Eyeriss v2: A Flexible and High-Performance Accelerator for Emerging Deep 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*—The design of deep neural networks (DNNs) has increasingly focused on reducing the computational complexity in addition to improving accuracy. While emerging DNNs tend to have fewer weights and operations, they also reduce the amount of data reuse with more widely varying layer shapes and sizes. This introduces a diverse set of DNNs, ranging from large ones with high reuse (e.g., AlexNet) to compact ones with high bandwidth requirements (e.g., MobileNet). However, many existing DNN processors depend on certain DNN properties, e.g., a large number of input and output channels, to achieve high performance and energy efficiency. As a result, they do not have sufficient flexibility to efficiently process a diverse set of DNNs.**

In this work, we present *Eyexam*, a performance analysis framework that quantitatively identifies the sources of performance loss in DNN processors. It highlights two architectural bottlenecks in many existing designs. First, their dataflows are not flexible enough to adapt to the widely varying layer shapes and sizes across different DNNs. Second, their on-chip data delivery network (NoC) cannot adapt to support *both* high data reuse and high bandwidth scenarios. Based on this analysis, we present *Eyeriss v2*, a high-performance DNN accelerator that can adapt to a wide range of DNNs. Eyeriss v2 features a new dataflow, called Row-Stationary Plus (RS+), that enables the spatial tiling of data from all dimensions to fully utilize the parallelism for high performance. To support RS+, it further proposes a low-cost and scalable NoC design, called hierarchical mesh, that connects the high-bandwidth global buffer to the array of processing elements (PEs) in a two-level hierarchy. This enables high-bandwidth data delivery while still being able to harness any available data reuse. Compared with Eyeriss, Eyeriss v2 shows a performance increase between 10.4×–17.9× for 256 PEs, 37.7×–71.5× for 1024 PEs, and 448.8×–1086.7× for 16384 PEs on DNNs with widely varying amounts of data reuse (i.e., AlexNet, GoogLeNet and MobileNet).

I. Introduction

The development of deep neural networks (DNNs) has shown tremendous progress in the past few years. Specifically, there is an increasing focus on reducing the computational complexity of DNNs [22]. This trend is evident in how the iconic DNNs[1] evolve over time. Early models, such as AlexNet [14] and VGG [21], are now considered *large* and over-parameterized. Techniques such as using deeper but narrower network structures and bottleneck layers were therefore proposed to pursue higher accuracy while restricting the size of the DNN (e.g., GoogLeNet [23] and ResNet [9]). This quest further continued with a focus on drastically reducing the amount of computation, specifically the number MACs, and the storage cost, specifically the number of weights. Techniques such as filter decomposition as shown in Fig. 1 have since become popular for building *compact* DNNs (e.g., SqueezeNet [11] and MobileNet [10]).

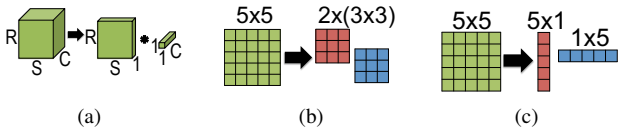


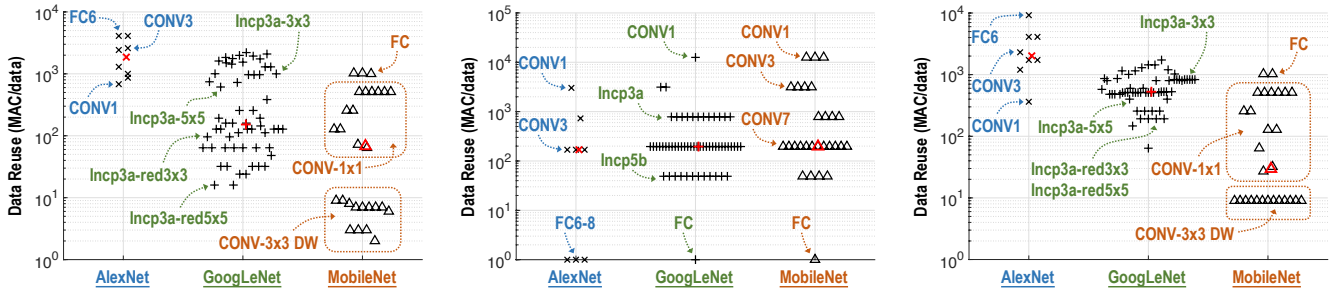
Fig. 1. Various filter decomposition approaches [10], [21], [24].

For computer architects, however, this transition brings more challenges than relief due to the change in a key property of DNNs: *data reuse*, which is the number of MACs that each data value is used for (i.e., MACs/data). Fig. 2 shows the amount of data reuse for all three data types, i.e., input activations (iacts), weights and partial sums (psums), in each layer of the three DNNs, ordered from large to compact models: AlexNet, GoogLeNet and MobileNet. When the DNN becomes more compact, the profiled results indicate that the variation in data reuse increases in all data types, and the amount of reuse also decreases in iacts and psums. This phenomenon is mainly due to the highly varying layer shapes and sizes in compact DNNs, in which any data dimension can diminish as summarized in Table I.[2] Diminishing data dimensions result in reduced reuse of related data types. For example, the amount of iact reuse reduces with fewer number of output channels.

This trend makes the design of DNN processors more challenging. For performance, widely varying data reuse is bad in two ways. First, many existing DNN processors [1], [6], [7], [8], [13], [18], [19], [27] depend on data reuse in certain data dimensions to fully exploit parallelism. For instance, the spatial accumulation array architecture (Fig. 3(a)) relies on both output and input channels to map the operations onto the PE array for the spatial reuse of iacts and spatial accumulation of

[1]We draw examples primarily from the field of computer vision, but this trend is universal across many fields.

[2]These workloads (convolutional layers, fully-connected layers, depth-wise layers, etc.) are used in a wide variety of DNNs such as CNNs, MLPs and LSTMs.

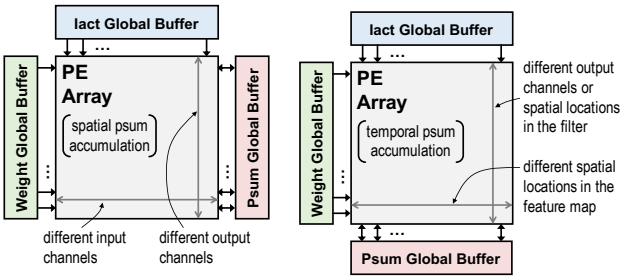


(a) Input activations (iacts) (b) Weights (batch size = 1) (c) Partial sums (psums)

Fig. 2. Data reuse of the three data types in each layer of the three DNNs. Each data point represents a layer, and the red point indicates the median amount of data reuse among all the layers in a DNN.

Data Dimension		Common Reasons for Diminishing Dimension
G	number of channel groups	non-depth-wise layers
N	batch size	low latency requirements
M	number of output channels	(1) bottleneck layers (2) depth-wise layers
C	number of input channels	(1) layers after bottleneck layers (2) depth-wise layers (3) first layer (e.g., 3 in visual inputs)
H / W	input feature map height/width	deeper layers in a DNN
R / S	filter height/width	(1) point-wise layer (i.e., 1×1)
E / F	output feature map height/width	(1) deeper layers in a DNN (2) fully-connected layers

TABLE I
REASONS FOR REDUCED DATA DIMENSIONS IN A DNN LAYER.



(a) Spatial accumulation array (b) Temporal accumulation array

Fig. 3. (a) Spatial accumulation array: iacts are reused vertically and psums are accumulated horizontally. (b) Temporal accumulation array: iacts are reused vertically and weights are reused horizontally.

psums, respectively. Similarly, the temporal accumulation array architecture (Fig. 3(b)) relies on another set of data dimensions to achieve spatial reuse of iacts and weights. When the reuse in these data dimensions is low, e.g., number of output channels in a layer (M) is less than the height of the PE array, it affects the number of active PEs used in processing. Second, a lower data reuse also implies that a higher data bandwidth is required to keep the PEs busy. If the on-chip network (NoC) for data delivery to the PEs is designed for high spatial reuse scenarios, e.g., a broadcast network, the insufficient bandwidth[3] can lead to reduced utilization of the active PEs, which further reduces the processor performance. However, if the NoC is optimized for high bandwidth scenarios, e.g., many unicast networks, it may not be able to take advantage of data reuse when available.

