Towards the FAIR Asset Tracking across Models, Datasets, and Performance Evaluation Scenarios*

Piotr Luszczek
EECS Dept., University of Tennessee
Knoxville, TN, USA

Tokey Tahmid
EECS Dept., University of Tennessee
Knoxville, TN, USA

***Abstract*—In order to ensure the reproducibility and give full account of the results obtained from scientific simulations assisted by ML/AI models, a new set of methodological innovations have to take place, that account for provenance, deployment, usage, updates, and archiving of variety of digital assets. We present a design and implementation of a methodology, that not only addresses these very aspects of modern computational science but is also a major step towards practically achieving this lofty goal for a variety of established models and their datasets, that are of particular importance to the progress of science simulations utilizing ML/AI models. We also show experimental results of applying our approach to a specific evaluation scenario and show how it maintains the performance efficiency, delivers accurate training results, and captures sufficiently rich context of the runtime behavior to inform both the particular domain science and machine learning communities.**

I. Introduction

The often cited manifesto of findable, accessible, interoperable, and reusable (FAIR) data management and stewardship [1] lays out a lofty vision to across many other scientific disciplines. In order to make this aspirational goal a reality in the specific context of computational simulations' enriched with ML/AI models, an entire infrastructure has to be reimagined, built, and maintained in order to uphold these principled tenants. Only when the support and assistance are provided from the middleware and frameworks, we can hope for meaningful contributions enriching this nascent incarnation of the modern computational science. This is even more important during the transition of the recent years that morphed the computational components based purely on *ab initio* parametric models to also include parameter-free data models that rely on large volumes of data, that are being fed during distributed training into the ever more complex implementations of contemporary ML and AI methods. The major thrust towards this augmentation takes form of surrogate models and their underlying Deep Neural Networks (DDN), Large Language Models (LLM),[1] Deep Reinforcement Learning (DRL), and even large scale foundation models. Section IV has more detail exposition on these advances that feature *surrogate models* for science.

To that end, our Surrogate AI Benchmarking Applications' Testing Harness (SABATH) project [3] aims to assume the role of intermediary between the science community in need of data set management for deployment of models and the ML/AI community designing them and overseeing the process

*This work was supported by the U.S. Department of Energy, Office of Science, ASCR under Award Number DE-SC0021419.

[1]They are also called Neural Language Models (NLM) [2].

SABATH (default Python 3)

Environments	Frameworks	Models	System S/W	Hardware	Communication
Python 3	TensorFlow	CloudMask	CUDA	CPU	Infiniband
C/C++, Fortran	PyTorch	CosmoFlow	ROCm	GPU	NVLink
SLURM	Keras	MiniWeatherML	SYCL	IPU	GigE
Containers			MPI		
			NCCL		
Package Repos, Hubs		**Datasets**	**Storage**	**Disk Datasets**	
Anaconda, Docker Hub, PyPI		CosmoUniverse	HDF5, Git LFS	HDD, SSD, NVMe	
Docker Hub		SuperCell	PNFS, Lustre, Ceph		

Fig. 1. Overall design of SABATH and how it interacts with external ecosystem of environments, frameworks, models, data sets, software repositories, and accelerator-based hardware.

of their training and inference. To achieve these goal of seamless provenance, management, and archiving of updates, SABATH interacts with the rich external ecosystem of environments, frameworks, models, data sets, software repositories, and accelerator-based hardware. Figure 1 shows a concise overview of the major components as well as example collections and technological artifacts comprising the design of and interacting with the implementation of our project. The figure is meant to briefly give the reader a good initial view of the complexity of managing such large collections of disparate digital assets while the details are in the sections below.

II. Related Work

Our proposed implementation brings together the set of disparate digital assets in a somewhat related way to how the commonly used software package managers operate. The similarity extends to how they handle the dependence structure, download process, and deployment configurations. However, for the three-way matching of domain science with ML/AI models and with the compatible data sets, the traditional software package management is insufficient. Primarily due to the single directory prefix, that houses a unique sets of sources, executables binaries, data blobs (e.g., firmware), and other auxiliary files (locale descriptions, documentation, etc.). The uniqueness property precludes custom implementations that may be necessary to satisfy functional, performance, or accuracy requirements. Those often show up as installation conflicts. We address this lack of implementation and/or deployment variants by managing multi-prefix installations with many ways of customizing the resulting environments.

