Using Dataflow to Optimize Energy Efficiency of Deep Neural Network Accelerators

THE AUTHORS DEMONSTRATE THE KEY ROLE DATAFLOWS PLAY IN OPTIMIZING ENERGY EFFICIENCY FOR DEEP NEURAL NETWORK (DNN) ACCELERATORS. THEY INTRODUCE BOTH A SYSTEMATIC APPROACH TO ANALYZE THE PROBLEM AND A NEW DATAFLOW, CALLED *ROW-STATIONARY*, THAT IS UP TO 2.5 TIMES MORE ENERGY EFFICIENT THAN EXISTING DATAFLOWS IN PROCESSING A STATE-OF-THE-ART DNN. THIS ARTICLE PROVIDES GUIDELINES FOR FUTURE DNN ACCELERATOR DESIGNS.

Yu-Hsin Chen
Massachusetts Institute of Technology

Joel Emer
Nvidia and Massachusetts Institute of Technology

Vivienne Sze
Massachusetts Institute of Technology

Recent breakthroughs in deep neural networks (DNNs) are leading to an industrial revolution based on AI. The superior accuracy of DNNs, however, comes at the cost of high computational complexity. General-purpose processors no longer deliver sufficient processing throughput and energy efficiency for DNNs. As a result, demands for dedicated DNN accelerators are increasing in order to support the rapidly growing use of AI.

The processing of a DNN mainly comprises multiply-and-accumulate (MAC) operations (see Figure 1). Most of these MACs are performed in the DNN's convolutional layers, in which multichannel filters are convolved with multichannel input feature maps (ifmaps, such as images). This generates partial sums (psums) that are further accumulated into multichannel output feature maps (ofmaps). Because the MAC operations have few data dependencies, DNN accelerators can use high parallelism to achieve high processing throughput. However, this processing also requires a significant amount of data movement: each MAC performs three reads and one write of data access. Because moving data can consume more energy than the computation itself,[1] optimizing data movement becomes key to achieving high energy efficiency.

Data movement can be optimized by exploiting data reuse in a multilevel storage hierarchy. By maximizing the reuse of data in the lower-energy-cost storage levels (such as local scratchpads), thus reducing data accesses to the higher-energy-cost levels (such as DRAM), the overall data movement energy consumption is minimized.

In fact, DNNs present many data reuse opportunities. First, there are three types of input data reuse: *filter reuse,* wherein each filter weight is reused across multiple ifmaps; *ifmap reuse,* wherein each ifmap

Published by the IEEE Computer Society

pixel is reused across multiple filters; and *convolutional reuse,* wherein both ifmap pixels and filter weights are reused due to the sliding-window processing in convolutions. Second, the intermediate psums are reused through the accumulation of ofmaps. If not accumulated and reduced as soon as possible, the psums can pose additional storage pressure.

A design can exploit these data reuse opportunities by finding the optimal MAC operation *mapping,* which determines both the temporal and spatial scheduling of the MACs on a highly parallel architecture. Ideally, data in the lower-cost storage levels is reused by as many MACs as possible before replacement. However, due to the limited amount of local storage, input data reuse (ifmaps and filters) and psum reuse cannot be fully exploited simultaneously. For example, reusing the same input data for multiple MACs generates psums that cannot be accumulated together and, as a result, consume extra storage space. Therefore, the system energy efficiency is maximized only when the mapping balances all types of data reuse in a multilevel storage hierarchy.

The search for the mapping that maximizes system energy efficiency thus becomes an optimization process. This optimization must consider the following factors: the data reuse opportunities available for a given DNN shape and size (for example, the number of filters, number of channels, size of filters, and feature map size), the energy cost of data access at each level of the storage hierarchy, and the available processing parallelism and storage capacity. The first factor is a function of workload, whereas the second and third factors are a function of the specific accelerator implementation.

Because of implementation tradeoffs, previous proposals for DNN accelerators have made choices on the subset of mappings that can be supported. Therefore, for a specific DNN accelerator design, the optimal mapping can be selected only from the subset of supported mappings instead of the entire mapping space. The subset of supported mappings is usually determined by a set of mapping rules, which also characterizes the hardware implementation. Such a set of mapping rules defines a *dataflow.*

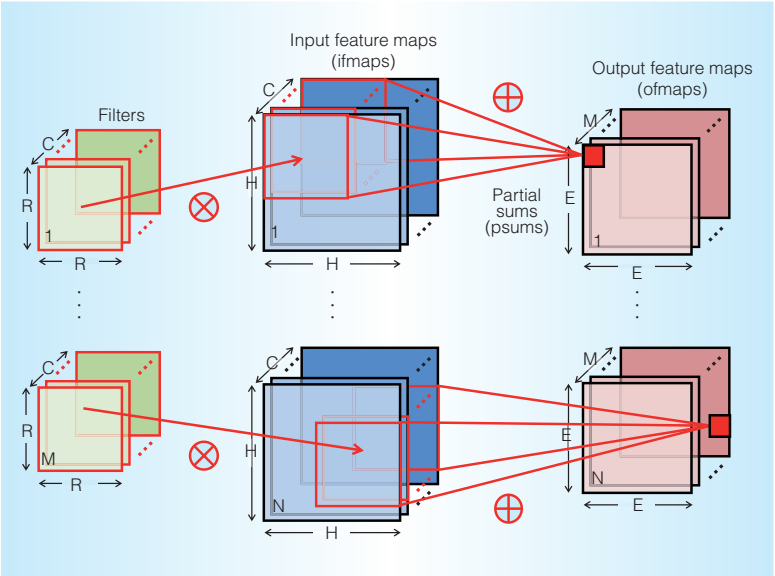

Figure 1. In the processing of a deep neural network (DNN), multichannel filters are convolved with the multichannel input feature maps, which then generate the output feature maps. The processing of a DNN comprises many multiply-and-accumulate (MAC) operations.