An additional challenge lies in the fact that all DNNs that the hardware needs to run will *not be known at design time* [5]; as a result, the hardware has to be flexible enough to efficiently support a wide range of DNNs, including *both* large and compact ones. To build a truly flexible DNN processor, the new challenge is to design an architecture that can accommodate a wide range of data reuse among large and compact DNNs. It has to maintain high performance to take advantage of the compact DNNs, but still be able to exploit data reuse with the memory hierarchy and high parallelism when the opportunity presents itself.

[3]In this paper, data bandwidth refers to the source bandwidth of the NoC if not otherwise indicated.

In summary, many existing DNN processors suffer from a performance bottleneck when dealing with a wide range of DNNs due to the following reasons: (1) the dataflow relies on certain data dimensions for data reuse to achieve high processing parallelism, which can result in low utilization of the parallelism when those data dimensions diminish; (2) the NoC for the delivery of each data type and its corresponding memory hierarchy are designed for either (i) high bandwidth and low spatial data reuse or (ii) low bandwidth and high spatial data reuse scenarios instead of being adaptive to the specific condition of the workload.

In order to design a flexible architecture that can deal with the issues above, it is crucial to first understand how different architectural attributes affect the performance, e.g., dataflows, data bandwidth, PE array size, etc. However, currently there is still no such framework for architects to quickly analyze an architecture and iterate through the design. Therefore, in this work, we will first introduce a performance analysis framework, called *Eyexam*, that can be used to systematically identify sources of performance loss in existing architectures. Specifically, we will use this framework to quantify how dataflows and NoCs impact the performance, and address it with a new architecture.

Based on the insights from Eyexam, we present *Eyeriss v2*, a flexible architecture for DNN processing that can adapt to a wide range of data reuse in both large and compact DNNs. This is achieved through the co-design of the dataflow and NoC, which are currently the bottleneck for dealing with a

more diverse set of DNNs. Specifically:

- **Flexible Dataflow** (Section III): we propose a new dataflow, named Row-Stationary Plus (RS+), that is based on the RS dataflow and further improves on the flexibility by supporting data tiling from all dimensions to fully utilize the PE array, preventing performance loss when the available reuse in certain dimensions is low.
- **Flexible and Scalable NoC** (Section IV): we propose a new NoC, called hierarchical mesh, that is designed to adapt to a wide range of bandwidth requirements. When data reuse is low, it can provide high bandwidth from the memory hierarchy to keep the PEs busy; when data reuse is high, it can still exploit spatial data reuse to achieve high energy efficiency. The NoC can also be easily scaled as its implementation cost increases linearly with the size of the PE array.

II. Eyexam: Performance Analysis Framework

In this section, we will present Eyexam, an analysis framework that quantitatively identifies the sources of performance loss in DNN processors. Instead of comparing the overall performance of different designs, which can be affected by many non-architectural factors such as system setup and process technology differences, Eyexam provides a step-by-step process that associates a certain amount of performance loss to each architectural design decision (e.g., dataflow, number of PEs, NoC, etc.) as well as the properties of the workload, which for DNNs are dictated by the layer shape and size (e.g., filter size, number of channels, etc.), which are defined by the parameters in Table I.

Eyexam focuses on two main factors that affect performance: (1) the *number of active PEs* due to the mapping determined by the dataflow, (2) the *utilization of active PEs* based on whether the NoC has sufficient bandwidth to deliver data to PEs to keep them active. The product of these two components can be used to compute the *overall utilization of the PE array* as follows

$$\text{overall utilization of the PE array} = \text{number of active PEs} \times \text{utilization of active PEs}$$

Later in this section, we will see how this approach can use an adapted form of the well-known roofline model [26] for the analysis of DNN processors.

We will perform this analysis on a generic DNN processor architecture based on a spatial architecture that consists of a global buffer (GLB) and an array of PEs as shown in Figure 4. Each PE can have its own scratchpad (SPad) and control logic, and communicates with its neighbor PEs and the GLB through the NoCs. Separate NoCs are used for the three data types, and Fig. 5 shows several commonly used NoC designs for different degrees of data reuse and bandwidth requirements. The choice largely depends on how the dataflow exploits spatial data reuse for a specific data type.

The dataflow of a DNN processor is one of the key attributes that define its architecture [3]. In this work, we will analyze

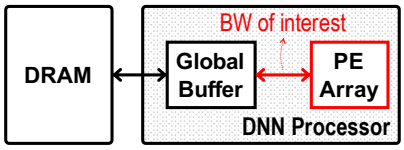


Fig. 4. A generic DNN processor architecture.

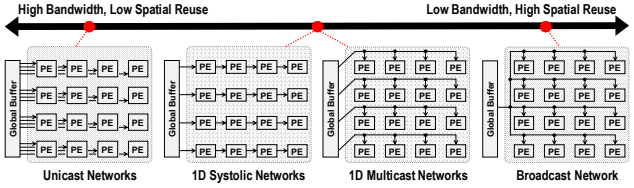


Fig. 5. Common NoC implementations in DNN processor architectures.

architectures that support the following four widely used dataflows [2], [20]: weight-stationary (WS), output-stationary (OS), input-stationary (IS), and row-stationary (RS).

To help illustrate the capabilities of Eyexam, we will set up a simple example of 1D convolution in Section II-A, and walk through the key steps of Eyexam in Section II-B with the example. We will then highlight various insights that Eyexam gives on real DNN workloads and architectures in Section II-C, which motivates the development of the Eyeriss v2 architecture discussed in Section III and Section IV.

A. Simple 1D Convolution Example

We will start with a simple 1D convolution example to illustrate the two components of the problem. The first is the *workload*, which is represented by the shape of the layer for a 1D convolution. This comprises the filter size R and the input feature map size H and the output feature map size E. The second is the *architecture* of the processing unit, for which a key characteristic is the dataflow shown in Fig. 6. In this example, the two `parallel-fors` represent the distribution of computation across multiple PEs (i.e., spatial processing); the inner two `for loops` represent the temporal processing and SPad accesses within a PE, and the outer two `for loops` represent the temporal processing of multiple passes across the PE array and GLB accesses.

A mapping assigns specific values to loop limits $E0$, $E1$, $E2$ and $R0$, $R1$, $R2$ to execute a specific workload shape and

```
Input Fmaps:    I[H]
Filter Weights: W[R]
Output Fmaps:   O[E]

for (e2=0; e2<E2; e2++) {
    for (r2=0; r2<R2; r2++) {
        parallel-for (e1=0; e1<E1; e1++) {
            parallel-for (r1=0; r1<R1; r1++) {
                for (e0=0; e0<E0; e0++) {
                    for (r0=0; r0<R0; r0++) {
                        O[e2*E1*E0+e1*E0+e0] +=
                            I[e2*E1*E0+e1*E0+e0 + r2*R1*R0+r1*R0+r0] *
                            W[r2*R1*R0+r1*R0+r0];
}}}}}}
```

Fig. 6. An example dataflow for a 1D convolution.

loop ordering. In other words, these loop limits determine how each data type is tiled temporally across different levels of the memory hierarchy and spatially across different PEs. This assignment of $E0$, $E1$, $E2$ and $R0$, $R1$, $R2$ is constrained by the shape of the workload and the hardware resources. The workload constraints in this example are $E0 \times E1 \times E2 = E$ and $R0 \times R1 \times R2 = R$.[4] The architectural constraint in this example is that $E1 \times R1$ must be less than the number of PEs (later we will see that the NoC can pose additional restrictions). The size of the SPad allocated to iacts, psums and weights will restrict $E0$ and $R0$, and the size of the GLB allocated to psums restricts $E1$ and $R1$.

While this is a simple 1D example, it can be extended to additional levels of buffering by adding additional levels of loop nest. Furthermore, extending it to support additional data dimensions (e.g., 2D and channels) will also results in additional loops.

B. Apply Performance Analysis Framework to 1D Example

The goal of Eyexam is to provide a fine-grain performance profile for an architecture. It is a sequential analysis process that involves seven major steps. The process starts with the assumption that the architecture has infinite processing parallelism, storage capacity and data bandwidth. Therefore, it has infinite performance (as measured in MACs/cycle).

For each of the following steps, certain constraints will be applied to reflect changes in the assumptions on the architecture or workload. The associated performance loss can therefore be attributed to that change, and the final performance at each step becomes the upper-bound for the next step.

Step 1 (Layer Shape and Size): In this first step, we look at the impact of the workload constraint, specifically the layer shape and size, assuming unbounded values for $R1$ and $E1$ since there are no architectural constraints. This allows us to set $R1 = R$, $E1 = E$, and $E2 = E0 = 1$, $R2 = R0 = 1$, so that there is all spatial (i.e., parallel) processing, and no temporal (i.e., serial) processing. Therefore, the performance upper bound is determined by the finite size of the workload (i.e., the number of MACs in the layer, which is $E \times R$).

