2017 IEEE International Parallel and Distributed Processing Symposium Workshops

Scaling Deep Learning Workloads: NVIDIA DGX-1/Pascal and Intel Knights Landing

Nitin A. Gawande, Joshua B. Landwehr, Jeff A. Daily, Nathan R. Tallent, Abhinav Vishnu, Darren J. Kerbyson
Pacific Northwest National Laboratory
{firstname}.{lastname}@pnnl.gov

***Abstract*—Deep Learning (DL) algorithms have become ubiquitous in data analytics. As a result, major computing vendors — including NVIDIA, Intel, AMD and IBM — have architectural road-maps influenced by DL workloads. Furthermore, several vendors have recently advertised new computing products as accelerating DL workloads. Unfortunately, it is difficult for data scientists to quantify the potential of these different products.**

This paper provides a performance and power analysis of important DL workloads on two major parallel architectures: NVIDIA DGX-1 (eight Pascal P100 GPUs interconnected with NVLink) and Intel Knights Landing (KNL) CPUs interconnected with Intel Omni-Path. Our evaluation consists of a cross section of convolutional neural net workloads: CifarNet, CaffeNet, AlexNet and GoogleNet topologies using the Cifar10 and ImageNet datasets. The workloads are vendor optimized for each architecture. GPUs provide the highest overall raw performance. Our analysis indicates that although GPUs provide the highest overall performance, the gap can close for some convolutional networks; and KNL can be competitive when considering performance/watt. Furthermore, NVLink is critical to GPU scaling.

***Index Terms*—NVIDIA DGX-1, Intel Knights Landing, Caffe, MaTEx, Deep Learning, Convolutional Neural Networks**

I. Introduction

Deep Learning (DL) algorithms have become ubiquitous in data analytics. A DL algorithm — typically implemented using a deep neural network — uses a cascade of layers to represent a non-linear model for a collection of observations. These models are effective for a variety of Machine Learning (ML) tasks, such as Computer Vision [1], [2], High Energy Physics [3], and Climate Modeling [4]. Several implementations of DL algorithms such as Caffe [5], TensorFlow [6], Theano [7], [8], and Torch [9] have become widely available.

The ubiquity of DL algorithms has generated significant interest among hardware vendors. Major vendors such as Intel, NVIDIA, IBM, and AMD have proposed architecture road maps influenced by DL algorithms. Recent offerings of two vendors – Intel and NVIDIA – are specifically geared towards optimizing DL algorithms, including training (model learning) and inferencing (model application). NVIDIA recently proposed the DGX-1 architecture [10], consisting of eight Pascal P100 GPUs tightly interconnected with NVLink. Intel recently released the Knights Landing (KNL) [11] CPU architecture with support for the Intel Omni-Path interconnect. Both architectures have the potential of scaling a variety of DL workloads across multiple compute devices, either to reduce response time or to solve larger problems.

Unfortunately, it is difficult for data scientists to quantify the potential of these different architectures for their workloads of interest. There is a lack of independent studies that provide thorough comparisons of these architectures, based on vendor-optimized DL implementations, with respect to both performance and power. Furthermore, system designers and computer architects would benefit from knowing what architectural features DL workloads can best exploit.

The goal of this paper is to serve as a reference point for data analysts and system designers who are considering these Intel and NVIDIA architectures. We make two contributions. *First*, we present an in-depth performance and power evaluation of important CNN workloads on (a) the NVIDIA DGX-1 and (b) a cluster with Intel KNL CPUs interconnected with Omni-Path. We select vendor-optimized Caffe implementations of the most applicable combinations of convolutional neural network (CNN) topologies and datasets. As CNN topologies, we select CifarNet, CaffeNet, AlexNet and GoogleNet; as datasets, we use Cifar10 and ImageNet. We identify scaling bottlenecks and analyze efficiency with respect to performance and power.

Second, we introduce an implementation of Intel-Caffe for distributed memory systems using Machine Learning Toolkit on Extreme Scale (MaTEx) extensions to Intel-Caffe. The implementation provides theoretical equivalence to the sequential algorithm (batch gradient descent).

Our performance evaluation shows that GPUs provide the highest overall raw performance. However, our results identify important caveats. To scale CNN computations, focusing DL architectural innovation on FLOPs can be misguided: the DGX-1's NVLink-based interconnect is critical to the DGX-1's GPU scaling advantage over the KNL cluster. When comparing one GPU and KNL, the latter is competitive when considering power consumption and networks with fewer features per layer. Finally, when cost is considered — a metric we do not explore because of its volatility — the gap between these architectures is may be closer than currently perceived.

II. Computing Systems

A. NVIDIA DGX-1

DGX-1 has eight Tesla P100-SXM2 GPUs conforming to Pascal architecture. The Tesla P100-SXM2 has a mezzanine connector which is known as SXM2 interface. In comparison to previous PCIe form factor, the SXM2 connector is an upgrade to enable the use of NVLink bus. Each GPU has 56 multiprocesors with 64 CUDA cores per multiprocesor. This makes each GPU

978-1-5386-3408-0 2017

DOI 10.1109/IPDPSW.2017.36

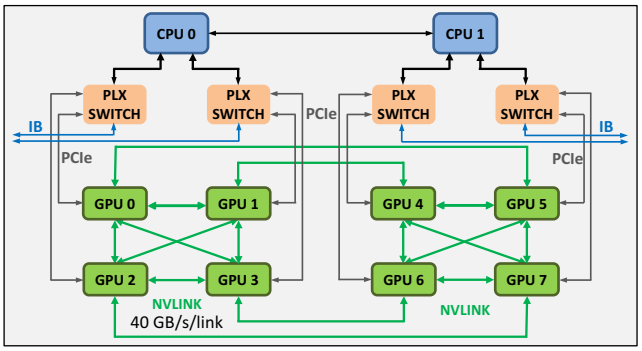

Fig. 1: Diagram of DGX-1 topology.