Because state-of-the-art DNNs come in a wide range of shapes and sizes, the corresponding optimal mappings also vary. The question is, can we find a dataflow that accommodates the mappings that optimize data movement for various DNN shapes and sizes?

In this article, we explore different DNN dataflows to answer this question in the context of a spatial architecture.[2] In particular, we will present the following key contributions:[3]

- An analogy between DNN accelerators and general-purpose processors that clearly identifies the distinct aspects of operation of a DNN accelerator, which provides insights into opportunities for innovation.
- A framework that quantitatively evaluates the energy consumption of different mappings for different DNN shapes and sizes, which is an essential tool for finding the optimal mapping.
- A taxonomy that classifies existing dataflows from previous DNN accelerator projects, which helps to understand a large body of work despite differences in the lower-level details.
- A new dataflow, called Row-Stationary (RS), which is the first dataflow to

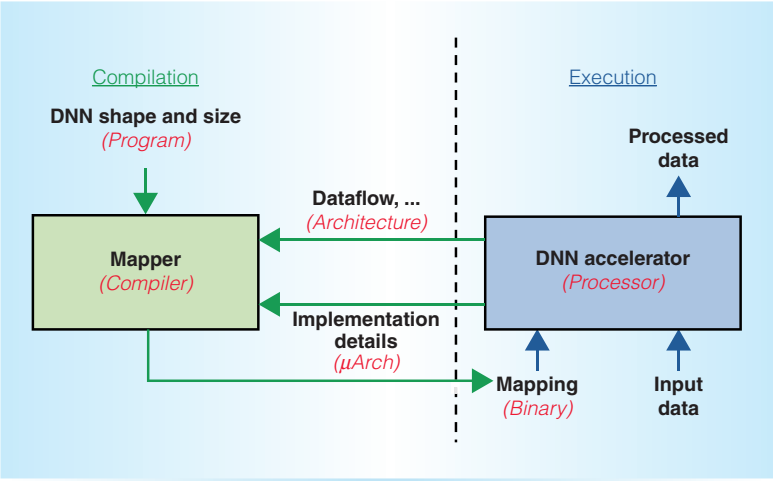

Figure 2. An analogy between the operation of DNN accelerators (roman text) and that of general-purpose processors (italicized text).

optimize data movement for superior system energy efficiency. It has also been verified in a fabricated DNN accelerator chip, Eyeriss.[4]

We evaluate the energy efficiency of the RS dataflow and compare it to other dataflows from the taxonomy. The comparison uses a popular state-of-the-art DNN model, AlexNet,[5] with a fixed amount of hardware resources. Simulation results show that the RS dataflow is 1.4 to 2.5 times more energy efficient than other dataflows in the convolutional layers. It is also at least 1.3 times more energy efficient in the fully connected layers for batch sizes of at least 16. These results will provide guidance for future DNN accelerator designs.

An Analogy to General-Purpose Processors

Figure 2 shows an analogy between the operation of DNN accelerators and general-purpose processors. In conventional computer systems, the compiler translates a program into machine-readable binary codes for execution; in the processing of DNNs, the mapper translates the DNN shape and size into a hardware-compatible mapping for execution. While the compiler usually optimizes for performance, the mapper especially optimizes for energy efficiency.

The dataflow is a key attribute of a DNN accelerator and is analogous to one of the parts of a general-purpose processor's architecture. Similar to the role of an ISA or memory consistency model, the dataflow defines the mapping rules that the mapper must follow in order to generate hardware-compatible mappings. Later in this article, we will introduce several previously proposed dataflows.

Other attributes of a DNN accelerator, such as the storage organization, also are analogous to parts of the general-purpose processor architecture, such as scratchpads or virtual memory. We consider these attributes part of the architecture, instead of microarchitecture, because they may largely remain invariant across implementations. Although, similar to GPUs, the distinction between architecture and microarchitecture is likely to blur for DNN accelerators.

Implementation details, such as those that determine access energy cost at each level of the storage hierarchy and latency between processing elements (PEs), are analogous to the microarchitecture of processors, because a mapping will be valid despite changes in these characteristics. However, they play a vital part in determining a mapping's energy efficiency.

The mapper's goal is to search in the mapping space for the mapping that best optimizes data movement. The size of the entire mapping space is determined by the total number of MACs, which can be calculated from the DNN shape and size. However, only a subset of the space is valid given the mapping rules defined by a dataflow. For example, the dataflow can enforce the following mapping rule: all MACs that use the same filter weight must be mapped on the same PE in the accelerator. Then, it is the mapper's job to find out the exact ordering of these MACs on each PE by evaluating and comparing the energy efficiency of different valid ordering options.

