S-Caffe: Co-designing MPI Runtimes and Caffe for Scalable Deep Learning on Modern GPU Clusters

Ammar Ahmad Awan
Dept. of Computer Science and Engg. The Ohio State University
awan.10@osu.edu

Khaled Hamidouche
Dept. of Computer Science and Engg. The Ohio State University
hamidouc@cse.ohio-state.edu

Jahanzeb Maqbool Hashmi
Dept. of Computer Science and Engg. The Ohio State University
hashmi.29@osu.edu

Dhabaleswar K. Panda
Dept. of Computer Science and Engg.
The Ohio State University
panda@cse.ohio-state.edu

Abstract

Availability of large data sets like ImageNet and massively parallel computation support in modern HPC devices like NVIDIA GPUs have fueled a renewed interest in Deep Learning (DL) algorithms. This has triggered the development of DL frameworks like Caffe, Torch, TensorFlow, and CNTK. However, most DL frameworks have been limited to a single node. In order to scale out DL frameworks and bring HPC capabilities to the DL arena, we propose, S-Caffe; a scalable and distributed Caffe adaptation for modern multi-GPU clusters. With an in-depth analysis of new requirements brought forward by the DL frameworks and limitations of current communication runtimes, we present a co-design of the Caffe framework and the MVAPICH2-GDR MPI runtime. Using the co-design methodology, we modify Caffe's workflow to maximize the overlap of computation and communication with multi-stage data propagation and gradient aggregation schemes. We bring DL-Awareness to the MPI runtime by proposing a hierarchical reduction design that benefits from CUDA-Aware features and provides up to a massive 133x speedup over OpenMPI and 2.6x speedup over MVAPICH2 for 160 GPUs. S-Caffe successfully scales up to 160 K-80 GPUs for GoogLeNet (ImageNet) with a speedup of 2.5x over 32 GPUs. To the best of our knowledge, this is the first framework that scales up to 160 GPUs. Furthermore, even for single node training, S-Caffe shows an improvement of 14% and 9% over Nvidia's optimized Caffe for 8 and 16 GPUs, respectively. In addition, S-Caffe achieves up to 1395 samples per second for the AlexNet model, which is comparable to the performance of Microsoft CNTK.

Keywords MPI_Reduce; CUDA-Aware MPI; Caffe; Deep Learning; Distributed Training

1. Introduction

We are witnessing the era of rapid advances in Artificial Intelligence (AI) fueled by the resurgence of Deep Learning (DL) algorithms and techniques. With widespread applications like Image Recognition, Speech Processing, Textual Analysis, Breast Cancer Detection [49], and Self Driving cars [11], DL has become one of the hottest topics of interest for both academic research groups as well as prominent companies like Google, Baidu, Facebook, and Microsoft. There are two main driving forces behind the momentum that DL has recently gained; first is the public availability of versatile training data sets like ImageNet [25] and CIFAR [35], and second is the affordability of modern high-performance and data-parallel hardware like GPUs and accelerators. This has triggered the development and adoption of DL frameworks like Caffe [33], Torch [21], Theano [17], TensorFlow [14], and Microsoft CNTK [12]. Research efforts from DL and AI communities have been mostly geared towards designing better, bigger, and deeper neural networks (DNN) for improving the accuracy of trained models like AlexNet [36], CaffeNet [7], GoogLeNet [45], Network in Network [38], and VGG [44]. While these models differ in their science and the order of computational layers, they commonly share the need for faster computation and communication capabilities of the system they run on.

At the heart of the Deep Learning resurgence, accelerators like NVIDIA GPUs have seen a phenomenal adoption

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PPoPP '17 Feb. 4–8, 2017, Austin, Texas, USA.
© 2017 ACM. ISBN 978-1-4503-4493-7/17/02...$15.00
DOI: http://dx.doi.org/10.1145/3018743.3018769

by the community to accelerate parallel training. Interestingly, GPUs are increasingly becoming popular in the HPC community as well. This is evident from the Top500 [40] supercomputers list where 22% of top 50 systems are GPU based. Due to their high throughput support and an architecture designed specifically for data parallel workloads, GPUs are considered a very good match for DL computational requirements. As a matter of fact, the operations involved in training a DL model have a very high arithmetic intensity [9, 20] thereby making GPU hardware a great match. Thus, most of the DL frameworks including Caffe, TensorFlow, CNTK, and various others are being designed to take advantage of the GPU capabilities. Using NVIDIA's Compute Unified Device Architecture (CUDA) [43] API, these frameworks boost their scale-up (single address space) efficiency using threads to utilize multiple GPUs in a single node [16]. However, the performance of single-node multi-GPU training is nearing saturation for large DL data sets and models as highlighted in Section 3. Thus, scale-out (distributed address space) efficiency for DL frameworks is an emerging requirement. The importance of training for DL frameworks has been stressed by many in the DL community. In particular, Jeffrey Dean from Google highlights in his keynote address [32]: *"training time is a key challenge at the root of the development of new DNN architectures."*

Thus, to develop distributed training support for DL frameworks, we first need to understand and utilize prevalent parallelization approaches for distributed-memory HPC clusters. We can clearly see that in the HPC world, Message Passing Interface (MPI) is perhaps the most popular and widely used parallel programming model for developing large-scale applications. Furthermore, MPI runtimes themselves have witnessed rapid changes and optimizations with the advent of GPUs and accelerators. As the CUDA API matured, several features like Unified Virtual Addressing (UVA) have led MPI researchers to propose the "CUDA-Aware" concept. The idea is to transparently provide support for GPU-based data transfers within and across cluster nodes. Several MPI implementations like MVAPICH2 [41], OpenMPI [47], and CrayMPI [22] now provide efficient CUDA-Aware support. This support includes direct transfer of data from the GPU's (device) memory to improve performance of GPU-GPU communication using techniques like CUDA IPC, GPUDirect RDMA [8], and pipelining. This achieves better performance and programmability for hybrid MPI+CUDA applications. In this context, a lot of traditional HPC applications have been successfully redesigned to scale-out using a hybrid programming model mixing MPI and CUDA [15, 18]. In such cases, CUDA is used to offload computation to the GPU device while MPI is responsible for inter-processes communication. However, most of the current DL frameworks have not been designed with such CUDA-Aware (MPI+CUDA) techniques.

Along with the broad challenge of *efficiently scaling-out a DL framework to take advantage of HPC resources*, we highlight the following design challenges that need to be addressed in order to provide a scalable and efficient DL framework:

- What are the performance and portability bottlenecks when moving from a shared address space DL framework to a distributed address space environment for a data-parallel design?
- How to design a scalable and distributed framework that provides both efficient scale-up performance for a single node and scale-out performance for multiple nodes in a distributed manner?
- What are the new requirements that DL frameworks bring forward for distributed communication runtimes?
- Can co-design of MPI runtimes and DL frameworks be exploited to boost performance and scalability for modern multi-GPU HPC systems?
- What are the performance trends and benefits that can be observed for a distributed and co-designed DL framework like S-Caffe?

In order to address the above challenges, using a co-design methodology, we propose *S-Caffe*; a novel and scalable DL framework that provides efficient scale-up and scale-out for distributed training on accelerated HPC systems. We investigate a co-design approach of S-Caffe and MPI runtimes to bring the distributed-memory space efficiency and CUDA-Awareness of MPI semantics to the S-Caffe framework. We redesign and enhance MPI protocols to satisfy the new requirements pertaining to movement of very large messages in DL frameworks. Table 1 positions the S-Caffe framework in the DL framework design and features space. To the best of our knowledge, this is the first study that provides an in-depth analysis of fundamental system-level bottlenecks being faced by DL frameworks like Caffe and highlights novel co-designed solutions to achieve high performance and scalability.

