2018 51st Annual IEEE/ACM International Symposium on Microarchitecture

Beyond the Memory Wall: A Case for Memory-centric HPC System for Deep Learning

Youngeun Kwon Minsoo Rhu
School of Electrical Engineering
KAIST
{yekwon, mrhu}@kaist.ac.kr

***Abstract*—As the models and the datasets to train deep learning (DL) models scale, system architects are faced with new challenges, one of which is the memory capacity bottleneck, where the limited physical memory inside the accelerator device constrains the algorithm that can be studied. We propose a memory-centric deep learning system that can transparently expand the memory capacity available to the accelerators while also providing fast inter-device communication for parallel training. Our proposal aggregates a pool of memory modules locally within the device-side interconnect, which are decoupled from the host interface and function as a vehicle for transparent memory capacity expansion. Compared to conventional systems, our proposal achieves an average $2.8\times$ speedup on eight DL applications and increases the system-wide memory capacity to tens of TBs.**

***Index Terms*—System architecture, HPC, machine learning**

I. Introduction

Deep learning (DL) models and its training datasets are scaling at a phenomenal rate, so the progress in DL is primarily limited by how fast the deep neural network (DNN) model can be evaluated and how large of a memory you can utilize for training. DL practitioners are therefore seeking efficient *parallel* training solutions, increasingly adopting a *dense* system node design, which houses several PCIe-attached co-processor devices [1]–[3] to increase the node-level throughput. As the system-level performance is contingent upon how the DL algorithm is parallelized across the devices and how effectively they communicate with each other, leading vendors in this space are employing a custom *device-side interconnection network* that utilizes proprietary high-bandwidth signaling solutions (e.g., NVIDIA's NVLINK which provides 100s of GB/sec of bandwidth) for fast communication and synchronization [3], [4]. Such *device-centric* deep learning system architecture (`DC-DLA`) is becoming mainstream for DL (Figure 1(a)) and we are seeing an increasing number of HPC systems that employ a standalone, device-side interconnection network for efficient DL parallelization [3], [5]–[7].

As researchers seek to deploy deeper and larger DNN topologies however, end-users are faced upon a memory "capacity" wall, where the limited on-device physical memory (based on on-package 3D stack memory [8] in NVIDIA's V100 [1], Google's TPUv2 [2], and Intel-Nervana's Lake Crest accelerator [3]) constrains the algorithm that can be trained [9]–[12]. Increasing the capacity of these on-package stacked DRAM however is challenging due to wireability of the silicon interposers, chip pinout required to drive the added DRAM stacks, and technology limits on how many DRAM stacks you can vertically integrate. Consequently, recent solutions have proposed to use the device (GPU/TPU) memory as an application level cache with respect to the host CPU memory [9], [10], [13]–[16], effectively *virtualizing* DNN memory usage across the host and device memory via PCIe (Section II-B). The effectiveness of these prior solutions however is sensitive to the host-device communication bandwidth as it determines the latency incurred in migrating DNN data in/out of these two memory regions. The left-axis in Figure 2 shows the execution time of state-of-the-art convolutional neural networks (CNNs) on successive versions of a *single*, high-end DL accelerator, which has been reduced by a factor of $20\times$ to $34\times$ over five years. During these time periods, the signaling circuitry and the overall communication bandwidth offered by the latest PCIe and InfiniBand has improved only by a factor of $1\times$ (PCIe gen3) and $3.5\times$ (i.e., IB-FDR to IB-HDR), respectively. This has led to a steadily increasing performance overhead of host-device memory virtualization via PCIe (right-axis in Figure 2), where the growing performance gap between the device computing power and host-device (PCIe) communication bandwidth aggravates system-level performance. Virtualizing DNN memory over a *multi*-GPU/TPU system incurs even higher performance overheads because the effective host–device communication bandwidth allocated per device gets proportionally reduced to the number of intra-node devices. Therefore, the overall system can experience a significant performance slowdown due to the additional latency incurred during host–device memory copies (Section V). Overall, current trends point to an urgent need for a system architectural solution that satisfies the dual requirements of (a) fast inter-device communication for parallel training, and (b) high-performance memory virtualization over a large memory pool to enable memory hungry DNNs to be trainable over accelerator devices.

In this paper, we make a case for a *memory-centric* deep learning system architecture (`MC-DLA`) that aggregates a pool of capacity-optimized memory modules within the device-side interconnect for transparent memory capacity expansion (Figure 1(b)). While our proposal is reminiscent of prior disaggregated memory proposals [17], [18], the CPU-centric memory disaggregation solutions suffer from the same performance bottlenecks of `DC-DLA` because of its reliance on PCIe. In our proposal, the pool of memory modules (henceforth referred to as *memory-nodes*) are completely decoupled from the legacy, host-device interface (e.g., PCIe) and are stationed

978-1-5386-6240-3/18/$31.00 ©2018 IEEE
DOI 10.1109/MICRO.2018.00021

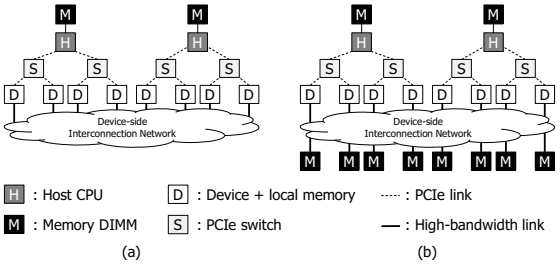

Fig. 1: (a) A device-centric deep learning system architecture, and (b) a memory-centric deep learning system architecture.

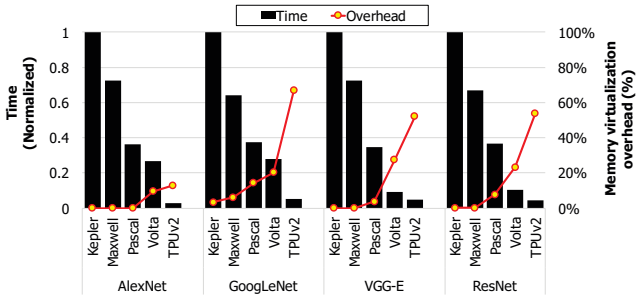

Fig. 2: Execution time of running state-of-the-art CNN models across five recent generation of a single DL accelerator device (left-axis) and the performance overhead incurred due to memory virtualization (right-axis). See Section IV for evaluation methodology.

locally within the device-side interconnect. We propose to interconnect the accelerators and the memory-nodes using the high-bandwidth, low-latency signaling links (e.g., NVLINK) and utilize the memory-nodes as a backing store to the accelerators. This allows the memory-nodes to function as a vehicle for transparent memory capacity expansion, allowing researchers to train DL algorithms that are much larger, deeper, and more complex. Because the accelerators access the memory-nodes via the high-bandwidth links, the performance overhead of virtualizing memory can be substantially reduced. At the same time, MC-DLA connects the accelerators and the memory-nodes in a manner that maximizes inter-device communication bandwidth so that DC-DLA's high-efficiency in conducting collective communication operations is maintained. Overall, this paper makes the following contributions:

- To the best of our knowledge, our work is the first that highlights the importance of device-side interconnects in training scaled up DL algorithms, presenting a quantitative analysis on parallel training in the context of HPC systems with multiple accelerator (GPU/TPU) devices.
- This work identifies key system-level performance bottlenecks on DC-DLA and motivates the need for a new system architecture that balances fast communication and user productivity in training large DNN algorithms.
- We propose and evaluate a system architecture called MC-DLA that provides transparent memory capacity expansion while also enabling fast inter-device communication. Compared to DC-DLA designs, our proposal achieves an average 2.8× performance improvement while expanding the system-wide memory capacity exposed to the accelerators to tens of TBs.

II. Background and Motivation

A. DL Training versus Inference

DNNs require *training* to be ready for *inference*. Training is a three-step process that involves learning the optimal values of the DNN weights using the backpropagation algorithm [19]. First, a serialized, layer-wise computation process called *forward propagation* is taken from the first (input) layer to the last (output) layer in a serialized fashion (the blue arrows from bottom to top in Figure 3). A given layer applies a set of mathematical operation (e.g., convolution, activation, recurrence, etc) to the input *feature maps* (X) and derives the output feature maps (Y), which is forwarded to the next layer to be used as its input feature maps. At the end of forward propagation, a prediction of the input is given, which is compared against the ground truth. The defined loss function quantifies the magnitude of the error between the current prediction and the ground truth, which is encapsulated in a value called *gradients* of the loss function with respect to the last layer's output. Then, *backpropagation* is performed in the opposite direction of forward propagation (the red arrows from top to bottom) again in a layer-wise manner, where the incoming gradients (dY) are used to derive the output gradients (dX) to be sent to the previous layer to be used as its input gradients. Using the dY and dX, each layer derives its own weight gradients (dW) to adjust its own layer's weights (W) so that the loss value is incrementally reduced, improving the performance of the DNN model.

