Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks

Yu-Hsin Chen*, Joel Emer*† and Vivienne Sze*

*EECS, MIT
Cambridge, MA 02139

†NVIDIA Research, NVIDIA
Westford, MA 01886

*{yhchen, jsemer, sze}@mit.edu

***Abstract*—Deep convolutional neural networks (CNNs) are widely used in modern AI systems for their superior accuracy but at the cost of high computational complexity. The complexity comes from the need to simultaneously process hundreds of filters and channels in the high-dimensional convolutions, which involve a significant amount of data movement. Although highly-parallel compute paradigms, such as SIMD/SIMT, effectively address the computation requirement to achieve high throughput, energy consumption still remains high as data movement can be more expensive than computation. Accordingly, finding a dataflow that supports parallel processing with minimal data movement cost is crucial to achieving energy-efficient CNN processing without compromising accuracy.**

In this paper, we present a novel dataflow, called *row-stationary* (RS), that minimizes data movement energy consumption on a spatial architecture. This is realized by exploiting local data reuse of filter weights and feature map pixels, i.e., activations, in the high-dimensional convolutions, and minimizing data movement of partial sum accumulations. Unlike dataflows used in existing designs, which only reduce certain types of data movement, the proposed RS dataflow can adapt to different CNN shape configurations and reduces all types of data movement through maximally utilizing the processing engine (PE) local storage, direct inter-PE communication and spatial parallelism. To evaluate the energy efficiency of the different dataflows, we propose an analysis framework that compares energy cost under the same hardware area and processing parallelism constraints. Experiments using the CNN configurations of AlexNet show that the proposed RS dataflow is more energy efficient than existing dataflows in both convolutional (1.4× to 2.5×) and fully-connected layers (at least 1.3× for batch size larger than 16). The RS dataflow has also been demonstrated on a fabricated chip, which verifies our energy analysis.

I. Introduction

The recent popularity of deep learning [1], specifically deep convolutional neural networks (CNNs), can be attributed to its ability to achieve unprecedented accuracy for tasks ranging from object recognition [2–5] and detection [6, 7] to scene understanding [8]. These state-of-the-art CNNs [2–5] are orders of magnitude larger than those used in the 1990s [9], requiring up to hundreds of megabytes for filter weight storage and 30k-600k operations per input pixel.

The large size of such networks poses both throughput and energy efficiency challenges to the underlying processing hardware. Convolutions account for over 90% of the CNN operations and dominates runtime [10]. Although these operations can leverage highly-parallel compute paradigms, such as SIMD/SIMT, throughput may not scale accordingly due to the accompanying bandwidth requirement, and the energy consumption remains high as data movement can be more expensive than computation [11–13]. In order to achieve energy-efficient CNN processing without compromising throughput, we need to develop dataflows that support parallel processing with minimal data movement. The differences in data movement energy cost based on where the data is stored also needs to be accounted for. For instance, fetching data from off-chip DRAMs costs orders of magnitude more energy than from on-chip storage [11, 12].

Many previous papers have proposed specialized CNN dataflows on various platforms, including GPU [14], FPGA [15–21], and ASIC [22–26]. However, due to differences in technology, hardware resources and system setup, a direct comparison between different implementations does not provide much insight into the relative energy efficiency of different dataflows. In this paper, we evaluate the energy efficiency of various CNN dataflows on a spatial architecture *under the same hardware resource constraints*, i.e., area, processing parallelism and technology. Based on this evaluation, we will propose *a novel dataflow that maximizes energy efficiency* for CNN acceleration.

To evaluate energy consumption, we categorize the data movements in a spatial architecture into several levels of hierarchy according to their energy cost, and then analyze each dataflow to assess the data movement at each level. This analysis framework provides insights into how each dataflow exploits different types of data movement using various architecture resources. It also offers a quantifiable way to examine the differences in energy efficiency between different dataflows.

Previously proposed dataflows typically optimize a certain type of data movement, such as input data reuse or partial

sum accumulation. Using our analysis framework, we show that a dataflow that exploits *all types* of data reuse, and takes into account the energy cost of data movement at different levels of the storage hierarchy, can deliver significant energy savings. In summary, the main contributions of this work include:

- A taxonomy that classifies existing CNN dataflows from previous research. (Section IV)
- A spatial architecture based on a new CNN dataflow, called *row stationary*, which is optimized for throughput and energy efficiency. It works on both convolutional and fully-connected layers, and optimizes all types of data movement in the storage hierarchy. This dataflow has also been demonstrated on a fabricated chip. (Section V)
- An analysis framework that can quantify the energy efficiency of different CNN dataflows under the same hardware constraints. It can also search for the most energy efficient mapping for each dataflow. The analytical model uses energy/area numbers from a commercial 65nm process and all R/W numbers are exact based on real CNN shape configurations, i.e., AlexNet. (Section VI-C)
- For a variety of CNN dataflows, we present a comparative analysis of the energy costs associated with data movement and the impact of different types of data reuse. (Section VII)

II. Spatial Architecture

Spatial architectures (SAs) are a class of accelerators that can exploit high compute parallelism using direct communication between an array of relatively simple processing engines (PEs). They can be designed or programmed to support different algorithms, which are mapped onto the PEs using specialized dataflows. Compared with SIMD/SIMT architectures, SAs are particularly suitable for applications whose dataflow exhibits producer-consumer relationships or can leverage efficient data sharing among a region of PEs.

SAs come in two flavors: coarse-grained SAs that consist of tiled arrays of ALU-style PEs connected together via on-chip networks [27–29], and fine-grained SAs that are usually in the form of an FPGA. The expected performance advantage and large design space of coarse-grained SAs has inspired much research on the evaluation of its architectures, control schemes, operation scheduling and dataflow models [30–35].

Coarse-grained SAs are currently a very popular implementation choice for specialized CNN accelerators for two reasons. First, the operations in a CNN layer (e.g., convolutional, fully-connected, pooling, etc. Details are described in Section III-A) are uniform and exhibit high parallelism, which can be computed quite naturally with parallel ALU-style PEs. Second, direct inter-PE communication can be used very effectively for (1) passing partial sums to achieve spatially distributed accumulation, or (2) sharing the same input data for parallel computation without incurring higher energy data transfers. ASIC implementations usually deploy dozens to hundreds of PEs and specialize the PE datapath only for CNN computation [22–26]. FPGAs are also used to build CNN accelerators, and these designs usually use integrated DSP slices to construct the PE datapaths [15–21]. However, the challenge in either type of design lies in the exact mapping of the CNN dataflow to the SA, since it has a strong implication on the resulting throughput and energy efficiency.

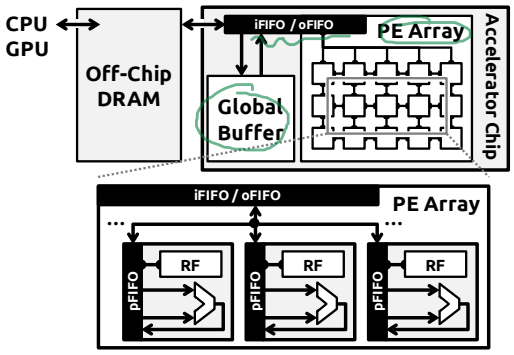


Figure 1. Block diagram of a general CNN accelerator system consisting of a spatial architecture accelerator and an off-chip DRAM. The zoom-in shows the high-level structure of a PE.

Fig. 1 illustrates the high-level block diagram of the accelerator system that is used in this paper for CNN processing. It consists of a SA accelerator and off-chip DRAM. The inputs can be off-loaded from the CPU or GPU to DRAM and processed by the accelerator. The outputs are then written back to DRAM and further interpreted by the main processor.

