Multi-dimensional Parallel Training of Winograd Layer on Memory-Centric Architecture

Byungchul Hong[†*], Yeonju Ro[*], and John Kim[‡]
[†]*FuriosaAI*
[*]*School of Computing, KAIST* [‡]*School of Electrical Engineering, KAIST*
{kaisthong, yeonju.ro, jjk12}@kaist.ac.kr

***Abstract*—Accelerating neural network training is critical in exploring design space of neural networks. Data parallelism is commonly used to accelerate training for Convolutional Neural Networks (CNN) where input batch is distributed across the multiple workers; however, the communication time of weight gradients can limit scalability for moderate batch size. In this work, we propose *multi-dimensional* parallel training (MPT) of convolution layers by exploiting both data parallelism and intra-tile parallelism available in Winograd transformed convolution. Workers are organized across two dimensions – one dimension exploiting intra-tile parallelism while the other dimension exploits data parallelism. MPT reduces the amount of communication necessary for weight gradients since weight gradients are only communicated within the data parallelism dimension. However, Winograd transform fundamentally requires more data accesses and the proposed MPT architecture also introduces a new type of communication which we refer to as *tile transfer* – gather/scatter of Winograd domain feature maps (tiles). We propose a scalable near-data processing (NDP) architecture to minimize the cost of data accesses through 3D stacked memory while leveraging a memory-centric network organization to provide high-connectivity between the workers to accelerate tile transfer. To minimize tile gathering communication overhead, we exploit prediction of activation of spatial domain neurons in order to remove the communication of tiles that are transformed to non-activated neurons. We also propose *dynamic clustering* of the memory-centric network architecture that reconfigures the interconnect topology between the workers for each convolution layer to balance the communication required for weight gradients and tile transfer. Our evaluations show that the proposed MPT with NDP architecture accelerates training by up to 2.7×, 21× compared to data parallel training on the NDP architecture and a multi-GPU system, respectively.**

***Index Terms*—Convolutional Neural Network, Winograd Transform, Near-data Processing, Memory-centric Network**

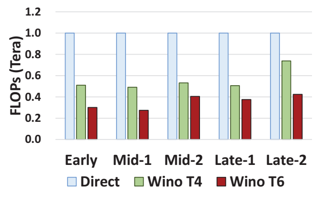
I. Introduction

Rapid training of neural network accelerates design space exploration of neural networks and enables research of larger-scale neural networks. In this work, we focus on the one of the more popular neural networks, Convolutional Neural Network (CNN) [1], [2] that is widely used for image recognition and classification. Data parallel [3]–[6] training is widely used for convolution layers in CNN, where input batch is distributed across multiple workers. While model parallelism [3], [7], [8] can be exploited in some neural networks, data parallelism is commonly used in convolution layers and in this work, we focus on data parallelism. In data parallel training, every worker generates partial sum of weight gradients (having the same size as weights) for its own subset of the input batch. The weight gradients are reduced or accumulated across all workers and updated weights are broadcasted to all workers using Stochastic gradient descent (SGD). With the fixed amount of data generated per worker, an optimal collective communication algorithm on ring topology (pipelined reduce/broadcast [9]) results in a constant communication time regardless of the number of workers [10]. However, even the optimal (pipelined) implementation for the communication can limit scalability with a fixed total batch size; as the number of workers increases, the *per worker batch size* or the amount of computation assigned to each worker decreases and the communication represents a bigger fraction of total execution time. One common approach to scale data parallelism is increasing the input batch size [11]–[13]. However, prior work has shown that large batch size can degrade the quality of trained models [14], [15] and slow down convergence [16], [17]. In this work, we assume synchronous SGD with moderate batch size (i.e., 128–256) used in CNNs [2], [18]–[21] but propose an alternative approach to enable scalable training.

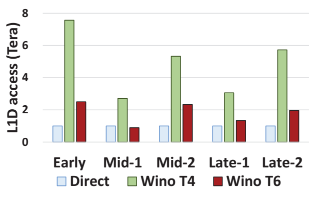
In particular, we exploit a new type of parallelism, *intra-tile parallelism* available in transformed convolutions, such as Winograd transform [22] and FFT [23], [24], to enable better scalability as the number of workers increase. Transformed convolution has been proposed to reduce the amount of computation by changing convolution operation in spatial domain to simpler dot products in the transformed domain. With Winograd transform, [1] feature maps and weights are transformed to Winograd domain tiles, with each tile size of $T \times T$. Element-wise dot products in Winograd domain consist of T^2 *independent* matrix multiplications, compared to direct convolution that consists of a single large matrix multiplication.

By exploiting the element-wise (or intra-tile) independence, we propose *multi-dimensional* parallel training (MPT) that combines data parallelism and intra-tile parallelism for Winograd transformed convolution. We propose two-dimensional organization of workers to properly exploit MPT, where workers in the same column constitutes a *group* for data parallelism while the workers in the same row constitutes a *cluster* for intra-tile parallelism. Input batch is distributed across the clusters (or within a group) while elements of Winograd domain tiles are distributed across the groups. Since there is no need to communicate weights between different groups (i.e., different parts of a tile), communication for weights reduction/broadcast is restricted to within each group. Thus,

[1]FFT has the same characteristic, but in this work, we use Winograd transform which is appropriate for small size weights, such as 3×3, and is similar to the weight size of CNNs that we use.



(a) Computation



(b) Data access count

Fig. 1. Comparison of (a) computation (FLOPs) and (b) memory accesses (L1 data cache access count) for direct convolution and Winograd transformed convolution with tile size of 4×4 (Wino T4) and tile size of 6×6 (Wino T6).

the size of weight gradients generated per worker is reduced by the number of groups and improves scalability.

While MPT reduces the communication of weights between workers, it introduces additional challenges. MPT involves a new type of communication within a cluster to gather/scatter Winograd domain feature maps, which we refer to as *tile transfer*. To efficiently support tile transfer, we propose a hybrid topology that exploits low-diameter (high-radix) topology for the tile communication across the groups while a ring topology is used for weight communication across the clusters. However, the optimal organization (i.e., number of workers for each dimension) can vary based on the communication requirements of the convolution layer. To enable flexibility in the architecture, we also propose dynamic clustering (or a reconfigurable topology) of the workers to reconfigure the interconnection network based on a pre-estimated communication amount across the two dimensions. To reduce the amount of communication for tile gathering, we also propose a method that predicts activation of neurons from the quantized values of Winograd domain data elements without loss of accuracy.

While the amount of computation decreases in the Winograd domain, the tiles and weights in the Winograd domain are larger than the feature maps and weights in the spatial domain and results in more data accesses. A comparison of computation and memory access is shown in Figure 1 for direct convolution and two Winograd-transformed convolutions of five different layers. [2] The results show that Winograd-transformed convolution can reduce computation by 2.8× on average, but also increases the amount of data access by 4.4× on average. [3] As a result, we propose a scalable near-data processing (NDP) architecture that exploits the high-bandwidth of 3D stacked memory for increased data access, and creates a memory-centric [27], [28] architecture. In particular, the contributions of this work include the following.

- We propose a novel *multi-dimensional* parallel training (MPT) of Winograd domain convolution that exploits both data and intra-tile (or element-wise) parallelism available in Winograd domain.
- To efficiently support communication among workers with MPT, we propose a 2D organization of workers with hybrid topology that consists of a ring topology for collective communication in data parallelism while a high connectivity topology is used for tile transfer.
- We propose dynamic clustering that enables optimal configuration of workers to balance the two different types of communication (weight gradients and tile transfer) in the MPT architecture.
- We propose prediction of activation in the spatial domain neurons from the quantized values of Winograd domain data, without accuracy loss, to reduce bandwidth needed for tile gathering.

[2] Intel Xeon E5540 CPU with Neon deep learning framework [25], OpenBLAS library, and vTune [26] was used for the measurements with a batch size of 256. The layers used are described later in Table II.

[3] Data reuse with on-chip buffer (or cache) can reduce the difference in memory accesses between direct and Winograd-transformed convolution; however, Winograd convolution still requires more data accesses.

II. Background

In this work, we use the following symbols to describe the convolution algorithms.

x, y, w	: Spatial domain input, output feature map, weight
X, Y, W	: Winograd domain input, output feature map, weight
I, J, B	: Number of input, output channels, batch size

A. Convolution Layer

Convolution layer training is composed of three phases : forward propagation, backward propagation, and weight update. During forward propagation (*fprop*), input feature map x_i is convolved with weight $w_{i,j}$ and passes non-linear activation function f, such as rectified linear unit (ReLU), to compute output feature map y_j.

$$y_{b,j} = f(\sum_i x_{b,i} * w_{i,j})$$

During backward propagation (*bprop*), gradient of loss function L with respect to inputs ($\frac{\partial L}{\partial x}$) are computed by convolving output gradients ($\frac{\partial L}{\partial y}$) and flipped weights ($w^f$), and then passes derivative of the activation function (f').

$$\frac{\partial L}{\partial x_{b,i}} = f'(\sum_j \frac{\partial L}{\partial y_{b,j}} * w^f_{i,j})$$

During weight update phase (*updateGrad*), weight gradients ($\frac{\partial L}{\partial w}$) are computed by convolving output gradients and input feature map.

$$\frac{\partial L}{\partial w_{i,j}} = \sum_b \frac{\partial L}{\partial y_{b,j}} * x_{b,i}$$

Weights are updated by adding (or computing a mean value of) weight gradients multiplied by the learning rate.