As in conventional compilers, performing evaluation is an integral part of the mapper. The evaluation process takes a certain mapping as input and gives an energy consumption estimation based on the available hardware resources (microarchitecture) and data reuse opportunities extracted from the DNN shape and size (program). In the next section, we will introduce a framework that can perform this evaluation.

Evaluating Energy Consumption

Finding the optimal mapping requires evaluation of the energy consumption for various mapping options. In this article, we evaluate energy consumption based on a spatial architecture,[2] because many of the previous designs can be thought of as instances of such an architecture. The spatial architecture (see Figure 3) consists of an array of PEs and a multilevel storage hierarchy. The PE array provides high parallelism for high throughput, whereas the storage hierarchy can be used to exploit data reuse in a four-level setup (in decreasing energy-cost order): DRAM, global buffer, network-on-chip (NoC, for inter-PE communication), and register file (RF) in the PE as local scratchpads.

In this architecture, we assume all data types can be stored and accessed at any level of the storage hierarchy. Input data for the MAC operations—that is, filter weights and ifmap pixels—are moved from the most expensive level (DRAM) to the lower-cost levels. Ultimately, they are usually delivered from the least expensive level (RF) to the arithmetic logic unit (ALU) for computation. The results from the ALU—that is, psums—generally move in the opposite direction. The orchestration of this movement is determined by the mappings for a specific DNN shape and size under the mapping rule constraints of a specific dataflow architecture.

Given a specific mapping, the system energy consumption is estimated by accounting for the number of times each data value from all data types (ifmaps, filters, psums) is reused at each level of the four-level memory hierarchy, and weighing it with the energy cost of accessing that specific storage level. Figure 4 shows the normalized energy consumption of accessing data from each storage level relative to the computation of a MAC at the ALU. We extracted these numbers from a commercial 65-nm process and used them in our final experiments.

Figure 5 uses a toy example to show how a mapping determines the data reuse at each storage level, and thus the energy consumption, in a three-PE setup. In this example, we have the following assumptions: each ifmap pixel is used by 24 MACs, all ifmap pixels can fit into the global buffer, and the RF of each PE can hold only one ifmap pixel at a time. The mapping first reads an ifmap pixel from DRAM to the global buffer, then from the global buffer to the RF of each PE through the NoC, and reuses it from the RF for four MACs consecutively in each PE. The mapping then switches to MACs that use other ifmap pixels, so the original one in the RF is replaced by new ones, due to limited capacity. Therefore, the original ifmap pixel must be fetched from the global buffer again

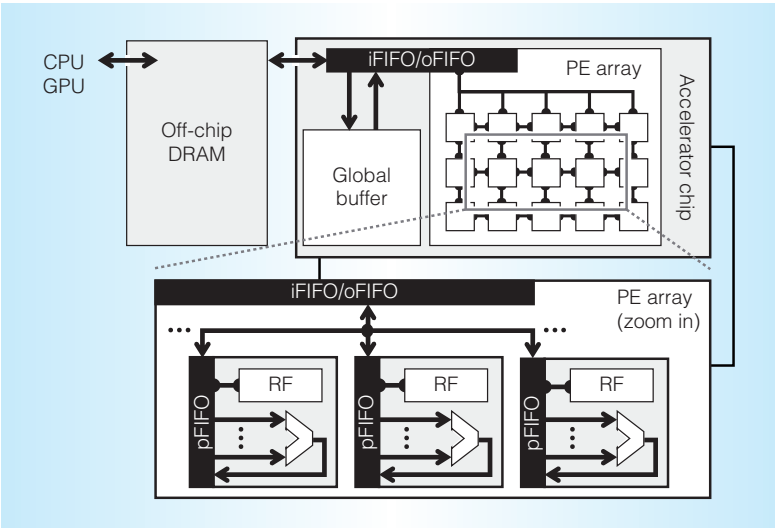

Figure 3. Spatial array architecture comprises an array of processing elements (PEs) and a multilevel storage hierarchy, including the off-chip DRAM, global buffer, network-on-chip (NoC), and register file (RF) in the PE. The off-chip DRAM, global buffer, and PEs in the array can communicate with each other directly through the input and output FIFOs (the iFIFO and oFIFO). Within each PE, the PE FIFO (pFIFO) controls the traffic going in and out of the arithmetic logic unit (ALU), including from the RF or other storage levels.

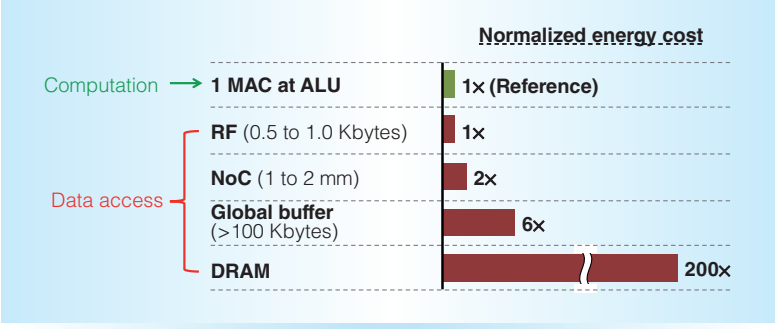

Figure 4. Normalized energy cost relative to the computation of one MAC operation at ALU. Numbers are extracted from a commercial 65-nm process.

