BenchCouncil Transactions on Benchmarks, Standards and Evaluations 1 (2021) 100005

Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: https://www.keaipublishing.com/en/journals/benchcouncil-transactions-on-benchmarks-standards-and-evaluations/

Comparative evaluation of deep learning workloads for leadership-class systems☆

Junqi Yin *, Aristeidis Tsaris, Sajal Dash, Ross Miller, Feiyi Wang, Mallikarjun (Arjun) Shankar

Oak Ridge National Laboratory, United States of America

ARTICLE INFO

Keywords:
CORAL benchmark
Deep learning stack
ROCm

ABSTRACT

Deep learning (DL) workloads and their performance at scale are becoming important factors to consider as we design, develop and deploy next-generation high-performance computing systems. Since DL applications rely heavily on DL frameworks and underlying compute (CPU/GPU) stacks, it is essential to gain a holistic understanding from compute kernels, models, and frameworks of popular DL stacks, and to assess their impact on science-driven, mission-critical applications. At Oak Ridge Leadership Computing Facility (OLCF), we employ a set of micro and macro DL benchmarks established through the Collaboration of Oak Ridge, Argonne, and Livermore (CORAL) to evaluate the AI readiness of our next-generation supercomputers. In this paper, we present our early observations and performance benchmark comparisons between the Nvidia V100 based Summit system with its CUDA stack and an AMD MI100 based testbed system with its ROCm stack. We take a layered perspective on DL benchmarking and point to opportunities for future optimizations in the technologies that we consider.

1. Introduction

The share of deep learning (DL) scientific applications has steadily increased in the allocation portfolio among High-Performance Computing (HPC) centers. In recent years, it has reached a tipping point that the procurement of next-generation HPC infrastructures has to take the performance of the DL stack into consideration. In the case of DOE leadership class platforms, a ***C***ollaboration of ***O***ak ***R***idge, ***A***rgonne, and ***L***ivermore (CORAL) has established a set of benchmarks to gauge the hardware/software competitiveness. For the first time in the CORAL-2 benchmarks [1] suite, DL workloads are included in the evaluation for the acquisition of the systems: Frontier at Oak Ridge, Aurora at Argonne, and El Capitan at Livermore. Ranging from DL kernels to distributed training, the CORAL-2 DL benchmarks consist of micro-benchmarks, such as DeepBench [2], and DL suites including both ResNet50 on ImageNet [3] and application benchmarks such as the cancer distributed learning environment (CANDLE) [4]. Comparing to the industry-led benchmarking effort, MLCommons HPC (also referred to as MLPerf HPC [5]), the CORAL-2 benchmarks focus more on throughput and fundamental building blocks.

Regardless of the increasing complexities of deep neural net (DNN) models, the compute operations essentially boil down to three types of mathematical kernels, i.e., general matrix multiply (GEMM), convolution, and recurrent operation. Considering that distributed training at scale has became a common practice at data centers, the communication operation has to be taken into account as well. The overall performance of DL applications is hence mostly determined by the hardware/software stack for the aforementioned three mathematical and one communication operations. (While I/O is also an important determining factor, the benchmarks we consider here do not face an I/O bottleneck when high-performance node local storage, e.g., SSD, is used for the data and proper pipelining practices are followed.)

Different from simulation codes that traditionally dominate HPC workloads, DL applications rely heavily on the underlying frameworks, e.g., TensorFlow [6] and PyTorch [7], which provide all the building blocks from model components to training and inference supports. On the one hand, this ecosystem lowers the barrier for DL application developers; on the other hand, it requires hardware vendors to provide an optimized DL software stack to support high-level frameworks.

Currently, Nvidia GPUs are the major platforms for DL workloads, and the corresponding software stack, i.e. CUDA [8], cuDNN [9], and

* Corresponding author.
E-mail addresses: yinj@ornl.gov (J. Yin), tsarisa@ornl.gov (A. Tsaris), dashs@ornl.gov (S. Dash), rgmiller@ornl.gov (R. Miller), fwang2@ornl.gov (F. Wang), shankarm@ornl.gov (M. Shankar).

https://doi.org/10.1016/j.tbench.2021.100005
Received 6 August 2021; Received in revised form 11 October 2021; Accepted 20 October 2021
Available online 11 November 2021

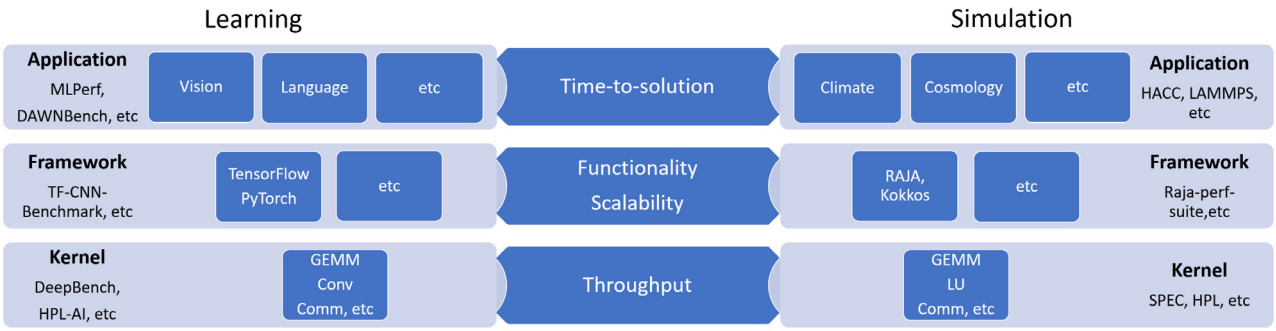

Fig. 1. Comparisons of simulation and learning benchmarks. The overall targets are similar, but facility focus varies due to distinctive development characteristics, e.g. framework plays a much bigger role in learning.

NCCL [10], are the dominant workhorses. As its counterpart, AMD GPUs and the associated ROCm [11], MIopen [12], and RCCL [13] stack, provide a similar ecosystem for DL applications. Though the Nvidia stack is more mature and widely deployed, the AMD stack is entirely open-sourced and progressing, and both platforms are supported by popular DL frameworks such as TensorFlow and PyTorch.

Employing the CORAL-2 DL benchmarks, in this paper we evaluate the performance of an early-access testbed for the upcoming Frontier Exascale system. From kernel primitives, model workloads, to framework and applications, we systematically explore benchmark performance differences between the MI100 based testbed with ROCm stack and the V100 based Summit [14] system with CUDA stack. Our contributions are the following,

- From the perspective of HPC facilities, we propose a layered approach and associated metrics, establish Roofline model, and FOM (Figure of Merits) to evaluate DL workloads from primitive kernels, popular models, to frameworks and applications.
- We provide the first look at an early-access emerging platform based on AMD MI100 GPUs, and show the performance comparisons against a top Nvidia V100 based system in production today.
- We introduce and leverage machine learning (ML) methods (XGBoost [15]) to model the relationship between input parameters and performance outcomes. It lays the groundwork to identify dominant factors to consider for further and future optimizations.
- We show an one-on-one comparison of the resource utilization for our two DL stacks on the same workloads.

