2018 IEEE 25th International Conference on High Performance Computing (HiPC)

OC-DNN: Exploiting Advanced Unified Memory Capabilities in CUDA 9 and Volta GPUs for Out-of-Core DNN Training

Ammar Ahmad Awan, Ching-Hsiang Chu, Hari Subramoni, Xiaoyi Lu, and Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering
The Ohio State University
{awan.10, chu.368, subramoni.1, lu.932, panda.2}@osu.edu

***Abstract*—Existing frameworks cannot train large DNNs that do not fit the GPU memory without explicit memory management schemes. In this paper, we propose *OC-DNN* - a novel Out-of-Core DNN training framework that exploits new Unified Memory features along with new hardware mechanisms in Pascal and Volta GPUs. *OC-DNN* has two major design components — 1) *OC-Caffe*; an enhanced version of Caffe that exploits innovative UM features like asynchronous prefetching, managed page-migration, exploitation of GPU-based page faults, and the cudaMemAdvise interface to enable efficient out-of-core training for very large DNNs, and 2) an interception library to transparently leverage these cutting-edge features for other frameworks. We provide a comprehensive performance characterization of our designs. OC-Caffe provides comparable performance (to Caffe) for regular DNNs. OC-Caffe-Opt is up to 1.9X faster than OC-Caffe-Naive and up to 5X faster than optimized CPU-based training for out-of-core workloads. OC-Caffe also allows scale-up (DGX-1) and scale-out on multi-GPU clusters.**

I. Introduction

The recent success of Deep Neural Networks (DNNs) and their contribution to the advancement of Artificial Intelligence (AI) is significant. DNNs have found widespread adoption for several application areas like Image Recognition, Text Analysis, Speech Processing, and Cancer Detection [22]. This has led to the rise of several Deep Learning (DL) frameworks and DNN architectures (models) primarily fueled by the availability of high-performance GPU hardware and versatile training datasets like ImageNet [4].

Earlier DNN architectures like AlexNet [11] achieved excellent training speedups on the GPUs but they were only comprised of a few convolution layers. However, recent incarnations of DNNs like GoogLeNet [21], VGG [20], and ResNet-50 [7] consist of several computational layers to achieve higher accuracy. This trend has led to a significant increase in the amount of memory required to stage and process the data on the GPU. Thus, DNN training on GPUs has become an inherently memory-bound process. We note that both Pascal and Volta GPUs only have limited memory (compared to the main memory of the host system). Thus, GPU-based training for DNNs that either consists of several layers or use a larger batch size becomes impossible without complex memory management mechanisms.

This scenario, where the data does not fit the available GPU memory is referred to as "Out-of-Core". Conversely, we refer to the models and batch sizes that fit a GPU's memory as "trainable". To deal with out-of-core DNN training on GPUs in an efficient and portable manner, we investigate different design choices for GPU-based frameworks. One can develop virtualized DNNs (vDNN) [18] that use explicit data movement of GPU buffers using the large CPU memory to deal with out-of-core workloads. vDNN may offer good performance but requires significant manual effort to modify applications thereby making it a "low-productivity" and "low-portability" solution. The other design choice is to utilize the newly introduced *Unified Memory (UM)* [6] features[1] in the CUDA programming model. However, a straight forward application of UM does not suffice as there are several challenges brought forward by complex DNN architectures. Thus, we investigate efficient schemes to exploit the cutting-edge hardware support on the Pascal/Volta GPUs along with the new cudaMemAdvise interface to guide the CUDA driver and develop a much better, more portable, yet a high-performance solution. In this work, we address the following broad challenge — ***Can a high-performance, high-productivity, and memory-efficient Deep Learning framework be designed such that it allows efficient out-of-core DNN training without compromising performance for trainable DNNs?***

A. Motivation

DNN architectures, presented in Table I, are state-of-the-art models used by many in the DL community. The table offers a perspective on the "trainability" of these DNNs for the Caffe framework. For example, a typical batch size of 40 for the ResNet-50 model fits under the 16 GB memory of a Pascal/Volta GPU, but anything higher than that will result in an "out-of-memory" error. Several other prevalent DNNs have been designed to use many layers in order to improve the accuracy of the model. However, it can lead to poor scalability because of the synchronization overhead [2], [8]. On the other hand, reducing the number of training parameters can negatively affect accuracy, which is dealt with by adding more layers and making more complex DNN architectures. These issues eventually lead to a DNN that cannot be made any smaller and thus will hit the fundamental GPU memory limit. ***Hence, over-subscription of the GPU memory and dealing with out-of-core DNNs becomes a need and not a choice, even for parallel and distributed training frameworks***.

In order to deal with out-of-core DNNs, the DL community needs to address the following concrete challenges:

[1]Note that the terms "Managed" and "Unified" are used interchangeably throughout the paper to refer to the new Unified-Memory features available since CUDA 8.

978-1-5386-8386-6/18/$31.00 ©2018 IEEE
DOI 10.1109/HiPC.2018.00024

TABLE I
TRAINABILITY OF VARIOUS DNN ARCHITECTURES WITH CAFFE

DNN Architecture	Max. Trainable Batch Size on 16 GB Volta (No. of Samples)
AlexNet [11]	1,100
GoogLeNet [21]	200
VGG-19 [20]	110
ResNet-50 [7]	40

- What are the fundamental bottlenecks in performing out-of-core DNN training on GPUs and what are the essential primitives involved in such applications?
- Can we exploit new CUDA Unified Memory primitives for out-of-core DNNs yet avoid performance degradation for regular DNNs?
- Will CUDA UM interface be helpful in simplifying DL framework design and offer high-productivity solutions for frameworks like Caffe?
- How can we exploit transparent mechanisms like interception to offer out-of-core solutions via Unified Memory for existing DL frameworks without explicit redesign?
- Can out-of-core DNN training frameworks also provide scale-up and scale-out for distributed training on multiple GPUs?

B. Contributions

The design space for DNN training frameworks is becoming increasingly complex and heterogeneous but the fundamental operations in all training frameworks remain the same. We choose Caffe as the candidate framework to investigate the challenges brought forward by "out-of-core" DNNs but we offer generic guidelines that can be used to design other DL frameworks as well. To the best of our knowledge, this is the first attempt to deal with memory-bound out-of-core DNN training on GPUs by exploiting the new CUDA Unified Memory interface in tandem with automatic page migration and prefetching capabilities of Pascal/Volta GPUs.

