A Communication-centric Approach for Designing Flexible DNN Accelerators

Hyoukjun Kwon
Ananda Samajdar
Tushar Krishna
Georgia Institute of Technology

High computational demands of DNNs coupled with their pervasiveness across cloud and IoT platforms have led to the emergence of DNN accelerators employing hundreds of processing elements (PE). Most DNN accelerators are optimized for regular mapping of the problems, or dataflows, emanating from dense matrix multiplications in convolutional layers. However, continuous innovations in DNN including myriad layer types/shapes, cross-layer fusion, and sparsity have led to irregular dataflows within accelerators, which introduces severe PE underutilization because of rigid and tightly-coupled connections among PEs and buffers. To address this challenge, this article proposes a communication-centric approach called MAERI for designing DNN accelerators. MAERI's key novelty is a light-weight configurable interconnect connecting all compute and memory elements that enables efficient mapping of both regular and irregular dataflows providing near 100% PE utilization.

Keywords: Deep Neural Networks, Accelerators, Dataflow, Network-on-chip

INTRODUCTION

The microarchitecture of DNN inference engines is currently an area of active research in the computer architecture community. Graphics processing units (GPUs) are extremely efficient for training due to the mass parallelism they offer, multi-core central processing units (CPUs) continue to provide platforms for algorithmic exploration, and FPGAs provide power-efficient and configurable platforms for algorithmic exploration and acceleration; but for mass deployment across various domains (smartphones, cars, etc.), specialized DNN accelerators are necessary to maximize performance per watt. This observation has led to a flurry of ASIC proposals for DNN inference accelerators over the recent years [1, 2, 4-8].

DNNs have millions of parameters, which needs to be mapped over limited compute and memory resources in the DNN accelerator. We define each unique mapping as a "***dataflow***". The dataflow determines the degree of data reuse (which is critical to throughput and energy efficiency of the accelerator), and determines the communication pattern between the compute and memory elements. We identify three sources that affect dataflow – DNN topology, DNN dataflow graph (DFG) partitioning and mapping, and DFG transformations. On the topology front, DNNs today use a mix of convolution, recurrent, pooling, and fully-connected layers; new layer types such inception [3], dilated convolution [13], and transposed convolution [14] are also being actively explored. DNN dataflow graph (DFG) can be partitioned in myriad ways to map over the compute array – layer by layer [1], across layers [4], kernels, channels, or outputs, as each approach has different trade-offs for data reuse and energy-efficiency. The DFG can also be transformed by removing some of the edges whose weights are zero or close to zero [5-6] to reduce power consumption inside DNN accelerators. Naturally, each of these three approaches can lead to unique and often irregular (especially with fused layer [4] or sparsity optimizations [5-6]) dataflows. Dataflows directly impact the performance and energy-efficiency of accelerators, as they have a direct impact on the amount of data movement, data reuse, and memory accesses, as prior works have shown [7, 15].

Unfortunately, state-of-the-art DNN accelerators today, such as Google TPU [8], Eyeriss [1], and their variants, employ dense 2D arrays – which are optimized for a very regular dataflow, namely dense matrix multiplications present in convolution operations. Irregular dataflows can lead to heavy underutilization (e.g., some long short-term memories (LSTMs) only utilize 4% of the MAC units on the TPU [8]). We claim that the reason DNN accelerators today cannot map arbitrary dataflows is because the interconnection fabric connecting the MAC units are either limited in terms of their connectivity (e.g., mesh), or their bandwidth (e.g., bus). Our community's approach to addressing this issue has been quite reactive – every new DNN topology/DFG mapping/sparsity optimization has led to a *new* DNN ASIC proposal [4,5,6]. This makes the hardening of DNN accelerators into an IP or a discrete chip that is future-proof for the rapidly evolving DNN landscape impractical.

How do we design a single accelerator substrate that can handle the growing number of dataflows resulting from multiple kinds of layers, dense and sparse connections, and various partitioning approaches? We propose to make the interconnects *within* the accelerator reconfigurable. Our insight is the following: The DNN DFG is fundamentally a multi-dimensional multiply-accumulate (MAC) calculation. Each dataflow is essentially some kind of transformation of this multi-dimensional loop [7]. Thus, at the heart of each dataflow that exists today or might be proposed in future, is still a collection of MAC operations spread across the processing engines. We propose to design DNN accelerators as a collection of multiply and adder engines, each augmented with tiny configurable switches (called switchlets) that can be configured to support different kinds of dataflows. Our design is called MAERI (Multiply-Accumulate Engine with Reconfigurable Interconnect). MAERI is a communication (rather than a compute)-centric approach for designing DNN accelerators. Figure 1 shows an overview. ***It enables on-demand allocation of multipliers and adders depending on the dataflow by configuring the switches, thereby providing high compute utilization.*** It also provides high-bandwidth non-blocking interconnect topologies tailored to the communication patterns within DNN accelerators for maximizing data reuse and minimizing stalls.

We demonstrate MAERI with multiple case studies. On average, MAERI reduces the run time by 42% and 57% over Eyeriss [1], and 23% and 148% over Systolic Arrays [8] for state-of-the-art CNNs and LSTMs respectively. This translates to energy reductions of up to 36%.