Step 2 (Dataflow): In this step, we define the dataflow and examine the impact of this architectural constraint. For example, to configure the example loop nest into a weight-stationary (WS) dataflow, we would set $E1 = 1$, $E0 = E$ and $R1 = R$, $R0 = 1$. This means that each PE stores one weight, that weight is reused $E0$ times within that PE, and the number of PEs is equal to the number of weights. This forces the absolute maximum amount of reuse for weights at the PE. The forced serialization of $E0 = E$ reduces the performance upper bound from $E \times R$ to R, which is the maximum parallelism of the dataflow.

Step 3 (Number of PEs): In this step, we define a finite number of PEs, and look at the impact of this architectural constraint. For example, in the 1D WS example, where $E1 = 1$ and $E0 = E$, $R1$ is constrained to be less than or equal to the number of PEs, which dictates the theoretical peak performance. There are two scenarios when the actual performance is less than the peak performance. The first scenario is called *spatial mapping fragmentation*, in which case R, and therefore $R1$, is smaller than the number of PEs. In this case, some PEs are completely idle throughout the entire period of processing. The second scenario is called *temporal mapping fragmentation*, in which case R is larger than the number of PEs but not an integer multiple of it. For example, when the number of PEs is 4, $R = 7$ and $R1 = 4$, it takes two cycles to complete the processing, and none of the PEs are completely idle. However, one of the 4 PEs will only be 50% active. Therefore, it still does not achieve the theoretical peak performance. In general, however, if the workload does not map into all of the PEs in all cycles, then some PEs will *not* be used at 100%, which should be taken into account in the performance evaluation.

Step 4 (Physical dimensions of the PE array): In this step, we consider the physical dimensions of the PE array (e.g., arranging 12 PEs as 3×4, 2×6 or 4×3, etc.). The spatial partitioning is constrained per dimension which can cause additional performance loss. To explain this step with our simple 1D example, we need to release the WS restriction. Let us assume $E1$ is mapped to the width of the 2D array and $R1$ is mapped to the height of the 2D array. If $E1$ is less than the width of the array or $R1$ is less than the height of the array (spatial mapping fragmentation), not all PEs will be utilized even if $E1 \times R1$ is greater than the number of PE. A similar case can be constructed for the temporal mapping fragmentation as well. This architectural constraint further reduces the number of active PEs.

Step 5 (Storage Capacity): In this step, we consider the impact of making the buffer storage finite. For the WS dataflow example, if the allocated storage for psums in the GLB is limited, it limits the number of weights that can be processed in parallel, which limits the number of PEs that can operate in parallel. Thus an architectural constraint on how many psums can be stored in the GLB restricts $E1$ and $R1$, which again can reduce the number of active PEs.

Step 6 (Data Bandwidth): In this step, we consider the impact of a finite bandwidth for delivering data across the different levels of the loop nest (i.e., memory hierarchy). The amount of data that needs to be transferred between each level of the loop nest and the bandwidth at which we can transmit the data dictate the speed at which the index of the loop can increment (i.e., number of cycles per MAC). For instance, the bandwidth of the SPad in the PE dictates the rate of change of $r0$ and $e0$, the bandwidth of the NoC and GLB dictates the rate of change of $r1$ and $e1$, and the off-chip bandwidth dictates the rate of change of $r2$ and $e2$. In this work, we will focus on the bandwidth between the GLB and the PEs.

To quantify the impact of insufficient bandwidth on performance, we can adapt the well-known roofline model [26] for the analysis of DNN processors. The roofline model, as shown in Fig. 7, is a tool that visualizes the performance of an architecture under various degrees of operational intensity. It

[4]We assume perfect factorization in this example. Imperfect factorization will lead to cycles where no work is done.

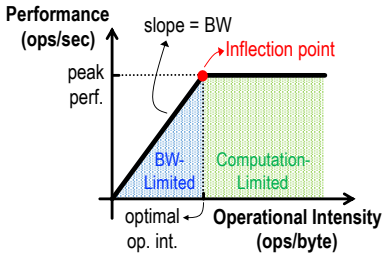


Fig. 7. The roofline model

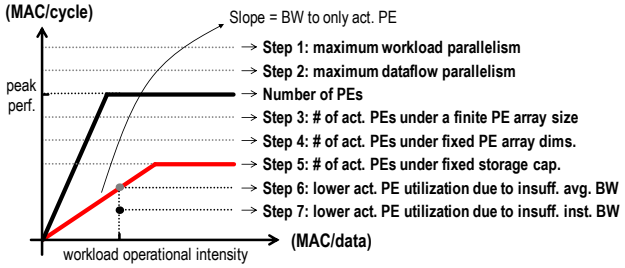


Fig. 8. Impact of Eyexam steps on the roofline model.

assumes a processing core, e.g., PE array, that has insufficient local memory to fit the entire workload, and therefore its performance can be limited by insufficient bandwidth between the core and the memory, e.g., GLB. When the operational intensity is lower than that at the inflection point, the performance will be bandwidth-limited; otherwise, it is computation-limited. The roofline indicates the performance upper-bound, and the performance of actual workloads sit in the area under the roofline.

For this analysis, we adapt the roofline model as follows:

- We use three separate rooflines for the three data types instead of one with the aggregated bandwidth and operational intensity.[5] This helps identify the performance bottleneck and is also a necessary setup since independent NoCs are used for each data type. The performance upper-bound will be the worst case of the three rooflines.
- The roofline is typically drawn with the peak performance of the core and the total bandwidth between the core and memory. However, since we have gone through the first 5 steps in Eyexam, it is possible to get a tighter bound (Fig. 8). The leveled part of the roofline is now at the performance bound from step 5; the slanted part of the roofline should only consider the bandwidth to the active PEs for each data type. Since performance is measured in MACs/cycle, the bandwidth should factor in the clock rate differences between processing and data delivery.
- For a workload layer, the operational intensity of a data type is the same as its amount of data reuse in the PE array, including both temporal reuse with the SPad and the spatial reuse across PEs. It is measured in MACs per data value (MAC/data) to normalize the differences in bitwidth.

Step 7 (Varying Data Access Patterns): In this step, we consider the impact of bandwidth varying across time due to the dynamically changing data access patterns (Step 6 only addresses average bandwidth). For the WS example, during ramp up, the weight NoC will require high bandwidth to load the weights into the SPad of the PEs, but in steady state, the bandwidth requirements of the weight NoC will be low since the weights are reused within the PE. The performance upper bound will be affected by ratio of time spent in ramp up versus steady state, and the ratio of the bandwidth demand versus available bandwidth. This step causes the performance point to fall off the roofline as shown in Fig. 8. There exist many common solutions to address this issue, including using double buffering or increased bus-width for the NoC. Therefore, we will focus less on the performance loss due to this step in this work.

Table II summarizes the constraints applied at each step. While Eyexam is useful for examining the impact of each step on performance, it can also be used in the architecture design process to iterate through a design. For instance, if one selects a dataflow in step 2 and discovers that the storage capacity in step 5 is not a good match causing a large performance loss, one could return to step 2 to make a different dataflow design choice and then go through the steps again. Another example is that double buffering could be used in step 7 to hide the high bandwidth during ramp up, however, this would require returning to step 5 to change the effective storage capacity constraints. Eyexam can also be applied to consider the trade-off between performance and energy efficiency in combination with the framework for evaluating energy efficiency as discussed in [2], as well as consider the impact of sparsity and workload imbalance on performance. However, this is beyond the scope of this paper.

C. Performance Analysis Results for DNN Processors and Workloads

In this section, we will highlight some of the observations obtained with Eyexam on DNN processors with real DNN workloads (e.g., AlexNet, MobileNet). We will provide results for architectures from all four representative dataflows, including WS, OS, IS, and RS, with different PE array sizes. The dataflows are evaluated on PE arrays where the height and the width are the same, regardless of the number of PEs.

Fig. 9 shows the number of active PEs for the four different architectures in different DNN layers and PE array sizes. It takes into account the mapping of different dataflows in each architecture for different layer shapes under a finite number of PEs. The results are normalized to the total number of PEs in the array. For each bar, the total bar height (white-portion + colored-portion) represent the performance at step 3 of Eyexam, which accounts for the impact of mapping fragmentation due to a finite number of PEs, and the colored-only portion represent the performance at step 4, which further accounts for the

[5]Ideally, we should draw a *roof-manifold* with the operational intensity of each data type on a separate axis; unfortunately, it will be a 4-D plot that is difficult to visualized.