We make the following key contributions in this paper:

- Propose and design an integrated OC-DNN framework that provides efficient out-of-core DNN training on a single GPU by exploiting managed-memory primitives.
- Design and Implement OC-Caffe, which provides comparable performance to Caffe for regular DNNs, and up to 1.9X speedup over straightforward implementations for out-of-core workloads.
- Evaluate performance benefits achieved by OC-DNN (Interception Library with Caffe) and OC-Caffe for training state-of-the-art DNNs like AlexNet, GoogLeNet, ResNet-50, and VGG-19 for out-of-core batch sizes on the latest Volta GPUs.
- Modify Caffe to exploit Unified Memory allocations to offer a simplified memory management design that results in the removal of about 3,000 lines of error-prone code.
- Extend and Evaluate OC-Caffe to support distributed training on multiple GPUs both within and across nodes on DGX-1 [14] and a multi-GPU cluster.

II. DESIGN SPACE FOR EXISTING DL FRAMEWORKS

Table II positions our proposed work in the rich space of existing tools and frameworks. We highlight the key features and cutting-edge technologies that we exploit in our proposed OC-DNN framework and how these differ from the existing works. Park et al. proposed software-based GPU memory swapping and memory object sectioning mechanisms to transparently handle out-of-core training for the VGG network using Caffe in [17]. Extension of cuDNN to enable out-of-core training with CNNs via model division and pipelined processing is presented in [9]. Minsoo et al. [18] present their designs to handle out-of-core models like VGG-416. The authors propose the concept of virtualized DNNs, but the designs presented in this study do not take advantage of any of the CUDA UM features like GPU page migration or managed allocations. In fact, our OC-DNN framework can complement vDNN because the primary principle of the vDNN study is the reduction of required memory while our designs deal with out-of-core workloads. Other frameworks like S-Caffe [2] exploit data-parallelism to deal with large DNNs but do not offer single-GPU "out-of-core" training support.

There are no thorough performance studies available for the CUDA 8/9 Unified Memory primitives. Sakharnykh presents fundamentals of Unified Memory for NVIDIA GPUs in [19]. In [12], the authors presented their findings of significant overhead caused by CUDA 6 managed allocations and reported that most use-cases did not perform well. In contrast, we only observed minor degradation in OC-Caffe through efficient exploitation of *cudaMemPrefetch* and *cudaMemAdvise* operations.

III. BACKGROUND

We provide a brief overview of the DNN training process in Caffe [10], new features in CUDA 8/9 that can be exploited for out-of-core workloads, and cutting-edge hardware including Pascal/Volta GPUs and the NVIDIA DGX-1.

A. DNN Training

Although there are several variations in the organization and architecture of DNNs, many system-level aspects are common for various DNNs. DNN training happens in two main phases: the forward and the backward propagation phases. The notion of *layers* is generic, but some frameworks like Caffe have realized them as real C++ objects that abstract out the complexity of operations performed in a layer. The forward pass comprises many such layers and results in a loss/error term. The loss is then back-propagated in the backward phase when the gradients are calculated and accumulated. In the context of communication, both the forward and backward passes involve data movement (memory copies) that can either use *cudaMempcy* variants or a specific CUDA kernel that performs a math operation (e.g., GEMM, Convolution, etc.) in addition to the copy operation.

TABLE II
PRODUCTIVITY AND PERFORMANCE FEATURES FOR EXISTING AND PROPOSED DNN TRAINING FRAMEWORKS

Related Studies	Communication Primitives		Productivity Features		Performance Features	
	Device Primitives	Managed Primitives	Unified Memory Allocation	Managed Data Movement	Prefetching	Advise / Hints
vDNN [18]	✔	×	×	×	✔	×
S-Caffe [2]	✔	×	×	×	×	×
CNTK [13]	✔	×	×	×	×	×
Caffe [10]	✔	×	×	×	×	×
Proposed (This paper)	✔	✔	✔	✔	✔	✔

B. CUDA Unified Memory and Pascal/Volta GPUs

Compute Unified Device Architecture (CUDA) is the default programming model for NVIDIA GPUs. New features and enhancements like the over-subscription of the GPU memory using Unified Memory (UM) buffers, GPU page-faults, and CUDA-driver based automatic page migration engine were introduced with CUDA 8. UM was originally introduced in CUDA 6 but without over-subscription support. The notion of unlimited GPU-accessible memory (limited to CPU memory), GPU page faults, page migration engine, prefetching capability to avoid page faults, and the ability to provide hints to the CUDA driver are all new features that got introduced with CUDA 8 and further improved in CUDA 9. However, CUDA 8/9 Unified Memory features that we exploit in this study only work with the latest Pascal and Volta GPUs. Many advanced hardware features including the support for fine-grain page faults and over-subscription through UM interface were introduced by NVIDIA in the Pascal architecture. Volta GPUs offer even better performance with enhanced support for UM [5] and NVLINK 2.0 [16].

IV. REALIZING OUT-OF-CORE TRAINING ON GPUS

We now discuss the fundamental challenges that need to be understood and solved in order to deal with "out-of-core" DNN training on GPUs.

A. Can we decompose DNN training operations into fundamental CUDA-level primitives?

Characterizing how a DNN training framework realizes the computation and communication on the GPU hardware is essential to understanding the limitations of existing designs. Furthermore, it is important to identify the building blocks or the primitive operations involved in training for both "trainable" and "out-of-core" DNNs. The main question is can we breakdown DNN training phases like the Forward pass and the Backward pass into fine-grain primitive operations?

B. How to deal with large amount of training data?

Reading training samples (data) is a fundamental operation for training a DNN. This data has to be first read to the host buffers allocated on the CPU memory. This becomes a memory usage challenge because we essentially need to allocate twice the amount of actual data. For instance, if the batch of training samples for a specific dataset like ImageNet requires one Gigabyte (GB) memory, we need to allocate one GB on the CPU memory for the staging area and another GB on the GPU memory for the actual training on the GPU. In addition, we need to take into account explicit Host to Device (*H2D*) copies of this one GB of data. A staging design is inevitable if separate Device and Host buffers are used. Thus, the question is if Unified Memory buffers are used, can we improve performance and/or can we simplify the design of the DL framework without loss in performance?

C. How to efficiently tackle intra-GPU communication in DNN training?

The layer-wise design of DNN training demands for maintaining the entire model and its associated buffers in the GPU memory. Irrespective of whether training is performed on the CPU or a GPU, the DNN training proceeds in a layer-wise fashion. Moreover, the entire model is kept in the GPU memory to avoid expensive Host to Device and Device to Host data movement operations during the training process. This process inherently involves several copies (or communication) between layers. Some of these issues have been discussed at length in [18]. The blocking nature of *cudaMalloc* and *cudaFree* calls also adds to the overhead if the buffers are allocated and de-allocated during the training to save the GPU memory. This so called intra-GPU communication is implemented using *cudaMemcpy* calls on source and destination buffers. Our performance and profiling analysis suggests that such cudaMemcpy operations are heavily utilized for layered GPU-based training of DNNs. The key question is how to deal with this intra-GPU communication when the DNNs do not fit the physical GPU memory?

