DMAZERUNNER: OPTIMIZING CONVOLUTIONS ON DATAFLOW ACCELERATORS

Shail Dave⋆ *Aviral Shrivastava*⋆ *Youngbin Kim*† *Sasikanth Avancha* ‡ *Kyoungwoo Lee*†

⋆ Compiler Microarchitecture Lab, Arizona State University
† Department of Computer Science, Yonsei University
‡ Parallel Computing Lab, Intel Labs

ABSTRACT

Convolution neural networks (CNNs) can be efficiently executed on dataflow accelerators. However, the vast space of executing convolutions on computational and memory resources of accelerators makes difficult for programmers to automatically and efficiently accelerate the convolutions and for architects to achieve efficient accelerator designs. We propose dMazeRunner framework, which allows users to optimize execution methods for accelerating convolution and matrix multiplication on a given architecture and to explore dataflow accelerator designs for efficiently executing CNN models. dMazeRunner determines efficient dataflows tailored for CNN layers and achieves efficient execution methods for CNN models within several seconds.

***Index Terms*—** Hardware accelerators, energy-efficiency, mapping, deep learning, design space exploration

1. INTRODUCTION

CNNs are widely used in several important domains including image recognition, object detection, media generation, and video analysis [1, 2, 3, 4]. CNNs exhibit many convolution and a few fully-connected (FC) layers; execution time is majorly spent in convolutions [5]. With advances in computing systems, it has become feasible to deploy CNNs for quick and accurate classification of even high-resolution images.

Many energy-efficient accelerators are being developed to execute CNNs with high throughput. In particular, many variations of dataflow accelerators, including systolic arrays, coarse-grained reconfigurable arrays, and spatial architectures have been shown as effective for accelerating CNNs [3, 5, 6, 7]. As shown in Fig. 1(c), dataflow accelerators comprise of an array of processing elements (PEs) with private register files (RFs) and a shared scratchpad memory (SPM). Since PEs are much simple in design (function units with little local control), and the scratchpad is non-coherent, these accelerators are a few orders of magnitude power-efficient than out-of-order CPU or GPU cores [3, 5]. Private and shared memories of PEs enable very high reuse of data, and through efficient data management, PEs can be engaged in continuous computations while data is communicated via memories

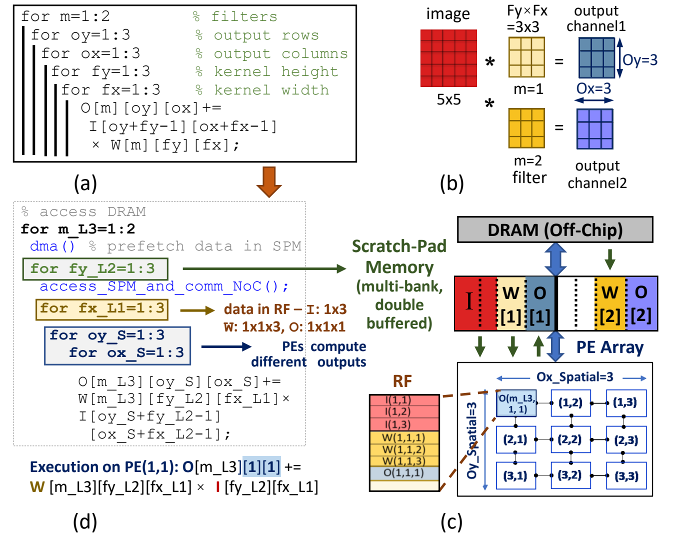

Fig. 1. (a)–(b) Convolution of a 5×5 image (single channel) with 3×3 weights of 2 filters. (c) Dataflow accelerator with 3×3 PEs accessing 16B RFs and 256B shared SPM (8 banks). (d) Spatiotemporal execution of loops on the accelerator.

[5, 6]. Thus, with minimized execution time, dataflow accelerators yield very high throughput and low latency.