On the other hand, the `rpath`-assisted installs based on Merkle trees [4], [5], that track the identity of the installed packages (often custom-built directly from source code) enabling much greater flexibility of maintaining a plethora of variants to satisfy sophisticated requirement structures. However, for practical usability they need to be augment with

additional tracking methods (such as multi-package collections), which serve as a necessary augmentations that overlay additional grouping information, that aid maintenance of coherent instances of dependence graphs. This exposes the users, package maintainers, and developers alike this multi-layered complexity and multiple concurrent groupings of packages must coexist in the package meta-data definitions often collected inside a single package definition file. Such compound definitions lead to conditional syntax structure thus increasing the conceptual burden required for meaningful comprehension and effective debugging across the deployment platforms.

III. Organization of Digital Assets

In order to bring together all the disparate components presented in Figure 1, their incoming data whichever form they are, have to be cataloged and cached locally as is necessary for effective use in training or inference workflows. Rather than centralized model of single source provenance, that is common to most package managers, we use a *federated* model for sourcing the assets together into a coherent view with local copies cached for performance. In broad terms, we recognize the following main classes of assets needed for tracking (they sorted by their approximate size with the caveat of the sorting order not holding uniformly for all the use cases):

- data sets (with or without labels),
- ML/AI models (weights, biases, hyper-parameters, and structure/connectivity graphs),
- non-executable binaries such as object files,
- executable binaries, and
- source code files.

By far, the first two items: data sets and models; are the largest in physical size and present the greatest potential for a number of issues during initial download, storage (either reads, writes, or updates) during operation, and long term archiving. For this reason a substantial amount of attention is devoted to preventing these issues and providing smooth access despite the unusual stress they present to the system resources.

IV. Surrogate Modeling in AI-Assisted Science

The transformational effects of advances in ML/AI imposed on the traditional simulation applications cannot be overstated [6], [7], [8]. There is a range specific modes of operation of the classic simulation code implementing a scientific model, called either *ab initio* or parametric, and the interaction with a data-driven model derived from either ML on AI methods. One modality is a *model substitution*: either some or the entirety of the scientific simulation is replaced with a ML/AI counterpart. The *assistive modality* pairs up both ab initio parametric model with ML/AI assistant that manages the data flow, finds patterns, and marks new domains of interest. In this mode, the regions needing extra scrutiny receive receive refinement and/or higher resolution thus running the original simulation for a new data set or initial conditions. Irregardless of the operational modality, there are high gains to be made from such new scientific campaigns ranging remarkable performance speedups and reduction in time or energy consumption. [9], [10], [11], [12], [13]

V. Data Transfer Considerations

As described in Section III, the physical size of two major asset groups, model and training data, pose unique challenges to not just users by the system administrators of either SysOps or MLops variety. Any asset management platform, our own SABATH project's fetcher notwithstanding, must address this starting with efficient transfer from the source to management of the large data volumes on site. The latter is covered in Section VI.

The main method of tackling the prolonged downloads of assets is to introduce parallelism across transfer layers. To that end, we utilized bot intra-protocol and inter-protocol methods that are detailed below. However, from the onset we want to stress that even with full utilization of the techniques outlined below, we cannot exceed the combined physical bandwidth limit of the underlying interconnect.