We make the following key contributions:

- Analyze and identify new requirements brought forward by DL frameworks, especially for large message communication and reductions in MPI runtimes.
- Co-Design, implement, and evaluate S-Caffe; a scalable and distributed CUDA-Aware MPI enabled DL framework.
- Propose novel exploitation of MPI-3 Non-blocking Collective (NBC) operations via multi-stage data propagation and helper-thread based gradient aggregation in DL frameworks.
- Co-design, implement, and evaluate a GPU-kernel based hierarchical CUDA-Aware reduction primitive (MPI_Re-

Deep Learning Frameworks	Framework Features and Platforms Supported							
	Distributed Address Space Systems				Shared Address Space Systems		Parallelization Strategy / Implementation	
	Basic MPI Support	CUDA-Aware MPI	Overlapped Designs (NBC Support)	Co-Designed with MPI runtimes	Single GPU Training	Multi-GPU Training	Model Parallel (MP) / Data Parallel (DP)	Parameter-Server (PS) / Reduction-Tree (RT)
Caffe [33]	×	×	×	×	✔	✔	DP	RT
FireCaffe [30]	✔	Unknown	×	Unknown	✔	✔	DP	RT
MPI-Caffe [37]	✔	×	×	×	✔	✔	MP	N/A
CNTK [12]	✔	×	×	×	✔	✔	MP/DP	PS
Inspur-Caffe [31]	✔	✔	×	×	✔	✔	DP	PS
S-Caffe (proposed)	✔	✔	✔	✔	✔	✔	DP	RT

Table 1. Design and Features Space for Modern Deep Learning Frameworks

duce) for supporting large-scale reductions in DL frameworks like S-Caffe.

- Evaluate and analyze performance of the proposed S-Caffe with small and large data sets like CIFAR10 and ImageNet using various networks like AlexNet and GoogLeNet on different GPU Clusters including a Cray CS-Storm dense multi-GPU system.

The rest of the paper is organized as follows. Section 2 contains important preliminaries including architecture of DL frameworks like Caffe. In section 3, we highlight the challenges and requirements for designing scalable DL frameworks like S-Caffe. In Section 4, we describe at length, the proposed S-Caffe co-design schemes and design alternatives. The details of the proposed hierarchical multi-level MPI_Reduce are provided in Section 5. In section 6, we present a comprehensive performance evaluation. Section 7 covers related DL frameworks and research efforts. We conclude the paper in Section 8.

2. Preliminaries: DL Frameworks, Caffe, and CUDA-Aware MPI

In this section, we provide an overview of different DL frameworks, a detailed description of Caffe's architecture followed by a brief discussion of CUDA-Aware MPI runtimes.

2.1 Architecture of Deep Learning Frameworks

DNNs like AlexNet, GoogLeNet, and VGG have several architectural variations ranging from the number of layers, filter dimensions, among others. Single GPU or CPU based DNN training broadly comprises of iterations over two compute intensive passes; the Forward pass for training the sample data, and the Backward pass for performing gradient computation w.r.t the weights and samples used in the forward pass. In general, all the DNN structures try to model high-level abstractions in a layered manner.

2.2 Caffe: Design and Implementation

We now define major design and implementation components of the Caffe framework; the Net (or the Model), the Layers, the Solvers, and the Forward/Backward computational passes. We discuss them in the context of this work but a detailed description of these can be obtained from [6]. Figure 1 highlights different phases of the Solvers and their interactions with Layers of a Net.

Net, Solvers, and Layers: Caffe's Net class is the base that holds important components of computation, communication, and I/O. Net is an abstraction for a DL network like AlexNet or GoogLeNet. Net is sometimes also called the Model. The heart of the framework that orchestrates the execution of different computations using layers is called a Solver. The Solver class is an abstraction that provides an interface that can be implemented by different solvers e.g., a Stochastic Gradient Descent (SGD) solver. Caffe uses a Solver object for each of the GPU. Each Solver has its own Net object that operates on the Layer data. Layers are abstractions for a defined set of computations on the data. It is an interface that can be implemented by DL scientists for specific functionalities. Examples include Convolution, Pooling, and Data Layers. Caffe layers contain two important data structures for communication between Solvers; the parameter data used in the Forward pass and the parameter gradients calculated during the Backward pass.

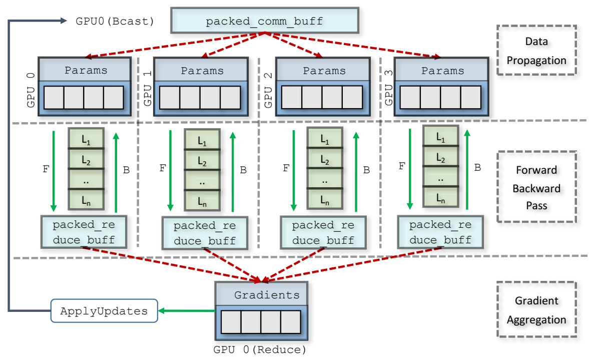

Figure 1. Caffe Architecture: Data Propagation, Forward/Backward Computation, and Gradient Aggregation

Major Computation and Communication Phases: Caffe trains a Net in multiple iterations. Each iteration consists of three main phases; the data propagation phase, the Forward/Backward computational passes, and the gradient aggregation phase. Data propagation provides the required parameters. The packed_comm_buffer in Figure 1 shows the combination of data for different layers that is packed onto a buffer at the root solver (GPU 0). The root solver then communicates this buffer to all of the other solvers on different GPUs. The Forward pass of layers 1 to n generates a loss value that is propagated back in the Backward pass to calculate gradients. The gradients place on the packed_reduction_buffer are reduced back to the root solver during the gradient aggregation phase. The root solver then calls the ApplyUpdate() function and updates the parameter data for the next iteration.

2.3 CUDA-Aware MPI

Before the arrival of UVA and GPUDirect, MPI application developers used explicit copies of data between GPU and CPU memory. To use the Send primitive, an explicit device to host copy was required. Similarly, a CPU to GPU copy was needed after receiving the data using a Receive primitive. For supporting such kind of operations transparently, several MPI implementations including MVAPICH2 and OpenMPI provide CUDA-Aware MPI primitives for performing point-to-point, one-sided, and collective operations. This concept enables applications to perform direct and efficient communication from GPU buffers transparently using MPI primitives.

3. Challenges and Requirements for Designing Scalable DL Frameworks

In this section, we elaborate specific challenges, requirements, and issues that need to be addressed in order to design a scalable DL framework on modern GPU systems.

3.1 Parallelization Strategies and Implementation Styles

DL community has explored at least two parallelization approaches [34]: data-parallel approach and model-parallel approach. In this paper, we focus on the data-parallel approach where the same model is replicated for every processing element (a CPU core or a GPU), but is fed with different parts of the training data. To implement the data-parallel approach, there are two different design choices; first is called the parameter-server approach and second is called reduction-tree approach. As shown in Table 1, one of the most recent Caffe adaptation [31] by Inspur, Microsoft CNTK [12], and Intel-Caffe [3] follow the parameter-server design. This is analogous to a classical master-worker design pattern where each of the workers operate on a piece of data and each one of them sends it to a server. The server then aggregates the gradients and sends updated gradients to each of the workers. Parameter-server has also been implemented by frameworks like DistBelief [24]. More recently, Project Adam [19], GeePS [23], and Distributed TensorFlow [14] have also used variants of the parameter-server paradigm.