The SA accelerator is primarily composed of a global buffer and an array of PEs. The DRAM, global buffer and PE array communicate with each other through the input and output FIFOs (iFIFO/oFIFO). The global buffer can be used to exploit input data reuse and hide DRAM access latency, or for the storage of intermediate data. Currently, the typical size of the global buffer used for CNN acceleration is around 100–300kB. The PEs in the array are connected via a network on chip (NoC), and the NoC design depends on the dataflow requirements. The PE includes an ALU datapath, which is capable of doing multiply-and-accumulate (MAC) and addition, a register file (RF) as a local scratchpad, and a PE FIFO (pFIFO) used to control the traffic going in and out of the ALU. Different dataflows require a wide range of RF sizes, ranging from zero to a few hundred bytes. Typical RF size is below 1kB per PE. Overall, the system provides four levels of storage hierarchy for data accesses, including DRAM, global buffer, array (inter-PE communication) and RF. Accessing data from a different level also implies a different energy cost, with the highest cost at DRAM and the lowest cost at RF.

III. CNN Background

A. The Basics

A convolutional neural network (CNN) is constructed by stacking multiple computation layers as a directed acyclic graph [36]. Through the computation of each layer, a higher-level abstraction of the input data, called a feature map (fmap), is extracted to preserve essential yet unique information. Modern CNNs are able to achieve superior performance by employing a very deep hierarchy of layers.

The primary computation of CNN is in the convolutional (CONV) layers, which perform high-dimensional convolutions. From five [2] to even several hundred [5] CONV layers are commonly used in recent CNN models. A CONV layer applies filters on the input fmaps (ifmaps) to extract embedded visual characteristics and generate the output fmaps (ofmaps). The dimensions of both filters and fmaps are 4D: each filter or fmap is a 3D structure consisting of multiple 2D planes, i.e., channels, and a batch of 3D ifmaps is processed by a group of 3D filters in a CONV layer. In addition, there is a 1D bias that is added to the filtering results. Given the shape parameters in Table I, the computation of a CONV layer is defined as

$$\mathbf{O}[z][u][x][y] = \mathbf{B}[u] + \sum_{k=0}^{C-1}\sum_{i=0}^{R-1}\sum_{j=0}^{R-1} \mathbf{I}[z][k][Ux+i][Uy+j] \times \mathbf{W}[u][k][i][j],$$
$$0 \le z < N, 0 \le u < M, 0 \le x,y < E, E = (H-R+U)/U. \quad (1)$$

O, **I**, **W** and **B** are the matrices of the ofmaps, ifmaps, filters and biases, respectively. U is a given stride size. Fig. 2 shows a visualization of this computation (ignoring biases).

A small number, e.g., 3, of fully-connected (FC) layers are typically stacked behind the CONV layers for classification purposes. A FC layer also applies filters on the ifmaps as in the CONV layers, but the filters are of the same size as the ifmaps. Therefore, it does not have the weight sharing property as in CONV layers. Eq. (1) still holds for the computation of FC layers with a few additional constraints on the shape parameters: $H = R$, $E = 1$, and $U = 1$. In between CONV and FC layers, additional layers can be added optionally, such as the pooling (POOL) and normalization (NORM) layers. Each of the CONV and FC layers is also immediately followed by an activation (ACT) layer, such as a rectified linear unit [37].

B. Challenges in CNN Processing

In most of the widely used CNNs, such as AlexNet [2] and VGG16 [3], CONV layers account for over 90% of the overall operations and generate a large amount of data movement. Therefore, they have a significant impact on the throughput and energy efficiency of CNNs. Even though FC layers use most of the filter weights, a recent study has demonstrated that these weights are largely compressible to 1–5% of their original size [38], which greatly reduces the impact of FC layers. Processing of POOL layers can share the same compute scheme used for CONV layers since its computation is a degenerate form of Eq. (1), where the MAC is replaced with a MAX operation. Computation of ACT layers is trivial, and we believe support for the NORM layer can be omitted due to its reduced usage in recent CNNs [3, 5].

Shape Parameter	Description
N	batch size of 3D fmaps
M	# of 3D filters / # of ofmap channels
C	# of ifmap/filter channels
H	ifmap plane width/height
R	filter plane width/height (= H in FC)
E	ofmap plane width/height (= 1 in FC)

Table I
SHAPE PARAMETERS OF A CONV/FC LAYER.

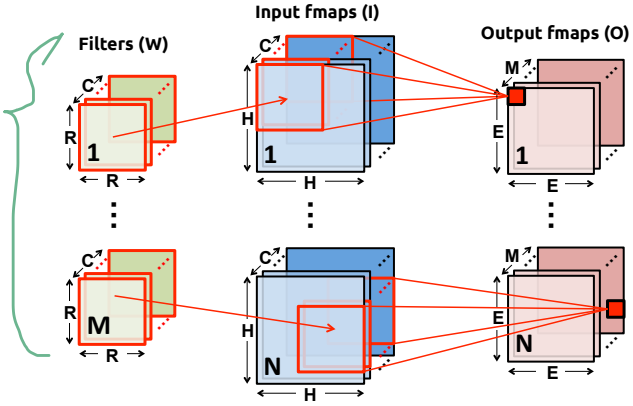


Figure 2. Computation of a CONV/FC layer.

Processing of the CONV and FC layers poses two challenges: *data handling* and *adaptive processing*. The detail of each is described below.

Data Handling: Although the MAC operations in Eq. (1) can run at high parallelism, which greatly benefits throughput, it also creates two issues. First, naïvely reading inputs for all MACs directly from DRAM requires high bandwidth and incurs high energy consumption. Second, a significant amount of intermediate data, i.e., partial sums (psums), are generated by the parallel MACs simultaneously, which poses storage pressure and consumes additional memory R/W energy if not processed, i.e., accumulated, immediately.

Fortunately, the first issue can be alleviated by exploiting different types of *input data reuse*:

- **convolutional reuse**: Due to the weight sharing property in CONV layers, a small amount of unique input data can be shared across many operations. Each filter weight is reused E^2 times in the same ifmap plane, and each ifmap pixel, i.e., activation, is usually reused R^2 times in the same filter plane. FC layers, however, do not have this type of data reuse.

Layer	H[1]	R	E	C	M	U
CONV1	227	11	55	3	96	4
CONV2	31	5	27	48	256	1
CONV3	15	3	13	256	384	1
CONV4	15	3	13	192	384	1
CONV5	15	3	13	192	256	1
FC1	6	6	1	256	4096	1
FC2	1	1	1	4096	4096	1
FC3	1	1	1	4096	1000	1

[1] This is the padded size

Table II
CONV/FC LAYER SHAPE CONFIGURATIONS IN ALEXNET [39].

- **filter reuse**: Each filter weight is further reused across the batch of N ifmaps in both CONV and FC layers.
- **ifmap reuse**: Each ifmap pixel is further reused across M filters (to generate the M output channels) in both CONV and FC layers.

The second issue can be handled by proper *operation scheduling* so that the generated psums can be reduced as soon as possible to save both the storage space and memory R/W energy. CR^2 psums are reduced into one ofmap pixel.

Unfortunately, maximum input data reuse cannot be achieved simultaneously with immediate psum reduction, since the psums generated by MACs using the same filter or ifmap value are not reducible. In order to achieve high throughput and energy efficiency, the underlying CNN dataflow needs to account for both input data reuse and psum accumulation scheduling at the same time.

Adaptive Processing: The many shape parameters shown in Table I gives rise to many possible CONV/FC layer shapes. Even within the same CNN model, each layer can have distinct shape configurations. Table II shows the shape configurations of AlexNet as an example. The hardware architecture, therefore, cannot be hardwired to process only certain shapes. Instead, the dataflow must be efficient for different shapes, and the hardware architecture must be programmable to dynamically map to an efficient dataflow.