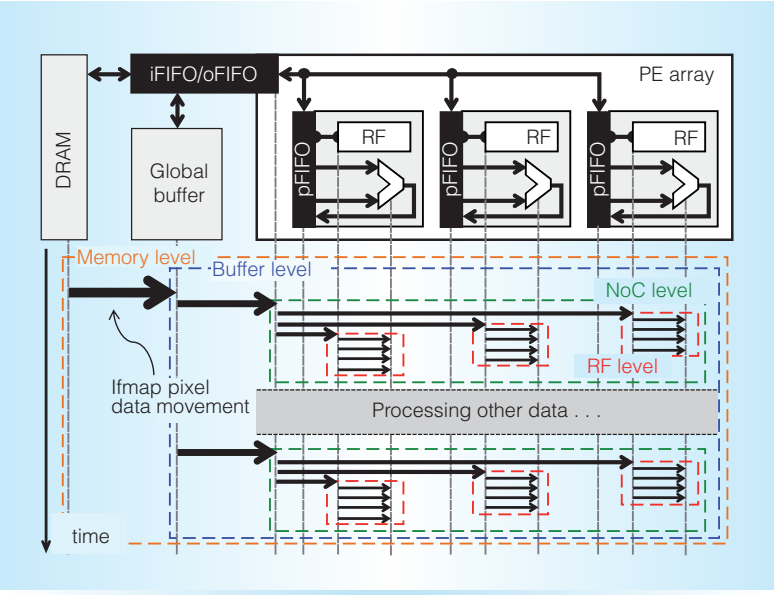

Figure 5. Example of how a mapping determines data reuse at each storage level. This example shows the data movement of one ifmap pixel going through the storage hierarchy. Each arrow means moving data between specific levels (or to an ALU for computation).

when the mapping switches back to the MACs that use it. In this case, the same ifmap pixel is reused at the DRAM, global buffer, NoC, and RF for 1, 2, 6, and 24 times, respectively. The corresponding normalized energy consumption of moving this ifmap pixel is obtained by weighing these numbers with the normalized energy numbers in Figure 4 and then adding them together (that is, $1 \times 200 + 2 \times 6 + 6 \times 2 + 24 \times 1 = 248$). For other data types, the same approach can be applied.

This analysis framework can be used not only to find the optimal mapping for a specific dataflow, but also to evaluate and compare the energy consumption of different dataflows. In the next section, we will describe various existing dataflows.

A Taxonomy of Existing DNN Dataflows

Numerous previous efforts have proposed solutions for DNN acceleration. These designs reflect a variety of trade-offs between performance and implementation complexity. Despite their differences in low-level implementation details, we find that many of them can be described as embodying a set of rules—that is, a dataflow—that defines the valid mapping space based on how they handle data. As a result, we can classify them into a taxonomy.

- The Weight-Stationary (WS) dataflow keeps filter weights stationary in each PE's RF by enforcing the following mapping rule: all MACs that use the same filter weight must be mapped on the same PE for processing serially. This maximizes the convolutional and filter reuse of weights in the RF, thus minimizing the energy consumption of accessing weights (for example, work by Srimat Chakradhar and colleagues[6] and Vinayak Gokhale and colleagues[7]). Figure 6a shows the data movement of a common WS dataflow implementation. While each weight stays in the RF of each PE, the ifmap pixels are broadcast to all PEs, and the generated psums are then accumulated spatially across PEs.
- The Output-Stationary (OS) dataflow keeps psums stationary by accumulating them locally in the RF. The mapping rule is that all MACs that generate psums for the same ofmap pixel must be mapped on the same PE serially. This maximizes psum reuse in the RF, thus minimizing energy consumption of psum movement (for example, work by Zidong Du and colleagues,[8] Suyog Gupta and colleagues,[9] and Maurice Peemen and colleagues[10]). The data movement of a common OS dataflow implementation is to broadcast filter weights while passing ifmap pixels spatially across the PE array (see Figure 6b).
- Unlike the previous two dataflows, which keep a certain data type stationary, the No-Local-Reuse (NLR) dataflow keeps no data stationary locally so it can trade the RF off for a larger global buffer. This is to minimize DRAM access energy consumption by storing more data on-chip (for example, work by Tianshi Chen and colleagues[11] and Chen Zhang and colleagues[12]). The corresponding

mapping rule is that at each processing cycle, all parallel MACs must come from a unique pair of filter and channel. The data movement of the NLR dataflow is to single-cast weights, multicast ifmap pixels, and spatially accumulate psums across the PE array (see Figure 6c).

The three dataflows show distinct data movement patterns, which imply different tradeoffs. First, as Figures 6a and 6b show, the cost for keeping a specific data type stationary is to move the other types of data more. Second, the timing of data accesses also matters. For example, in the WS dataflow, each ifmap pixel read from the global buffer is broadcast to all PEs with properly mapped MACs on the PE array. This is more efficient than reading the same value multiple times from the global buffer and single-casting it to the PEs, which is the case for filter weights in the NLR dataflow (see Figure 6c). Other dataflows can make other tradeoffs. In the next section, we present a new dataflow that takes these factors into account for optimizing energy efficiency.

