An In-depth Performance Characterization of CPU- and GPU-based DNN Training on Modern Architectures

Conference Paper · November 2017

DOI: 10.1145/3146347.3146356

Ammar Ahmad Awan
The Ohio State University
26 PUBLICATIONS **148** CITATIONS

Hari Subramoni
The Ohio State University
169 PUBLICATIONS **1,220** CITATIONS

D.K. Panda
The Ohio State University
678 PUBLICATIONS **9,827** CITATIONS

An In-depth Performance Characterization of CPU- and GPU-based DNN Training on Modern Architectures

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

Hari Subramoni
Dept. of Computer Science and Engg.
The Ohio State University
subramon@cse.ohio-state.edu

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

ABSTRACT

Traditionally, Deep Learning (DL) frameworks like Caffe, TensorFlow, and Cognitive Toolkit exploited GPUs to accelerate the training process. This has been primarily achieved by aggressive improvements in parallel hardware as well as through sophisticated software frameworks like cuDNN and cuBLAS. However, recent enhancements to CPU-based hardware and software has the potential to significantly enhance the performance of CPU-based DL training. In this paper, we provide a complete performance landscape of CPU- and GPU-based DNN training. We characterize performance of DNN training for AlexNet and ResNet-50 for a wide-range of CPU and GPU architectures including the latest Intel Xeon Phi (Knights Landing) processors and NVIDIA Pascal GPUs. We also present multi-node DNN training performance results for AlexNet and ResNet-50 using Intel Machine Learning Scaling (MLSL) Library and Intel-Caffe. In addition, we provide a CPU vs. GPU comparison for multi-node training using OSU-Caffe and Intel-Caffe. To the best of our knowledge, this is the first study that dives deeper into the performance of DNN training in a holistic manner yet provides an in-depth look at layer-wise performance for different DNNs. We provide multiple key insights: 1) Convolutions account for the majority of time (up to 83% time) consumed in DNN training, 2) GPU-based training continues to deliver excellent performance (up to 18% better than KNL) across generations of GPU hardware and software, and 3) Recent CPU-based optimizations like MKL-DNN and OpenMP-based thread parallelism leads to excellent speed-ups over under-optimized designs (up to 3.2X improvement for AlexNet training).

KEYWORDS

High-Performance Computing, Deep Learning, Caffe, Unified Memory, Pascal Architecture

ACM Reference Format:
Ammar Ahmad Awan, Hari Subramoni, and Dhabaleswar K. Panda. 2017. An In-depth Performance Characterization of CPU- and GPU-based DNN Training on Modern Architectures. In *Proceedings of MLHPC'17: Machine Learning in HPC Environments (MLHPC'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3146347.3146356

1 INTRODUCTION

Recent advancements in Artificial Intelligence (AI) fueled by the resurgence of Deep Neural Networks (DNN) have resulted in various Deep Learning (DL) frameworks like Caffe [15], TensorFlow [8], and Microsoft CNTK [19]. DNNs have found widespread applications in classical areas like Image Recognition, Speech Processing, Textual Analysis, as well as in more recent areas like Cancer Detection, Medical Imaging, and even Autonomous Vehicle systems. DNNs and the associated research challenges have become hot topics of interest for both academic research groups as well as for prominent companies like Google, Baidu, Facebook, and Microsoft.

Two driving elements can be attributed to the momentum that DL has gained; first is the public availability of various data sets like ImageNet [10] and CIFAR [17], and second is the widespread adoption of data-parallel hardware like GPUs and accelerators. GPUs with their raw number crunching capabilities (FLOPS) have made the training of DNNs computable. This was just not possible even a few years ago. Today, the community is designing better, bigger, and deeper networks for improving the accuracy through models like AlexNet [18], GoogLeNet [25], and ResNet-50 [12]. The models differ in the order of computational layers but share the common requirement of faster computation and communication capabilities of the underlying system.

Although GPUs have been at the heart of DL resurgence, Intel has recently made advancements in modern CPU hardware as well as developed toolkits like MKL-DNN [14] that promise better performance for DL applications running on CPUs. In this context, Caffe has been the most widely investigated framework for experimentation and research as it has a simple and modular C++ backend. Caffe has seen wide adoption from the community and both NVIDIA and Intel have their own respective Caffe versions. NVIDIA-Caffe [6] and Intel-Caffe [5] are the two main frameworks that we investigate in this paper. We note that Intel-Caffe also supports multi-node training through Message Passing Interface (MPI) primitives via the Intel Machine Learning Scaling Library (MLSL) [13] but neither Berkeley's official version of Caffe [15] nor NVIDIA-Caffe have multi-node support for GPU-based training. Thus, to compare performance for GPU-based multi-node training, we use OSU-Caffe [26], an MPI-based (MVAPICH2-GDR [20]) framework built on top of NVIDIA-Caffe. The brief summary of important features available in different Caffe variants is provided in Table 1.