C. CNN vs. Image Processing

Before CNNs became mainstream, there was already research on high-efficiency convolution due to its wide applicability in image signal processing (ISP) [40]. Many high-throughput ISP techniques have also been proposed for handling convolutions, including tiling strategies used in multiprocessors and SIMD instructions. However, they are not directly applicable for CNN processing for two reasons:

- The filter weights in CNNs are obtained through training instead of fixed in the processing system. Therefore, they can consume significant I/O bandwidth and on-chip storage, sometimes comparable to that of ifmaps.
- The ISP techniques are developed mainly for 2D convolutions. They do not optimize processing resources for data reuse nor do they address the non-trivial psum accumulation in the 4D convolutions of CNN.

IV. EXISTING CNN DATAFLOWS

Numerous previous efforts [15–26] have proposed solutions for CNN acceleration, but it is difficult to compare their performance directly due to differences in implementation and design choices. In this section, we present a taxonomy of these existing CNN dataflows based on their data handling characteristics. Following are descriptions of these dataflows, which are summarized in Table III.

A. Weight Stationary (WS) Dataflow

Definition: *Each filter weight remains stationary* in the RF to maximize *convolutional reuse* and *filter reuse*. Once a weight is fetched from DRAM to the RF of a PE, the PE runs through all NE^2 operations that use the same filter weight.
Processing: $R \times R$ weights from the same filter and channel are laid out to a region of $R \times R$ PEs and stay *stationary*. Each pixel in an ifmap plane from the same channel is *broadcast* to the same $R \times R$ PEs sequentially, and the psums generated by each PE are further *accumulated spatially* across these PEs. Multiple planes of $R \times R$ weights from different filters and/or channels can be deployed either across multiple $R \times R$ PEs in the array or onto the same $R \times R$ PEs.
Hardware Usage: The RF is used to store the stationary filter weights. Due to the operation scheduling that maximally reuses stationary weights, psums are not always immediately reducible, and will be temporarily stored to the global buffer. If the buffer is not large enough, the number of psums that are generated together has to be limited, and therefore limits the number of filters that can be loaded on-chip at a time.
Examples: Variants of the WS dataflow appear in [15–17, 19, 25, 26].

B. Output Stationary (OS) Dataflow

Definition: The accumulation of *each ofmap pixel stays stationary* in a PE. The psums are stored in the same RF for accumulation to *minimize the psum accumulation cost*.
Processing: This type of dataflow uses the space of the PE array to process a region of the 4D ofmap at a time. To select a region out of the high-dimensional space, there are two choices to make: (1) multiple ofmap channels (MOC) vs. single ofmap channels (SOC), and (2) multiple ofmap-plane pixels (MOP) vs. single ofmap-plane pixels (SOP). This creates three practical OS dataflow subcategories: SOC-MOP, MOC-MOP, and MOC-SOP.

- SOC-MOP is used mainly for CONV layers, and focuses on processing a single plane of ofmap at a time. It further maximizes *convolutional reuse* in addition to psum accumulation.
- MOC-MOP processes multiple ofmap planes with multiple pixels in the same plane at a time. By doing so, it tries to further exploit both *convolutional reuse* and *ifmap reuse*.
- MOC-SOP is used mainly for FC layers, since it processes multiple ofmap channels but with only one

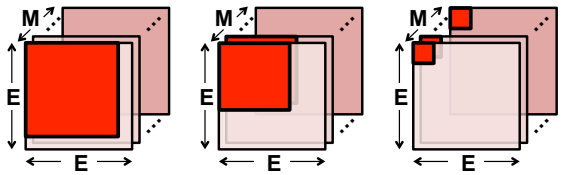


Figure 3. Comparison of the three different OS dataflow variants: (a) SOC-MOP, (b) MOC-MOP, and (c) MOC-SOP. The red blocks depict the ofmap region that the OS dataflow variants process at once.

pixel in a channel at a time. It focuses on further exploiting *ifmap reuse*.

The difference between the three OS dataflows is illustrated in Fig. 3. All additional input data reuse is exploited at the array level, i.e., inter-PE communication. The RF level only handles psum accumulation.

Hardware Usage: All OS dataflows use the RF for psum storage to achieve stationary accumulation. In addition, SOC-MOP and MOC-MOP require additional RF storage for ifmap buffering to exploit convolutional reuse within the PE array.

Examples: A variant of MOC-MOP dataflow appears in [20], and variants of SOC-MOP and MOC-SOP dataflows appear in [23] and [18]. Note that the MOC-MOP variant in [20] does not exploit convolutional data reuse since it simply treats the convolutions as a matrix multiplication.

C. No Local Reuse (NLR) Dataflow

Definition: The NLR dataflow has two major characteristics: (1) it *does not exploit data reuse at the RF level*, and (2) it uses inter-PE communication for *ifmap reuse* and *psum accumulation*.

Processing: NLR divides the PE array into groups of PEs. PEs within the same group read the same ifmap pixel but with different filter weights from the same input channel. Different PE groups read ifmap pixels and filter weights from different input channels. The generated psums are accumulated across PE groups in the whole array.

Hardware Usage: There is no RF storage required by the NLR dataflow. Since the PE array is simply composed of ALU datapaths, it leaves a large area for the global buffer, which is used to store psums as well as input data for reuse.

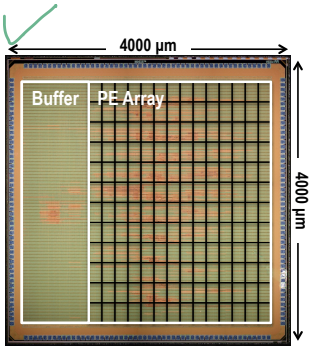
Examples: Variants of the NLR dataflow appear in [21, 22, 24]. In [22], special registers are implemented at the end of each PE array column to hold the psums, which reduces the number of global buffer R/W for psums.

V. Energy-Efficient Dataflow: Row Stationary

While existing dataflows attempt to maximize certain types of input data reuse or minimize the psum accumulation cost, they fail to take all of them into account at once. This results in inefficiency when the layer shape or hardware resources vary. Therefore, it would be desirable if the dataflow could adapt to different conditions and optimize for all types of data movement energy costs. In this section, we will introduce a novel dataflow, called *row stationary* (RS) that achieves this goal. The RS dataflow is a key feature of the Eyeriss architecture, which has been implemented in a fabricated chip [41] (Fig. 4), and whose functionality has been verified using AlexNet.

Dataflow	Data Handling
WS	Maximize *convolutional reuse* and *filter reuse* of weights in the RF.
SOC-MOP OS	Maximize *psum accumulation* in RF. *Convolutional reuse* in array.
MOC-MOP OS	Maximize *psum accumulation* in RF. *Convolutional reuse* and *ifmap reuse* in array.
MOC-SOP OS	Maximize *psum accumulation* in RF. *Ifmap reuse* in array.
NLR	*Psum accumulation* and *ifmap reuse* in array.

Table III
DATA HANDLING COMPARISON BETWEEN EXISTING DATAFLOWS.



Process	65nm CMOS
# of PEs	168
RF Size/PE	0.5 kB
Buffer Size	108 kB
Clock Rate	200 MHz
Precision	16-bit Fixed-Point

Figure 4. Die photo and spec of the Eyeriss chip [41].

A. 1D Convolution Primitives

The implementation of the RS dataflow in Eyeriss is inspired by the idea of applying a strip mining technique in a spatial architecture [42]. It breaks the high-dimensional convolution down into 1D convolution primitives that can run in parallel; each primitive operates on one row of filter weights and one row of ifmap pixels, and generates one row of psums. Psums from different primitives are further accumulated together to generate the ofmap pixels. The inputs to the 1D convolution come from the storage hierarchy, e.g., the global buffer or DRAM.

Each primitive is mapped to one PE for processing; therefore, the computation of *each row pair stays stationary* in the PE, which creates convolutional reuse of filter weights and ifmap pixels at the RF level. An example of this sliding window processing is shown in Fig. 5. However, since the entire convolution usually contains hundreds of thousands of primitives, the exact mapping of all primitives to the PE array is non-trivial, and will greatly affect the energy efficiency.