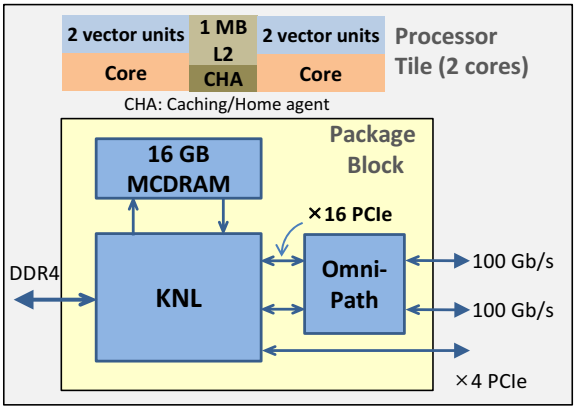

Fig. 2: Diagram of KNL CPU [13].

equipped with a total of 3584 CUDA cores. The GPU and memory clock rates are 1328 MHz and 715 MHz, respectively. The GPU has 4096-bit memory bus width, 16 GB global memory, and a 4MB L2 cache. A CUDA driver version 8.0 and CUDA capability version 6.0 was used to enable use of GPUs. The CPU on DGX-1 is a dual socket 40 core Intel Xeon E5-2698 v4 with a clock rate of 2.20GHz. Each socket has 20 cores with two threads enabled per cores. The main memory of the system is 504 GB with 50 MB L3 cache, a 256 KB L2 cache shared between two cores, and 64 KB L1 cache per core.

DGX-1 is equipped with NVLink interconnect for intra-node GPU clustering. DGX-1 utilizes multi GPU collective communications library called NCCL for communication over NVLink interconnect. NCCL is a library of accelerated collectives that is hardware-topology-aware [12]. NCCL is implemented as monolithic CUDA kernels and provides peer-to-peer GPUDirect access and intra-kernel synchronization between GPUs. Collectives such as broadcast, all-gather, copy, reduce, reduce-and-copy are available in this library.

Figure 1 shows an eight GPU cluster in DGX-1. The network of NVLink interconnect is wired such that any GPU is no more than two GPU hops away from other GPUs. The GPU cluster is connected to a switch (PLX-switch) with a PCIe x16 interconnect. This switch serves as a connecting point between GPUs and CPUs, and the Infiniband network. The maximum bandwidth for NVLink interconnect with Tesla P100 is reported at 160 GB/s, whereas the maximum bandwidth of PCIe x16 interconnected is 32 GB/s.

B. Intel Knights Landing and TACC Stampede Cluster

Intel's Knights Landing (KNL), the second generation of Xeon Phi, is a new many core architecture that resembles a GPU in certain ways [13]. KNL microarchitecture is fabricated with a 14 nm process and reported to provide 3 Teraflops of double precision and 6 Teraflops of single precision peak FP performance at a TDP of 215 Watts [13]. In this study we used the Intel Xeon Phi 7250 with clock speed of 1.40 GHz and 1.50 GHz Turbo. This chip has 68 hardware processor cores with four hyper-threads per core. KNL has configurable 16 GB high bandwidth memory (HBM) called multi-channel DRAM (MCDRAM). In addition, KNL supports the new vector extensions instruction set, AVX-512 which enables use of 512-bit-wide vector instructions and vector registers. The KNL block diagram is shown in Figure 2.

Unlike NVIDIA Pascal, computing systems with non-PCIe intra-node KNL clusters are not readily available. In contrast, KNL clusters with Intel Omni-Path interconnects are just becoming available at several supercomputing sites. We used the Texas Advanced Computing Center Stampede KNL Cluster. Each node has 96 GB DDR4 as main memory. The interconnect was a 100 Gb/s Omni-Path configured as fat tree of 8 spine switches and 320 leaf switches with 5/4 oversubscription.

An important feature of KNL is the ability to select a caching mode. Each pair of hardware cores with symmetrically shared 1 MB L2 cache is organized as a processor tile (Figure 2). The KNL architecture [13] offers the flexibility to configure the clustering of these tiles as per the requirement of user to achieve highest bandwidth for communication with caches [13] [14]. There are three possible ways to cluster processor tiles:

- All-to-All: Memory addresses are uniformly distributed across all distributed tag directories (DTD) on the chip.
- Quadrant/Hemisphere: Tiles are divided into four identical quadrants or two identical hemispheres. Here, the memory controller which is local to each quadrant or hemisphere is mapped only to tag directories in the same quadrant or hemisphere. In this mode the latency of communication with L2 is reduced in comparison to all-to-all mode.
- Sub-NUMA Cluster (SNC) modes: In SNC-4 or SNC-2 again the chip is divided into either four quadrants or two hemispheres. In this mode, with the use of NUMA aware software it is possible to pin software threads to a particular quadrant which contains tag directory. In addition, threads in each quadrant or hemisphere can access local memory in NUMA domain.

In addition to configuring processor tiles, KNL architecture provides the flexibility to configure the multi channel DRAM (MCDRAM) memory in three modes:

- Cache: MCDRAM serves as the last level cache.
- Flat: MCDRAM serves as regular memory.

- Hybrid: MCDRAM used partly as cache and partly as regular memory.

III. The Caffe Framework for Convolutional Neural Networks

The Convolutional Architecture for Fast Feature Embedding (Caffe) is a widely used framework for Convolutional Neural Networks (CNN) models. We selected four well known models: CifarNet [15], CaffeNet [16], AlexNet [17], and GoogLeNet [18] with model parameters typically adopted in Caffe framework were used in this evaluation work. CifarNet is a simple model typically used for small images whereas GoogLeNet being a more complex model which can be used for large image sizes.

We selected these three CNN architectures to cover ML model differences arising out of (1) model complexity; (2) number of parameters used; (3) computational complexity. In order to use specific implementations of one of these models, it is important to follow generally acceptable and already verified implementations. We used the Convolutional Architecture for Fast Feature Embedding (Caffe) [19] maintained by the Berkeley Vision and Learning Center (BVLC) [20]. Caffe is a collection of state-of-the-art deep learning algorithms and reference models in a clean and modifiable framework accessible through a open source repository [16].

The three models and their input datasets are described briefly below.

A. CifarNet

CifarNet [15] made use of the Cifar10 dataset. The Cifar10 dataset which has images of size 32 × 32 × 3 channels has 10 object classes. During its introduction, CifarNet was the state-of-the-art object classification model. CifarNet has three convolution layers, three pooling layers, and one fully-connected layer. This CNN architecture has 10 K parameters.