B. Virtualizing Memory for Deep Learning

The chain-rule based backpropagation algorithm requires a given layer's input feature maps values (X) to derive the gradient values of the layer's weights (dW) [19]. Consequently, the overall memory allocation size of DNN training scales proportionally to the network depth (i.e., memory cost of *O(N)* to train a DNN with *N* layers). End-users must therefore carefully tune their network topology (i.e., the number of layers in a DNN and the inter-layer connections) and the training batch size to make sure the overall memory requirement fits within the physical memory capacity, which can severely limit user productivity. Given recent research trends where DL practitioners are seeking to deploy ever larger and deeper network algorithms (e.g., the memory allocations required for training can easily exceed 100s of GBs [20]–[26]), tackling this memory capacity bottleneck while minimizing performance overheads becomes vital in enabling researchers to keep studying scaled up DL algorithms. Prior work on virtualizing memory usage of DNNs [9], [10], [13]–[16] have proposed to utilize both host and device memory concurrently for allocating data structures for DNN training. By leveraging the user-level DNN topology graph as means to extract a compile-time data dependency information (which is encapsulated as a direct acyclic graph (DAG) data structure) of the memory-hungry data structures, e.g., feature maps (X) and/or weights (W), DNN virtual memory can leverage this

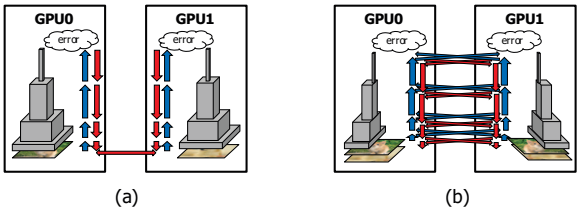

Fig. 3: (a) Data-parallel and (b) model-parallel training. Blue and red arrows inside each device represent a given layer's computation during forward and backward propagation, respectively. The blue and red arrows that crosses the boundaries of two devices represent the per layer inter-device communication and synchronization operations. As shown, model-parallel training incurs much more frequent synchronization than data-parallel training [31], [32].

data dependency information to derive the DNN data *reuse distance* to schedule performance-aware data copy operations via *memory-overlaying* across host and device memory via PCIe [27]–[29]. Existing DL frameworks [10], [15], [30] therefore opt to leverage this DAG to schedule software-level DMA initiated data transfers between host and device, overlapping it with the DNN forward and backward propagation, to maximally utilize the PCIe communication bandwidth and minimize the performance overheads of data migration. By only keeping soon-to-be used DNN data inside the device memory, the memory allocation size of training a network with N layers can be reduced from *O(N)* to $O(1)$, enhancing DL practitioners' ability to train scaled-up algorithms.

C. Parallelization of DL Algorithms

As the DNN algorithm gets more complex and deeper [20]–[22], the need for distributed multi-node systems, each with multiple accelerator devices, have significantly increased to provide high computing horsepower for DL practitioners. Consequently, efficient parallelization of DL algorithms and fast communication among the devices become vital for maximally exploiting the HPC systems based on these dense multi-device nodes. Note that the scope of this paper is on developing an efficient *intra*-node system architecture, so as in conventional designs, we assume that *inter*-node communication is handled using MPI via Ethernet or InfiniBand.

Parallel DL Training. The most popular parallel training strategies employed by DL frameworks are *data-parallel* and *model-parallel* training (Figure 3). Data-parallel training is a parallelization scheme that allocates the same network model across all the workers, but each worker is assigned with a different batch of the overall training dataset. In model-parallel training however, all workers work on an identical batch of the training dataset (i.e., the problem size is fixed at batch size `N`), but each are allocated with different portions of the network model. The parallel tasks distributed across the workers must periodically *synchronize* to have a consistent DNN model trained within each worker, preventing both data-parallel and model-parallel training from achieving perfect scaling. Model-parallel training generally incurs much frequent synchronization than data-parallel approaches as the input feature maps (`X`) and gradients (`dX`, `dW`) must be aggregated across layer boundaries due to the nature of its

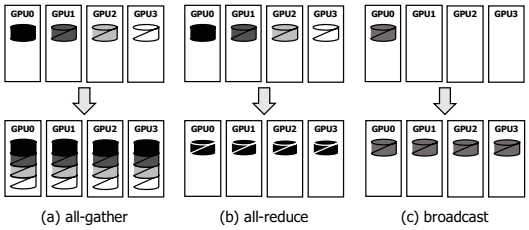

Fig. 4: Key collective communication primitives for parallel training.

parallelization algorithm. Data-parallel training, on the other hand, only requires the accumulation of `dW` during backpropagation and is therefore assumed to be more amenable for achieving close to linear speedup. However, not all networks or layers can be data-parallelized easily, especially for DNNs with large models [24], [33], so both model and data-parallel training are considered important in quantifying the robustness of the system interconnect design of HPC systems.

Communication. As implied through previous discussions, minimizing communication overheads is key in high-performance parallel training. Consequently, maximally utilizing the communication bandwidth provisioned across accelerators within and across compute nodes is crucial. Key collective communication primitives for parallel training are all-gather (`X`), all-reduce (`dX` and `dW`), and broadcast (`dW`), which are shown in Figure 4. Prior work [34], [35] has demonstrated that the *ring-algorithm* based collective communication can provide optimal link bandwidth utilization for the aforementioned collective operations. Leading system vendors in this space are therefore employing a topology-aware, ring-algorithm based collective communication library (e.g., NVIDIA's NCCL [36], IBM's PowerAI DDL [30], and Baidu's AllReduce [37]). These libraries cast the underlying system interconnect as multiple ring networks and orchestrate the DL communication operations based on the ring-algorithm for maximizing bandwidth utilization while minimizing latency.

Device-side Interconnects for DL. For efficient communication and synchronization across accelerator devices, recent HPC systems for DL are employing proprietary, high-bandwidth *device-side interconnection networks* that provide 100s of GB/sec of inter-device communication bandwidth. Intel-Nervana's Lake Crest accelerator [3] employs 12 high-bandwidth signaling links (20× that of what PCIe provides) that can tightly couple the DL accelerator devices with each other. NVIDIA's DGX system [5] is equipped with 8 Volta V100 [1] GPUs where each V100 comes with 6 high-bandwidth NVLINKs (bi-directional bandwidth of 50 GB/sec per link, aggregate channel bandwidth of 300 GB/sec per GPU), which are used to form a cube-mesh topology across eight V100s (Figure 5). By casting the cube-mesh topology as three ring interconnects, the eight GPUs communicate through these high-bandwidth ring networks using the NCCL library, which helps achieve optimal bandwidth utilization and minimize latency. While such *device-centric* deep learning system architecture (`DC-DLA`) solution has advantages in terms of inter-device synchronizations, communicating with the host-side CPU can only be done using the legacy PCIe link, which

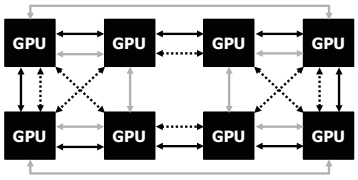

Fig. 5: The cube-mesh, device-side interconnect employed in NVIDIA's DGX system. The black, gray, and dotted arrows form three ring networks for collective communication operations [5], [7].

can cause severe performance bottlenecks for devices utilizing the host CPU memory for memory virtualization.

One might expect that a system architecture that is optimized in the other end of the spectrum, where the high bandwidth links are all or partially used to access CPU memory, could achieve the best of both system-level communication (that is, providing decent communication bandwidth using the now singular or duo ring networks) and high-performance virtual memory (having high enough bandwidth to read/write to/from CPU memory). Such *host-centric* DL system architecture (HC-DLA) however falls short on several aspects. First, an HC-DLA system is simply not a feasible option to begin with for the vast majority of the current, x86-based HPC systems because these proprietary high-bandwidth signaling links for device-side interconnects are incompatible with x86 CPUs. HC-DLA systems that do enable high-bandwidth CPU-GPU communications (e.g., IBM Power + NVIDIA GPUs) still face the following challenges. First, allocating a subset of the high-bandwidth links to connect to the host CPU leaves smaller inter-device bandwidth available for device communication, which could potentially slowdown the effective node-level throughput for algorithms sensitive to inter-device communications. More importantly, however, having just a single high-bandwidth link per each accelerator device directly connected to the host CPU can leave little memory bandwidth available to the CPU itself, leading to a highly unbalanced system design. As we detail in the next section, virtualizing the memory usage of DL algorithms require the DMA engine to fully utilize the communication link bandwidth in order to effectively hide the latency of copying data in and out of device memory. This means that a singular high-bandwidth link of 25 GB/sec per device would amount to a total of 100 GB/sec of worst-case host-side memory bandwidth consumption when accounting for the four PCIe-attached devices connected to a single CPU socket. As a point of reference, the maximum memory bandwidth available for a high-end Intel Xeon CPU and the IBM Power9 is "only" 80 GB/sec [38] and 120 GB/sec [39] per socket, respectively, due to CPU's latency-oriented design (rather than the throughput-oriented GPUs or Google's TPUs [2] that require high bandwidth rather than low latency). As quantitatively discussed in Section V, HC-DLA can consume an average 92% of host-side memory bandwidth for certain workloads, leaving only 8% of memory bandwidth available for the host CPU itself. While we explore such design point in this paper, we argue that such unbalanced system architecture is less practical for future DL systems as it severely lacks design flexibility; that is, the amount of read/write throughput the system designer can provision for host-device memory virtualization is limited by the maximum memory bandwidth available per each CPU socket, regardless of how much device-side high-bandwidth links are available, which can cause severe bottlenecks for future algorithms that are much larger, deeper, and more complex.

III. Memory-centric HPC System Architecture for Deep Learning

In this paper, we propose a new architectural solution for future HPC systems optimized for deep learning. Our goal is to develop a DL system architecture that enables *fast inter-device communication for parallel training* while at the same time *provisioning high-bandwidth communication channels to a pool of capacity-optimized memory modules* (which we refer to as *memory-nodes* in the rest of this paper) for high-performance virtual memory. We argue that HPC system architectures for DL training should be designed in a *memory-centric* manner as the memory "capacity" wall poses one of the biggest challenges in training deep and large learning algorithms [9]–[11]. Prior work on disaggregated memory [17], [18] can similarly expand the pool of memory exposed to the system through a separate memory-blade accessed over PCIe or the NIC. Similar to DC-DLA however, the growing performance gap between (GPU/TPU) device computing power and host-device communication (Figure 2) renders the CPU-centric, PCIe-based memory disaggregation solutions impractical for deep learning training as the latency to access the added memory pool will become bottlenecked by PCIe. Consequently, the memory-nodes in our memory-centric DL system architecture (MC-DLA) are stationed *locally* inside the device-side interconnection network, eliminating all its ties with the host PCIe interface. This section details the design of the memory-node architecture and its application for our proposed MC-DLA system that leverages these memory-nodes as building blocks to achieve the aforementioned design goals. As the scope of this paper is on studying the *intra*-node system architecture, we refer to the PCIe-attached accelerator devices (e.g., GPUs or TPUs) as *device-nodes* in the rest of this paper because both the memory-nodes and device-nodes function as separate nodes inside the device-side interconnect (Figure 1(b)). MC-DLA is applicable for both GPUs and TPUs as our proposal concerns an efficient *system* architecture design for DL accelerators (i.e., the device-nodes). For ease of explanation, we assume the device-nodes are based on GPUs and use terminologies defined in NVIDIA's CUDA hardware/software interface in the remainder of this section.

