Integrating Deep Learning in Domain Sciences at Exascale

Rick Archibald[1], Edmond Chow[2], Eduardo D'Azevedo[1], Jack Dongarra[1,3], Markus Eisenbach[1], Rocco Febbo[3], Florent Lopez[3], Daniel Nichols[3], Stanimire Tomov[3](✉), Kwai Wong[3], and Junqi Yin[1]

[1] Oak Ridge National Laboratory, Oak Ridge, USA
{archbaldrk,dazevedoef,eisenbackm,yinj}@ornl.gov
[2] Georgia Institute of Technology, Atlanta, USA
echow@cc.gatech.edu
[3] University of Tennessee, Knoxville, USA
{dongarra,flopez,tomov}@icl.utk.edu, {rfebbo,dnicho22}@vols.utk.edu, kwong@utk.edu

Abstract. This paper presents some of the current challenges in designing deep learning artificial intelligence (AI) and integrating it with traditional high-performance computing (HPC) simulations. We evaluate existing packages for their ability to run deep learning models and applications on large-scale HPC systems efficiently, identify challenges, and propose new asynchronous parallelization and optimization techniques for current large-scale heterogeneous systems and upcoming exascale systems. These developments, along with existing HPC AI software capabilities, have been integrated into MagmaDNN, an open-source HPC deep learning framework. Many deep learning frameworks are targeted at data scientists and fall short in providing quality integration into existing HPC workflows. This paper discusses the necessities of an HPC deep learning framework and how those needs can be provided (e.g., as in MagmaDNN) through a deep integration with existing HPC libraries, such as MAGMA and its modular memory management, MPI, CuBLAS, CuDNN, MKL, and HIP. Advancements are also illustrated through the use of algorithmic enhancements in reduced- and mixed-precision, as well as asynchronous optimization methods. Finally, we present illustrations and potential solutions for enhancing traditional compute- and data-intensive applications at ORNL and UTK with AI. The approaches and future challenges are illustrated in materials science, imaging, and climate applications.

This manuscript has been authored by UT-Battelle, LLC., under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

© Springer Nature Switzerland AG 2020
J. Nichols et al. (Eds.): SMC 2020, CCIS 1315, pp. 35–50, 2020.
https://doi.org/10.1007/978-3-030-63393-6_3

1 Background

Deep learning (DL) has become one of the most prevalent technologies today. Applications extend from image recognition to natural language processing. While its commercial applications have garnered the efforts of major technology companies, DL has proven vital to works in numerous scientific domains.

Some of these domain problems are vital to solving many of today's global issues—for instance, in solving modern, data-dense problems in climate science. This field has seen several advancements with the recent integration of deep neural networks (DNNs). Ise et al. [1] propose a modern approach towards utilizing neural networks to predict global temperature data. They further conclude that the accuracy of their model increases with the number of input images used. Thus, the model's accuracy and utility grow with the amount of compute power available. Wang et al. [2] provide further uses for DNNs in weather forecasting, which compete with the current state-of-the-art, physically based simulations. The neural networks can also reduce the computations necessary, as they possess the ability to extend one trained network to similar environments, or train once, and predict frequently.

Additionally, DL has proven useful in materials science, even leading the field of materials informatics [3]. Feng et al. [4] were able to utilize deep networks to predict material defects with an accuracy above 98%, even for small data sets. However, these great accuracies require deep and/or pre-trained networks. As with climate modelling, deep learning can be used to replace current physically based simulations, which are expensive and/or do not scale well [5].

Providing high performance hardware and software stacks to facilitate deep learning has become a focus in the high-performance computing (HPC) field. An overview of the parallel algorithms and their challenges for distributed, deep neural networks can be found in Ben-Nun and Toefler [6]. Large-scale modern systems, such as the US DOE's Summit, are fully loaded with high-performance GPUs to support the compute-intensive nature of deep learning. Additionally, some HPC distributed file systems under-perform in providing I/O for data-hungry DL training [7]. Even in terms of sheer performance, many state-of-the-art deep learning models and libraries fail to utilize a significant portion of the maximum floating-point operations (FLOPs) rate of current top systems [8]. There is significant research in the area of accelerating DL on cloud and other commercial systems, but these methods may not integrate well with varying HPC technologies and modern scientific workflows.