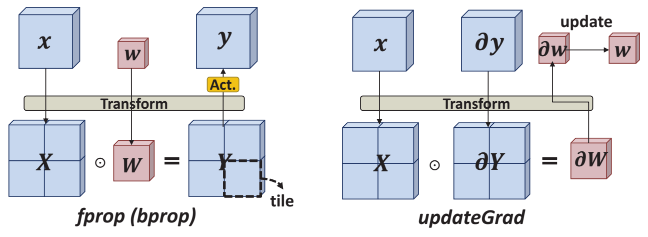
B. Winograd Layer

Winograd algorithm [22] for 2D convolution was proposed to reduce the number of multiplications by changing convolution operations to element-wise dot product operations ($\odot$). A 2D Winograd transform can be represented as $F(m \times m, r \times r)$ and described with the following equation,

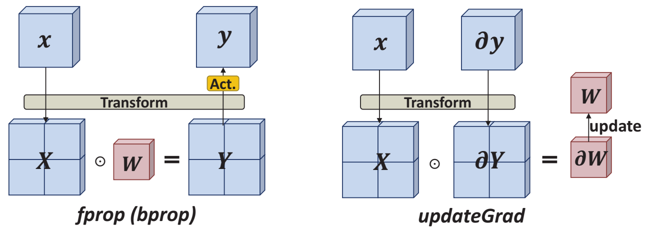
$$y = A^T[(GwG^T)] \odot (B^T xB)]A \tag{1}$$

where w is $r \times r$ size weight and y is $m \times m$ size output, and G, B, A are co-efficient matrices. [4] Theoretically, $F(4\times4, 3\times3)$

[4] We assumed square weights and outputs, but the Winograd transform can be applied to non-square weights and outputs.



(a) Convolution layer with Winograd Transform



(b) Winograd layer

Fig. 2. (a) Three phases of convolution layer with Winograd transform, and (b) Winograd layer [29] that directly updates weights in Winograd domain.

reduces multiplication by $4\times$ compared to direct convolution [22], by transforming the convolution operation to dot products.

To apply the Winograd algorithm to convolution layer, prior work [22] proposed tile-based approach. Input feature map is transformed to multiple tiles, with each tile size of $T \times T$, where $T = m + r - 1$, and weights are also transformed to a tile with the same size of $T \times T$. Dot product between multiple input tiles and weight tile generates output tiles, which are inversely transformed to an output feature map. This overall computation for *fprop* can be represented with the following matrix equation,

$$Y_{b,j}^{(u,v)} = \sum_i X_{b,i}^{(u,v)} W_{i,j}^{(u,v)} \tag{2}$$

where (u,v) represents the element in the u-th row and v-th column in a tile.

Figure 2(a) shows *fprop, updateGrad* of convolution layer with Winograd transform. *bprop* has similar computation and data sets with *fprop* except inputs and outputs replaced to $\partial y (= \frac{\partial L}{\partial y})$ and $\partial x (= \frac{\partial L}{\partial x})$ respectively. Winograd layer [29] was recently proposed where weights are updated directly in Winograd domain (Figure 2(b)). It was shown that Winograd layer does not decrease the quality of trained network for the small weight/tile size used in this work, but can actually *increase* the accuracy since Winograd domain weights have larger size than spatial domain weights [29]. Current cuDNN from Nvidia [30] supports Winograd transformed convolution but are limited to weight size of 3x3 and 5x5 where Winograd transformed convolution does not impact accuracy. However, as weight/tile size grow, numerical instability can grow and impact accuracy. Recent work [31] has explored how to achieve better stability and improve accuracy through better Winograd transform matrix. If accuracy can be improved through such approach, the proposed multi-dimensional parallel architecture from this work can be extended to larger weight sizes.

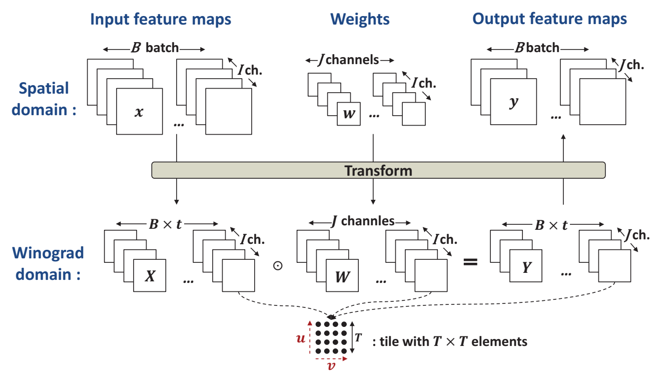
C. Related Work

Accelerators: Different CNN accelerator architectures [32]–[36] have been proposed that maximize data reuse and power efficiency for inference in a single chip. While most accelerators that have been proposed focused on inference, recently a heterogeneous server architecture [37] for neural network training has been proposed for scalable and efficient computation. Scalable parallel training by using multiple GPUs have been widely studied including accelerating collective communication with ring topology [38], [39], or tree topology [5] for data parallelism to efficiently utilize the available communication bandwidth. In comparison, this work proposes to reduce the amount of communication as the number of workers increases by exploiting intra-tile parallelism in transformed convolution and creating independent collective operation groups.

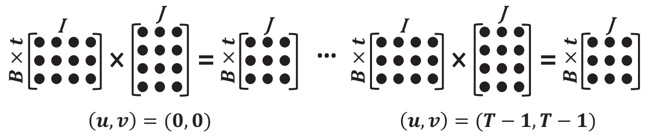
Scalable Parallel Training: A different approach to scale parallel training is increasing the batch size, but neural network models trained with large batch size tend to generalize less well (known as "generalization gap") [14], [40]. Recently, deep learning community has actively explored increasing batch size [11]–[13] while maintaining the quality of the trained model by scaling learning rate linearly with the batch size [11]. However, large batch size can reduce the range of learning rates that provides stable convergence with acceptable test accuracy, since smaller batch size provides more up-to-date weight values [15]. Training with large batch size can also take longer to provide the same quality of the trained network [17]. With a limited scaling of batch size, the ratio of communication to computation increases as the number of workers increases (or compute capability of each worker increases through advances in architecture [5]). To provide high scalability at reasonable batch size, this work reduces the communication time through the proposed multi-dimensional parallelism and hybrid interconnect architecture.

3D integration: Recent advances in 3D integration technology allow 3D stacked dies with TSVs (Through-Silicon Vias). Hybrid Memory Cube (HMC) [42] is an example of 3D-stacked memory with a logic die on the bottom that also includes processing elements [43]–[45] for near-data processing (NDP). HMC modules, each becoming a router, can be interconnected with high-speed links to scale the system and create a memory-centric network [27], [28]. Recently, near-data processing architectures for CNN have been proposed to utilize the high-bandwidth and energy-efficiency of 3D-stacked memory [35], [36]. However, prior work focused on inference with a single memory module while we focus on training across multiple memory modules. Some recent studies [46], [47] have evaluate multiple memory modules but the number of memory modules is limited to a few (up to four), and

[5] Recent Volta GPU has 12x higher compute capability for deep learning training [41] compared to the previous generation GPU.

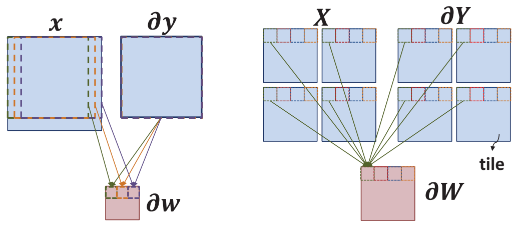


(a) Winograd transform and dot products of tiles in Winograd domain



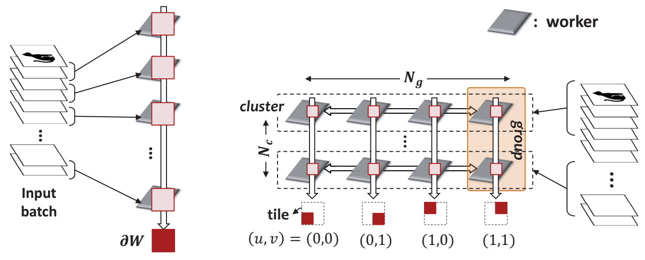
(b) Dot products are element-wise matrix multiplications

Fig. 3. Shows (a) 4D tensors in Spatial domain transformed to dot products of tiles in Winograd domain, and (b) the dot products represented as element-wise matrix multiplications.



(a) Direct convolution (b) Transformed conv. (dot product)

Fig. 4. Intra-tile parallelism available in transformed convolution.



(a) Data parallel (b) MPT with 4 groups

Fig. 5. (a) Data parallel training and (b) our proposed multi-dimensional parallel training (MPT). N_c (N_g) is the number of clusters (groups).

communication overhead between the memory modules has not been well evaluated. In comparison, we focus on scalable training with large number of workers (memory modules), and the impact of communication on scalability. This work assumes high bandwidth serial link interconnect between the workers. NVlink [48] is one example of such interconnect and recently announced interconnect standard from the industry including CCIX [49], OpenCAPI [50], and GenZ [51] are examples of interconnect whose objective includes interconnecting accelerators and/or memory modules to provide high bandwidth.

III. Scalable Parallel Training

In this section, we describe our proposed multi-dimensional parallel training (MPT) that combines data parallelism and intra-tile parallelism to increase scalability but also describe the challenges of scalability for MPT.

A. Parallelism in Transformed Convolution

Dot products of transformed convolution can be represented as shown in Figure 3(a). Each input feature map is transformed into t tiles, with each tile size of $T \times T$, resulting in a total of $B \times t$ tiles per input channel. By breaking up each tile (with a size of $T \times T$) into T^2 elements, the dot products of the tiles can be represented as T^2 independent element-wise matrix multiplications, as shown earlier in Equation 2 and Figure 3(b). [6]

In Winograd domain dot products, there is no computation between the elements placed at different positions in a tile (i.e., different (u, v)). This independence or parallelism available in the transformed convolution can be exploited for parallelization with minimal communication, which we refer to as *intra-tile* parallelism. Figure 4 provides a different view of intra-tile parallelism compared to direct convolution. With direct convolution, most of the input data elements (or neurons) are shared to compute different output elements, and element-wise partitioned execution requires duplication of the large input data. In comparison, transformation changes the convolution to element-wise dot products and enables element-wise separation of input data, leading to efficient element-wise (or intra-tile) parallel execution. Figure 4 shows the *updateGrad* phase as an example, but the different characteristics of direct convolution and transformed convolution apply to the *fprop* and *bprop* phases.