Due to high throughput support and an architecture designed specifically for data parallel workloads, GPUs are considered a very good match for the computational requirements of DNN training tasks. However, with tailored software support, Intel CPUs are catching up fast on the performance front. However, the challenge

Caffe Variant	Multi-GPU Support	Multi-Node Support	Multi-node Communication
BVLC-Caffe	Yes	No	N/A
NVIDIA-Caffe	Yes	No	N/A
Intel-Caffe	N/A	Yes	Intel MLSL 2017.1.016 (with Intel MPI 2017)
OSU-Caffe	Yes	Yes	MVAPICH2-GDR 2.2

Table 1: Features Available in Different Caffe Variants

is that most performance studies about CPU vs. GPU have been performed by vendors themselves. In some cases, this leads to high-level performance numbers that cannot be understood completely as different versions of the underlying libraries and software/hardware support varies. In this paper, we tackle the broad challenge of *"Can we provide a holistic yet comprehensive view of DNN training performance for a diverse set of hardware architectures including Intel Xeon Phi (KNL) processors and NVIDIA Pascal GPUs?"*

Building on this broad theme, we investigate the following key questions:

- What are the computation and communication characteristics of popular DL workloads?
- How various datasets and networks are handled differently in DL frameworks that execute on CPUs and/or GPUs?
- Can we devise any possible strategies to evaluate the performance of DL frameworks on different compute architectures in a standard manner?
- What are the performance trends that can be observed when only a single GPU or a single CPU/processor is used?
- How does the performance behavior change if certain hardware features like MCDRAM [7] are exploited?
- To what degree can scale-out of DNN training help if multiple nodes for both CPU-based and GPU-based DNN training are utilized?

1.1 Contributions

There have been various studies for GPU-based DNN training but there is a lack of studies that provide a holistic and fair comparison of CPU vs. GPU based DNN training. To the best of our knowledge, this is the first study that dives deeper into the performance of DNN training in a holistic manner yet provides an in-depth look at layer-wise performance for two of the most widely used DNNs. Unlike other benchmarks, we perform a top down analysis of all the software and hardware components involved in DNN training. In this context, we use Caffe as our candidate framework but the findings are generic and can help understand the performance of other frameworks too, especially the ones that take advantage of libraries like cuDNN [4] and MKL-DNN [14]. The broad goal is to understand the impact of different hardware configurations and features like clock speed, number of cores, memory configurations as well as the impact of software libraries like cuDNN and MKL-DNN. To address this, we make the following key contributions:

- We provide an overview of DL frameworks and highlight how a DL framework interacts with other software and hardware components in the system.
- We highlight the key components of one of the most widely used DL framework (Caffe) and its different versions optimized for CPUs, GPUs, and multi-node distributed training.
- We present a detailed performance evaluation of different Caffe variants to offer a holistic view of how a DL framework and its interaction with different BLAS libraries, DNN libraries, and execution hardware impacts the performance of overall DNN training.
- We discuss insights we gained by performing single-node as well as multi-node performance comparison of cutting-edge CPU and GPU hardware along with frameworks that use communication runtimes like MPI and MLSL.

2 BACKGROUND

We describe NVIDIA GPUs and the CUDA programming model followed by a discussion on Intel Xeon processors.

2.1 NVIDIA Pascal GPUs and CUDA

Compute Unified Device Architecture (CUDA) [21] is the programming model for writing GPU-based programs for NVIDIA hardware. The programs can be expressed using the CUDA APIs provided by NVIDIA. The next-generation NVIDIA GPUs, based on the Volta architecture, have been announced but are not yet available. Currently, Pascal GPUs are the most widely used GPUs for DL frameworks and applications. Pascal GP100 GPUs have 56 Streaming Multiprocessors (SM) and a total of 3,840 CUDA cores. The theoretical peak Device to Device copy bandwidth is around 750 GB/sec but we are able to achieve up to 510 GB/sec in our local benchmarks. P100 GPUs introduced 49-bit virtual memory addressing capability that is sufficient to address up to 512 TB of memory. Detailed information about Pascal GPUs and how to get the best performance using its advanced features can be studied from [11].

2.2 Intel Knights Landing (KNL) and Multi-/Many-core Systems

Intel's latest iteration of processors include the next-generation Xeon Phi architecture, which is also known as the Intel Knights Landing (KNL) architecture. It is a many-core processor with 68 physical cores where each core has support for a total of four hardware threads. Theoretically, KNL can be considered as a 272 core processor. However, in practice, most applications will have some optimized core to process ratio for their programs to extract the best performance. Intel KNL is the 2nd generation Xeon Phi architecture. The details of the processor appear in [24]. KNL processors also take advantage of the latest high bandwidth memory hardware called the MCDRAM [7].

3 UNDERSTANDING DL FRAMEWORKS AND IMPACT OF EXECUTION ENVIRONMENTS

First, we discuss the overall architecture of DL frameworks. Next, we explain how DL applications and frameworks interact with underlying libraries and parallel hardware. Finally, we discuss the design of Caffe and how it realizes different components needed for orchestrating DL operations.

3.1 Architecture of Deep Learning Frameworks

Deep Neural Networks (DNN) have gained significant traction in the CS and AI communities alike. Starting with AlexNet [18], a variety of DNN architectures such as GoogLeNet, ResNet, and others have arrived in a very short span of time, each offering better feature detection and higher accuracy over the other. DNNs have several architectural variations ranging from the number of layers, filter dimensions, etc. However, all share the common requirement of the need for speed through better hardware and system software. DNN training broadly comprises of iterations over two compute intensive phases; the feed forward phase for training the network using the training samples and the back-propagation phase (or simply Backward pass) to perform gradient descent and thus converging on a well-trained model.