One major area in which HPC can innovate deep learning is model parallelism. In many domain applications, networks can become large, leading to growth beyond the memory of a single GPU. By utilizing existing HPC communication standards, a robust model-parallel training standard needs to be developed in addition to efficient implementations. Currently, several libraries attempt to integrate model parallelism [9], but this requires extensive knowledge of the code base, model, and hardware. There is an active research push into further developing model parallel approaches [10–13].

R. Stevens presented a vision of the landscape of machine learning as DOE moves towards the exascale era [14], stressing the importance of machine learning as an integral part of DOE exascale applications. The new paradigm of "HPC + AI" on exascale computing leads towards an "Integrated Sim, Data, Learn Stack." There has been a long history of application code development for compute-intensive applications in the areas of climate, materials, medical, high energy, and urban sciences applications. In light of these needs, we will highlight developments and discuss the necessity of a set of tools and methods particularly relevant to deploying AI on HPC systems.

Realizing the paradigm of "HPC + AI" requires three major components: (1) the HPC system and investments in scaling application-driven capabilities; (2) the AI software achieving the potential performance; and (3) the "+", innovative algorithms and implementations/tools that combine the components of compute- and data-driven applications seamlessly on exascale machines.

2 Deep Learning Software on Modern HPC Systems

There are countless deep learning frameworks in existence today: TensorFlow [15], PyTorch [16], MxNet [17], and many others. Each framework has its own advantages and each falls short in some areas. One advantage of each of these frameworks is their strong corporate backings. Companies pour significant amounts of money into making production-ready libraries, which support various hardware. However, when using these frameworks in an HPC setting, this can cause issues. Much of the research and advancements by the library designers are production focused and target cloud environments. Some of these advancements translate well onto HPC systems, such as the use of hardware accelerators and mixed precision. However, many communication paradigms and implementations differ dramatically between the two. For example, the parameter server method in TensorFlow is more suited for heterogeneous cloud environments, although it is more bandwidth efficient than allreduce; and the socket-based GLOO communication backend built into PyTorch cannot efficiently utilize high-performance interconnects on HPC systems, although it is more flexible.

2.1 Towards a Deep Learning Framework for HPC

We evaluated a number of existing packages for their readiness to efficiently run deep learning models and applications on large-scale HPC systems. As pointed out previously, we found various limitations. Our vision of what is important and needed in deep leaning AI packages for HPC is incorporated into the design of the MagmaDNN open source framework [18,19] (see Fig. 1) with current release MagmaDNN v1.2 [20], which we discuss briefly in what follows.

Firstly, MagmaDNN targets HPC systems, rather than cloud systems. It is written in C++ and is open source (available at bitbucket.org/icl/magmadnn).

Many of the core components of MagmaDNN are the same as other frameworks: operations are represented in a compute graph, data is passed around via tensors, gradients are computed using backpropagation, etc. However, each of these was designed with an HPC environment in mind.

MagmaDNN is centered around existing high-performance software packages, enabling easy integration into existing environments and Exascale Computing Project (ECP) software products. First and foremost, MagmaDNN uses the Matrix Algebra on GPU and Multicore Architectures (MAGMA) [21] package for heterogeneous linear algebra, since deep learning is heavily dependent on linear algebra routines. For strictly CPU computations, the package allows the use of any Basic Linear Algebra Subprograms (BLAS)/LAPACK packages such as Intel MKL, OpenBLAS, ATLAS, etc.

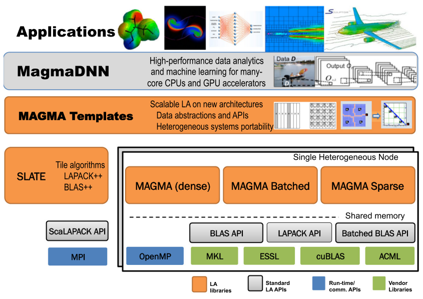

Fig. 1. MagmaDNN software stack.

To better operate with hardware accelerators, MagmaDNN uses CUDA packages to harness GPU acceleration. Custom CUDA kernels, along with CUDNN's routines for GPU-accelerated deep learning, allows the package to get maximal performance on GPU-enabled systems.