The rest of the paper is organized as follows: Section 2 provides general background on differentiating aspects of traditional simulation-based HPC workloads versus emerging DL workloads, as well as an overview of DL benchmarks proposed for the CORAL systems. Section 3 details a layered approach, methodology, and metrics we will use for performance evaluation and comparison. Section 4 presents our results based on the proposed methodology covering compute kernel, model and workloads, frameworks, and applications, which aims to provide an end-to-end perspective on key performance metrics. Section 5 presents our conclusions and discusses opportunities for future work.

2. Background and overview

With the rise of DL applications and specialized hardware, DL benchmarking [16] has attracted a lot of attentions recently. Ranging from application level benchmarks, such as MLPerf, to kernel and model level benchmarks, such as DeepBench and HPL-AI [17], the scope touches almost every aspect of DL. The areas of focus, however, are quite different, as shown in Fig. 1. For application developers, the time-to-solution matters most. But for an emerging field such as DL, where the scientific DL community codes are still maturing in comparison to well-adopted simulation codes (e.g., LAMMPS [18]), understanding the kernel performance is of greater interest.

Table 1
CORAL-2 kernel, model workload, framework, and application benchmarks for learning.

Type	Benchmark	Task	Distributed
Kernel	DeepBench	GEMM	N
		CNN	N
		RNN	N
	N/RCCL-tests	Allreduce	Y
		Allgather	Y
		Reduce	Y
		ReduceScatter	Y
Model Workload	Deep Learning Suite	AlexNet	N
		GoogleNet	N
		OverFeat	N
		VGG	N
		RNN-Net	N
Framework	TF_CNN_Benchmark	ResNet50	Y
Application	CANDLE	P1B1	N
		P3B1	N

At HPC facilities, we make the following observations regarding traditional simulation and DL applications:

1. Unlike simulation applications, most DL applications strongly depend on the frameworks, and are implemented in high-level scripting languages and use pre-compiled framework binaries at run time.
2. The number of DL frameworks are converging to the two most popular ones, i.e., TensorFlow and PyTorch, while the adoption of simulation frameworks (e.g., RAJA, Kokkos) is still at a relative low level.
3. DL frameworks hide most of the complexities in code porting and optimization from developers, since hardware vendors of GPU, TPU, etc., generally upstream the optimized DL stack support to frameworks.

Overall, most DL developers interact mainly with frameworks (e.g., TensorFlow and PyTorch) in Python, and are transparent to underlying compute kernels and platform. This is one of the major distinctions from simulation codes, where the programming framework (e.g., C/C++/Fortran) provides merely basic APIs. In light of the above observations, we focus more on DL primitives and frameworks in facility benchmarking instead of application-level benchmarks. Nevertheless, an end-to-end application benchmark (CANDLE) is included to show the performance of the overall pipeline. A side by side comparison of key components of the DL and the traditional simulation stack is shown in Fig. 1.

CORAL-2 DL Benchmarks In Table 1, we list the benchmarks under study in this work. It covers key DL primitives such as operations for convolution, recurrent neural network (CNN/RNN), and model workloads, frameworks, and applications representative to HPC facilities.

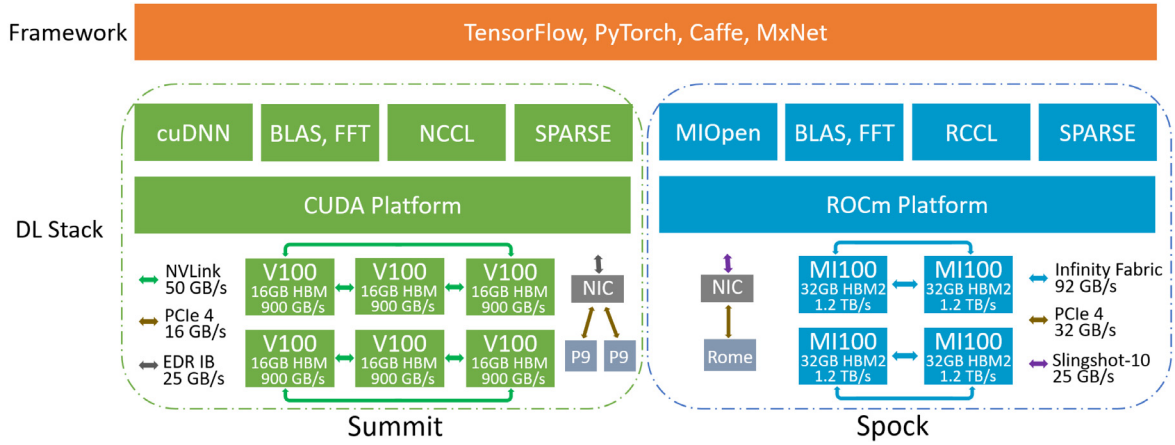

Fig. 2. The node architecture, DL core stack, and supporting framework for Summit and Spock systems.

The kernel benchmarks include DeepBench on a single device and N/RCCL tests for cross device communication. The model workloads consist of CNN models such as AlexNet [19], GoogleNet [20], etc., and an RNN model. The TF_CNN_Benchmarks [21] is used to evaluate TensorFlow framework for data parallel training. The CANDLE application is used to benchmark overall time-to-solution. In all, the scope involves a full spectrum of DL benchmarks corresponding to the application layer down to the foundational kernels as shown in Fig. 1.

DL Stack To evaluate the AMD and Nvidia DL stack, we execute the CORAL-2 benchmarks on both the Summit supercomputer and a testbed system for Frontier called Spock [22]. The node configurations of these two systems are shown in Fig. 2. Each Summit node is equipped with 6 Nvidia Volta GPUs (V100) and 2 IBM Power9 (P9) CPUs. Pairs of 3 V100s are fully connected with NVLink fabrics of 50 GB/s bandwidth, and nodes are then connected via EDR InfiniBand with a capability of 25 GB/s. Spock is an early-access system with an architecture similar to Frontier's but is a generation earlier in accelerator technology (MI100) compared to Frontier (MI200). Each Spock node is equipped with 4 AMD Instinct MI100 GPUs and 1 EPYC 7662 Rome CPU. All 4 MI100s are connected with each other using 92 GB/s Infinity Fabric, and nodes are connected via Slingshot-10. The node local storage are not illustrated because this study focuses on accelerator devices and associated software stack.

For DL frameworks, the support of different accelerator hardware (e.g. GPU, TPU, ARM) requires the corresponding linear algebra software for the devices. As shown in Fig. 2, for Nvidia GPUs, DL primitives of the CNN/RNN etc., are provided via cuDNN on top of the CUDA platform. Depending on the implementations (e.g., CNN can be based on matrix multiplication, Fourier transform, etc.), cuBLAS or cuFFT can be invoked. Similarly, MIOpen is the core DL primitive library for AMD GPUs on top of the ROCm platform, and works with rocBLAS, rocFFT, etc., to support upper level frameworks. In terms of the support for scaling up DL operations, both Nvidia and AMD provide a GPU direct communication library, i.e., NCCL and RCCL, respectively. The following studies are performed with CUDA v10.2, ROCm v4.1, and TensorFlow v2.3.

3. Methodology and metrics

Depending on the category and purpose (See Fig. 1 and Table 1) of the benchmarks, different metrics are utilized. Typically, for throughput benchmarks, floating point operations per second (FLOPS) is used and a similar metric in DL is the processed data samples per second (e.g., images/s [21]). For distributed DL in framework scalability benchmarks, we measure the scaling efficiency in terms of throughput. Application benchmarks usually resort to the end-to-end time-to-solution. To calculate the FLOPS for the GEMM operation, the formula

Fig. 3. The illustration of the key parameters in the GEMM.