In general, all the DNN structures try to model high-level abstractions in a layered manner. DL frameworks use a cascade of many layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. The learning process is supervised in the sense that ground truth labels are provided to calculate the loss (or the distance) in the feed-forward phase. However, it can be considered unsupervised in the context of feature extraction as DNNs automatically infer features unlike classical machine learning where features were developed/tuned by hand. The network or the DL model is composed of layers in most DL frameworks; layers can be considered as some grouping or set of operations. Some frameworks like Caffe use the Layer abstraction, which is at a higher level than the Operations (or Operators) abstraction used by others like Caffe2 [2] and TensorFlow [8]. DNN architectures are typically maintained in a configuration or a protocol buffer file, which is utilized at runtime by the DL framework to create an in-memory representation of the DNN.

For extending the training process to multiple CPU cores or GPUs; two major approaches have been proposed [16]; *Data Parallelism* and *Model Parallelism*. In Data Parallelism, 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. Model parallelism, on the other hand, requires splitting the model across different processing elements and same data is used to train the model. In this paper, we focus on data parallelism employed by both Intel-Caffe as well as OSU-Caffe.

3.2 Understanding the Execution Environments for DL Frameworks

Most of the studies that present performance evaluation of DL frameworks try to compare the performance at a very high level. This may not provide any insights into performance behavior of the framework rather it actually is reporting the difference of performance that may have been caused by different versions of underlying libraries and hardware architectures. To address this, we provide a complete picture of how a DL framework exploits the underlying helper libraries like cuBLAS [3], MKL-DNN [14], and cuDNN [4] in Figure 1. To better extract performance from the prevalent hardware architectures like Pascal GPUs and KNL processors, these helper libraries need to be optimized in an architecture-specific manner. It is pertinent to mention that the the performance evaluation we present in Section 4 clearly suggests that fundamental operations like Convolutions are actually the key to getting better performance. In essence, all DL frameworks will usually depend on the underlying libraries like BLAS and DNN. In this context, if generic libraries are used, performance degradation can be significant. However, if optimized versions are utilized for the appropriate combination of hardware and software architectures, similar performance can be extracted from both CPUs and GPUs.

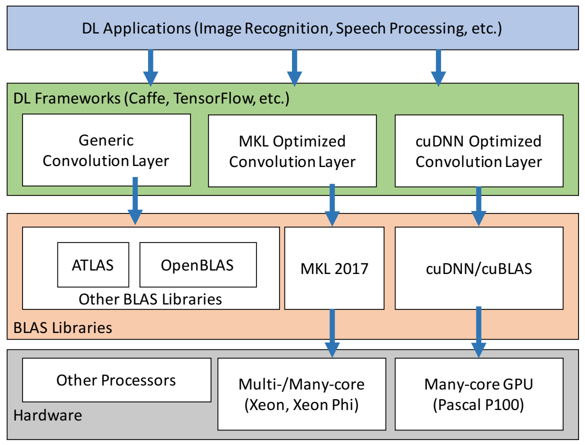

DL Applications (Image Recognition, Speech Processing, etc.)
DL Frameworks (Caffe, TensorFlow, etc.)
Generic Convolution Layer
MKL Optimized Convolution Layer
cuDNN Optimized Convolution Layer
ATLAS
OpenBLAS
Other BLAS Libraries
MKL 2017
cuDNN/cuBLAS
BLAS Libraries
Other Processors
Multi-/Many-core (Xeon, Xeon Phi)
Many-core GPU (Pascal P100)
Hardware

Figure 1: Overview of Execution Environments for Caffe-based DL Frameworks

3.3 Caffe: Design and Implementation

We now define major design and implementation components of the Caffe framework; the Net (or the DNN), the Layers, the Solvers, and how the Forward and Backward phases utilize these components in a cohesive fashion to orchestrate the training process. We discuss them at a higher level in this paper but detailed descriptions can be studied from [1]. A graphical illustration of different components that are involved in parallel training appear in [22]. Figure 2 borrowed from [22] highlights different phases of the solvers and their interaction with layers of a neural network.

The Net: Caffe's Net class is the base class that provides a logical abstraction for a DNN or a DL model. It holds important components of computation, communication, and file access. Net is an in-memory representation of a DNN like AlexNet that is defined using a proto-buffer file. DNN, Net, Network, and DL model are used interchangeably throughout this paper. Different pieces of DNN training are held together by a network. The Net in Caffe contains a vector of Layer objects. The Net object is actually contained in the Solver object to allow different solvers to operate on a network. Caffe uses two Net objects; the training Net and the testing Net. Training Net uses labeled training data during the training phase while the testing Net is used to test the accuracy of the trained model using the validation/testing data that is different from the training data. We will focus on the training network in this study as it is of a higher interest in terms of computation.

The Solvers: Caffe's most fundamental abstraction is the Solver object. It is the solver object that executes and orchestrates the training by utilizing layers of a network. Solvers perform the forward and the backward computation phases as well as perform communication for performing parallel training. Solver class is an abstraction that that can be implemented by different solvers e.g., a Stochastic Gradient Descent (SGD) solver, or an AdaGrad solver.

The Layers (or The Operations): Caffe has Layer objects that are basically an abstraction for a defined set of operations (computation) on the training data. Layer interface can be implemented for any function like convolution, ReLU, pooling, softmax, etc. The DNN designer can provide a proto-buffer description of a layer and its corresponding C++ classes to support computation on the CPU or the GPU using underlying BLAS/DNN libraries.