A. Memory Node Architecture

The key objective of our memory-node design is to unlock the high-bandwidth communication channels of the device-side interconnect for high-performance virtual memory. Figure 6 illustrates the design of our memory-node architecture, which contains N high-bandwidth links for communicating with the device-side interconnection network. The N links are logically partitioned into M groups ($M \leq N$) and each group of (N/M) links are used exclusively by a designated device-node for DNN memory virtualization. A protocol engine that

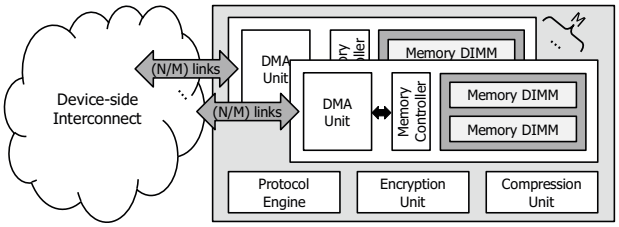

Fig. 6: Memory-node architecture.

is compatible with the device-side interconnect is used to provide a maximum bandwidth of B GB/sec per link, so a device-node assigned with a group of (N/M) links can utilize the DMA engine to read (write) data from (to) the memory DIMMs with (N/M)×B GB/sec of throughput. The DMA engine forwards a device-node's data transfer request to the memory controller which has an array of commodity memory DIMMs it manages. An ASIC that handles encryption or compression can optionally be added to the memory-node. This paper assumes that the memory DIMMs are populated with capacity and density optimized commodity memory solutions: from 8–16 GB DDR4 RDIMMs (registered DIMMs) to 32–128 GB LRDIMMs (load-reduced DIMMs). To narrow down the design space we explore, the rest of this paper assumes that the board housing a single memory-node is sized equivalent to a high-end PCIe accelerator board to be compatible with existing device-side interconnects and minimize the design costs of the server chassis enclosure. A memory-node built out of a mezzanine board sized equivalent to Volta V100's (14 cm × 8 cm) can house ten DDR4 DIMMs, providing a maximum of 170 GB/sec (PC4-17000) to 256 GB/sec (PC4-25600) of memory bandwidth with an overall memory capacity expansion of 80 GB to 1.3 TB per memory-node.

B. System Architecture

As our work is the first that highlights the importance of device-side interconnects in training scaled-up DL algorithms, system architects are given a wide design space under our proposal. A full design space exploration is beyond the scope of this paper, so this section presents three system interconnect design points that incorporate our memory-nodes and discuss their trade-offs in terms of link bandwidth utilization and overall performance. To narrow down our design options, we assume that the number of device-nodes and memory-nodes are identical and that all device-nodes and memory-nodes have N high-bandwidth communication links to interface with the other nodes in the network (each link providing B GB/sec of uni-directional communication bandwidth, Figure 6). We use the system configuration of NVIDIA's DGX system (N=6 high-bandwidth links per device, each link providing B=25 GB/sec communication bandwidth) as a running example to describe the design intuitions behind MC-DLA.

System Interconnect. The design objective of the MC-DLA device-side interconnect is to balance communication, memory virtualization, and overall design complexity. A straightforward and an intuitive interconnect design that can utilize our memory-nodes as a backing store to the device-nodes

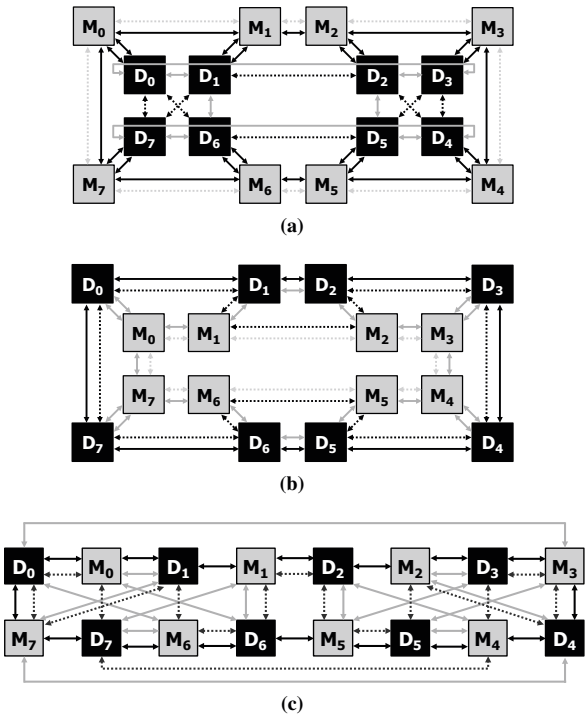

Fig. 7: MC-DLA system interconnect containing 8 device-nodes and 8 memory-nodes: (a) a derivative interconnect design based on the cube-mesh topology of Figure 5, where the device-nodes (D_{0-7}) communicate with the memory-nodes (M_{0-7}) under a star topology, (b) the memory-nodes folded inward, and (c) the proposed ring-based system interconnect. Nodes that are part of the same ring are interconnected using the same color-coded arrows (links).

is shown in Figure 7(a). Here, the communication links that constitute one of the (N/2)=3 ring networks in Figure 5 (e.g., the singular ring network constructed using the 8 bi-directional black arrows) are rearranged to construct a ring network using the 8 memory-nodes and 8 device-nodes. Each device-node is now provided with the ability to access its designated memory-node using two high-bandwidth links (50 GB/sec communication bandwidth between $D_n \leftrightarrow M_n$), significantly reducing the latency to migrate data to/from the backing store. There are two significant limitations with this design however, as (1) the 3 rings used for inter-device collective communication are constructed in a highly unbalanced fashion (i.e., 2 rings are constructed with a maximum 8 hop count while the remaining ring incurs a maximum 24 hop count[1]), rendering the overall communication latency be bottlenecked by the longest ring, and (2) the 8 light-gray/dotted bi-directional links are neither being utilized for communication nor for memory virtualization, failing to maximally utilize available communication resources[2]. Figure 7(b) is an alternative design point that

[1]In Figure 7(a), each memory-node is visited *twice* when traversing the black-arrowed ring network, e.g., $\cdots M_0 \rightarrow D_0 \rightarrow M_0 \rightarrow M_7 \rightarrow D_7 \rightarrow M_7 \cdots$

[2]The light-gray/dotted arrows form the 4th ring with only the 8 memory-nodes, without any device-nodes (i.e., all device-nodes are already fully utilizing the N=6 links). For parallel DL training, the messages to be communicated across the devices are generated by the device-nodes (stored inside GPU memory) and never inside the memory-nodes, so the 4th ring does not help improve the performance of communication nor memory virtualization.

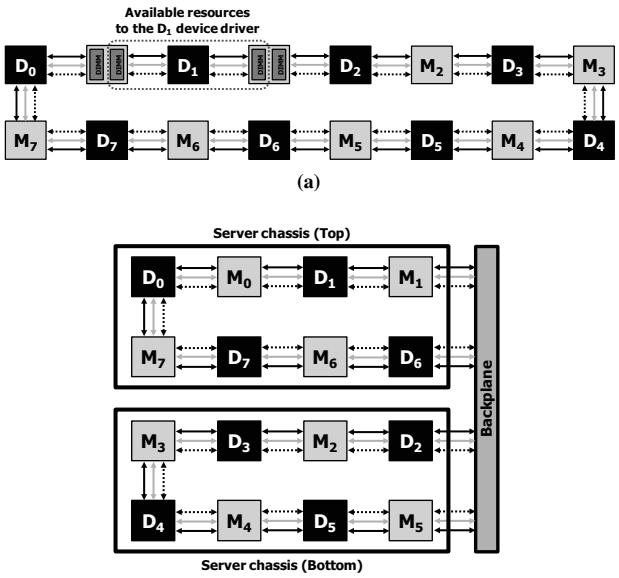

Fig. 8: (a) Ring-based MC-DLA system interconnect optimized for packaging and design complexity (e.g., equal length inter-node links), and (b) the physical design of MC-DLA where each half of the ring is connected via the enclosure backplane.

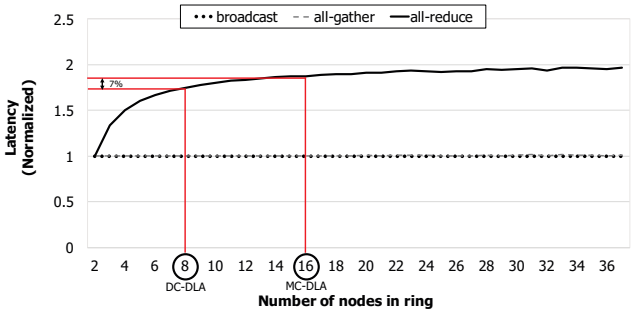

Fig. 9: Latency incurred when performing collective communication primitives, as a function of the number of nodes inside the ring network (normalized to a ring with 2 nodes). Each link has 50 GB/sec of bi-directional bandwidth and the nodes communicate with a message size of 4 KB with a target synchronization size at 8 MB.

better balances the 3 ring networks' performance (the 3 rings containing the device-nodes are constructed with a maximum 8, 12, and 20 hop count, respectively), but it similarly suffers from the aforementioned limitations: unbalanced ring design and underutilization of communication resources.