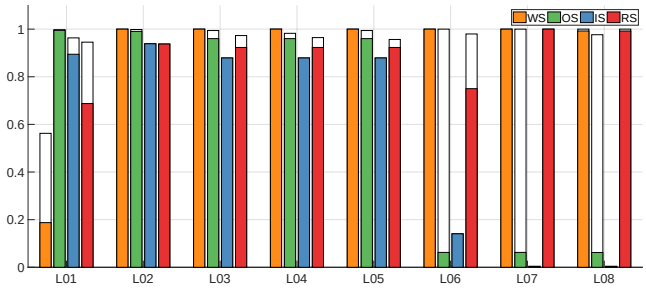
Step	Constraint	Type	New Performance Bound	Reason for Performance Loss
1	Layer Size and Shape	Workload	Max workload parallelism	Finite workload size
2	Dataflow loop nest	Architectural	Max dataflow parallelism	Restricted dataflow mapping space by defined by loop nest
3	Number of PEs	Architectural	Max PE parallelism	Additional restriction to mapping space due to shape fragmentation
4	Physical dimensions of PEs array	Architectural	Number of active PEs	Additional restriction to mapping space due to shape fragmentation for *each* dimension
5	Fixed Storage Capacity	Architectural	Number of active PEs	Additional restriction to mapping space due to storage of intermediate data (depends on dataflow)
6	Fixed Data Bandwidth	Microarchitectural	Max data bandwidth to active PEs	Insufficient average bandwidth to active PEs
7	Varying Data Access Patterns	Microarchitectural	Actual measured performance	Insufficient instant bandwidth to active PEs

TABLE II
SUMMARY OF STEPS IN EYEXAM.

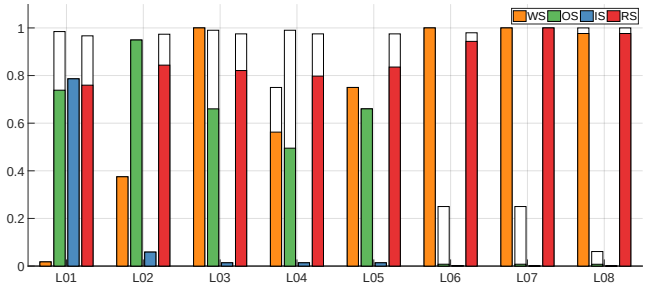
impact of the physical dimensions of the PE array. Therefore, the white portion indicates the performance loss from step 3 to 4, which indicates the mapping limitation in the dataflows to adapt to the physical dimensions of the PE array. We observe the following:

- Fig. 9(a) and 9(b) shows the performance impact when scaling the size of PE array. Many of the architectures are not flexible enough to fully utilize the parallelism when it scales up (i.e., increase number of PEs), which indicates that simply increasing hardware resources is not sufficient to achieve a higher performance.
- Fig. 9(b) and 9(c) shows the performance impact when having to support many different layer shapes. Mapping the different layers onto the same architecture according to its dataflow can result in widely varying performance. For example, the featured IS and OS architectures cannot map well in the layers with smaller feature map sizes, while the RS dataflow does not map well in the depth-wise layers due to the lack of channels. Table I summarizes the common reasons why each data dimension diminishes. In order to support a wide variety of DNNs, the dataflow has to be flexible enough to deal the diminished reuse available in any data dimensions.

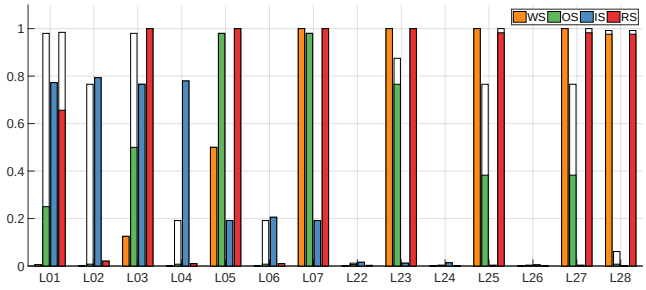
In addition to the loss due to the finite number of PEs and the physical PE array shape, there is loss from insufficient bandwidth for data delivery. To avoid performance loss due to insufficient data bandwidth from the GLB, which results in low utilization of the active PEs (step 6), the NoC design should meet the worst-case bandwidth requirement for every data type. Furthermore, the NoC design should also aim to exploit data reuse to minimize the number of GLB accesses, which is usually realized by the multicast or broadcast of data from GLB. On the one hand, for an architecture in which the pattern of spatial data reuse is unchanged with mapping, it is straightforward to meet the two requirements at the same time. For example, if a certain type of data is always reused across an entire PE row or column, the systolic or multicast networks will provide sufficient bandwidth and data reuse from GLB. However, this fixed pattern of data delivery can also cause performance loss in step 3 or 4 of Eyexam. On the other hand, if the architecture support very flexible spatial mappings of



(a) AlexNet, 256 PEs (16×16)



(b) AlexNet, 16384 PEs (128×128)



(c) MobileNet, 16384 PEs (128×128)

Fig. 9. Impact of the number of PEs and the physical dimensions of the PE array on number of active PEs. The y-axis is the performance normalized to the number of PEs.

```
Input Fmaps:    I[G][N][C][H][W]
Filter Weights: W[G][M][C][R][S]
Output Fmaps:   O[G][N][M][E][F]

// DRAM levels
for (g3=0; g3<G3; g3++) {
  for (n3=0; n3<N3; n3++) {
    for (m3=0; m3<M3; m3++) {
      for (f3=0; f3<F3; f3++) {
        // Global buffer levels
        for (g2=0; g2<G2; g2++) {
          for (n2=0; n2<N2; n2++) {
            for (m2=0; m2<M2; m2++) {
              for (f2=0; f2<F2; f2++) {
                for (c2=0; c2<C2; c2++) {
                  for (s2=0; s2<S2; s2++) {
                    // NoC levels
                    parallel-for (g1=0; g1<G1; g1++) {
                      parallel-for (n1=0; n1<N1; n1++) {
                        parallel-for (m1=0; m1<M1; m1++) {
                          parallel-for (f1=0; f1<F1; f1++) {
                            parallel-for (c1=0; c1<C1; c1++) {
                              parallel-for (s1=0; s1<S1; s1++) {
                                // SPad levels
                                for (f0=0; f0<F0; f0++) {
                                  for (n0=0; n0<N0; n0++) {
                                    for (e0=0; e0<E; e0++) {
                                      for (r0=0; r0<R; n++) {
                                        for (c0=0; c0<C0; c++) {
                                          for (m0=0; m0<M0; m++) {
                                            O += I × W;
}}}}}}}}}}}}}}}}}}}}}}}}
```

Fig. 10. The definition of the Row-Stationary Plus (RS+) dataflow. For simplicity, we ignore the bias term and the indexing in the data arrays.

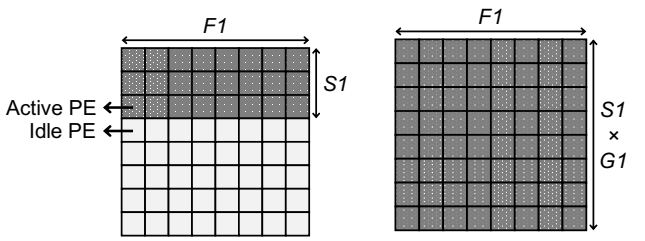
operations, which potentially can preserve the performance up to step 5 of Eyexam, the pattern of spatial data reuse can vary widely for different layer shapes. While a single broadcast network can exploit data reuse in any spatial reuse pattern, it sacrifices the data bandwidth from GLB. When the amount of data reuse is low, e.g., delivering weights in FC layers with a small batch size, the broadcast network will result in significant performance loss. Therefore, step 6 will become a performance bottleneck. We will address this problem with a proposed flexible architecture in Section III and IV.

III. Flexible Dataflow: Row-Stationary Plus (RS+)

The RS dataflow was designed with the goal to optimize for the best overall system energy efficiency. While it is already more flexible than other existing dataflows, we have identified several techniques that can be applied to further improve the flexibility of the RS dataflow to deal with a more diverse set of DNNs.