However, we argue that this design paradigm provides limited scalability. Using a single GPU as the master to perform the aggregation of very large buffers (order of megabytes) for every iteration renders the server to be the bottleneck. Using multiple servers may take away the GPUs from performing the actual training operation especially in the case of GPU-based parameter servers like GeePS [23]. Multiple servers may also pose additional overheads due to the second layer of synchronization between servers. Further, limited GPU memory may pose additional restrictions on the scalability of this approach. Inspur-Caffe, for instance, has only been able to work up to 16 processes. On the other hand, researchers have shown clear benefits of the reduction-tree approach [30]. The original Caffe maintained by BVLC as well as the Nvidia's fork, despite being multi-threaded (non-MPI) implementation, use a symmetric parallelization approach where all GPUs (solvers) communicate using a tree-like pattern. Hence, for the proposed S-Caffe, we investigate the data-parallel approach and its implementation using symmetric SPMD-style solvers where each GPU performs the same amount of work and the communication is performed in a collective fashion before the start and end of each iteration.

3.2 Distributed Address-Space Design and Parallel Data Reading

While GPU systems with multiple GPUs per node are very popular in HPC centers, the number of GPUs in a node is limited and it is common to have 2 or 4 GPUs per node. Hence, in order to satisfy the computing requirement of DL models, distributed frameworks are needed. Indeed, Caffe has been designed primarily for a single address space system where a single process can use multiple threads to take advantage of multiple GPUs in a node. This approach is limited by design to only work for an intra-node scale-up system.

On the other hand, HPC systems usually use a parallel file system (PFS) and storage nodes. In order to limit the I/O overhead in reading large size datasets available in the DL field, frameworks need to be redesigned with parallel reading capabilities to take advantage of PFS like Lustre [39]. Indeed, one of the challenges in developing a distributed system is feeding the training data efficiently to all the solvers. For Caffe, the training data is image files maintained in an LMDB [10] database. Caffe uses a Data Reader thread to constantly bring data from disk to memory queues. A single reader can share the data with the different solvers using a shared queue in Caffe. However, for a distributed framework, we need to design an efficient parallel data reading mechanism.

3.3 Exploiting Overlap of Computation and Communication

DL frameworks like Caffe are both computation and communication intensive. Hence, in order to scale out such frameworks, it is mandatory to overlap the computation with the communication. Indeed, Caffe orchestrates the training process in clearly marked phases like data propagation, forward/backward passes, and gradient aggregation. The issue with marked phases that block for the completion of the previous phase is the sequential nature of workflow. Such separation of computation and communication phases severely limits the system efficiency and scalability. In order to alleviate such limitations, a total redesign of the workflow to bring overlap between the different phases is required. Such a redesign requires: 1) an in-depth analysis of the workflow and its different phases, 2) a fine-grain restructuring of the coarse-grain phases to multiple fine-grain phases thereby exposing potential for overlap, and 3) exploitation of new communication semantics to perform computation/communication overlap.

3.4 Designing DL-Aware Communication Runtimes

MPI, as the defacto programming model for distributed system, has been optimized for HPC workloads. MPI communication protocols and runtimes including the CUDA-Aware ones, are optimized for small and large message sizes up to 4 MB. However, DL frameworks, with their extensively large message sizes (256 MB), bring forward new requirements and entail designing new DL-Aware communication runtimes and protocols. Further, as hinted earlier, DL frameworks are targeting GPU systems. Hence, the new runtime should be GPU/CUDA-Aware in addition to being DL-Aware. For instance, in most of HPC applications, a Reduce operation is performed on 16-64 byte buffer so MPI runtimes can use the CPU to perform such small reductions. However, for DL frameworks, where the aggregation phase requires a reduction on a 256 MB buffer, new algorithms, and runtime-level communication infrastructure are needed that can take advantage of GPU computing capabilities and features like GDR and IPC.

4. S-Caffe: Proposed Architecture and Co-designs

We now describe how we tackled the above challenges in designing S-Caffe. Figure 2 illustrates the architectural overview of the proposed S-Caffe framework at an abstract level. First, we illustrate our experiences in designing a basic CUDA-Aware MPI version. Then, we describe how to push the performance frontiers of such a framework via a co-design approach of the different computational layers (Caffe) and the communication runtime (MPI). Our approach includes: 1) re-designing S-Caffe workflow to maximize the overlap potential by taking advantage of MPI-3 semantics and features including non-blocking collective (NBC) operations in a multi-staged fashion to overlap the computation and communication; and 2) a new CUDA-Aware MPI parallel reduction design that combines GPUDirect RDMA and GPU kernels to efficiently support large-size GPU based reduction operations.

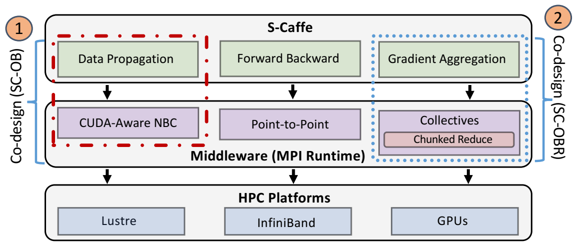

Figure 2. Architectural Overview of S-Caffe

4.1 Basic CUDA-Aware MPI Design (SC-B)

As mentioned earlier, Caffe has been designed using the single-process multi-threaded programming paradigm. Moving from a single address space to a distributed address space is a challenging process that requires a significant code analysis and reorganization effort due to the complexity of Caffe's layered architecture. Further, re-designing an object oriented code where threads share objects across heterogeneous CPU and GPU memory spaces increases the complexity of the work. Below, we highlight how we tackled these challenges. *We note that this design is called S-Caffe Basic (SC-B) in the results section.*

Explicit Data Movement Operations: To move from the shared data access pattern with *Parent-Child* relationship between the solvers to a distributed system, an explicit communication between different processes (solvers) is required. A naive implementation can serialize the different C++ objects and exchange them with MPI operations. This can introduce a significant overhead. To avoid this, we identify minimal data exchange that is required, utilize existing GPU-based communication buffers, and communicate them across processes in a CUDA-Aware fashion. This avoids unnecessary copies between the CPU and GPUs.

Parallel Readers: The second design goal for S-Caffe is to optimize file access operations. To make it possible, we investigated two alternative design choices; first is to use a single data reader thread and communicate the data to other processes, and second is to use parallel data reader threads for each of the processes and maintain separate distributed queues. We chose the second option because it can be optimized for parallel file systems like Lustre where parallel reads from disk can be faster than exchanging images using MPI-level communication. The proposed design is presented in Figure 3. In addition, the parallel reader design can readily be used with both LMDB and ImageDataLayer of Caffe. It is pertinent to mention that even-though LMDB allows parallel accesses, its scalability is very limited. Indeed as discussed later in Section 6, LMDB does not scale for more than 64 parallel readers. On the other hand, ImageDataLayer allows reading image files directly from Lustre storage and can scale to any number of processes. The results section (Figure 8) highlights how our proposed parallel data reading combined with ImageDataLayer achieves scalability up to 160 GPUs.