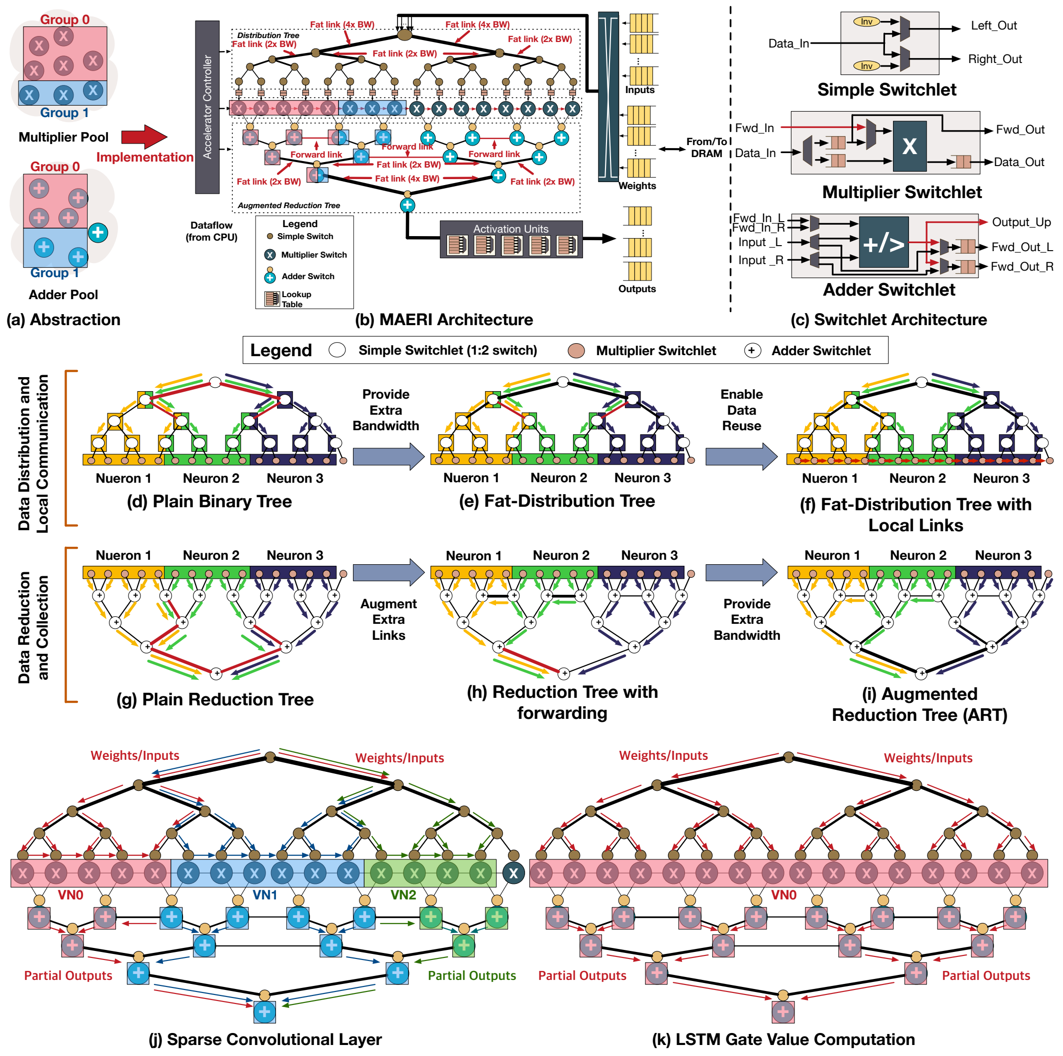

Figure 1: (a) Compute unit pool abstraction provided by MAERI (b) MAERI architecture to support the abstraction, and (c) Switchlet architectures. (dNetworks in MAERI for data distribution (fat tree with local forwarding at leaves) to the multiplier switchlets (a-c) and collection (fat-tree with local forwarding at upper levels) to the buffer (d-f). Red links indicate insufficient bandwidth or link conflicts. (g) and (h) show examples of mapping irregular dataflows over MAERI.

BACKGROUND AND MOTIVATION

Deep Neural Networks (DNNs)

Neural networks are a rich class of algorithms that can be trained to model the behavior of complex mathematical functions. Neural networks model human brain with a large collection of “neurons" connected with “synapses." Each neuron is connected with many other neurons and its output enhances or inhibits the actions of the connected neurons. The connection is based on weights associated with synapse. Therefore, computations for a neuron can be translated as weighted sum operations. Deep neural networks (DNNs) have multiple internal (called hidden) neuron layers before the final output layer that performs the classification. DNNs also involve pooling and activation operations. Pooling is a sampling operation that reduces output feature map dimensions by the pooling window size. Activation is a non-linear functions such as rectifier linear units (ReLU). Pooling and activation operations follow after some of hidden layers in a DNN. Therefore, the weighted sum consists of the majority of computations in a

DNN.