The improved dataflow, named Row-Stationary Plus (RS+), is defined in Fig. 10. *The computation of each row convolution is still stationary in the SPad of each PE*, as $f0$ and $s0$ always loop through the entire dimension of F and S, respectively. The key features of the RS+ are summarized as follows:

- Additional loops $g1$ and $n1$ are added at the NoC level compared to the RS dataflow, which provides more options to parallelize different dimensions of data for processing. In fact, since row-stationary always fully loops through data dimensions F and S at the SPad level in loops $f0$ and $s0$, respectively, all the rest of the data dimensions are provided at the NoC level as options for parallel processing. As an example, one type of layer that benefits the most from this is the depth-wise (DW) CONV layer [10], in which the number of input and output channels are both one (i.e., $C = M = 1$). The RS dataflow cannot fully utilize the PE array due to the lack of channels, while the RS+ dataflow adapts better as shown in Fig. 11.
- Not only does the RS+ dataflow allow the parallelization of processing from more data dimensions, it allows data to be tiled from multiple dimensions and mapped in parallel onto the same PE array dimension (i.e., height or width) at the same time, which is similar to the idea of mapping replication introduced in Eyeriss [4]. For example, an PE array with height of 16 can be fully utilized by mapping both $C1 = 4$ and $M1 = 4$ simultaneously onto the height of the array.
- The data tile from the same dimension can also be mapped spatially onto different physical dimensions. For example, a $M1$ of 16 can be split into $M1_{horz} = 4$ and $M1_{vert} = 4$, which are the portions of $M1$ that are mapped horizontally and vertically on the PE array, respectively. This creates more flexibility for the mapper to find a way to fully utilize the PE array.
- In the RS dataflow, all rows of the filter are mapped spatially in order to exploit the available data reuse in the 2D convolution. However, this can create spatial mapping fragmentation if the PE array height is not an integer multiple of the filter height, and results in lower utilization of the PE array. In the RS+ dataflow, this restriction is relaxed by allowing tiling in data dimension R with loop $r1$. When the mapping is optimized for performance, the mapper can find the R1 that best fits in the PE array height. However, when the mapping is optimized for energy efficiency, the mapper can still find the same mapping as in the RS dataflow by setting $R1 = R$.
- An additional loop $e0$ is added at the SPad level compared to the RS dataflow, which concatenates $E0$ feature map rows to be computed with the same row of weights in a PE, therefore creating more reuse of weights in the SPad. However, this adds a storage requirement for the additional fmap rows to be stored in the global buffer. Therefore, $E0$ is constrained by the size of the global buffer.



(a) DW CONV: RS Mapping (b) DW CONV: RS+ Mapping

Fig. 11. The mapping of depth-wise (DW) CONV layers [10] with the (a) RS and (b) RS+ dataflows.

In summary, the RS+ dataflow provides a much higher flexibility in mapping than the RS dataflow. In fact, the mapping space of the RS+ dataflow is a strict super-set of the RS dataflow, i.e., the optimal mapping of the RS+ dataflow for a

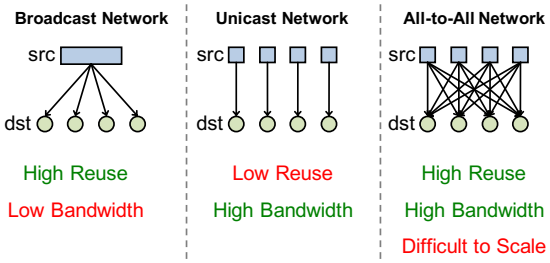


Fig. 12. The pros and cons of different NoC implementations.

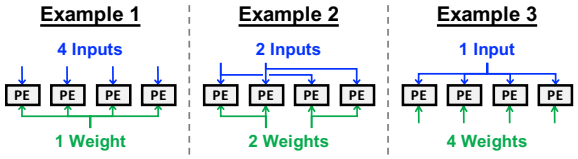


Fig. 13. Example data delivery patterns of the RS+ dataflow.

DNN layer with any optimization objective is at least the same or better than the optimal mapping of the RS dataflow. With a more powerful dataflow, the remaining challenge is to deliver data to the PEs according to the mapping for processing. In the next section, we will describe a new NoC that can unleash the full potential of the RS+ dataflow.

IV. Flexible Network: Hierarchical Mesh NoC

NoC design is a well-studied field in various contexts, e.g., manycore processors [12]. However, the complexity of a core in a multicore processor is much higher than that of a PE, which usually has specialized datapaths with little control logic, and the memory hierarchy is also highly customized; thus, a DNN processor cannot use the conventional sophisticated NoC used to connect cores on multicore processor [15]. Accordingly, most DNN processors adopt the NoC implementations with minimum routing and flow control complexity, such as the ones shown in Fig. 5, and it is important to keep it that way to maintain the efficiency of the architecture.

However, these NoC implementations have their drawbacks as shown in Fig. 12. On the one hand, the broadcast network can achieve high spatial data reuse. More importantly, it allows any patterns of data reuse, making it possible to be used for any dataflow. However, the data bandwidth from the source (e.g., the global buffer) is quite limited. If the amount of data reuse is low, i.e., different destination PEs require unique data for processing, the broadcast NoC has to deliver data to different destinations sequentially, resulting in reduced performance. Even though it is possible to increase the data bus width to deliver more data to the same destination at once, it would create high buffering requirements at each destination. This solution also does not scale as the buffering requirement will go higher with the number of destinations. On the other hand, the unicast network can achieve high bandwidth from the source by leveraging many independent sources, e.g., banked memory. However, it cannot exploit spatial data reuse, which will reduce the energy efficiency. A possible solution is the all-to-all network, which has the ultimate flexibility that can adapt between delivering high spatial data reuse and high data bandwidth from the source. However, it is very hard to scale as both the implementation cost and energy consumption will increase quadratically with the number of sources/destinations.

In order to build a NoC that is both adaptive and easy to scale, it is important to first understand the specific requirements from the dataflow, which is RS+ in this case. From the features described in Section III, we can summarize the types of data delivery patterns required by the RS+ dataflow in the examples shown in Fig. 13. These three examples show four data delivery patterns:

- **Broadcast:** weights in Example 1 & inputs in Example 3
- **Unicast:** inputs in Example 1 & weights in Example 3
- **Grouped Multicast:** weights in Example 2
- **Interleaved Multicast:** inputs in Example 2

Note that the patterns always come in a pair, and there are two possible combinations. The broadcast-unicast pair is required when the data dimensions mapped onto the same physical dimension of the PE array only address one data type. For example, when only the output channel (M) dimension is mapped onto the entire PE array width, the pattern will be the same as in Example 3, where the same input activation is broadcast across the PEs to be paired with unique weights from different output channels. The multicast pair is required when multiple data dimensions are mapped onto the same physical PE array dimension, and each data type is addressed by a unique subset of these data dimensions. For example, when the input fmap height (E) and output channels (M) are mapped simultaneously onto the width of the PE array, the pattern will be similar to the one in Example 2. In order to support a wide range of mappings, it is critical to be able to support all four data delivery patterns.

One possible solution that has the potential to support these data delivery patterns and is easy to scale is the mesh network. The mesh network can be constructed by taking the unicast network, inserting a router in between each pair of source and destination, and linearly connect the routers. Fig. 14 demonstrates how the mesh network can be configured to support each of the data delivery patterns. While it can easily support broadcast, unicast and grouped multicast, the interleaved multicast will cause a problem as the middle route between the routers (colored in black) needs a higher bandwidth than other routes. The number of routes with higher bandwidth requirement and the required bandwidth itself also grow with the size of the mesh network. Therefore, the mesh network alone is still not the answer.

To solve this problem, we propose a new NoC based on the mesh network, called a hierarchical mesh network. Fig. 15 shows a simple example of a 1D hierarchical mesh. It has the following features:

- The architectural components, including sources, destinations and routers, are grouped into clusters. The size

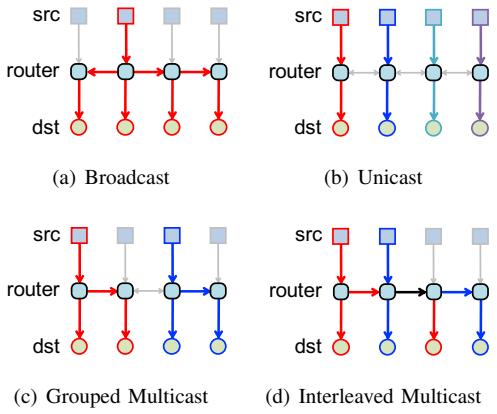


Fig. 14. Configurations of the mesh network to support the four different data delivery patterns. In a DNN processor *source(src)* is GLB and *destination(dst)* is a PE.

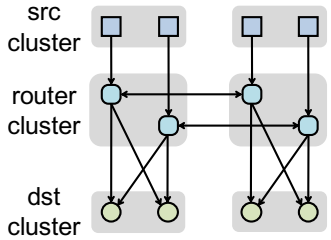


Fig. 15. A simple example of a 1D hierarchical mesh network. In a DNN processor, the sources could be the GLB, and the destinations could be the PEs.

of each type of cluster is determined at design time and fixed at compile time and runtime.