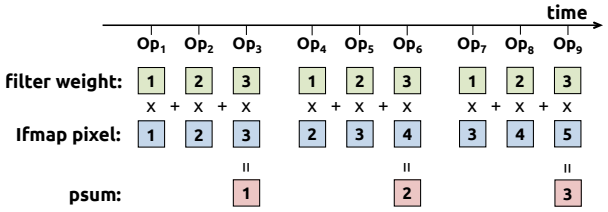


Figure 5. Processing of an 1D convolution primitive in the PE. In this example, $R = 3$ and $H = 5$.

B. Two-Step Primitive Mapping

To solve this problem, the primitive mapping is separated into two steps: logical mapping and physical mapping. The logical mapping first deploys the primitives into a logical PE array, which has the same size as the number of 1D convolution primitives and is usually much larger than the physical PE array in hardware. The physical mapping then *folds* the logical PE array so it fits into the physical PE array. Folding implies serializing the computation, and is determined by the amount of on-chip storage, including both the global buffer and local RF. The two mapping steps happen statically prior to runtime, so no on-line computation is required.

Logical Mapping: Each 1D primitive is first mapped to one logical PE in the logical PE array. Since there is considerable spatial locality between the PEs that compute a 2D convolution in the logical PE array, we group them together as a *logical PE set*. Fig. 6 shows a logical PE set, where each filter row and ifmap row are horizontally and diagonally reused, respectively, and each row of psums is vertically accumulated. The height and width of a logical PE set are determined by the filter height (R) and ofmap height (E), respectively. Since the number of 2D convolutions in a CONV layer is equal to the product of number of ifmap/filter channels (C), number of filters (M) and fmap batch size (N), the logical PE array requires $N \times M \times C$ logical PE sets to complete the processing of an entire CONV layer.

Physical Mapping: Folding means mapping and then running multiple 1D convolution primitives from different logical PEs on the same physical PE. In the RS dataflow, folding is done at the granularity of logical PE sets for two reasons. First, it preserves intra-set convolutional reuse and psum accumulation at the array level (inter-PE communication) as shown in Fig. 6. Second, there exists more data reuse and psum accumulation opportunities across the $N \times M \times C$ sets: the same filter weights can be shared across N sets (filter reuse), the same ifmap pixels can be shared across M sets (ifmap reuse), and the psums across each C sets can be accumulated together. Folding multiple logical PEs from the same position of different sets onto a single physical PE exploits input data reuse and psum accumulation at the RF level; the corresponding 1D convolution primitives run on the same physical PE in an interleaved fashion. Mapping multiple sets spatially across the physical PE array also exploits those opportunities at the array level. The exact amount of logical PE sets to fold and to map spatially at each of the three dimensions, i.e., N, M, and C, are determined by the RF size and physical PE array size, respectively. It then becomes an optimization problem to determine the best folding by using the framework in Section VI-C to evaluate the results.

After the first phase of folding as discussed above, the physical PE array can process a number of logical PE sets, called a *processing pass*. However, a processing pass still may not complete the processing of all sets in the CONV layer. Therefore, a second phase of folding, which is at the granularity of processing passes, is required. Different processing passes run sequentially on the entire physical PE array. The global buffer is used to further exploit input data reuse and store psums across passes. The optimal amount of second phase folding is determined by the global buffer size, and also requires an optimization using the analysis framework.

C. Energy-Efficient Data Handling

To maximize energy efficiency, the RS dataflow is built to optimize all types of data movement by maximizing the usage of the storage hierarchy, starting from the low-cost RF to the higher-cost array and global buffer. The way each level handles data is described as follows.

RF: By running multiple 1D convolution primitives in a PE after the first phase folding, the RF is used to exploit all types of data movements. Specifically, there are *convolutional reuse* within the computation of each primitive, *filter reuse* and *ifmap reuse* due to input data sharing between folded primitives, and *psum accumulation* within each primitive and across primitives.

Array (inter-PE communication): *Convolutional reuse* exists within each set and is completely exhausted up to this level. *Filter reuse* and *ifmap reuse* can be achieved by having multiple sets mapped spatially across the physical PE array. *Psum accumulation* is done within each set as well as across sets that are mapped spatially.

Global Buffer: Depending on its size, the global buffer is used to exploit the rest of *filter reuse*, *ifmap reuse* and *psum accumulation* that remain from the RF and array levels after the second phase folding.

D. Support for Different Layer Types

While the RS dataflow is designed for the processing of high-dimensional convolutions in the CONV layers, it can also support two other layer types naturally:

FC Layer: The computation of FC layers is the same as CONV layers, but without convolutional data reuse. Since the RS dataflow exploits all types of data movement, it can still adapt the hardware resources to cover filter reuse, ifmap reuse

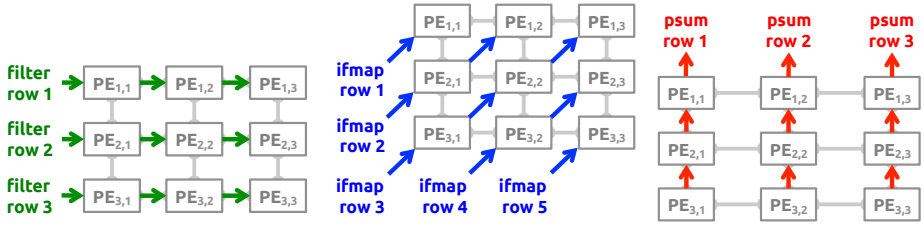


Figure 6. The dataflow in a logical PE set to process a 2D convolution. (a) rows of filter weight are reused across PEs horizontally. (b) rows of ifmap pixel are reused across PEs diagonally. (c) rows of psum are accumulated across PEs vertically. In this example, $R = 3$ and $H = 5$.

and psum accumulation at each level of the storage hierarchy. There is no need to switch between different dataflows as in the case between SOC-MOP and MOC-SOP OS dataflows.

POOL Layer: By swapping the MAC computation with a MAX comparison function in the ALU of each PE, the RS dataflow can also process POOL layers by assuming $N = M = C = 1$ and running each fmap plane separately.

E. Other Architectural Features

In the Eyeriss architecture, the dataflow in Fig. 6 is handled using separate NoCs for the three data types: global multicast NoCs for the ifmaps and filters, and a local PE-to-PE NoC for the psums. The architecture can also exploit sparsity by (1) only performing data reads and MACs on non-zero values and (2) compressing the data to reduce data movement. Details on these techniques are described in [41]. This brings additional energy savings on top of the efficient dataflow presented in this paper.

VI. Experimental Methodology

A. Dataflow Implementation

A simulation model of each dataflow is implemented for the energy efficiency analysis using our proposed framework (Section VI-C). For the RS dataflow, we have implemented the model as described in Section V and it is verified by the measurement results of the Eyeriss chip. For each of the existing dataflows, however, different variants are demonstrated in previous designs. Therefore, our implementations of existing dataflows try to find the common ground that represents their key characteristics, and is described as follows:

Weight Stationary: Each PE holds a single weight in the RF at a time. The psum generated in a PE at each cycle is either passed to its neighbor PE or stored back to the global buffer, and the PE array operates as a systolic array with little local control. This also leaves a large area for the global buffer, which is crucial to the operation of WS dataflow.

Output Stationary: Each PE runs the psum accumulation of a single ofmap pixel at a time. We also model the MOC-MOP OS dataflow to capture convolutional reuse in the PE array, which exploits more reuse compared with the plain matrix multiplication implementation in [20]. Unlike SOC-MOP, which dedicates the PE array for 2D convolutional reuse, the MOC-MOP model uses the PE array for both 1D convolutional reuse and ifmap reuse.

No Local Reuse: The PE array consists of only ALU datapaths with no local storage. All types of data, including ifmaps, filters and psums, are stored in the global buffer.