For distributed computing, MagmaDNN relies on existing HPC tools such as the Message Passing Interface (MPI) for its communication. Most systems have an MPI distribution tuned to that hardware, while some even have custom distributions. Relying on MPI for inter-node communication allows the framework to utilize existing optimized implementations while providing simpler integration with existing scientific codes.

This integration of existing HPC frameworks allows MagmaDNN to compile and link with the fastest libraries for particular hardware. Utilizing existing optimized packages is a consistent design pattern within HPC libraries, and MagmaDNN uses interfaces that are commonly installed on HPC clusters. In terms of integrating with existing systems and scientific codes, this approach offers the package a significant advantage over other frameworks and workflows.

Not only does MagmaDNN integrate with current HPC frameworks, it provides modularity in its design and linking so that users have flexibility in use. Conforming to standard APIs allows linking with any choice of framework; any BLAS/LAPACK/MPI choice implementations can be used. However, it is not just libraries with which MagmaDNN is modular. By utilizing computation graphs and extendable C++ classes, the package allows users to fully customize their neural networks. For instance, to utilize a unique network interface for distributed training, one can define a custom node on the computation graph which handles the transfer of data on that network component. This can be further utilized to define any desired data movement or computational elements. By providing modular components, MagmaDNN provides a structure onto which domain scientists can easily implement desired custom functionality.

Much of this customizability is fully supported through MagmaDNN's interfaces. For instance, working with GPU and distributed memory is abstracted away in the `MemoryManager` class. Thus, library users do not need to worry about where memory is stored and how it is transferred. The package identifies the memory type, and whenever a memory operation is used (e.g., copy, put, get, etc.), and the MagmaDNN implementation handles the low-level specifics. Likewise, communication operations between devices or nodes are abstracted into the library as well. However, users can define their own operations, meaning they can redefine those in existing code. This makes it easy, for example, to try out different convolution algorithms and determine which one performs optimally. Another common example is to determine which distribution strategy is optimal for a specific machine. The same training code will run on any machine, and only the communication code might need to be altered.

Looking towards next-generation hardware and exascale systems, MagmaDNN aims to incorporate more technology stacks such as AMD's HIP (e.g., hipMAGMA is already available) and Intel's OneAPI. Support for more technologies will allow researchers at various labs and institutions to focus on domain research—and not on how to integrate their code with HPC stacks. We also hope to provide streamlined and optimal support for model parallelism. As the size of models grows, so does the necessary memory requirements on devices. Soon many cutting-edge networks, which can provide state-of-the-art accuracy in classification, will require model parallelism. Supporting fast, scalable model parallelism on HPC systems is crucial to facilitating next-generation research, and MagmaDNN is moving towards this goal.

2.2 Workflow Software for Modern HPC Systems

Another major component in the integration of AI and HPC is the availability of workflow software: which is critical, for example, in launching, collecting results from, and analyzing various simulations in hyperparameter tuning, DL networks discoveries, etc. Exascale applications largely have their own set of workflow procedures and tools to launch their simulations. Integrating traditional applications seamlessly with existing AI software—either TensorFlow, PyTorch, or MagmaDNN—takes on a new level of challenges. Most workflow frameworks available today focus on cloud deployment, yet a workflow framework tailored to scale on HPC systems is critical to the success to "Automate and Accelerate" the task of "Sim, Data, and Learn Stack". SWIFT-T [22] is a tool that offers the choice to run on large-scale HPC systems. At The University of Tennessee, Knoxville (UTK), we have developed a parallel workflow framework called openDIEL that aims to give researchers and users of HPC machines an efficient way to coordinate, organize, and interconnect many disparate modules of computation in order to effectively utilize and allocate HPC resources [23,24]. It provides users an adaptive avenue to run compute-intensive and data science jobs concurrently, allowing specification of DNN architectures, data processing, and hyperparameter tuning. Existing ML tools can be readily used in openDIEL, allowing for easy experimentation with various models and approaches. Most importantly, openDIEL provides a platform to run existing ECP applications in their own workflow procedures. When conducting multi-discipline simulations, it is often required to use a large variety of software, and serial or parallel codes, to answer a research question. Utilizing disparate modules of computation requires the careful coordination of dependencies, communication, and synchronization between them, and there is not always a clear path. This problem is solved by openDIEL: it enables researchers to define complex dependencies between modules and schedule communication between them. OpenDIEL consists of three primary components: the workflow control file, the communication library (COMMLIB), and the Executive, shown in Fig. 2. Workflow is defined by users via the control file, which consists of two parts, the definition of functional modules and the interactions and sequence of execution of modules. OpenDIEL provides two communication libraries, one for direct point-to-point data transfer between two modules, and a store-and-move tuple space library for collective and asynchronous data transfers. The Executive is a lightweight manager designed to run and scale on heterogeneous HPC platforms.