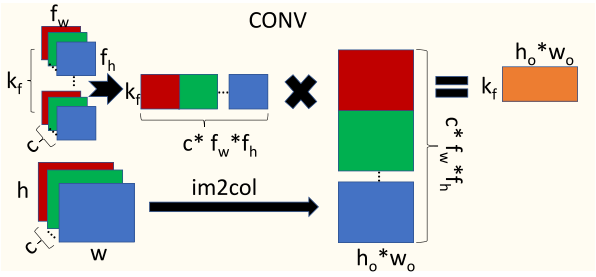

Fig. 4. The illustration of the key parameters in the convolution. It can be converted into matrix multiplication via "im2col" [9].

is defined as:

$$\text{FLOPS}_{\text{GEMM}} \sim 2 \times m \times n \times k/t, \tag{1}$$

where (m, k) and (k, n) are matrix dimensions as shown in Fig. 3, and t is the measured run time.

Since key compute operations in both CNN and RNN can be broken into matrix multiplications (See Figs. 4 and 5), the FLOPS formulas follow a similar scheme. (There are other types of implementations for convolution, e.g., Winograd and FFT [23] - for the simplicity of discussion, we focus on the GEMM based implementation.)

For the 2D Convolution operation (GEMM based) on input dimension of height h, width w, and channel c, FLOPS is calculated via,

$$\text{FLOPS}_{\text{Conv2D}} \sim 2 \times \left(h_o \times w_o\right) \times k_f \times \left(c \times f_w \times f_h\right)/t \tag{2}$$

$$h_o = \frac{\left(h + 2 \times \text{pad}_h - f_h\right)}{\text{stride}_h} + 1 \tag{3}$$

$$w_o = \frac{\left(w + 2 \times \text{pad}_w - f_w\right)}{\text{stride}_w} + 1 \tag{4}$$

where k_f is the number of filters each of dimension (f_h, f_w) with padding (`pad`) and `stride` specified in h and w dimension, respectively, and h_o and w_o [9] are the effective height and width after applying a filter.

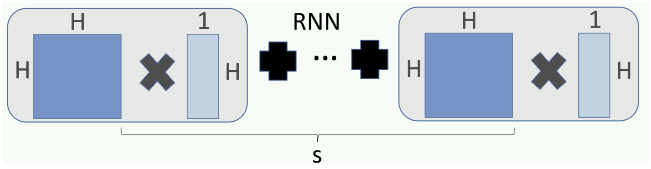

Fig. 5. The illustration of the key parameters in the RNN operation. The basic building block is also matrix multiplication.

Similarly, for the RNN operation (See Fig. 5), the FLOPS calculation follows,

$$\text{FLOPS}_{\text{RNN}} \sim 2 \times H \times H \times s/t, \tag{5}$$

Where H and s are the hidden size and time steps, respectively. For the data input with N samples (i.e., batch size N), the FLOPS for the operation will be simply multiplied by a factor of N.

Roofline Model In addition to FLOPS, another important metric to gauge the compute and memory performance is the so called Roofline model, which can visually demonstrate the bottleneck of the benchmark and hardware, i.e., whether it is compute or memory bound. To that end, the arithmetic intensity I, i.e., floating operations per memory load, needs to be calculated. For single-precision GEMM, this is given by,

$$I = \frac{FLOPs}{4(m * k + k * n + 2n * m)} \tag{6}$$

assuming the ideal data re-use of the two input matrices of element size $m * k$ and $k * n$. The Roofline model is then obtained by plotting the performance (FLOPS) versus the arithmetic intensity (FLOPs/bytes).

Figure of Merit Regardless of the types of the benchmarks, a relative metric, i.e., figure of merit (FOM), is often used in procurement. In this study, it is defined as follow,

$$\text{FOM} = \prod_{i}^{N} \left(\frac{\text{metric}^i}{\text{metric}^i_b} \right)^{1/N} \tag{7}$$

where the $metric^i_b$ is for the performance metric of ith task on the baseline system. To account for a balanced performance, the geometric mean is taken over either N sub-tasks within the benchmark or across N benchmarks. The metric for each sub-task or benchmark can be aforementioned FLOPS, images/s, scaling efficiency, or time-to-solution.

ML modeling The performance of DL kernels depends on many factors including algorithm, implementation, input shape, etc. It is hard to predict kernel runtime especially when there are multiple algorithms for the same operation (e.g., convolution) and built-in heuristics (e.g., in cuDNN, FFT-based convolution is used when f_h or f_w is bigger than 5) to select the algorithm at runtime. For the closed-source library such as cuDNN, it becomes even more challenging.

To identify the important parameters on kernel performance, we use XGBoost [15] to model the relationship between input parameters and performance outcome, and then rank the parameter based on its feature importance. Because the features are well-structured (in contrast to text and image) and limited in size, the traditional ML method such as XGBoost is well suited for the task.

Resource Utilization Another important way of understanding the performance of deep learning applications is by tracking resource utilization. This is typically used to find bottlenecks of the workload and identify operations that need optimization. In this work we use the `nvidia-smi` for the V100 GPUs on Summit and the `rocm-smi` for the MI100 GPUs on Spock to monitor the memory used and the GPU utilization for the framework and application benchmarks. Specifically the `memory.used` and the `utilization.gpu` flags were used for the `nvidia-smi`, and the `showuse` and `showmemuse` for the `rocm-smi`.

Even though those low level tools may not have been configured the same way, it is important to show early their default behavior on deep learning workloads, so that further optimization strategies can be made as more realistic HPC/DL workloads are applied. For example one noticeable difference from the documentation provided for those tools is that `nvidia-smi` sample period may be between 1 s and 1/6 s depending on the product, where `rocm-smi` samples every millisecond. Also higher level custom profilers usually use directly those low level tools, and by showing those results we hope to give a better understanding for the future developers on the current status.

The strategy is to request data from those tools on each batch/epoch iteration on the training stage, rather than monitoring the benchmark application itself. This way we can better focus on comparison between training steps, and eliminate differences between job schedulers or initial environment/system conditions between Spock and Summit, which might change over time. In all cases the flag `TF_FORCE_GPU_ALLOW_GROWTH` was used as true for better comparison between the two.

4. Evaluation results

Following the approach described in Section 3, we perform systematic evaluations of the DL stack on Summit and Spock system in terms of kernel, model, framework, and application benchmarks.

4.1. Kernel benchmarks

As previously discussed, we focus on the performance characteristics of kernel and model workloads (listed in Table 1) because they serve as common denominators across DL applications. For example, in DeepBench, the inputs for GEMM, CNN, and RNN kernels are selected from representative real DL workloads.

For kernel benchmarks, we employ DeepBench and N/RCCL tests for computing and communication primitives, respectively. These kernels usually account for a single tensor/layer operation of a neural network. Moving one level up, the workload benchmarks put together the kernel operations for popular DL models. Considering DL frameworks operate in single precision by default, we evaluate the kernels and model workloads in the same single precision.

