvdla.org/primer.html.

Qin, E., Samajdar, A., Kwon, H., Nadella, V., Srinivasan, S., Das, D., Kaul, B., and Krishna, T. Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 58–70, 2020. doi: 10.1109/HPCA47549.2020.00015.

Reuther, A., Michaleas, P., Jones, M., Gadepally, V., Samsi, S., and Kepner, J. Ai and ml accelerator survey and trends, 2022. URL https://arxiv.org/abs/2210.04055.

Samajdar, A., Pellauer, M., and Krishna, T. Self-adaptive reconfigurable arrays (sara): Using ml to assist scaling gemm acceleration. *ArXiv*, abs/2101.04799, 2021.

Wang, D., Wu, T., Tan, H., and Li, H. An inference accelerator design for sparse convolution neural network. In *2021 IEEE International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI)*, pp. 498–515, 2021. doi: 10.1109/CEI52496.2021.9574454.

Xilinx. Xilinx Deep Learning Unit (DPU), 2022. URL https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Deep-Learning-Processor-Unit.

Zhou, X., Du, Z., Guo, Q., Liu, S., Liu, C., Wang, C., Zhou, X., Li, L., Chen, T., and Chen, Y. Cambricon-s: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 15–28, 2018. doi: 10.1109/MICRO.2018.00011.