The disparate assets brought together by SABATH for surrogate model training, inference, or benchmarking are unlikely to come from nearby geographical locations and this has repercussions on the latency profile that our fetcher has to contend with. The long distance disparity experience the single packet delays on the order of tens of milliseconds inside a single country. But the cross-continent long haul links often break the 100 millisecond barrier. This is in contrast to sub-millisecond latency available in modern high-bandwidth networks such as Inifiniband XDR and 100 GB Ethernet and its variants such as Cray/HPE Slingshot, which all must use high-frequency signaling to fulfill the bandwidth requirements. Thus, achieving high level of cross-packet parallelism inside a single TCP stream on these long long delay data paths is critical to maximize the cumulative bandwidth allocated to the SABATH fetcher based on network partitioning policies and Qualit of Service (QoS) guarantees.

The protocol-level solution is to manipulate the kernel socket parameters to artificially increase TCP window size for high-latency routes beyond its default value that is mostly geared towards small scale LAN and WAN setups. However, this may be a problematic solution in practice. For one, the low-level interface may not be enabled for user-level codes, and, for another, a single optimal value may not exist for long lasting transfers with changing network load along the way. For such situations, a more adaptive scheme is appropriate and this is the solution we take by switching to existing network transfer tools. In particular, either Streaming TCP (STCP) or Globus end points offer us an efficient and functional alternative to adaptive shape the protocol response to a changing network load as well as maximizing the number parallel streams encapsulating the data packets while assembling them properly at the receiving site to accommodate varying network speeds experienced by individual packets. It is worthwhile to point out the versatility of incorporating Globus that has many additional features and could accommodate multi-source fetches transparently.

Finally, we also allow incorporating external commands as SABATH fetchers whereby we export the protocol details, data flow management and potential authentication to a specific tool efficiently targeting external storage repositories. Currently, we expose Command Line Interface (CLI) to aws-cli for accessing Amazon Web Services (AWS) S3 buckets but we also make available schemes for scp, rsync, curl, wget.

VI. Storage, Asset Caching, and File System Maintenance Tasks

To underscore the importance of comprehensive management of storage devices in the context of large data collections, we first start with a simple use case of managing allocations and available storage pools and linking those with the incoming assets made available through any of the available SABATH fetchers described in Section V. Of particular importance for long lasting data transfers is anticipating the total size of the incoming data volume and pro-actively managing the fetch operation only if sufficient space is available on the destination device or the file system mount point.

There are however a number of scenarios that have to be handled outside of direct control of SABATH because they need more comprehensive treatment across hardware and/or software layers often beyond purview of a strictly user-level application. One such service is creating and maintaining reliable device-compounding overlays that appear as a single mount point volume. This functionality is much better served with Linux' Logical Volume Manager (LVM) because its API surface spans the traditional UNIX service calls and is easily outside our own control. Indeed, it makes unreasonable for us to expect projects external to SABATH to implement their storage access through our API and therefore we remain agnostic towards the underlying device volumes' managers in either kernel or user space. This addresses the potential issues with allocation of sufficient space for large models such as those based on DDNs or LLMs or the data sets they need for training even when applying modern model scaling principles [2]. An even more reliable way satisfying the bandwidth-demanding model training or inference tasks is to coalesce storage devices while maintaining the API surface of a modern Linux kernel. The most common example that we rely on the use of the appropriate level of RAID that has the potential to make regular HDDs to appear as fast as SDDs or NVMe devices. Another aspect of disk volume management is maintaining sufficient bandwidth to and from the storage media to accommodate both network transfers, training, and inference data traffic. This is also relegated to the kernel space and thus lower level layers of the middleware and operating system software stack. In a similar vain, we do not yet include support for versioning and backups, which are now well served by system-level services such as file system snapshots (available, for example, in ReiserFS and ZFS) and its semantically weaker relative: journal-based file system transactions. Finally, the versioning service of large digital collection of assets is now relegated to the tools akin to Git LFS that remain very efficient at handling this type of tasks. However, there is still an important task left to SABATH, a user-level tool, that has much better view of the assets than a low-level service ever would. This is the case for avoiding both downloading and storage of multiple copies of the same item. Some storage systems offer deduplication functionality but it is mostly maintained are relatively low-level such as file system blocks. While partially effective, it does not address the problem of wasted bandwidth, which occurs before the entire asset is stored locally on the device to be identified as an extraneous replica. But SABATH already identifies potential duplication by hashing the data sources without the user assistance of identifying the potential for unnecessary download. Even beyond, source-based identification, we also use content-based hashing so the repeated data can be recognized even if its available from multiple different sources. Additionally, multi-source assets gives us the potential for parallel fetching already described in detail in Section V.