Compute Kernels In Fig. 6, the generated FLOPS (See Eqs. (1), (3)) of a single device on Summit and Spock are plotted for a list of GEMM, CNN, and RNN operations, respectively. For GEMM, MI100 performs better for more computationally expensive operations, while for less expensive ones, the performance differences between MI100 and V100 are generally small. Of the predefined inputs (see examples in Table 4) in DeepBench, the best performance of 17.7 (77% of peak) and 14.7 (93% of peak) TFLOPS are achieved for MI100 and V100, respectively. The corresponding input parameters are annotated in Fig. 6 (GEMM), i.e., $(m = 6144, n = 48000, k = 2048)$ for MI100 and $(m = 4096, n = 7000, k = 4096)$ for V100. For ill-shaped inputs, e.g., $(m = 512, n = 8, k = 500000)$, the kernels become memory bound. For CNN, specifically the GEMM based (so called "im2col" implementation, see Fig. 4) 2D convolution, the MI100 outperforms V100 in most of cases, and a similar 79% of peak is obtained while 50% for V100 for the best case. On the other hand, V100 seems to perform better for RNN kernels, especially for larger inputs, but because RNN is more memory intensive, both devices are far below the peak compute performance.

To provide an overall comparison, in Fig. 7, we plot the FOM for all three kernel types combining performances for various inputs. With Summit as the baseline (FOM=1), Spock performs similarly for GEMM and RNN, and shows an edge over CNN. Considering the run time of a model usually dominated by the most expensive layer, we also calculate the FOM for the top 10 most expensive kernel operations. Spock shows a 20% and 10% advantage for GEMM and CNN, respectively.

According to the Roofline model, as shown in Fig. 8, the boundary for the memory and compute capability is 17.4 and 19.2 for V100

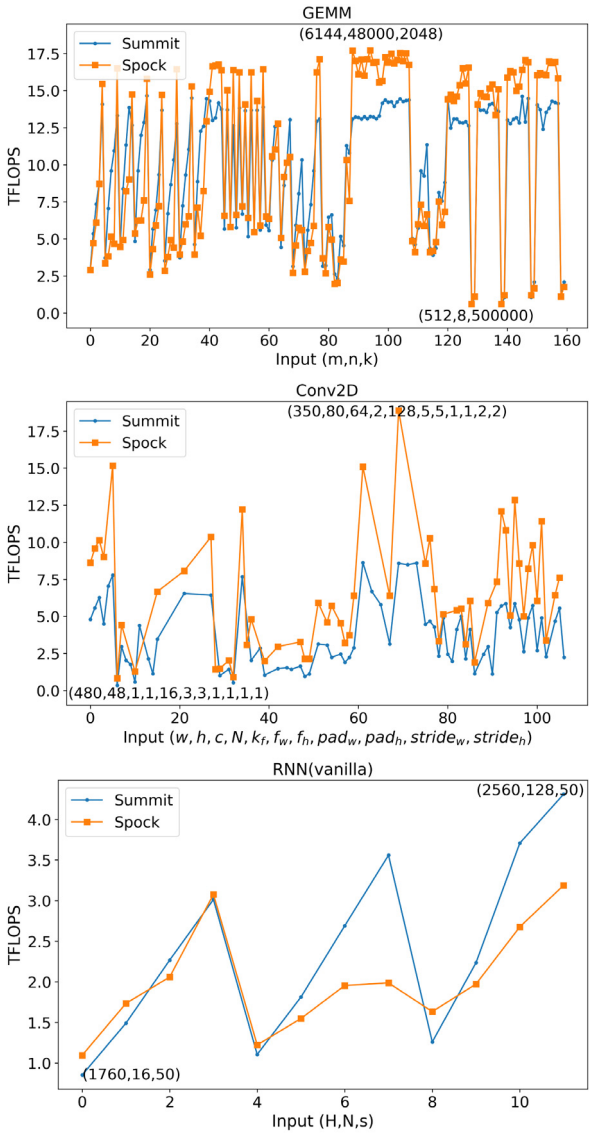

Fig. 6. The FLOPS of the GEMM, CNN, and RNN primitive operations for identified representative inputs in DeepBench. The parameters are annotated for the best and worst performance, respectively.

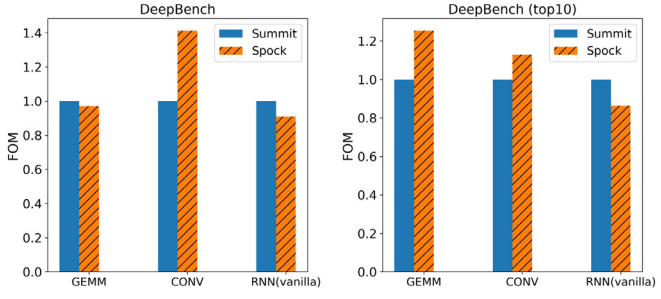

Fig. 7. The aggregated FOM of DeepBench benchmark for all and top 10 most expensive tasks, respectively.

on Summit and MI100 on Spock, respectively. In both regions (left and right of the dashed boundary line), data points for Summit is closer to the upper limit, i.e. maximum bandwidth and theoretical peak

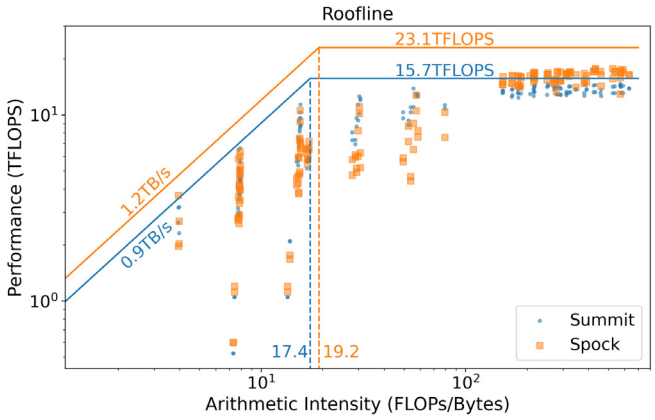

Fig. 8. The Roofline model for DeepBench GEMM benchmark (FP32).

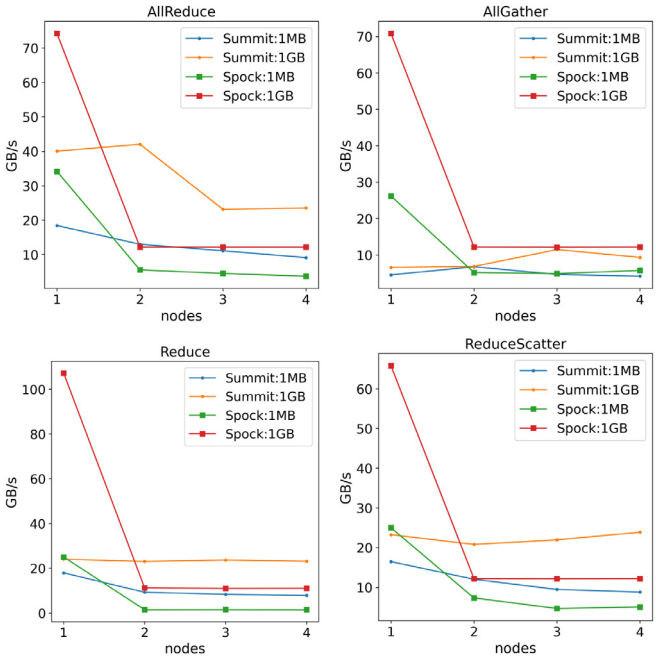

Fig. 9. The bandwidth of typical DL communication kernels up to four nodes on Summit and Spock.