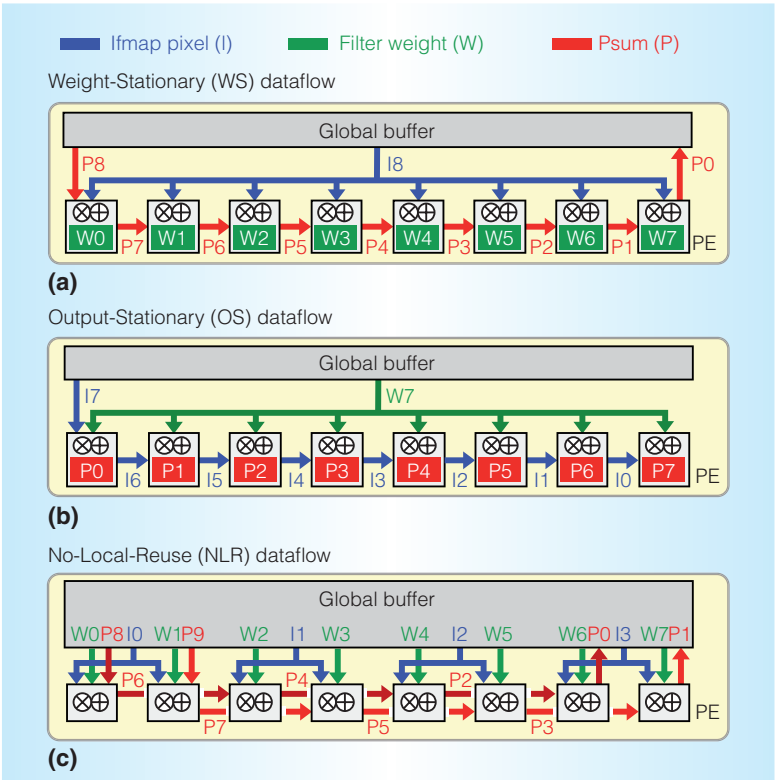

Figure 6. Dataflow taxonomy. (a) Weight Stationary. (b) Output Stationary. (c) No Local Reuse.

An Energy-Efficient Dataflow

Although the dataflows in the taxonomy describe the design of many DNN accelerators, they optimize data movement only for a specific data type (for example, WS for weights) or storage level (NLR for DRAM). In this section, we introduce a new dataflow, called Row-Stationary (RS), which aims to optimize data movement for all data types in all levels of the storage hierarchy of a spatial architecture.

The RS dataflow divides the MACs into *mapping primitives,* each of which comprises a subset of MACs that run on the same PE in a fixed order. Specifically, each mapping primitive performs a 1D row convolution, so we call it a *row primitive,* and intrinsically optimizes data reuse per MAC for all data types combined. Each row primitive is formed with the following rules:

- The MACs for applying a row of filter weights on a row of ifmap pixels, which generate a row of psums, must be mapped on the same PE.
- The ordering of these MACs enables the use of a sliding window for ifmaps, as shown in Figure 7.

Convolutional and psum reuse opportunities within a row primitive are fully exploited in the RF, given sufficient RF storage capacity.

Even with the RS dataflow, as defined by the row primitives, there are still a large number of valid mapping choices. These mapping choices arise both in the spatial and temporal assignment of primitives to PEs:

1. One spatial mapping option is to assign primitives with data rows from the same 2D plane on the PE array, to lay out a 2D convolution (see Figure 8). This mapping fully exploits convolutional and psum reuse opportunities across primitives in the NoC: the same rows of filter weights and ifmap pixels are reused across PEs horizontally and diagonally, respectively; psum rows are

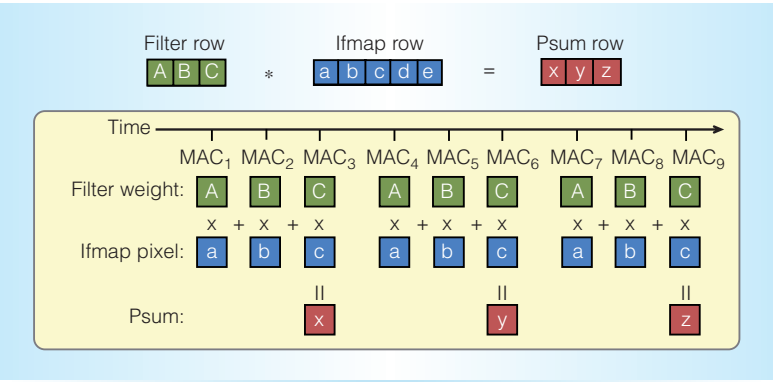

Figure 7. Each row primitive in the Row-Stationary (RS) dataflow runs a 1D row convolution on the same PE in a sliding-window processing order.

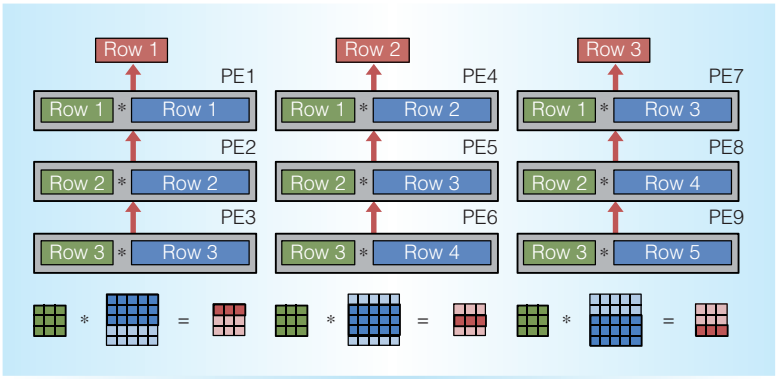

Figure 8. Patterns of how row primitives from the same 2D plane are mapped onto the PE array in the RS dataflow.

further accumulated across PEs vertically.