To holistically address these challenges, we arrive at our *ring* based MC-DLA interconnect that maintains competitive collective communication performance of DC-DLA while, at the same time, significantly improving and maximizing the communication bandwidth available for memory virtualization. Figure 7(c) illustrates our proposed ring-based MC-DLA interconnect design (N/2=3 rings overall), where a given device-node is provided with a pair of high-bandwidth links to two memory nodes located on its left and right side of any given ring. The key advantage of our ring-based MC-DLA architecture is twofold. First, each device-node is now able to utilize the two memory-nodes located on its (logical) left and right side of a given ring, maximally utilizing the N=6 links for virtualizing memory (3× higher bandwidth than in Figure 7(a,b)). This allows MC-DLA to achieve (number of rings)*(link bandwidth to left and right nodes) = (N/2)*(2*B) = 150 GB/sec of communication bandwidth, a significant improvement over the legacy PCIe. Second, the communication bandwidth to the memory-nodes can linearly *scale*, proportional to the signaling technology used to implement these high-bandwidth links, as opposed to the PCIe-based DC-DLA or HC-DLA design, whose maximal communication bandwidth is capped at the maximum CPU socket-level memory bandwidth. For instance, both DC-DLA and HC-DLA, regardless of whether the host-device interface is designed using NVLINK or the next-generation PCIe, can only provide up to the maximum per CPU socket-level memory bandwidth, which is approximately 80 GB/sec and 120 GB/sec for high-end Intel Xeon and IBM Power CPUs, respectively. The eight accelerator devices in our MC-DLA has (150 GB/sec per device × 8 devices) = 1200 GB/sec of communication bandwidth to its neighboring memory-nodes, the number of which will proportionally grow as a function of the link bandwidth of B GB/sec. Figure 8 is an illustration of our ring-based MC-DLA re-designed to be optimized for packaging costs, easing its adoption for real-world HPC systems.

One might be concerned that the latency incurred for collective communications will increase as MC-DLA adds 8 additional (memory) nodes, effectively doubling the number of nodes inside the ring. For reasonably large messages, our ring-based MC-DLA with 16 nodes incur negligible latency overheads for all-gather, broadcast, and all-reduce (Figure 9). When the communication size is small, MC-DLA does incur higher latency than DC-DLA, but in such scenario the communication latency is not a performance limiter to begin with (Amdahl's law). We demonstrate in Section V that the impact of this latency overhead is negligible on system performance.

Software Interface. MC-DLA builds upon the memory-overlaying based DNN memory virtualization solutions [9], [10], which assume the following: (1) the high-level DL framework analyzes the neural network DAG structure at compile-time and derives the data-dependencies of memory-hungry DNN data, (2) this information is utilized by the runtime memory manager to schedule performance-aware, software-managed memory overlaying operations (i.e., DMA-initiated `cudaMemcpyAsync`) across host-device memory to expand the reach of memory available for training. The MC-DLA design introduces another tier of memory region in addition to the host and device memory – the capacity-optimized memory inside the memory-nodes, which we refer to as $\texttt{device}_{remote}$ memory in the rest of this paper. We propose to utilize $\texttt{device}_{remote}$ memory to supplant the role of host memory for stashing DNN data with long reuse distance. In other words, memory virtualization is implemented using the local device ($\texttt{device}_{local}$) memory and $\texttt{device}_{remote}$ memory without having the CPU memory involved. To allow the runtime memory manager to (de)allocate data structures inside

TABLE I: Software API extensions for MC-DLA.

API	Arguments	Semantics
cudaMallocRemote	&src, size	malloc size bytes to $\texttt{device}_{remote}$ memory and return ptr to src
cudaFreeRemote	&src	free memory that is allocated under $\texttt{device}_{remote}$ memory
cudaMemcpyAsync	&src, &dst, size, direction	copy size bytes from src to dst, but direction now includes LocalToRemote and RemoteToLocal

$\texttt{device}_{remote}$ memory and initiate DMA data transfers in/out of this memory region, we introduce three extensions to the CUDA runtime APIs (libcudart.so) for $\texttt{device}_{remote}$ memory (de)allocation and memory copy (Table I). Using these APIs, existing DL frameworks can seamlessly exploit the additional pool of memory inside our memory-nodes.

System Software Support. MC-DLA requires the device driver to be able to (de)allocate memory in $\texttt{device}_{remote}$ memory and be able to map that address space to user-level programs. Under our design, any given memory-node is logically partitioned into two groups and all the resources within a group (e.g., DMA engine, memory controller, and memory DIMMs) are exclusively assigned to a single device-node for servicing its requests (Figure 8). As these resources are not to be shared by any two device-nodes by design, the device driver manages both its client device-node and the each half of the left and right side memory-nodes's physical memory under a single device memory address space. Consequently, the $\texttt{device}_{local}$ physical memory lives at the bottom of this single device memory address space and each half of the two $\texttt{device}_{remote}$ physical memory is concatenated and mapped into the higher address space (Figure 10). From the driver's perspective, the device-node augmented with its share of memory-nodes can be thought of as a single PCIe device but with a larger memory capacity (e.g., Maxwell M40 containing 12 GB versus Volta V100 with 16 GB), hence existing system software APIs (e.g., mmap) can be used as-is to map the enlarged device memory address region to the user-level space. The current design of MC-DLA can add up to 1.3 TB $\times$ 8 = 10.4 TB of additional physical memory (Section V-C), well fitting under the addressing capabilities of current GPUs (e.g., 49-bit virtual addressing (512 TB) and 47-bit physical memory addressing (128 TB)) [40]. The added memory capacity to each device-node is informed to the device driver at boot-time so that the driver takes it into consideration when (de)allocating memory. Allocating pages in both $\texttt{device}_{local}$ and $\texttt{device}_{remote}$ memories can be done using existing device-side page-tables and the page-table walker, but our page allocation/placement policy is designed in a *bandwidth-aware* (BW_AWARE) manner in order to maximally exploit the high-bandwidth communication channels to the left and right memory nodes. Consider a cudaMallocRemote call with D Bytes of memory allocation requested to the driver. Rather than having the entire D Bytes of data be allocated under a single memory-node (which we refer to as LOCAL allocation policy[3]), our proposal splits the requested malloc size into two equal sized chunks (aligned in page granularity) and maps the pages within each chunk to the left and right memory-node's share of the memory address space in a round-robin fashion. This allows the device-node to utilize all N high-bandwidth links to read/write data from the two memory-nodes, maximally utilizing the N×B GB/sec of memory bandwidth for memory virtualization.

[3]The LOCAL allocation policy is named after the *local* NUMA zone page allocation policy of libNUMA in Linux and is not intended to imply that allocation is done inside $\texttt{device}_{local}$ memory.

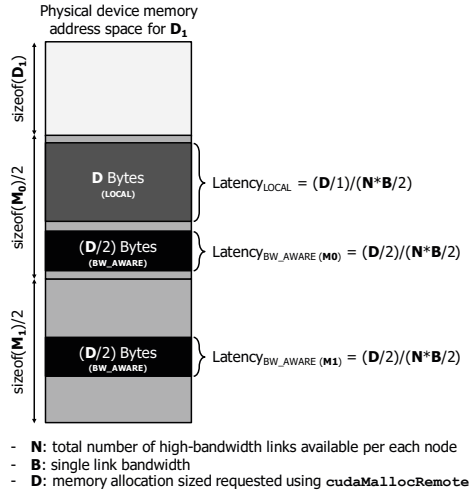

Fig. 10: The LOCAL and BW_AWARE page allocation policy employed in MC-DLA. BW_AWARE allows the device-node D_1 to read (write) data from (to) the left and right memory-nodes concurrently, reducing the overall latency by half compared to LOCAL.

IV. Methodology

Device and Memory Nodes. We developed an in-house system-level simulator for evaluating MC-DLA. The high-level architecture design of our DL accelerator resembles that of Eyeriss [41] or DaDianNao [42] in that our device architecture also employs a spatial array of processing elements (PEs), each of which contains (1) a multitude of MAC operators for handling vector operations, (2) local SRAM buffers (double-buffered to overlap computation with data fetches) to leverage data locality, and (3) a high-bandwidth on-package memory (e.g., HBM [8]) for local $\texttt{device}_{local}$ allocations. The baseline device-node has been configured as summarized in Table II but we also evaluate MC-DLA's sensitivity to alternative configurations in Section V-B. Our model is designed to optimize generic GEMM (general matrix multiplication) operations so that it handles not only convolutional layers, but also recurrent layers, fully-connected layers, activation layers, and etc. Based on our analysis, an output-stationary dataflow (i.e., output feature maps are stationed locally on-chip) as discussed by Chen et al. [41] provides a good balance in terms of MAC utilization and energy-efficiency across all of the layers we evaluate, hence our device accelerator employs the output-stationary dataflow rather than the row-stationary dataflow. It is worth pointing out that the scope of this paper is on studying HPC system architectures for DL training, rather than the development of a high-efficiency accelerator device. Therefore, our proposal is equally effective for alternative DL accelerator designs and DNN dataflows. Note that a single

TABLE II: Device-/memory-node configuration parameter.