B. AlexNet and CaffeNet

AlexNet [17] made use of the ImageNet (ILSVRC2012) [21] dataset. A total of about 1.43 million images from this dataset with size 256 × 256 were used in AlexNet [17]. Alexnet performed significantly better over other non-deep learning methods for ILSVRC2012. AlexNet has five convolution layers, three pooling layers, and two fully-connected layers. This CNN architecture has about 60 M parameters. We also used CaffeNet model, which is essentially a replication of the AlexNet model with minor differences.

C. GoogLeNet

GoogLeNet [18] is more complex model than any of its predecessor CNN models. GoogLeNet has two convolution layers, two pooling layers, and nine inception layers. Each inception layer consists of six convolution layers and one pooling layer. The concept of inception layer is to cover bigger area of images while maintaining fine resolution for small information on these images. The inception module of GoogLeNet concatenates filters of different sizes into a single new filter. This avoids parameter explosion with the use of inception layers. GoogLeNet performs significantly better than AlexNet for the ImageNet and the recent ILSVRC [21] challenge datasets. This CNN architecture has about 5.5 M parameters. GoogLeNet in relation to AlexNet has (i) more layers; (ii) fewer features per layer, and; (iii) more activations. A higher computational intensity of GoogLeNet relative to AlexNet is attributed to above three reasons.

D. Input Datasets

We used two different datasets as input to CNN models. Figure 3 gives information on two datasets used here and a brief description of these datasets is given below.

1) The Cifar10 [15] is a labeled image dataset which is a subset of the *80 million tiny images* [22]. Cifar10 contains containing 60,000 32×32 color images with 3 channels in 10 classes. All these images are split into 50,000 training images and 10,000 test images.
2) The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [21] dataset is a benchmark of object classification and detection. The dataset consists of large number of object categories for large images. In this study we used the ILSVRC2012 which has 1000 object classes and 1,431,167 annotated images.

Dataset	Image Size	Description
Cifar10 [15]	32 × 32	60,000 images, 10 classes with 1000 images/class
ImageNet [21] (ILSVRC2012)	256 × 256	1.43 M images, 1000 classes

Fig. 3: Input datasets and parameters.

IV. Optimized Caffe Variants

We made use of optimized versions of Caffe models provided by the vendors of the computing hardware described in Section II. During the implementation of these ML models, we ensured that all executions satisfied following criteria:

- Produce semantically meaningful results. In particular, parallel versions should provide sequentially equivalent execution.
- Retain accuracy and convergence properties as per the implementations in the Caffe framework; e.g., it is possible to achieve better overall execution rates at the expense of both accuracy and convergence, or training time. Overall training time cannot be ignored.

We also considered different batch sizes and their effect on performance. The ML models that we used are further described in following sub-sections.

A. NVIDIA Caffe

This is an implementation of Caffe, a fork from BVLC-Caffe [16] optimized for the DGX-1 architecture [23]. This implementation of Caffe uses multiple GPU devices connected through high bandwidth NVLink interconnect and makes use

CNN	Dataset	Caffe Model	Batch Sizes
CifarNet	Cifar10	CIFAR10_full	96, 196
CaffeNet	ImageNet	bvlc_CaffeNet	256
AlexNet	ImageNet	bvlc_AlexNet	256, 512
GoogleNet	ImageNet	bvlc_GoogleNet	256, 512

Fig. 4: CNN architecture models and input datasets.

of optimized primitives for collective multi-GPU communication [12].

B. Intel Caffe

We used Intel distribution for Caffe [24] which is available as a open software through a Github repository [25]. This version of BVLC-Caffe is a fork dedicated to improving Caffe performance when running on CPU, in particular Intel Xeon processors. In order to make this version of Caffe run efficiently on the second generation of Xeon Phi processors, we made use of the *Intel-2017* software suite including the latest Intel Math Kernel Library (MKL). The multi-node distributed implementation of this vendor provided Caffe using a client-server-based MPI was not equivalent to sequential or single node run [26]. We made appropriate modifications to the vendor provided code [25] which ensured identical implementation with sequential runs and also improved its performance when using multiple nodes. These modifications are explained in the following sub-section.

C. MaTEx-Intel-Caffe

Machine Learning Toolkit for Extreme Scale (MaTEx) provides high performance distributed memory implementations of Machine Learning and Data Mining algorithms. In this paper, we considered MPI extensions to Intel Caffe available with MaTEx. Specifically, MaTEx-Intel-Caffe provides implementation of batch gradient descent, such that the batch (set of samples such as images, tabular data) is divided among processes (usually one process per compute node) to maintain equivalence to the default sequential algorithm. By splitting the data and replicating the model, MaTEx-Intel-Caffe achieves *data parallelism*. To maintain equivalence to the default sequential algorithm, an all-to-all reduction (achieved by `MPI_Allreduce`) is executed after each batch. Besides MPI based extensions, MaTEx-Intel-Caffe supports parallel reading of datasets using Parallel NetCDF format – which is used in this paper as well.

V. Evaluation

We used the Caffe framework described in Section IV to implement three different CNN architectures as shown in Figure 4. We considered four architecture models falling within broad three categories of CNN architectures. We made use of identical input files provided with Caffe for these three models, files with extension *.prototxt* for implementation on the two different computing architectures described in Section II.

A. Methodology

We used the latest implementations of the vendor provided CNN models described in Section IV. We also assured that the testing error for these models after running a large number of iterations was nearly equal for the two computing architectures. The performance was measured using two parameters: performance-time-efficiency and performance-correctness-efficiency. The number of training iterations per second during model run was used as a measure of time efficiency. The correctness/error rate obtained after a prescribed number of iterations in Caffe was used as a measure of correctness efficiency.

Performance of various implementations was profiled using available software tools. We used Nvprof [27] to profile implementations on DGX-1, whereas Intel Advisor 2017 [28] was used to profile implementations on KNL.

The power usage for the computing architectures was measured using combination of vendor supplied hardware and software tools. CUDA *nvidia-smi* API was used to collect power and memory usage. The KNL chip supports the "Running Average Power Limit (RAPL)" interface [29]. The results of this model are updated on the order of milliseconds and are accessible to user via a model specific register (MSR). Power for KNL was estimated using RAPL measurements.