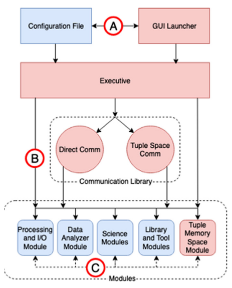

Fig. 2. OpenDIEL structure.

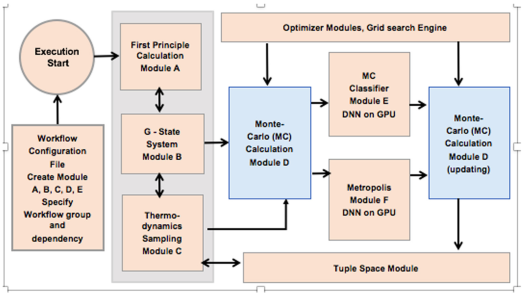
Execution Start
Workflow Configuration File Create Module A, B, C, D, E Specify Workflow group and dependency
First Principle Calculation Module A
G - State System Module B
Thermo-dynamics Sampling Module C
Optimizer Modules, Grid search Engine
Monte-Carlo (MC) Calculation Module D
MC Classifier Module E DNN on GPU
Metropolis Module F DNN on GPU
Monte-Carlo (MC) Calculation Module D (updating)
Tuple Space Module

Fig. 3. Workflow in materials sciences.

OpenDIEL specifically focuses on unifying modules into a single executable that uses MPI for communication. Climate and materials sciences simulations, with typical workflow shown in Fig. 3, are two of the applications that are incorporating deep learning into their simulations.

3 Algorithmic Improvements for DNN AI in HPC

In this section, we select and highlight the development of two technologies that we believe are critical for the scalability and efficient use of HPC systems. The first (Sect. 3.1) addresses scalability of DL training, stressing the benefits and need for asynchronous methods on today's highly parallel and heterogeneous systems, featuring powerful nodes and relatively slow interconnects. The second (Sect. 3.2) addresses hardware changes motivated by DL—in particular powerful hardware acceleration for low-precision arithmetic—and techniques for harnessing this in scientific HPC applications.

3.1 Asynchronous Methods

In a multi-core or many-core shared-memory environment, the speed at which the parameters can be updated by different threads may be a limiting factor for stochastic gradient descent (SGD) performance. This is because threads must lock the parameter vector while they are updating it, preventing other threads from reading or updating the parameters, so that they must wait. Asynchronous versions of SGD may relax the data dependencies and order of operations compared to the classical SGD algorithm, allowing the algorithm to update parameters more rapidly—but also making the asynchronous algorithm mathematically different and non-deterministic in its execution [25].

In a distributed-memory environment, a common implementation option for SGD is for each compute node to store a copy of the model parameters [26]. When each compute node has computed a gradient for its mini batch of data, all the gradients must be summed and then shared among the compute nodes so that they can update their copies of the model parameters. This leads to a potential bottleneck in the global all-reduce computation at every step of SGD. In addition, nodes that compute their gradients faster than others must wait for the other nodes (sometimes called "stragglers"). This can be a very high synchronization overhead that may be difficult to reduce by load balancing on heterogeneous architectures. An asynchronous approach can address these high overheads, as we demonstrate below.