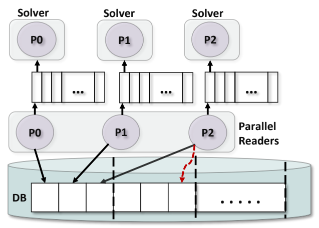

Figure 3. S-Caffe: Proposed Parallel Data Reader Design with Distributed Queues

Collective Operation Patterns for Data Propagation and Aggregation: After extracting the different memory access patterns to shared buffers and changing them to explicit MPI point-point communication as described earlier, with extra analysis of the MPI communication pattern, we noticed an opportunity for improvement. Thus, we enhanced the on_start() data propagation phase to use the MPI broadcast primitive (MPI_Bcast). It is worth mentioning that we use the CUDA-Aware version of this operation. In a similar fashion, for the gradient aggregation part, we use an MPI_Reduce operation. Even-though such designs can be efficient for small scale and especially for small data sets like MNIST and CIFAR, to push the scalability and efficiency for large DL models, a co-design approach is required as discussed in the next section.

4.2 Multi-stage Non-blocking Collectives for Maximal Overlap (SC-OB)

The CUDA-Aware MPI design presented in the previous section provides reasonable performance for small and medium scale data sets like MNIST and CIFAR10. However, as the data set sizes and communication parameters increase with scale, the basic design (SC-B) becomes communication-bound where communication overhead gets significant. To minimize the overhead of this communication, we re-designed S-Caffe workflow to provide opportunities for overlap and co-designing with MPI runtimes.

Figure 1 shows how Caffe performs communication and computation in clearly marked phases that proceed after the completion of one another. This phase based computation and communication is intuitive but limits the efficiency of the system. On the other hand, MPI-3 has introduced Non-Blocking Collective (NBC) operations that provide support for overlapping computation and communication. Naively using NBC may result in performance degradation if the computation/communication overlap ratio is small. In order to efficiently co-design the communication and computation using NBCs, we did a deeper investigation of the data structures and re-designed the core workflow. First, we have moved from a coarse-grain workflow with mainly three successive phases to a fine-grain workflow with multiple phases. We proposed a new workflow which maximizes the overlap of computation and communication phases. The proposed S-Caffe workflow communicates the model parameters and the gradient data in an *on-demand* fashion instead of a coarse-grained approach that performs communication of the entire data in a single operation.

The new workflow overlaps the data propagation with the Forward pass using MPI_Ibcast() calls to send the required data before starting the Forward pass. An Ibcast call is a non-blocking call that returns immediately and MPI runtimes typically progress the communication in the background using different strategies [28, 29]. An MPI_Wait() call is made to complete the operation. Wait call blocks until the progression and completion of the corresponding request. The challenge is to find the best place to use the MPI_Wait() operation. Calling this operation too soon will lead to poor computation/communication overlap as the communication will be mainly progressed during this call. On the other hand, calling it too late might lead to degradation as it limits the potential of overlapping future computation phases. To overlap the propagation with the forward layers, we investigated two designs. As shown in Figure 4, a naive design tries to overlap the communication of layer $i + 1$ with the computation phase of layer i. However, this limits the asynchronous progress and hence the overlap potential. To maximize the overlap potential, as shown in Figure 5, we propose a multi-stage on-demand design. It starts all Ibcast operations at the beginning and posts the appropriate Wait operation of ith Ibcast just before the ith Forward pass of a layer that actually needs the corresponding data. *This co-design approach is called S-Caffe Optimized Broadcast (SC-OB) in the results section.*

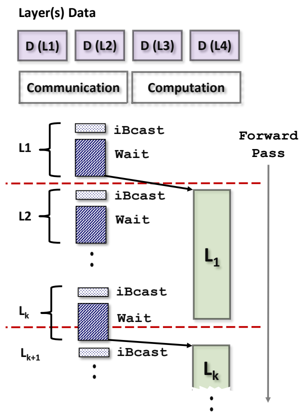

Figure 4. Naive NBC Design for Data Propagation

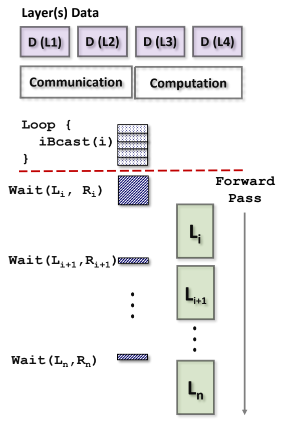

Figure 5. Overlapped Data Propagation with Forward

Taking such overlapped design one step further, we investigated schemes to overlap the gradient aggregation operation with the Backward pass. However, this is not straightforward and cannot exploit an Ibcast-like design proposed earlier that can use MPI_Ireduce operations. Such an approach is limited by the fact that MPI runtimes do not provide efficient NBC reduction primitives as they require the CPU to progress and perform the computation, which clearly nullifies the overlap potential. Further, falling back to a blocking reduce operation and trying to take advantage of a multi-stage scheme to split the reduce operation to multiple reduce operations interleaved between the different backward phases will not provide any overlap as: 1) the number of steps in performing N medium sized reductions are equivalent to a single big reduction, and 2) Even-though the CPU is free, each reduce call is required to wait for the completion of the GPU based backward layer before performing the reduce operation.

To overcome these bottlenecks, we propose a different co-design approach to overlap the gradient aggregation with the backward passes. Our approach includes two main parts: 1) a helper thread to separate the kernel completion and wait for communication progress in order to achieve overlap, and 2) A novel DL-Aware MPI Reduce design that exploits multi-level communicators and kernel-based computation to efficiently reduce large size buffers. Helper-thread co-design is discussed in the next sub-section while the DL-Aware Reduce Design is presented at length in Section 5.

4.3 Co-design Gradient Aggregation for Maximal Overlap and Performance: Helper control thread (SC-OBR)

To split the control flow between the communication and computation phases, we have offloaded the invocation of a layers' backward pass to a helper thread. The helper thread signals the main thread to invoke the reductions in a multi-stage fashion. In this layered design, the $n'th$ layer's reduce requires the completion of the $n'th$ layer's computation. We have designed the two threads to synchronize using a C++ condition flag. This design ensures the overlap of $n'th$ reduce with the layer $n-1's$ compute. While this scheme gives an opportunity for an overlap, however, in order to maximize its efficiency, the reduce operation needs to be optimized as well. Figure 6 shows how the overlapped gradient aggregation hides the latency of reductions under the large computation of layers in the Backward pass.

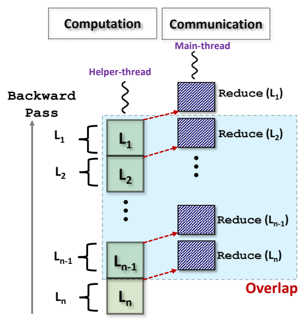

Figure 6. Overlapped Gradient Aggregation with Backward

5. Efficient DL-Aware Hierarchical Reduce (HR):

In modern MPI runtimes like MVAPICH2 and OpenMPI, we typically observe that large-scale collective communication and reduction primitives are implemented using a flat single algorithm like Binomial Tree (and many others), or a hierarchical algorithm with intra-node and inter-node communication. However, this simple intra- and inter-node hierarchical scheme is not sufficient in the context of GPU-based communication and large-scale reductions, especially on systems that contains only 2-4 GPUs per node. Thus, we need to design a hierarchical mechanism for MPI communicators such that a lower level communicator can also span multiple nodes. Detailed discussion about multi-level communicators and their implementation is beyond the scope of this paper. Thus, we present the relevant details for Hierarchical Reduce (HR) and S-Caffe co-design only. Figure 7 presents a two-level design where the lower level communicator spans two nodes and each node has four GPUs. Thus, a total of eight GPUs are in the lower level communicator. We call the number of GPUs in the communicator as the chain-size. We note that chain-size is a runtime parameter that can be dynamically tuned for different systems and configurations. The upper level communicator, in this case, uses a binomial tree algorithm. The selection of algorithms for both upper and lower level communicators can be controlled using runtime parameters.