Device-node	
Number of PEs	1024
MACs per PE	125
PE operating frequency	1 GHz
Local SRAM buffer size per PE	32 KB
Memory bandwidth	900 GB/sec
Memory access latency	100 cycles
Number of high-bandwidth links (N)	6
Communication bandwidth per link (B)	25 GB/sec
Memory-node	
Memory bandwidth	256 GB/sec
Memory access latency	100 cycles
Number of high-bandwidth links (N)	6
Communication bandwidth per link (B)	25 GB/sec

iteration of training can take hundreds of milliseconds even on a real high-end GPU card, so being able to perform simulation in tractable amount of time is crucial. We therefore model the device-node's HBM memory and the memory DIMMs inside the memory-nodes as having fixed memory bandwidth and latency, rather than resorting to a cycle-level DRAM simulator [43]–[45]. We believe our methodology provides accurate estimations without losing fidelity due to the following two reasons: (1) as DNN computation and memory accesses have high data locality with highly deterministic dataflow, existing designs [42], [46], [47] primarily employ a lightweight FSM or microcontrollers to orchestrate on and off-chip data movements in coarse-granular data sizes and (2) all inter-node (e.g., $\texttt{host-device}_{local}$, $\texttt{device}_{local}$-$\texttt{device}_{remote}$) data copy operations are conducted as coarse-grained, bulk data transfers using DMAs (Section II-B) with high data locality, rendering the system-level performance being less sensitive to the underlying behavior of the DRAM microarchitecture (e.g., bank conflicts).

System Architecture. We assume an 8 device-node system configuration in all of our experiments (Figure 1). The baseline DC-DLA system architecture is modeled after NVIDIA's DGX system [5] and IBM's PowerAI DDL system [30]. Both of these HPC systems employ the cube-mesh device-side interconnect, which is flattened into multiple ring networks (three rings in our evaluation, N=6 links per device node, Section III-B) to maximally utilize inter-node link bandwidth and minimize the latency incurred in conducting inter-device communications [34], [35]. DC-DLA uses PCIe (gen3) to communicate with the host memory for memory virtualization. The HC-DLA system architecture is modeled after IBM-NVIDIA's Power9 Summit [48], which assumes the following: (1) among the N high-bandwidth links available to each device-node, HC-DLA allocates half of them to be connected to the CPU memory for reads and writes, trading off fast memory virtualization over communication in a *balanced* manner, (2) four device-nodes are connected to a single CPU socket (i.e., 8 devices sharing two sockets), and (3) the maximum CPU socket memory bandwidth is large enough to fully service the aggregate CPU memory bandwidth usage of the four device-nodes that are connected to that CPU socket. Consequently, this hypothetical CPU in HC-DLA has 300 GB/sec of per socket CPU memory bandwidth (3× to 4× overprovisioned than real systems [38], [39]) which allows half of the N(=6) high-bandwidth links to be used to read/write CPU memory (i.e., 4×B×3 = 300 GB/sec). As HC-DLA can consume up to 100% of the provisioned CPU memory bandwidth (we discuss the maximum and average CPU memory bandwidth usage of all our system design points in Section V-A), such high CPU memory bandwidth usage could potentially incur destructive interference on CPU's role [49] in the overall DL training process (e.g., running the DL framework software, interacting with the backing storage HDD/SSD to fetch training data, etc), slowing down the overall training time. For a conservative evaluation, we assume that HC-DLA's CPU memory bandwidth usage has *no* effect on system performance. We omit the results of HC-DLA designs that partitions high-bandwidth links in an asymmetric manner as these design points were shown to be less robust than the studied, balanced HC-DLA. An *oracular* version of DC-DLA was also established by having an infinitely sized on-package, $\texttt{device}_{local}$ memory available inside each device-node, obviating the need for CPU-GPU data migration. We explore such (unbuildable) system design point to evaluate the effectiveness of MC-DLA. The memory-nodes in MC-DLA are configured to house ten DDR4 DIMMs providing a maximum of 256 GB/sec of memory bandwidth to the neighboring device-nodes.

TABLE III: Evaluated benchmarks.

Network	**Application**	**# of layers**
AlexNet	Image recognition	8
GoogLeNet	Image recognition	58
VGG-E	Image recognition	19
ResNet	Image recognition	34
Network	**Application**	**Timesteps**
RNN-GEMV	Speech recognition	50
RNN-LSTM-1	Machine translation	25
RNN-LSTM-2	Language modeling	25
RNN-GRU	Speech recognition	187

Benchmarks. We study a diverse set of eight DNN applications (Table III) that encompasses not only convolutional neural networks (CNNs) but also recurrent neural networks (RNNs). We choose four CNN topologies that show state-of-the-art performance in ImageNet [50], namely AlexNet, GoogLeNet, VGG-E, and ResNet. The four RNN applications have been chosen from Baidu's DeepBench application suite [23], which includes one GEMV-based vanilla RNN topology, two LSTM-based, and one GRU-based RNNs. We use a batch size of 512 for all our evaluations and study both data-parallel and model-parallel training (Figure 3) for partitioning the DL algorithm across the eight device-nodes. For model-parallel training, we employ the model-parallelization strategy as employed by Krizhevsky et al. [51].

Memory-overlaying for DNN Virtual Memory. We implemented the runtime memory management policy as described in [9], [10], [30], [52], which leverages the network DAG to analyze inter-layer data dependency to schedule memory-overlaying operations for virtual memory. Under our implementation, the device memory is utilized as an application-level cache with respect to the host memory. Concretely, the runtime memory manager pushes all layer's feature maps to the backing store after its last reuse during forward propaga-

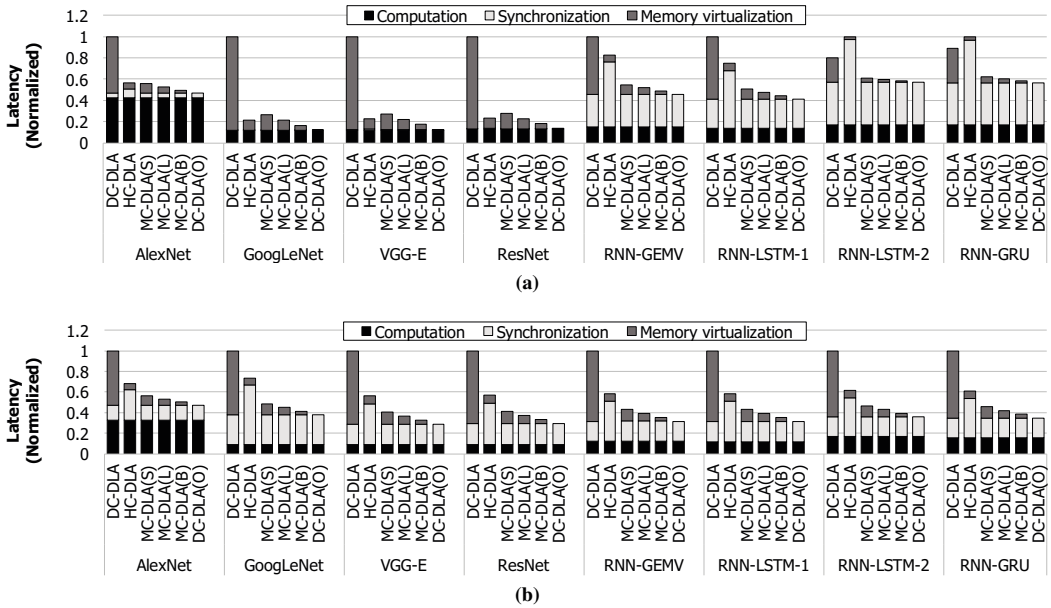

Fig. 11: Breakdown of latencies incurred during: (a) data-parallel training, (b) model-parallel training. Figures are normalized to the highest stacked bar chart. Note that the sum of these three latency categories does not directly translate into the system-level performance because DL frameworks try to overlap computation time with synchronization and memory virtualization.

tion and prefetches them back to the local device memory during backpropagation[4]. While some of these local↔remote data migration operations might not be necessary for DNNs that fit within the memory capacity limits, following prior work [9], [14], [52], [55], we employ such memory management policy to maximally stress the system interconnect. In other words, the 8 DNN applications we study are used as microbenchmarks to stress test the system interconnect and evaluate its robustness in providing a performant virtual memory system without compromising communication performance.

V. Evaluation

This section evaluates six system design points, the baseline `DC-DLA`, a hypothetical `HC-DLA` design (Section IV), one star-topology based `MC-DLA` (Figure 7(b), `MC-DLA(S)`), two ring-based `MC-DLA` design points (with `LOCAL` and `BW_AWARE` page allocation policy, denoted as `MC-DLA(L)` and `MC-DLA(B)`, respectively), and an oracular `DC-DLA` with infinite memory size (`DC-DLA(O)`). All average values are based on harmonic means.

A. Identifying System Bottlenecks

Convolutional layers are generally compute-limited (e.g., sliding window based dataflow manifests high data locality) and its feature maps, rather than weights, dominate memory allocation during training. Conversely, fully-connected layers and recurrent layers are memory bandwidth-limited where its weights take up a larger fraction of the memory allocation than feature maps. Consequently, data-parallel training of CNNs are generally insensitive to the underlying system's ability to provide fast inter-device communication because the synchronization data size (i.e., size of the weight gradients, `dW`) is relatively much smaller than its feature map size. Memory virtualization can therefore become a performance bottleneck for data-parallel training of CNNs. RNNs however have a relatively larger `dW` size for synchronization hence both fast communication and high-bandwidth memory virtualization is required for data-parallel RNN training. Model-parallel training, as discussed in Section II-C, incurs much frequent (and larger) synchronization operations than its data-parallel counterparts, so a high-bandwidth device-side interconnect is crucial for scalable DL training.

In this context, to clearly illustrate the system-level performance bottlenecks, we derive the latencies incurred in performing the (a) computations required for forward and backward propagation, (b) inter-device synchronization, and (c) memory-overlaying for memory virtualization, the three of which are stacked altogether in a single bar chart as shown in Figure 11. Overall, `DC-DLA` spends the least amount of time on synchronization thanks to its high-bandwidth device-side interconnection network. Memory virtualization however causes a significant performance bottleneck for `DC-DLA` on 14 out of the 16 training examples because the PCIe links only provide a small fraction of what the high-bandwidth links can service. Thanks to the high-bandwidth links allocated to access CPU memory, `HC-DLA` can significantly reduce the latency incurred in memory virtualization (average 88% reduction). This however comes at a cost where: (1) the smaller inter-device communication bandwidth now leads to an increase in synchronization time (average 90% increase), and (2) the

[4]We employ one exception to this rule: for layers that have short computation time (e.g., activation layers, pooling layers, ...), the memory manager chooses to *re-compute* the feature maps during backpropagation rather than migrating these data to the backing store. Such optimization minimizes the number of memory overlaying operations and is currently employed in MXNet [53], [54]. We adopt such optimization for a conservative evaluation and make sure the system performance is not unnecessarily degraded.