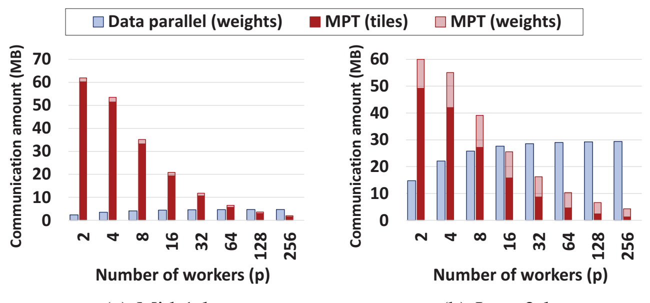
B. Multi-dimensional Parallel Training (MPT)

For the parallel training of CNNs, data parallel training is widely used for convolution layers [3]–[6]. While data parallel training does not require communication between workers during the *fprop* and *bprop* phases, the *updateGrad* phase requires a reduction of weight gradients ($\partial L/\partial w$, or ∂w) generated by all workers as well as broadcasting the updated weights. Thus, the amount of communication for each worker to transfer in the *updateGrad* phase is almost fixed and proportional to the weigh size $|w|$ regardless of the number of workers (p). [7]

In this work, we propose multi-dimensional parallel training (MPT), which combines data parallelism and intra-tile parallelism. With MPT, workers are placed in two-dimensions with the workers in the same row constituting a *cluster* while the workers in the same column constitute a *group*, as shown in Figure 5. Data parallelism is applied across the clusters,

[6] (u, v) represents the coordinates of the elements inside a tile.

[7] Note that the weight gradients have the same size as weights, $|\partial w| = |w|$.



(a) Mid-1 layer (b) Late-2 layer

Fig. 6. The amount of communication for each worker to transfer at each iteration of the two different layers (from Table II) with different parallelism strategies.

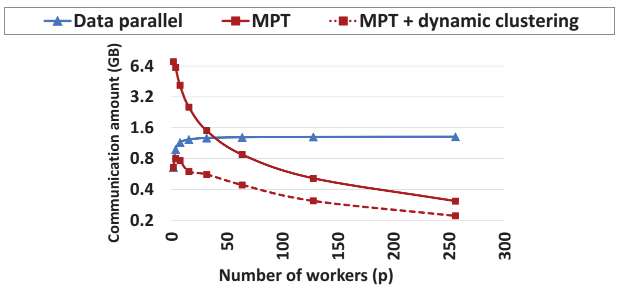


Fig. 7. The amount of communication for each worker to transfer at each iteration of FractalNet with different parallelism strategies. *y*-axis is log-scale.

as the input batch is distributed across the N_c clusters, and intra-tile parallelism is applied across groups, which distributes elements of the Winograd domain tiles to N_g groups. Thus, $N_c \times N_g = p$ and if $N_g = 1$, only data parallelism is exploited across the workers. As shown in Figure 3(b) and Figure 4(b), there is no computation between different tile elements, or different groups for the dot products – thus each part of the Winograd domain weights ($W_{u,v}$) is only used within the associated group. Tile transfer occurs across the groups (or within each cluster) only for transformation between spatial domain neurons and Winograd domain tiles in the *fprop* and *bprop* phases, which are described in the following Section III-C. In the *updateGrad* phase, each worker generates weight gradients for part of the weights assigned to the worker's group, and communication of the weight gradients occurs within each group. As a result, the size of the weight gradients that needs to be reduced and broadcast per worker is reduced by N_g.

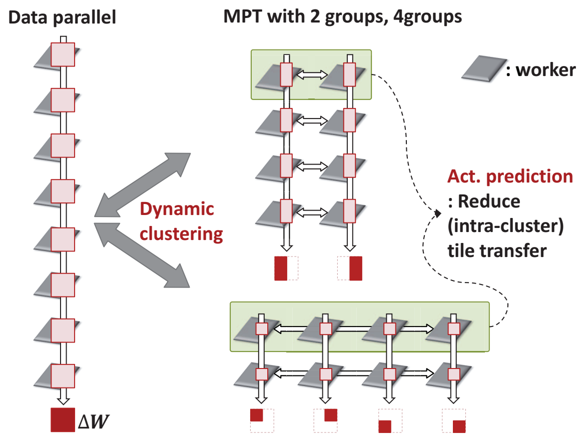
C. Challenges of MPT

While MPT reduces communication for weight updates, it introduces a new type of communication. In the *fprop* and *bprop* phases, input tiles are element-wise partitioned and distributed to workers within a cluster for intra-tile parallelism, which we refer to as *tile scattering*. Each worker loads the weight element(s) and computes the output element(s) for a particular part of the tile assigned to the worker's group. The different parts of each output tile are combined to build a complete tile for the inverse transform to a spatial domain feature map (or neurons), resulting in communication of tiles within a cluster, which we refer to as *tile gathering*. The spatial domain neurons subsequently pass the non-linear activation and pooling (if exists) layers and become the input to the next layer. Figure 6 compares the amount of communication for each worker to transfer at each iteration of two different layers from Table II. We assumed a batch size of 256 and $N_g = N_c = \sqrt{p}$. For the late layers with smaller feature maps and larger weight sizes (e.g., Late-2), MPT significantly reduces weight communication and tile transfer comprises a small portion of the total communication amount with a large number of workers (p), as shown in Figure 6(b). For layers with relatively larger feature map sizes (e.g., Mid-1), MPT still significantly reduces weight communication, but tile transfer results in a large amount of communication reducing the benefits of MPT (Figure 6(a)).

To understand the scalability of MPT from the added communication, we first analyze the amount of communication from data parallelism and MPT. For data parallel training with a ring topology, each worker sends weight gradients with a size of $|W|$ to the next worker, and the hop count is $p-1$. Thus, the communication amount per worker is $|W|\frac{p-1}{p}$ and remains almost constant with large p. For MPT, the weights are partitioned across N_g groups and each worker transfers only $\frac{|W|}{N_g}$ weight gradients, with $N_c - 1$ hop traversals per group. Thus, the communication amount per worker for weight gradients becomes $\frac{|W|}{N_g} \cdot \frac{N_c-1}{N_c}$. For tile transfer, the input batch (size of B) is partitioned across N_c clusters, and tiles are partitioned across the N_g groups inside each cluster. Thus, each worker holds $\frac{|Tiles|}{N_g N_c}$ tile data, and transfers $\frac{N_g-1}{N_g}$ portion of the tile data to other workers in the same cluster. The communication amount per worker for tile transfer becomes $\frac{|Tiles|}{N_g N_c} \cdot \frac{N_g-1}{N_g} = \frac{|Tiles|}{p} \cdot \frac{N_g-1}{N_g}$.

In Figure 7, the amount of data communicated per worker for each iteration of FractalNet [52] training across all layers is shown. We assume $N_g = N_c = \sqrt{p}$ with a fixed batch size of 256. When the number of workers is small, the communication overhead is actually higher with MPT compared to data parallel training, because of the tile transfer overhead. With data parallel training, the communication amount per worker remains constant with larger p and results in poor scalability, since the amount of computation assigned to each worker decreases as the number of workers increases with a fixed total batch size. For MPT, the communication amount per worker becomes proportional to $\frac{1}{\sqrt{p}}$ for weight gradients and $\frac{B}{p}$ for tile transfer – thus, the communication amount per worker decreases in MPT as the number of workers (p) increases.

The amount of communication required for weight and tile transfer differs significantly based on the structure of the convolution layer, and the size of the two parallelism dimensions (i.e., N_g, N_c) determines the communication amount of weight and tile transfer. In addition, there is a trade-off between communication amount for weights and tiles. Having more groups reduces weight communication but increases tile transfer, while having fewer groups increases weight communication but reduces tile transfer. For example, with



Data parallel
MPT with 2 groups, 4groups
: worker
Dynamic clustering
Act. prediction
: Reduce (intra-cluster) tile transfer
ΔW

Fig. 8. Different parallelization configurations with dynamic clustering.

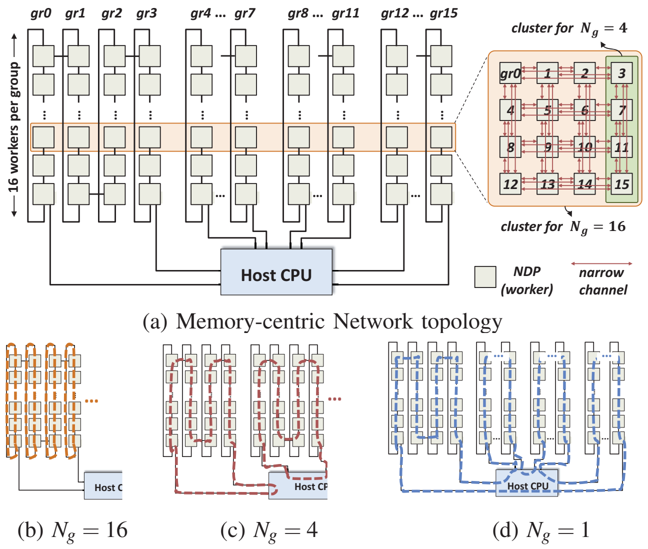


Fig. 9. (a) High-level block diagram of scalable NDP architecture with $p = 256, N_g = 16, N_c = 16$. A 2D flattened-butterfly topology is used within a cluster to interconnect the 16 workers. Three different configurations from dynamic clustering are shown in (b,c,d).

a single group, only data parallelism remains and tile transfer is completely removed. Thus, the optimal configuration of MPT that minimizes total communication amount changes for each convolution layer. [8] We propose dynamic clustering to enable the reconfiguration of MPT per layer to minimize overall communication. Figure 7 also shows results for MPT with dynamic clustering and the amount of communication required is reduced. [9] For small number of workers, the benefit of dynamic clustering is significant since fixed interconnect results in inefficient bandwidth usage across the different layers. However, the benefit of dynamic clustering still results in reduction of communication by 1.4× for $p = 256$. In the following section, the details of dynamic clustering with a hybrid topology is presented, while the activation prediction scheme is presented in Section V which further reduces the amount of communication of tile gathering by predicting the activation of neurons and bypassing the collection of tiles transformed into non-activated neurons. Figure 8 shows the application of dynamic clustering and activation prediction to MPT.