(annotated in the plot) than those of Spock, indicating that there are still room for optimization in ROCm DL stack.

Communication Kernels Given that distributed training has became common practice to manage ever-growing data and model sizes, the communication kernels play increasingly important roles. For the popular data parallel training (each device has a model replica working on different data batch, and the gradient information is exchanged periodically), `allreduce` is the dominant communication pattern that is executed each (synchronized) or a few (stale or asynchronized) batch steps. Depending on the implementation, the `allreduce` can be realized via a single API or a combination of `allgather` and `reducescatter`, or `reduce` and `broadcast`. The performance depends on device communication libraries (e.g., N/RCCL) and the specific network topology of the platform.

In Fig. 9, we plot the bus bandwidth (GB/s) up to four nodes on Summit and Spock for four commonly used communication APIs in N/RCCL. The message size ranges from small (1 MB) to large (1 GB), covering the gradient size for popular DL models (e.g., 100 MB for ResNet50). For the intra-node communication, Spock shows an up to 3x lead across the board, thanks to high-bandwidth Infinity Fabric (see

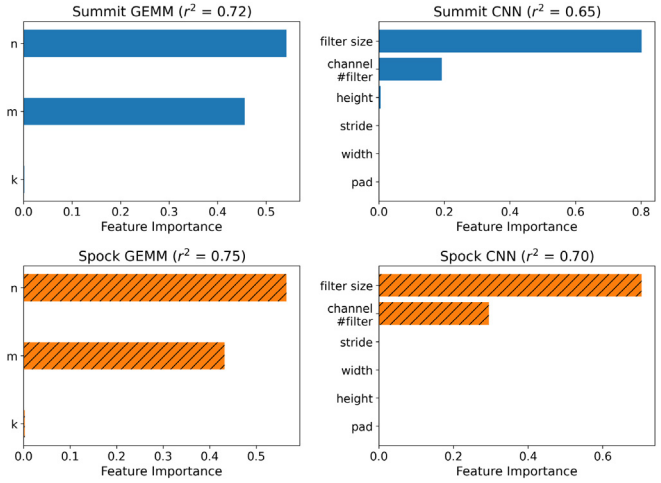

Fig. 10. The feature importance of XGBoost modeling of GEMM and CNN benchmarks in DeepBench. The ratio of explained variance (r^2) is listed.

Fig. 2). In the case of inter-node, Spock seems to perform better for `allgather` but lags behind in others, which is due to the slower PCIe connection comparing to Summit's NvLink between CPU and GPU. It indicates that a combination of inter-node `allgather` and intra-node `reducescatter` is the best way to realize the gradient `allreduce` on this particular system. Note that the network topology of the Frontier system will be significantly different from that of Spock.

Machine Learning on DL To further understand how the input parameters affect the kernel performance, we use the ML method to model the performance data on DL kernels. Because both NCCL and RCCL are open sourced and communication optimization typically relies on the framework-level libraries (e.g., torch.DDP, TF.distribute, Horovod) to overlap communication with computation, we mainly focus on the compute kernels. Because the input features are well structured, XGBoost method is used to model the relationship between input parameters and kernel performance. As shown in Fig. 10, the feature importance for GEMM and CNN are quite similar on both Summit and Spock, respectively, i.e., for GEMM, since in DL operations it is often between a squared matrix and an ill-shaped one (see example input parameters in Table 4), the run time can be well predicted by the shape of resulted matrix; for CNN (GEMM based), the filter size and input channel (often strongly correlated with number of filters) are dominant factors for run time. We can thus hypothesize that the implementations of GEMM based convolution are similar in cuDNN and MIOpen.

4.2. Model benchmarks

Putting together the tensor/layer operations, workload benchmarks on popular DL models show the combined performance of typical DL workloads. Because accelerators are of primary interest here, we focus on compute workloads and isolate them from the noise from I/O and communication.

In Table 2, we list the operation breakdown for the candidate CNN models. The model size ranges from 61M (AlexNet [19]) to 146M (OverFeat [24]) parameters with the number of convolution layers from 5 to 57. Comparing the earlier AlexNet to VGG [25], and then to ResNet50 and GoogleNet, the trend in DL modeling favors deeper models with relatively thin layers.

The number of parameters in a model is counted by

1. for a convolution layer with input channel c and k filters, each of size (f_w, f_h), the number of parameters is $c * k * f_w * f_h + k$.
2. for a recurrent layer with input size n and H hidden units, the number of parameters is the same as a feed-forward neural network, i.e. $H * (H + n) + H$.

Table 2
CORAL-2 CNN model workloads.

Model	# conv layers	Filter size	# filters	# weights	# MACs	% conv
AlexNet	5	3,5,11	96–384	61M	724M	92
OverFeat	5	3,5,11	96–1024	146M	2.8B	95
VGG	13	3	64–512	138M	15.5B	99
ResNet50	53	1,3,7	64–2048	25.5M	3.9B	99
GoogleNet	57	1,3,5,7	16–384	7M	1.4B	~100

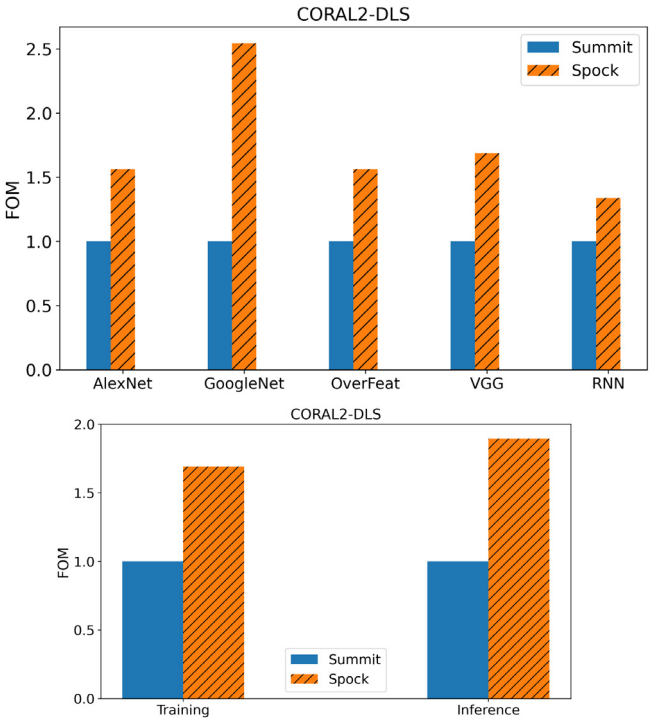

Fig. 11. The FOM for individual model workload and combined training and inference benchmarks in CORAL-2 DLS.

The corresponding multiply and accumulation (MAC) operations follow similar FLOPS counts discussed in Section 3. From Table 2, VGG is the most computationally expensive model with 15.5B MAC operations.

In Fig. 11, we plot the performance comparisons for DL model workloads. The FOM numbers are calculated from the processed samples/s with Summit being the baseline. Spock shows better performance across the board with the best FOM (~2.5x) for GoogleNet. To obtain an overall view for DL training and inference, we further break down the run time for forward pass (inference) and forward–backward pass (training), and calculate the FOM across all workload tasks. It is shown (See Fig. 11) that a speedup of 1.7x and 1.9x is achieved on Spock for training and inference, respectively. Note this is with the CORAL-2 deep learning suite (DLS) baseline implementation (TensorFlow, single precision).