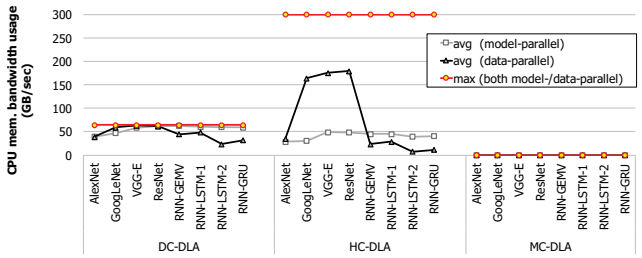

Fig. 12: CPU memory bandwidth usage under different DLA designs.

multiple devices in HC-DLA that are utilizing CPU memory for virtual memory are now consuming a significant fraction of CPU memory bandwidth as shown in Figure 12. For DL training, CPUs play the role of (1) running the DL framework software, and (2) getting the training datasets ready to be fed into the accelerator devices (e.g., reading and batching input datasets from the HDD/SSD storage devices, preprocessing the input batches, and uploading the preprocessed input batches to the CPU memory) [49]. While it is possible that such high host-device interaction can cause destructive interference and cause performance slowdown on HC-DLA, for a conservative analysis, we do not take such behavior into account when evaluating overall performance in Section V-B. The three MC-DLA designs are able to achieve the best of both DC-DLA and HC-DLA as the system interconnect successfully reduces the latency incurred in virtualizing memory while not having to incur noticeable overhead in inter-device communications. Additionally, because memory virtualization is provided using $\texttt{device}_{remote}$ memory, there are no CPU memory bandwidth consumption whatsoever, enabling a system design that scales independently to its ties with the host interface.

B. Performance

Figure 13 summarizes the performance of MC-DLA compared to DC-DLA, HC-DLA, and the oracular DC-DLA. The HC-DLA design provides an average 32% and 38% speedup over DC-DLA for data-parallel and model-parallel training, respectively. This is due to HC-DLA's ability to balance fast communication and memory virtualization, which DC-DLA fails in achieving due to its asymmetric partitioning of communication bandwidths (i.e., more than 10× difference in bandwidth provisioned for inter-device communication and memory virtualization). HC-DLA however is only able to leverage half of its high-bandwidth links for communication and virtual memory, failing to maximally benefit from the device-side interconnect. Our proposed MC-DLA(B) design fully unlocks the N high-bandwidth links for both communication and memory virtualization, leading to an average 3.5× and 2.1× speedup over DC-DLA for data-parallel and model-parallel training, respectively (average 2.8×). Moreover, MC-DLA(B) reaches 84%–99% of the performance of an unbuildable, oracular DC-DLA (average 95%). While MC-DLA(S) does much better than DC-DLA or HC-DLA, its suboptimal utilization of high-bandwidth links leaves significant performance left on the table (maximum 24%, average 14% performance loss than MC-DLA(B)). It is worth pointing

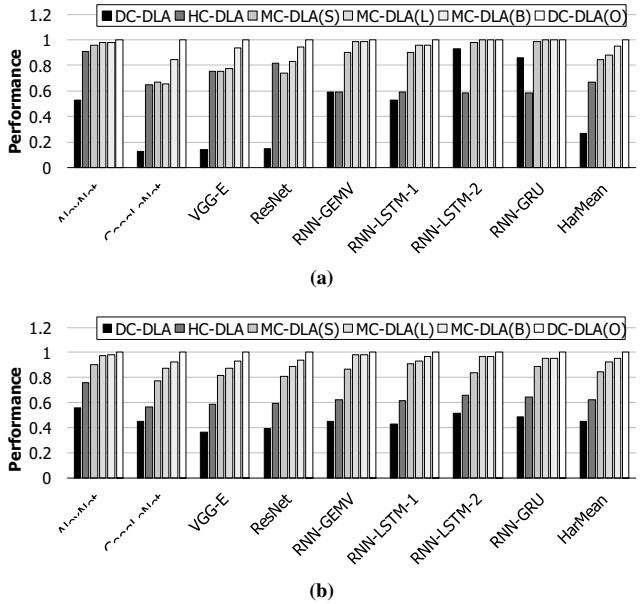

Fig. 13: Performance improvements offered by MC-DLA for: (a) data-parallel training, (b) model-parallel training.

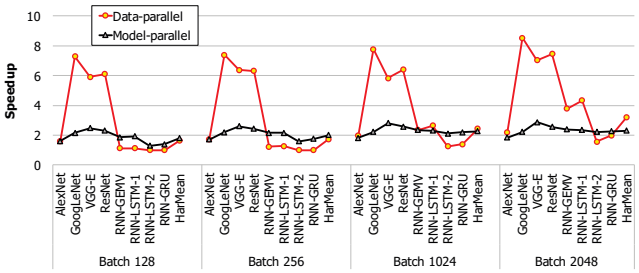

Fig. 14: MC-DLA performance sensitivity to input batch size.

out that the relatively sub-optimal, but simpler MC-DLA(L) design achieves 96% of the performance of MC-DLA(B). Although MC-DLA(L) is only provisioned with half the memory virtualization bandwidth of MC-DLA(B), the high-bandwidth communication channels for synchronization are equally provided for both designs thanks to its ring-based system interconnect. While MC-DLA(L) and MC-DLA(B) provides similar benefits over the 8 applications we study in this paper, we believe MC-DLA(B) to be a more robust and scalable design option as it can maximally utilize the interconnect bandwidth with reasonable design costs (Section III-B).

Sensitivity. Figure 14 shows MC-DLA(B)'s performance sensitivity to the input batch size, demonstrating MC-DLA's robustness (an average 2.17× speedup over DC-DLA across all batch sizes). We also studied DC-DLA with the next generation PCIe (gen4) which doubles the PCIe link bandwidth and improves DC-DLA's memory virtualization performance. Such design point improves DC-DLA's performance by 38%, narrowing the performance gap between DC-DLA and MC-DLA to 2.1× (as opposed to 2.8×), but comes at a cost of significant CPU memory bandwidth consumption (proportional to the increase in PCIe link bandwidth). System designs with (1)

TABLE IV: Memory-node power consumption (DDR4-2400).

Single DIMM		Memory-node	
DDR4 modules	**TDP (W)**	**TDP (W)**	**GB/W**
8 GB RDIMM [59]	2.9	29	2.8
16 GB RDIMM [60]	6.6	66	2.4
32 GB LRDIMM [61]	8.7	87	3.7
64 GB LRDIMM [62]	10.2	102	6.3
128 GB LRDIMM [63]	12.7	127	10.1

a faster device-node configuration such as TPUv2 and (2) scaled-up node configuration such as DGX-2 (2 PFLOPS and 2.4 TB/s of device-side interconnect bandwidth[5]) have also been explored which leads to MC-DLA with an average 3.2× and 2.9× speedup over DC-DLA, respectively. Rhu et al. [56] proposed to leverage CNN activation sparsity to compress and reduce local-device communication traffic to alleviate the PCIe bottleneck. This technique provides an average 2.6× reduction in PCIe traffic, narrowing the performance gap between DC-DLA and MC-DLA to 2.3× for the 4 CNN applications we study. Overall, MC-DLA exhibited robustness across various sensitivity studies we conducted (e.g., different chip configurations, input batch sizes, and etc.) as it guarantees high-performance memory virtualization and inter-device communication by design.

C. Power Efficiency

MC-DLA utilizes existing accelerators as-is, so the major power overhead comes from the memory-nodes added to the ring network. NVIDIA's DGX system (i.e., DC-DLA) has a TDP of 3,200 W, where the eight V100 GPUs consume 75% of the system power (i.e., 300 W × 8). Table IV summarizes our estimation of a single memory-node's power consumption using publicly available power measurements of DDR4 DIMMs [57] and Micron's DDR4 system power calculator [58]. For power-limited environments, memory-nodes with 8 GB RDIMM would be most appropriate which incurs an additional (29 × 8) = 232 W power consumption (7% increase over DGX-1V). For consumers more so focused on capacity expansion, the 128 GB LRDIMM based memory-node would provide high value (1.3 TB of memory under 127 W, highest GB/W). System-wide power consumption will increase by 31% (i.e., 127 × 8 = 1,016 W), but such option would drastically increase the pool of memory by 10.4 TBs. Microsoft's custom-built HGX-1 [6], a 4U server chassis featuring 8 Pascal GPUs, can have a TDP up to 9,600 W, so we believe the design overheads of MC-DLA is reasonable given its unique value proposition. Overall, MC-DLA achieves (2.8×/1.31)=2.1× to (2.8×/1.07)=2.6× increase in performance per watt while substantially enhancing the pool of memory exposed to the device-nodes.