IV. Dynamic Clustering

In this section, we discuss our proposed dynamic clustering that enables the number of groups and clusters (N_g and N_c) to be reconfigured to match the characteristics of the neural network layer. High-level architecture is shown in Figure 9(a) and consists of workers or "memory modules" that are interconnected through a memory-centric network [27], [28]. Figure 9 shows an architecture with 256 workers organized across two dimensions (groups and clusters) to match the multi-dimensional parallel training (MPT) that we propose in this work. The workers are interconnected through a memory-centric network with a ring interconnect within each group to support collective operations of weights [9], [53]. [10] Compared to data-parallel training, MPT results in multiple, shorter rings – i.e., for p workers, the length of the ring decreases from p for data parallel to N_c for MPT but result in N_g rings, with each ring performing collective operation for only $|W|/N_g$ size data. A high-radix network, 2D flattened butterfly (FBFLY) [54], is used for 16 workers inside a cluster to efficiently support *all-to-all* traffic for tile gathering/scattering.

As discussed in the previous Section, the number of groups (N_g) and the number of clusters (N_c) create trade-off in the amount of communication for weight gradients and tile gathering/scattering. Smaller N_g (and larger N_c) is preferred for early layers which have larger feature map size to minimize the amount of tile transfer while larger N_g (and smaller N_c) is preferred for late layers that have larger weight size to minimize the amount of weight gradients. To enable flexible organization of the workers for each layer, we propose *dynamic clustering* – reconfiguration of the interconnect to modify the number of groups and clusters and minimize the communication time while maximizing scalability.

To dynamically change the topology (or modify the N_g and N_c), we exploit the connectivity through the host that interconnects the multiple groups together. As N_c increases (or N_g decreases), the connectivity through host is leveraged to increase the size of a group. Since neural networks have fixed layer structures and the required communication amount and the link bandwidth can be calculated in advance, the optimal configuration per layer to minimize the communication time is pre-determined and does not change. Reconfiguration between

[8] Early layers having large feature map sizes can be configured with small number of groups to reduce tile transfer, and late layers having large weights can be configured with large number of groups to reduce weight communication.

[9] We assume an optimal reorganization of the communication for each layer.

[10] Ring is a bandwidth optimal algorithm for collective operation but can have an overhead of more frequent start-up [9]. For collective operations on a high-speed links that we leverage, start-up time overhead is negligible and ring algorithm enables uniform utilization of all links.

two layers modifies the route (or destination) of tile transfer and weight communication, and does not incur any additional data transfer or overhead.

In our evaluation, we support the following three different configurations of workers:

- 16 N_g, 16 N_c : No routing through host
- 4 N_g, 64 N_c : gr0⇔gr3, gr4⇔gr7, gr8⇔gr11, gr12⇔gr15
- 1 N_g, 256 N_c : gr0⇔gr15, gr3⇔gr4, gr7⇔gr8, gr11⇔gr12

where ⇔ represents the additional connectivity made through the host between the two groups. The first configuration (16 N_g, Figure 9(b)) can be used to minimize weight communication. For a $F(2\times2,3\times3)$ transform with a tile size of 4×4, a single element in a tile is assigned to each group. To interconnect the 16 workers inside each cluster, 2D flattened butterfly topology is used and tile data can be transferred with a maximum of 2 hop count. The second configuration (4 N_g, Figure 9(c)) has more weight communication compared to the configuration with 16 N_g, but less tile transfer. For a $F(2\times2,3\times3)$ transform with a tile size of 4×4, four elements (a single line of 2D tile) are assigned to each group, and the tile transfer can be further reduced by executing 1D Winograd transform before transferring tile data. More details are described in the following Section V. Four fully connected workers (e.g., workers 3,7,11,15 in Figure 9(a)) constitute a cluster, and tile data can be transferred in a single hop. The third configuration (1 N_g, Figure 9(d)) returns to data parallelism with no communication for tile transfer, but has the largest amount of weight communication. Based on the characteristics of each layer, the more optimal configuration is selected to minimize communication time and maximize performance.

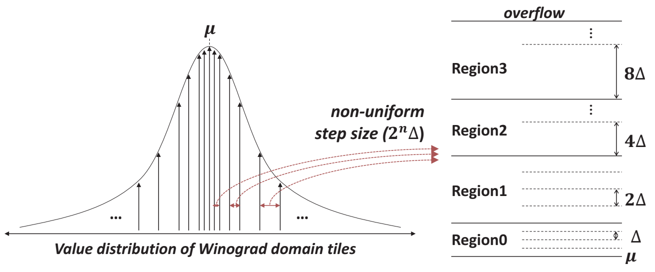
V. Activation Prediction

With the proposed multi-dimensional parallel training (MPT), *tile transfer* – gathering/scattering of Winograd domain feature map (tiles) is necessary. In this section, we discuss how to minimize communication required for the tile transfer. To reduce the amount of communication for tile scattering, we use skipping of zero values [33], [34], and for tile gathering we propose an activation prediction method.

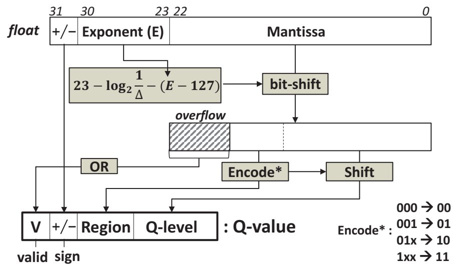
A. Activation Prediction without Accuracy Loss

To reduce communication amount of the tile gathering, we propose activation prediction method without accuracy loss. In this work, we assume Rectified linear unit (ReLU) [2], [55] as the non-linear activation function that only sends neurons with positive values to the next layer. If the neurons transformed from a tile are all non-activated, inverse transform of the tile can be omitted. To skip unnecessary tile gathering, we propose to have source workers send *quantized* values of tile elements for prediction before sending real values to the destination worker. [11] A key challenge is to predict the activation of

[11] To gather all elements of a tile, the *destination* worker is responsible for receiving data from *source* workers or all of the workers within the same cluster. The actual destination worker is determined based on input batch parameter.


μ
overflow
Region3
8Δ
non-uniform step size ($2^n\Delta$)
Region2
4Δ
Region1
2Δ
Region0
Δ
μ
Value distribution of Winograd domain tiles

(a) Non-uniform quantization of Winograd domain elements



(b) Quantization logic for single-precision float

Fig. 10. Quantization of Winograd domain tile values to predict activation

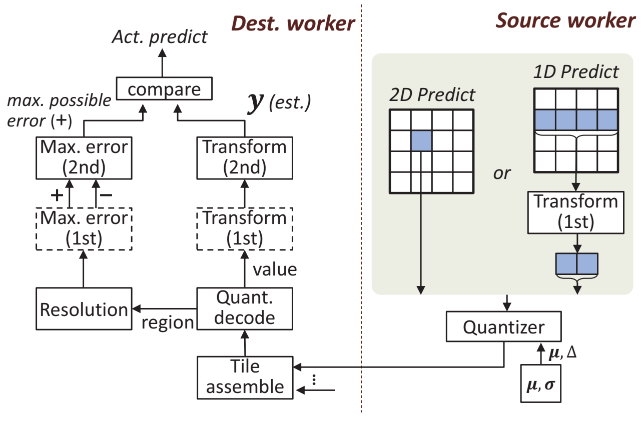


Fig. 11. Activation prediction flow with quantization.

spatial domain neurons from the quantized values of Winograd domain data (tiles), since a transform is involved. In our proposed method, destination worker calculates both estimated values of neurons and *maximum possible error of quantization* together to make a *conservative* prediction. Quantization error is accumulated during the transform process since multiple quantized values are multiplied with co-efficients and added.

From our experiments with ImageNet data sets [56] and trained CNNs, we observed that the values of Winograd domain tiles follow normal distribution. To follow the normal distribution of the real values appropriately, we use non-uniform quantization [57], [58] as shown in Figure 10(a). The range of real values is divided to multiple regions and each region has the same number of steps. (Real values outside of the range are indicated as overflow.) Step size (Δ) doubles when the region of the real value changes, and step size is decided by the standard deviation (σ) of real values. To implement the non-uniform quantization for a single precision float number, a simple hardware implementation is shown in Figure 10(b) using only simple integer arithmetic logic, bit-shifter, and log value of $1/\Delta$ can be precomputed and stored.

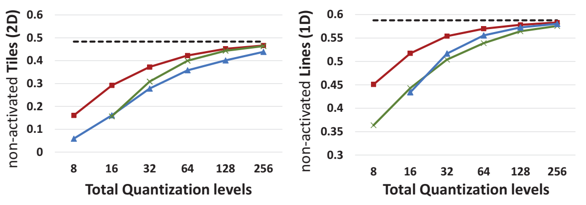
Figure 11 shows overall flow for activation prediction and can be classified as either 1D predict or 2D predict, depending on the amount of data assigned to each group. When the number of groups is small, each worker can contain a complete line (a row or a column) of a tile, and the first 1D transform can be computed at the source worker with real values. After the first 1D transform in the source worker, quantized values of the 1D transform output are sent to the destination worker for prediction, which we refer to as *1D predict*. In comparison, with a large number of groups, each worker does not contain the entire line of a tile, and source worker sends quantized values of output elements (a part of a tile) to destination worker for activation prediction, which we refer to as *2D predict*.