B. Algorithm Implementation and Error Rate

Identical implementations of any ML model using identical dataset and model parameters are expected to generate identical error rates. Error rate generated after a given number of iterations of training and testing procedures of algorithm implementations is an important indicator of correctness-efficiency of implementation. The ML architecture model variants described in Section IV were implemented on the architectures described in Section II. We measured the error rate during training and testing for different implementations for a specified number of iterations during training and testing. We compared error rates from implementations on DGX-1 and KNL for CifarNet, CaffeNet and AlexNet, and GoogLeNet after 60,000, 2000, 2000, and 1600 iterations, respectively. These error rates were identical for implementations on two systems and matched the error rates produced by sequential BVLC-Caffe [20] implementation.

C. Single-node Configurations of KNL

Various clustering modes for processor tiles and memory modes were described in Section II-B. Single node runs on KNL for different ML models were tested with three configurations, each with a different combination of processor tile and MCDRAM configuration. A comparative result of performance for these three modes is shown in Figure 5. We see that the quadrant clustering with MCDRAM as cache mode gives the highest performance. As expected the all-to-all clustering mode with flat mode of MCDRAM gives the lowest performance. KNL processor using AVX-512 instruction set may benefit from use of MCDRAM in a flat mode. However, code optimizations and memory allocations specific to this

CNN / Dataset	Time per iteration, ms		
	Mode a	Mode b	Mode c
CifarNet / Cifar10			
Batch size 96	18	18	26
Batch size 192	27	27	29
CaffeNet / ImageNet			
Batch size 256	354	415	422
AlexNet / ImageNet			
Batch size 256	371	434	628
Batch size 512	732	810	821
GoogLeNet / ImageNet			
Batch size 256	1740	1846	2523
Batch size 512	3427	*	*

Mode a: Quadrant clustering with MCDRAM as cache
Mode b: Quadrant clustering with MCDRAM as flat
Mode c: All-to-all clustering with MCDRAM as flat
*Manually terminated execution due to virtual memory thrashing

Fig. 5: Single node performance of selected Intel KNL modes.

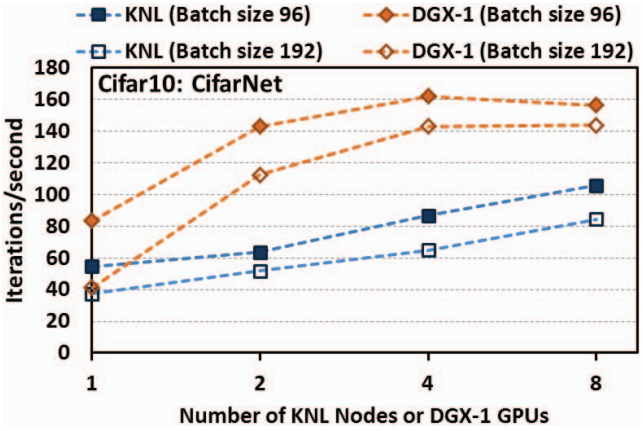

Fig. 6: Performance scaling of CifarNet/ImageNet.

CUDA Operation	1 gpu	2 gpu	4 gpu	8 gpu
cudaStreamSynchronize	23.2%	80.9%	71.2%	59.8%
cudaFree	52.4%	0.9%	1.3%	1.7%
cudaMalloc	8.5%	0.2%	0.5%	1.1%
cudaEventDestroy	8.0%	0.1%	0.2%	0.5%
cudaLaunch	3.4%	8.6%	10.9%	14.4%
cudaMemcpy	1.0%	1.5%	2.0%	2.7%
Other CUDA ops	3.5%	7.8%	13.9%	19.8%

Fig. 7: Time distribution of CUDA operations for CifarNet (batch size 192) on DGX-1.

MCDRAM mode need to be adopted to obtain optimum performance. We used the quadrant clustering with MCDRAM as cache as the mode to conduct scaling experiments with KNL.

D. Scaling Comparison

Next we compared performance of CNN model runs with scaling over multiple GPUs and CPU-nodes for DGX-1 and KNL, respectively. We scaled on 1, 2, 4, and 8 DGX-1 GPUs or KNL compute nodes, respectively. In the case of DGX-1, we considered scaling with runs on 1, 2, 4, and 8 GPUs. Whereas in the case of KNL, we considered scaling with runs on 1, 2, 4, and 8 nodes.

1) CifarNet/Cifar10: Figure 6 shows performance/scaling of CifarNet model for batch sizes 96 and 192. The performance of KNL improves with increasing number of nodes from one through eight. However, this performance gain is sub-linear due to synchronization and communication overhead which is involved with increasing number of nodes. The performance of DGX-1 improves significantly with the use of more than one GPU for both batch sizes. Interestingly, the performance gain with use of two GPUs is super-linear over the use of only one GPU for batch size of 192. However, the performance gain diminishes with the use of four and eight GPUs.

Figure 7 shows time distribution of different CUDA operations for batch size 192. Here we see that the time for *cudaFree* API is significantly higher for the run on single GPU. The time cost of *cudaFree* significantly increases with the size of memory buffer. Whereas in the case of runs on two GPUs or more the DGX-1 architecture is able to efficiently make use of memory across multiple GPUs and hence the time cost of *cudaStreamSynchronize* API was higher. The reason for more than ideal speed up with the use of two GPUs for the batch size of 192 was attributed to making best use of the memory required as buffer during computation. Efficient use of memory across multiple GPUs is further helped by high bandwidth NVLink memory inter-connect. The computation workload of CifarNet model using images from Cifar10 dataset is not large enough to utilize all GPUs of DGX-1 and therefore the performance does not scale well with 4-GPUs and degrades further with the use of 8-GPUs. Average GPU utilization with batch size 192 was 87% for 2-GPUs drops to 84% and 63% for 4-GPUs and 8-GPUs, respectively.

2) CaffeNet/ImageNet: Figure 8 shows the performance scaling of CaffeNet model. In Figure 8, we see that single node and single GPU performance of KNL and DGX-1, respectively are nearly identical. The DGX-1 performance scales well with increasing number of GPUs. The synchronization cost makes this scaling suboptimal. The average GPU utilization was 96.0%, 93.4%, 89.5%, and 83.4% with the use of 1, 2, 4, 8 GPUs, respectively.