- The router clusters are connected linearly as in a mesh network. The individual routers in between adjacent clusters are one-to-one connected.
- Each router cluster is connected with a source cluster with one-to-one links between each pair of source and router.
- Each router cluster is also connected with a destination cluster. The links between the routers and destinations in the cluster are all-to-all connections.

Fig. 16 shows how the hierarchical mesh network supports the four data delivery patterns. It is able to support all four patterns by explicitly defining the bandwidth between all types of clusters through setting the size of these clusters at design time. Compared with the plain mesh network, only the all-to-all network in between the router cluster and the destination cluster incurs a higher cost, and this cost can be well-controlled locally. When setting the size of these clusters, the key characteristics to consider are (1) what is the bandwidth required from the source cluster in the worst case, i.e., unicast mode, (2) what is the bandwidth required in between the router clusters in the worst case, i.e., interleaved multicast mode, and (3) what is the tolerable cost of the all-to-all network.

The hierarchical mesh network has two advantages. First, there is no routing required at runtime. All active routes

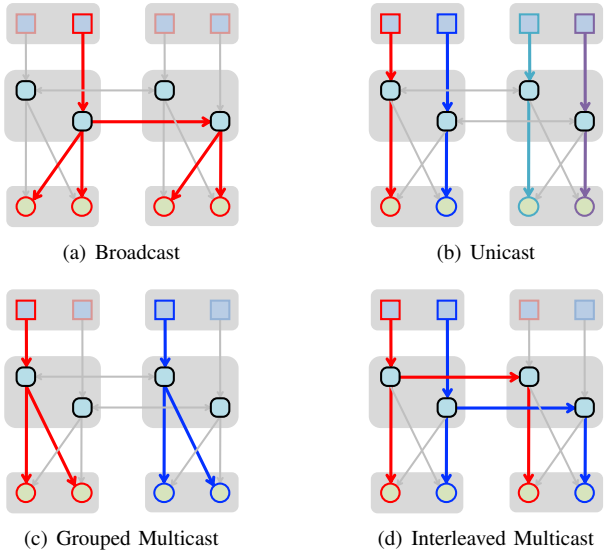


Fig. 16. Configurations of the hierarchical mesh network to support the four different data delivery patterns.

are determined at configuration time based on the specific data delivery pattern in use. As a result, no flow control is required, and the routers are simply multiplexers for circuit-switched routing that has minimum implementation cost. Second, the network can be easily scaled. Once the cluster size is determined, the entire architecture can be scaled at the cluster level, in which case the cost only increases linearly instead of quadratically as in the plain all-to-all network.

One restriction the hierarchical mesh network imposes on the mapping space of the RS+ dataflow is that, in addition to the total height and width of the PE array, the tile size of the mapped data dimensions is further constrained by (1) the cluster size and (2) the number of clusters. For example, in the multicast delivery patterns, the data delivered with grouped multicast has the maximum reuse constrained by the size of the PE cluster. Similarly, the data delivered with interleaved multicast has the maximum reuse constrained by the number of clusters. In Section V-B, we will discuss how does this affect the performance of the architecture.

Fig. 17 shows an example DNN accelerator built based on the hierarchical mesh network. The router clusters are now connected in a 2D mesh. The global buffer is banked and distributed into each source cluster, and the PEs are grouped into the destination clusters instead of one single array.

V. Evaluation Results

In this section, we profile the performance of the co-design of the RS+ dataflow and the hierarchical mesh network. We will first describe the methodology used to conduct the experiment in Section V-A, and then demonstrate and discuss the experiment results in Section V-B.

A. Experiment Methodology

The area and energy numbers are obtained from synthesized RTL implementations of both Eyeriss v2 and Eyeriss v1 [4],

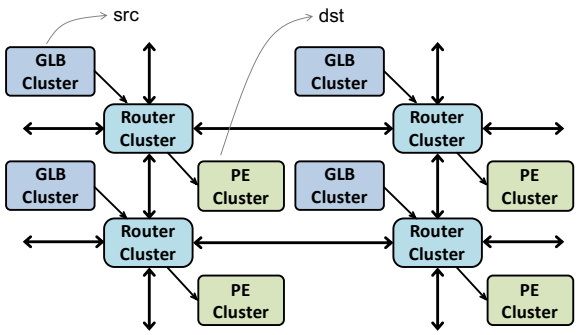


Fig. 17. A DNN accelerator architecture built based on the hierarchical mesh network.

including the PE, NoC and memory hierarchy, using Synopsys Design Compiler in a 65nm CMOS process. The performance results are obtained through an analytical model based on Eyexam. It generates mappings through exhaustively searches the mapping space and then profiles the corresponding number of active PEs. It then takes the limited data delivery bandwidth into account with the adapted roofline models for each data type, which are determined with the configuration of the Eyeriss v2 NoCs.

B. Experiment Results

In this section, we will examine the performance of the combination of the RS+ dataflow and the hierarchical mesh network, named Eyeriss v2, and compare it to the combination of the RS dataflow and the broadcast network (with multicast capabilities) implemented in Eyeriss v1 [4]. First, we will compare them in terms of the number of active PEs from the optimal mappings generated by their respective dataflows. Then, we will discuss the impact of data bandwidth on the performance.

For each architecture, we simulate with three different PE array sizes, including 256, 1024 and 16384 PEs. For Eyeriss v1, the PE array is a square, i.e., 16×16, 32×32 and 128×128. For Eyeriss v2, we fix the PE cluster size at 4×4, and scale the number of PE clusters at 4×4, 8×8 and 32×32. In terms of data bandwidth, Eyeriss v1 uses a single broadcast network for each of the three data types, which is capable of delivering 1 data/cycle. For Eyeriss v2, each data type has a separate hierarchical mesh network with a router cluster size of 4. Each pair of source to router link can deliver 1 data/cycle. We assume a PE architecture similar to the one described in [4], which has the SPad sizes for input activation, weight and psum at 12, 192 and 16, respectively. The PE is also assumed to be able to sustain a processing throughput of 1 MAC/cycle if not bandwidth limited. Each PE cluster in Eyeriss v2 is accompanied with a GLB cluster of size 11.25 kB. Therefore, the total GLB size are 180 kB, 720 kB and 11520 kB for the PE array size of 256, 1024 and 16384, respectively. In this setup, we observe that the area of the hierarchical mesh network accounts for only 1.5% of the total area. For Eyeriss v1, we allocate the same amount of GLB size at each PE array size.