Spatial DNN Accelerators

The most dominant computation in DNNs, a weighted-sum, contains massive parallelization opportunity for element-wise multiplications and accumulations. To exploits this parallelism, spatial architectures with hundreds of processing elements (PE) fed by a global buffer have been the most popular hardware substrate for accelerating DNNs [1-2, 4-9]. The PEs consist of fixed- or floating-point multiply-accumulate-unit (MAC) and local scratch pad memory usually implemented using SRAMs. Global buffer is also a SRAM-based scratchpad memory, but it has larger size than local scratch pad memories in PEs. Unlike GPUs, spatial architectures support direct PE to PE communication, enabling *data reuse* both within and across PEs, reducing the number of memory accesses, and thereby energy consumption.

Network-on-Chip (NoC)

To support data reuse, spatial DNN accelerators use an on-chip communication substrate between the PEs and the global buffer, called a network-on-chip (NoC). Buses, meshes and crossbars are the most popular NoC topologies. Buses are cheap to implement but provide extremely low bandwidth (as they support only one transmission at a time) while crossbars provide extremely high bandwidth but scale horribly in terms of area and power; meshes provide a reasonable trade-off between these extremes and are used extensively in modern multi-core CPUs. Mesh NoCs provide all-to-all connectivity that is required for cache coherence traffic. However, they provide a skewed bandwidth distribution across the NoC links for DNN traffic, as highlighted by prior work [9], leading to PE stalls. Moreover, their area and power is proportional or higher than that of the PEs themselves, making them an inefficient design choice. To mitigate these issues, most DNN accelerators today tightly couple a fixed number of PEs together using buses [1] or trees [2, 3] or neighbor-to-neighbor connections [8], and scale up by using hierarchical NoCs [6]. Unfortunately, this approach naturally leads to under-utilization of PEs for irregular dataflows due to the inflexibility of mapping across arbitrary number of PEs, and leads to stalls if the connections leading to the PEs do not support the required bandwidth for that dataflow. This work addresses these challenges by designing an extremely light-weight communication fabric with the right amount of flexibility to support arbitrary DNN dataflows, without requiring a full-fledged NoC.

APPROACH AND DESIGN

Communication Classes in DNN Accelerators

We classify communication flows within DNN accelerators [9] into the following traffic classes:

- **Distribution:** Distribution is communication from global buffer to PEs, which delivers weights and input activations to be computed in each PE. Distribution requires multiple simultaneous unicasts or multicasts (depending on the accelerator implementation) from the global buffer to the PEs.
- **Local forwarding:** Local forwarding is direct communication (forwarding) between PEs for data reuse in convolutional layers, reducing the number of reads from the global buffer. Local forwarding requires multiple simultaneous unicasts between PEs.
- **Reduction:** Reduction is direct communication between PEs for accumulating partial sums. Each reduction requires multiple simultaneous unicasts between PEs.
- **Collection:** Collection is communication from PEs to the global buffer to deliver final output activations reduced within the PE array. Collection requires multiple unicasts from one or more PEs (depending on how many unique outputs the array generates based on the mapping strategy) to the global buffer.

We leverage these traffic classes to construct a communication-driven spatial DNN accelerator architecture.

Designing a Fully-flexible DNN Accelerator

Supporting full PE utilization with myriad irregular DNN dataflows essentially requires DNN accelerators to support *customizable grouping of compute resources at runtime*. To enable the illusion of multiplier/adder pools for fully flexible DNN accelerator described in Figure 1 (a), we separate out the multipliers and adders from PEs to enable fine-grained mapping and embed them into tiny switches to enable customizable groupings. We name the switches as *multiplier switchlet* and *adder switchlet*, based on their functional units, and the entire architecture as **MAERI** (Multiply-Accumulate Engine with Reconfigurable Interconnects), as shown in Figure 1 (b).

MAERI supports multiple dataflows by reconfiguring each switchlet to directly deliver its computation results to the desired destination switchlet for correct functionality. This approach programs data movement, not computation, unlike traditional computer systems, and allows users to fully explore optimization opportunities that involve irregular dataflow.

A key requirement for enabling this functionality is the right set of physical connections between the switchlets. Instead of using a full-fledged all-to-all NoC and pay its area, power, and latency overheads, we design a novel interconnection topology tuned for the traffic classes described in the previous section. MAERI uses two separate networks – a distribution network and a collection network, as shown in Figure 1. MAERI's interconnection is fat-tree-based hierarchical networks with additional forwarding links for non-blocking reductions, which we discuss next.

Distribution and Collection Networks with Local Forwarding

Based on the observation of traffic in DNN accelerators [9], we specialize NoC structures for each traffic pattern: distribution with local forwarding and collection with reduction.

Distribution Network

The distribution network connects the global buffer to all the multiplier switchlets, and is responsible for delivering inputs and filter weights. We list three features of our distribution network following the order described in Figure 1(d)-(f).

Binary Tree. To support multicasting, we employ a binary tree-based distribution network, as shown in Figure1(d). Although a bus also provides multicasting functionality (and has been used in prior accelerators [1]), tree-based distribution is more energy efficient because a tree activates only the links going towards the destinations while a bus broadcasts data all the time, even for unicasts.