2. Another spatial mapping option arises when the size of the PE array is large, and the pattern shown in Figure 8 can be spatially duplicated across the PE array for various 2D convolutions. This not only increases utilization of PEs, but also further exploits filter, ifmap, and psum reuse opportunities in the NoC.
3. One temporal mapping option arises when row primitives from different 2D planes can be concatenated or interleaved on the same PE. As Figure 9 shows, primitives with different ifmaps, filters, and channels have filter reuse, ifmap reuse, and psum reuse opportunities, respectively. By concatenating or interleaving their computation together in a PE, it virtually becomes a larger 1D row convolution, which exploits these cross-primitive data reuse opportunities in the RF.
4. Another temporal mapping choice arises when the PE array size is too small, and the originally spatially mapped row primitives must be temporally folded into multiple processing passes (that is, the computation is serialized). In this case, the data reuse opportunities that are originally spatially exploited in the NoC can be temporally exploited by the global buffer to avoid DRAM accesses, given sufficient storage capacity.

As evident from the preceding list, the RS dataflow provides a high degree of mapping flexibility, such as using concatenation, interleaving, duplicating, and folding of the row primitives. The mapper searches for the exact amount to apply each technique in the optimal mapping—for example, how many filters are interleaved on the same PE to exploit ifmap reuse—to minimize overall system energy consumption.

Dataflow Comparison

In this section, we quantitatively compare the energy efficiency of different DNN dataflows in a spatial architecture, including those from the taxonomy and the proposed RS dataflow. We use AlexNet[5] as the benchmarking DNN because it is one of the most popular DNNs available, and it comprises five convolutional (CONV) layers and three fully connected (FC) layers with a wide variety of shapes and sizes, which can more thoroughly evaluate the optimal mappings from each dataflow.

In order to have a fair comparison, we apply the following two constraints for all dataflows. First, the size of the PE array is fixed at 256 for constant processing throughput across dataflows. Second, the total hardware area is also fixed. For example, because the NLR dataflow does not use an RF, it can allocate more area for the global buffer. The corresponding hardware resource parameters are based on the RS dataflow implementation in Eyeriss, a DNN accelerator chip fabricated in 65-nm CMOS.[4] By applying these constraints, we fix the total cost to implement the microarchitecture of each dataflow.