At the destination worker, 2D transform is applied [12] to both the quantized values and quantization resolution to compute the estimated values of neurons and maximum (positive) possible error of quantization respectively. Resolution represents quantization step size (1/2/4/8Δ in Figure 10(a)), or the *maximum gap* between the real value and quantized value. During the first 1D transform, both positive (+) and negative (−) maximum possible errors are calculated. Note that Winograd transform is basically a matrix multiplication with a co-efficient matrix, as shown in Equation 1. The positive (negative) maximum possible error of the first 1D transform is calculated by adding only positive (negative) terms during the matrix multiplication. During the second 1D transform, positive (negative) co-efficient is multiplied with the positive (negative) maximum error of the first 1D transform to calculate the final positive maximum possible error. To predict conservatively without any accuracy loss, the neuron is predicted to be non-activated only if the "estimated value + max. error" is less than zero. Thus, the proposed method does not allow false-negatives that predict to non-activated for activated neurons.

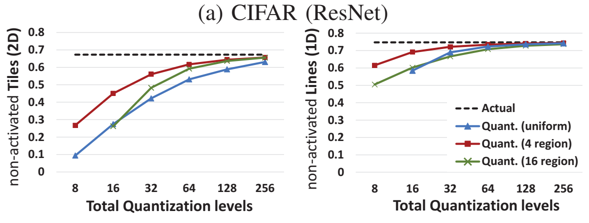
B. Prediction Accuracy and Tile Transfer

Figure 12 shows the actual and prediction ratio of non-activated tiles and lines with $F(2\times 2, 3\times 3)$ transform. We measure the ratio with the pre-trained weights [59] and two data sets – CIFAR [60] and ImageNet [56]. Non-activated tile (line) represents that the neurons transformed from a tile (a line) are predicted to all non-activated at the 2D predict case (1D predict case). The dotted line in Figure 12 shows the ratio measured with real values and represents the upper limit of the prediction. Non-uniform quantization with 4 regions best matches the distribution of Winograd domain values, and results in the best prediction results for all test cases.

[12] For 1D predict case, the first 1D transform process is skipped at the destination worker since it was already done at the source worker.



(a) CIFAR (ResNet)



(b) ImageNet (VGG, GoogLeNet)

Fig. 12. Ratio of non-activated tiles/lines (actual), and predicted to non-activated tiles/lines for two different data sets.

With 64 quantization levels (6-bit) for 2D predict and 32 quantization levels (5-bit) for 1D predict, activation prediction can reduce tile gathering communication by 34.0% and 78.1%, respectively. Overall, 1D predict results in higher prediction accuracy than 2D predict since accumulation of quantization error is reduced as the first 1D transform is performed at the source worker and bypassed at the destination worker.

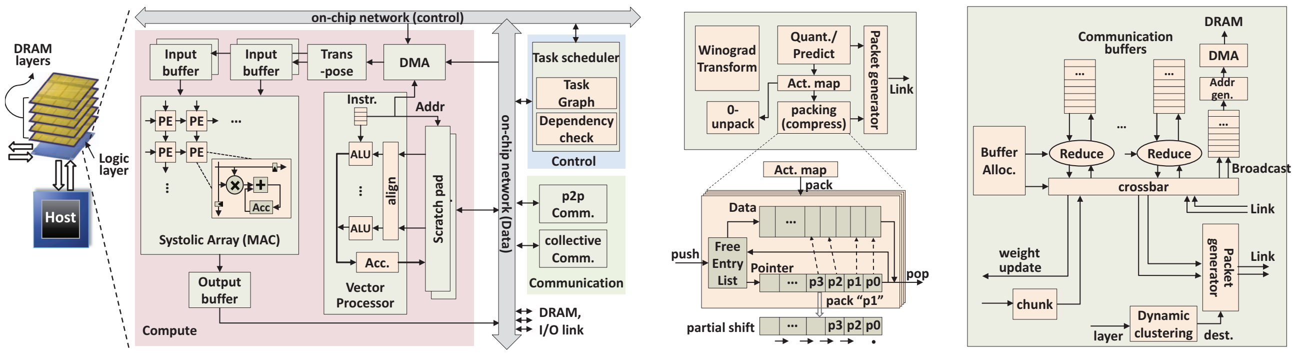
For the scattering of input tiles, zero values of input tiles can be omitted. Zero-skipping reduces input tile scattering communication by 39.3%, 64.7% for the 2D predict and 1D predict, respectively. The skipped data are filled with zero values at the worker computing dot products. The information of skipped data for zero-skipping, and the tile data predicted to be non-activated for activation prediction is shared between the destination workers and source workers through activation map of input and output tiles.

VI. Scalable Near-Data Processing Architecture

In this work, we use near-data processing (NDP) as the worker. As shown earlier in Figure 1, Winograd transformed convolution reduces computation, but increases the amount of data accessed. To enable high bandwidth to memory, we exploit 3D stacked memory with through-silicon vias (TSVs) and logic layer on the bottom that enables near-data processing. In this section, we discuss our proposed scalable near-data processing architecture that implements the proposed multi-dimensional parallel training (MPT), dynamic clustering, and activation prediction. Figure 13(a) shows the block diagram of near-data processing added to logic layer of each memory module, consisting of control, compute, and communication units.

A. Control Units

At the start of CNN training, the host builds a task graph of the given CNN structure and parameters (such as batch size and transform algorithm). A node represents a single task and edge between the nodes represents dependency on the data. We



(a) Block diagram of NDP (b) p2p comm. logic (c) collective comm. logic

Fig. 13. (a) High-level block diagram of near-data processing architecture for MPT, and detailed diagram of (b) peer-to-peer communication logic and (c) collective communication logic.

build task as a computation block that fits on the computation unit (systolic array) and computed concurrently. Thus, a single convolution layer can be composed of multiple task nodes. To reduce synchronization overhead, we use update counter based dependency check. Feature maps may have dependency to the previous layers, and weights may have dependency to the previous iteration. After completion of each task, update counter is incremented, and the next task can start when the update counters of all connected prior tasks are updated. The task graph is stored to each NDP, and the task scheduler in the NDP control loads a task with a pre-defined order and initiates computation and programs DMA when dependency check is passed.

B. Computation Units

Since most of the neural network layers can be mapped to matrix multiplication, a systolic array is commonly used in CNN hardware accelerators [33], [61]–[63]. We also assume a systolic array in our architecture. In this work, we assumed 64×64 FP32 MAC units, which takes up 31% of the logic layer area, assuming that the logic layer is fabricated at 28nm. [13] The number of MAC units was determined by considering not only the area overhead but also balance the computation with the available DRAM bandwidth. Systolic array receives the input data stream from the two sides of the array boundary, and one of the two input data streams needs to be changed to compute a new output. Thus, we assume that half of the input data is unchanged and reused from the on-chip buffer, while the other half of the input data changes and is fetched from DRAM in the worst case scenario. This requires 256 GB/sec of data bandwidth with 1 GHz of logic frequency, as one side of the input data stream is 64-lanes wide with 4-bytes width per lane. This matches well with the 320 GB/sec bandwidth of the 3D-stacked memory [42]. For overlap of the computation and communication, input buffers have double buffering and are sized to 512 KB for each instance of the input buffer (total 2 MB) to fully store the weights of the typical structure of the convolution layers, while the output buffer has a capacity of 128 KB. The areas for the input and output buffers were estimated from CACTI6.5 [67] and found to be 3.19mm^2 at 28nm (4.7% of overhead area).

The vector processor adds programmability to the NDP computation unit for pre- and post-processing of matrix multiplication, such as activation (ReLU), pooling, and simple addition between feature maps (like a residual connection [19], or join operation [52]), where the instruction sequence is loaded from the task descriptor. We used a vector processor based on scratch-pad memory, which loads/stores input/output data directly from/to the scratch-pad memory. The layers of neural networks usually have large input/output data dimensions and have a static and regular computation. Pre- and post-processing for convolution operations are also mostly computation for streaming data (computed once for each data), such as applying ReLU/pooling to the output feature maps. With these characteristics (wide and aligned input data), scratch-pad memory can support wide vector processing units efficiently [68], [69]. [14] Scratch-pad memory of the vector processor also has a double-buffering architecture (with a size of 512KB for each buffer) for the DMA to store output data to DRAM and pre-load input data.

C. Communication Units

Figures 13(b) and (c) show the architecture of two communication processing elements. The peer-to-peer (P2P), unicast communication logic used for tile transfer contains transformation unit and quantize/predict logic described earlier in Section V. After activation prediction, data packing (or compressing) of unnecessary tile data is performed before sending the tile data. Prior work also proposed a packing DMA architecture for CNN [71], but this work differs as a pointer-based shift register is used, instead of shifting the data itself, to reduce data movement in the registers. Output tiles are copied from the output buffer of the systolic array to free data buffers (each buffer is assigned for each desti-

[13] From the synthesis with TSMC 45nm at prior work [64], 32-bit floating point MUL+ADD is 11884um^2, which can be scaled to 5100um^2 with 2.33× scaling [65]. Thus, 4K MAC units result in 20.9mm^2, which is 30.7% of HMC logic die size(68mm^2) [66].

[14] The functional unit of RISC-V vector processor [70] has approximately 0.9um^2 for 4-lanes at 28nm, and we assume 64-lanes, which results in 14.4um^2 (21.2% of logic die size).

