Demystifying the MLPerf Training Benchmark Suite

Snehil Verma[†] snehilv@utexas.edu
Qinzhe Wu[†] qw2699@utexas.edu
Bagus Hanindhito[†] bagus@utexas.edu
Gunjan Jha[‡] gdv482@my.utsa.edu
Eugene B. John[‡] eugene.john@utsa.edu
Ramesh Radhakrishnan[*] ramesh_radhakrishnan@dell.com
Lizy K. John[†] ljohn@ece.utexas.edu

[†]*The University of Texas at Austin,* [‡]*The University of Texas at San Antonio,* [*]*Dell Inc.*

***Abstract*—MLPerf, an emerging machine learning benchmark suite, strives to cover a broad range of machine learning applications. We present a study on the characteristics of MLPerf benchmarks and how they differ from previous deep learning benchmarks such as DAWNBench and DeepBench. MLPerf benchmarks are seen to exhibit moderately high memory transactions per second and moderately high compute rates, while DAWNBench creates a high-compute benchmark with low memory transaction rate, and DeepBench provides low compute rate benchmarks. We also observe that the various MLPerf benchmarks possess unique features that allow unveiling various bottlenecks in systems. We also observe variation in scaling efficiency across the MLPerf models. The variation exhibited by the different models highlight the importance of smart scheduling strategies for multi-GPU training. Another observation is that dedicated low latency interconnect between GPUs in multi-GPU systems is crucial for optimal distributed deep learning training. Furthermore, host CPU utilization increases with an increase in the number of GPUs used for training. Corroborating prior work, we also observe and quantify improvements possible by mixed-precision training using Tensor Cores.**

***Index Terms*—Benchmarking, Machine Learning, Training**

I. Introduction

The recent advances in machine learning have led to an evolution of a myriad of applications, revolutionizing scientific, industrial and commercial fields. Machine learning, primarily deep learning, is the state-of-the-art in providing models, methods, tools, and techniques for developing autonomous and intelligent systems.

Among the two parts of machine learning (training and inference), training is the long-running task. This is not only because of the massive datasets needed for high accuracy but also because the weights in the neural network need to be iteratively tuned until the model meets the desired quality. As the system's compute power plays a significant role in how fast the neural network learns, training is usually done using high-performance compute clusters such as multi-GPU systems.

Evaluation of training capability necessitates benchmarks that encompass the training requirements of modern DL models from different domains. MLPerf [22] is an emerging consortium that provides separate benchmark suites for machine learning training and inference. The training suite helps to measure the performance of machine learning frameworks, hardware accelerators, and cloud platforms [11], [19], [42]. The major contributors of the benchmarks include Google, NVIDIA, Baidu, Intel, and other commercial vendors, as well as universities such as Harvard, Stanford, and the University of California, Berkeley. MLPerf's initial release `v0.5` in 2018 consisted of benchmarks only for training, but inference benchmarks have been added in June 2019. MLPerf training benchmark suite covers the areas of computer vision, product recommendation, and other key areas where deep learning models have shown success and the datasets are available publicly. In this work we solely focus on the MLPerf `v0.5` training benchmark suite.

We evaluate the MLPerf benchmarks with experiments on diverse hardware platforms. Additionally, we investigate whether the execution characteristics of these benchmarks point out sufficient dissimilarities, or they are mostly similar in spite of diverse domains. The objective of this work is to unfold the answers to following enigmas:

- How different are the MLPerf benchmarks from the prior deep learning benchmarks?
- How different are the MLPerf benchmarks from each other?
- Does the MLPerf suite contain models that can achieve a performance differential by using reduced precision and NVIDIA's tensor cores?
- How well does the training performance scale with increasing the number of GPUs?
- What is an efficient way for a user to operate on multiple GPUs to train several models: should they run distributed jobs one-by-one on all GPUs or should they run jobs assigning one model to each GPU or is there any other better solution?
- How are CPU, GPU and interconnect utilizations? Is there a significant performance impact from the high-bandwidth GPU interconnects?

The key insights revealed by this work are summarized in Table I, and the rest of paper is organized as follows: Section II introduces the emerging MLPerf [22] benchmark suite as well as some prior deep learning benchmarks (e.g., DAWNBench [7] and DeepBench [3]). In Section III, we expand on the system configurations and topologies, on which various experiments were performed. We investigate various benchmark characteristics, performance impacts from mix precision scheduling, and present the similarity of various

TABLE I: Summary of key insights from the work.

Observation	Location	Insight/Explanation
MLPerf benchmark suite has a disjoint envelope from DAWNBench and DeepBench.	Figure 1a	MLPerf, DAWNBench, and DeepBench suite stress HBM2 memory at different levels, and are optimized to different extents. Throughput and arithmetic intensity: DAWNBench $>$ MLPerf $>$ DeepBench.
DeepBench, MLPerf, and DAWNBench are located in different regions in the roofline graph.	Figure 2	
Every benchmark in MLPerf benchmark suite is on the boundary of the workload space.	Figure 1b	There is a great diversity existing in MLPerf benchmark suite, e.g., in terms of the scaling efficiency. This information is helpful for resource scheduling in systems with multiple devices, such as data centers and cloud platforms.
Different benchmarks scale up differently, and by exploiting these differences, the optimal scheduling can save hours of training on multi-GPU systems.	Table IV Figure 4	
Data points representing machine learning workloads are close to the slanted roof line.	Figure 2	It's easy to exploit the abundant parallelism in ML applications and finally end up being bound by hardware resources.
Mixed precision in combination with TensorCores earns significant speedup on MLPerf.	Figure 3	Hardware support for reduced precision arithmetics is important, especially for machine learning workloads.
When scaling to more GPUs, many benchmarks have a super-linear increase in PCIe / NVLink utilization.	Table V	Machine learning applications can become communication-heavy workloads, so it is worth paying attention to the buses in ML processor designs. Direct connections between GPUs facilitate better performance in machine learning workloads.
Training time: GPU-system with NVLink enabled $<$ GPU-system with PCIe switch enabled $<$ system with GPUs connected using CPU PCIe ports.	Figure 5 Table III	

benchmarks in Section IV. Section V presents measurements on the system resources utilization to provide insights on CPU's, GPU's, and interconnect's impact on machine learning training performance as well as the memory requirement to store the dataset during processing. Then we conclude the paper with Section VI.

II. Background

In this section, we introduce MLPerf [22], DAWNBench [7], and DeepBench [3] benchmarks for machine learning.

A. MLPerf Benchmarks

The MLPerf [22] benchmark suite includes workloads from image classification, object detection, translation, recommendation and reinforcement learning.