The KNL performance does not scale well. The only marginal performance gain with the use of two KNL nodes and further degradation with four and eight nodes is attributed to communication related bottlenecks as discussed later in this section.

Figure 9 shows the distribution of costs when scaling CaffeNet on DGX-1. Costs are represented as CUDA operations. We see that progressively with increasing the number of GPUs from one through eight, the sum of time for *cudaStreamSynchronize* and *cudaEventSynchronize* decreases. The sum of times for these two APIs is an indirect indicator of computation time of kernels launched on GPUs. We see that this time progressively reduces from 94% to 71% for 1-GPU and 8-GPUs, respectively. This reduction occurs because other costs associated with

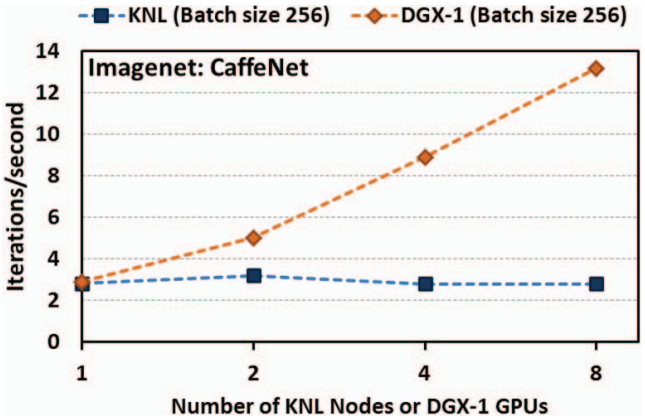

Fig. 8: Performance scaling of CaffeNet/ImageNet.

thread management across multiple GPU devices is increasing. This helps to explain the deviation of scaling below ideal for CaffeNet on DGX-1.

CUDA Operation	1 gpu	2 gpu	4 gpu	8 gpu
cudaStreamSynchronize	68.0%	63.4%	57.4%	48.9%
cudaEventSynchronize	25.5%	26.2%	25.4%	22.2%
cudaMemcpy	2.8%	3.8%	5.3%	7.1%
cudaLaunch	1.1%	2.1%	3.5%	6.0%
Other CUDA ops	2.6%	4.5%	8.4%	15.8%

Fig. 9: Time distribution of CUDA operations for CaffeNet (batch size 256) on DGX-1.

3) AlexNet/ImageNet: Figure 10 shows the performance scaling for AlexNet model. The computations of AlexNet model are comparable to the computations of CaffeNet. We therefore see that the single node and single GPU performance of KNL and DGX-1, respectively are nearly identical for both batch sizes. Again the scaling of DGX-1 performance for AlexNet is identical to that of CaffeNet given the similarity of the two ML models. The KNL implementation again shows no scaling with the increasing number computer nodes. AlexNet architecture has 60 M parameters. Considering that each parameter is 4-bytes (a float), sending parameters requires a message size of 240 MB. Thus, performing a parameter all-reduction has notable network and synchronization cost. This cost impedes scaling of AlexNet on KNL cluster.

The degradation in scaling of performance of KNL due to communication overhead while using a complex model like CaffeNet and AlexNet leads intuitively to try CNN model with even more computational load with relatively less number of model parameters. The GoogLeNet model is an ideal case of such CNN model.

4) GoogLeNet/ImageNet: Figure 11 shows the performance scaling of GoogLeNet. In Figure 11, data from use of one and two GPUs were not available. This was due to the specific implementation of GoogLeNet model which required large GPU memory which was available only with the use of four and eight GPUs. DGX-1 performance was much higher than

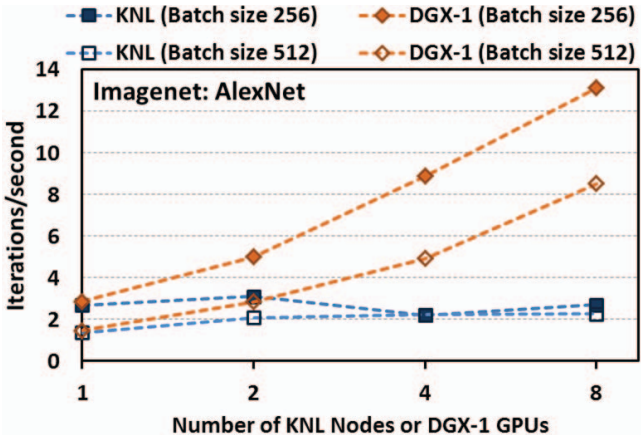

Fig. 10: Performance scaling of AlexNet/ImageNet.

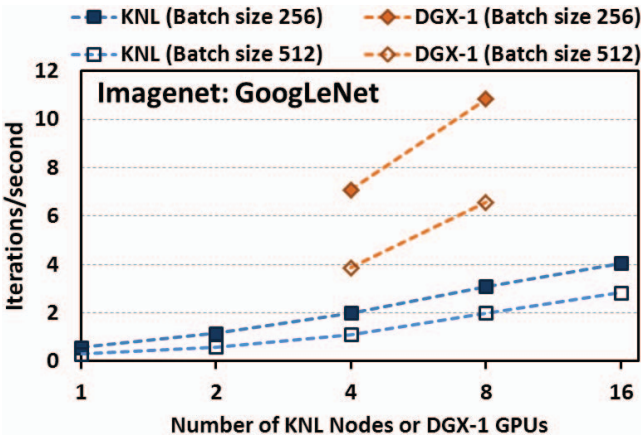

Fig. 11: Performance scaling of GoogLeNet/ImageNet.

that of KNL even with the use of 16 nodes. The performance on KNL showed scaling with increasing the number of nodes from 1 through 16. However, this scaling was sub-linear.

Figure 12 shows the time distribution of different CUDA operations with the use of 4 and 8 GPUs. With the use of maximum 8 GPUs the sum of costs of *cudaStreamSynchronize* and *cudaEventSynchronize* decreases. The relative time cost of *cudaMalloc* and *cudaFree* however become significant. Associated costs of memory allocation and freeing up the memory proportionately become relatively high. This in effect causes some deterioration in scaling of performance from 4-GPUs to using 8-GPUs.