The vast mapping space and accelerator design space make it challenging for programmers to determine efficient ways of executing various convolution layers on a single accelerator and for hardware designers to explore efficient architecture for accelerating multiple layers of different CNN models. This is because, convolution layers feature 7-deep nested loops, exhibiting many ways of the data reuse and spatial execution. For example, PEs can process different subsets of the computational graph (e.g., different outputs in an output stationary dataflow [`Oy|Ox`] of Fig. 1(c)–(d), with unrolling of `Oy` and `Ox` loops on PEs for spatial execution [6, 8]). Moreover, the accesses of data tensors from the on-chip scratchpad (L2) and off-chip (L3) memory can be scheduled in many ways, impacting the reuse of the data available in the registers and SPM [6, 7, 8, 9]. Thus, it opens up many implementation choices for architecture design in-

cluding varying sizes and configurations of PEs, RFs, SPM, and for each design, many ways to execute the loops both spatially and temporally onto the computational and memory resources of the accelerator [5, 6, 8].

These different ways or "*execution methods*" significantly impact how PEs process different subset of the tensor data, data accessed from memories, data communication via interconnect, etc. [5, 6, 8], and therefore, have a dramatic impact on the energy consumption and execution time [7]. Due to the lack of a tool for systematic and efficient exploration of the vast design space, experts have considered only certain ways like row-stationary or output-stationary dataflow mechanisms [5, 10, 11], or explored a tiny fraction of the space during manual optimizations [9] or randomization-based explorations [12]. This may not always be very efficient for accelerating convolution layers of different shapes and tensors of different sizes. Moreover, tools [7, 11, 13] analytically evaluate dataflow acceleration, but they either do not optimize the execution time of convolutions or lack a detailed performance model which accounts for the miss penalty and stall cycles for PEs due to data communication via memories and interconnect. Lastly, [14, 15, 16] employ software pipelining and achieve instruction-level parallelism while executing loops on dataflow accelerators with a few PEs. However, without exploring abundant data- and thread-level parallelisms, they may not efficiently map loops on accelerators with large PE-arrays accessing larger memories. So, it is crucial to automatically optimize the execution method for efficient acceleration of a convolution layer on a target dataflow accelerator and to explore efficient accelerator design for different convolution layers of multiple CNN models [5, 6].

We propose dMazeRunner framework that automatically and efficiently optimizes the execution of different convolution and FC layers on dataflow accelerators. In particular,
(i) Users can specify the targeted convolution or matrix multiplication operation and accelerator architecture specification and the execution method of their choice, for which the framework provides estimations of the execution metrics.
(ii) dMazeRunner automatically optimizes execution methods for a specific or all convolution and FC layers of DNN models, such that even non-expert programmers can quickly explore the execution space for an accelerator architecture. Several optimizations to prune the search space and multithreaded implementation enables the explorations of optimized execution methods within a few seconds.
(iii) dMazeRunner allows designers to explore efficient accelerator designs for various convolution layers of CNN models.
(iv) Leveraging the TVM environment [17], our framework supports optimizations for CNN models from multiple machine learning libraries like MXNet and Keras. It is available at `https://github.com/cmlasu/dMazeRunner`.

Our experiments on different convolution layers from the widely used ResNe(X)t model demonstrate that dMazeRunner can optimize execution of convolutions on a dataflow accelerator with various dataflow mechanisms; achieved execution methods exhibit higher architectural resource utilization, reuse of multiple data tensors in memories, low accesses to off-chip memory, with almost entire execution time spent in performing useful computations on PEs. Using dMazeRunner, users can automatically optimize execution methods for convolutions of CNN models within a few seconds and explore efficient designs of dataflow accelerators.

2. DMAZERUNNER

2.1. Analyzing Execution Methods for Convolutions

dMazeRunner allows programmers to define parameters of their convolution or FC layer (tensor sizes, strides, batch size), accelerator architecture, and the execution method of their choice (tiling and ordering of loops) for managing spatiotemporal execution. Then, it analyzes these inputs through an analytical model (proposed in [8]) and provides the estimation of execution metrics i.e., execution cycles, energy consumption, and energy-delay product. Since domain-specific dataflow accelerator designs are simpler, it is feasible to achieve a near-actual estimate of the execution metrics by considering the execution patterns throughout the accelerator architecture. From the specified convolution parameters and architecture, dMazeRunner determines the computation and communication patterns, including reuse of the data in memories and among PEs, accesses to off-chip memory, miss penalty, data distribution via network-on-chip, and inter-PE communication for reduction operations. Users can vary architecture specification in terms of organization of the PEs, sizes and configurations of the private and shared memories for PEs, interconnect, and direct memory access (DMA) model. Thus, programmers and domain-experts can explore the impact of different mappings of the various convolution layers onto dataflow accelerators.