Image classification identifies the object classes present in the image. This benchmark uses ResNet-50 [13], [14] model. ResNet-50 signifies a 50-layered residual network, which effectively overcomes the problem of degradation of training accuracy and is easier to optimize, and can gain accuracy from considerably increased depth.

Object detection is a technology that classifies individual objects and localizes each using a bounding box. Mask R-CNN [12] adds a branch for predicting segmentation masks on each Region of Interest, along with the existing branch for classification and bounding box regression. In Mask R-CNN, the additional mask output is distinct from the class and box outputs, as it extracts a finer spatial layout of an object. On the contrary, Single Shot Detection (SSD) [18] discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. SSD eliminates proposal generation and subsequent pixel or feature resampling stage and encapsulates all computation in a single network. This makes SSD easy to train and integrate into systems that require a detection component.

Translation is the task of converting an input text from one language to another. The model architecture - Transformer [34], avoids recurrence and relies on an attention mechanism to generate global dependencies between input and output. The attention weights apply to all symbols in the sequences. On the other hand, Google's Neural Machine Translation system (GNMT) [39] model uses residual connections as well as attention connections. GNMT provides a decent balance between the flexibility of "character"-delimited models and the efficiency of "word"-delimited models, and handles translation of rare words.

Recommendation is a task accomplished by a recommendation system, that predicts the "rating" or "preference" to an item. This benchmark uses Neural Collaborative Filtering model (NCF) [15] that can express and generalize matrix factorization under its framework. To supercharge NCF modeling with non-linearities, a multi-layer perceptron can be utilized in this model to learn the user-item interaction function.

Reinforcement Learning is associated with how software agents should take actions in an environment to maximize the notion of cumulative reward. This benchmark is based on a fork of the mini-go project [30], inspired by DeepMind's AlphaGo algorithm [28], [29]. [1]

Table II displays a summary of the various workloads of MLPerf `v0.5` release, including respective models as well as the datasets used. The metric used by MLPerf is the time taken to reach a specified accuracy or quality target, which is also listed in Table II for each benchmark. MLPerf benchmark implementations provided by the submitters currently include frameworks such as PyTorch [26], MXNet [6] and TensorFlow [1]. Many of the workloads consume days of training time on powerful GPUs, as indicated in Table IV for MLPerf's reference machine which has an NVIDIA Tesla P100 GPU.

B. DAWNBench

DAWNBench [7], developed by Stanford University in 2017, evaluates deep learning systems across different optimization strategies, model architectures, software frameworks, clouds, and hardware. It supports benchmarking of *Image Classification* on CIFAR10 [17] and ImageNet [8], and *Question*

[1]Since our evaluation focus is on MLPerf `v0.5` on GPU platforms, and the only GPU code of *Reinforcement Learning* is the reference one, which spends more time on the CPU than the GPU, *Reinforcement Learning* is excluded in the rest of the paper.

TABLE II: Summary of benchmarks in MLPerf (top), DAWNBench (middle), and DeepBench (bottom) used in this study.

Abbreviation	Domain	Model	Framework	Submitter	Dataset	Quality Target
MLPf_Res50_TF	Image Classification	ResNet-50	TensorFlow	Google	ImageNet	Accuracy: 0.749
MLPf_Res50_MX			MXNet	NVIDIA		
MLPf_SSD_Py	Object Detection	SSD (light-weight)	PyTorch	NVIDIA	Microsoft COCO	mAP: 0.212
MLPf_MRCNN_Py		Mask RCNN (heavy-weight)				Box mAP: 0.377, Mask mAP: 0.339
MLPf_XFMR_Py	Translation	Transformer	PyTorch	NVIDIA	WMT17	BLEU score (uncased): 25
MLPf_GNMT_Py		RNN GNMT				Sacre BLEU score (uncased): 21.80
MLPf_NCF_Py	Recommendation	Neural Collaborative Filtering	PyTorch	NVIDIA	MovieLens 20-million	Hit rate @ 10: 0.635
Abbreviation	**Domain**	**Model**	**Framework**	**Submitter**	**Dataset**	**Quality Target**
Dawn_Res18_Py	Image Classification	ResNet-18 (modified)	PyTorch	bkj	CIFAR10	Test accuracy: 94%
Dawn_DrQA_Py	Question Answering	DrQA	PyTorch	Yang et al.	SQuAD	F1 score: 0.75

Abbreviation	Operation	Parameters		Targeted Application
Deep_GEMM_Cu	Dense Matrix Multiply	all specified in the repository		N/A
Deep_Conv_Cu	Convolution	all specified in the repository		N/A
Deep_RNN_Cu	Vanilla Recurrent	Units=1760	N=16	DeepSpeech
	GRU Recurrent	Units=2816	N=32	
	GRU Recurrent	Units=1024	N=32	Speaker ID
	LSTM Recurrent	Input=512	N=16	Machine Translation
	LSTM Recurrent	Input=4096	N=16	Language Modeling
	LSTM Recurrent	Input=256	N=16	Character Language Modeling
Deep_Red_Cu	Communication (AllReduce)	all specified in the repository		N/A

Answering on SQuAD [27]. DAWNBench assesses the performance based on four metrics: training time to a specified validation accuracy, cost (in USD) of training, average latency of performing inference, and the cost (in USD) of inference. It provides reference implementations and seed entries, implemented in two popular deep learning frameworks: PyTorch [26] and TensorFlow [1]. The hyperparameters that DAWNBench considers for optimizations are optimizer for gradient descent, minibatch size, and regularization.

C. DeepBench

DeepBench [3] [4], primarily uses the neural network libraries to benchmark the performance of basic operations on different hardware. The performance characteristics of models built for various applications are different from each other. DeepBench essentially benchmarks the underlying operations such as dense matrix multiplication, convolutions, recurrent layers, and communication. For training, DeepBench specifies the minimum precision requirements as 16 and 32 bits for multiplication and addition, respectively [3]. The benchmarks are written in CUDA and thus, are more fundamental than any deep learning framework or model implementation. Additionally, there is no concept of a quality target.

With research in the field of deep learning, various other benchmarks have also appeared in the past, such as Fathom [2], Training Benchmark for DNNs (TBD) [43], etc., but our study is restricted to MLPerf, DAWNbench and DeepBench.

III. Methodology

A. System configurations

For experimentation, we used different system configs, whose hardware specifications are highlighted in Table III.