Figures 6 through 11 show the performance and scaling with increasing number of nodes or GPUs. In the case of runs with KNL, we see poor scaling for CaffeNet and AlexNet in Figures 8 and 10, respectively. We investigated the runs on KNL further by measuring the relative cost of computation and communication with the use of more than one nodes. Figure 13 shows the communication overhead as percentage of total time spent per iteration. We see that CaffeNet and

CUDA Operation	4 gpu	8 gpu
cudaStreamSynchronize	66.7%	28.0%
cudaEventSynchronize	7.3%	10.4%
cudaLaunch	6.4%	7.6%
cudaMalloc	5.7%	19.0%
cudaFree	4.6%	14.4%
cudaMemGetInfo	1.6%	4.3%
Other CUDA ops	7.7%	16.3%

Fig. 12: Time distribution of CUDA operations for GoogLeNet (batch size 256) on DGX-1.

CNN / Dataset	Communication time per iteration		
	2 node	4 node	8 node
CifarNet / Cifar10			
Batch size 96	1.9%	5.3%	3.1%
Batch size 192	2.6%	6.9%	8.6%
CaffeNet / ImageNet			
Batch size 256	32.5%	60.1%	77.7%
AlexNet / ImageNet			
Batch size 256	46.2%	61.4%	76.7%
Batch size 512	19.1%	47.3%	66.8%
GoogLeNet / ImageNet			
Batch size 256	5.6%	14.3%	18.0%
Batch size 512	3.9%	14.2%	38.7%

Fig. 13: Communication overhead as percentage of total time per iteration for KNL.

AlexNet runs on 2 through 8 nodes show progressive increase in communication cost. CaffeNet and AlexNet ML make use of a large number of model parameters, 60 M for AlexNet. A large number of parameters require more time to perform *all-reduce* at every iteration. The communication overhead for AlexNet using 8-nodes reduces from 76.71% to 66.81% with the increase in batch size from 256 to 512, respectively. This is because the computation time increases for batch size 512 while the communication time increases only marginally. As long as the communication overhead is less than the computational cost as measured at each iteration, we see improvement in performance with the use of increasing number of nodes. The CifarNet model offers such potential to scale with increasing number of compute nodes. For GoogleNet with batch size 256, the communication cost is relatively low at 18.0% while using 8-nodes.

E. Floating Point Rates

The measure of performance reported in Figures 6 through 11 is the number of iterations per second. With the use of identical workload in terms of batch size, relevant ML-model parameters, and algorithm implementation we were able ensure an identical workload over the two computing systems evaluated here. The number of floating point operations (FLOPS) performed in single precision is another metric to report performance. The measurement of FLOPS may differ significantly for CPU and GPU architecture, and the software tools used for this measurement.

We made use of Intel Advisor 2017 [28] and Intel VTune to get the FLOPS count for different ML architecture models. The FLOPS count was obtained using these software tools on KNL for a specified number of iterations. The number of FLOPS per iteration was obtained by running the simulations twice for up to two different number of iterations. The difference of FLOPS thus obtained were divided by the time per iterations to get the number of FLOPS per second (FLOPS/s).

In Figure 14 the performance of KNL with 1-Node is reported in GFLOPS/s in the second column. Subsequent columns report the performance scaling factor when compared with 1-Node KNL performance. The scaling factors reported in Figure 14 when multiplied with the GLOPS/s cannot provide an accurate measure of FLOPS rate for 8-Node KNL, and 1-GPU or 8-GPU DGX-1. These scaling factors however provide an approximate comparison of FLOPS rate.

F. Power Efficiency

Power efficiency is a critical evaluation metric. In general, the power efficiency is at odds with the performance efficiency. Therefore there exists a trade-off between the power and performance efficiencies of a computing hardware for an application under consideration. Considering the results presented in Section V-D that represent a case of strong scaling problem, the power efficiency exhibit an inverse relationship with the performance. Figures 15a through 18b show plots of power with respect to performance for different ML architecture models and using different batch sizes. The power numbers for KNL were directly read from the in-build hardware register during the course of running an application. We did not find any performance overhead cost associated with collection of power numbers. The power for KNL includes the CPU package power and the DRAM package power. Here, we ignored the network power for inter-node communication for KNL. The power for DGX-1 was measured using the *nvidia-smi* API. There is an overhead cost associated with using the *nvidia-smi* API. Therefore we kept the sampling interval coarse enough to keep the performance overhead low. A sampling rate of 900 ms or above ensured keeping the performance overhead below 4%. Separate runs were conducted to obtain the power numbers in addition to the runs for performance for both KNL and DGX-1. The CPU and DRAM memory power for DGX-1 were small and nearly constant and therefore not included in Figures 15a through 18b.

The CifarNet plots in Figures 15a and 15b shows that DGX-1 mostly outperforms KNL in power efficiency along with performance for two different batch sizes. Figures 16 through 17b show plots of power and performance for CaffeNet and AlexNet. We see that the performance of KNL using one node is comparable to DGX-1 with one GPU. Poor scaling of KNL with increasing number of nodes keep the DGX-1 power-performance data plotted in Figures 16 through 17b on the power-performance Pareto front. With the absence of data points for DGX-1 using one or two GPUs for GoogLeNet, we see in Figures 18a and 18b that single node KNL is most power efficient while DGX-1 with 8 GPUs gives highest performance.

CNN / Dataset	GFLOPS/s: KNL/1-node	Iteration/s ratio: vs. KNL/1-node		
		KNL/8-node	DGX-1/1-gpu	DGX-1/8-gpu
CifarNet / Cifar10				
Batch size 96	55	1.9	1.5	2.9
Batch size 192	74	2.3	1.1	3.8
CaffeNet / ImageNet				
Batch size 256	357	1.0	1.0	4.7
AlexNet / ImageNet				
Batch size 256	258	1.0	1.1	4.9
Batch size 512	295	1.7	1.1	6.2
GoogLeNet / ImageNet				
Batch size 256	90	5.4	*	18.9
Batch size 512	78	6.8	*	22.5

*DGX-1 run required use of more than 1-GPU

Fig. 14: FLOP performance for 1 KNL; and relative iteration/s ratio for selected configurations (vs. 1 KNL).