D. What are the alternatives for out-of-core training?

The community has proposed other approaches to deal with large DNNs. Authors of [18] have focused on reducing the memory consumption by creating staging areas in the host memory and applying heuristics to allocate/de-allocate judiciously. Other studies [2], [8] have offered orthogonal solutions that utilize multiple GPUs in order to deal with large batch sizes and/or large DNNs. Both approaches have their limitations. The first approach requires complicated mechanisms and application level changes that may not be portable where heuristics are used to prefetch and evict the data. Essentially, this approach will result in overlapped/hidden communication between CPU and GPU memory. On the other hand, the parallelization approach requires additional GPUs to divide the work. Furthermore, parallelization has other challenges to deal with like large-message reduction overhead for accumulating gradients via Message Passing Interface (MPI) as well as issues like slower convergence if asynchronous

parallelism is used. The main challenge is to investigate if a framework can be designed in a way such that it allows "out-of-core" training on a single GPU using a simple and portable design as well as offer distributed training on multiple GPUs.

V. Proposed Design: OC-DNN Framework

We now discuss how we have addressed the challenges discussed in Section IV under the proposed OC-DNN framework. Figure 1 provides a high level overview of the OC-DNN framework. We discuss the classification and exploitation of primitive operations followed by the description of the proposed Interception Library (IL) that can be used with any CUDA-based DL framework.

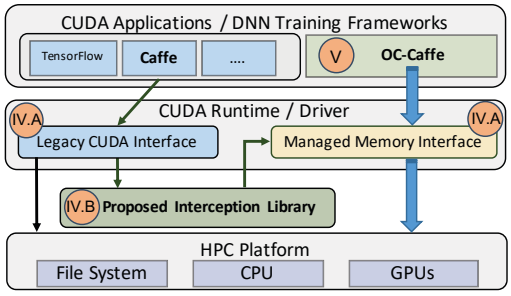


Fig. 1. Proposed OC-DNN Framework for Out-of-Core Training on GPUs

A. Categorization and Exploitation of Primitives for Out-of-Core DNN Training on GPUs

Based on our in-depth analysis of the different DNN training frameworks, we classify primitive operations involved in DNN training as 1) File access operations and 2) CUDA operations. We further divide file access operations into two primitives:

1) File System to Host (CPU) memory (*F2H*) operation
2) Host memory to File System (*H2F*) operation

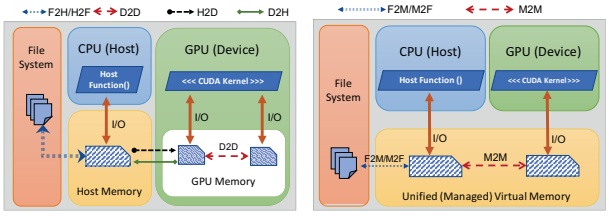
Similarly, we divide CUDA operations into the following:

1) Host to Device (*H2D*) copy for moving the training samples from the CPU memory to the GPU memory
2) Device to Device (*D2D*) copy to transfer data between the layers of a DNN residing on the GPU memory
3) CUDA *Kernels* to perform operations on a layer's data and pass it on to the next layers
4) Device to Host (*D2H*) memory copies to write back the trained DNN from the GPU memory to the CPU memory so that it can be persisted to permanent storage media.

An illustration of how these CUDA-level primitives are used for DNN training is provided in Figure 2a. However, these primitives are not sufficient to design out-of-core training as they are built around the underlying assumption that all the data is resident in the GPU memory (explicit device allocations). To deal with out-of-core DNNs, we develop approaches where the new unified memory based primitives can be utilized to offer efficient out-of-core DNN training on the GPUs. We propose the following primitives:

1) File System to Managed Memory (*F2M*) primitive to replace *F2H* + *H2D* operation
2) *M2M* transfers on the managed buffers to replace *D2D* transfers
3) Managed Memory to File System (*M2F*) primitive to replace *D2H* + *H2F* operation.

To summarize, we proposed three new primitives for managed out-of-core DNN training — *F2M*, *M2F*, and *M2M*. Figure 2b illustrates the overview of these new primitives and how they differ from the existing *F2H*, *H2D*, *D2D*, *D2H*, and *H2F* primitives shown in Figure 2a. Figure 2b also highlights how the proposed primitives allow CUDA kernels to directly access the managed buffers without the need for an explicit communication as data migration is handled by the CUDA driver in a transparent fashion.



(a) Existing Primitives: Explicit CPU and GPU Allocation (*malloc, cudaMalloc, and cudaMallocHost*)

(b) Proposed Primitives: Unified Memory Allocations (*cudaMalloc-Managed*)

Fig. 2. Comparison of Existing and Proposed Primitives for DNN Training

B. Interception Library (IL) for Rapid Experimentation

To allow rapid experimentation and enable the use of managed primitives without explicitly modifying DL applications, we propose an Interception Library (IL) as illustrated in Figure 1. With this independent library, any existing application (that uses *cudaMalloc*) can transparently leverage the proposed Out-of-core design schemes without any changes to the code. We note that NVIDIA has announced plans to make managed allocations even easier using regular *malloc* calls through modifications in the Linux kernel. However, legacy applications will still need some interception or modification to take advantage of the current managed-memory interface.

The Interception Library (IL) design exploits the *LD_PRELOAD* facility available to any application that uses dynamically loaded libraries. To allow out-of-core DNN training, we intercept CUDA memory allocation and management calls and redefine their behavior inside the IL. By using *cuMemAllocManaged* instead of *cuMemAlloc* inside the IL code, we are able to provide performance on par with our basic OC-Caffe design discussed in Section VI. In addition, we exploit optimizations like prefetching and advising for *cuMemcpy* calls in the IL code. The optimized IL designs offer much better performance. Details of IL performance have been provided in Section VIII-C1.

VI. Design Details: Out-of-Core Caffe (OC-Caffe)

We now discuss at length how actual DNN training is realized in the proposed OC-Caffe framework. We illustrate the flow of data from the File System to the Host memory and finally to the Device buffers in the default Caffe framework and compare it with the optimized design of OC-Caffe

in Figure 3. We also highlight hotspots where prefetching and advising the CUDA driver can improve performance by avoiding excessive GPU page faults.