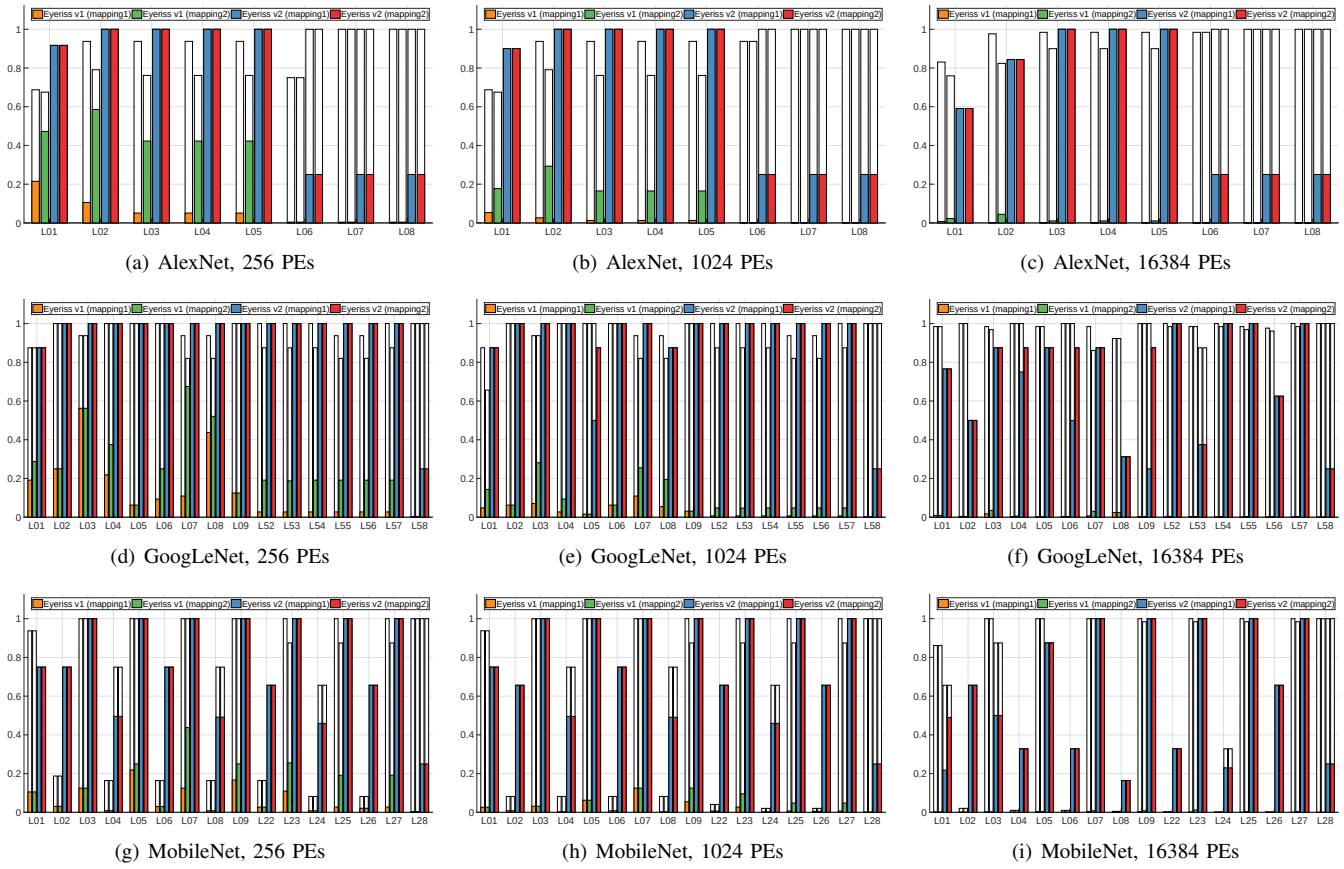
We evaluated the architectures with three different DNNs, including AlexNet [14], GoogLeNet [23] and MobileNet [10]. Each layer in the three DNNs is mapped on the two architectures for processing independently, and two types of performance are quantified: (1) *the number of active PEs*, which is the performance at step 5 of Eyexam assuming an infinite data bandwidth. (2) *the overall utilization of the PE array*, which further models the impact of a finite bandwidth on the performance and is the performance at step 6 of Eyexam. We serialize the layers in each DNN and name them starting from L01. The layers in the inception modules of GoogLeNet are serialized in the following order: 3×3 reduction, 5×5 reduction, 1×1 CONV, 3×3 CONV, 5×5 CONV and 1×1 CONV after pooling. In all cases we use a batch size of 1, which is a crucial criterion for many low-latency applications but also greatly reduces the reuse of weights.

Fig. 18 shows the performance comparison between Eyeriss v1 and Eyeriss v2 for each DNN layer in AlexNet, GoogLeNet and MobileNet, at different PE array sizes. For each PE array size, the performance in the y-axis is normalized to its total number of PEs (i.e., peak performance). The total bar height (white + colored portion) indicates the performance accounting for the impact of workload and architectural constraints excluding the impact of the NoC bandwidth, i.e., it is the performance assuming all active PEs run at 100% utilization (step 5 of Eyexam). The color-only portion of the bars indicates the performance further accounting for the impact of the finite bandwidth from the specific NoC design, which reduces the utilization of the active PEs and represents the overall utilization of the PE array (step 6 of Eyexam). Therefore, the white portion of the bars, if any, indicates the performance loss due to the constraint of a finite NoC bandwidth.

For each combination of the architecture and DNN layer, we generate the optimal mappings for the following two different objectives:

- *Mapping 1* is optimized to get the highest number of active PEs regardless the actual utilization of them, i.e., it is optimized for the overall bar height (white + color).
- *Mapping 2* is optimized to get the best overall utilization of the PE array that further accounts for the impact of the finite bandwidth on performance, i.e., it is optimized for the height of the colored-only bar.

First, we compare only the *number of active PEs*, i.e., total bar height, of the two architectures. In most cases, Eyeriss v2 shows a better performance than Eyeriss v1 except for a few cases at the PE array size of 16384. In these cases, the performance degradation in Eyeriss v2 is because the number of clusters becomes too large while the cluster size is kept small. A small cluster size ensures that the implementation cost of the all-to-all network is limited. As mentioned in Section IV, the hierarchical mesh network imposes mapping constraints due to its two-level structure. It requires a high amount of reuse in one data type to fully map the large number of clusters, while the other data type can only exploit reuse up to the



(a) AlexNet, 256 PEs (b) AlexNet, 1024 PEs (c) AlexNet, 16384 PEs

(d) GoogLeNet, 256 PEs (e) GoogLeNet, 1024 PEs (f) GoogLeNet, 16384 PEs

(g) MobileNet, 256 PEs (h) MobileNet, 1024 PEs (i) MobileNet, 16384 PEs

Fig. 18. Performance of AlexNet at the PE array size of (a) 256, (b) 1024, and (c) 16384, GoogLeNet at the PE array size of (d) 256, (e) 1024, and (f) 16384, and MobileNet at the PE array size of (g) 256, (h) 1024, and (i) 16384. The y-axis is the performance normalized to the total number of PEs. The top of the white bar indicates the performance accounting for impact of of workload and architectural constraints up to step 5, which reduce number of active PEs. The color bars indicate the performance accounting for the impact of finite bandwidth which reduces the utilization of the active PEs (step 6). Performance is normalized to the peak performance.

amount of the cluster size. Eventually, this can lead to spatial mapping fragmentation, which reduces the number of active PEs. Eyeriss v1, on the other hand, may adapt better at large PE array sizes in terms of number of active PEs since multiple layer dimensions can be mapped onto the same physical dimension of its PE array even with less number of data dimensions to choose from for spatial mapping. This phenomenon is more significant in GoogLeNet than in AlexNet, since the layers in AlexNet are usually much larger than that in GoogLeNet and have plenty of use for all data types, which results in less mapping fragmentation. In general, mapping fragmentation is also less severe in smaller PE array sizes, and the more flexible spatial mapping of the RS+ dataflow gives Eyeriss v2 an edge over Eyeriss v1. Specifically, the RS+ dataflow handles the mapping in the DW CONV layers of MobileNet much better than the RS dataflow as shown in the even layers of MobileNet (except for L28, which is a FC layer). However, the Eyexam framework in Section II shows that the number of active PEs is not a sufficient metric to evaluate the final performance of the hardware, since the actual utilization of the active PEs are not taken into account yet.

Next, we compare the performance in terms of the *overall utilization of the PE array*, which is the colored-only portion of the bars. This comparison shows a drastic performance difference between the two architectures. For Eyeriss v2, most of the time it can keep the overall utilization of the PE array up to the level of the number of active PEs. However, for Eyeriss v1, there is usually a big gap between the number of active PEs and the overall utilization of the PE array. The performance difference grows even higher when the PE array size scales. In certain cases, however, Eyeriss v2 still loses performance due to the finite bandwidth. For example, in the FC layers, e.g., L6 to L8 in AlexNet, L58 in GoogLeNet and L28 in MobileNet, the overall utilization of the PE array is only one fourth of the number of active PEs. The performance bottleneck comes from the insufficient bandwidth for delivering weights. In our setup, each weight router cluster has 4 routers and connects to 4 GLB banks and 16 PEs, which means that 4 PEs in a cluster share the same GLB bank for weight delivery. In the case when batch size is 1, there is no weight reuse, and therefore the performance is reduced by 4 times. Overall, however, Eyeriss v2 still achieves much higher performance than Eyeriss v1.

Finally, we compare the impact of having the mapping being optimized for different objectives. For Eyeriss v1, mapping 1

usually results in a higher number of active PEs than mapping 2; however, mapping 2 still shows a higher overall utilization of the PE array than mapping 1. This shows that optimizing for the maximum number of active PEs does not necessarily yield the best performance after considering the finite bandwidth, especially when the deliverable bandwidth is low. This is mainly because the mapping often relies on certain layer dimensions that can provide a large tile to fully utilize the high parallelism. However, this also results in a higher bandwidth requirement for a specific data type. Instead, optimizing the mapping according to the actual overall utilization of the PE array, mapping 2 takes the bandwidth constraints into account and avoids placing too much pressure on the bandwidth of certain data types. For example, in AlexNet L01 with a PE array size of 256, mapping 1 relies on a large number of output channels to fill the parallelism, which results in a high bandwidth requirement for weights, while mapping 2 has a more balanced tile size between the input and output channels, and therefore distributes the bandwidth requirement between weights and input activations. This phenomenon, however, is less prominent in Eyeriss v2. Since Eyeriss v2 can provide a more scalable data bandwidth, optimizing for the number of active PEs is usually enough to guarantee a high overall utilization of the PE array.