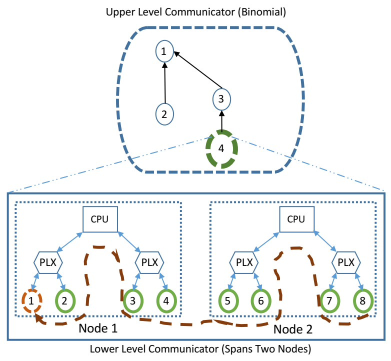
Upper Level Communicator (Binomial)
1
2
3
4
CPU
PLX
PLX
1
2
3
4
Node 1
CPU
PLX
PLX
5
6
7
8
Node 2
Lower Level Communicator (Spans Two Nodes)

Figure 7. Hierarchical DL-Aware Reduction Design with a Chain-Binomial Combination

To reinforce the need for multi-level communicator design, we now present a mathematical model that governs and verifies the performance benefits of our proposed reduce design alternatives.

Let,
P = number of processes
n = number of chunks
$t(b)$ = time to reduce buffer of size b
$t(c)$ = time to reduce buffer of size c [$c = b/n$]
And,
$T(CC)$ = Total Time for Chunked Chain Algorithm
$T(Bin)$ = Total Time for Binomial Tree Algorithm

Then, the total time for reducing a buffer of size b for P processes for both approaches can be understood by the following equations:

$T(Bin) = log(P) * t(b)$ .... (1)
$T(CC) = (P-1) * t(c) + (n-1) * t(c)$
$T(CC) = (n+P-2) * t(c)$ .... (2)

Analyzing equations (1) and (2) for different number of processes P and buffer size b, we observe:
for small P and large b, $T(CC) << T(Bin)$
But,
for large P and small b, $T(CC) >> T(Bin)$

Thus, the ideal choice is a trade-off between the two algorithms as we can clearly see that P becomes a significant factor in the time of reduction for a chunked chain (CC) while $t(b)$ becomes the dominant time factor for the binomial tree reduction (Bin) as we move towards larger b. Hence, we need a hybrid of the two approaches where the final design is both skew-tolerant (process count) as well as size-tolerant (buffer size).

To implement the above schemes in an efficient manner that provides best performance for all cases, we experimentally determine the ideal P and b for each of the cases and then apply the aforementioned two-level communicator design to realize an efficient and skew-tolerant reduction design. The chunked-chain algorithm is essentially a single-sided pipeline that transfers and overlaps the communication and reduction of successive chunks of a large buffer. The last process in the chain divides the buffer into n chunks and sends the chunk to the process on the left. The receiving process reduces this chunk with its own chunk and forwards it to its left neighbor. This process continues till the chunk reaches the root process. Hence, the entire chain is a directed chain of chunks starting from the last process and terminating at the root process.

For buffer sizes greater than eight megabytes (8M), we observed that chunked chain (CC) performs much better than the binomial tree (Bin) regardless of the number of chunks. However, the performance of CC degrades as we increase the number of processes (P) in the chain. Experimental evaluation for our platform also suggests that eight is the ideal P for CC approach. The benefits start to decrease beyond P >8. Thus, we resort to a two-level design that can scale beyond 8 GPUs. The addition of two-level communicator allows us to scale well beyond 8 GPUs even if we apply a chain of chain approach, i.e., both the upper level communicator and lower level communicator use CC as the desired algorithm. However, as expected, the two-level chains can only scale to a process count of 64 (eight being the ideal size of one chain). For larger process counts, we utilize a hybrid of the two approaches where the upper level communicator uses a binomial tree based reduction while the lower level communicator uses the chunked-chain algorithm as shown in Figure 7. We note that, we have tuned the selection of the appropriate combination, chain-of-chain or chain-binomial depending on the system size. Further, it is worth noting that the proposed design is very extensible. Thus, in future, we can exploit multi-level combinations like chain-of-chain combined with a top level binomial for very large scale reductions. The hierarchical reduction designs have been made publicly available with the MVAPICH2-GDR 2.2 [41] release. Detailed performance evaluation is presented in Section 6.5.

6. Performance Evaluation

6.1 Computational Testbed and Environment

We have used a Cray CS-Storm based GPU cluster called KESCH [2] located at the Swiss National Supercomputing Center. This a dense GPU cluster consisting of 12 hybrid nodes each containing 8 NVIDIA K-80 GK210GL GPUs. This makes a total of 192 GPUs for the 12 nodes and just 24 conventional CPUs. Please note that each K-80 is a dual-GPU card which means that a total of 16 CUDA devices are available in one node. The nodes are connected with the InfiniBand Connect-IB dual-port network cards (HCAs). We refer to this cluster as Cluster-A. To illustrate that the proposed designs are generic and applicable for different types of GPU clusters, we have also used a small 20 nodes GPU cluster containing one K-80 GPU per node. The total number of GPUs we can utilize are 40 for this cluster. The nodes are connected to each other using InfiniBand EDR HCAs. This cluster is henceforth referred as Cluster-B.

6.2 Evaluation Metrics and Data Sets

In DL community, scientists and DL model designers strive for achieving the best accuracy and minimum loss. To achieve this, various hyper-parameters like base learning rate (lr), weight decay, momentum, and learning policy are tuned for the best accuracy. However, training time is one of most crucial metrics for comparing performance of DL frameworks. Large-scale DL networks like AlexNet and CaffeNet can take several days of training to reach the desired accuracy. Thus, training time becomes a decisive factor in using a DL framework. In this paper, we present the parallel training time trend observed for a fixed number of iterations. This can help DL scientists to tweak and tune hyper-parameters. We note that challenges associated with Hyper-parameter Search (HS) are at a different level of granularity compared to S-Caffe. HS parallelization is performed at the SGD algorithm's level while the focus of S-Caffe is on synchronizing parallel SGD solvers for distributed training.

In order to cover a wide variety of networks and data sets, we have used **GoogLeNet** and **AlexNet** for comparing performance of parallel training on the ImageNet dataset. We use the term ImageNet in this paper to refer to the widely used ILSVRC 2012 [5] data set. ImageNet contains over a million images spread across 1,000 categories. For smaller scale, we present results using the CIFAR10 network as well. We present **strong** scaling results, where the batch-size is divided by the number of GPUs (or solvers). E.g. if we specify a batch-size of 1,024 for 32 GPUs, the effective batch-size for a single GPU becomes 32 (as 1,024/32 = 32). For weak scaling, the batch-size of 1,024 remains constant for each of the GPUs. These results are not presented but can be obtained using the public version of S-Caffe [46] by specifying *-scal weak* command line option. We have used default hyper-parameters (lr, momentum, etc.) provided by the reference Caffe and CNTK models for all the experiments. Caffe reports accuracy during the Testing phase only. Therefore, we obtained accuracy for CIFAR10 only as it is possible to train and test fairly quickly. We observed no difference in accuracy between Caffe and S-Caffe. Similarly, the decrease in loss (reported every iteration) for AlexNet and GoogLeNet was similar to the multi-GPU training of Caffe. This validates that S-Caffe's distributed training indeed works as expected.