B. Setup for Dataflow Comparison

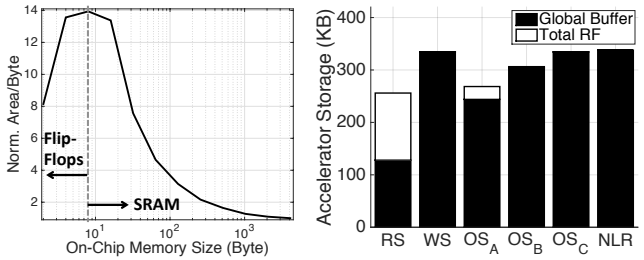
To compare the performance of different dataflows, the constraints of a fixed total hardware area and the same processing parallelism are imposed, i.e., all dataflows are given the same number of PEs with the same storage area, which includes the global buffer and RF. Based on the storage requirement of each dataflow, the storage area can be divided up differently between the global buffer and RF across dataflows. For example, RS can use a larger RF for better data reuse, but NLR does not require a RF at all.

In our simulations, a baseline storage area for a given number of PEs is calculated as

$$\#\text{PE} \times Area(512\text{B RF}) + Area((\#\text{PE} \times 512\text{B}) \text{ global buffer}). \quad (2)$$

For instance, with 256 PEs, the baseline storage area for all dataflows is calculated from the setup with 512B RF/PE and an 128kB global buffer. This baseline storage area is then used to calculate the size of the global buffer and RF in bytes for each dataflow. The total on-chip storage size will then differ between dataflows because the area cost per byte depends on the size and type of memory used as shown in Fig. 7a. In general, the area cost per byte in the RF is higher than the global buffer due to its smaller size, and thus the dataflows requiring a larger RF will have a smaller overall on-chip storage size. Fig. 7b shows the on-chip storage sizes of all dataflows under a 256-PE SA. We fix the RF size in RS dataflow at 512B since it shows the lowest energy consumption using the analysis described in Section VI-C. The difference in total on-chip storage size can go up to 80kB. For the global buffer alone, the size difference is up to 2.6×. This difference in storage size will be considered when we discuss the results in Section VII.

The accelerator throughput is assumed to be proportional to the number of active PEs for a dataflow. Although throughput



(a) (b)

Figure 7. The trade-off between storage area allocation and the total storage size. (a) A smaller memory have a higher cost on area utilization. (b) Due to the area allocation between global buffer and RF, the total on-chip storage size varies between dataflows.

is also a function of data movement, since it adds latency when there is limited storage bandwidth, there are many existing techniques commonly used to compensate for the impact, such as prefetching, double buffering, caching and pipelining. For CNN acceleration, these techniques are quite effective at hiding latency. Therefore, data movement is not expected to impact overall throughput significantly.

C. Framework for Energy Efficiency Analysis

The way each MAC operation in Eq. (1) fetches inputs (filter weight and ifmap pixel) and accumulates psums introduces different energy costs due to two factors:

- how the dataflow exploits input data reuse and psum accumulation scheduling as described in Section III-B.
- fetching data from different storage elements in the architecture have different energy costs.

The goal of an energy-efficient CNN dataflow is then to perform most data accesses using the data movement paths with lower energy cost. This is an optimization process that takes all data accesses into account, and will be affected by the layer shape and available hardware resources.

In this section, we will describe a framework that can be used as a tool to optimize the dataflows for spatial architectures. Specifically, it defines the energy cost for each level of the storage hierarchy in the architecture. Then, it provides a simple methodology to incorporate any given dataflow into an analysis using this hierarchy to quantify the overall data movement energy cost. This allows for a search for the optimal mapping for a dataflow that results in the best energy efficiency for a given CNN layer shape. For example, it describes the folding of the logical PEs onto physical PEs. For all of the dataflows, this takes into account folding in each of the three dimensions, i.e., number of filters, images and/or channels. It optimizes to maximize reuse of data in the RF, array and global buffer.

Data Movement Hierarchy: As defined in Section II, the SA accelerator provides four levels of storage hierarchy. Sorting their energy cost for data accesses from high to low, it includes DRAM, global buffer, array (inter-PE communication) and RF. Fetching data from a higher-cost level to the ALU incurs higher energy consumption. Also, the energy cost of moving data between any of the two levels is dominated by the one with higher cost. Similar to the energy consumption quantification in previous experiments [11, 12, 43], Table IV shows the energy cost of accessing each level relative to a MAC operation under the listed conditions. The numbers are extracted from a commercial 65nm process. The DRAM and global buffer energy costs aggregate the energy of accessing the storage and the iFIFO/oFIFO; the array energy cost includes the energy of accessing the iFIFO/oFIFO/pFIFO on both sides of the path as well as the cost from wiring capacitance.

	DRAM	Global Buffer (>100kB)	Array (inter-PE) (1-2mm)	RF (0.5kB)
Norm. Energy	200×	6×	2×	1×

Table IV
NORMALIZED ENERGY COST RELATIVE TO A MAC OPERATION EXTRACTED FROM A COMMERCIAL 65NM PROCESS.

Analysis Methodology: Given a dataflow, the analysis is formulated in two parts: (1) the input data access energy cost, including filters and ifmaps, and (2) the psum accumulation energy cost. The energy costs are quantified through counting the number of accesses to each level of the previously defined hierarchy, and weighting the accesses at each level with a cost from Table IV. The overall data movement energy of a dataflow is obtained through combining the results from the two types of input data and the psums.

1) Input Data Access Energy Cost: If an input data value is reused for many operations, ideally the value is moved from DRAM to RF once, and the ALU reads it from the RF many times. However, due to limited storage and operation scheduling, the data is often kicked out of the RF before exhausting reuse. The ALU then needs to fetch the same data again from a higher-cost level to the RF. Following this pattern, data reuse can be split across the four levels. Reuse at each level is defined as *the number of times each data value is read from this level to its lower-cost levels during its lifetime*. Suppose the total number of reuses for a data value is $a \times b \times c \times d$, it can be split into reuses at DRAM, global buffer, array and RF for a, b, c, and d times, respectively. An example is shown in Fig. 8, in which case the total number of reuse, 24, is split into $a = 1$, $b = 2$, $c = 3$ and $d = 4$. The energy cost estimation for this reuse pattern is:

$$\begin{aligned} &a \times EC(\text{DRAM}) + ab \times EC(\text{global buffer}) + \\ &abc \times EC(\text{array}) + abcd \times EC(\text{RF}), \end{aligned} \tag{3}$$

where $EC(\cdot)$ is the energy cost from Table IV. [1]

2) Psum Accumulation Energy Cost: Psums travel between ALUs for accumulation through the 4-level hierarchy.

[1] Optimization can be applied to Eq. (3) when there is no reuse opportunity. For instance, if $d = 1$, the data is transferred directly from a higher level to the ALU and bypasses the RF, and the last term in Eq. (3) can be dropped.

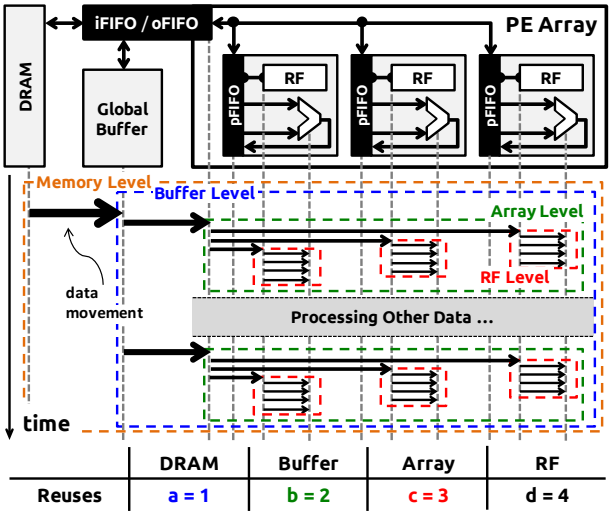


Figure 8. An example of the ifmap pixel or filter weight being reused across four levels of hierarchy.