4.3. Framework benchmarks

Although model workloads, as discussed in Section 4.1, already exercise the framework on a single device, there are many other aspects of the framework that require further examination. To this end, we use the `TF_CNN_Benchmark` to perform the distributed training on ResNet50, which is required by CORAL-2 DLS.

Functionality In terms of the performance functionalities, both TensorFlow and PyTorch support automatic mixed precision, runtime compilation e.g., accelerated linear algebra (XLA), etc. Frameworks operate in single precision by default because the mixed precision requires special

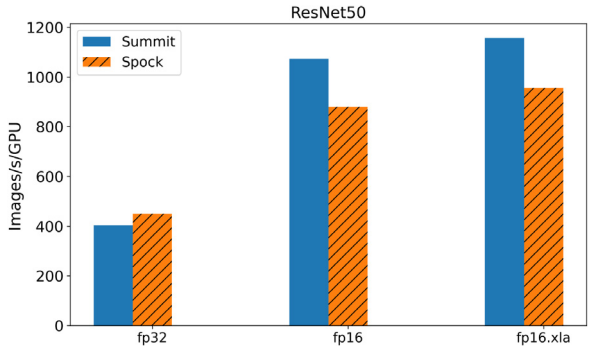

Fig. 12. The training images/s per GPU in FP32, FP16, and FP16 with XLA for `TF_CNN_Benchmarks`.

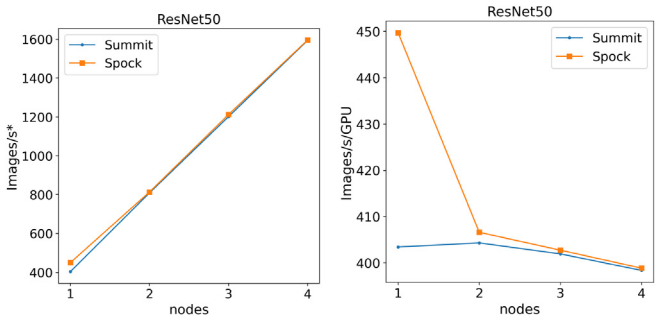

Fig. 13. The scaling of distributed training throughput for `TF_CNN_Benchmarks`. Images/s* is normalized to the number of GPUs per node for Summit (6) and Spock (4), respectively.

care, and by automating the mixed precision support it enables easier access to the full hardware capability. TensorFlow XLA can further accelerate the execution by generating optimized tensor operations for specific model rather than using the pre-built binary.

In Fig. 12, we plot the single device training performance for ResNet50 (batch size 128) with different accelerations. Consistent with Fig. 11, Spock has an edge at single precision, but lags behind in half precision. The speedup due to XLA though, are more or less the same.

Scaling Another important aspect of the framework is its scalability. Here we use a popular third-party distributed training library, i.e., Horovod, because it supports multiple frameworks including TensorFlow and PyTorch, and is highly optimized for HPC platforms. As shown in Fig. 13, the training images/s per GPU gradually decreases on Spock with a scaling efficiency $\sim 89\%$ up to four nodes, while Summit scales almost perfectly (scaling efficiency $\sim 99\%$). Given N/RCCL are used as communication backends, the results are consistent with Fig. 9.

Resource Utilization As described in Section 3, we use NVIDIA and ROCm tools to measure the resource utilization for every training step iteration between Summit and Spock. Fig. 14 shows the memory used and the GPU utilization for the ResNet50 benchmark (batch size 128) in single precision. The memory used for V100s seems to be constant across training steps, while for the MI100s it appears to vary across steps. This behavior more likely reflects the different sample frequencies as described in Section 3. The GPU utilization for Spock seems to be able to keep up more with each iteration compare to Summit, and that might reflect the fact that we get more number of images per second for single precision on Summit, as shown in Fig. 12.

4.4. Application benchmarks

Our goal at facilities is to enable leadership-scale scientific discoveries, hence the performance of scientific application is of ultimate interest. In CORAL-2, there are two sub-tasks enlisted from CANDLE

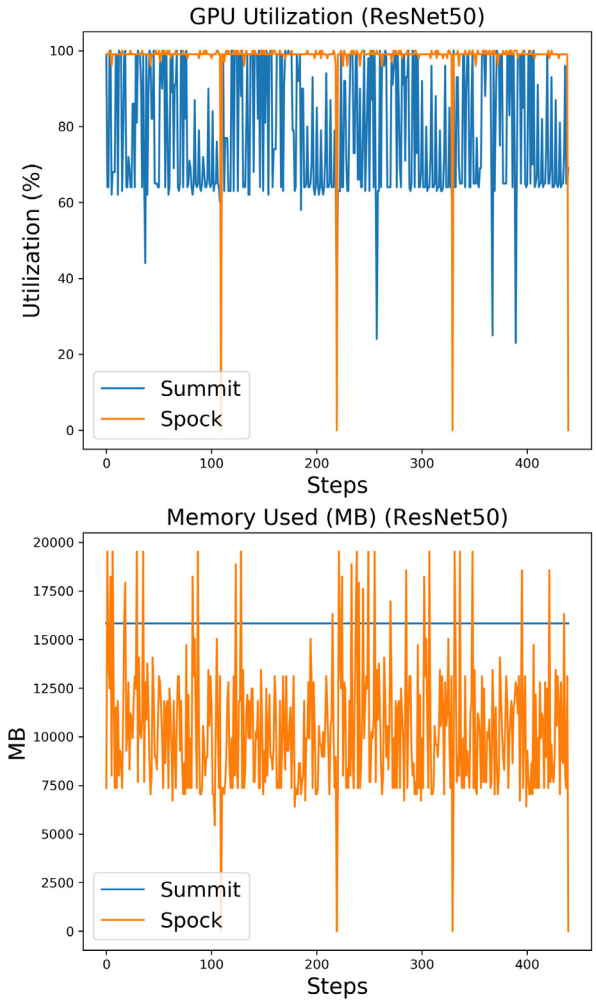

Fig. 14. Timeline plots for memory used and GPU utilization for `TF_CNN_Benchmarks`.

Table 3
Specification of 2 sub-tasks in CANDLE benchmark.

Task	Sample size	Model	Layer type	# layers	Hidden layer size	# weights
P1B1	4000	Autoencoder	Dense	6	2, 600, 1000	183M
P3B1	3000	Multi-task MLP	Dense	11	400, 1200	10M

benchmark (see Table 3): P1B1 is a regression task that use autoencoder to compress the gene expression; P3B1 is a classification task that use multi-task multilayer perceptron (MLP) for data extraction from clinic reports. Both models are based on fully connected dense layers, so the compute is dominated by GEMM kernel operations. The input data size is rather small (less or around 1 GB), and the impact of I/O is negligible (no noticeable performance differences in running with or without local storage).

In Fig. 15, the FOMs of time-to-solution are plotted for both tasks in CANDLE. Different random seeds are used to obtain the run time to the target reconstruction mean square error (P1B1) and classification accuracy (P3B1), but due to using the same baseline implementation and hyperparameter selection, it requires the same number of training steps to converge. Spock performs better in P3B1 task while Summit shows advantage in the other, mainly because of the performance differences in different shapes of matrices in GEMM (See Fig. 6).