B. Benchmarks

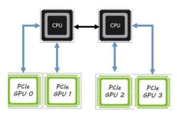
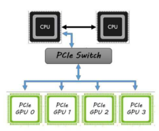
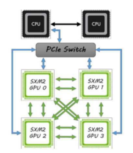
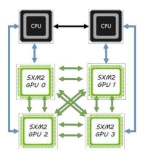
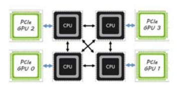
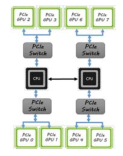
The benchmarks we chose to conduct research on are:

- GPU submissions of the MLPerf [22] training benchmarks, which were made by Google (cloud) and Nvidia (on-premise). The submitted source codes were optimized for performance on their respective hardware. Among the various submissions, we picked Google's submission on `8x Volta V100` and NVIDIA's submission on `DGX-1` as we had access to platforms with a maximum of 8 GPUs. Note that, as there was no GPU submission for *Reinforcement Learning* benchmark (one of the MLPerf training benchmarks), we exclude this benchmark from the study.
- From DAWNBench [7], for *Image Classification (CIFAR10)* training we selected the ResNet-18 implementation [5] provided by `bkj`, and for *Question Answering (SQuAD)* training we choose the DrQA implementation [41] submitted by Yang et al.
- In the case of DeepBench [3], we used four NVIDIA training benchmarks: `gemm_bench`, `conv_bench`, `rnn_bench`, and `nccl_single_all_reduce`. We omit the MPI version of `all_reduce` as our study focus on single machine. The aggregated numbers are used for all the kernels with different sizes, except for `rnn_bench`, for which we only take six configurations because the benchmark takes long time to profile.

C. Measurement tools

nvprof: We use `nvprof` profiler from CUDA-toolkit to profile the Region of Interest (ROI) in the benchmarks. Information collected are: invocation and duration of kernels,

TABLE III: Hardware specifications of systems for experimentation.

Systems	**T640**	**C4140 (B)**	**C4140 (K)**	**C4140 (M)**	**R940 XA**	**DSS 8440**
			CPUs (**Intel Xeon Gold**)			
Model #	6148	6148	6148	6148	6148	6142
Base freq.	2.40GHz	2.40GHz	2.40GHz	2.40GHz	2.40GHz	2.60GHz
			Memory (**Samsung/Micron DDR4**)			
# DIMM	12	12	12	24	24	12
Size	16GB	16GB	16GB	16GB	16GB	32GB
			GPUs (**NVIDIA Tesla V100**)			
Form Factor	PCIe Full Height/Length	PCIe Full Height/Length	SXM2	SXM2	PCIe Full Height/Length	PCIe Full Height/Length
Inter-connect	PCIe & UPI[3]	PCIe	NVLink	NVLink	UPI[3]	PCIe & UPI[3]
# GPUs	4	4	4	4	4	8
Memory	32GB HBM2	16GB HBM2	16GB HBM2	16GB HBM2	32GB HBM2	16GB HBM2
			System (**Dell PowerEdge**)			
Topology						

floating point operation counts, and memory read/write transactions. With this information, we added data points as the representatives of machine learning workloads to the roofline plot.

dstat: Additionally, we used `dstat` [37] to obtain the real-time statistics of system resource usage such as CPU usage, memory usage, disk activity, and network traffic. In UNIX platform, `dstat` gives more flexibility that combines `vmstat` (virtual memory statistics) [33], `iostat` (storage input/output statistics) [31], and `netstat` (network statistics) [32]. The statistics can then be exported to comma-separated values for further analysis. Moreover, we can extend the functionality of `dstat` by adding plugins such as one to measure NVIDIA GPU Utilization [36].

dmon: Finally, we also make use of `dmon` which is available in Nvidia System Management Interface (`nvidia-smi`) [25] to get individual GPU usage statistics that includes GPU streaming multiprocessor usage, GPU memory usage, temperature, frequency, and PCI Express bus usage. A feature to measure the NVLink bus utilization using hardware counters is also employed in `nvidia-smi`.

IV. Benchmark Analysis

The analysis is presented on the optimized codes submitted by Google and NVIDIA to MLPerf unless specified otherwise. It may be noted from the MLPerf website that only three vendors (Google, NVIDIA, and Intel) have submitted results to MLPerf, and no vendor has submitted results for all benchmarks. The effort to run MLPerf codes on the systems mentioned in Table IV was non-trivial, and some of the benchmarks are omitted from some studies due to difficulties with runs. A statistic of kernels is available online [35].

[3]UPI: Ultra Path Interconnect

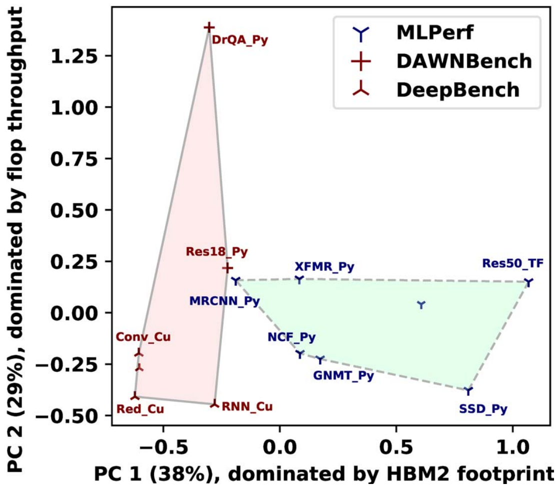
A. Similarity/Dissimilarity analysis

We perform Principal Component Analysis (PCA) on 8 collected workload characteristics (namely, PCIe utilization, GPU utilization, CPU utilization, DDR memory footprint, HBM2 footprint, flop throughput, memory throughput, and number of epochs), and visualize the distribution of the targeted machine learning benchmarks in the workload space. This analysis helps us to understand how similar and different these benchmarks are.

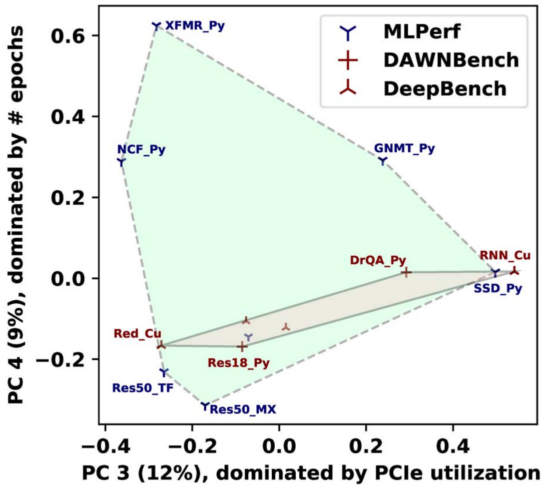
As shown in Figure 1a, MLPerf benchmarks are so different from DeepBench kernels as well as DAWNBench benchmarks on PC1, that they become two isolated clusters (with outliers labeled) sitting in two sides. PC1 is dominated by GPU memory footprint. The location in the space is actually a reflection of the fact that DeepBench kernels and DAWNBench benchmarks are working on relatively smaller datasets, and they cannot stress GPU memory as much as MLPerf benchmarks can. On the PC2 axis MLPerf benchmarks have a shorter span than other benchmark do, mainly because MLPerf benchmarks are optimized end-to-end applications, having a stable floating point operation throughput, while more diversity exists in the other benchmarks (e.g., the communication kernel Deep_Red_Cu even has zero floating point operations). MLPerf benchmarks are more sparsely-spread on the PC3-PC4 plane (Figure 1b), and cover what other benchmarks cover. The intra-suite diversity is exposed in Figure 1 as well. For PC1 to PC4 (covering 88% variance), each MLPerf benchmark gets at least one chance to extend the boundary, and there are no two MLPerf benchmarks that are very close to each other.