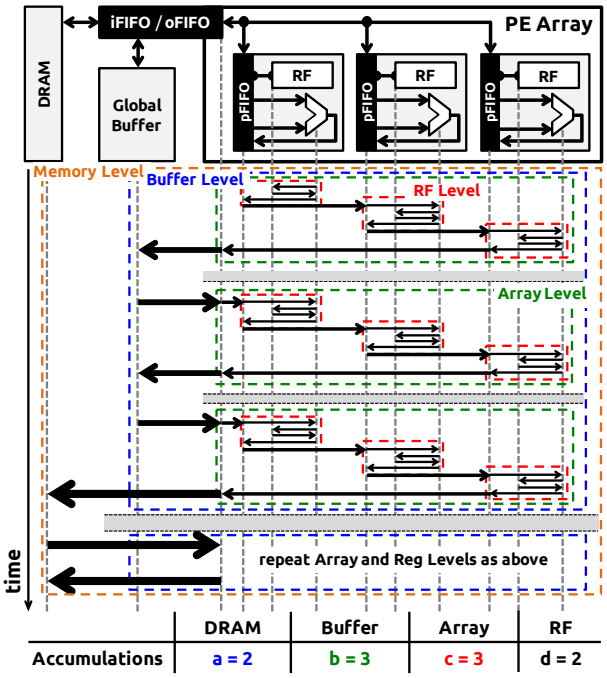


Figure 9. An example of the psum accumulation going through four levels of hierarchy.

In the ideal case, each generated psum is stored in a local RF for further accumulation. However, this is often not achievable due to the overall operation scheduling, in which case the psums have to be stored to a higher-cost level and read back again afterwards. Therefore, the total number of accumulations, $a \times b \times c \times d$, can also be split across the four levels. The number of accumulation at each level is defined as *the number of times each data goes in and out of its lower-cost levels during its lifetime*. An example is shown in Fig. 9, in which case the total number of accumulations, 36, is split into $a = 2$, $b = 3$, $c = 3$ and $d = 2$. The energy cost can then be estimated as

$$(2a-1) \times EC(\text{DRAM}) + 2a(b-1) \times EC(\text{global buffer}) + ab(c-1) \times EC(\text{array}) + 2abc(d-1) \times EC(\text{RF}). \quad (4)$$

The factor of 2 accounts for both reads and writes. Note that in this calculation the accumulation of the bias term is ignored, as it has negligible impact on overall energy.

3) Obtaining the Parameters: For each dataflow, there exists a set of parameters (a, b, c, d) for each of the three data types, i.e., ifmap, filter and psum, that describes the optimal mapping in terms of energy efficiency under a given CNN layer shape. It is obtained through an optimization process with objective functions defined in Eq. (3) and (4). The optimization is constrained by the hardware resources, including the size of the global buffer, RF and PE array.

D. Dataflow Modeling Side Note

While we charge the same energy cost at each level of the storage hierarchy across all dataflows, the real cost varies due to the actual implementation required by each dataflow. For example, a larger global buffer should be charged with a higher energy cost, which is the case for all dataflows other than RS. At array level, short-distance transfers, such as communicating with a neighbor PE, should be charged a lower energy cost than longer-distance transfers, such as broadcast or direct global buffer accesses from all PEs, due to smaller wiring capacitance and simpler NoC design. In this case, WS, OS_A, OS_C and NLR might see a bigger impact since they all have long-distance array transfers. At the RF level, a smaller RF should be charged with less energy cost. Except for RS and OS_A, the other dataflows will see a reduction in RF access energy. Overall, however, we find our results to be conservative for RS compared to the other dataflows.

VII. Experiment Results

We simulate the RS dataflow and compare its performance with our implementation of existing dataflows (Section VI-A) under the same hardware area and processing parallelism constraints. The mapping for each dataflow is optimized by our framework (Section VI-C) for the highest energy efficiency. AlexNet [2] is used as the CNN model for benchmarking due to its high popularity. Its 5 CONV and 3 FC layers also provide a wide range of shapes that are suitable for testing the adaptability of different dataflows. The simulations assume 16 bits per word, and the result aggregates data from all CONV or FC layers in AlexNet. To save space, the SOC-MOP, MOC-MOP and MOC-SOP OS dataflows are renamed as OS_A, OS_B and OS_C, respectively.

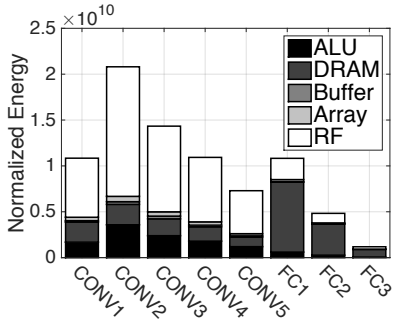


Figure 10. Energy consumption breakdown of RS dataflow in AlexNet.

A. RS Dataflow Energy Consumption

The RS dataflow is simulated with the following setup: (1) 256 PEs, (2) 512B RF per PE, and (3) 128kB global buffer. Batch size is fixed at 16. Fig. 10 shows the energy breakdown across the storage hierarchy in the 5 CONV and 3 FC layers of AlexNet. The energy is normalized to one ALU operation, i.e., a MAC.

The two types of layers show distinct energy distributions. On one hand, the energy consumption of CONV layers is dominated by RF accesses, which shows that RS fully exploits different types of data movement in the local RF and minimizes accesses to storage levels with higher cost. This distribution is verified by our Eyeriss chip measurement results where the ratio of energy consumed in the RF to the rest (except DRAM) is also roughly 4:1. On the other hand, DRAM accesses dominate the energy consumption of FC layers due to the lack of convolutional data reuse. Overall, however, CONV layers still consume approximately 80% of total energy in AlexNet, and the percentage is expected to go even higher in modern CNNs that have more CONV layers.

B. Dataflow Comparison in CONV Layers

We compare the RS dataflow with existing dataflows in (1) DRAM accesses, (2) energy consumption and (3) energy-delay product (EDP) using the CONV layers of AlexNet. Different hardware resources (256, 512 and 1024 PEs) and batch sizes (N = 1, 16 and 64) are simulated to further examine the scalability of these dataflows.

DRAM Accesses: DRAM accesses are expected to have a strong impact on the overall energy efficiency since their energy cost is orders of magnitude higher than other on-chip data movements. Fig. 11 shows the average DRAM accesses per operation of the 6 dataflows. DRAM writes are the same across all dataflows since we assume the accelerator writes only ofmaps but no psums back to DRAM. In this scenario, RS, OS_A, OS_B and NLR have significantly lower DRAM accesses than WS and OS_C, which means the former achieve more data reuse on-chip than the latter. Considering RS has much less on-chip storage compared to others, it shows the importance of co-designing the architecture and dataflow. In fact, RS can achieve the best energy efficiency when taking the entire storage hierarchy into account instead of just DRAM accesses (see Energy Consumption section).

The WS dataflow is optimized for maximizing weight reuse. However, it sacrifices ifmap reuse due to the limited number of filters that can be loaded on-chip at a time, which leads to high DRAM accesses. The number of filters is limited by (1) insufficient global buffer size for psum storage, and (2) size of PE array. In fact, Fig. 11a shows a case where WS cannot even operate due to the global buffer being too small for a batch size of 64. OS_C also has high DRAM accesses since it does not exploit convolutional reuse of ifmaps on-chip.

In terms of architectural scalability, all dataflows can use the larger hardware area and higher parallelism to reduce DRAM accesses. The benefit is most significant on WS and OS_C, which also means that they are more demanding on hardware resources. For batch size scalability, increasing N from 1 to 16 reduces DRAM accesses for all dataflows since it gives more filter reuse, but saturates afterwards. The only exception is WS, which cannot handle large batch sizes well due to the psum storage issue.