TABLE I
THREE CNNS USED IN OUR EVALUATION

Network	Configuration	Data Set	param. size (3x3)
Wide ResNet [20]	WRN-40-10	CIFAR	55.6M (55.5M)
ResNet [19]	ResNet-34	ImageNet	21.6M (21.1M)
FractalNet [52]	4 block, 4 column	ImageNet	164M (163M)

nation worker). The output activation map stores the results of activation prediction, and unnecessary data (non-activated tiles) are packed by partially moving (shifting) the pointer registers. For tile gathering, only the data indicated by the point registers are packetized and sent to destination workers. For tile scattering with zero-skipping, skipped data are filled with zeroes on the receiving side of the worker by referencing the input activation map, which is shared between the source and destination workers.

For collective communication (reduce, broadcast) of weights, ring-based collective operation is used with the proposed hardware architecture (Figure 13(c)). Messages are divided into multiple chunks and transferred in parallel, referred to as *pipelined* transfer [72]. Each chunk is mapped to a single packet inside the memory-centric network. To avoid the blocking of a whole ring by a slow worker and to maximize the bandwidth, we allow concurrent collective operation of multiple messages. Chunks from the same message will arrive in a pre-defined order, but the chunks from different messages can arrive in any order. To properly handle the chunks that arrive without a pre-defined order, we implement multiple Reduce blocks and communication buffers, with each Reduce block assigned to a specific message. Upon the arrival of a chunk from the link or local compute unit, the Reduce block checks whether the chunk exists in the connected communication buffer or not, and updates (if exists) or stores the chunk in the communication buffer (if it does not exist). The updated chunk can be sent to the next worker when the next worker is ready to receive the chunk. The next worker, or destination, is determined by a ring direction and dynamic clustering configuration. After the completion of weight gradients reduction, weights are updated and broadcasted to all workers before being stored in DRAM. [15]

VII. EVALUATION

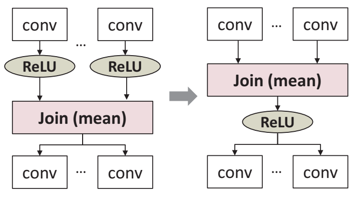
A. Methodology

In our evaluation, we evaluated five typical layers of CNNs (Table II) to evaluate the impact of our proposed architecture in detail. In addition, three state-of-the-art CNNs with large weight size were also evaluated (Table I). For join operation of FractalNet [52] that computes the mean value of the connected input layers, we slightly modified it as the ReLU activation is applied after join operation as shown in Figure 14(a).

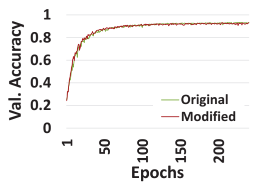
[15] Output data of each task constitutes a single message, and the message and chunk numbers are used to match the chunks to be accumulated at the Reduction process. The message and chunk numbers also generate DRAM address when the updated weights are stored to DRAM at Broadcast process.

TABLE II
FIVE CONVOLUTION LAYERS USED IN OUR EVALUATION

Abbr.	CNN layer	I,J	$x(y)$ dim.	w dim.
Early	ResNet-34 conv2_x	128,128	56×56	3×3
Mid-1	ResNet-34 conv4_x	256,256	14×14	3×3
Late-1	ResNet-34 conv5_x	512,512	7×7	3×3
Mid-2	WRN-40-10 conv3	320,320	16×16	3×3
Late-2	WRN-40-10 conv4	640,640	8×8	3×3



(a) Join modification



(b) Validation accuracy

Fig. 14. Shows (a) modified Join operation of FractalNet, and validation accuracy with the modified Join for CIFAR-10 data set

Transforming to spatial domain is necessary when a non-linear function such as ReLU is applied, but the join operation (consisting of computing the mean value) is a linear function. Thus, the modified join operation can be moved to Winograd domain (i.e., computing mean value of Winograd domain tiles) and reduces the number of transformations to spatial domain and results in reduced tile transfer. The impact of this modification on accuracy was evaluated by training FractalNet with CIFAR-10 data set for 250 epochs, and the modified join resulted in the same validation accuracy as shown in Figure 14(b). Thus, we use the modified join operation for our evaluation of FractalNet on our proposed architecture.

For simulation of the proposed memory-centric network and NDP architectures, we performed cycle-accurate simulation with the configurations summarized in Table III. For both the inter-chip and on-chip network, we modified cycle-accurate network simulator, Booksim [73], to implement the proposed architecture. The logic layer, DRAM accesses, network communication, and the execution model were implemented in the network interface. NDP logic operates with the same frequency with the router (1GHz).

We evaluated a scalable near-data processing architecture with 256 memory modules, or workers, connected to a single multi-core host CPU, with a ring topology inside each group and 2D flattened butterfly (FBFLY) within each cluster (Figure 9). 256 byte packet (chunk) size was used for collective operations to reduce packet overhead while 64 byte packet size was used for other packets. An on-chip network crossbar was used to provide connectivity on the logic layer to interconnect the NDP units, I/O links, and vault controllers.

Full-width link or channel is composed of 16 lanes per direction and 15 Gbps per lane (used for ring topology), while narrow link composed of 8 lanes per direction and 10 Gbps per lane was used within each cluster for the 2D FBFLY. SerDes latency is assumed to be 5ns per hop, with 2.5ns for serialization and 2.5ns for deserialization. For energy

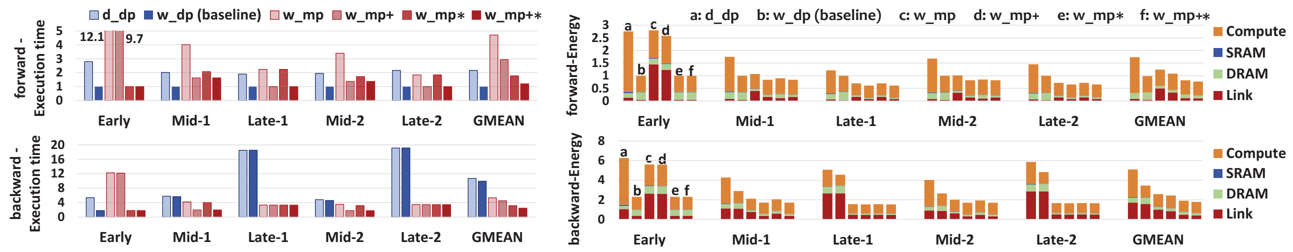


Fig. 15. Execution time and energy of forward (*fprop*) and backward (*bprop* + *updateGrad*) pass, normalized to `w_dp` of forward pass

TABLE III
SIMULATOR CONFIGURATION

	Parameter	Configuration
Network	Router clock	1.0GHz
	Inter-chip link	*full*: 16 lanes × 15 Gbps per lane *narrow*: 8 lanes × 10 Gbps per lane
	Topology	Ring + FBFLY
	Routing	Minimal
Memory	HMC org.	16 vaults, 16 banks per vault
	scheduler	FR-FCFS
	Bandwidth	320GB/sec

TABLE IV
EVALUATED SYSTEM CONFIGURATIONS

Abbr.	System configuration
`d_dp`	Direct convolution with data parallelism (update w)
`w_dp`	Winograd convolution with data parallelism (update w)
`w_mp`	Winograd convolution with MPT (update W)
`w_mp+`	`w_mp` + activation predict/0-skip
`w_mp⋆`	`w_mp` + dynamic clustering
`w_mp+⋆`	`w_mp` + activation predict/0-skip + dynamic clustering

consumption, we used CACTI-3DD [74] and CACTI6.5 [67] to model the 3D-stacked DRAM and SRAM. We used the energy model described in the prior work [45] for the link energy calculation. To estimate the energy in the computation unit, we used estimated values of 0.9pJ (3.7pJ) for 32bit FP ADD (MUL) [75].

B. *Results for layer-wise evaluation*

Figure 15 shows the execution time and energy consumption of the different system configurations shown in Table IV for the five typical convolution layers (Table II). Forward pass includes *fprop* phase, and backward pass includes *bprop* and *updateGrad* phases. Since we focused on comparing different parallelism strategies, we used Winograd convolution with data parallelism (`w_dp`) as the baseline, but we also added a comparison with direct convolution (`d_dp`) to evaluate the impact of the Winograd transformed convolution. To make a fair comparison, we assumed the memory modules have the same I/O link bandwidth for all configurations – four bi-directional full-width links (i.e., total 240 GB/sec [42]). Thus, `w_dp` uses four rings (each ring having a length of 256 workers) for collective operations in *updateGrad*. We optimized the baseline by applying a pipelined collective operation that sends multiple chunks in parallel as described in Section VI. We also assumed that unused links are turned-off for fair energy comparison (e.g., *forward pass* of `w_dp`) while maintaining minimal connectivity to the host.

For MPT, we used half of the I/O link bandwidth for collective operation, which resulted in two rings with each ring having a length of N_c workers for each group. The other half of the bandwidth was used for tile transfer (FBFLY). For `w_mp` configuration, we used a cluster configuration of (N_g, N_c) = (16,16), and dynamic clustering was selected from the three configurations – (1,256), (4,64), and (16,16). In our evaluations, two Winograd transform algorithms are used – $F(2\times2,3\times3)$ and $F(4\times4,3\times3)$. $F(2\times2,3\times3)$ transform is used for MPT configurations with multiple groups to have smaller size of Winograd domain weights, while $F(4\times4,3\times3)$ transform is used for a MPT configuration with a single group to further reduce computation.