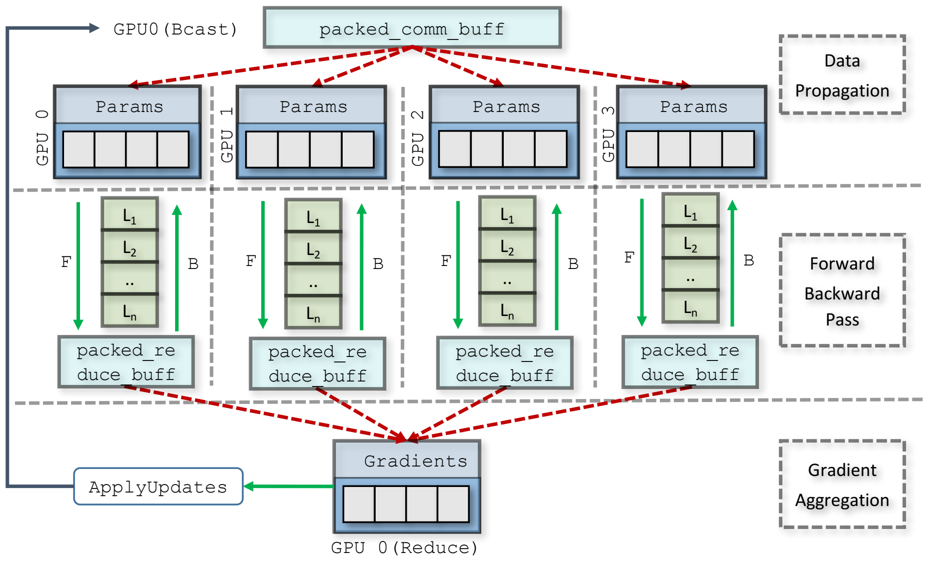


Figure 2: Caffe Architecture: Network, Layers, and Solvers [22]

Some example of Layers in Caffe are: 1) Data Layers that usually feed the Solvers with training data, 2) Convolution Layers that slide the desired filter to create an activation map, and 3) Loss layers that calculate the loss at the end of a Forward pass. From a communication standpoint, Caffe layers contain two important data structures for communication between Solvers; the parameter data used in the Forward pass and parameter gradients calculated in the Backward pass.

Newer frameworks like Caffe2 [2] have removed the abstraction of Layer and resorted to a lower level abstraction called the Operation. The rationale behind this is to provide a more flexible abstraction for input and output variables. Layers can become a strict abstraction and thus do not allow the flexibility to express and design unconventional and emerging neural networks.

Computation and Communication Phases: Caffe uses a training Net in an iterative fashion. Iterations of Caffe are three marked phases; the Forward phase, the Backward phase, and the communication phase at the end of an iteration to update the model. Forward pass begins by computing layer operations starting from the first layer $L1$. At the end of Layer 'n', the forward pass ends with loss calculation. The Backward pass is fed with the loss to calculate the gradients that need to be exchanged if parallel training is to be performed. The packed_reduction_buffer in Figure 2 is used to exchange the gradients among different solver objects using an MPI reduction (e.g. MPI_Reduce, MPI_Allreduce, etc.) operation.

4 PERFORMANCE CHARACTERIZATION OF CPU AND GPU BASED DNN TRAINING

4.0.1 Overall Training Time for AlexNet and ResNet-50. We describe the evaluation platforms in Section 4.1 followed by details of different versions of softwares used for the performance evaluation in Section 4.2. Further, we divide the performance characterization into two broad categories; single node performance for different hardware architectures presented in Section 4.3 and multi-node distributed training performance using Intel MLSL and MVAPICH2-GDR for communication presented in Section 4.4.