Fat Links. To support multiple multicasts simultaneously, we provide higher bandwidth at the upper levels of the tree. When the bandwidth at each level is double the one at the lower level, the distribution network can support N/2 (where N is the number of leaves), which constructs a fat tree. This case is extremely beneficial to fully-connected layers because it requires unique data to be delivered to each PE at the same time, but comes at the cost of large wire overhead and more switches. Such a high bandwidth is, however, not required for convolutional layers where data reuse requires lower distribution bandwidth. MAERI allows designers to choose either the same bandwidth or twice at higher levels of tree at design-time, based on the area and power budget of the accelerator. This is shown in Figure 1(e).

Local Forwarding Links. The leaves (multiplier switchlets) in the distribution network are connected to each other directly, for local forwarding of weights between the multipliers, as shown in Figure 1(f). This reduces the number of reads from the global buffer, and reduces the bandwidth requirement from the distribution tree. Since we control the computation mapping across the multipliers, we only need unidirectional forwarding links.

Collection Network

The collection network performs multiple simultaneous reductions and delivers multiple output activations to the global buffer via activation units. We call our topology as an augmented reduction tree (ART). We list three features of ART following the order described in Figure 1(g)-(i).

Binary Tree. Binary-trees are well-suited for performing reductions and have been used in prior DNN implementations to implement adder trees within PE clusters, and form our strawman design. However, they have a key inefficiency: the fixed topology of a tree is inherently inefficient whenever the number of partial sums to be accumulated is smaller than the width of the adder tree. Figure 1(g) illustrates this. Suppose there are 16 multipliers, all connected via a 16-node binary reduction tree. Each node in the reduction tree is an adder. This tree is perfect for performing a reduction for a 16-input neuron. However, suppose we map three neurons over these multipliers, each generating five partial sums, as Figure 1 (g) shows. Each neuron requires four additions to generate an output, so the total additions required is 12. The reduction tree contains 16 adders, which should be sufficient to perform the additions for all neurons in parallel. However, the four links in red are shared by multiple neurons, limiting the tree from generating all three outputs simultaneously.

Augmented Forwarding Links. We introduce forwarding links at intermediate levels between adjacent adders that do not share the same parent, as Figure 1 (h) illustrates, to overcome the challenge of supporting multiple parallel reductions. Using these forwarding links, neuron 1 and 2 can perform reduction simultaneously. However, neuron 1 still conflicts with neuron 2 because of limited bandwidth near the root node highlighted as red in Figure 1 (h).

Fat Links. As Figure 1 (i) shows, we provide extra bandwidth near the root so that multiple reduction results can be delivered to the root node connected with global buffer. With this final feature, we construct augmented reduction tree (ART), a key component of MAERI.

In the following section, we discuss how these network structures enable mapping arbitrary dataflow using case studies from DNN layers.

Mapping Regular and Irregular Dataflows over MAERI

To enable efficient mapping of regular and irregular dataflows on the same architecture, MAERI constructs instances called *virtual* neuron (VN) for each neuron and map them onto the computation resources.

Virtual Neurons

Because each weighted-sum operation corresponds to a neuron in each DNN, grouping compute units for a neuron is analogous to constructing a VN structure inside the DNN accelerators. Therefore, we name a group of compute units allocated for a neuron as a VN. Note that different dataflows can effectively be viewed as different neuron sizes; thus we can abstract various dataflows into the size of VNs. For CNNs, VN sizes for convolutional, pooling, and FC layers are weight filter size, pooling window size, and input activation size, respectively. For LSTMs, VN sizes for gate values, state, and output computation are (input size) +2, (input size) +2, 2, and 1, respectively. The problem of mapping different dataflows is analogous to mapping different sized virtual neurons over the multipliers and adders. Moreover, sparsity and cross-layer [4] mapping lead to multiple VN sizes at the same time.

Mapping Example

MAERI supports simultaneous mapping of VNs with different sizes, as shown in Figure 1(j) and (k). We present two mapping examples: a sparse convolution whose full filter size is 9 in Figure 1(j), and the forget/input/output gate computation of an LSTM in Figure 1(k). For sparse convolution, we map virtual neuron 0, 1, and 2 for weight channel 0,1, and 2 for the 6th convolutional layer of VGGnet-16 [10] and assume the number of non-zero weights are 5, 6, and 4, respectively. MAERI supports structured weight sparsity that does not require dynamic pair matching of input and weight values. Because weight values are known before runtime, we utilize a simple compiler technique to order input values corresponding to the structural sparsity we exploit.

For LSTM example, we map gate computation of a hidden layer in Google translation decoder LSTM [11]. Because the input size of a hidden layer is 1024, we fold it onto 16 multiplier switchlets in this example. The mapping (i.e., VN construction) is performed by the ART reconfiguration controller. The controller recursively inspects active multiplier switch for a virtual neuron and determines the use of each forwarding link, as described in supplementary material.