Resource Utilization Fig. 16 shows the memory used and the GPU utilization for P1B1. The GPU utilization is higher for Spock,

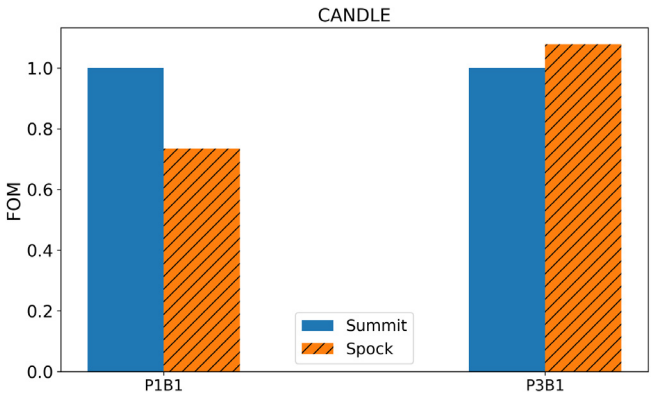

Fig. 15. The FOM of time-to-solution for 2 tasks in the CANDLE benchmarks.

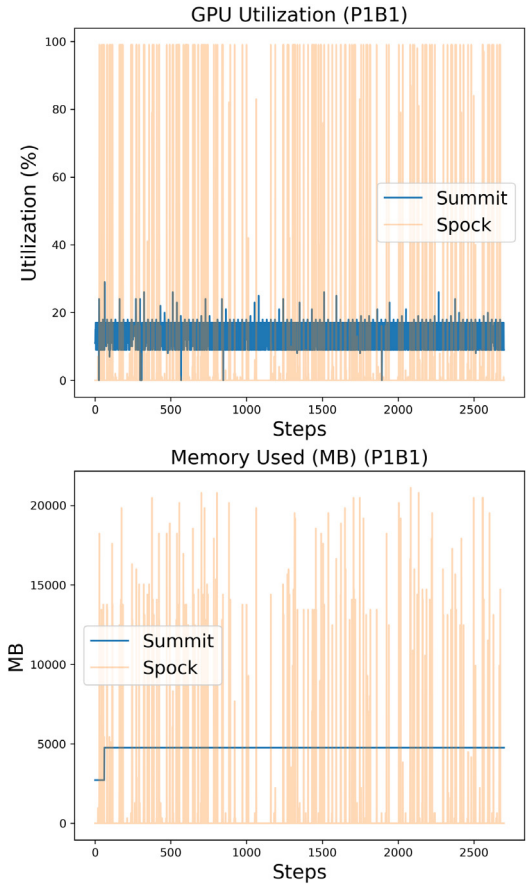

Fig. 16. Timeline plots for memory used and GPU utilization for `CANDLE-P1B1` benchmark.

compare to Summit and as expected for the V100s the `nvidia-smi` (as mentioned in Section 3) samples less frequently than `rocm-smi`. The memory used is higher for Spock compare to Summit. Compare to ResNet50 results, as shown in Fig. 14, the memory used is higher for Summit. We note that the model architecture is very different (dense vs convolution layers), and also that this benchmark uses a Keras implementation.

Fig. 17 shows the memory used and the GPU utilization for P3B1. This benchmark implementation is a mix of Keras and TF. It shows higher but more sparse GPU utilization for Spock, compare to Summit. Also the memory usage appears to be higher on Summit compare to Spock. If the `nvidia-smi` and `rocm-smi` are one-to-one comparable we could argue that larger models, or larger input size vectors can fit

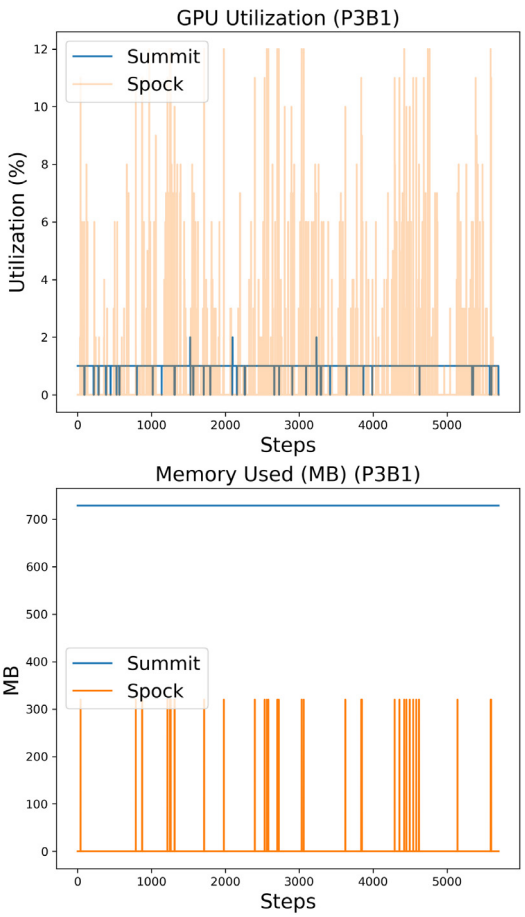

Fig. 17. Timeline plots for memory used and GPU utilization for `CANDLE-P3B1` benchmark.

on the MI100s with this implementation, but because of the differences in the tools, further investigation is needed.

5. Conclusion

We have presented a layered methodology and metrics to benchmark DL workloads at scale, involving kernels, models, frameworks, and applications. From the perspective of HPC facilities, we argue that understanding kernel and model level performance, and framework level scalability are more important than application FLOP counts given current scientific DL use cases and patterns. Using the CORAL-2 DL benchmarks, we evaluated the performance of Spock, an early-access testbed system for Frontier. Compared to the V100 based Summit system with CUDA DL stack, the MI100 based Spock with ROCm DL stack shows an edge in single precision performance for most kernel and model benchmarking tasks. However, there is currently a gap in its half precision performance, specifically for TensorFlow. Roofline modeling also indicates rooms for improvement in the ROCm stack, which is still maturing.

We also explored and demonstrated using machine learning an approach to model the relationship between input parameters and benchmark performance outcomes. And through a one-on-one comparison of the resource utilization for the two DL stacks on the same DL workloads, we are able to comment on the sources of performance differences. Although these two ways of gaining insight into performance comparisons are not conclusive in deducing underlying implementations or bottlenecks, our data does shed light on the direction for future optimizations in the DL stacks.

Table 4
The kernel input parameters in DeepBench [2]. RNN (vanilla) input parameters and first 20 of GEMM and Conv2D input parameters are listed.