A. OC-Caffe Basic Design

DNN training typically requires a huge amount of training data and most of the DL applications need to perform file access to start this phase. Thus, DNN training starts with file access that continues to proceed throughout the training in a batch by batch fashion.

1) Optimizing File Access in OC-Caffe: Caffe follows a very clear phase-based workflow. The *data layer* and *base prefetching data layer* are two layers that deal with accessing training samples that reside on a disk or a shared file system. In either case, a file access operation will transfer the data from the file system to the CPU memory. As mentioned earlier, we call this an *F2H* communication. Since the training is performed on the GPU, these host buffers have to be copied to the device buffers explicitly via *H2D* copies. To deal with this, we propose a unified memory buffer allocated using the *cudaMallocManaged* interface. This buffer is accessible to both the file system as well as the CUDA kernels that need to operate on this data. Thus, in the proposed design, we convert the *F2H* + *H2D* communication to a single *F2M* communication operation. This provides two advantages; 1) It helps to considerably save the GPU memory that allows trainability for large models, and 2) This unified buffer design provides a possibility of overlapping some computation with driver managed copies. In addition to the file access of training samples, the trained DNN state is written to the disk or file system at the end of the training phase. The trained DNN is a relatively small file. This means that the *D2H* copies and the *H2F* operation at the end of the training may not be very expensive. However, we observed that certain very large models see improvement if we use the*cudaCpuDeviceId* hint to set the preferred location of this buffer via the *cudaMemAdvise* interface.

2) Designing Intra-GPU Communication in OC-Caffe: *Layer 2 and onward* mostly comprise of computation and communication intensive phases of the training process. Conceptually, these are called the Forward propagation and the Backward propagation phase. To realize these concepts, Caffe uses member functions of the form *Forward (bottom, top)* and *Backward (top, bottom)* for its *Layer* class. The exact functionality of the layer depends on the type of the layer (e.g., Pooling, Loss, Convolution, etc.). However, both of these functions usually perform two types of primitive operations; a source buffer to destination buffer transfer using a *D2D* primitive, and/or a CUDA kernel that may perform some computation using Device buffers. These transfers are unavoidable because these are real data dependencies that need to be maintained in the GPU memory throughout the training process. Thus, we investigate if the managed alternative of *D2D* copy, i.e., *Managed source to Managed destination — M2M copy* can be exploited to gain significant memory savings. For OC-Caffe basic design, we use unified allocations so kernels can directly operate on the managed buffers. *D2D* copies are simply treated as *M2M* copies in OC-Caffe. *We refer to this basic design of OC-Caffe as* ***oc-caffe-naive*** *in Section VIII.*

B. Advanced Designs of OC-Caffe

The basic design enables "trainability" for out-of-core models. However, there is room for further investigation. We explore if advanced prefetching and advising interface can be exploited to improve performance of *M2M* transfers and CUDA kernels.

1) Optimizing Intra-GPU Communication for Faster Training: The cudaMemcpy operation in *D2D* mode is a copy from a source buffer on the GPU memory to a destination buffer on the GPU memory. The bandwidth for such GPU transfers is usually very high (order of hundreds of Megabytes). For the Volta V100 GPU on our test system, we were able to achieve approximately 740 GB/sec for these transfers. We investigate the performance using a micro-benchmark to understand the difference between a *D2D* copy and a *M2M* copy. We observed that performance of a naive *M2M* transfer is slower than a *D2D* transfer. In this specific case, we are only dealing with the *D2D* copy on the same GPU (intra-GPU communication). We note that this performance difference is expected but the challenge is to hide the overhead of managed buffers involved in (*M2M*) copies. We investigate if advanced designs are possible that leverage *cudaMemPrefetch* and *cudaMemAdvise* interfaces. For *cudaMemcpy* operation, we observe that prefetching the destination buffer provides much better performance for an *M2M* transfer whereas prefetching the source buffer does not have any effect on performance. We note that this is because the source buffer may already be resident in the GPU memory as it is the result of a previously executed kernel. We also note that adding unnecessary *cudaMemPrefetch* calls can lead to extra overhead as well. Further, the *cudaMemAdvise* interface can also negatively impact performance if used in a wrong manner. Thus, careful optimization is required to get the best performance for *M2M* copies. Once the copy completes, the source buffer can be marked as "used" and can be evicted by the CUDA driver when it runs out of physical GPU memory.

For layered DNN training in Caffe, the source buffer is most likely on the GPU memory because for *Layer 1*, and onward, the source buffer becomes a destination buffer of the previous layer's Forward operation. Thus, prefetching the destination buffer of the copy operation can provide performance improvement by reducing GPU page faults. Since we are following a layered approach, the source buffer involved in the copy can be evicted as soon as the copy operation completes. Although there is no direct eviction interface for managed buffers, an advise interface that provides hints to the CUDA driver is available. We use the *cudaMemAdviseSetPreferredLocation* and pass *cudaCpuDeviceId* as the device option to the *cudaMemAdvise* call on the source buffer. When needed, the CUDA driver will move this buffer to the Host memory and clear up space on the GPU memory. The eviction of buffers in the Forward pass may lead to degradation when

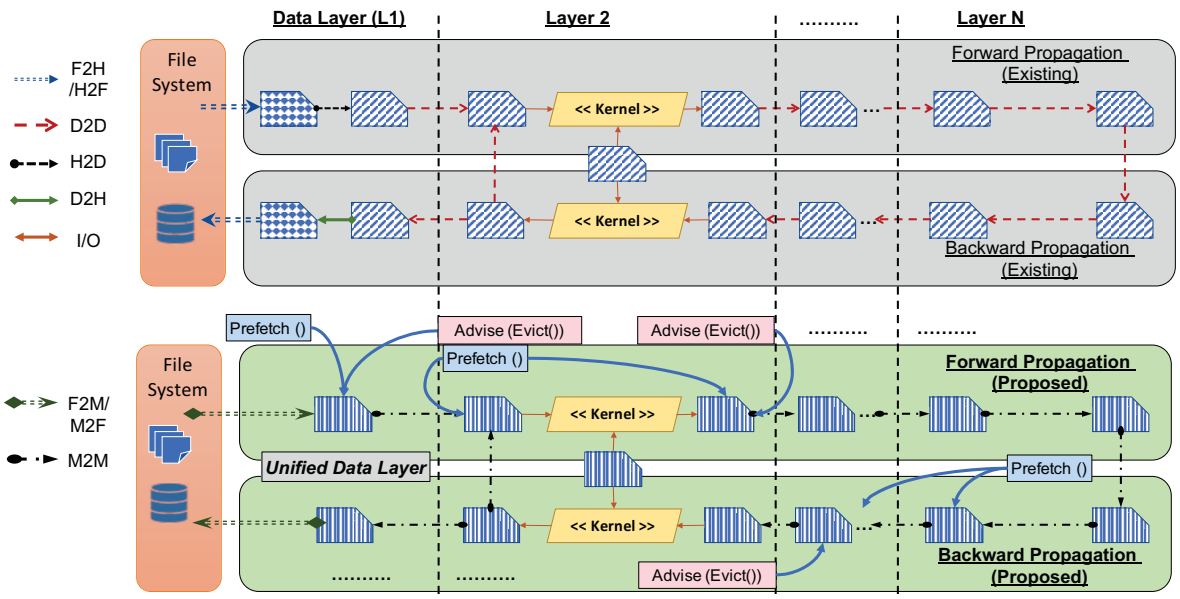


Fig. 3. Comparison of Existing Caffe Design and Proposed OC-Caffe Design: File Access, Forward, and Backward Propagation