2.2. Optimizing Accelerations of CNN Models

With the vast space of execution methods, it is hard for non-expert programmers to figure out optimized execution methods for efficient accelerations. Moreover, depending on the model, its depth (total layers) and shape of each convolution layer (number of channels, aka width, and height and width of the image or filter data, aka resolution) vary significantly [18, 19]. With continuous advances in DNN model development, it becomes challenging for even domain experts to quickly explore efficient execution methods or accelerator designs for new models.

dMazeRunner facilitates optimizing execution methods so that the programmers and architects can determine efficient execution methods for specified convolution layers as well as for the entire CNN models from machine learning libraries like MXNet and Keras. Using analytical models, it quickly determines the effectiveness of many execution

methods. dMazeRunner's auto-optimizer employs a search reduction heuristic [8], which focuses on highly efficient methods that make high utilization of architectural resources of the accelerator and require fewer accesses to off-chip memory. Moreover, its multi-threaded implementation along with caching of the commonly invoked routines yield quick optimizations. For example, on an Intel i7-6700 quad-core platform, dMazeRunner optimizes execution of different convolution layers of ResNet [1] in a second or a few and AlexNet and ResNet18 models in about 18 and 180 seconds, respectively. It outputs the execution methods which are optimized for minimizing execution time, energy, and energy-delay-product (EDP), along with the corresponding estimates of the execution metrics.

Furthermore, dMazeRunner does not preclude the experienced programmers from performing a directed exploration of the search space, but rather enables a quick and systematic search. It provides support for a few common in-built optimization strategies for experts to flexibly explore the space.

2.3. Exploring Efficient Accelerator Designs

dMazeRunner framework allows fast design space exploration (DSE) so that the accelerator designers can quickly land upon better architectural design solutions, e.g., fine-tuning the memory sizes. This is because, with analytical models, dMazeRunner can quickly explore efficient execution methods for executing convolutions on a variety of dataflow accelerator architectures. For example, for DSE, users can specify variations in sizes of register files, scratch-pad memory, and total PEs, and can explore the implications of different designs. The framework processes each design point and evaluates it by finding optimized execution methods and provides estimates of the execution metrics for the designs. Then, dMazeRunner outputs information about design variations, EDP, total mappings evaluated for a design point, and the time taken to evaluate each accelerator design.

3. EXPERIMENTAL METHODOLOGY

CNN models: For evaluating the execution methods optimized by dMazeRunner, we consider various convolution layers from widely used CNN models ResNet [1] and ResNeXt for classifying images from ImageNet dataset. We executed models on dMazeRunner with a batch of 4 images.

Specification of target accelerator: Our dataflow accelerator architecture features 16×16 PEs with 16-bit precision. PEs access private 1024B double-buffered RFs and a 128 kB shared double-buffered scratch-pad [20]. Each pipelined PE features a 2-stage multiplier and an adder. The accelerator exhibits four single-cycle multi-cast networks [5] for communicating the tensor data to PEs and one such interconnect enables inter-PE communication to perform reduction operations. The global scratchpad memory houses 64 banks (2 kB each) for storing the data of tensors. DMA controller communicates the data between DRAM and the scratchpad memory. Our DMA latency model for data transfers is the same as Cell processors that featured scratchpads [21]. We model the energy consumption of accelerator resources as per hardware evaluations by Yang et al. [6] for a 28 nm technology.

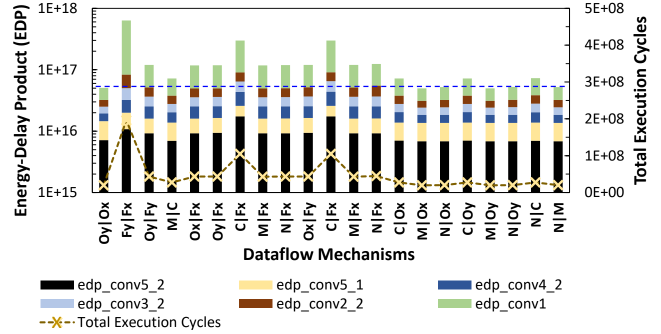

Fig. 2. For ResNet layers, dMazeRunner achieves efficient execution methods with multiple dataflow mechanisms.