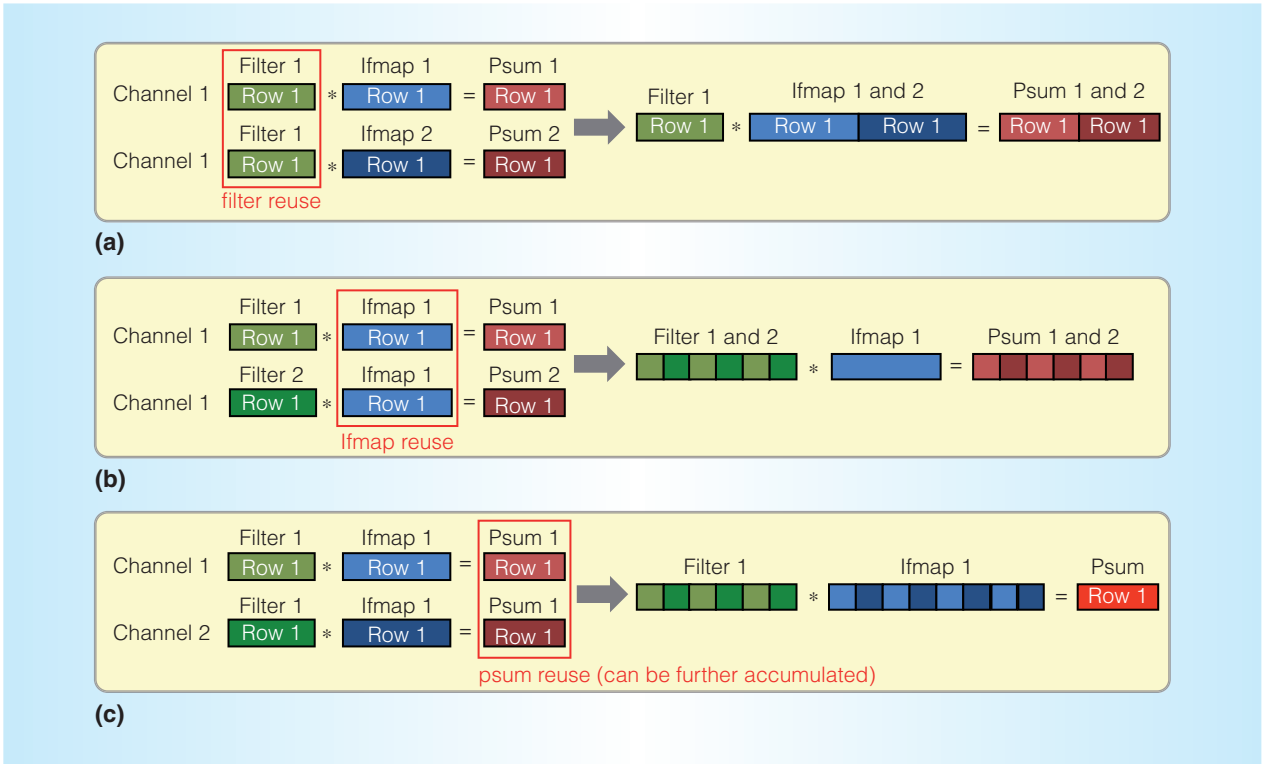

Figure 9. Row primitives from different 2D planes can be combined by concatenating or interleaving their computation on the same PE to further exploit data reuse at the RF level. (a) Two-row primitives reuse the same filter row for different ifmap rows. (b) Two-row primitives reuse the same ifmap row for different filter rows. (c) Two-row primitives from different channels further accumulate psum rows.

Therefore, the differences in energy efficiency are solely from the dataflows.

Figures 10a and 10b show the comparison of energy efficiency between dataflows in the CONV layers of AlexNet with an ifmap batch size of 16. Figure 10a gives the breakdown in terms of storage levels and ALU, and Figure 10b gives the breakdown in terms of data types. First, the ALU energy consumption is only a small fraction of the total energy consumption, which proves the importance of data movement optimization. Second, even though NLR has the lowest energy consumption in DRAM, its total energy consumption is still high, because most of the data accesses come from the global buffer, which are more expensive than those from the NoC or RF. Third, although WS and OS dataflows clearly optimize the energy consumption of accessing weights and psums, respectively, they sacrifice the energy consumption of moving other data types, and therefore do not achieve the lowest overall energy consumption. This shows that DRAM alone does not dictate energy efficiency, and optimizing the energy consumption for only a certain data type does not lead to the best system energy efficiency. Overall, the RS dataflow is 1.4 to 2.5 times more energy efficient than other dataflows in the CONV layers of AlexNet.

Figure 11 shows the same experiment results as in Figure 10b, except that it is for the FC layers of AlexNet. Compared to the CONV layers, the FC layers have no convolutional reuse and use much more filter weights. Still, the RS dataflow is at least 1.3 times more energy efficient than the other dataflows, which proves that the capability to optimize data movement for all data types is the key to achieving the highest overall energy efficiency. Note that the FC layers account for less than 20 percent of the total energy consumption in AlexNet. In recent DNNs, the number of FC layers have also been greatly reduced, making their energy consumption even less significant.

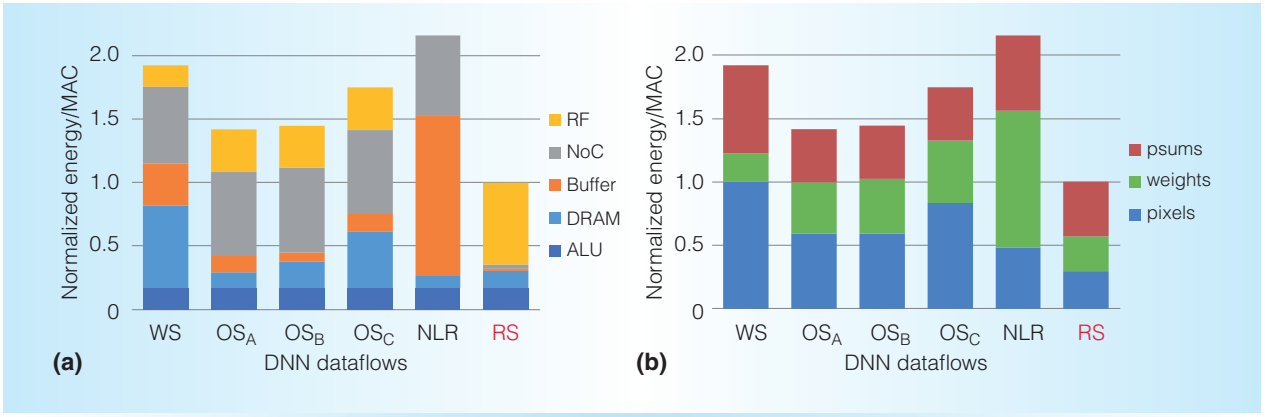

Figure 10. Comparison of energy efficiency between different dataflows in the convolutional (CONV) layers of AlexNet.[5] (a) Breakdown in terms of storage levels and ALU versus (b) data types. OS_A, OS_B, and OS_C are three variants of the OS dataflow that are commonly seen in different implementations.[3]

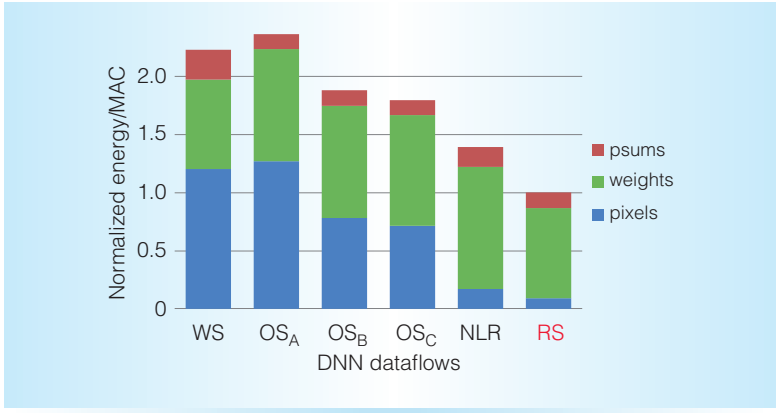

Figure 11. Comparison of energy efficiency between different dataflows in the fully connected (FC) layers of AlexNet.