Table III quantifies the performance speedup of Eyeriss v2 over Eyeriss v1 in terms of the overall utilization of the PE array. For each combination of DNN and PE array size, it shows the range of speedup along with the averages across the layers in the same DNN. At the end of each column, it also shows the average speedup across all layers in the three DNNs at the same PE array size. The FC layers achieve the highest speed up, as the increase in bandwidth for delivering weights has a significant impact on performance. Another highlight is the performance of the DW CONV layers in MobileNet, which receives a speedup ranging from 25.4× (256 PEs) to 997.5× (16384 PEs).

		256 PEs	1024 PEs	16384 PEs
AlexNet	range	4.3×–64.0×	16.8×–256.0×	80.0×–4096.0×
	average	33.1×	132.4×	2082.6
	weighted average	17.9×	71.5×	1086.7×
GoogLeNet	range	1.8×–64.0×	9.1×–256.0×	13.1×–4096.0×
	average	17.3×	65.7×	757.0×
	weighted average	10.4×	37.8×	448.8×
MobileNet	range	3.0×–64.0×	8.0×–256.0×	22.4×–4096.0×
	average	26.1×	101.0×	1083.2×
	weighted average	15.7×	57.9×	873.0×
Overall	average	21.3×	81.9×	967.0×
	weighted average	13.3×	50.3×	693.3×

TABLE III
PERFORMANCE SPEEDUP OF EYERISS V2 OVER EYERISS V1. THE AVERAGE SPEEDUP SIMPLY TAKES THE MEAN OF SPEEDUPS FROM ALL LAYERS IN A DNN; THE WEIGHTED AVERAGE IS CALCULATED BY WEIGHTING THE SPEEDUP OF EACH LAYER WITH THE PROPORTION OF MACS OF THAT LAYER IN THE ENTIRE DNN.

	256 PEs	1024 PEs	16384 PEs
AlexNet	1.15× (1.1%)	0.98× (1.3%)	0.78× (1.8%)
GoogLeNet	1.19× (1.5%)	0.97× (1.9%)	0.90× (3.8%)
MobileNet	1.30× (1.3%)	1.01× (2.0%)	0.77× (4.6%)

TABLE IV
ENERGY CONSUMPTION OF EYERISS V2 COMPARED WITH EYERISS V1. THE PERCENTAGE SHOWS THE PROPORTION OF ENERGY CONSUMED IN THE HIERARCHICAL MESH NETWORK IN EYERISS V2.

Table IV shows the energy consumption of Eyeriss v2 compared with Eyeriss v1. At 256 PEs, Eyeriss v2 consumes 15% to 30% higher energy than Eyeriss v1 for over 10×speedup. This is due to the mapping being optimized to maximize the overall performance, which sometimes sacrifices the energy efficiency. However, at a larger number of PEs, the larger mapping space makes it easier to find the mappings that can achieve even higher performance while improving energy efficiency by up to 23%. The hierarchical mesh network is able to deliver data at high throughput while consuming less than 5% of the total energy due to its scalability.

VI. RELATED WORK

Designing accelerators for DNN has been a very active field of research over the past few years [1], [6], [13], [18], [19], [27]. Previous work that explored flexible hardware for DNNs include FlexFlow [17], DNA [25] and Maeri [16], which propose methods to support multiple dataflows within the same NoC. Rather than supporting multiple dataflows, Eyeriss v2 proposes using a single but highly flexible dataflow, RS+, that can efficiently support a wide range of layer shapes to maximize the number of active PEs and overall performance. In addition, previous work primarily address DNNs such as AlexNet and VGG rather than compact DNNs such as MobileNet or rely on a batch size of 64 to increase the reuse when processing GoogLeNet. In our work, we *do not* rely on modifying the workload (e.g., we can use batch size of 1 for the compact models, which may be desirable for certain low latency applications). Instead, our RS+ dataflow is sufficiently flexible to efficiently map challenging layer shapes such as depth-wise layer and bottleneck, and in the event that there is not enough reuse, our scalable NoC is able to deliver sufficient bandwidth to keep the PEs busy.

Chen et al. [28] explores various optimization techniques, such as loop tiling and transformation, to map a DNN workload onto an FPGA, and then uses the roofline model of the FPGA to identify the solution with best performance and lowest resource requirement. Jouppi et al. [13] also use a roofline model to illustrate the impact of limited bandwidth on performance. In this work, we propose a method called Eyexam to *tighten the bounds of the roofline model* based on the various architectural design choices and their interaction with the given DNN workload. We also adapt the roofline model for DNN processing by accounting for the fact that different data types (iact, weight, psums) will have different NoC and bandwidth and thus require three separate roofline models, and normalized the operational

intensity to be in terms of MACs/cycle. We then use the adapted roofline model to design Eyeriss v2.

VII. Conclusions

DNNs are rapidly evolving due to the significant amount of research in the field. In order to keep up, it is imperative that hardware architectures do not rely on specific properties of any particular DNN to achieve high performance or energy efficiency. Instead, it is critical that the DNN processors stay sufficiently flexible such that they can adapt to the wide range of potential DNNs, and exploit combinations of different properties for improved efficiency.

In this work, we proposed Eyeriss v2, which provides this much needed flexibility to efficiently process a wide range of DNNs, from large ones with high reuse (e.g., AlexNet) to compact ones with high bandwidth requirements (e.g., MobileNet). It addresses the key need to provide efficient mapping of different layer shapes in order to increase the number of active PEs. Unlike previous works that pin certain data dimensions of a DNN layer to each side of the PE array, Eyeriss v2 uses a RS+ dataflow that can tile the data from any dimensions and map it to the PE array to ensure high utilization. Furthermore, in order to increase the compute intensity of each active PE, it uses a hierarchical mesh network that allows the system to exploit spatial data reuse for large DNNs and deliver high bandwidth for compact DNNs. At the same time, the hierarchical structure of the NoC makes it much more scalable than an all-to-all NoC. The highly flexible hardware architecture also opens up a larger design space for future algorithm exploration.

Finally, the number of DNN architecture designs are also growing rapidly. It can often be challenging to evaluate the impact of various architectural design decisions. To address this, we propose Eyexam, which provides a seven step process to systematically identify the sources of performance loss and can be used to develop a set of roofline models to assess the impact of bandwidth constraints. In this paper, we show how Eyexam can be used to highlight the limitations of existing hardware architectures, and then how the output of this framework can be used to guide the design of Eyeriss v2. Moving forward, we hope that this framework can be used by computer architects to analyze different existing designs to put them into perspective, as well as point out important directions for future DNN processor designs.

References

[1] 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 *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2014, pp. 269–284.

[2] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[3] Y.-H. Chen, J. Emer, and V. Sze, "Using Dataflow to Optimize Energy Efficiency of Deep Neural Network Accelerators," *IEEE Micro's Top Picks from the Computer Architecture Conferences*, vol. 37, no. 3, May-June 2017.

[4] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, pp. 127–138, 2016.

[5] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Understanding the limitations of existing energy-efficient design approaches for deep neural networks," in *SysML*, 2018.

[6] 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 *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, 2015, pp. 92–104.

[7] C. Farabet, C. Poulet, and Y. LeCun, "An FPGA-based Stream Processor for Embedded Real-time Vision with Convolutional Networks," in *ICCV Workshop on Embedded Computer Vision*, 2009.

[8] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep Learning with Limited Numerical Precision," in *ICML*, 2015.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[11] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size," *arXiv:1602.07360*, 2016.

[12] N. E. Jerger and L.-S. Peh, "On-chip networks," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–141, 2009.

[13] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM, 2017, pp. 1–12.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

[15] H. Kwon, A. Samajdar, and T. Krishna, "Rethinking nocs for spatial neural network accelerators," in *Proceedings of the Eleventh IEEE/ACM International Symposium on Networks-on-Chip*. ACM, 2017, p. 19.

[16] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2018, pp. 461–475.

[17] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 553–564.

[18] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," in *Symp. on VLSI*, 2016.

[19] Nvidia, "NVDLA Open Source Project," 2017. [Online]. Available: http://nvdla.org/

[20] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks," in *Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2017.

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

[22] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec 2017.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper With Convolutions," in *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016.

[25] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2220–2233, 2017.

[26] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, Apr 2009.

[27] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, L. Liu, and S. Wei, "A 1.06-to-5.09 tops/w reconfigurable hybrid-neural-network processor for deep learning applications," in *VLSI Circuits, 2017 Symposium on*. IEEE, 2017, pp. C26–C27.

[28] 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.