VII. Compute and Communication Efficiency

The modern computing hardware that we consider the main target to use for our training and inference tasks spans CPUs, GPUs, and FPGAs as well as specialized hardware designed in recent years for a mix of HPC and ML/AI workloads such as GraphCore's dataflow processing IPUs or Cerebras' Wafer Scale Engine chips. There is clearly a need to address the complexities associated with programming such a wide range of platforms that differ substantially in the underlying architecture, suitable programming model, and vendor-supplied software stack for implementing and deploying scientific codes. To that end, high-level frameworks abstract away a lot of the underlying difficulties in matching the mathematical models with the software stack that targets these various hardware pieces as we show in Figure 1.

Along the same lines, the efficient utilization of the communication interconnect hardware requires use the vendor-specific API for the particular network, be it NVIDIA's Mellanox Infiniband or node-level NVLink with an exception of generic GigE infrastructure but not its HPC variants such as Cray/HPE Slingshot. This is overcome in a similar manner as is the case for the variety of compute hardware: it starts by generalizing the interface to be widely applicable even at low-level of moving rudimentary data across the compute cluster while retaining the efficiency by keeping the generic functionality close to the capabilities of the Network Interface Card (NIC) instances and the switches that connect them. An example of such API definitions are Message Passing Interface (MPI), NVIDIA Collective Communication Library (NCCL), and OpenShmem. Still, while these interface definitions are both expressive and efficient, their abstraction is too low to succinctly express the common patterns in distributed training and inference scenarios, that need higher-level means of performing the basic collective reductions commonly occurring in gradient applications on the models' weight parameters. An example of such a highly abstract and yet expressive framework is Horovod [14]. These abstraction layers are illustrated in Figure 1.

In summary, we support SABATH the entire software stack that allows efficient hardware utilization, be it the compute units or the interconnect. This comes at the cost of wide ranging and complex dependence structure for the required libraries, tools, and system extensions. A comprehensive way of handling this is detailed in Section VII and here we only note the ease of deployment that is the aim of our implementation.

VIII. Use Case Studies

In Sections III, V, VI, and VII, we presented in general terms both the motivation and overarching design decisions of our platform for FAIR asset management. In what follows, we present how these generic guidelines apply to a specific context of a surrogate model training in a strictly science-driven scenario.

A. CosmoFlow Surrogate Model

CosmoFlow [15] is pre-trained surrogate model specialized for analysis of large, 3D cosmological datasets. By design, the model structure lends itself well to scalable execution on High Performance Computing (HPC) and supercomputing clusters while using the TensorFlow framework for expressing and updating its DDN structure and weights. Unlike its proto-designs, in its current form, the model's structure was redesigned, which enabled the new design to accept suitably larger problem sizes. In fact, the most recent input sets measure up to 128 voxels. The output directly estimates three relevant cosmological parameters.

The initial performance tuning of the model was performed on CPUs only but it relied on vendor-optimized DDN kernels available in the relevant component of Intel's Math Kernel Library (MKL). More specifically, the underlying model is comprised of 3D Convolutional Neural Networks (CNN). Additionally, the scaling is enabled through the use of MPI and additional efficiency was afforded by tight integration with the Cray Programming Environment and its Machine Learning plugin, which enable very efficient utilization of the Cray hardware platform.

The ultimate test of CosmoFlow came in as the parallel efficiency was delivered during large scale tests on Cori supercomputer at the NERSC supercomputing facility. In particular, the global performance level of 3.5 Pflop/s in single precision arithmetic was demonstrated in the capability mode of execution while confirming to the data parallel training paradigm. The physical hardware comprised 8192 Cori Intel® Xeon Phi™ processors (KNL). It is also worth noting, that even at these unprecedented (at the time) node counts, the model's convergence achieved, and, equally important, its ability to make accurate predictions was also ascertained.