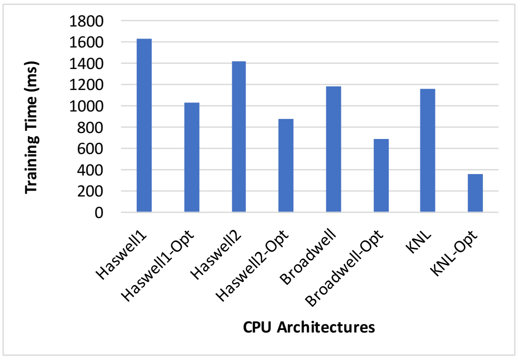


Figure 3: AlexNet: Performance Comparison of Default and MKL-Optimized Caffe

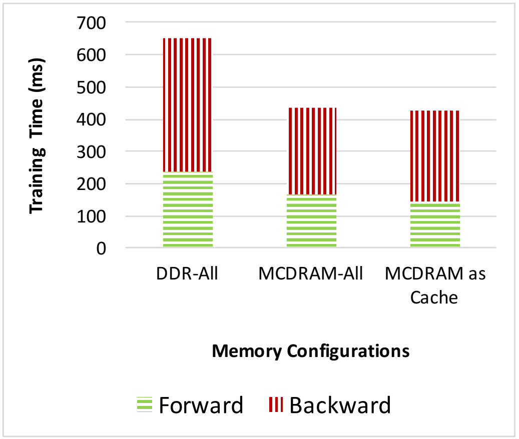
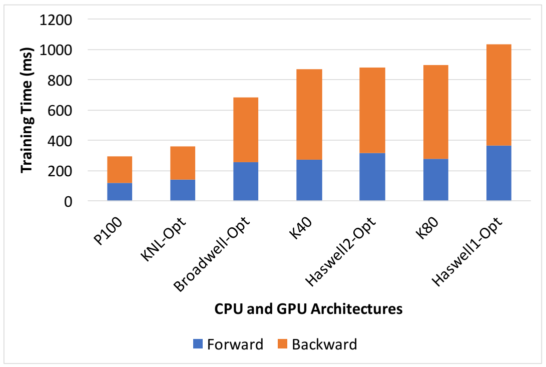
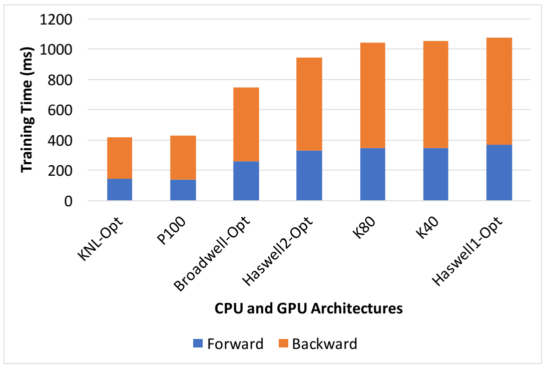


Figure 4: ResNet-50: Impact of Memory Configurations for Training on KNL

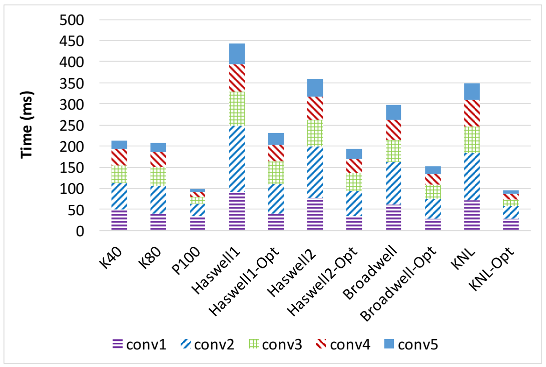


(a) AlexNet: Time Taken by Forward and Backward Pass

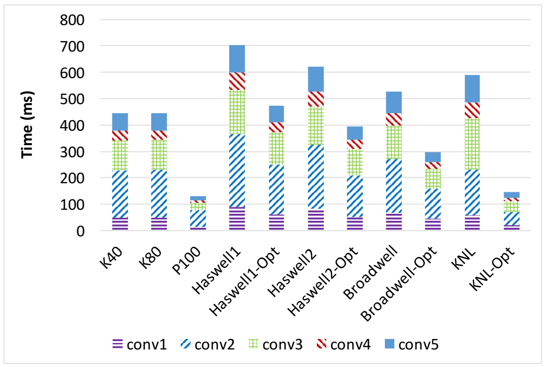


(b) ResNet-50: Time Taken by Forward and Backward Pass

Figure 5: Overall Training Time Reported by Caffe 'time' Benchmark for Different Hardware Architectures



(a) AlexNet: Forward Propagation



(b) AlexNet: Backward Propagation

Figure 6: Layer-wise Breakdown of Performance for Different Hardware Architectures

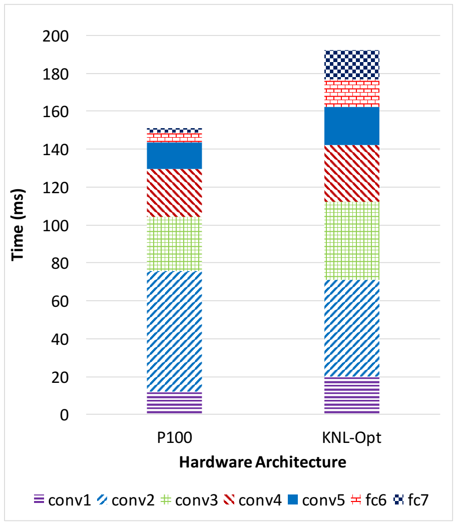


Figure 7: AlexNet: Diving Deeper into Layer-wise Breakdown for P100 and KNL (Backward Pass)

4.1 Evaluation Platform

The details of different CPU and GPU architectures we have utilized for our study along with the labels used in different figures in this section are provided in Table 2. The multi-node study was performed on a local cluster consisting of 20 nodes. Each node has a dual-socket 'Broadwell' processor and a K80 GPU (details in Table 2). We also utilized additional 12 nodes containing 'Broadwell' CPU but these nodes do not have any GPU. Thus, the largest multi-node CPU vs. GPU comparison we provide is for 20 nodes only. As mentioned in Section 2, Pascal P100 GPUs are the latest publicly available GPUs. Volta GPUs have been announced but are not available yet. We plan to update our evaluation as soon as we get access to Volta GPUs.

4.2 Details of Caffe, MKL, and cuDNN versions

We compare the performance of Intel-Caffe for two configurations; default engine called 'CAFFE' and the optimized MKL 2017 engine called 'MKL2017'. We denote the results presented with 'CAFFE' engine as '<ProcessorName> e.g. Broadwell' whereas for MKL2017 engine, we use ' <ProcessorName-Opt> e.g. Broadwell-Opt'. Intel-Caffe allows the computation engine to be changed using a command line flag (`-engine <ENGINE-NAME>`). For CPU-based multi-node experiments, we have used Intel-Caffe and Intel-MLSL.