Dataflows and Mappings Supported by MAERI

With the capability of constructing arbitrary sized virtual neurons at runtime based on the actual neuron sizes, MAERI can map any DNN layer, partitioning, and weight-sparsity approach, since DNNs are which are inherently multi-dimensional MAC operations. The only limiting factor of mapping is the *number* of multiplier and adder switchlets of MAERI. For example, if the size of a neuron exceeds the number of multiplier switchlets of MAERI, the neuron needs to be temporally folded. This limitation is common in any hardware accelerator because of finite resources. MAERI's flexible ART structure enables it to support anywhere from VNs of size 1 to the entire array size. However, the number and size of the VNs affects the distribution and reduction bandwidth requirement. When the number of VN is large, the bandwidth requirement also increases. We present the effect of distribution and reduction bandwidth in Figures 2(m) and 3. Moreover, MAERI is tuned for enabling efficient spatial reduction. Given a DNN layer, finding the optimal dataflow [15], and optimal mapping strategy over MAERI is an open research question for future work.

We also note that one can potentially swap the MAERI multipliers and adders with other compute units/ALUs, enabling MAERI to map and run any variant of map-reduce-based applications.

IMPLEMENTATION

We implement MAERI architecture with Bluespec System Verilog (BSV) and synthesize the design using a Synopsys 28nm educational library. We compare the area and power of MAERI with Eyeriss [1] and a systolic array (similar to TPU [9]). We provide two flavors of comparison; compute-match and area-match in Figure 2. The compute match fixes the number of PEs (or multiplier switchlets) to 168, the number of PEs in Eyeriss, and the area-match restricts the area to that of Eyeriss, placing as many compute units as possible.

In compute-match, MAERI consumes 37% less area but 7% more power compared to Eyeriss. However, MAERI requires 32% more area and consumes 96% more power compared to systolic array in compute-match comparison. The area is the cost of reconfigurability that provides performance benefits. Please note that the power consumption is post-synthesis power that does not consider actual run time. Because MAERI reduces overall runtime, the actual energy consumption is comparable in CNNs and less in LSTMs, as we show later in Figure 2. In area-match, MAERI and systolic array houses 2.33X and 7.09X more compute units compared to Eyeriss with 98% and 137% more power, respectively. Although systolic array is area- and power-efficient compared to MAERI, it requires more run time and sometimes more energy with irregular dataflows because of its rigid interconnects. In the following section, we discuss such run time and energy aspects. We also study the area impact of the distribution and reduction bandwidth in Figure 2(f) and (g). With higher distribution bandwidth, additional simple switchlets are needed, adding area. Increased reduction bandwidth however only adds additional forwarding wires through the adders, no logic, leading to minimal area increase. In all cases, we observe the SRAMs in the prefetch buffer dominating area.