In terms of domain science, this enhanced version of the model enabled detailed analysis of dark matter distributions in three dimensions.

B. CosmoUniverse Dataset

The CosmoUniverse dataset, weighing in tens of GBs in size, was used with the CosmoFlow model for training. The origin of the dataset traces back to state-of-the-art simulations of cosmological scale phenomena. The most relevant three parameters describing the dark matter phenomena and macro scale were varied for best training efficiency.

Nearly $10,000$ cosmological simulations were performed and they involved modeling the n-body interactions focusing on dark matter. As a preprocessing step, the output data from these simulations were organized into a 3D histogram, that showcased a large number of particles' interactions. The simulations were performed inside $512 \times 512 \times 512$ cubes, and were sampled at four distinct red-shift stages.

It is recommended by the authors to utilize their Globus end point for downloading the entire data set due to its size and storage. By comparison, use the wget fetcher is significantly slower due to the combination of reasons detailed in Section V. The primary data layout inside the dataset is fully managed by the HDF5 format and its container structure fully capable of handling both the size and dimensionality of the data without any loss of information. In particular, each universe configuration and simulation result corresponds directly to an HDF5 file of size 1 GB. Furthermore, each file contains a large number of descriptive attributes, that may be readily interpreted by the domain scientists. The examples include 'dataset_tag', 'universe_tag', 'seed9', 'namePar', 'physPar', 'unitPar', 'redshifts', and 'full'. Each of these has a unique role in the simulation and functions as feedback for training the models. Further details are beyond the scope of this writing.

More specifically, the statistics of the dataset can be characterized by uniform variation around a mean value for each of the cosmological parameters, with a spread value of 30%. These particular parameters have their labels stored inside the dataset as both the normalized unit values contained fully inside the $(-1, 1)$ range and also as their counterparts taken directly from the physics of the problem. This allows for easy cross-correlation between the ab initio physics model and data-driven DDN model. Additionally, they allow for reflecting these numbers directly to the values injected into the original simulations. It was observed that this type of normalization is particularly beneficial for training machine learning models targeting scientific applications.

Also contained in the dataset are several specific parameters that vary across each simulated universe instance and they capture four distinct red-shift snapshots. More specifically, the cosmological parameter Ω_m fluctuates around 0.30 point with a deviation value of $\pm 30\%$. On the other hand, the physical simulation parameter Ω_L is computed as $1 - \Omega_m$, which corresponds to the flat Universe hypothesis. Yet another parameter, σ_8, oscillates around value 0.80, also having with a 30% variation. The N_{spec} parameter remains at 0.96 with a 30% fluctuation. Ω_b is a fixed parameter at 0.045. The parameter H_0 varies around a mean of 70 with a 30% deviation. The individual box size was calculated using the formula $512 \times H_0/70$. Finally, the dataset also includes four red-shift stages, which are 0, $\frac{1}{2}$, $\frac{3}{2}$, and 3.

IX. Environments for Building and Launching

The SABATH project uses plain Python 3 for launching subprocess shells thus not imposing undue requirements on the default setup of the user. However, the supported assets include the ML/AI models have their own programming language and dependencies required for successful launch. The languages that are currently supported include C/C++ and Fortran. In turn, the specific dependencies of the ML/AI models are currently managed using two external package managers: Anaconda and PyPI. These choices ensure broad applicability across surrogate scientific models and allows our project to maintain an important balance between its performance, versatility, and ease of development, thus carefully catering to various needs of the majority of ML/AI models.