We use the official BVLC-Caffe for all GPU-based experiments along with CUDA 8 and cuDNN 6. *We note that GPU optimizations performed by NVIDIA-Caffe have been pushed to official BVLC-Caffe so there is no performance difference between these two versions.* For

Name (Label)	Processor Architecture (Description)	No. of Cores	No. of Sockets
Haswell1	Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz	20 = (2*10) Cores	2
Haswell2	Intel(R) Xeon(R) CPU E5-2687W v3 @ 3.10GHz	20 = (2*10) Cores	2
Broadwell	Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz	28 = (2*14) Cores	2
KNL	Intel(R) Xeon Phi(TM) CPU 7250 @ 1.40GHz	68 = (1*68) Cores	1
K40	NVIDIA Tesla K40 11.8GB (Kepler) @ 0.75 GHz	2880 CUDA Cores	N/A
K80	NVIDIA Tesla K80 11.8GB (Kepler) @ 0.82 GHz	2496 CUDA Cores	N/A
P100	NVIDIA Tesla P100-PCIE-16GB (Pascal) @ 1.33 GHz	3584 CUDA Cores	N/A

Table 2: Processor Architectures utilized for Performance Characterization of DNN Training using Caffe Variants

GPU-based multi-node experiments, we have used OSU-Caffe 0.9 with cuDNN 5 and CUDA 8.

Table 3 provides the summary of different software versions we have used for our experiments.

4.3 Single-node Performance Comparison for AlexNet and ResNet-50

First, we present the impact of MKL optimizations on CPU-based training of AlexNet [18] as well ResNet-50 [12]. Next, we present the impact of different memory configurations for ResNet-50 training on KNL CPUs. We then provide the overall performance comparison for all the hardware architectures listed in Table 2. Further, we discuss a detailed layer-wise breakdown for both Forward and Backward passes for AlexNet followed by a separate comparison for KNL and P100 GPU.

We have used the Caffe 'time' benchmark for all the single-node performance numbers. We run 50 iterations of the benchmark and report the average for all the results presented in this paper.

4.3.1 Impact of MKL-optimized Engine for CPU-based Training. Authors of [23] present various comparisons including a K-80 GPU versus a Xeon E5 CPU and show that CPU is up to 10X slower. Clearly, this can be because of using under-optimized version of the framework and/or the underlying libraries. In this context, we see a contrasting picture where performance of default Caffe (-engine CAFFE) for CPU-based training is sub-par but significantly improves by using Intel MKL-2017 optimized designs (-engine MKL2017). As shown in Figure 3, the biggest performance benefit of the MKL optimizations can be observed for KNL.

4.3.2 Impact of MCDRAM for Training on KNL. We now discuss the impact of using the high bandwidth memory on the KNL systems in different configurations. KNL allows memory to be configured in multiple ways; 1) MCDRAM can be used for all allocations in a program, 2) standard DDR can be used for all allocations in a program, and 3) MCDRAM is used as a third level cache. The most commonly used mode is MCDRAM as Cache mode. Figure 4 shows the performance trend for single-node ResNet-50 training when different memory configurations are used on KNL. As we can see, using MCDRAM as Cache provides the best performance. It is important to note that for all allocations on MCDRAM (*labeled MCDRAM-All*), performance is very similar to MCDRAM as Cache mode. On the other hand, forcing all allocations to be on DDR (*labeled DDR-All*) can lead to severe performance degradation. The result has been presented for ResNet-50 only but we see a similar trend for AlexNet training as well.

For the rest of the results presented in this paper, we have used MCDRAM as Cache for all KNL-based experiments.

Comprehensive performance comparisons of all different CPU and GPU architectures listed in Table 2 for training AlexNet and ResNet-50 are provided in Figure 5. The performance is reported as a high level breakdown of overall training time into forward and backward passes. The CPU-based numbers in these figures are reported only for the optimized MKL-2017 engine of Intel-Caffe. KNL-based numbers are presented in MCDRAM as Cache mode.

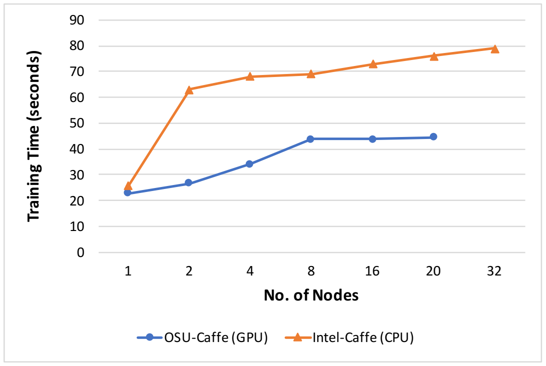
We observe excellent speedups for Intel-Caffe when the optimized MKL 2017 engine is utilized. On the other hand, we can see that despite significant performance improvements, CPU-based training is still slower than the GPU-based AlexNet training on Pascal GPUs. To understand the impact and characteristics of this performance difference, we dive deeper into the layer-wise breakdown of this model in the next sub-section. The results here have been presented using the AlexNet and ResNet-50 networks. It is very interesting to note that K80 and K40, the previous generation GPUs, are much slower compared to Broadwell and KNL CPUs when MKL-optimizations are exploited. For ResNet-50, KNL provides slightly better performance than P100 as shown in Figure 5(b).