6.3 S-Caffe: Primary Highlights

GoogLeNet: We first present the primary benefits observed for the proposed S-Caffe co-designs for the GoogLeNet model. In Figure 8, we present strong scaling results for GoogLeNet network up to 160 GPUs. The numbers presented in this figure compare the performance of Caffe with different configurations of S-Caffe. **S-Caffe-L** means that we have utilized LMDB database to access the ImageNet data set. **S-Caffe**, on the other hand, exploits the parallelism in Lustre file system and directly accesses image files through the ImageDataLayer. Caffe, being a single process code, has been run with LMDB as it does not require parallel access. In fact, ImageDataLayer with Caffe suffers degradation, so we have omitted those results. We report speedups of 3.3x over 16 GPUs and 2.5x over 32 GPUs for GoogLeNet training with 128 and 160 GPUs, respectively. We note that this scaling, unlike conventional HPC applications, is considered very promising as large DL models are notoriously hard to scale. This has been a major problem for the DL community as highlighted in [1, 9].

The numbers presented are with different batch sizes written in parentheses. We ran the training for 100 iterations. Beyond 64 GPUs, we experienced severe degradation or race conditions for LMDB. Thus, we present ImageDataLayer based results for process counts larger than 64. Missing data points are for the cases where solvers ran out of memory. It is important to note that a larger batch divided over a smaller number of solvers renders the effective batch-size to be too large to fit inside a GPU's memory. Thus, an added motivation to develop multi-process frameworks like the proposed S-Caffe is to enable training using very large batch-sizes for current as well as upcoming DL models that are expected to deal with much larger data sets and networks.

CIFAR10 Quick Solver: Figure 9 highlights the performance of CIFAR10 Quick Solver, provided as a reference solver for CIFAR10 dataset in the Caffe repository. We note that Caffe numbers are only available for 1 node (up to 16 GPUs). Since S-Caffe can scale beyond one node, the results presented here are for a total of 64 GPUs on 4 nodes. We observed a decent scale-out performance considering the size of CIFAR10 dataset. We run the solver for 1,000 iterations and the batch-size is for 8,192. It is pertinent to mention that this is a dense GPU system, so the results presented are for 4 nodes only. On a conventional GPU cluster, 64 GPUs can mean anywhere between 32 and 64 nodes. We observed similar speedup for CIFAR10 on Cluster-B as well but we have omitted the graph to save space. The overall

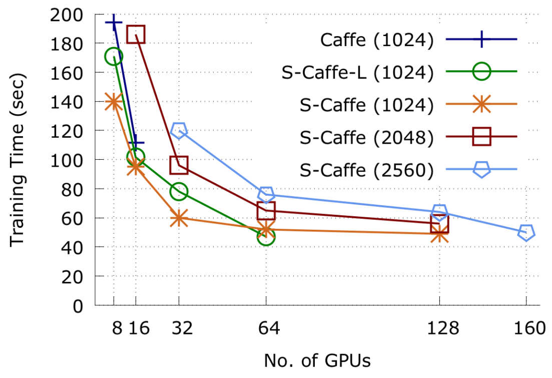

Figure 8. GoogLeNet: Comparison of S-Caffe (up to 160 GPUs) and Caffe (up to 16 GPUs) on Cluster-A

speedup S-Caffe has achieved is 32x over single GPU training. We note that S-Caffe and Caffe perform very similar up to 16 GPUs as CIFAR-10 is a compute-intensive model with small-scale communication between solvers. Thus, it is an encouraging highlight that S-Caffe does not suffer any overhead. This trend was reversed and we saw benefits for S-Caffe over Caffe when training GoogLeNet because it is a communication-intensive model.

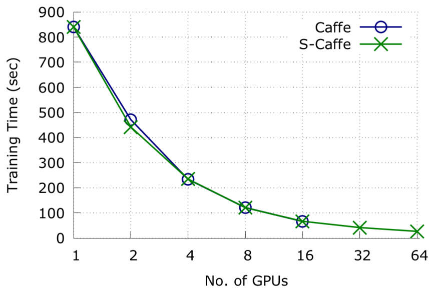

Figure 9. CIFAR10: Comparison of S-Caffe (up to 64 GPUs) and Caffe (up to 16 GPUs) on Cluster-A

Discussion: We note that the results presented above are on the fastest Tesla GPUs available i.e. Kepler K-80, which provides at least 3X faster performance than the K-20x [13] cards. It is pertinent to mention that the largest state-of-the-art scaling presented in [30] is for 128 K-20x GPUs. Thus, the scaling we present here is different and not directly comparable to K-20x level scaling because of the much faster GPUs used for parallel training. As Fire-Caffe [30] is not publicly available for download, we are currently unable to compare our results with them. We plan to do so when the software becomes available.

6.4 Performance Comparison of S-Caffe, CNTK, and Inspur-Caffe

In order to provide a comparison of S-Caffe with different DL frameworks that support distributed training, we have identified three main frameworks: 1) Inspur-Caffe, which is an MPI-based parameter-server implementation, 2) TensorFlow, which is a gRPC based distributed training framework, and 3) Microsoft CNTK, which is also an MPI-based framework. Due to some technical difficulties [27], we were not able to run TensorFlow on our cluster. Therefore, we only compare S-Caffe with Inspur-Caffe and CNTK. We have used the AlexNet network and Cluster-B for this comparison. Results are presented in Figure 10. We note that we present this result in Samples Per Second (SPS) instead of the training time. A higher SPS denotes better performance. For Inspur-Caffe and S-Caffe, we have used LMDB for file reading while default file reading mechanism was used for CNTK. Inspur-Caffe results are shown only for 2 and 4 GPUs as it didn't run for less than 2 GPUs and more than 16 GPUs. For other cases, we were not able to report numbers because the execution hangs after completing a few iterations. We plan to report this to Inspur-Caffe developers. For CNTK, we used their 32-bit SGD design that uses MPI-based communication among workers. A detailed discussion of differences between CNTK, TensorFlow, and S-Caffe can be seen in Section 7.

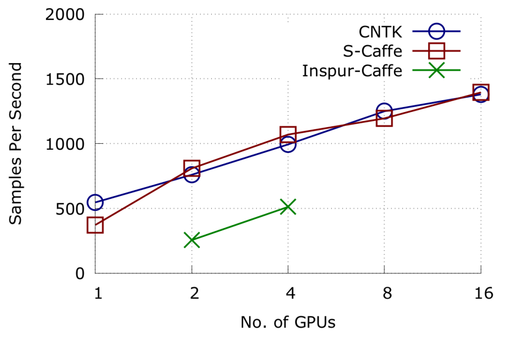

Figure 10. AlexNet: Comparison of S-Caffe, CNTK, and Inspur-Caffe (up to 16 GPUs) on Cluster-B

6.5 Performance of Hierarchical Reductions

We now present the performance evaluation of Hierarchical Reduce (HR) designs. These have been tuned for best performance for all message sizes and process counts. The results are labeled with appropriate design/implementation category. Lower level chain communicator (C) combined with upper level binomial (B) is labeled as **CB**. Upper level chain combined with lower level chain is called **CC**. **MV2** is used for existing MVAPICH2 reduce implementation. **HR (Tuned)** is the new tuned design that builds on top of the tuning infrastructure in MVAPICH2 and efficiently uses the fastest combination for the desired message size and process count range. The numbers in the legend e.g. (-4) represent the size of the chain used. For instance, **CC-4** means that Chain algorithm is applied for both upper and lower level communicators. This micro-benchmark level evaluation has been performed using the OMB [42] suite and is presented in Figure 11.