B. Roofline analysis

A roofline model [38] is a visual representation of the maximum attainable performance for a given workload in a given hardware by combining the processing core performance, memory bandwidth, and the data locality.



(a) PC1 - PC2



(b) PC3 - PC4

Fig. 1: The distribution of MLPerf, DAWNBench, and DeepBench in the dominant principal component workload space. The dominant metric is the one with the greatest absolute value in the eigenvector of a principal component.

Figure 2 presents the roofline model for a single Tesla V100 GPU and machine learning workloads we studied. The runs were carried out on the T640 system, invoking just one GPU. The vertical axis represents the compute capability that can be expressed usually in a unit of Floating Point Operation per Second (FLOPS/sec). Meanwhile, the horizontal axis denotes the arithmetic intensity, which is the ratio between floating point operations and data amount, using Floating Point Operations per Byte (FLOPs/Byte) as the unit. Memory-bound workloads have lower arithmetic intensity, hence their performance are limited by memory bandwidth (corresponding to the slope of the slash lines in Figure 2). Compute-bound workloads have high enough arithmetic intensities so their performance are limited by the computational resources (the horizontal lines in Figure 2). We indicate the location of different machine learning workloads with points in different shapes. Workloads from the same benchmark suite are assigned the same color. As we can see from the Figure 2, MLPerf benchmarks are more optimized than DeepBench kernels so that there is more data reuse, achieving higher arithmetic intensity, while the two DAWNBench workloads shows even higher arithmetic intensities with higher throughputs. Nevertheless, all the workloads are memory-bound (have not cross the turn point, and touch the horizontal lines). This observation implies memory is the system bottleneck for machine learning workloads, and we should dedicate more resources to memory interface for a well-balanced system.

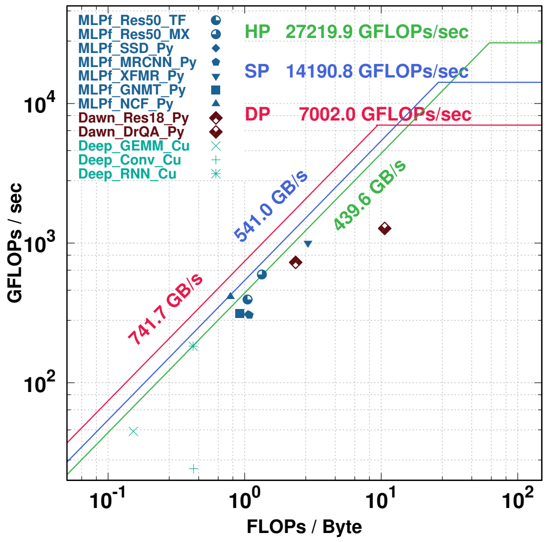


Fig. 2: V100 roofline model. Red, blue, green polylines show the empirical limitations (from available memory bandwidth and computational resources) for V100 to perform double, single, and half-precision floating point operations (measured with Empirical Roofline Toolkit [40]). MLPerf benchmarks are labeled in blue shapes, DAWNBench programs labeled in red shapes, and DeepBench programs labeled in cyan shapes.

C. Sensitivity of MLPerf models to Mixed Precision Training using Tensor Cores

Prior work ([10], [16], [20]) suggests that deep learning benefits from reduced precision in the following ways:

- Lowering the on-chip memory requirement for the neural network models.
- Reducing the memory bandwidth requirement by accessing less or equal bytes compared to single precision.
- Accelerating the math-intensive operations especially on GPUs with Tensor Cores.

Typically, only some pieces of data employ reduced precision leading to mixed precision implementations. Moreover, employing mixed precision for training is getting easier for programmers with the release of NVIDIA's Automatic Mixed Precision (AMP) [24] feature on different frameworks like TensorFlow [1], PyTorch [26] and MXNet [6]. Figure 3 shows the speedup observed in different MLPerf training benchmarks by employing half-precision along with single-precision when tested on DSS 8440 using 8 GPUs. The speedup observed is in the range of 1.5× in MRCNN_Py to 3.3× in Res50_TF. Thus, it can be inferred that MLPerf, an end-to-end benchmark suite, is capable of testing the reduced precision support of processors. For example, TensorCores are tested here.

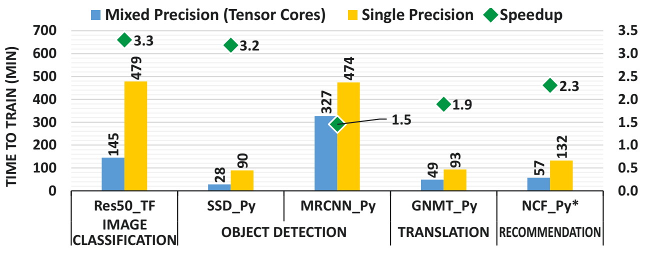
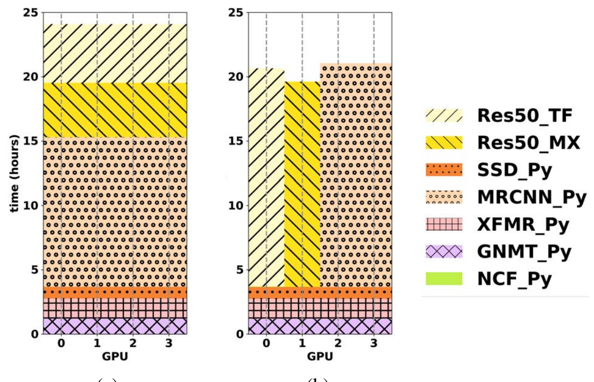


Fig. 3: Mixed Precision training (supported by Tensor Cores) results in 1.5× to 3.3× speedups over single precision. (Note the time of NCF_Py is in seconds)



(a) (b)

Fig. 4: Scheduling a mix of MLPerf workloads on 4 GPUs: (a) naive scheduling, which distributes one benchmark on all the GPUs one by one; (b) optimal scheduling, found by searching through the possible space, saves 3.0 hours.

D. Scalability of the benchmarks

TABLE IV: Scaling efficiency.