A. Examples of Handling Environment Dependencies

1) sciml-bench (CloudMask) and CosmoFlow Models: For the SciML-bench [16] and CosmoFlow models, the environment dependencies are provided by both the Anaconda and PyPIP main repositories. The dependencies satisfied by Anaconda repo include several packages, including but not limited to bokeh, cuDNN, mpi4py, NCCL, PyTorch, and TensorFlow-GPU. Most notably, the nvcc-linux-64 package configures the environment to be "CUDA-aware," a critical feature for ML/AI models that use GPU computation and need to communicate directly between the GPUs using CUDA pointers. The Horovod and NCCL dependencies ensure efficient distributed deep learning, while the Gloo library ensures highly optimized collective communications in PyTorch. The requirements satisfied by PyPIP repo supplement the Anaconda dependencies with additional packages such as matplotlib, scikit-learn, pandas, and wandb. Among these, Horovod-specific requirement is to build it directly from the source code, which ensures the compatibility of horovod with the other dependencies such as tensorflow, PyTorch, MPI, Gloo, MxNET.

2) MiniWeatherML: MiniWeatherML model has a distinct set of dependencies that are managed by Anaconda, including hdf5, curl, zlib, and netcdf4, with netcdf4 particularly chosen for compatibility with GNU GCC version 9.5.0. This model also requires specific modules such as gcc, CUDA, CMake, and Open MPI. The Open MPI requires specific compilation flags –with cuda and –with gdrCopy, thus enhancing the performance of MPI when performing communication directly involving GPU memory. MiniWeatherML also depends on several more specific modules, including YAKL (Yet Another Kernel Launcher), PONNI, yaml-cpp, and PnetCDF. They provide additional functionality for kernel launching, neural network inferencing, and parallel I/O operations.

B. Special Handling of Dependencies

There are specialized dependencies are only required by few ML models managed and supported by SABATH. Due to their complexity, they are discussed separately in more detail here. Also, one of the main dependencies across many models is Horovod which is discussed first.

1) Horovod – a Distributed Deep Learning Training Framework: Horovod package is required by both SciML-bench (and CloudMask speifically) and CosmoFlow. Horovod is a distributed deep learning framework for training across large compute cluster with GPU accelerators. The package supports multiple machine learning platforms such as TensorFlow, Keras, PyTorch, and Apache MxNet. The main objective of Horovod developers was to simplify the process of scaling a single-GPU training across multiple GPUs in parallel, thus delivering crucial optimization to the distributed deep learning process.

Inside SABATH-provisioned environments, Horovod is built with both Conda and PyPI resolving the dependencies. The specific installation requirements can be found on their respective websites. Here, we give only a brief overview of the installation process and the corresponding requirements for building Horovod with the GPU support through the combination of downloads from either Anaconda or PyPI repositories.

a) Installation requirements: Horovod requires: GNU Linux or macOS, Python 3.6 and above, GNU g++ version 5 or above (or an alternate compiler supporting the C++14 standard), CMake 3.13 or newer, TensorFlow 1.15.0 or above, PyTorch 1.5.0 and above, or MxNet 1.4.1 and above. Optionally MPI can also be used. For best performance on multiple GPUs, NCCL 2 is recommended. When installing Horovod with TensorFlow 2.10 or later, a compiler that supports the C++17 such as GNU g++ version 8 or newer is needed.

b) Building Horovod with Conda and PyPIP:: building Horovod in a Conda environment with GPU support involves several steps, including the installation of the NVIDIA's CUDA Toolkit manually due to the lack of NVCC in the Conda cudatoolkit package. The Conda environment.yml file that is used to specify as many dependencies as possible in a single command, while the requirements.txt file lists all dependencies that need to be satisfied by PyPI, including Horovod itself. Additional packages for enabling GPU and CPU resource monitoring are enable by Google's Tensorboard: these include jupyterlab-nvdashboard and jupyter-tensorboard, and they can also be added to the requirements.txt file. After defining all these necessary dependencies (they are associated with the SABATH meta-data in the appropriate JSON files), the Conda environment can be created using a trivial set of build commands, thus creating the environment so it can be setup using the conda activate.

When building the Conda environment, the SABATH-provisioned build script sets the relevant Horovod's build variables, and creates the Conda environment, and subsequently activates the environment in order to initiate the build of JupyterLab with any additional extensions. To verify the Conda environment, the horovodrun –check-build command can be used to ensure the state of Horovod's build and its correctness with support for the required deep learning frameworks and their controllers.