we need to access those buffers again during the Backward pass. To deal with it, during the backward pass, we prefetch the required buffers for any *M2M* copy operations as well. We illustrate some of these optimization points in the Figure 3 as **Prefetch()** and **Advise(Evict())**.

Some CUDA kernels behave very similar to *cudaMemcpy* operations, and in fact, some applications use specific CUDA kernels to copy data instead of using the *cudaMemcpy* interface. Furthermore, convolution layers usually use kernels instead of *M2M* copies so incorporating prefetching and advising for managed buffers involved in CUDA kernels becomes necessary. We design the prefetch and advise optimizations for CUDA kernels by following the assumption we made for *M2M* copies, i.e., the destination buffers are prefetched while the source buffers are most likely residing on the GPU memory already. Further, the source buffers can be marked for eviction as soon as the kernel completes. Prefetching and eviction workflow in OC-Caffe is provided as pseudo-code in Algorithm 1.

We refer to these advanced designs that exploit prefetching and eviction for cudaMemcpy calls as well as CUDA kernels as ***oc-caffe-opt*** *in Section VIII.*

Algorithm 1 Prefetching and Eviction in OC-Caffe

```
1:  Definition of Variables
2:      top: source buffer (managed)
3:      bottom: destination buffer (managed)
4:      lid: ID of a layer
5:      evictHint: cudaMemAdviseSetPreferredLocation ←
6:                 cudaCpuDeviceId (hint to advise eviction)
    FORWARD PASS:
7:      cudaMemPrefetchAsync(top)
8:      while ∀ lid ∈ layers do
9:          /* layers[lid].Forward (bottom, top) */
10:         cudaMemAdvise(bottom, evictHint)
11:     end while
    BACKWARD PASS:
12:     cudaMemPrefetchAsync(bottom)
13:     while ∀ lid ∈ layers do
14:         /* layers.[lid].Backward (top, bottom) */
15:         cudaMemAdvise(top, evictHint)
16:     end while
```

VII. Productivity Benefits for OC-Caffe

We now present our analysis and evaluation of the productivity benefits achieved for the proposed *OC-Caffe* framework. The biggest promise of CUDA Managed Memory is higher productivity because of a much simpler programming interface. Indeed, we observe that existing Caffe code has been structured using a complex class hierarchy to deal with the heterogeneity of memory allocations and data movement for GPU-based training. In particular, Caffe utilizes a *SyncedMemory* class and a *Blob* class to abstract out the details of memory management and provide a single interface to access internal buffers that reside on the CPU as well as on the GPU.

In OC-Caffe, we have modified these complex schemes to deal with memory management and data movement between CPU and GPU. Furthermore, we alleviate the need for the *SyncedMemory* class that was used just to maintain the state of a given buffer. Clearly, OC-Caffe's core is simpler and more maintainable due to the use of CUDA Unified Memory buffers across the board.

From a DL user's perspective, Caffe uses the *Layer* class and its member functions of the form *Forward_cpu() / Backward_cpu* and *Forward_gpu() / Backward_gpu()* to deal with heterogeneous training on CPUs and GPUs. To highlight the productivity features offered by the UM interface, we provide a simplified comparison of existing Caffe design and the proposed OC-Caffe design for the *ConvolutionLayer* class in Figure 4. The *ConvolutionLayer* like all other layer classes extend the *Layer* class. Batch Normalization, ReLU, Softmax, Loss, Data, and several other follow a similar design scheme. With the proposed OC-Caffe design, we eliminate the need

for *_cpu() and *_gpu() style separate functions inside *Layer, SyncedMemory, and Blob* classes. We eliminate around 3,000 lines of repetitive code that is both error-prone as well as affects performance as discussed in Section VIII. Community efforts that add custom layer classes will be able to take advantage of the simplified interface.

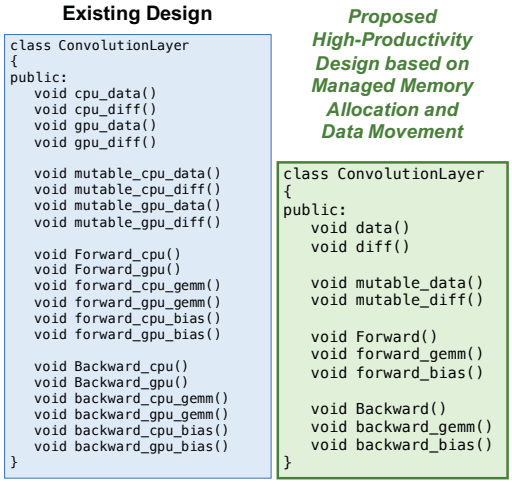


Fig. 4. Productivity Benefits of Unified Memory for OC-Caffe

VIII. Performance Evaluation

We now provide the details of the experimental setup followed by primary and secondary results using multiple combinations of DNNs, batch-sizes, and platforms.

A. Experimental Setup

The experiments were performed on three different platforms labeled P1, P2, and P3.

- *P1:* A single node with one Volta V100-PCIe (16 GB) GPU, a dual-socket Intel Xeon E5-2687W v3 (Haswell) CPU and 64 GB host memory.
- *P2:* A single DGX-1 box with 8 Pascal P100-SXM2 (NVLINK) GPUs, a dual-socket Intel Xeon E5-2698 v4 2.2 GHz CPU, and 512 GB host memory
- *P3:* A multi-node cluster where each node is equipped with 4 Pascal P100-SXM2 (NVLINK) GPUs, a 12-core single-socket Intel Xeon E5-2650 v4 (Broadwell) CPU, and an InfiniBand EDR HCA.