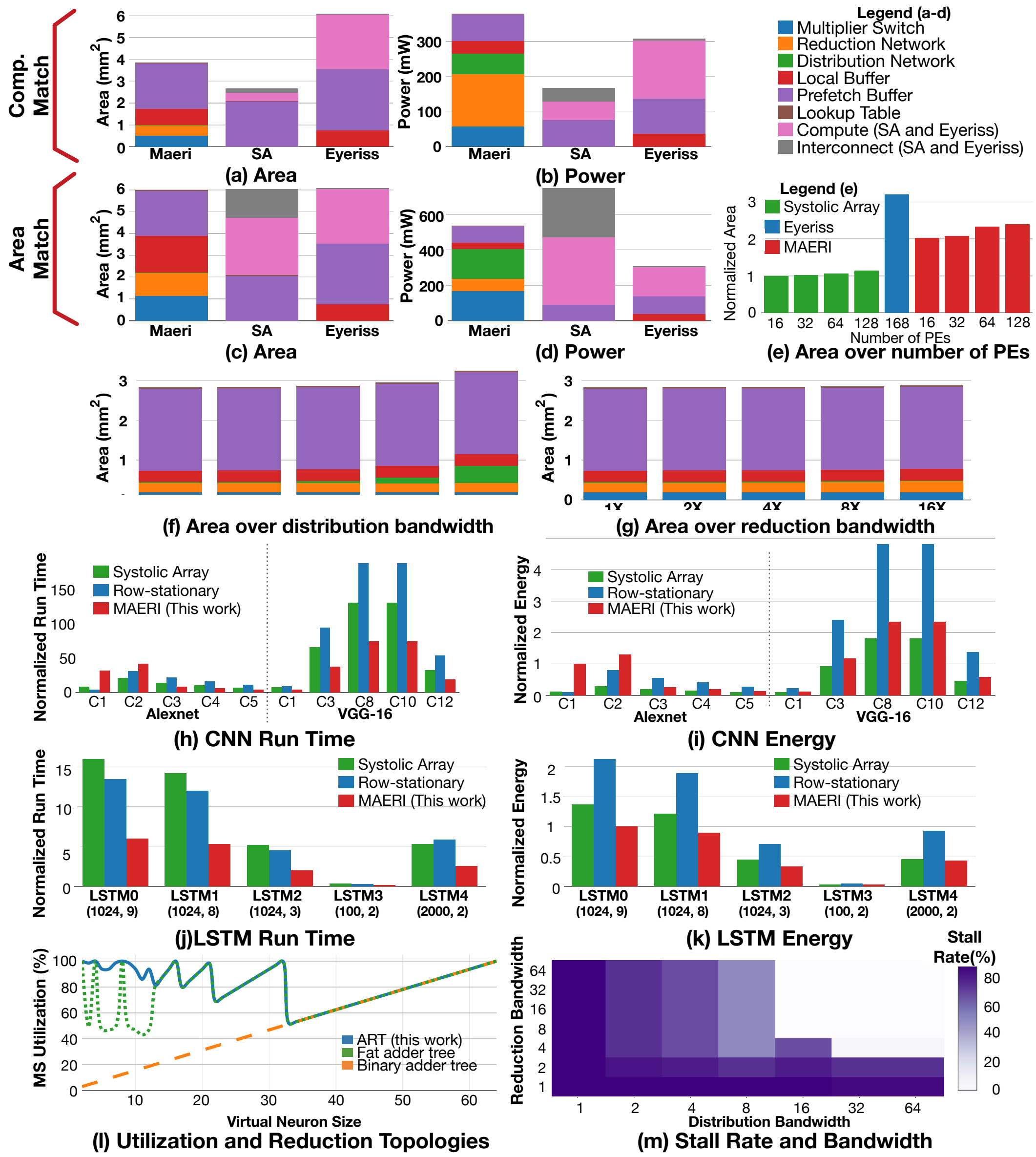

Figure 2: (a-g) Area requirement and post-layout power consumption of MAERI, systolic array (SA), and Eyeriss. (a) and (b) assume the same number of compute units (PE/multiplier switchlet). (c) and (d) assumes the same area as Eyeriss for all designs. (e) shows the area over number of PEs. (f) and (g) shows the area increase by extra distribution and reduction bandwidth, respectively, for MAERI with 64 multiplier switchlets. Total run time normalized to an ideal accelerator with 64 compute units (PEs/multiplier switchlets) that has infinite NoC bandwidth and one-cycle compute units for CNN (h) and LSTM (j). Energy consumption normalized to that of MAERI for Alexnet C1 (i) and LSTM0 (k). Numbers below each LSTM label indicates the LSTM dimension (input size and number of hidden layers). (l) The effect of reduction network topology on MS utilization in steady state with 64 multipliers and adders. (m) The effect of bandwidth on stall rate when processes non-edge inputs in VGG16- CONV1 with 64 multiplier switches. We present stall rate in a heatmap with distribution and reduction bandwidth on x and y axis, respectively. The dark area shows bandwidth pair with high stall rate. White space shows bandwidth combinations that achieve zero stalls.

EVALUATION

Runtime and Energy. To evaluate runtime, we run RTL simulation and measure total runtime in cycles. We compute power-delay product to estimate energy consumption. For workloads, we use Alexnet [12] and VGGnet-16 [11], and five RNNs (LSTM) from recent publications such as Google translator [11]. We use an 8x8 systolic array and a row-stationary accelerator with 64 PEs similar to Eyeriss [1] as our baselines. MAERI has 64 multiplier switchlets to match the number of compute units with the baselines. Figure 2(h-m) shows our results. In CNNs, MAERI reduced the run time up to 64% and by 42% in average. These benefits come from (i) higher multiplier utilization in MAERI compared to the baselines (depending on the filter dimensions), and (ii) fewer stalls (due to high bandwidth distribution and collection networks). The reduced runtime also decreased the energy consumption. MAERI requires 28% and 7.1% less energy in average compared to Eyeriss and Systolic array, respectively. In LSTMs, MAERI reduces 60% and 55% of run time and 21% and 52% less energy compared to systolic array and row-stationary accelerator. MAERI is significantly better in LSTMs because of high distribution bandwidth and its flexibility of mapping large size of neurons that appear in LSTMs.

Impact of Communication Topology and Bandwidth on Utilization and Stalls.

The *topology* of the reduction network of MAERI affects the both the mapping efficiency (i.e., whether or not the multiplier can map a computation) and its utilization (i.e., whether or not it can be used every cycle). This is because it determines how many spatial reductions can occur simultaneously. Figure 2(i) quantifies the utilization of the MSs in steady state (while processing non-edge input volume) with various VN sizes with three reduction network choices – binary adder tree, fat adder tree, and ART. We assume sufficient bandwidth from the distribution tree in this experiment. The binary adder tree can provide 100% utilization only when the VN size is close to the multiplier array size (64 in this case). The Fat Tree provides 100% utilization only when VN size is powers of two. MAERI provides close to 100% utilization for all VN sizes, except for cases when 64 mod VN size is not zero. This is not a limitation of the topology – but of the mapper since we do not have a compiler to support partial VNs yet. The figure also shows that mapping multipler smaller VNs is more efficient than larger VNs.

The *bandwidth* of the distribution and reduction networks of MAERI affects the stalls at the multipliers and adders. Insufficient distribution bandwidth results in it waiting for new weights/inputs, while insufficient reduction bandwidth serializes reductions across the VNs. This naturally depends on the DNN layer being mapped. Figure 2(m) shows the percentage of multipliers stalled when running VGG16-CONV1 over MAERI with various combination of distribution and reduction bandwidth. We observe that the bandwidth needs to be larger than 16X and 8X respectively to have zero stalls in the example.