Benchmark	Training Time (min)		Scalability (speedup)			
	1×P100	1×V100	P-to-V	1-to-2	1-to-4	1-to-8
Res50_TF	8831.3	1016.9	8.68×	1.92×	3.84×	7.04×
Res50_MX	8831.1	957.0	9.23×	1.92×	3.76×	5.92×
SSD_Py	827.7	206.1	4.02×	1.94×	3.72×	7.28×
MRCNN_Py	4999.5	1840.4	2.72×	1.76×	2.64×	5.60×
XFMR_Py	1869.8	636.0	2.94×	1.42×	2.92×	5.60×
NCF_Py	46.7	2.2	21.23×	1.88×	2.16×	2.32×

The scalability study is performed on a system with 8 GPUs, the DSS 8440, where the number of GPUs employed to train the model is controlled. Ideally performance speedup of using 2 GPUs, 4 GPUs and 8 GPUs over 1 GPU should be 2×, 4× and 8×, respectively. Table IV shows the scalability trends for every MLPerf benchmark except for GNMT_Py. The training time using a single GPU is also added to provide a better understanding. We can see that some of the benchmarks like Res50_TF, Res50_MX, and SSD_Py scale well with the number of GPUs while for others increasing the number of GPUs beyond a certain point is not rewarding enough. For instance, in the case of Res50_TF, when the number of GPUs is increased from 1-to-2, from 1-to-4, and from 1-to-8, training time improves by approximately 1.9×, 3.8×, and 7×, respectively. On the contrary, for NCF_Py the speedup achieved over a single GPU is 1.9×, 2.2×, and 2.3× when the number of GPUs is increased to 2, 4, and 8, respectively. This data does not justify increasing the number of GPUs beyond 2 for training *Recommendation* benchmark. We believe the small dataset (MovieLens 20-million) causes this behavior for the benchmark. Small dataset limits the maximum batch size which as a result restricts the scalability of the benchmark. Few other benchmarks, such as XFMR_Py and MRCNN_Py fall between the most and the least scalable ones, providing a scale-up by a factor of roughly 1.6×, 2.8×, and 5.6× for 2, 4, and 8 GPUs, respectively.

Such differences in scalability between different workloads give users hints to schedule a mix of machine learning training tasks. The naive scheduling scheme, that sequentially distribute every workload to all resources at once, avoids fragmentation, and keeps the resources busy all the time. However, it may not be the most efficient way in terms of total training time, because users having multiple GPUs can choose to distribute some scalable workloads, while they decide to run workloads with poor scalability in sets simultaneously on fewer GPUs. Thus, the system administrators associated with super computing clusters might be interested in finding an effective algorithm to schedule various of machine learning training jobs submitted from researchers, developers, and all other kinds of machine learning users. To show the potential benefit, we search through all permutations of scheduling 7 MLPerf benchmarks on multiple GPUs, and Figure 4 presents 4-GPU scheduling for illustration. In each subfigure, available GPUs are listed along the x-axis, with vertical dashed lines as their timelines. Different color shades under the timeline correspond to the executions of the 7 different MLPerf workloads. Figure 4b shows the shortest scheduling of the 7 MLPerf benchmarks on 4 GPUs. Compared with the naive scheduling in Figure 4a, it saves about 3 hours to finish all the training tasks. In the optimal scheduling, the workloads chosen to be distributed on 4 GPUs, namely XFMR_Py and SSD_Py are the scalable benchmarks we observed above. MRCNN_Py gets two GPUs to execute due to its medium scalability. Two *Image Classification* workloads, Res50_MX and Res50_TF, are assigned to single GPUs separately to achieve faster training time. Note that, two similar workloads running in parallel will provide lower training time than running them in a distributed fashion even if they are highly scalable. Similarly, optimal scheduling could save around 4.1 hours and 0.4 hours for 2-GPU and 8-GPU settings, respectively. It is worth mentioning that this performance gain is without any effort in optimizing the software or adding costly hardware.

V. System Level Measurements

In this section we present observations on system level utilization measurements performed with the tools `dstat` and `dmon` in order to better understand the impact of running DL training workloads and system requirements for the different models. The experimentation is conducted on C4140 (K) system by appropriately regulating the number of GPUs.

A. CPU utilization across different workloads

In the previous section, we presented the scalability of each benchmark for 1, 2, 4, and 8 GPUs runs. Although most of the computation are offloaded to the GPUs, it will be interesting to know how the CPU is utilized during the execution of the benchmarks. We run each workload on C4140 (K) platform and configure accordingly to use 1, 2, or all 4 GPUs available on that platform and sample the CPU usage with `dstat`.

The average CPU usage while running 1, 2, and 4 GPUs is summarized in Table V. Note that, the average CPU usage includes the operating system (e.g., kernel, low-level driver) usage as well as that used by the user programs. In general, as we double the number of GPUs used to run the workloads, the CPU utilization roughly doubles. This trend is observable for all submissions to MLPerf which indicates that the CPU must have adequate performance to keep all GPUs busy otherwise it will become a bottleneck during the run.

Among the MLPerf submissions, MLPf_Res50_TF has the highest CPU Utilization followed by MLPf_Res50_MX. This is because, compared to other workloads, both *Image Classification* benchmarks require CPU to perform more packaging of the data before dispatching them to the GPUs and post-processing the data after the GPUs finish the requested tasks. Moreover, the dataset used for *Image Classification* benchmark is significantly bigger (around 300GB) compared to datasets for other benchmarks. Since it is not feasible to store such a big chunk of data on GPU memory, the CPU has to coordinate small parts of the dataset that can be stored in GPU memory at one time. The GPU can then perform a partial computation. This copying back and forth between CPU memory and GPU memory also increases the utilization of CPU. MLPf_NCF_Py shows lowest CPU utilization followed by MLPf_GNMT_Py and MLPf_XFMR_Py. The Object Detection workloads are in the middle in terms of CPU utilization.

Another observation comes from Dawn_DrQA_Py. Although this benchmark runs on a single GPU, it has the highest CPU usage of all the workloads included in the Table V. Unfortunately, this benchmark also shows least GPU utilization among all the workloads, around 20%, which indicates that a major part of the computation is performed on the CPU with few tasks that can be offloaded to the GPU.

B. GPU utilization for different workloads

The GPU utilization as given in Table V is the sum of the utilization of every GPU that is used during the runtime. Therefore, single-, dual-, and quad-GPU run will have maximum utilization of 100%, 200%, and 400%, respectively. For *Image Classification* workloads, both MLPf_Res50_TF and MLPf_Res50_MX, show near identical GPU utilization with around 85% GPU usage for single-GPU run, around 190% GPU usage (i.e., around 95% utilization per GPU) for dual-GPU run, and around 375% GPU usage (i.e., around 93.5% utilization per GPU) for quad-GPU run.