All single-node experiments were performed on P1, scale-up experiments on P2, and scale-out experiments on P3. Results have been reported in Images/second labeled as *Img/sec*. A higher value for *Img/sec* denotes better performance. We ran all the experiments at least three times and provide the average of three runs to rule out any effects of system noise.

B. Primary Results: OC-Caffe vs. other Caffe Variants

We now present our primary results on the latest Volta V100 (P1) for four well-known DNN architectures: 1) AlexNet [11], 2) GoogLeNet [21], 3) VGG-19 [20], and 4) ResNet-50 [7].

1) Trainable Models and Batch Sizes: We call the batch sizes trainable for a specific DL model if the GPU memory can hold the entire DNN. We compare the default version of Caffe (BVLC-Caffe 1.0) labeled as *caffe-gpu* and our proposed OC-Caffe designs (*oc-caffe-naive* and *oc-caffe-opt*) for trainable batch sizes.

Figures 5c, 5d, 5b, and 5a show the results for VGG-19, ResNet-50, GoogLeNet, and AlexNet for *batch sizes 110, 40, 200, and 1,100*, respectively. With the optimized OC-Caffe design (*oc-caffe-opt*), we see slight degradation (2 – 8%) compared to default version of Caffe (*caffe-gpu*). Despite the slight overhead, these findings highlight that a high-productivity design with unified memory allocations and data movement can perform as well as a complicated (low-productivity) design that uses separate GPU and CPU buffers with explicit data movement.

2) Out-of-core Models and Batch Sizes: We call the batch sizes that do not fit the memory as "out-of-core". Figures 6c, 6d, 6b, and 6a show the results for out-of-core batch sizes. Note that the default *caffe-gpu* case is not valid for out-of-core training results as we run into the so called "out-of-memory" error. For our proposed designs, we show both *oc-caffe-opt* and *oc-caffe-naive* to highlight the impact of optimizations in *oc-caffe-opt* that include prefetching and advising hints to the CUDA driver.

What is a quantifiable baseline for out-of-core comparisons? The main challenge for comparing out-of-core training results is that there is no well-established performance baseline. Thus, we first identify a quantifiable baseline for evaluating the performance of our proposed out-of-core designs. We note that no other publicly available solution for out-of-core training exists yet. Therefore, we use the default CPU implementation of Caffe as a baseline since CPU memory is far greater than the GPU memory. Hence, the out-of-core GPU workloads are not out-of-core for CPU. To this end, we go a step further beyond the default CPU implementation in Caffe (*caffe-cpu*) and also include two Intel-optimized Caffe [1] variants to our comparisons; 1) Intel-Caffe with default engine (*intel-caffe*) and 2) Intel-Caffe with optimized MKL-2017 engine (*intel-caffe-opt*). Please note that the purpose of comparing with CPU-based training is not to show the absolute difference in CPU and GPU performance as CPU-based training is dependent of several other factors like the CPU generation, its architecture, the core-count, and its memory sub-system. Therefore, these comparisons are provided to offer a quantifiable reference on a given node. Details about CPU versus GPU performance is beyond the scope of this paper and can be seen in [3].

Discussion of out-of-core comparisons:

We now discuss the out-of-core results. As shown in Figure 6c, the proposed *oc-caffe-opt* design provides up to 10% improvement over *oc-caffe-naive* design for VGG-19 training with batch size 120. In addition, *oc-caffe-opt* is able to achieve 2.1X speedup over MKL-optimized CPU implementation *intel-caffe-opt*. Figure 6d shows that *oc-caffe-opt* is able to give 35% better performance than *oc-caffe-naive* and up to

80% better than *intel-caffe* for batch-size of 45. [2]Figure 6b shows out-of-core GoogLeNet training with a batch size of 210.

Figure 6a shows out-of-core results for the AlexNet model with batch size 1,200. *oc-caffe-opt* offers **1.9X** speedup over *oc-caffe-naive* as well as achieves a significant **5X** speedup over *intel-caffe-opt*. The main reason for higher performance with AlexNet is the fact that it is a relatively smaller model and hence the intra-GPU communication (see Section VI-A2) is not the main contributor in the training time. Unlike AlexNet, ResNet-50 and VGG-19 are very deep architectures and up to 35% time is spent in communication. We verified this using the NVIDIA profiler called (*nvprof*).

C. Secondary Results: Evaluation of Interception Library, Layer-wise Breakdown, and Profiling Analysis

1) OC-Caffe vs. Interception Library (IL) Approach: The proposed Interception Library (IL), as discussed in Section V-B, has been developed as a rapid experimentation tool for exploiting unified memory allocations without any explicit design changes. We used the default version of Caffe (*caffe-gpu*) with IL. The results for Caffe with IL are labeled as *caffe-gpu-IL*. Figure 7a shows the results for trainable ResNet-50 model. Similar to results presented in Figure 5d, we see the same minor (6%) degradation for both *oc-caffe-opt* and *caffe-gpu-IL*. For out-of-core training, shown in Figure 10b, we see that *oc-caffe-opt* offers 19% better performance than the optimized IL approach (*caffe-gpu-IL-opt*). As expected, *caffe-gpu-IL-naive* is about 20% slower than *caffe-gpu-IL-opt* as it does not fully exploit the prefetching features.

2) Layer-wise Breakdown: In addition to the overall training performance comparison, we provide a layer-wise breakdown for the out-of-core AlexNet model in Figure 8. The goal of this result is to gain additional insight at a deeper level. As discussed earlier during Section VIII-B, we observed that *oc-caffe-opt* offers much better performance compared to *oc-caffe-naive*. In this layer-wise breakdown, we are able to corroborate the findings presented in the end-to-end comparison. *oc-caffe-opt* is up to **6.8X** faster than *oc-caffe-naive* for the Layer *pool1*. Please note that we only show select layers of the AlexNet model where we observed significant improvements. It is pertinent to mention that end-to-end time is the summation of time spent in Forward as well as Backward pass over all the layers.

3) Profiling Analysis for ResNet-50 Training: We now discuss the profiling results to explain the performance improvements at an even deeper level. The NVIDIA profiler (nvprof) since CUDA 9 allows profiling of unified memory allocations and data movement operations. To better understand the performance characteristics and verify the end-to-end improvements, we show a comparison for *oc-caffe-opt* and *oc-caffe-naive* for ResNet-50 training in Figure 9 using three metrics reported by the profiler. GPU page fault groups are the primary reason for slower performance for out-of-core workloads. The optimized prefetching and advising in *oc-caffe-opt* allows us to avoid excessive faults and thus it results in **6X** lesser time spent in faults compared to *oc-caffe-naive*. This behavior is in sync with the end-to-end training performance presented in Figure 6d.