4. RESULTS AND ANALYSIS

4.1. Validation

Using the same architecture specification, we obtained the optimized execution methods through DNN optimizer of [6] and evaluated our model for accelerating ResNet conv5_2. For different dataflow execution methods, our analytical model achieved the same PE utilization as Yang et al. [6, 13], and the energy consumption (in mJ) for estimated by dMazeRunner closely matched the energy estimated by [13] with a difference of 4.19% on average. In fact, for highly optimized execution methods, the energy estimation for register accesses was the major component, while the energy estimation for off-chip memory accesses was very low.

4.2. Optimized Dataflow Executions

Fig. 2 demonstrates the evaluation of optimized execution methods for different dataflow mechanisms. The primary axis shows the EDP of each convolution layer and the secondary axis shows the total execution cycles for these six layers (lower the better). For better visualization, we plot EDP results on a logarithmic scale.

dMazeRunner can optimize the execution of convolution layers for multiple dataflow mechanisms, which can be visualized in Fig. 2 (bars of nearly the same height). There are two reasons: (i) Highly optimized execution methods exhibit common characteristics [6] including high utilization of architectural resources (PEs, registers, scratchpad), higher reuse of different data tensors at different levels of memory hierarchy, lower accesses to off-chip memory, efficiently inter-

Table 1. dMazeRunner drastically reduces the vast space of valid execution methods for optimizing DNNs acceleration.

ResNet Conv Layers	Loop Tilings	Loop Orderings	dMzRnr explored Tilings	dMzRnr explored Orderings
1	1.2E+08	7!×7!	46812	3×3
2_2	1.1E+09	7!×7!	122092	3×3
3_2	6.2E+08	7!×7!	53690	3×3
4_2	1.8E+08	7!×7!	10938	3×3
5_1	1.4E+07	7!×7!	877	3×3
5_2	1.7E+07	7!×7!	753	3×3

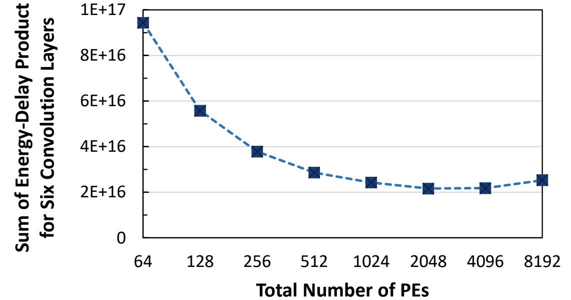

Fig. 3. Optimizing designs for different convolution layers.

leaved computation with communication of data (such that PEs do not need to stop computation while waiting for newer data to process), etc. (ii) For efficient spatiotemporal execution, often execution methods feature spatial execution of multiple loops, which leads to the similar or the same execution methods for multiple dataflow mechanisms.

For example, for executing ResNet conv5_2 with output stationary dataflow, dMazeRunner's execution method allocated 36 elements of image tensor (I), 144 elements of weights (W), and 64 elements for output tensor (O) (244 from 256) in RFs, achieving image and weight reuse through processing the data of 4 images and 16 filters simultaneously. Cycles for performing *Multiply and ACcumulate (MAC)* operations on this data in the registers were 576, which perfectly interleaved with the communication latency of 576 cycles (to communicate $4\times1\times9\times9$ = 324 elements of I and $64\times1\times3\times3$ = 576 elements of W via different interconnects). Data elements of O ($4\times64\times7\times7$ = 12,544) were reused in RFs by accumulating partial summation during 8 consequent execution passes. Thus, execution pass latency was 576 cycles, and 8 passes processed the data from the scratchpad memory (4608 cycles). Again, this execution was efficiently interleaved with the latency to communicate the data of I and W tensors from off-chip memory to a double-buffered SPM via DMA transfers (3641 cycles). Considering the stall cycles for writing back the data of output tensor, total execution cycles were 2,459,648, which exceeded an ideal execution time (2,359,296 cycles for executing 462,422,016 MACs on $4\times7\times7$ PEs) by a mere 4%. Thus, dMazeRunner achieves highly optimized and non-intuitive mappings that account for multiple factors attributing to efficient accelerations.