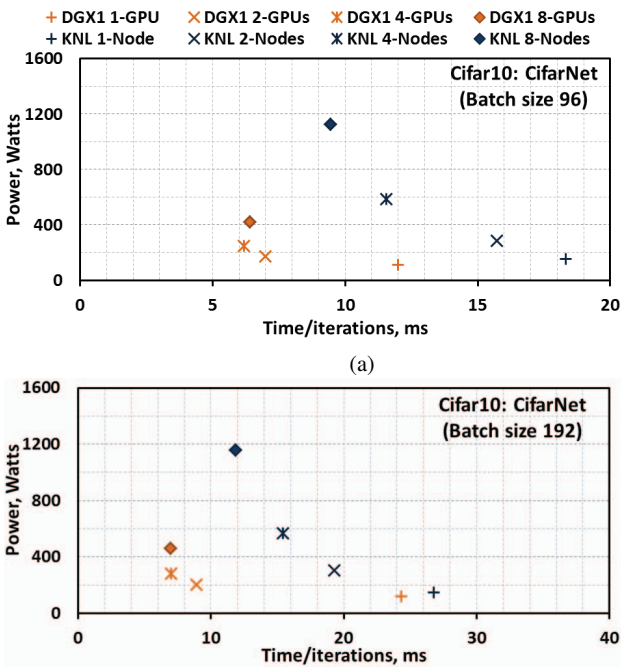

Fig. 15: Power and performance of CifarNet/Cifar10 with batch sizes (a) 96 and (b) 192.

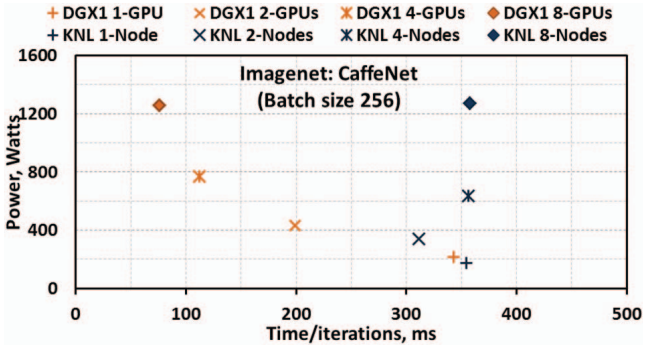

Fig. 16: Power and performance of CaffeNet/ImageNet with batch size 256

VI. Related Work

One of the widely used convolutional network benchmarks focuses primarily on GPU performance [30]. In one recent study by Shi et al., [31], performance comparison of a few deep learning software tools on multi-core CPU and many-core GPUs was presented. Multicore scalability of tensor convolution optimized for deep networks was presented in [32]. In one other comparative study of deep learning software framework [33], the performance of specific kernels used in different ML models on GPU was measured that used cuDNN library [34]. A study by Song et al., [35] presented characterization of CNN based data processing on GPUs and CNN models implementations on distributed framework. The only study comparing multi-core CPU and many-core GPUs [31] considered relatively small batch sizes for different DNN/CNN architecture models. Also this study did not considered scaling over multiple-CPU distributed framework or scaling with the use of more than one GPU. To date performance evaluation on multi-node CPU and comparison with implementation on many GPU architecture is not available in the literature. Power consumption is another important aspect in addition to the information of performance and scaling of CNN models. The work presented in this manuscript highlights performance with respect to time and correctness and the power consumption of state of the art CPU and GPU architectures. Such evaluation may provide extremely useful information on decision making to the users of ML-CNN architecture models.

VII. Conclusions

We have provided a detailed performance and power scaling analysis of important CNN workloads on two architectures: (a) NVIDIA DGX-1 (eight Pascal P100 GPUs interconnected with NVLink) and (b) a cluster with Intel Knights Landing (KNL) CPUs interconnected with Intel Omni-Path. We used the

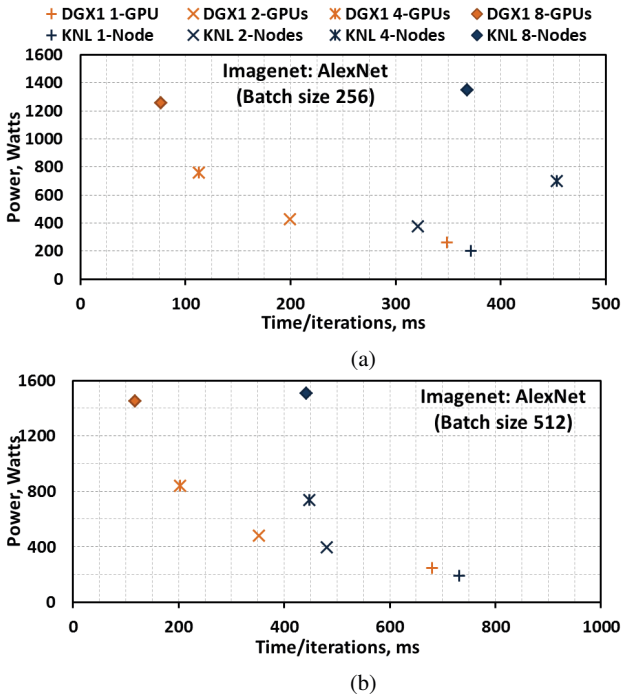

Fig. 17: Power and performance of AlexNet/ImageNet with batch sizes (a) 256 and (b) 512.