Energy Consumption: Fig. 12 shows the normalized energy consumption per operation of the 6 dataflows. Overall, RS is 1.4× to 2.5× more energy efficient than other dataflows. Although OS_A, OS_B and NLR have similar or even lower DRAM accesses compared with RS, RS still consumes lower total energy by fully exploiting the lowest-cost data movement at the RF for all data types. The OS and WS dataflows use the RF only for psum accumulation and weight reuse, respectively, and therefore spend a significant amount of energy in the array for other data types. NLR does not use the RF at all. Most of its data accesses come from the global buffer directly, which results in high energy consumption.

Fig. 12d shows the same energy result at a PE array size of 1024 but with energy breakdown by data type. The results for other PE array sizes show a similar trend. While the WS and OS dataflows are most energy efficient at weight and psum accesses, respectively, they sacrifice the reuse of other data types: WS is inefficient at ifmap reuse, and the OS dataflows cannot reuse ifmaps and weights as efficiently as RS since they focus on generating psums that are reducible. NLR does not exploit any type of reuse of weights in the PE array, and therefore consumes most of its energy for weight accesses. RS is the only dataflow that optimizes energy for all data types simultaneously.

When scaling up the hardware area and processing parallelism, the energy consumption per operation roughly stays the same across all dataflows except for WS, which sees a decrease in energy due to the larger global buffer size. Increasing batch size helps to reduce energy per operation similar to the trend shown in the case of DRAM accesses. The energy consumption of OS_C, in particular, improves significantly with batch sizes larger than 1, since there is no reuse of weights at RF and array levels when batch size is 1.

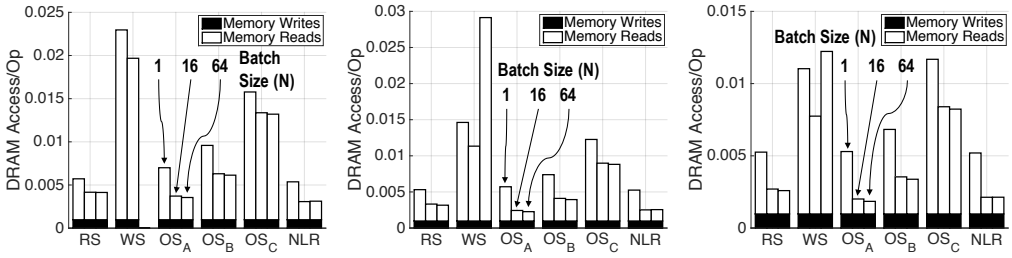


Figure 11. Average DRAM accesses per operation of the six dataflows in CONV layers of AlexNet under PE array size of (a) 256, (b) 512 and (c) 1024.

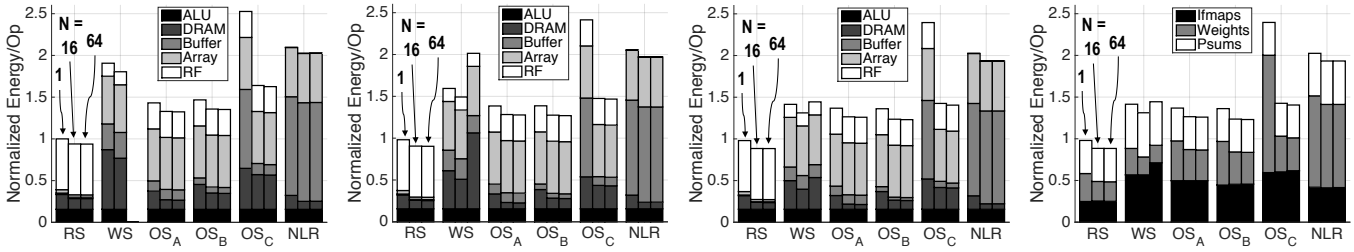


Figure 12. Energy consumption of the six dataflows in CONV layers of AlexNet under PE array size of (a) 256, (b) 512 and (c) 1024. (d) is the same as (c) but with energy breakdown in data types. The energy is normalized to that of RS at array size of 256 and batch size of 1. The RS dataflow is 1.4× to 2.5× more energy efficient than other dataflows.

Energy-Delay Product: Energy-delay product is used to verify that a dataflow does not achieve high energy efficiency by sacrificing processing parallelism, i.e., throughput. Fig. 13 shows the normalized EDP of the 6 dataflows. The delay is calculated as the reciprocal of number of active PEs. A dataflow may not utilize all available PEs due to the shape quantization effects and mapping constraints. For example, when batch size is 1, the maximum number of active PEs in OS_A and OS_C are the size of 2D ofmap plane (E^2) and the number of ofmap channels (M), respectively. Compared with the other dataflows, RS has the lowest EDP since its mapping of 1D convolution primitives efficiently utilizes available PEs. OS_A and OS_C show high EDP at batch size of 1 due to its low PE utilization, especially at larger array sizes.

C. Dataflow Comparison in FC Layers

We run the same experiments as in Section VII-B but with the FC layers of AlexNet. Fig. 14 shows the results of 6 dataflows under a PE array size of 1024. The results for other PE array sizes show the same trend. The batch size now starts from 16 since there is little data reuse with a batch size of 1, in which case the energy consumptions of all dataflows are dominated by DRAM accesses for weights and are approximately the same. The DRAM accesses, however, can be reduced by techniques such as pruning and quantization of the values [38].

Compared with existing dataflows, the RS dataflow has the lowest DRAM accesses, energy consumption and EDP in the FC layers. Even though increasing batch size helps to improve energy efficiency of all dataflows due to more filter reuse, the gap between RS and the WS/OS dataflows becomes even larger since the energy of the latter are dominated by ifmap accesses. In fact, OS_A runs FC layers very poorly because its mapping requires ifmap pixels from the same spatial plane, while the spatial size of FC layers is usually very small. Overall, the RS dataflow is at least 1.3× more energy efficient than other dataflows at a batch size of 16, and can be up to 2.8× more energy efficient at a batch size of 256.

D. Hardware Resource Allocation for RS

For the RS dataflow, we further experiment changing the hardware resource allocation between processing and storage under a fixed area. This is to determine its impact on energy efficiency and throughput. The fixed area is based on the setup using 256 PEs with the baseline storage area as defined in Eq. (2). We sweep the number of PEs from 32 to 288 and adjust the size of RF and global buffer to find the lowest energy cost in CONV layers of AlexNet for each setup.

Fig. 15 shows the normalized energy and processing delay of different resource allocations. First, although the throughput increases by more than 10× by increasing the number of PEs, the energy cost only increases by 13%. This is because a larger PE array also creates more data reuse opportunities. Second, the trade-off between throughput and energy is not monotonic. The energy cost becomes higher when the PE array size is too small due to (1) there is little data reuse in the PE array, and (2) the global buffer is already large enough that increasing the buffer size does not contribute much to data reuse.

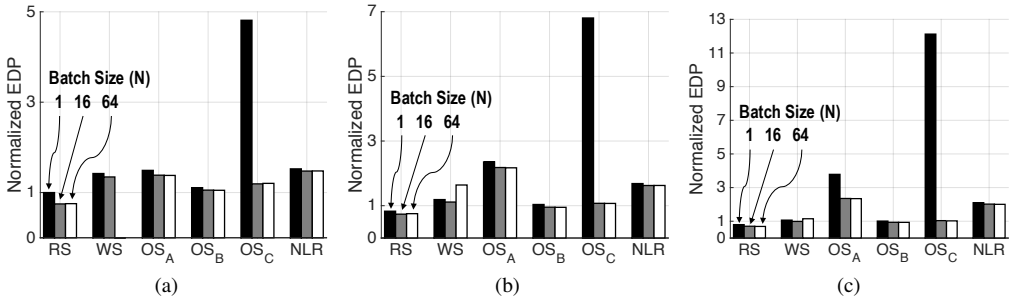


Figure 13. Energy-delay product (EDP) of the six dataflows in CONV layers of AlexNet under PE array size of (a) 256, (b) 512 and (c) 1024. It is normalized to the EDP of RS at PE array size of 256 and batch size of 1.