In addition, we separately present the performance comparison for our proposed design (HR) versus MVAPICH2 and OpenMPI using a log scale for the sake of clarity. Figure 12 shows this comparison. It worth noting that we use OpenMPI v1.10.2 and MVAPICH2 2.2RC1 with their performance features enabled including GDR and IPC (and GDRCOPY for MVAPICH2). As can be seen from the figure, our proposed design is almost 3X faster than MVAPICH2 and up to 133X faster than OpenMPI.

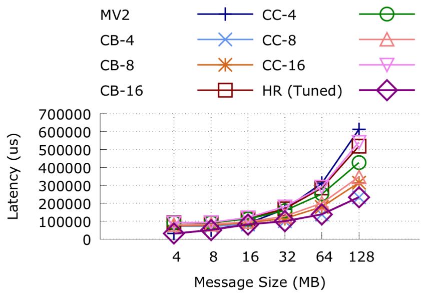

Figure 11. Performance for 160 Processes (GPUs): MVAPICH2, Chain-Binomial, Chain-Chain, and Proposed HR (Tuned) on Cluster-A

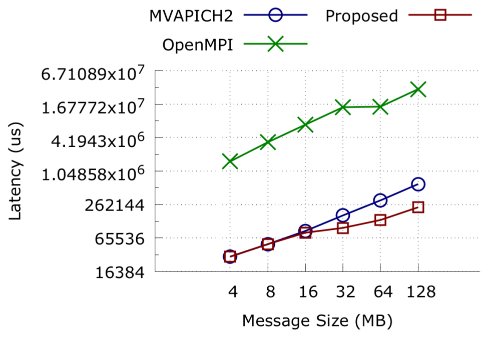

Figure 12. Performance Comparison: MVAPICH2, OpenMPI, and Proposed on Cluster-A

6.6 Impact of Various S-Caffe Co-designs

We now provide further insights on the performance and scalability of the proposed S-Caffe framework and its different co-designs. We compare CUDA-Aware MPI design (SC-B), the overlapped data propagation design (SC-OB), and the overlapped gradient aggregation (SC-OBR) + hierarchical reduce (HR) co-design. We note that numbers presented in this section are not comparable to the overall numbers presented in Section 6.3 as the configuration, batch-size, and number of iterations are different for these runs.

Overlapped Data Propagation (SC-OB): Benefits of SC-OB are shown in Figure 13 by comparing the time required for data propagation and F/B compute passes. We can see that SC-OB co-design provides an excellent overlap of the communication and hides the large latency behind compute intensive Forward pass of layers. Time taken by the reduce phase is not shown as it does not have any effect on the SC-OB design. We report up to 15% improvement for SC-OB design.

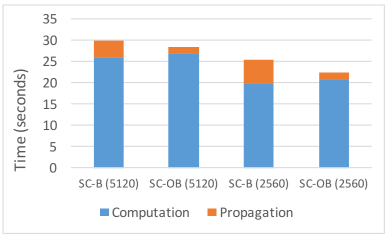

Figure 13. Comparison of SC-B with SC-OB

Overlapped Gradient Aggregation (SC-OBR) and HR:

To further illustrate the benefits we achieved by using HR co-designs, we present a comparison of SC-B vs. SC-B (+HR) in Table 2. SC-B (+HR) means that the hierarchical reduction co-design has been applied to improve performance. Furthermore, the configuration of communicator and chain-size has been varied to illustrate the details. We achieve a 2.3x speedup for SC-B (+HR) with optimal chain-size and communicator selection (CB-8). For SC-OBR, we saw a 20% improvement over SC-B for CaffeNet on 8 GPUs and 12% improvement for 16 GPUs.

Algorithm / Communicator	SC-B SC-B (+HR)	Aggregation Time	Total Time	Speedup for Aggregation	Overall Speedup
N/A	SC-B	40.6	113.6	1	1
CC-8	SC-B (+HR)	28.6	101.6	1.47	1.11
CB-4	SC-B (+HR)	19.8	92.8	2.04	1.22
CB-8	SC-B (+HR)	17.6	90.6	2.3	1.25

Table 2. Comparison of SC-B vs. SC-B with HR

7. Related Work

We now compare S-Caffe with prevalent research and development efforts. FireCaffe [30] is one of the most recent efforts in scaling out Caffe using MPI. It discusses scaling of DL models to multi-GPU clusters with prime focus being on observing the behavior of model parameters e.g., batch-size in conjunction with training accuracy for large-scale runs. However, S-Caffe provides a systems' perspective of new co-designs and highlights strategies to accelerate and better scale communication phases in Caffe. Inspur-Caffe [31] is an MPI-based Caffe fork that exploits parameter-server approach with stale asynchronous gradient updates. S-Caffe, on the other hand, uses synchronous gradient aggregation using a reduction-tree. Microsoft CNTK [12] uses MPI to provide distributed training support as well. CNTK, however, does not take advantage of CUDA-Aware MPI and/or co-designed DL-Aware collectives like the proposed reduce (HR). As reported in Section 6, CNTK and S-Caffe achieve comparable performance.

Distributed frameworks can also exploit task-based programming models and scheduling frameworks. StarPU [4] is one such framework to schedule tasks on heterogeneous systems where some tasks are scheduled on GPUs and some on the CPU. However, in S-Caffe, all the computation (solver) is performed on the GPUs in a deterministic fashion. Hence, StarPU will always provide the same configuration with little benefits. In addition, StarPU may still require the MPI communication to be defined. Google TensorFlow is another widely used DL framework that uses a task-based model where computation can be performed on either a CPU or a GPU using the *tf.device* abstraction. The initial release of TF only supported single node training but recent versions (0.8 and onwards) provide distributed training support using the Google RPC [26] library. An MPI version of TensorFlow has recently been proposed in [48] but the current version only supports CPU-based training. Thus, the design challenges for S-Caffe and [48] are different because the primary contribution and focus of S-Caffe is on GPU-based training and that too for large DL models like AlexNet.

Designs presented in GeePS [23] focus on DL models that don't fit well in a GPU's memory as they used K20 GPUs with limited memory (5 GB). However, recent GPUs by Nvidia, like the K-80 and P100 have 12 and 16 GB memory, respectively. Thus, the memory problem is being solved at a hardware level as well. In contrast, we are focusing on scaling distributed training for models that 'do fit' on a GPU's memory.