Impact of Communication Bandwidth on Runtime and Energy. Figure 3(a), (b), (c), and (d) present the LSTM and CNN run time and energy of MAERI with varying bandwidth of the distribution network. A 16X design is a fat tree, a 1X is a binary tree, with other data points in between. The LSTM computation is highly sensitive to distribution bandwidth because of its lack of data reuse opportunity unlike CNNs. Thus, a wide distribution bandwidth significantly reduces runtime, as shown in Figure 3(a). However, in CNNs, distribution bandwidth does not provide run time improvement beyond certain point. This is because the maximum distribution bandwidth requirement in a steady state (when processing non-edges, which is the common case) is determined by the number of VNs mapped on MAERI. We can interpret these results as roof-line performance as suggested in a recent work [8]. The energy consumption presented in (b) and (d) implies that extra distribution bandwidth can reduce total energy consumption. However, beyond a certain point, energy consumption greatly increases because of roof-line performance limited by number of VNs and increased power consumption to support extra bandwidth. The 16X distribution bandwidth requires 16% more area compared to 1X, as presented in Figure 2(f). Therefore, based on the target application, distribution bandwidth should be carefully selected.

Reduction bandwidth also dramatically changes the overall run time, as Figure 3(e) and (g) present. However, unlike distribution bandwidth, the cost of extra reduction bandwidth is relatively minimal as Figure 2(f) and (g) show. Therefore, energy consumption results show similar pattern as run time, as Figure 3(f) and (h) present. Also, reduction bandwidth reaches its roof-line performance earlier than distribution bandwidth because the roof-line of reduction bandwidth is determined by number of virtual neurons with smaller order than distribution. Therefore, it is beneficial to employ roof-line reduction bandwidth for target neural network. From an area perspective, providing 16X reduction bandwidth has nearly the same area cost as the 1X reduction bandwidth since our collection structure is light-weighted so the area increase is minor compared to the entire accelerator area. For pure collection network area, the area cost increases by 20% for 16X reduction bandwidth, as presented in Figure 2(g). This sub-linear overhead is because the collection network is dominated by the adders – the fat links are essentially bypass paths that do not add much to the area. However, note that the actual benefits can be limited by the write bandwidth to the global buffer; larger reduction network bandwidth than that of global buffer does not improve the run time.

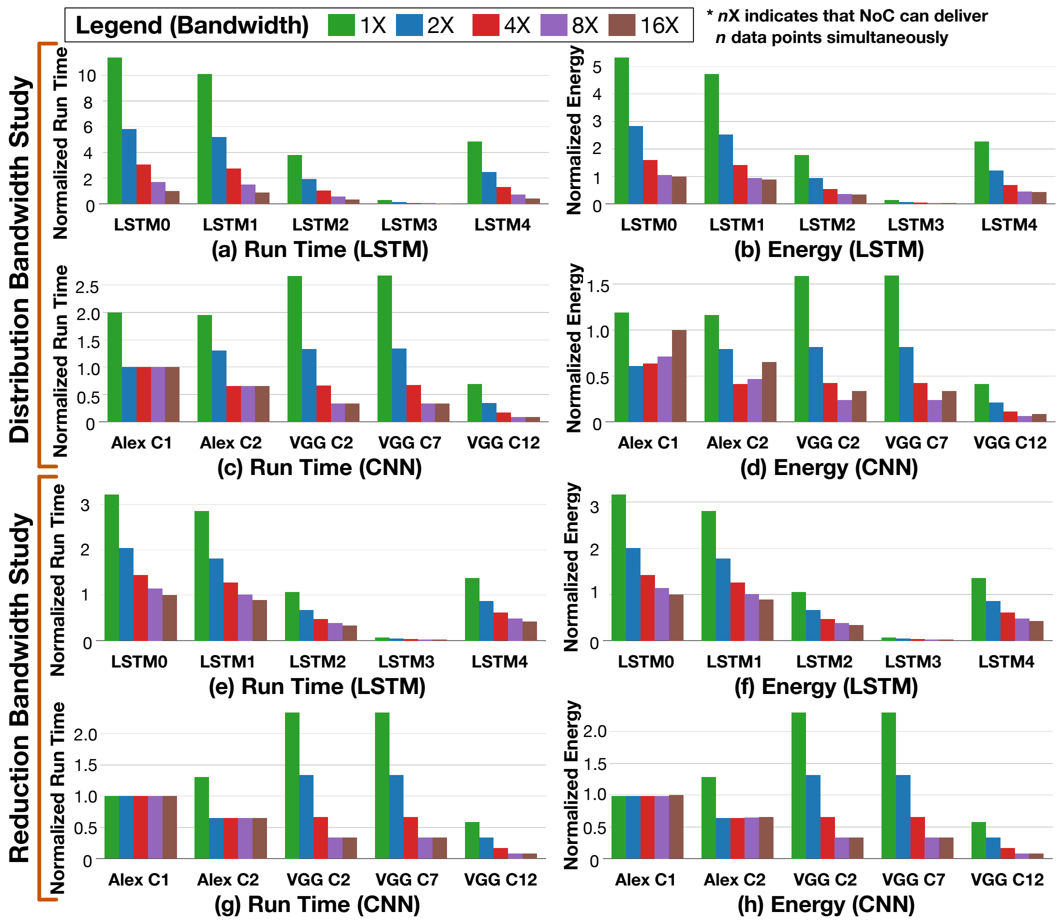

Figure 3: The impact of distribution/reduction bandwidth on LSTM **(a)/(e)** run time and **(b)/(g)** energy and CNN **(c)/(g)** run time and **(d)/(h)** energy. All the results are based on 64-multiplier switchlet MAERI and 64X bandwidth at the root to remove any impact of reduction bandwidth. LSTM and CNN results are normalized to the results of 16X bandwidth on LSTM0 and Alexnet C1 respectively.