2) Open MPI with support for CUDA and GPU Direct: This is an important use case for SABATH-managed deploy-

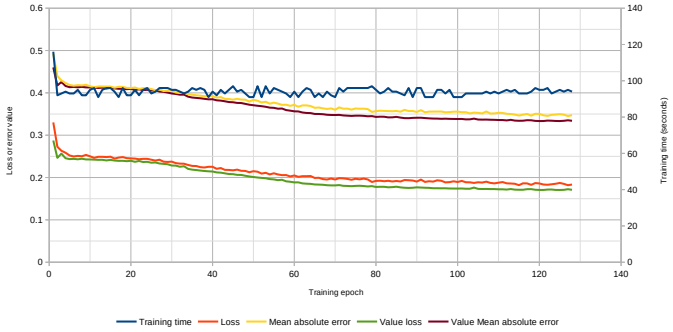

Fig. 2. CosmoFlow results on 8 NVIDIA A100 GPUs.

ments. Namely, the ability to pass CUDA pointers as MPI buffers combined with direct transfers between GPU memory spaces using GDRCopy inside UCX. This scenario is particularly beneficial for the MiniWeatherML model that uniquely requires a specialized build of CUDA-aware MPI. Furthermore, incorporating GDRCopy inside the build of the UCX layer may see drastic communication bandwidth improvements especially in configuration that connect GPUs directly to NICs rather than shuttling the data through interactions with the CPUs. Such build configurations are also supported in order to satisfy the particular model's requirements meta-data.

To test this deployment scenario, we used Open MPI version tagged 4.1.6alpha1, which is based on the development branch of the project rather than any of the recently publicly released versions. The Open MPI was configured and built on an NVIDIA DGX V100 node. For testing purposes, CUDA 11.8.0 toolkit was used for the build. The GDRCopy deployed was using source code of the matching NVIDIA's kernel driver for CUDA. Instead of libfrabric, the build has been configured to use UCX as the communication layer by simply supply –with-ucx flag during the configuration stage. It also included support for the CMA kernel feature, IPv6, and included only C and Fortran bindings. The build used GNU GCC 9.5.0.

Furthermore, the development snapshot of Open MPI had official standard API compatible with version 3.1.0, and the support for multi-thread mode was enabled (MPI THREAD MULTIPLE mode). Some features were left off: no support heterogeneous systems, no MPI 1 compatibility, and no support for neither fault tolerance nor check-pointing.

Based on our performance tests, such custom builds of Open MPI, and in particular those that combine CUDA-aware functionality with GDRCopy inside UCX are critical for optimal performance on multi-GPU systems with fast on-node interconnect such as NVLink.

X. Results from a Sample Experiment

The experiments were performed on Rocky Linux release 9.2 (Blue Onyx) with kernel 5.14.0-284.18.1.el9_2. The CPU was configured in two socket each featuring 64-core AMD EPYC 7742. The system also featured 8 GPUs from NVIDIA, model Ampere A100 SXM4 with 80 GB memory and NVLink system interconnect for the accelerators.

Figure 2 shows results from a run using the CosmoFlow model and one of its smaller data sets. The essential metrics from the training experiment are recorded collectively are charted simultanously with double y-axis to reflected different scales from loss and error values on the left axis and time-per-epoch on the right axis.

References

[1] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg *et al.*, "The FAIR guiding principles for scientific data management and stewardship," *Scientific Data*, vol. 3, no. 160018, Mar. 2016. [Online]. Available: https://doi.org/10.1038/sdata.2016.18

[2] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv*, vol. 2001, no. 08361, Jan. 2020.

[3] P. Luszczek and C. Brown, "Surrogate ML/AI model benchmarking for FAIR principles' conformance," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2022, pp. 1–5.

[4] R. C. Merkle, "Method of providing digital signatures," 1982, US patent 4309569 published Jan 5, 1982, assigned to The Board of Trustees of the Leland Stanford Junior University.