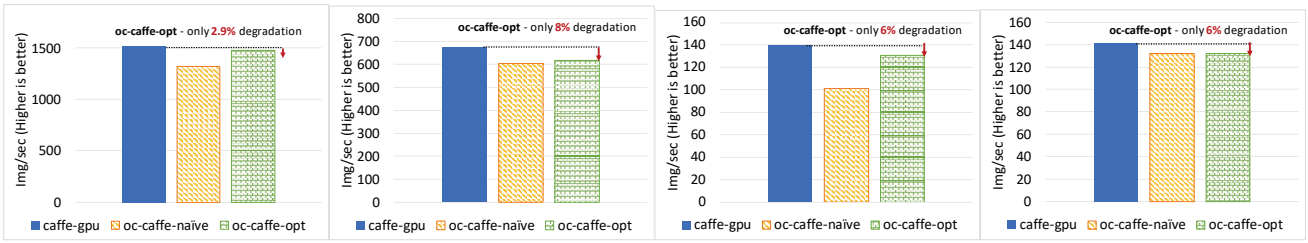
[2]ResNet-50 is a special case where we do not compare the performance against *intel-caffe-opt*. This is because we see that MKL-engine in *intel-caffe-opt* drops out several layers from the training process so the comparison may not be fair. We are still investigating this issue.

D. Scale-up and Scale-out for OC-Caffe

We report both Scale-up (multiple GPUs in a node) and Scale-out (multiple GPUs across multiple nodes) for OC-Caffe. The *Images/second* metric used in the following two results is derived using weak scaling on multiple GPUs. We use the GoogLeNet model with a *batch-size of 32* per GPU. We verify that the training accuracy is not negatively affected. We consider synchronous SGD data-parallelism for both NCCL2 [15] and MPI-based distributed training.

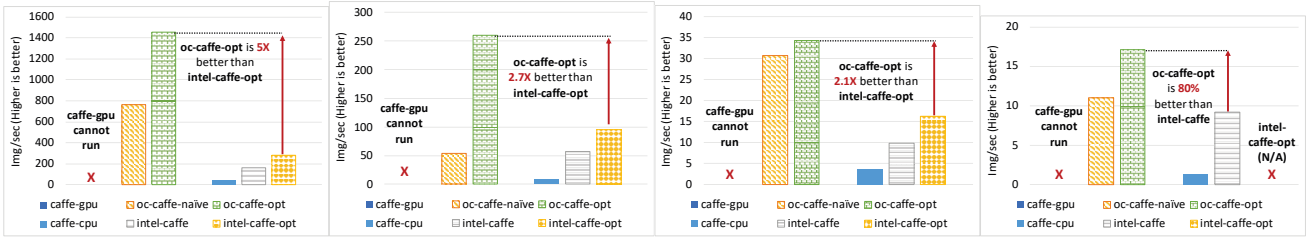
1) Scale-up on DGX-1 with NCCL2: NCCL2 [15] is the recommended library for communication on the NVLINK equipped DGX-1 systems. As Caffe has NCCL support for scale-up only, we exploit this feature in OC-Caffe as well. However, the challenge is that NCCL is not well optimized for managed buffers. Thus, we slightly modify the design and allocate pure device buffers for the gradient communication even when all other allocations are done using the managed memory interface. This results in a comparable scale-up trend for *oc-caffe* and *caffe-gpu* as shown in Figure 10a. *oc-caffe*, despite its usage of unified memory allocations is able to achieve 6X speedup (over single GPU) on 8 Pascal GPUs in the DGX-1 system.

2) Scale-out on Pascal GPU Cluster with MPI: The challenge with scale-out experiments is that default Caffe does not support distributed training across nodes. In order to deal with it, we use data-parallel version of the popular Stochastic Gradient Descent (SGD) algorithm using MPI-based designs for distributed training inside the OC-Caffe framework. The DNN is thus replicated on all processes whereas each of them gets a different piece of training dataset. Our MPI design thus offers a unified-memory based framework that allows over-subscription for a single-GPU case yet provides efficient distributed training across multiple GPU nodes of a cluster. As DL frameworks have several underlying dependencies and libraries that perform differently on different architectures, it is very hard to provide exact performance comparisons. Thus, we restrict ourselves to Caffe and its variants only. In Figure 10b, we present the comparison of *oc-caffe-opt* with another Caffe-based framework called S-Caffe [2] (*osu-caffe*). We note that this comparison is provided only for showing the scalability trends and not the raw performance numbers. S-Caffe only supports cuDNN v5 whereas OC-Caffe is able to take advantage of the new cuDNN v7 library. Because of the faster cuDNN library, we get much better performance. Thus, *oc-caffe-opt* is up to **45%** faster than *osu-caffe* and it achieved speedups (over single GPU training) of 10X and 17X on 16 and 32 Pascal GPUs, respectively.



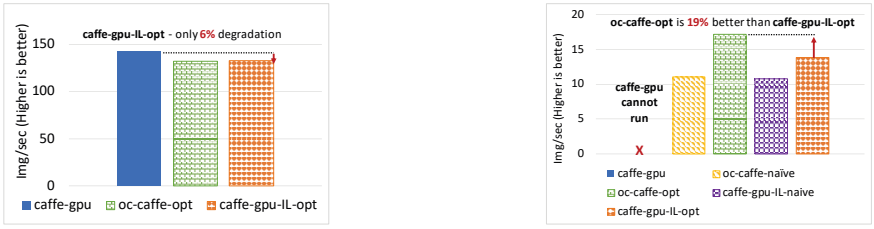
(a) AlexNet (b) GoogLeNet (c) VGG-19 (d) ResNet-50

Fig. 5. Performance Comparison: Proposed OC-Caffe vs. Existing Caffe Variants for Trainable (in-memory) Workloads



(a) AlexNet (b) GoogLeNet (c) VGG-19 (d) ResNet-50

Fig. 6. Performance Comparison: Proposed OC-Caffe vs. Existing Caffe Variants for Out-of-core Workloads



(a) Trainable ResNet-50 Model (b) Out-of-Core ResNet-50 Model

Fig. 7. Performance Comparison: Caffe with Interception Library (IL) vs. OC-Caffe for ResNet-50 Training