Execution Time: Early layer has the largest feature map size and the smallest weight size, and `w_mp` resulted in poor performance, compared to `w_dp`, because of the large amount of tile transfer in *fprop* and *bprop*. To achieve the best performance with the Early layer, dynamic clustering (`w_mp⋆`) was configured as (N_g, N_c) = (1,256) to completely remove tile transfer in *fprop* and *bprop*. Mid layers have medium sized feature maps and weight sizes, and applying MPT alone (`w_mp`) resulted in longer execution time than applying `w_dp` at the forward pass, with a small performance increase in the backward pass. Late layers with small feature maps and large weight sizes showed the biggest benefit from MPT since the large weight size required longer collective communication time. By applying activation prediction and zero-skipping (`w_mp+`) to reduce tile transfer, the overall execution time (forward + backward) was reduced by 2.24× and 4.54× compared to the baseline `w_dp` for Mid layers and Late layers, respectively. MPT with activation prediction and dynamic clustering (`w_mp+⋆`) resulted in 2.74× overall execution time reduction, compared to the baseline `w_dp` on average.

Energy Consumption and DRAM access: We estimated energy consumption from four major factors – Compute units, SRAM access, DRAM access, and I/O link energy in the memory-centric network. Since the Winograd layer required more data accesses than direct convolution, `w_dp` increased DRAM energy consumption compared to `d_dp`. However, the increase in DRAM energy consumption is offset by the reduction in the compute energy from the reduced computation. In addition, `w_mp` actually resulted in lower DRAM

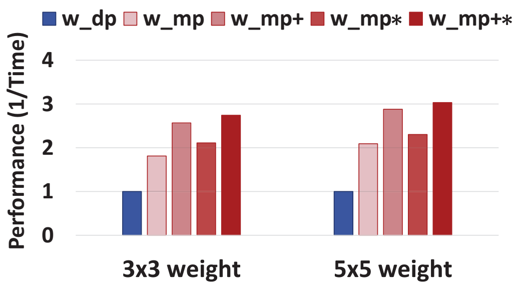


Fig. 16. Average performance of the five layers shown in Table II for two different weight sizes (normalized to w_dp of each configuration).

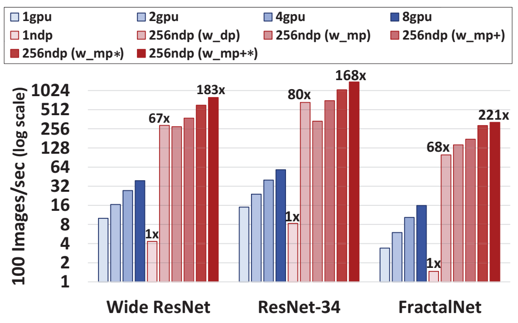


Fig. 17. Performance comparison with multi-GPU system and the proposed MPT with NDP architecture for 256 batch size.

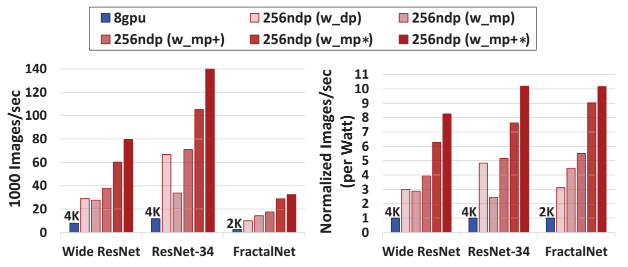


Fig. 18. Performance comparison with large batch size for multi-GPU system (numbers on the plot represent the total batch size that showed the highest performance on multi-GPU system), and performance per watt.

energy consumption compared to d_dp and w_dp. While data parallelism duplicates the full weights across all workers, MPT partitions weights to multiple groups, and thus each worker stores only a fraction of the weights and reduces the overall DRAM energy from accessing the weights. MPT also increases per worker batch size since the input batch is partitioned to N_c clusters, and resulted in more data reuse from the input buffer compared to the data parallel training. The high-speed serial interface of the I/O link consumes energy even in an idle state, and the reduced execution time of MPT (w_mp+∗) in backward pass significantly reduced energy consumption by up to 3.0× for the late layers compared to w_dp.

Network Traffic: With MPT, each worker generates only a fraction of the weight gradients and thus, the number of network packets (or flits) transferred over the memory-centric network for collective operation is reduced by a factor of N_g. Considering the larger size of Winograd domain weights ($|W|$) compared to spatial domain weights ($|w|$), the actual network transfer reduction is proportional to $N_g \cdot \frac{|w|}{|W|}$ in MPT.

Though we used 3×3 weights for layer-wise evaluation, Winograd transform and the proposed MPT apply to weight sizes other than 3×3. For the 5×5 weights, Winograd transform $F(2 \times 2, 5 \times 5)$ can be used with a tile size of 6×6 while for the 3×1 weights, $F(2,3)$ can be used with a tile size of 4×1. Since MPT further reduces weight gradient communication for 5×5 weights compared to 3×3 weights, the benefit from MPT is higher for 5×5 weights. We evaluated the five layers in Table II by replacing 3×3 weight size with 5×5 weight, and used $F(2 \times 2, 5 \times 5)$ transform. Figure 16 shows the normalized performance of the five layers on average with two different weight sizes – 3×3 and 5×5. For 3×3 weights, MPT with dynamic clustering and activation prediction (w_mp+∗) resulted in 2.74× performance improvement compared to baseline data parallel training (w_dp), but for 5×5 weights, w_mp+∗ showed a higher performance improvement of 3.03× compared to w_dp.

In summary, early layers benefit from the reduced computation of Winograd transform itself, but the proposed MPT and activation prediction have a significant impact on performance and energy reduction for the middle and late layers. Additionally, dynamic clustering minimizes the overhead of MPT in the early layers to increase the benefits of MPT. As CNN becomes deeper, more layers are added to the late layers of CNN for the extraction of diverse high level features [19], and the impact of the proposed system on the entire CNN training time will become larger.

C. Results for entire CNN evaluation

To evaluate the impact of MPT on the entire CNN, we evaluated three CNNs (Table I), and compared the results to an NVidia DGX-1 multi-GPU system. We measured the performance and power of the DGX-1 system with eight Volta (V100) GPUs, each with six high-speed NVlinks. NCCL library [38] was used for fast ring-based collective operation, and 6 independent rings were formed when all 8 GPUs were used. We also used TensorFlow-1.4.0 [76] for data parallel training with CUDA9 (cuBLAS) and cuDNN7. Winograd transform for the convolution layers was also enabled and we used the FP16 data type to use the Tensor cores in the Volta GPU. [16]

Figure 17 shows the performance comparison of the multi-GPU system (data parallelism) and the proposed MPT with NDP architecture. A fixed total batch size of 256 was used for all configurations. The multi-GPU system results show sub-linear scaling [78] as the number of GPUs increases since while the amount of communication (or the weight update across the GPUs) remains relatively constant, the amount of computation per GPU *decreases* since the amount of work

[16] NDP worker was also changed to have FP16 multiplication and FP32 adder units [77]. Systolic array is configured to 96×96 MAC array for entire CNN evaluation, which have similar area and power consumption compared to the 64×64 FP32 configuration.

done per GPU decreases with a fixed total batch size. We compare the performance of the multi-GPU system with MPT using 256 NDP workers since the power consumption with 256 NDP workers is approximately similar to 8 GPUs. The same configurations described earlier in Table IV are evaluated and the baseline used is 1 NDP system.

Applying only MPT (`w_mp`) does not necessarily improve the performance compared to data parallelism (`w_dp`) because of the longer execution time in the early layers, and can actually decrease performance for CNNs that have more layers with a large feature map size in particular (e.g., ResNet-34). Dynamic clustering (`w_mp⋆`) configures the early layers to data parallelism (or MPT with a small number of groups) and improves performance 2.2× compared to `w_dp`. Activation prediction (`w_mp+`) reduces tile transfer for all the layers configured to have multiple groups, and improves performance by 1.4× compared to `w_dp`. When both are combined (`w_mp+⋆`), the overall benefit increases to 2.7× compared with `w_dp`. Overall, 256 NDP with data parallelism (`w_dp`) and MPT with dynamic clustering and activation prediction (`w_mp+⋆`) resulted in 71× and 191× increase in performance, compared to a single NDP worker system on average, as MPT (`w_mp+⋆`) resulted in better scalability compared to data parallelism (`w_dp`) by 2.7×. MPT with 256 NDP workers (`w_mp+⋆`) resulted in 21.6× performance improvement over the 8-GPU system with fixed batch size. FractalNet showed better scaling for MPT than the other two CNNs due to the reduced tile transfer with the modified join operation.

To compare the performance and power efficiency without limiting the batch size, we increased the batch size for the multi-GPU system and selected the batch size that resulted in the best performance and the results are shown in Figure 18. (NDP systems still use 256 batch size.) The numbers in Figure 18 indicate the total batch size used for 8-GPU performance measurement (2K to 4K). Since 8-GPU has similar power consumption as 256 NDP workers (1800–2600 Watts), we compared the performance per watt as shown in Figure 18(b). The proposed NDP architecture with MPT (`w_mp+⋆`) showed 9.5× higher performance per watt than 8-GPU system on average.

VIII. Conclusion

In this work, we exploit element-wise computation, or *intra-tile* parallelism, of Winograd transformed convolution to enable higher scalability for training large-scale neural networks. In particular, we propose multi-dimensional parallel training (MPT) of the Winograd layer that combines data parallelism and intra-tile parallelism to reduce the collective communication required for reduction and broadcasting of weight gradients. MPT requires a new type of communication for tile gathering/scattering, which can limit its applicability to layers with large feature maps. To minimize the overhead of MPT, we propose dynamic clustering that balances the two types of communication per layer through dynamic configuration of the connection between the workers. Activation prediction is also proposed to reduce tile gathering communication. We exploit the high-bandwidth of 3D stacked memory with near-data processing including specialized communication logic and high-bandwidth interconnection between the workers to evaluate MPT. Our simulation evaluation demonstrates that MPT with the proposed NDP architecture shows 2.7× and 9.5× higher performance (iso-power) compared to data parallelism on NDP architecture and multi-GPU system, respectively.