4.3. Efficient Exploration of Execution Methods

Table 1 shows the valid execution methods (brute-force search) and the methods explored by dMazeRunner. These execution methods feature different ways of tiling the loops for spatiotemporal execution and loop orderings (per each way of tiling) for scheduling data transfers. dMazeRunner drastically pruned orderings of the loops from 7!×7! to 3×3. Moreover, its search reduction heuristics targeted highly optimized execution methods that exhibited at least 80% utilization of PEs, 80% of RFs, and 50% of SPM. dMazeRunner also discarded execution methods that required many non-contiguous data accesses to off-chip memory and inter-PE communication for the spatial accumulation of output tensor. Overall, for optimizing executions of these six convolution layers, dMazeRunner reduced exploration of the total ways of tiling the loops by 9020×, re-ordering the loops by 2.82E+06 times, and execution methods by 2.55E+10 times.

4.4. Design Space Exploration

To explore efficient designs, we varied the sizes of the on-chip memories and PE-array (varying a parameter at a time). We limited total on-chip memory up to 512 kB for RFs (a total for all PEs) and 8 MB for the shared SPM. Fig. 3 shows the achieved EDP (summation of the EDP for each of six convolution layers) when PEs vary from 64 to 8192; memory sizes are optimized for each configuration of the PE-array size.

When total PEs are considerably small, scaling of the EDP is faster, since an efficient mapping would almost linearly increase the throughput. The increase is marginal beyond the total of 1024 PEs. For the designs with 8192 or more PEs, the EDP summation increases because efficient partitioning of the spatiotemporal execution of loops becomes challenging with considerably larger arrays; poor resource allocation can easily impact the EDP. Similarly, the communication requirements increase considerably for a larger PE array, which directly impacts the overall execution time.

Acknowledgements

This research was partially supported by funding from NSF grant CCF 1723476 - NSF/Intel joint research center for Computer Assisted Programming for Heterogeneous Architectures (CAPA), and from grants NRF-2015M3C4A7065522 and 2014-3-00035, funded by MSIT. Any opinions, findings, and conclusions presented in this material are those of the authors and do not necessarily reflect the views of their employers or the sponsoring agencies.

5. REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, et al., "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications," *arXiv preprint arXiv:1811.09886*, 2018.

[3] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, et al., "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2017, pp. 1–12.

[4] Yann LeCun, "1.1 deep learning hardware: Past, present, and future," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2019, pp. 12–19.

[5] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[6] Xuan Yang, Mingyu Gao, Jing Pu, Ankita Nayak, Qiaoyi Liu, Steven Emberton Bell, Jeff Ou Setter, Kaidi Cao, Heonjae Ha, Christos Kozyrakis, et al., "Dnn dataflow choice is overrated," *arXiv preprint arXiv:1809.04070*, 2018.

[7] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 304–315.

[8] Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava, "Dmazerunner: Executing perfectly nested loops on dataflow accelerators," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–27, 2019.

[9] Hongbo Rong, "Programmatic control of a compiler for generating high-performance spatial hardware," *arXiv preprint arXiv:1711.07606*, 2017.

[10] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015, pp. 1737–1746.

[11] "Scale-sim," `https://github.com/ARM-software/SCALE-Sim`.

[12] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 16–25.

[13] Xuan Yang et al., "Dnn energy model and optimizer," `https://github.com/xuanyoya/CNN-blocking/tree/dev`.

[14] Shail Dave, Mahesh Balasubramanian, and Aviral Shrivastava, "Ramp: resource-aware mapping for cgras," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[15] Dhananjaya Wijerathne, Zhaoying Li, Manupa Karunarathne, Anuj Pathania, and Tulika Mitra, "Cascade: High throughput data streaming via decoupled access-execute cgra," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 50, 2019.

[16] Shail Dave, Mahesh Balasubramanian, and Aviral Shrivastava, "Ureca: Unified register file for cgras," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1081–1086.

[17] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Haichen Shen, Eddie Q Yan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy, "Tvm: end-to-end optimization stack for deep learning," *arXiv preprint arXiv:1802.04799*, pp. 1–15, 2018.

[18] Mingxing Tan and Quoc Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, 2019, pp. 6105–6114.

[19] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[20] Yongjoo Kim, Jongeun Lee, Aviral Shrivastava, Jonghee W Yoon, Doosan Cho, and Yunheung Paek, "High throughput data mapping for coarse-grained reconfigurable architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 11, pp. 1599–1609, 2011.

[21] Michael Kistler, Michael Perrone, and Fabrizio Petrini, "Cell multiprocessor communication network: Built for speed," *IEEE micro*, vol. 26, no. 3, pp. 10–23, 2006.