Index	GEMM			Conv2D										RNN		
	m	n	k	w	h	c	N	k_f	f_w	f_h	pad_w	pad_h	$stride_w$	$stride_h$	H	N	s
0	1760	16	1760	700	161	1	4	32	20	5	0	0	2	2	1760	16	50
1	1760	32	1760	700	161	1	8	32	20	5	0	0	2	2	1760	32	50
2	1760	64	1760	700	161	1	16	32	20	5	0	0	2	2	1760	64	50
3	1760	128	1760	700	161	1	32	32	20	5	0	0	2	2	1760	128	50
4	1760	7000	1760	341	79	32	4	32	10	5	0	0	2	2	2048	16	50
5	2048	16	2048	341	79	32	8	32	10	5	0	0	2	2	2048	32	50
6	2048	32	2048	341	79	32	16	32	10	5	0	0	2	2	2048	64	50
7	2048	64	2048	341	79	32	32	32	10	5	0	0	2	2	2048	128	50
8	2048	128	2048	480	48	1	16	16	3	3	1	1	1	1	2560	16	50
9	2048	7000	2048	240	24	16	16	32	3	3	1	1	1	1	2560	32	50
10	2560	16	2560	120	12	32	16	64	3	3	1	1	1	1			
11	2560	32	2560	60	6	64	16	128	3	3	1	1	1	1			
12	2560	64	2560	108	108	3	8	64	3	3	1	1	2	2			
13	2560	128	2560	54	54	64	8	64	3	3	1	1	1	1			
14	2560	7000	2560	27	27	128	8	128	3	3	1	1	1	1			
15	4096	16	4096	14	14	128	8	256	3	3	1	1	1	1			
16	4096	32	4096	7	7	256	8	512	3	3	1	1	1	1			
17	4096	64	4096	224	224	3	8	64	3	3	1	1	1	1			
18	4096	128	4096	112	112	64	8	128	3	3	1	1	1	1			
19	4096	7000	4096	56	56	128	8	256	3	3	1	1	1	1			

Finally, we do note that Spock is a testbed early access system. Our benchmarking results and comparisons are most useful to concretely present our systematic approach to DL benchmarking. The kernels and frameworks are maturing and will continue to evolve (particularly for the ROCm ecosystem) and, therefore, specific observations reported in this paper are likely to change even if it does not affect the overall methodology that we have presented.

Acknowledgments

This research was sponsored by and used resources of the Oak Ridge Leadership Computing Facility (OLCF), which is a DOE Office of Science User Facility at the Oak Ridge National Laboratory supported by the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Appendix. Kernel parameters

In Table 4, we list the input parameters for GEMM, Conv2D, RNN (vanilla) kernels defined in DeepBench [2].

References

[1] ORNL, ANL, LLNL, CORAL-2 benchmarks, 2017, https://asc.llnl.gov/coral-2-benchmarks. (Accessed 30 July 2021).
[2] Baidu, DeepBench, 2016, https://github.com/baidu-research/DeepBench. (Accessed 30 July 2021).
[3] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, 2015, arXiv preprint arXiv:1512.03385.
[4] CANDLE, Cancer distributed learning environment, 2018, http://candle.cels.anl.gov. (Accessed 30 July 2021).
[5] P. Mattson, C. Cheng, G. Diamos, C. Coleman, 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, D. Kang, D. Kanter, N. Kumar, J. Liao, D. Narayanan, T. Oguntebi, G. Pekhimenko, L. Pentecost, V. Janapa Reddi, T. Robie, T. St John, C.-J. Wu, L. Xu, C. Young, M. Zaharia, Mlperf training benchmark, in: I. Dhillon, D. Papailiopoulos, V. Sze (Eds.), Proceedings of Machine Learning and Systems, Vol. 2, 2020, pp. 336–349, URL https://proceedings.mlsys.org/paper/2020/file/02522a2b2726fb0a03bb19f2d8d9524d-Paper.pdf.
[6] M. Abadi, et al., TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, URL https://www.tensorflow.org/. Software Available from: tensorflow.org.
[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 32, Curran Associates, Inc. 2019, pp. 8024–8035, URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.
[8] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable parallel programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for? Queue 6 (2) (2008) 40–53, http://dx.doi.org/10.1145/1365490.1365500.
[9] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, E. Shelhamer, Cudnn: Efficient primitives for deep learning, CoRR (2014) arXiv:1410.0759.
[10] Nvidia, Nvidia communication collectives library (NCCL), 2019, https://github.com/NVIDIA/nccl. (Accessed 30 July 2021).
[11] AMD, ROCM, 2016, https://rocmdocs.amd.com. (Accessed 30 July 2021).
[12] J. Khan, P. Fultz, A. Tamazov, D. Lowell, C. Liu, M. Melesse, M. Nandhimandalam, K. Nasyrov, I. Perminov, T. Shah, V. Filippov, J. Zhang, J. Zhou, B. Natarajan, M. Daga, Miopen: An open source library for deep learning primitives, 2019, arXiv arXiv:1910.00078.
[13] AMD, ROCM communication collectives library (RCCL), 2020, https://github.com/ROCmSoftwarePlatform/rccl. (Accessed 30 July 2021).
[14] S.S. Vazhkudai, B.R. de Supinski, A.S. Bland, A. Geist, J. Sexton, J. Kahle, C.J. Zimmer, S. Atchley, S. Oral, D.E. Maxwell, V.G.V. Larrea, A. Bertsch, R. Goldstone, W. Joubert, C. Chambreau, D. Appelhans, R. Blackmore, B. Casses, G. Chochia, G. Davison, M.A. Ezell, T. Gooding, E. Gonsiorowski, L. Grinberg, B. Hanson, B. Hartner, I. Karlin, M.L. Leininger, D. Leverman, C. Marroquin, A. Moody, M. Ohmacht, R. Pankajakshan, F. Pizzano, J.H. Rogers, B. Rosenburg, D. Schmidt, M. Shankar, F. Wang, P. Watson, B. Walkup, L.D. Weems, J. Yin, The design, deployment, and evaluation of the CORAL pre-exascale systems, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, in: SC '18, IEEE Press, 2018, http://dx.doi.org/10.1109/SC.2018.00055.
[15] T. Chen, C. Guestrin, XGBoost: A Scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, in: KDD '16, ACM, New York, NY, USA, 2016, pp. 785–794, http://dx.doi.org/10.1145/2939672.2939785.
[16] J. Yin, S. Gahlot, N. Laanait, K. Maheshwari, J. Morrison, S. Dash, M. Shankar, Strategies to deploy and scale deep learning on the summit supercomputer, in: 2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers, DLS, 2019, pp. 84–94, http://dx.doi.org/10.1109/DLS49591.2019.00016.
[17] ICL, HPL-AI, 2019, https://icl.bitbucket.io/hpl-ai/. (Accessed 30 July 2021).
[18] S. Plimpton, Fast parallel algorithms for short-range molecular dynamics, J. Comput. Phys. 117 (1) (1995) 1–19, http://dx.doi.org/10.1006/jcph.1995.1039, URL https://www.sciencedirect.com/science/article/pii/S002199918571039X.
[19] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems, Vol. 1, in: NIPS'12, Curran Associates Inc. Red Hook, NY, USA, 2012, pp. 1097–1105.
[20] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2015, pp. 1–9, http://dx.doi.org/10.1109/CVPR.2015.7298594.
[21] TensorFlow, Tf_cnn_benchmarks: High performance benchmarks, 2019, https://github.com/tensorflow/benchmarks/tree/master/scripts/tf_cnn_benchmarks. (Accessed 30 July 2021).
[22] OLCF, Spock, 2021, https://docs.olcf.ornl.gov/systems/spock_quick_start_guide.html. (Accessed 30 July 2021).

[23] A. Lavin, S. Gray, Fast algorithms for convolutional neural networks, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, IEEE Computer Society, Los Alamitos, CA, USA, 2016, pp. 4013–4021, http://dx.doi.org/10.1109/CVPR.2016.435.

[24] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun, Overfeat: Integrated recognition, localization and detection using convolutional networks, 2013, URL arxiv:1312.6229.

[25] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, CoRR (2014) arXiv:1409.1556.