CONCLUSION

Most DNN accelerators today interconnect PEs in a rigid tightly coupled manner which makes it impossible to map irregular dataflows (due to layer types, cross-layer fusion, sparsity, and so on), leading to under-utilization and/or stalls. We present a communication-driven approach called MAERI to address this challenge. MAERI carefully places tiny switches along fine-grained compute resources and connects them in topologies tuned for the traffic classes within DNN accelerators, namely distribution, local forwarding, reduction, and collection. MAERI translates each dataflow into a problem of mapping different sized neurons, which the MAERI networks support through reconfiguration of the switches. This approach provides extremely high compute-unit utilization, boosting performance and energy-efficiency.

ACKNOWLEDGEMENT

We thank Michael Pellauer from NVIDIA for feedback on this approach. We thank Bluespec Inc. for providing us with a free university license and tech support for Bluespec System Verilog.

REFERENCES

1. Yu-Hsin Chen, Joel Emer, and Vivienne Sze. "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks." in ACM/IEEE International Symposium on Computer Architecture (ISCA), 2016.
2. NVIDIA, "NVDLA Deep Learning Accelerator." NVIDIA, 2017, nvdla.org
3. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, "Going Deeper with Convolutions." in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.
4. Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. "Fused-layer CNN accelerators." in IEEE/ACM International Symposium on Microarchitecture (MICRO), Article No. 22, 2016.
5. Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. "EIE: efficient inference engine on compressed deep neural network." in ACM/IEEE International Symposium on Computer Architecture (ISCA), 2016.
6. Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. "Scnn: An accelerator for compressed-sparse convolutional neural networks." in ACM/IEEE International Symposium on Computer Architecture (ISCA), 2017.
7. Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li, “Flexflow: A fexible dataflow accelerator architecture for convolutional neural networks,” in IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017.
8. Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates et al. "In-datacenter performance analysis of a tensor processing unit." in ACM/IEEE International Symposium on Computer Architecture (ISCA), pp.1-12, 2017.
9. Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna, "Rethinking NoCs for Spatial DNN Accelerators.", in ACM International Symposium on Network-on-Chip (NOCS), 2017.
10. Karen Simonyan and Andrew Zisserman, "Very Deep Convolutional Networks for Large Scale Image Recognition.", in International Conference on Learning Representations (ICLR), 2015.
11. Yonghui We, et. al., "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.", Arxiv Preprint, 2016.
12. Alex Krizhevsky, Ilya Sutskever, and Geoffery E. Hinton, "Imagenet Classification with Deep Convolutional Neural Networks.", in Conference on Neural Information Processing Systems (NIPS), 2012, pp.1097-1105.
13. Fisher Yu and Vladlen Koltun, “Multi-scale Context Aggregation by Dilated Convolutions.”, in International Conference on Learning Representations (ICLR), 2016.
14. Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus, “Deconvolutional Networks.”, in IEEE Computer, 2010, pp. 2528-2535.

15. Hyoukjun Kwon, Michael Pellauer, and Tushar Krishna, “An Analytic Model for Cost-Benefit Analysis of Dataflows in DNN Accelerators”, https://arxiv.org/abs/1805.02566

ABOUT THE AUTHORS

Hyoukjun Kwon is a Ph.D. student at School of Computer Science, Georgia Institute of Technology. He received bachelor’s degree in computer science and engineering and environmental material science at Seoul National University in 2015. His research interest includes computer architecture, network-on-chip, and spatial accelerators for deep learning and graph applications.

Ananda Samajdar is pursuing his PhD from School of Electrical and Computer Engineering at Georgia Tech. He has completed his bachelors in Electronics and Communication Engineering from Indian Institute of Information Technology, Allahabad, in 2013. Prior to his PhD, Anand was a full-time VLSI design engineer at Qualcomm India. His research interests are computer architecture, VLSI design and machine learning.

Tushar Krishna (S’08–M’15) received the B.Tech. degree in electrical engineering from IIT Delhi, New Delhi, India, in 2007, the M.S.E. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2009, and the Ph.D. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2014. He was a Researcher with the VSSAD Group, Intel, Hudson, MA, USA, from 2013 to 2015, and a Post-Doctoral Researcher with the MIT SMART-LEES Center in 2015. He has been an Assistant Professor with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA, since 2015. His current research interests include computer architecture, interconnection networks, on-chip networks, deep-learning accelerators, and FPGAs.