Most of the submissions to MLPerf show a similar trend for single-GPU and dual-GPU runs. Moreover, MLPf_NCF_Py shows decreasing individual GPU usage for quad-GPU run compared to dual-GPU run. This observation agrees with the one mentioned in Section IV-D that due to the limited scope of increase in the batch size for the workload, it is unable to utilize the GPUs efficiently. The increasing of communication cost for multi-GPU run that can impact individual GPU utilization is confirmed by Deep_Red_Cu benchmark from DeepBench which shows the same trend.

C. CPU and GPU memory footprint

The system memory is mostly used to store the dataset that is used for the training as well as the intermediate data required between computations. In the case when the dataset is too large to fit in the GPU memory, the system memory acts as a buffer to store the dataset. The user program will move the data back and forth between the system and GPU memory to perform partial calculations. Moreover, in an extreme case, the dataset can be too large to be stored inside the system memory. Thus the disk storage (e.g., hard disk drive, solid state drive) is used to store them, and the CPU is responsible for coordinating the switching between each part of the dataset.

From Table V, we can notice that the system memory footprint roughly doubles every time we double the number of GPUs. The GPU memory footprint is the total memory footprint for every GPU used during the run. Note that, the

TABLE V: System resource usage statistics on C4140 (K). Utilization and footprint increase along # GPUs. Note: for DeepBench G. is GEMM_Cu, C. is Conv_Cu, R. is RNN_Cu, and for DAWNBench R. is Res18_Py, D. is DrQA_Py.

		#	Util. (%)		Footprint (MB)		Bus Util. (MBps)	
		GPU	**CPU**	**GPU**	**DRAM**	**HBM**	**PCIe**	**NVLink**
MLPerf	Res50_TF	1	10.76	85.84	17,922	15,927	1,251	0
		2	16.25	188.08	18,521	31,896	2,609	967
		4	29.06	372.43	19,970	62,214	4,269	2,867
	Res50_MX	1	4.56	85.84	7,091	10,343	1,251	0
		2	9.16	190.90	14,924	20,605	6,913	1,871
		4	18.12	378.94	28,781	40,959	11,480	21,755
	SSD_Py	1	3.89	96.13	4,100	15,406	4,720	0
		2	7.21	180.58	10,305	30,772	6,998	509
		4	13.69	334.84	20,273	60,539	9,791	1,500
	MRCNN_Py	1	2.45	62.46	7,208	4,762	258	0
		2	4.83	144.40	13,561	15,933	2,219	2,472
		4	10.39	283.88	24,923	33,935	3,444	6,547
	XFMR_Py	1	1.80	91.14	3,992	14,926	47	0
		2	3.35	189.30	7,167	29,493	123	11,247
		4	6.39	376.91	14,244	58,229	249	35,862
	GNMT_Py	1	1.91	89.94	7,210	12,098	2,743	0
		2	3.32	185.71	13,561	24,479	4,609	1508
		4	6.41	360.89	24,923	46,016	7,692	33,262
	NCF_Py	1	0.76	96.39	1,550	13,870	42	0
		2	2.41	194.44	3,077	24,847	110	17,887
		4	5.69	333.11	5,978	39,634	200	75,051
Dawn	R.	1	4.67	76.90	2,670	2,056	176	0
	D.	1	48.84	20.30	6,721	2,657	52	0
DeepBench	G.	1	1.80	99.60	333	1,067	13	0
	C.	1	1.73	99.10	948	783	13	0
	R.	1	1.80	94.80	994	2,536	3,747	0
	Red_Cu	1	0.75	91.30	313	631	27	0
		2	0.96	193.20	430	994	86	77,992
		4	1.68	366.24	1123	2320	134	404,376

footprint of GPU memory depends on the batch size, and the batch sizes for the experiments are scaled accordingly from the original submissions as mentioned in Section III-B.

Although the table only shows the memory footprint of each benchmark, we would like to emphasize that the heterogeneity of the medium where the dataset is stored may become a bottleneck especially for memory-bounded applications which perform data exchange frequently. In this case, the interconnect bandwidth between each storage medium and the intelligence of the program to overlap the data transfer just before the next computation and to manage the locality of the data can play a crucial factor.

In our C4140 (K) platform, for example, each CPU has 96GB of memory consisting six 16GB DDR4-2666 DIMMs in hexa-channel memory configuration. The theoretical unidirectional memory bandwidth available to each CPU is around 128GBps [9] while the Intel's proprietary Ultra Path Interconnect (UPI) that links two CPUs has only unidirectional theoretical bandwidth of 20.8GBps [21]. In a case when a CPU needs a part of the dataset stored in other CPU's memory, the performance of data transfer will be significantly reduced (i.e., 128GBps direct access for local DRAM v.s. 20.8GBps neighbor DRAM access via UPI).

The same thing happens with a GPU that has more limited dedicated memory. In our C4140 (K) platform, each Nvidia Tesla V100 is equipped with 16GB HBM2 stacked memory which is capable of 450GB/s unidirectional bandwidth. In the case that the dataset cannot be fully stored inside the GPU memory, the CPU should bring a part of the dataset from the system memory into the GPU memory. This data exchange uses PCIe 3.0 bus which connects the CPU and GPU and able to provide theoretical unidirectional bandwidth of 15.8GBps for x16 lanes which limits the performance of data transfer.

D. System and GPU bus utilization

In the previous section, we have mentioned that interconnection bus between CPU-GPU and between GPU-GPU may play an important role in determining the overall system performance. Moreover, as we have learnt previously, the choice interconnection topology between CPU and GPU should be considered carefully. In this section, we will explain more details about how the performance is impacted by the interconnection bus based on the data on Table V.

Modern microprocessor systems use PCI Express (PCIe) bus as the interconnection standard between CPU and external peripheral that requires high-speed data communication. PCIe 3.0 standard, introduced in 2010, has been widely adopted by most computer system products available in today's market. PCIe 3.0 provides theoretical unidirectional bandwidth up-to 984.6 MBps per lane and up-to 15.8 GBps per PCIe 3.0 compatible device connected using 16 PCIe 3.0 lanes (PCIe 3.0 x16). This massive bandwidth, in theory, should be sufficient for most of the peripheral devices including GPU, network interface card, and non-volatile memory storage.