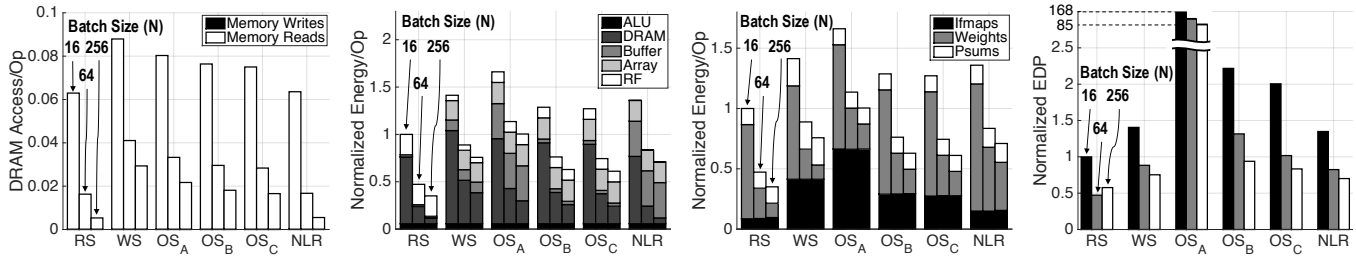


Figure 14. (a) average DRAM accesses per operation, energy consumption with breakdown in (b) storage hierarchy and (c) data types, and (d) EDP of the six dataflows in FC layers of AlexNet under PE array size of 1024. The energy consumption and EDP are normalized to that of RS at batch size of 1.

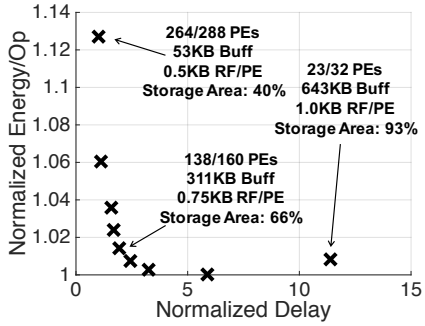


Figure 15. Relationship between normalized energy per operation and processing delay under the same area constraint but with different processing area to storage area ratio.

VIII. Conclusions

This paper presents an analysis framework to evaluate the energy cost of different CNN dataflows on a spatial architecture. It accounts for the energy cost of different levels of the storage hierarchy under fixed area and processing parallelism constraints. It also can be used to search for the most energy-efficient mapping for each dataflow. Under this framework, a novel dataflow, called *row stationary* (RS), is presented that minimizes energy consumption by maximizing input data reuse (filters and feature maps) and minimizing partial sum accumulation cost simultaneously, and by accounting for the energy cost of different storage levels. Compared with existing dataflows such as the output stationary (OS), weight stationary (WS), and no local reuse (NLR) dataflows using AlexNet as a benchmark, the RS dataflow is 1.4× to 2.5× more energy efficient in convolutional layers, and at least 1.3× more energy efficient in fully-connected layers for batch sizes of at least 16.

We also observe that DRAM bandwidth alone does not dictate energy-efficiency; dataflows that require high bandwidth to the on-chip global buffer can also result in significant energy cost. For all dataflows, increasing the size of the PE array helps to improve the processing throughput at similar or better energy efficiency. Larger batch sizes also result in better energy efficiency in all dataflows except for WS, which suffers from insufficient global buffer size. Finally, for the RS dataflow, the area allocation between processing and storage has a limited effect on energy-efficiency, since more PEs allow for better data reuse, which balances out the effect of less on-chip storage.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.

[3] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1556, 2014.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper With Convolutions," in *IEEE CVPR*, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE CVPR*, 2016.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *IEEE CVPR*, 2014.

[7] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," *CoRR*, vol. abs/1312.6229, 2013.

[8] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning Deep Features for Scene Recognition using Places Database," in *NIPS*, 2014.

[9] Y. Le Cun, L. Jackel, B. Boser, J. Denker, H. Graf, I. Guyon, D. Henderson, R. Howard, and W. Hubbard, "Handwritten digit recognition: applications of neural network chips and automatic learning," *IEEE Communications Magazine*, vol. 27, no. 11, 1989.

[10] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *ICANN*, 2014.

[11] B. Dally, "Power, Programmability, and Granularity: The Challenges of ExaScale Computing," in *IEEE IPDPS*, 2011.

[12] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE ISSCC*, 2014.

[13] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding Sources of Inefficiency in General-purpose Chips," in *ISCA*, 2010.

[14] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," *CoRR*, vol. abs/1410.0759, 2014.

[15] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A Massively Parallel Coprocessor for Convolutional Neural Networks," in *IEEE ASAP*, 2009.

[16] V. Sriram, D. Cox, K. H. Tsoi, and W. Luk, "Towards an embedded biologically-inspired machine vision processor," in *FPT*, 2010.

[17] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A Dynamically Configurable Coprocessor for Convolutional Neural Networks," in *ISCA*, 2010.

[18] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for Convolutional Neural Networks," in *IEEE ICCD*, 2013.

[19] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks," in *IEEE CVPRW*, 2014.

[20] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep Learning with Limited Numerical Precision," *CoRR*, vol. abs/1502.02551, 2015.

[21] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *FPGA*, 2015.

[22] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning," in *ASPLOS*, 2014.

[23] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *ISCA*, 2015.

[24] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *MICRO*, 2014.

[25] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, "A 1.93TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications," in *IEEE ISSCC*, 2015.

[26] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A Convolutional Network Accelerator," in *GLSVLSI*, 2015.

[27] E. Mirsky and A. DeHon, "MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources," in *IEEE FCCM*, 1996.

[28] J. R. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor," in *IEEE FCCM*, 1997.

[29] B. Mei, S. Vernalde, D. Verkest, H. D. Man, and R. Lauwereins, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix," in *FPL*, 2003.

[30] A. Parashar, M. Pellauer, M. Adler, B. Ahsan, N. Crago, D. Lustig, V. Pavlov, A. Zhai, M. Gambhir, A. Jaleel, R. Allmon, R. Rayess, S. Maresh, and J. Emer, "Triggered Instructions: A Control Paradigm for Spatially-programmed Architectures," in *ISCA*, 2013.

[31] V. Govindaraju, C.-H. Ho, and K. Sankaralingam, "Dynamically Specialized Datapaths for Energy Efficient Computing," in *IEEE HPCA*, 2011.

[32] S. Swanson, A. Schwerin, M. Mercaldi, A. Petersen, A. Putnam, K. Michelson, M. Oskin, and S. J. Eggers, "The WaveScalar Architecture," *ACM TOCS*, vol. 25, no. 2, 2007.

[33] H. Schmit, D. Whelihan, A. Tsai, M. Moe, B. Levine, and R. Reed Taylor, "PipeRench: A virtualized programmable datapath in 0.18 micron technology," in *IEEE CICC*, 2002.

[34] D. Burger, S. W. Keckler, K. S. McKinley, M. Dahlin, L. K. John, C. Lin, C. R. Moore, J. Burrill, R. G. McDonald, and W. Yoder, "Scaling to the End of Silicon with EDGE Architectures," *Computer*, vol. 37, no. 7, 2004.

[35] T. Nowatzki, V. Gangadhar, and K. Sankaralingam, "Exploring the Potential of Heterogeneous Von Neumann/Dataflow Execution Models," in *ISCA*, 2015.

[36] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *IEEE ISCAS*, 2010.

[37] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *ICML*, 2010.

[38] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding," in *ICLR*, 2016.

[39] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[40] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, "Convolution Engine: Balancing Efficiency and Flexibility in Specialized Computing," in *ISCA*, 2013.

[41] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *IEEE ISSCC*, 2016.

[42] J. J. Tithi, N. C. Crago, and J. S. Emer, "Exploiting spatial architectures for edit distance algorithms," *IEEE ISPASS*, 2014.

[43] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz, "Towards energy-proportional datacenter memory with mobile dram," in *ISCA*, 2012.