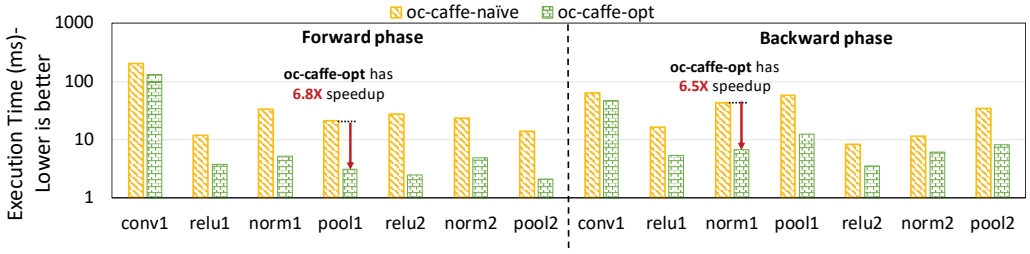


Fig. 8. Layer-wise Comparison for Select Layers of AlexNet in Forward and Backward Passes

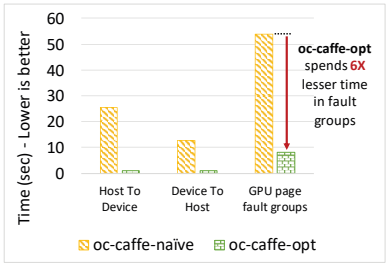
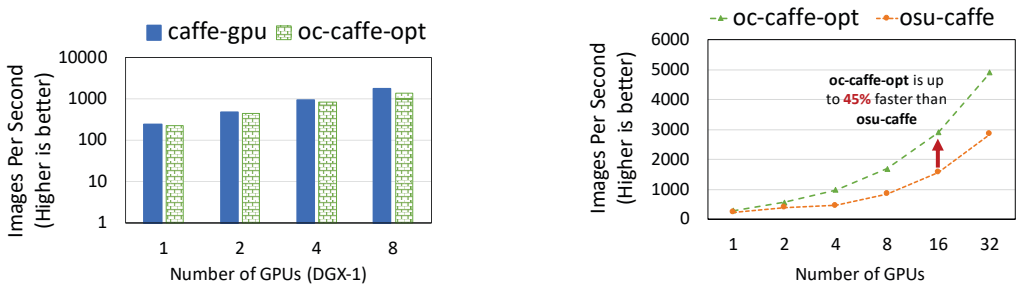


Fig. 9. Profiling Results for ResNet-50 Training

IX. Conclusion

We are witnessing an ever increasing interest in large DNN architectures to solve diverse set of problems using the powerful GPUs like the Volta V100. To allow efficient training for emerging out-of-core DNN workloads, we designed efficient schemes in the proposed OC-DNN framework that exploit managed memory communication primitives and innovative features like prefetching and advising to avoid GPU page faults. We envision that our work will act as a guideline for development of future CUDA applications that deal with "out-



(a) Scale-up on 8 Pascal GPUs (NVIDIA DGX-1 System)

(b) Scale-out on 32 Pascal GPUs (Multi-node Cluster)

Fig. 10. Performance Comparison: Caffe with Interception Library (IL) vs. OC-Caffe for ResNet-50 Training

of-core" workloads. We highlight both productivity and performance benefits achieved by OC-Caffe; 1) We removed 3,000 lines of repetitive and error-prone code by exploiting unified memory primitives, and 2) We presented OC-Caffe designs that provide up to 1.9X improvement over naive designs for out-of-core DNN training and comparable performance for regular workloads. OC-Caffe is up to 45% faster than OSU-Caffe, provides up to 6X speedup for distributed training on DGX-1 system, and offers multi-node training speedups of 10X and 17X on 16 and 32 Pascal GPUs, respectively. We plan to publicly release the OC-Caffe framework.

ACKNOWLEDGEMENT

This research is supported in part by National Science Foundation grants #CNS-1513120, #ACI-1450440 and #CCF-1565414.

REFERENCES

[1] "Intel Caffe," https://github.com/intelcaffe, 2018, [Online; accessed Mar-2018].

[2] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, "S-Caffe: Co-designing MPI Runtimes and Caffe for Scalable Deep Learning on Modern GPU Clusters," in *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '17. New York, NY, USA: ACM, pp. 193–205. [Online]. Available: http://doi.acm.org/10.1145/3018743.3018769

[3] A. A. Awan, H. Subramoni, and D. K. Panda, "An In-depth Performance Characterization of CPU- and GPU-based DNN Training on Modern Architectures," in *Proceedings of the Machine Learning on HPC Environments*, ser. MLHPC'17. New York, NY, USA: ACM, 2017, pp. 8:1–8:8. [Online]. Available: http://doi.acm.org/10.1145/3146347.3146356

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A Large-Scale Hierarchical Image Database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[5] L. Durant, O. Giroux, M. Harris, and N. Stam, "Inside Volta: The Worlds Most Advanced Data Center GPU," https://devblogs.nvidia.com/parallelforall/inside-volta/, 2017, [Online; accessed Mar-2018].

[6] M. Harris, "Unified Memory for CUDA Beginners," https://devblogs.nvidia.com/parallelforall/unified-memory-cuda-beginners/, 2017.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[8] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer, "FireCaffe: Near-linear Acceleration of Deep Neural Network Training on Compute Clusters," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2592–2600.

[9] Y. Ito, R. Matsumiya, and T. Endo, "ooc_cuDNN: Accommodating convolutional neural networks over GPU memory capacity," in *2017 IEEE International Conference on Big Data (Big Data)*, Dec 2017, pp. 183–192.

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

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

[12] R. Landaverde, T. Zhang, A. K. Coskun, and M. Herbordt, "An Investigation of Unified Memory Access Performance in CUDA," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, Sept 2014, pp. 1–6.

[13] Microsoft, "CNTK," http://www.cntk.ai/, 2017.

[14] NVIDIA, "DGX-1," https://www.nvidia.com/en-us/data-center/dgx-1/, 2017, [Online; accessed Mar-2018].

[15] ——, "NCCL 2," https://developer.nvidia.com/nccl, 2017.

[16] ——, "The nvlink fabric," https://www.nvidia.com/en-us/data-center/nvlink/, 2018.

[17] J. Park, H. Cho, W. Jung, and J. Lee, "Transparent GPU Memory Management for DNNs," in *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '18. New York, NY, USA: ACM, 2018, pp. 411–412. [Online]. Available: http://doi.acm.org/10.1145/3178487.3178531

[18] 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 *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–13.

[19] N. Sakharnykh, "Unified Memory on PASCAL and VOLTA," http://on-demand.gputechconf.com/gtc/2017/presentation/s7285-nikolay-sakharnykh-unified-memory-on-pascal-and-volta.pdf, 2017, [Online; accessed Mar-2018].

[20] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[22] D. Wang, A. Khosla, R. Gargeya, H. Irshad, and A. H. Beck, "Deep Learning for Identifying Metastatic Breast Cancer," *arXiv preprint arXiv:1606.05718*, 2016.