Usually a GPU is connected to the CPU using PCIe 3.0 x16 to assure that there is plenty of bandwidth between them. High bandwidth is easy to achieve for a single-GPU system, but more complicated for a multi-GPU system since the number of PCIe 3.0 lanes that the CPU has are limited. High-end Intel Xeon may have up to 48 lanes of PCIe 3.0 which are then allocated to various devices. With this constraint, each GPU on a four GPUs system, for example, may only be assigned eight PCIe 3.0 lanes. While it depends on how we use the GPU and how intense the data exchange happens between the CPU and GPU, some applications like gaming may find PCIe 3.0 x8 already provides plenty of bandwidth. On the other hand, this much bandwidth may not be optimal for deep learning training.

Alternatively, a PCIe switch, such as those manufactured by PLX Technology, can be used to provide additional PCIe lanes; thus each GPU can have PCIe 3.0 x16 lanes. This switch will be useful for GPU-to-GPU communication since the data exchange will only take place on the switch without going over to the CPU. However, the switch will not be beneficial to improve the bandwidth between CPU and all GPUs on the system as the effective CPU-to-GPU bandwidth is still limited by what the CPU has. We will discuss the interconnection topology in detail and how it affect the performance in Section V-E.

Furthermore, apart from CPU-to-GPU communication, PCIe bus can be used for GPU-to-GPU communication for a multi-GPU system. Although each GPU can be allocated with PCIe 3.0 x16 lanes, the available bandwidth may not be sufficient for some workloads that require intensive data exchange between the GPUs. Therefore, an additional bus has been developed to be used specifically for GPU-to-GPU communication such as NVLink which is high-speed proprietary interconnect system in NVIDIA GPUs. Each NVLink lane provides 25 GBps theoretical unidirectional bandwidth. The Nvidia Tesla V100 GPU in SXM2 form factor has six NVLink lanes which are capable of transferring data with theoretical unidirectional bandwidth of 150GBps. This is significantly faster than what PCIe 3.0 x16 can offer.

Table V shows the PCIe 3.0 bus utilization between CPU and GPU available on the system as well as NVLink utilization between GPU and GPU. The value presented in the table is the sum of PCIe 3.0 bidirectional PCIe bus utilization for each GPU that is used during the run, and the sum of NVLink lane utilization from each GPU used during the run. As we can see from the table, the data transfer rate over NVLink bus increases as we add more GPU for the run. The Deep_Red_Cu and the MLPf_NCF_Py use the highest bandwidth of NVLink which means that the data exchanges between GPU for those benchmark are intensive. On the other hand, the utilization of PCIe 3.0 bus increases as we add more GPU which is as expected. In a multi-GPU system equipped with NVLink, the PCIe 3.0 bus is used only for communication between CPU and each GPU because the GPU to GPU communication has been offloaded into the higher speed NVLink.

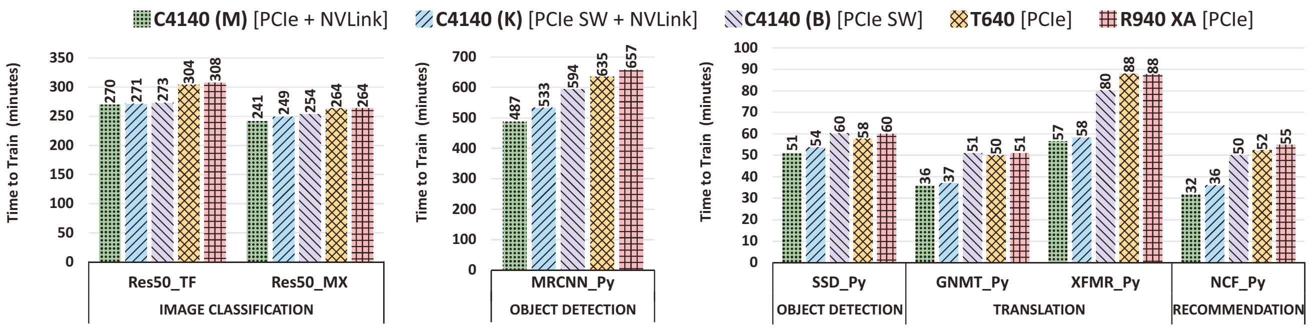


Fig. 5: Training time on 4-GPU systems (topologies shown in Table III). Time on systems with NVLink interconnect (the first 2 bars) is less than training time on the remaining systems. (Note that, the time of NCF_Py is in seconds)

E. *Impact of GPU-Interconnect Topology*

To reduce the training time it is becoming increasingly common to scale deep learning (DL) training across multiple GPUs within a system. There are multiple ways in how GPUs can be connected within the system. Primarily there are two options available - using a PCIe based interconnect (which may include PCIe switches if the number of lanes from the CPU is not sufficient) and using NVIDIA's proprietary interconnect like NVLink. The theoretical bandwidth of an NVLink interconnect is 10$\times$ higher than PCIe (300 GB/s vs. 32GB/s) [23]. Additionally, communication libraries like NCCL from NVIDIA are optimized to perform GPUDirect peer-to-peer (P2P) direct access when NVLink is available between GPUs, which can lower training times if there is significant peer-to-peer communication during model training. GPUDirect P2P is also feasible in certain PCIe topology designs where GPUs are the same PCIe domain (single root complex). Using MLPerf, we conduct a performance evaluation of five different 4-GPU platforms, each of them with a unique GPU interconnect topology. Table III shows how the GPUs are interconnected for the servers used in this study.

Two of the five servers, C4140 (M) and C4104 (K) include the high-speed proprietary NVLink interconnect to provide 100GB/s bandwidth between any two GPUs. The difference between the two NVLink based designs is the use of a PCIe switch in the C4140 (K) to aggregate the PCIe connections to the GPUs. The remaining three systems use PCIe based interconnects. They use very different approaches in how the GPUs are connected to the CPUs and in how they communicate with other GPUs. One system C4140 (B), uses a 96-lane PCIe switch that allows for 4 GPUs to be hosted in a single PCIe domain where it can perform GPUDirect peer-to-peer (P2P) between the GPUs using the PCIe switch. This is not feasible in the other two PCIe based interconnect platforms - the T640, where two GPUs are hosted per CPU and R940 XA which is a 4 CPU platform with each GPU connected directly using the PCIe lanes of the CPU.

The training times for the different servers are plotted in Figure 5 which illustrates the impact of GPU interconnect topology on DL training times. As expected, due to lack of GPUDirect P2P capability between any of the GPUs, the T640 and R940 XA take the longest time to train all the MLPerf models. Conversely, the two servers that use NVLink interconnect (the C4140 (M) and (K) systems) show the best training times across all the MLPerf models. However, the performance improvements differ depending on the model that is being trained and ranges from 42% and 17% for the *Translation* benchmarks, 30% for MLPf_MRCNN_Py to 11% for the *Image Classification* benchmarks. The C4140 (B) which uses a PCIe topology, but can perform GPUDirect P2P between GPUs due to all GPUs connected to a PCIe switch, shows performance parity to the NVLink platform for the *Image Classification* benchmarks and better performance than the R940 XA and T640 servers for remaining benchmarks. This platform provides a mix of flexibility that is available when using PCIe based GPU cards in addition to higher performance over PCIe based designs that do not support GPUDirect P2P transactions between GPUs.