D. Scalability

Although the image classification problem [11] is gradually gaining less traction from the DL algorithm community, there is still on-going research in parallelizing and distributing CNN training to 1000s of GPUs/TPUs to reduce training time and achieve performance scalability. Recent advances in this domain of research [64]–[67] employ data-parallel training with extremely large batch sizes (e.g., 32K in [65]) to reduce the intra-/inter-node communication overheads and achieve near perfect performance scaling. As the memory usage of these existing CNN algorithms are optimized to fit within the physical GPU memory constraints, training is done without any CPU-GPU data migration involved. Using our simulation infrastructure, we observe similar (perfect) performance scalability with DC-DLA when memory virtualization is disabled (i.e., close to 4× and 8× reduction in training time when the 4 CNN applications in Table III are data-parallelized across 4/8 GPUs). However, when memory virtualization is enabled and the feature maps are migrated in/out of local-remote memory, the performance improvements achieved with 4/8 GPUs under DC-DLA is only 1.3×/2.7× because of the host-device communication bottleneck[6]. Performance scalability is regained using MC-DLA thanks to its ability to perfectly hide data migration overhead (Figure 11).

E. User Productivity

As state-of-the-art CNN algorithms for *image* classification [20], [22] reach super-human performance, the DL research community has shifted towards more challenging tasks such as *video* understanding (e.g., video classification and captioning [25], [68], [69], video question and answering [26], [70], [71]). Given an input video stream, the goal is to capture the context of the scenes, objects, and activities and be able to express how these relate to each other in a complete sentence. State-of-the-art video understanding algorithms are commonly implemented as a mixture of CNNs, LSTMs, and memory networks [72], [73], but training these algorithms end-to-end under current HPC systems becomes practically impossible because of the memory capacity bottleneck. DL practitioners are therefore forced to compromise their learning algorithm (e.g., freezing subset of the algorithm without end-to-end training, reducing the number of input video frames and recurrent timesteps per training iteration, cropping video frame sizes, ...) so that the overall memory footprint fits within the physical GPU memory. With the advent of large-scale video training datasets such as YouTube-8M [74], providing sufficient amount of memory that enhances user productivity will become vital. Aside from being able to train DNNs that are deeper and larger, MC-DLA can open up a wider range of complex learning algorithms (e.g., end-to-end training of aforementioned video-to-text algorithms employing larger CNNs/LSTMs) that are currently impossible to train due to memory capacity limits, propelling continued innovation in this active research space.

[5]By provisioning even higher compute and communication bandwidth than the baseline system, the benefits of MC-DLA is even more pronounced as DC-DLA becomes completely bottlenecked by memory virtualization.

[6]Although host-device data migration for these CNN workloads is arguably unnecessary, following prior work [9], [14], [52], [55], we use existing workloads to study performance scalability as there are no publicly available DNN algorithms that exceed the memory capacity limits of current systems (i.e., you cannot train a DNN algorithm unless its memory requirement fits within the physical memory size limits, the chicken-and-egg problem).

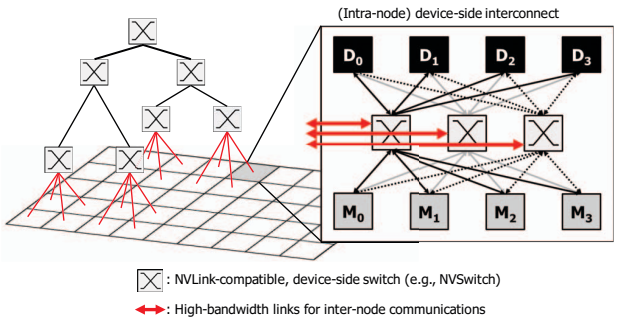

Fig. 15: Scale-out, datacenter-level device-side interconnect plane. Figure assumes that a given system node houses 8 nodes and each device-/memory-node is provided with N=3 high-bandwidth links. The device-side switch allows each of the 8 nodes to communicate with any of the other nodes inside the system node, enabling it to be casted into any interconnection topology (e.g., the ring-based MC-DLA interconnect, Figure 7(c)).

VI. Future Research Direction

To the best of our knowledge, our work is the first in the literature that highlights the growing significance of device-side interconnects in training DL algorithms across multiple devices. Due to space limitations and the wide design space, this paper focused on *intra*-node system architectural issues, assuming *inter*-node communication is handled using MPI over Ethernet/InfiniBand. NVIDIA recently announced the NVSwitch [75] technology which is an NVLINK-compatible switch, enabling system vendors opportunities to scale-up/out device-side interconnection networks, for instance (1) incorporating a larger number of GPUs within a system node [7] or (2) tightly integrating thousands of GPUs across hundreds of system nodes (Figure 15), similar to Microsoft's BrainWave [76]. The introduction of these device-side switching technologies for accelerators that enable scale-out device-side interconnects emphasizes the importance of device-side interconnection networks moving forward and opens up interesting research opportunities. Exploring our memory-node architecture as part of these scale-out device-side interconnects with 100s of device-nodes and memory-nodes across a distributed network is part of our next future work.

VII. Related Work

Memory disaggregation [17], [18] expands the CPU memory hierarchy to include a remote level provided by a separate memory blade connected over PCIe, which helps increase the pool of CPU accessible memory. Kim et al. [77], [78] proposed to interconnect multiple CPUs/GPUs by leveraging the packet routing capabilities of HMCs [79], effectively composing a memory network that provides flexible processor bandwidth utilization. The scope of [77], [78] is significantly different than what our work focuses on, but more importantly, our proposal is not tied with a particular memory technology whereas [77], [78] assumes a 3D stacked memory with routing capabilities embedded inside the logic layers. Slowdown on Moore's law and Denard scaling have driven researchers to pursue "chiplet" based processor designs [80]–[82], where a large SoC is decomposed into multiple smaller (but higher yield) chiplets and are re-assembled as a single package. One can envision combining the concept of chiplet-based GPUs with the notion of memory networks [77] as means to tightly integrate GPUs and HMCs within a package for memory capacity expansion. However, the maximum number of GPUs as well as HMCs that can be integrated inside a single package is bounded by various technology constraints, load distribution, and ease of programmability (e.g., recent MCM-GPU assumes only up-to 4 GPUs integrated within a single package). The focus of MC-DLA is on efficient parallelization and workload partitioning in a system-level context as opposed to these prior chiplet-context studies focusing on package-level or board-level integrations. A large body of prior work has explored the design of a single accelerator device architecture for deep learning inference [42], [46], [83]–[95] with an increased interest on leveraging DNN sparsity for further energy-efficiency improvements [47], [96]–[105]. Park et al. [106] proposed a scale-out acceleration platform for training machine learning algorithms using an FPGA-based 16-node distributed system. These prior studies are orthogonal to our MC-DLA proposal and can be adopted further for additional enhancements.

VIII. Conclusion

As the models and datasets to train DL models scale, system vendors are employing a custom device-side interconnection network for fast communication and synchronization across accelerator devices. This paper is the first to describe the growing significance of device-side interconnects for training scaled up DL algorithms and highlights the importance of balancing inter-device communication and fast memory virtualization. We make a case for a memory-centric DL system and presented a scalable, programmable, and energy-efficient HPC platform for DL training, which provides an average 2.8× speedup over DC-DLA while drastically expanding the pool of memory accessible to accelerators to 10s of TBs.

Acknowledgments

This work was supported by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-TB1703-03.

References

[1] NVIDIA, "NVIDIA Tesla V100," 2017.
[2] Google, "Cloud TPUs: ML accelerators for TensorFlow," 2017.
[3] Intel-Nervana, "Intel Nervana Hardware: Lake Crest," 2017.
[4] NVIDIA, "NVLINK High-Speed Interconnect," 2016.
[5] NVIDIA, "The NVIDIA DGX-1V Deep Learning System," 2017.
[6] Microsoft, "Microsoft Project Olympus Hyperscale GPU Accelerator (HGX-1)," 2017.
[7] NVIDIA, "The NVIDIA DGX-2 Deep Learning System," 2017.
[8] JEDEC, "High Bandwidth Memory (HBM2) DRAM," 2018.
[9] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, October 2016.
[10] IBM, "Realizing the value of Large Model Support (LMS) with PowerAI IBM Caffe," 2018.
[11] H. Zhu, B. Zheng, A. Phanishayee, B. Schroeder, and G. Pekhimenko, "DNN-Train: Benchmarking and Analyzing DNN Training," in *SysML*, February 2018.

[12] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Phanishayee, B. Schroeder, and G. Pekhimenko, "TBD: Benchmarking and Analyzing Deep Neural Network Training," in *arxiv.org*, 2018.

[13] C. Meng, M. Sun, J. Yang, M. Qiu, and Y. Gu, "Training Deeper Models by GPU Memory Optimization on TensorFlow," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, December 2017.

[14] J. Park, W. Jung, H. Cho, and J. Lee, "Transparent GPU Memory Management for DNNs," in *Proceedings of the Symposium on Principles and Practice of Parallel Programming (PPOPP)*, February 2018.

[15] IBM, "Chainer: Out-of-core training," 2017.

[16] Google, "Tensorflow: Memory-optimizer," 2017.

[17] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated Memory for Expansion and Sharing in Blade Servers," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2009.

[18] K. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, "System-level Implications of Disaggregated Memory," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2012.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, November 1998.

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

[21] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger, "Deep Networks with Stochastic Depth," https://arxiv.org/abs/1603.09382, 2016.

[22] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2017.

[23] Baidu, "DeepBench: Benchmarking Deep Learning Operations on Different Hardware," https://github.com/baidu-research/DeepBench, 2017.

[24] J. Kim, Y. Park, G. Kim, and S. J. Hwang, "SplitNet: Learning to Semantically Split Deep Networks for Parameter Reduction and Model Parallelization," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[25] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to Sequence - Video to Text," in *Proceedings of the International Conference on Computer Vision (ICCV)*, October 2015.

[26] S. Na, S. Lee, J. Kim, and G. Kim, "A Read-Write Memory Network for Movie Story Understanding," in *Proceedings of the International Conference on Computer Vision (ICCV)*, October 2017.

[27] T. Zheng, D. Nellans, A. Zulfiqar, M. Stephenson, and S. W. Keckler, "Toward High-Performance Paged-Memory for GPUs," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, March 2016.