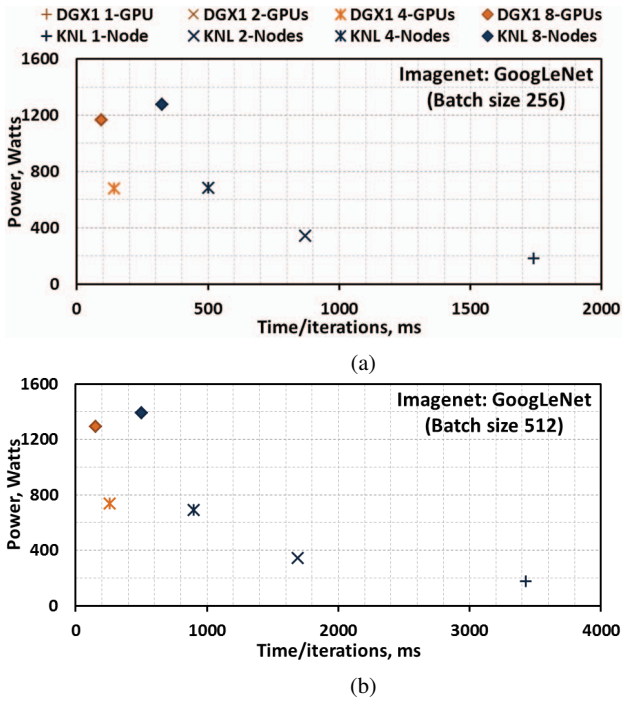

Fig. 18: Power and performance of GoogLeNet/ImageNet with batch sizes (a) 256 and (b) 512.

vendor recommended implementations of ML models. In case of multi-node implementation on KNL, we made appropriate modifications to the code which ensured results identical to sequential run. This modification also improved multi-node performance of ML models on KNL. We ensured identical algorithm implementations on KNL and DGX-1systems that produced results identical to a sequential run.

Our conclusions are as follows.

First, we find that GPUs provide the highest overall raw performance. Most DL architectural innovation focuses on FLOPs (e.g. half-precision), whereas *interconnect* performance is the scaling bottleneck. Specifically, the DGX-1's high speed NVLink-based interconnect is very important for scaling the performance of these workloads. Networks such as AlexNet require 10× more parameters than the dense GoogLeNet. This leads to per-rank all-reductions with very large payloads. The NVLink-based interconnect is fast enough that compute times dominate network times while scaling out to 8 GPUs. In contrast, the Omni-Path-based interconnect results in synchronization costs that rival compute costs.

Second, despite the above, we also find that a single KNL can be competitive with a single Pascal in certain cases. The performance of CifarNet and AlexNet on single node of KNL or single Pascal GPU are comparable. When considering performance and power tradeoffs, we also find that a single KNL can be competitive.

Finally, we observe that potential purchases will want to also consider cost when making any procurement decisions. Adding this additional consideration may make one platform more or less attractive.

Acknowledgments

The authors thank the staff at The Texas Advanced Computing Center (TACC) for providing assistance and computing time; Antonino Tumeo (PNNL) for valuable discussion; and Matthew Macduff (PNNL) for evaluation assistance. We are grateful for funding support from the U.S. Department of Energy's (DOE) Office of Advanced Scientific Computing Research as part of "Convergence of Deep Learning and Machine Learning for HPC Simulation and Modeling" and the "Center for Advanced Technology Evaluation" (CENATE). Pacific Northwest National Laboratory is operated by Battelle for the DOE under Contract DE-AC05-76RL01830.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR 2015*, 2015.

[3] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for Exotic Particles in High-Energy Physics with Deep Learning," *Nature Commun.*, vol. 5, p. 4308, 2014.

[4] Y. Liu, E. Racah, J. Correa, A. Khosrowshahi, D. Lavers, K. Kunkel, M. Wehner, W. Collins *et al.*, "Application of deep convolutional neural networks for detecting extreme weather in climate datasets," *arXiv preprint arXiv:1605.01156*, 2016.

[5] 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.

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[7] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Jun. 2010, oral Presentation.

[8] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[9] R. Collobert, S. Bengio, and J. Marithoz, "Torch: A modular machine learning software library," 2002.

[10] NVIDIA, "NVIDIA DGX-1," http://www.nvidia.com/dgx-1, Jan 2017.

[11] Intel, "How Intel Xeon Phi processors benefit machine learning/deep learning apps and frameworks," https://software.intel.com/en-us/blogs/2016/06/20/how-xeon-phi-processors-benefit-machine-and-deep-learning-apps-frameworks, 2017.

[12] "NCCL: Optimized primitives for collective multi-GPU communication," https://github.com/NVIDIA/caffe, 2016.

[13] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu, "Knights landing: Second-generation intel xeon phi product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, Mar 2016.

[14] A. Vladimirow and R. Asai, "Clustrering modes in Knights Landing processors: Developer's guide," 2016.

[15] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[16] Berkeley Vision and Learning Center, "Convolutional architecture for fast feature embedding (caffe)," https://github.com/BVLC/caffe/, 2016.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

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

[19] 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.

[20] Berkeley Vision and Learning Center, "Berkeley vision and learning center: Caffe," http://caffe.berkeleyvision.org, 2016.

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[22] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 11, pp. 1958–1970, Nov 2008.

[23] "Convolutional architecture for fast feature embedding (caffe)," https://github.com/NVIDIA/caffe, 2016.

[24] "Intel distribution for caffe," https://software.intel.com/en-us/machine-learning/deep-learning, 2016.

[25] "BVLC-caffe fork dedicated to improving caffe performance when running on CPU, commit: f2e66a797ccbd075bf1f85da269e674591e165d6," https://github.com/intel/caffe, 2016.

[26] 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*, 2016.

[27] NVIDIA, "CUDA toolkit documentation," 2016.

[28] Intel, "Intel advisor 2017 for linux," 2016.

[29] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: Memory power estimation and capping," in *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, Aug 2010, pp. 189–194.

[30] S. Chintala, "convnet-benchmark, 2017," github.com/soumith/convnet-benchmarks.

[31] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," *arXiv preprint arXiv:1608.07249*, 2016.

[32] D. Budden, A. Matveev, S. Santurkar, S. R. Chaudhuri, and N. Shavit, "Deep tensor convolution on multicores," *CoRR*, vol. abs/1611.06565, 2016.

[33] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative study of deep learning software frameworks," *arXiv preprint arXiv:1511.06435*, 2015.

[34] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[35] M. Song, Y. Hu, Y. Xu, C. Li, H. Chen, J. Yuan, and T. Li, "Bridging the semantic gaps of gpu acceleration for scale-out cnn-based big data processing: Think big, see small," in *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, ser. PACT '16. New York, NY, USA: ACM, 2016, pp. 315–326.