VI. Conclusion

We have presented a detailed characterization of the recent MLPerf benchmark suite in this paper. While MLPerf benchmark characteristics may be heavily influenced by the specific implementations, the suite does provide a diverse set of benchmarks which allows to unveil various bottlenecks in the system. Our experiments point towards (i) the importance of powerful interconnects in multi-GPU systems, (ii) the variation in scalability exhibited by different ML models, (iii) the opportunity for smart scheduling strategies in multi-gpu training exploiting the variability in scaling, and (iv) the need for powerful CPUs as host with increase in number of GPUs.

We also present the dissimilarity of the benchmarks to other benchmarks in the suite (intra-suite dissimilarity) and dissimilarity against other suites such as DAWNBench and DeepBench (inter-suite dissimilarity). MLPerf provides benchmarks with moderately high memory transactions per second and moderately high compute rates. DAWNBench creates a high-compute benchmark with low memory transaction rate, whereas DeepBench provides low compute rate benchmarks. The various MLPerf benchmarks show uniqueness such as high NVLink utilization in NCF_Py, low NVLink utilization in SSD_Py, near-perfect scalability with increasing GPU counts in Res50_TF and SSD_Py, and low scalability in NCF_Py.

Acknowledgment

This research was supported in part by National Science Foundation under grant numbers 1725743, 1745813, and 1763848. Authors would also like to acknowledge computational resources from Texas Advanced Computing Center (TACC) and Dell EMC Corporation. Any opinions, findings, conclusions or recommendations are those of the authors and not of the National Science Foundation or other sponsors.

References

[1] 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. [Online]. Available: http://tensorflow.org/

[2] R. Adolf, S. Rama, B. Reagen, G.-Y. Wei, and D. Brooks, "Fathom: Reference workloads for modern deep learning methods," 2016.

[3] Baidu, "Deepbench: Benchmarking deep learning operations on different hardware," 2017. [Online]. Available: https://github.com/baidu-research/DeepBench

[4] Baidu, "An update to deepbench with a focus on deep learning inference," 2017. [Online]. Available: https://github.com/baidu-research/DeepBench

[5] bkj, "Resnet18 + minor modifications (submission at DAWNBench)," https://github.com/bkj/basenet/tree/49b2b61e5b9420815c64227c5a10233267c1fb14/examples, 2018.

[6] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *CoRR*, vol. abs/1512.01274, 2015. [Online]. Available: http://arxiv.org/abs/1512.01274

[7] C. A. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, "Dawnbench : An end-to-end deep learning benchmark and competition," in *NIPS ML Systems Workshop*, 2017.

[8] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 248–255.

[9] FUJITSU, "White paper fujitsu server primergy & primequest memory performance of xeon scalable processor(skylake-sp) based systems," https://sp.ts.fujitsu.com/dmsp/Publications/public/wp-skylake-memory-performance-ww-en.pdf, 2018.

[10] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," 2015.

[11] L. Gwennap, "Ai benchmarks remain immature," *Microprocessor Report*, January 28, 2019.

[12] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask r-cnn," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2018.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," 2016.

[15] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," 2017.

[16] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," 2016.

[17] A. Krizhevsky, "Learning multiple layers of features from tiny images," https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf, 2009.

[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," 2015.

[19] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, D. Brooks, D. Chen, D. Dutta, U. Gupta, K. Hazelwood, A. Hock, X. Huang, B. Jia, D. Kang, D. Kanter, N. Kumar, J. Liao, D. Narayanan, T. Oguntebi, G. Pekhimenko, L. Pentecost, V. J. Reddi, T. Robie, T. S. John, C.-J. Wu, L. Xu, C. Young, and M. Zaharia, "Mlperf training benchmark," 2019.

[20] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," 2017.

[21] Microway, "Performance characteristics of common transports and buses," https://www.microway.com/knowledge-center-articles/performance-characteristics-of-common-transports-buses/, 2019.

[22] "MLPerf," https://mlperf.org/, MLPerf, 2018.

[23] NVIDIA, "Nvidia tesla v100 gpu accelerator," https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf, 2018.

[24] NVIDIA, "Automatic mixed precision (amp)," https://developer.nvidia.com/automatic-mixed-precision, 2019.

[25] NVIDIA Corporation, "Nvidia system management interface program," https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf, 2016.

[26] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[27] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," 2016.

[28] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

[29] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, 10 2017.

[30] "Minigo: A minimalist Go engine modeled after AlphaGo Zero, built on MuGo," https://github.com/tensorflow/minigo, tensorflow.

[31] The FreeBSD Project, "Iostat: I/o statistics tool," https://www.freebsd.org/cgi/man.cgi?query=iostat&manpath=FreeBSD+12.0-RELEASE+and+Ports.

[32] The FreeBSD Project, "Netstat: Network status and statistics tool," https://www.freebsd.org/cgi/man.cgi?query=netstat&sektion=1&manpath=FreeBSD+12.0-RELEASE+and+Ports.

[33] The FreeBSD Project, "Vmstat: Virtual memory statistics tool," https://www.freebsd.org/cgi/man.cgi?query=vmstat&sektion=8&manpath=FreeBSD+12.0-RELEASE+and+Ports.

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[35] S. Verma, Q. Wu, B. Hanindhito, G. Jha, E. B. John, R. Radhakrishnan, and L. K. John, "Demystifying the mlperf benchmark suite," 2019. [Online]. Available: https://arxiv.org/abs/1908.09207

[36] V. Vryniotis, "Nvidia gpu utilization plugin for dstat," https://raw.githubusercontent.com/datumbox/dstat/master/plugins/dstat_nvidia_gpu.py, 2017.

[37] D. Wieërs, "Dstat: Versatile resource statistics tool," http://dag.wiee.rs/home-made/dstat/.

[38] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009. [Online]. Available: http://doi.acm.org/10.1145/1498765.1498785

[39] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016.

[40] C. Yang, "Berkeley cs roofline toolkit," https://bitbucket.org/berkeleylab/cs-roofline-toolkit.

[41] R. Yang, Facebook-ParlAI, and B. Koonce, "DrQA (submission at DAWNBench)," https://github.com/hitvoice/DrQA, 2018.

[42] C. Young, "Why machine learning needs benchmarks," *Computer Architecture Today, ACM SIGARCH*, 2018.

[43] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Phanishayee, B. Schroeder, and G. Pekhimenko, "Tbd: Benchmarking and analyzing deep neural network training," 2018.