Research on architectures for DNN accelerators has become very popular for its promising performance and wide applicability. This article has demonstrated the key role of dataflows in DNN accelerator design, and it shows how to systematically exploit all types of data reuse in a multilevel storage hierarchy for optimizing energy efficiency with a new dataflow. It challenges conventional design approaches, which focus more on optimizing parts of the problem, and shifts it toward a global optimization that considers all relevant metrics.

The taxonomy of dataflows lets us compare high-level design choices irrespective of low-level implementation details, and thus can be used to guide future designs. Although these dataflows are currently implemented on distinct architectures, it is also possible to come up with a union architecture that can support multiple dataflows simultaneously. The questions are how to choose a combination of dataflows that maximally benefit the search for optimal mappings, and how to support these dataflows with the minimum amount of hardware implementation overhead.

This article has also pointed out how the concept of DNN dataflows and the mapping of a DNN computation onto a dataflow can be viewed as analogous to a general-purpose processor's architecture and compiling onto that architecture. We hope this will open up space for computer architects to approach the design of DNN accelerators by applying the knowledge and techniques from a well-established research field in a more systematic manner, such as methodologies for design abstraction, modularization, and performance evaluation.

For instance, a recent research trend for DNNs is to exploit data statistics. Specifically, different proposals on quantization, pruning, and data representation have all shown promising results on improving the performance of DNNs. Therefore, it is important that new architectures also take advantage of these findings. As compilers for general-purpose processors can take the profile of targeted workloads to further improve the performance of the generated binary, the analogy between general-purpose processors and DNN accelerators suggests that the mapper for DNN accelerators might also take the profile of targeted DNN statistics to further optimize the

generated mappings. This is an endeavor we will leave for future work. MICRO

References

1. M. Horowitz, "Computing's Energy Problem (And What We Can Do About It)," *Proc. IEEE Int'l Solid-State Circuits Conf.* (ISSCC 14), 2014, pp. 10–14.
2. A. Parashar et al., "Triggered Instructions: A Control Paradigm for Spatially-Programmed Architectures," *Proc. 40th Ann. Int'l Symp. Computer Architecture* (ISCA 13), 2013, pp. 142–153.
3. Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," *Proc. ACM/IEEE 43rd Ann. Int'l Symp. Computer Architecture* (ISCA 16), 2016, pp. 367–379.
4. Y.-H. Chen et al., "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *Proc. IEEE Int'l Solid-States Circuits Conf.* (ISSCC 16), 2016, pp. 262–263.
5. A. Krizhevsky, I. Sutskever, and G.E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Proc. 25th Int'l Conf. Neural Information Processing Systems* (NIPS 12), 2012, pp. 1097–1105.
6. S. Chakradhar et al., "A Dynamically Configurable Coprocessor for Convolutional Neural Networks," *Proc. 37th Ann. Int'l Symp. Computer Architecture* (ISCA 10), 2010, pp. 247–257.
7. V. Gokhale et al., "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks," *Proc. IEEE Conf. Computer Vision and Pattern Recognition Workshops* (CVPRW 14), 2014, pp. 696–701.
8. Z. Du et al., "ShiDianNao: Shifting Vision Processing Closer to the Sensor," *Proc. ACM/IEEE 42nd Ann. Int'l Symp. Computer Architecture* (ISCA 15), 2015, pp. 92–104.
9. S. Gupta et al., "Deep Learning with Limited Numerical Precision," *Proc. 32nd Int'l Conf. Machine Learning*, vol. 37, 2015, pp. 1737–1746.
10. M. Peemen et al., "Memory-Centric Accelerator Design for Convolutional Neural Networks," *Proc. IEEE 31st Int'l Conf. Computer Design* (ICCD 13), 2013, pp. 13–19.
11. T. Chen et al., "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," *Proc. 19th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 14), 2014, pp. 269–284.
12. C. Zhang et al., "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," *Proc. ACM/SIGDA Int'l Symp. Field-Programmable Gate Arrays* (FPGA 15), 2015, pp. 161–170.

Yu-Hsin Chen is a PhD student in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. His research interests include energy-efficient multimedia systems, deep learning architectures, and computer vision. Chen received an MS in electrical engineering and computer science from the Massachusetts Institute of Technology. He is a student member of IEEE. Contact him at yhchen@mit.edu.

Joel Emer is a senior distinguished research scientist at Nvidia and a professor of electrical engineering and computer science at the Massachusetts Institute of Technology. His research interests include spatial and parallel architectures, performance modeling, reliability analysis, and memory hierarchies. Emer received a PhD in electrical engineering from the University of Illinois. He is a Fellow of IEEE. Contact him at jsemer@mit.edu.

Vivienne Sze is an assistant professor in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. Her research interests include energy-aware signal processing algorithms and low-power architecture and system design for multimedia applications, such as machine learning, computer vision, and video coding. Sze received a PhD in electrical engineering from the Massachusetts Institute of Technology. She is a senior member of IEEE. Contact her at sze@mit.edu.