[5] ——, "A digital signature based on a conventional encryption function," in *Advances in Cryptology – CRYPTO '87. Lecture Notes in Computer Science*, vol. 293, 1988, pp. 369–378.

[6] G. Fox and S. Jha, "Understanding ML driven HPC: Applications and infrastructure," in *IEEE eScience 2019 Conference*, San Diego, California, 2019, https://escience2019.sdsc.edu/.

[7] ——, "Learning everywhere: A taxonomy for the integration of machine learning and simulations," in *IEEE eScience 2019 Conference*, San Diego, California, 2019, https://arxiv.org/abs/1909.13340.

[8] G. Fox, J. A. Glazier, J. Kadupitiya, V. Jadhao, M. Kim, J. Qiu, J. P. Sluka, E. Somogyi, M. Marathe, A. Adiga, J. Chen, O. Beckstein, and S. Jha, "Learning everywhere: Pervasive machine learning for effective high-performance computation," in *IPDPSW Workshops: HPDC Workshop at IPDPS*, Rio de Janeiro, Brazil, 2019, https://arxiv.org/abs/1902.10810.

[9] M. F. Kasim, D. Watson-Parris, L. Deaconu, S. Oliver, P. Hatfield, D. H. Froula, G. Gregori, M. Jarvis, S. Khatiwala, J. Korenaga, J. Topp-Mugglestone, E. Viezzer, and S. M. Vinko, "Up to two billion times acceleration of scientific simulations with deep neural architecture search," *arXiv [stat.ML]*, 2020, http://arxiv.org/abs/2001.08055.

[10] J. C. S. Kadupitiya, G. C. Fox, and V. Jadhao, "Machine learning for performance enhancement of molecular dynamics simulations," in *International Conference on Computational Science ICCS2019*, Faro, Algarve, Portugal, 2019, http://dsc.soic.indiana.edu/publications/ICCS8.pdf.

[11] A. Moradzadeh and N. R. Aluru, "Molecular dynamics properties without the full trajectory: A denoising autoencoder network for properties of simple liquids," *J. Phys. Chem. Lett.*, vol. 10, no. 24, pp. 7568–7576, Dec. 2019, available: http://dx.doi.org/10.1021/acs.jpclett.9b02820.

[12] Y. Sun, R. F. DeJaco, and J. I. Siepmann, "Deep neural network learning of complex binary sorption equilibria from molecular simulation data," *Chem. Sci.*, vol. 10, no. 16, pp. 4377–4388, Apr. 2019, http://dx.doi.org/10.1039/c8sc05340e.

[13] F. Häse, I. F. Galván, A. Aspuru-Guzik, R. Lindh, and M. Vacher, "How machine learning can assist the interpretation of ab initio molecular dynamics simulations and conceptual understanding of chemistry," *Chem. Sci.*, vol. 10, no. 8, pp. 2298–2307, Feb. 2019, http://dx.doi.org/10.1039/c8sc04516j.

[14] A. Sergeev and M. D. Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," *arXiv*, vol. 1802.05799, 2018.

[15] A. Mathuriya, D. Bard, P. Mendygral, L. Meadows, J. Arnemann, L. Shao, S. He, T. Karna, D. Moise, S. J. Pennycook, K. Maschoff, J. Sewall, N. Kumar, S. Ho, M. Ringenburg, Prabhat, and V. Lee, "CosmoFlow: using deep learning to learn the universe at scale," *arXiv*, vol. 1808, no. 04728, Aug. 2018, revised 9 Nov 2018.

[16] J. Thiyagalingam, G. von Laszewski, J. Yin, M. Emani, J. Papay, G. Barrett, P. Luszczek, A. Tsaris, C. Kirkpatrick, F. Wang, T. Gibbs, V. Vishwanath, M. Shankar, G. Fox, and T. Hey, "AI benchmarking for science: Efforts from the MLCommons science working group," in *Proceedings ISC 2022 Workshops, Lecture Notes in Computer Science*, vol. 13387, 2022.