4.3.3 Layer-wise Breakdown. The AlexNet model consists of 24 layers including five convolution layers. Based on our performance evaluation, we present the most compute intensive layers, i.e., the convolution layers separately for all combinations of hardware architectures and software optimizations in Figure 6. As DNN training comprises of the forward and backward pass, we illustrate them separately in Figures 6(a) and 6(b). On the other hand, ResNet-50 is a very deep network and has 245 layers with several convolution layers. Because of this complexity, it is very hard to show the layer-wise breakdown for ResNet-50.

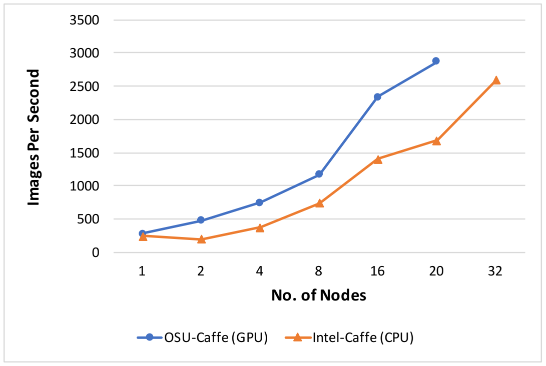
4.3.4 AlexNet: P100 vs. KNL. It is clear from the detailed breakdown that several different layers have been improved by using the optimized MKL2017 engine. However, as mentioned earlier, the meat of performance benefit comes from the highly optimized convolution layers of the MKL 2017 engine. To emphasize the need of better convolution approaches and/or implementations, we illustrate the same results presented earlier but only for Backward propagation on KNL CPU and P100 GPU in Figure 7. The purpose here is to highlight that despite KNL being much faster than other CPUs, it is still performing slower than the P100 GPU. We observe that some of this slowdown is caused by slower fully connected layers *fc6* and *fc7* where P100 is much faster than KNL. Similarly, *conv1* and *conv3* take more time on KNL than P100. On the other hand, *conv2* runs faster on KNL compared to P100.

Caffe Variant	DNN/BLAS Library	Runtime/Compiler	Communication Library
BVLC-Caffe (commit 4efdf7ee49cffefdd7ea099c00dc5ea327640f04)	cuDNN 6 / cuBLAS	CUDA 8 and gcc 4.9.3	N/A
Intel-Caffe (commit 8012927bf2bf70231cbc7ff55de0b1bc11de4a69)	Intel MKL 2017 (2018.0.20170425)	gcc 4.9.3	Intel MLSL 2017.1.016 (with Intel MPI 2017)
OSU-Caffe (0.9)	cuDNN 5 / cuBLAS	CUDA 8 and gcc 4.9.3	MVAPICH2-GDR 2.2

Table 3: Details of Caffe Variants, DNN Libraries, Compiler/Runtime Versions, and Communication Libraries



(a) AlexNet Weak Scaling: Training Time (BatchSize=64)



(b) AlexNet Weak Scaling: Images Per Second (BatchSize=64)

Figure 8: Performance Comparison for AlexNet Multi-node Training on 'Broadwell' CPUs and K-80 GPUs with Intel-Caffe and OSU-Caffe

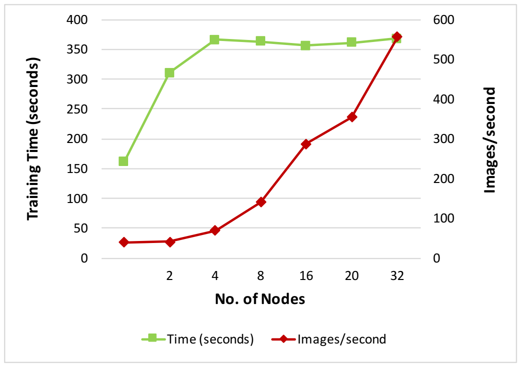


Figure 9: Weak Scaling Performance of Intel-Caffe for ResNet-50 Training

4.4 Multi-node Performance Comparison using Intel-Caffe and OSU-Caffe

Now we present our multi-node weak scaling study using Intel-Caffe and Intel-MLSL for CPU-based training and OSU-Caffe for GPU-based multi-node training. Intel-MLSL is used along with Intel MPI 2017 runtime (packaged with Intel-MLSL). We use two performance metrics; the training time and the number of images processed per second. The performance for AlexNet training is shown in Figure 8. ResNet-50 training performance is illustrated in Figure 9.

4.4.1 AlexNet Multi-node Scaling. Figure 8 shows the overall performance trend for training AlexNet on multiple nodes using Intel-Caffe for CPU-based training and OSU-Caffe for GPU-based training. It is pertinent to mention that both these frameworks have very different designs for multi-node training. Intel-Caffe uses Intel-MLSL that is built on top of MPI primitives while OSU-Caffe uses MPI_Reduce and MPI_Bcast directly [9]. Because of this, it is very hard to provide a direct comparison for these frameworks. However, the purpose of these results is to highlight the scalability trends and identify certain bottlenecks where performance can be improved. As discussed earlier in the single-node performance studies (Figure 6), we observed that Broadwell CPU performs much better than K80 GPU for AlexNet training. However, the trend changes significantly for multi-node training. The sudden jump in training time as we move from one node to two nodes in Figure 8 reveals that there exist certain overheads in either the MPI level or the MLSL level designs that Intel-Caffe exploits for multi-node training. On the other hand, OSU-Caffe doesn't exhibit such sharp degradation when moving from one GPU to a higher GPU count. *The GPU numbers are only provided up to 20 nodes as we only have 20 GPU nodes in our cluster.*

4.4.2 ResNet-50 Multi-node Scaling. ResNet-50 training is illustrated for CPU-based training in Figure 9. For CPU-based training, we see that performance degradation experienced for AlexNet training also exists for ResNet-50 training. In fact, the performance degradation is worse for 2 and 4 nodes case. Clearly, this seems to be a multi-node design issue with Intel-Caffe as this 2X degradation when moving from one node to the other node is not expected. We plan to report this issue to the Intel-Caffe developers. On the other hand, images per second or the throughput still improves as we increase the number of nodes so there is still an overall advantage in using multiple nodes despite the sudden jump in training time.