Acknowledgment

This research was supported by National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (MSIP) (2015M3C4A7065647) and (2017R1A2B4011457), and in part by Future Non-Von Neumann Architecture Cluster Program of Samsung Electronics.

References

[1] Y. LeCun *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, 1995.
[2] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Proceedings of NIPS*, 2012.
[3] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.
[4] H. Cui *et al.*, "Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server," in *Proceedings of EuroSys*, 2016.
[5] F. N. Iandola *et al.*, "Firecaffe: near-linear acceleration of deep neural network training on compute clusters," in *Proceedings of CVPR*, 2016.
[6] M. Li *et al.*, "Scaling distributed machine learning with the parameter server." in *Proceedings of OSDI*, 2014.
[7] J. Dean *et al.*, "Large scale distributed deep networks," in *Proceedings of NIPS*, 2012.
[8] A. Coates *et al.*, "Deep learning with cots hpc systems," in *Proceedings of ICML*, 2013.
[9] R. Thakur *et al.*, "Optimization of collective communication operations in mpich," *The International Journal of High Performance Computing Applications*, vol. 19, 2005.
[10] C. Woolley, "Nccl: Accelerated multi-gpu collective communications," https://images.nvidia.com/events/sc15/pdfs/NCCL-Woolley.pdf, 2015.
[11] P. Goyal *et al.*, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.
[12] T. Akiba *et al.*, "Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes," *arXiv preprint arXiv:1711.04325*, 2017.
[13] Y. You *et al.*, "Imagenet training in 24 minutes," *arXiv preprint arXiv:1709.05011*, 2017.
[14] N. S. Keskar *et al.*, "On large-batch training for deep learning: Generalization gap and sharp minima," in *Proceedings of ICLR*, 2017.
[15] D. Masters *et al.*, "Revisiting small batch training for deep neural networks," *arXiv preprint arXiv:1804.07612*, 2018.
[16] R. H. Byrd *et al.*, "Sample size selection in optimization methods for machine learning," *Mathematical programming*, vol. 134, 2012.
[17] E. Hoffer *et al.*, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," in *Proceedings of NIPS*, 2017.
[18] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[19] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of CVPR*, 2016.
[20] S. Zagoruyko *et al.*, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.
[21] G. Huang *et al.*, "Densely connected convolutional networks," in *Proceedings of CVPR*, 2017.
[22] A. Lavin *et al.*, "Fast algorithms for convolutional neural networks," in *Proceedings of CVPR*, 2016.
[23] M. Mathieu *et al.*, "Fast training of convolutional networks through ffts," *arXiv preprint arXiv:1312.5851*, 2013.
[24] N. Vasilache *et al.*, "Fast convolutional nets with fbfft: A gpu performance evaluation," *arXiv preprint arXiv:1412.7580*, 2014.
[25] Nervana, "Neon deep learning framework," https://github.com/NervanaSystems/neon.

[26] Intel, “vtune,” https://software.intel.com/en-us/intel-vtune-amplifier-xe.

[27] G. Kim *et al.*, “Memory-centric system interconnect design with hybrid memory cubes,” in *Proceedings of PACT*, 2013.

[28] G. Kim *et al.*, “Multi-gpu system design with memory networks,” in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*. IEEE, 2014.

[29] S. Li *et al.*, “Enabling sparse winograd convolution by native pruning,” *arXiv preprint arXiv:1702.08597*, 2017.

[30] S. Chetlur *et al.*, “cudnn: Efficient primitives for deep learning,” *arXiv preprint arXiv:1410.0759*, 2014.

[31] K. Vincent *et al.*, “On improving the numerical stability of winograd convolutions,” in *ICLR Workshop*, 2017.

[32] T. Chen *et al.*, “Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” in *ACM Sigplan Notices*, 2014.

[33] Y.-H. Chen *et al.*, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE Journal of Solid-State Circuits*, 2017.

[34] S. Han *et al.*, “Eie: efficient inference engine on compressed deep neural network,” in *Proceedings of ISCA*, 2016.

[35] D. Kim *et al.*, “Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory,” in *Proceedings of ISCA*, 2016.

[36] M. Gao *et al.*, “Tetris: Scalable and efficient neural network acceleration with 3d memory,” in *Proceedings of ASPLOS*, 2017.

[37] S. Venkataramani *et al.*, “Scaledeep: A scalable compute architecture for learning and evaluating deep networks,” in *Proceedings of ISCA*, 2017.

[38] S. Jeaugey, “Nccl 2.0,” in *GTC*, 2017.

[39] M. Cho *et al.*, “Powerai ddl,” *arXiv preprint arXiv:1708.02188*, 2017.

[40] Y. LeCun *et al.*, “Efficient backprop in neural networks: Tricks of the trade,” *Lecture Notes in Computer Science*, vol. 1524.

[41] L. Durant *et al.*, “Inside volta: The worlds most advanced data center gpu,” 2017.

[42] Hybrid Memory Cube Consortium, “Hybrid Memory Cube Specification 2.1,” 2014.

[43] G. H. Loh, “A register-file approach for row buffer caches in die-stacked DRAMs,” in *Proceedings of MICRO*, 2011.

[44] S. H. Pugsley *et al.*, “Ndc: Analyzing the impact of 3d-stacked memory+logic devices on mapreduce workloads,” in *Proceedings of ISPASS*, 2014.

[45] B. Hong *et al.*, “Accelerating linked-list traversal through near-data processing,” in *Proceedings of PACT*, 2016.

[46] E. Azarkhish *et al.*, “Neurostream: Scalable and energy efficient deep learning with smart memory cubes,” *arXiv preprint arXiv:1701.06420*, 2017.

[47] L. Xu *et al.*, “Scaling deep learning on multiple in-memory processors,” in *WoNDP: 3rd Workshop on Near-Data Processing*, 2015.

[48] D. Foley, “Nvlink, pascal and stacked memory: Feeding the appetite for big data,” 2014.

[49] CCIX Consortium, “Cache Coherent Interconnect for Accelerators (CCIX),” http://www.ccixconsortium.com.

[50] OpenCAPI Consortium, https://opencapi.org.

[51] GenZ Consortium, https://genzconsortium.org.

[52] G. Larsson *et al.*, “Fractalnet: Ultra-deep neural networks without residuals,” in *Proceedings of ICLR*, 2017.

[53] A. Faraj *et al.*, “Bandwidth efficient all-to-all broadcast on switched clusters,” *International Journal of Parallel Programming*, vol. 36, 2008.

[54] J. Kim *et al.*, “Flattened butterfly topology for on-chip networks,” in *Proceedings of MICRO*, 2007.

[55] G. E. Dahl *et al.*, “Improving deep neural networks for lvcsr using rectified linear units and dropout,” in *Proceedings of ICASSP*, 2013.

[56] J. Deng *et al.*, “Imagenet: A large-scale hierarchical image database,” in *Proceedings of CVPR*, 2009.

[57] Z. Cai *et al.*, “Deep learning with low precision by half-wave gaussian quantization,” in *Proceedings of CVPR*, 2017.

[58] D. Miyashita *et al.*, “Convolutional neural networks using logarithmic data representation,” *arXiv preprint arXiv:1603.01025*, 2016.

[59] Nervana, “Modelzoo,” https://github.com/NervanaSystems/ModelZoo.

[60] A. Krizhevsky *et al.*, “Learning multiple layers of features from tiny images,” in *Technical report, University of Toronto*, 2009.

[61] N. P. Jouppi *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of ISCA*, 2017.

[62] S. Gupta *et al.*, “Deep learning with limited numerical precision,” in *Proceedings of ICML*, 2015.

[63] K. Ovtcharov *et al.*, “Toward accelerating deep learning at scale using specialized hardware in the datacenter,” in *Hot Chips 27 Symposium*, 2015.

[64] W. Dally, “High-performance hardware for machine learning,” *NIPS Tutorial*, 2015.

[65] F. Arnaud *et al.*, “Competitive and cost effective high-k based 28nm cmos technology for low power applications,” in *International Electron Devices Meeting (IEDM)*, 2009.

[66] J. Jeddeloh *et al.*, “Hybrid memory cube new dram architecture increases density and performance,” in *Symposium on VLSI Technology*, 2012.

[67] N. Muralimanohar *et al.*, “Cacti 6.0: A tool to model large caches,” *HP Laboratories*, 2009.

[68] C. H. Chou *et al.*, “Vegas: Soft vector processor with scratchpad memory,” in *Proceedings of FPGA*, 2011.

[69] A. Severance *et al.*, “Venice: A compact vector processor for fpga applications,” in *Proceeding of FPT*, 2012.

[70] Y. Lee *et al.*, “Hwacha preliminary evaluation results, version 3.8.” *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-264*, 2015.

[71] M. Rhu *et al.*, “Compressing dma engine: Leveraging activation sparsity for training deep neural networks,” in *Proceedings of HPCA*, 2018.

[72] J. Worringen, “Pipelining and overlapping for mpi collective operations,” in *Proceeding of Local Computer Networks*, 2003.

[73] N. Jiang *et al.*, “A detailed and flexible cycle-accurate network-on-chip simulator,” in *Proceedings of ISPASS*, 2013.

[74] K. Chen *et al.*, “Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory,” in *Proceedings of DATE*, 2012.

[75] M. Horowitz, “1.1 computing’s energy problem (and what we can do about it),” in *IEEE Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014.

[76] M. Abadi *et al.*, “Tensorflow: A system for large-scale machine learning.” in *Proceedings of OSDI*, 2016.

[77] P. Micikevicius *et al.*, “Mixed precision training,” *arXiv preprint arXiv:1710.03740*, 2017.

[78] S. A. Mojumder *et al.*, “Profiling dnn workloads on a volta-based dgx-1 system,” in *Proceedings of IISWC*, 2018.