8. Conclusion

With an increasing interest in accelerating Deep Learning frameworks through GPU-based systems, in this paper, we tackled the challenge of designing a scalable and distributed DL framework called S-Caffe. Using a co-design approach, we fully exploit the resources available in modern GPU clusters to accelerate training of both medium and large scale DL networks. S-Caffe is a culmination of an extensive co-design process that addresses bottlenecks in the sequential phase-based workflow of Caffe. Through data propagation co-design (SC-OB) that provides maximal overlap using MPI-3 Non-blocking Collectives, we reported 15% improvement over the basic CUDA-Aware MPI design (SC-B). We pushed the envelope of performance further with aggressive co-design of gradient aggregation through a helper-thread based overlap of communication and computation (SC-OBR) and an MPI runtime level hierarchical reduction design (HR). Exploiting SC-OBR and HR, we achieve 20% improvement for GoogLeNet-based training on 160 GPUs. We report encouraging speedups for two different solvers; 33X speedup over 1 GPU training for CIFAR10 quick solver utilizing 64 GPUs and 3.3X speedup over 32 GPU training for CaffeNet solver using 128 GPUs. S-Caffe provides comparable performance to Microsoft CNTK for the AlexNet model. S-Caffe has been made publicly available under the OSU HiDL project [46].

Acknowledgments

We would like to thank the anonymous reviewers for the invaluable feedback that helped us in improving this paper. We also extend thanks to our colleague Hari Subramoni for his significant efforts in making the public release of S-Caffe possible. This research is supported in part by National Science Foundation grants #ACI-1450440, #CNS-1513120, and #CCF-1565414.

References

[1] Caffe: Multi-GPU Usage and Performance. https://github.com/yahoo/caffe/blob/master/docs/multigpu.md.

[2] KESCH: Cray CS-Storm System. http://www.cscs.ch/computers/kesch_escha/index.html.

[3] Intel Caffe. https://github.com/intelcaffe.

[4] A Unified Runtime System for Heterogeneous Multicore Architectures. http://starpu.gforge.inria.fr.

[5] ILSVRC2012 Dataset. http://image-net.org/challenges/LSVRC/2012/index, 2012. [Online; accessed Dec-2016].

[6] Caffe Website. http://caffe.berkeleyvision.org/, 2015. [Online; accessed Dec-2016].

[7] CaffeNet. http://papers.nips.cc/book/advances-in-neural-information-processing-systems-25-2012, 2015. [Online; accessed Dec-2016].

[8] GPU Direct RDMA. http://docs.nvidia.com/cuda/gpudirect-rdma/, 2015. [Online; accessed Dec-2016].

[9] HPC: Powering Deep Learning. http://computing.ornl.gov/workshops/SMC15/docs/bcatanzaro_smcc.pdf, 2015. [Online; accessed Dec-2016].

[10] LMDB. http://symas.com/mdb/, 2015. [Online; accessed Dec-2016].

[11] Nvidia Development Platform for Autonomous Cars. http://www.nvidia.com/object/drive-px.html, 2016. [Online; accessed Dec-2016].

[12] CNTK. http://www.cntk.ai/, 2016. [Online; accessed Dec-2016].

[13] Nvidia GPUs Comparison. http://www.extremetech.com/computing/194391-nvidias-new-tesla-k80-doubles-up-on-gpu-horsepower, 2016. [Online; accessed Dec-2016].

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. *Software available from tensorflow. org*.

[15] J. A. Anderson, C. D. Lorenz, and A. Travesset. General Purpose Molecular Dynamics Simulations Fully Implemented on Graphics Processing Units. *Journal of Computational Physics*, 227(10):5342–5359, 2008.

[16] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah. Comparative Study of Caffe, Neon, Theano, and Torch for Deep Learning. *CoRR*, abs/1511.06435, 2016.

[17] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio. Theano: New Features and Speed Improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[18] D. Case, J. Berryman, R. Betz, D. Cerutti, T. Cheatham III, T. Darden, R. Duke, T. Giese, H. Gohlke, A. Goetz, et al. AMBER 2015. *University of California, San Francisco*, 2015.

[19] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 571–582, Berkeley, CA, USA, 2014. USENIX Association. ISBN 978-1-931971-16-4. URL http://dl.acm.org/citation.cfm?id=2685048.2685094.

[20] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew. Deep Learning with COTS HPC Systems. In *Proceedings of the 30th international conference on machine learning*, pages 1337–1345, 2013.

[21] R. Collobert, S. Bengio, and J. Mariéthoz. Torch: A Modular Machine Learning Software Library. Technical report, IDIAP, 2002.

[22] Cray. http://docs.cray.com/books/004-3689-001/html-004-3689-001/004-3689-001-toc.html, 2016. [Online; accessed Dec-2016].

[23] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 4:1–4:16, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4240-7. doi: 10.1145/2901318.2901323. URL http://doi.acm.org/10.1145/2901318.2901323.

[24] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large Scale Distributed Deep Networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[25] 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*, pages 248–255. IEEE, 2009.

[26] Google. Google's Remote Procedure Call Library (gRPC). http://www.grpc.io, .

[27] Google. Distributed TensorFlow: Github Issues. https://github.com/tensorflow/models/issues/698, .

[28] R. L. Graham, S. Poole, P. Shamis, G. Bloch, N. Bloch, H. Chapman, M. Kagan, A. Shahar, I. Rabinovitz, and G. Shainer. Overlapping Computation and Communication: Barrier Algorithms and ConnectX-2 CORE-Direct Capabilities. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010.

[29] T. Hoefler, A. Lumsdaine, and W. Rehm. Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI. In *Supercomputing, 2007. SC'07. Proceedings of the 2007 ACM/IEEE Conference on*, pages 1–10. IEEE, 2007.

[30] F. N. Iandola, K. Ashraf, M. W. Moskewicz, and K. Keutzer. FireCaffe: Near-Linear Acceleration of Deep Neural Network Training on Compute Clusters. *arXiv preprint arXiv:1511.00175*, 2015.

[31] Inspur. https://github.com/Caffe-MPI/Caffe-MPI.github.io, 2016.

[32] J. Dean. Keynote: Large Scale Deep Learning.

[33] 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.

[34] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.

[35] A. Krizhevsky and G. Hinton. Learning Multiple Layers of Features from Tiny Images, 2009.

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[37] S. Lee, S. Purushwalkam, M. Cogswell, D. J. Crandall, and D. Batra. Why M Heads are Better than One: Training a Diverse Ensemble of Deep Networks. *arXiv*, 2015. URL http://arxiv.org/abs/1511.06314.

[38] M. Lin, Q. Chen, and S. Yan. Network in Network. *arXiv preprint arXiv:1312.4400*, 2013.

[39] Lustre. Parallel File System. http://lustre.org.

[40] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon. TOP 500 Supercomputer Sites. http://www.top500.org.

[41] MVAPICH2: MPI over InfiniBand, 10GigE/iWARP and RoCE. https://mvapich.cse.ohio-state.edu/.

[42] Network Based Computing Laboratory. OSU Micro-Benchmarks. http://mvapich.cse.ohio-state.edu/benchmarks/, 2016.

[43] C. Nvidia. Programming Guide, 2008.

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

[45] 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*, pages 1–9, 2015.

[46] The HiDL Team. High Performance Deep Learning (HiDL) Project. http://hidl.cse.ohio-state.edu.

[47] The Open MPI Development Team. Open MPI : Open Source High Performance Computing. http://www.open-mpi.org.

[48] A. Vishnu, C. Siegel, and J. Daily. Distributed TensorFlow with MPI. *arXiv preprint arXiv:1603.02339*, 2016.

[49] D. Wang, A. Khosla, R. Gargeya, H. Irshad, and A. H. Beck. Deep Learning for Identifying Metastatic Breast Cancer. *ArXiv e-prints*, June 2016.