5 CONCLUSION

An ever increasing interest in Deep Learning has led to a diverse set of approaches for improving performance of DNN training and inference. In this paper, we discussed and analyzed different variants of the popular Caffe framework. We provide an in-depth discussion on how DL frameworks are positioned in the rich stack of software and hardware components that enable DNN training on modern platforms. We observed that both Intel and NVIDIA have their respective optimizations specific to their hardware both at the framework level as well as at the level of underlying math kernel libraries like BLAS and DNN.

We presented three key results for single-node training using CPUs and GPUs: 1) MKL optimizations lead to significant (up to 3.2X) performance improvement for AlexNet training on KNL CPUs, 2) Pascal P100 GPU is still the overall best performer across all architectures and provides up to 18% improvement over KNL for AlexNet training, and 3) KNL provides a slight (3%) performance improvement over P100 for ResNet-50 training. We also observed that using MCDRAM as Cache or in flat mode provides up to 32% better performance than using the standard DDR for training on KNL CPUs. Thus, it is clear that the performance of single-node training for CPUs and GPUs is very comparable when appropriate optimized versions of helper libraries and code optimizations are properly utilized. For multi-node training, GPU-based training seems to have an edge as OSU-Caffe beats Intel-Caffe for AlexNet training across the full range of process/GPU count. However, this provides an opportunity for researchers to investigate better designs that can lead to enhanced scalability of CPU-based multi-node training.

ACKNOWLEDGMENTS

This research is supported in part by National Science Foundation under grants CCF-1565414, ACI-1664137, ACI-1450440, and CNS-1513120.

REFERENCES

[1] 2017. Caffe Website. http://caffe.berkeleyvision.org/. (2017). [Online; accessed Sep-2017].

[2] 2017. Caffe2 Framework. https://caffe2.ai. (2017).

[3] 2017. cuBLAS Library. https://developer.nvidia.com/cublas. (2017).

[4] 2017. cuDNN Library. https://developer.nvidia.com/cudnn. (2017).

[5] 2017. Intel Caffe. https://github.com/intelcaffe. (2017). [Online; accessed Sep-2017].

[6] 2017. Nvidia Caffe. https://github.com/NVIDIA/caffe. (2017). [Online; accessed Sep-2017].

[7] 2017. Xeon Phi Memory Latency. https://sites.utexas.edu/jdm4372/2016/12/06/memory-latency-on-the-intel-xeon-phi-x200-knights-landing-processor/. (2017).

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. *arXiv preprint arXiv:1603.04467* (2016).

[9] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda. 2017. 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 (To be presented) (PPoPP '17)*. ACM, New York, NY, USA.

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

[11] Mark Harris. 2017. Inside Pascal: NVIDIA's Newest Computing Platform. https://devblogs.nvidia.com/parallelforall/inside-pascal/. (2017). [Online; accessed Aug-2017].

[12] K. He, X. Zhang, S. Ren, and J. Sun. 2015. Deep Residual Learning for Image Recognition. *ArXiv e-prints* (Dec. 2015). arXiv:cs.CV/1512.03385

[13] Intel. 2017. Machine Learning Scaling Library. https://github.com/01org/MLSL. (2017).

[14] Intel 2017. MKL-DNN for Scalable Deep Learning. https://software.intel.com/en-us/articles/introducing-dnn-primitives-in-intelr-mkl. (2017).

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

[16] A. Krizhevsky. 2014. One Weird Trick for Parallelizing Convolutional Neural Networks. *CoRR* abs/1404.5997 (2014).

[17] A. Krizhevsky and G. Hinton. 2009. Learning Multiple Layers of Features from Tiny Images. http://www.cs.toronto.edu/˜kriz/learning-features-2009-TR.pdf. (2009).

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

[19] Microsoft. 2017. CNTK. http://www.cntk.ai/. (2017). [Online; accessed April-2017].

[20] MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE. 2017. https://mvapich.cse.ohio-state.edu/. (2017).

[21] CUDA Nvidia. [n. d.]. Programming Guide. ([n. d.]). [Online; accessed April-2017].

[22] D. K. Panda, A. A. Awan, and H. Subramoni. 2017. High Performance Distributed Deep Learning for Dummies. *Tutorial presented at Hot Interconnects 17* (2017). http://www.hoti.org/hoti25/tutorials/

[23] S. Shi. 2017. Benchmarking State-of-the-Art Deep Learning Software Tools. http://dlbench.comp.hkbu.edu.hk/. (2017). [Online; accessed Aug-2017].

[24] A. Sodani. 2015. Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor. In *2015 IEEE Hot Chips 27 Symposium (HCS)*. 1–24.

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

[26] HiDL Team. 2017. High Performance Deep Learning Project. http://hidl.cse.ohio-state.edu. (2017).