[28] J. Power, M. Hill, and D. Wood, "Supporting x86-64 Address Translation for 100s of GPU Lanes," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, February 2014.

[29] B. Pichai, L. Hsu, and A. Bhattacharjee, "Architectural Support for Address Translation on GPUs: Designing Memory Management Units for CPU/GPUs with Unified Address Spaces," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)*, March 2014.

[30] M. Cho, U. Finkler, S. Kumar, D. Kung, V. Saxena, and D. Sreedhar, "PowerAI Distributed Deep Learning (DDL)," in *arxiv.org*, 2017.

[31] NVIDIA, "NVIDIA DGX-1 System Architecture: The Fastest Platform for Deep Learning," 2017.

[32] A. Krizhevsky, "One Weird Trick For Parallelizing Convolutional Neural Networks," https://arxiv.org/abs/1404.5997, 2014.

[33] J. Dean, "Machine Learning for Systems and Systems for Machine Learning," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, December 2017.

[34] E. Chan, R. van de Geijn, W. Gropp, and R. Thakur, "Collective Communication on Architectures that Support Simultaneous Communication Over Multiple Links," in *Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 2006.

[35] C. Woolley, "NCCL: Accelerated Multi-GPU Collective Communications," 2015.

[36] NVIDIA, "NVIDIA Collective Communications Library (NCCL)," 2017.

[37] Baidu, "Bringing HPC Techniques to Deep Learning," 2017.

[38] Intel, "Intel Xeon Processors," 2017.

[39] IBM, "IBM Power9 Microprocessor," 2017.

[40] NVIDIA, "NVIDIA Tesla P100," 2016.

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

[42] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2014.

[43] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," 2011.

[44] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti, "USIMM: the Utah SImulated Memory Module," 2012.

[45] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," 2015.

[46] Y. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *Proceedings of the International Solid State Circuits Conference (ISSCC)*, February 2016.

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

[48] R. Smith, "NVIDIA Volta, IBM Power9 Land Contracts For New US Government Supercomputers," 2014.

[49] NVIDIA, "Meet AIRI: AI-Ready Infrastructure That Scales for Every Enterprise," 2018.

[50] ImageNet, "ImageNet dataset," www.image-net.org, 2016.

[51] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the International Conference on Neural Information Processing Systems (NIPS)*, December 2012.

[52] M. Cho, T. Le, U. Finkler, H. Imai, Y. Negishi, T. Sekiyama, S. Vinod, V. Zolotov, K. Kawachiya, D. Kung, and H. Hunter, "Large Model Support for Deep Learning in Caffe and Chainer," in *SysML*, February 2018.

[53] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems," in *Proceedings of the Workshop on Machine Learning Systems*, December 2015.

[54] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training Deep Nets with Sublinear Memory Cost," *arXiv preprint*, 2016.

[55] NikolaymSakharnykh, "Unified Memory on Pascal and Volta," 2017.

[56] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, "Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, February 2018.

[57] T. Hardware, "Measuring DDR4 Power Consumption," 2015.

[58] Micron, "Micron: System Power Calculator (DDR4)," 2017.

[59] Samsung, "(8GB, 1Gx72 Module) 288pin Registered DIMM based on 4Gb E-die," 2016.

[60] Samsung, "(16GB, 2Gx72 Module) 288pin Registered DIMM based on 4Gb E-die," 2016.

[61] Samsung, "(32GB, 4Gx72 Module) 288pin Load Reduced DIMM based on 8Gb B-die," 2016.

[62] Samsung, "(64GB, 8Gx72 Module) 288pin Load Reduced DIMM based on 8Gb B-die," 2016.

[63] Samsung, "(128GB, 16Gx72 Module) 288pin Load Reduced DIMM based on 8Gb B-die," 2017.

[64] P. Goyal, P. Dollar, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," in *arxiv.org*, 2017.

[65] T. Akiba, S. Suzuki, and K. Fukuda, "Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes," in *arxiv.org*, 2017.

[66] Y. You, Z. Zhang, C. Hsieh, J. Demmel, and K. Keutzer, "ImageNet Training in Minutes," in *arxiv.org*, 2018.

[67] S. Sridharan, K. Vaidyanathan, D. Kalamkar, D. Das, M. Smorkalov, M. Shiryaev, D. Mudigere, N. Mellempudi, S. Avancha, B. Kaul, and P. Dubey, "On Scale-out Deep Learning Training for Cloud and HPC," in *SysML*, February 2018.

[68] Z. Wu and T. Yao and Y. Fu and Y. Jiang, "Deep Learning for Video Classification and Captioning," in *arxiv.org*, 2018.

[69] H. Yu, J. Wang, Z. Huang, Y. Yang, and W. Xu, "Video Paragraph Captioning Using Hierarchical Recurrent Neural Networks," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[70] M. Tapaswi, Y. Zhu, R. Stiefelhagen, A. Torralba, R. Urtasun, and S. Fidler, "MovieQA: Understanding Stories in Movies through Question-Answering," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[71] K. Zeng, T. Chen, C. Chuang, Y. Liao, J. Niebles, and M. Sun, "Leveraging Video Descriptions to Learn Video Question Answering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[72] A. Graves and G. Wayne and I. Danihelka, "Neural Turing Machines," in *arxiv.org*, 2014.

[73] C. Gulcehre and S. Chandar and K. Cho and Y. Bengio, "Dynamic Neural Turing Machine with Soft and Hard Addressing Schemes," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[74] Google, "YouTube-8M Dataset," https://research.google.com/youtube8m/, 2018.

[75] NVIDIA, "NVSwitch: Leveraging NVLink to Maximum Effect," 2018.

[76] Microsoft, "Microsoft unveils Project Brainwave for real-time AI," 2017.

[77] G. Kim, J. Kim, J. H. Ahn, and J. Kim, "Memory-centric System Interconnect Design with Hybrid Memory Cubes," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, September 2013.

[78] G. Kim, M. Lee, J. Jeong, and J. Kim, "Multi-GPU System Design with Memory Networks," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2014.

[79] J. Jeddeloh and B. Keeth, "Hybrid Memory Cube: New DRAM architecture increases density and performance," in *VLSI Technology (VLSIT), 2012 Symposium on*. IEEE, 2012.

[80] A. Kannan, N. Jerger, and G. Loh, "Enabling Interposer-based Disintegration of Multi-core Processors," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2015.

[81] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C. Wu, and D. Nellans, "MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2017.

[82] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. Altaf, N. Jerger, and G. Loh, "Modular Routing Design for Chiplet-based Systems," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2018.

[83] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)*, March 2014.

[84] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2015.

[85] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Temam, X. Feng, X. Zhou, and Y. Chen, "PuDianNao: A Polyvalent Machine Learning Accelerator," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)*, April 2015.

[86] Z. Du, D. Rubin, Y. Chen, L. He, T. Chen, L. Zhang, C. Wu, and O. Temam, "Neuromorphic Accelerators: A Comparison Between Neuroscience and Machine-Learning Approaches," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2015.

[87] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. Lee, J. Miguel, H. Lobato, G. Wei, and D. Brooks, "Minerva: Enabling Low-Power, High-Accuracy Deep Neural Network Accelerators," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2016.

[88] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2016.

[89] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An Instruction Set Architecture for Neural Networks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2016.

[90] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2016.

[91] D. Mahajan, J. Park, E. Amaro, H. Sharma, A. Yazdan-bakhsh, J. K. Kim, and H. Esmaeilzadeh, "TABLA: A unified Template-based Framework for Accelerating Statistical Machine Learning," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, February 2016.

[92] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Misra, and H. Esmaeilzadeh, "From High-level Deep Neural Models to FPGAs," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2016.

[93] D. Moss, E. Nurvitadhi, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and P. Leong, "High Performance Binary Neural Networks on the Xeon+FPGA Platform," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2017.

[94] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)*, 2017.

[95] D. Moss, S. Krishnan, E. Nurvitadhi, P. Ratuszniak, C. Johnson, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and P. Leong, "A Customizable Matrix Multiplication Framework for the Intel HARPv2 Xeon+FPGA Platform: A Deep Learning Case Study," in *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2018.

[96] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2016.

[97] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-Neuron-Free Deep Convolutional Neural Network Computing," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2016.

[98] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-X: An Accelerator for Sparse Neural Networks," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, October 2016.

[99] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, and A. Moshovos, "Stripes: Bit-serial Deep Neural Network Computing," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, October 2016.

[100] J. Albericio, A. Delmas, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic Deep Neural Network Computing," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, October 2017.

[101] G. Venkatesh, E. Nurvitadhi, and D. Marr, "Accelerating Deep Convolutional Networks using Low-precision and Sparsity," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[102] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong, Y. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh, "Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?" in *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.

[103] P. Whatmough, S. Lee, H. Lee, S. Rama, D. Brooks, and G. Wei, "A 28nm SoC with a 1.2 GHz 568nJ/Prediction Sparse Deep-Neural-Network Engine with >0.1 Timing Error Rate Tolerance for IoT Applications," in *Proceedings of the International Solid State Circuits Conference (ISSCC)*, February 2017.

[104] P. Whatmough, S. Lee, N. Mulholland, P. Hansen, S. Kodali, D. Brooks, and G. Wei, "DNN ENGINE: A 16nm Sub-uJ Deep Neural Network Inference Accelerator for the Embedded Masses," in *Hot Chips: A Symposium on High Performance Chips*, August 2017.

[105] A. Delmas, P. Judd, D. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, and A. Moshovos, "Bit-Tactical: Exploiting Ineffectual Computations in Convolutional Neural Networks: Which, Why, and How," https://arxiv.org/abs/1803.03688, 2018.

[106] J. Park, H. Sharma, D. Mahajan, J. K. Kim, P. Olds, and H. Esmaeilzadeh, "Scale-Out Acceleration for Machine Learning," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2017.