Another common implementation option for distributed-memory computers is the parameter server approach [27]. Here, one or more compute nodes are dedicated to storing and updating the model parameters. The worker nodes compute gradients for their partition of the data and send these gradients to the parameter server. The parameter server then sends back the updated values of the model parameters. The parameter server model can be used to run SGD synchronously—with the same issues of stragglers slowing down the iteration for all nodes—but it has the advantage that it can also be run asynchronously (i.e., the worker nodes do not have to send gradients at the same time). Workers, however, may be working with different sets of values for the parameters at any instant. This makes the computation asynchronous. In distributed memory implementations, the gradients are typically updated by the parameter server using atomic memory operations, which differs from the shared memory case.

Asynchronous parameter server implementations were proposed around 2012 or earlier, but it was Hogwild! [25], with inconsistent reads and writes of the parameter vector in shared memory, that popularized the asynchronous approach. In particular, Hogwild! showed that, at least in cases of convex problems and gradient sparsity, asynchrony with inconsistent reads and writes does not have to harm convergence asymptotically. Hogwild! was extended in many directions, such as HogBatch [28], designed for efficiently running on multi-core CPU architectures, and Buckwild [29] which exploits low-precision arithmetic to accelerate the training. Improvements on the analysis of Hogwild! showed that convergence can be maintained for the non-convex and non-sparse case [30]. In MagmaDNN, we designed a parallel asynchronous variant of SGD that is similar to a parameter server approach, but on multi-core CPU and GPU architectures [31].

In practice, the accuracy of machine learning models can be dramatically improved by increasing the number of model parameters as well as the sizes of the training sets. As a result, the memory footprint needed for model training often exceeds the memory storage available on a single node. In such cases, models must be trained on large-scale machines where the dataset, and possibly the model as well, are distributed among the compute nodes. From a performance perspective, it is crucial to efficiently parallelize the training in this setup—and this means overcoming the communication costs. It has been shown that when targeting large-scale platforms, asynchronous algorithms can outperform the synchronous ones as in the shared-memory case. In Google's DistBelief framework [27], for example, the Downpour SGD algorithm uses a centralized parameter server for storing the model, and distributes the dataset to other participating nodes. Asynchrony enables communications to overlap with computations, thus improving resource usage. In addition, the algorithm is particularly well adapted to heterogeneous computing environments where resources may have different processing and communication speeds.

In MagmaDNN, we take a different approach, exploiting remote memory access (RMA) using MPI one-sided communication capabilities for implementing asynchronous SGD in order to maximize the scalability of parallel training. This approach has already been proven efficient in implementing several asynchronous numerical algorithms such as the Jacobi [32] or optimized Schwarz [33] methods for solving sparse systems of linear equations.

3.2 Reduced and Mixed Precision

Deep neural networks can be efficiently trained using low-precision floating-point arithmetic, such as single (fp32) or lower precision, improving the training times on modern hardware [34]. Popular floating-point formats include the 16-bit IEEE half-precision format (fp16) and the 16-bit bfloat16 [35] format. With fewer bits in the mantissa compared to fp16, the bfloat16 format offers less precision but has the advantage of having the same range as fp32—thereby removing the risk of overflow and underflow when converting from fp32 data.

Motivated by the success of low-precision arithmetic in machine learning applications, many specialized hardware accelerators have been designed with reduced- or mixed-precision arithmetic capabilities. For example, the NVIDIA Tensor Cores introduced in the Volta GPU can issue 4×4 matrix multiplication $D = C + A * B$ in one clock cycle, where the input matrices A and B are fp16, whereas D and C can be either fp16 or fp32. The theoretical performance peak on Volta GPU is 125 teraFLOP/s, which is 8 times the performance peak for fp32; and, although the accuracy of tensor core operations is lower than fp32, the ability to accumulate the result of matrix multiplication in fp32 rather than fp16 yields a considerable gain with respect to accuracy over fp16 [36]. In [37], authors use Tensor Cores for computing a reduced-precision LU factorization of a dense linear system, initially generated in double precision (fp64), and use these factors as a preconditioner for a Krylov solver in order to retrieve an fp64 solution. By doing this, they manage to reduce the time to solution up to 4× fp64 arithmetic for the factorization.

Tensor Cores can also be used in performing FFTs [38]. Note that a matrix in fp32 can be well approximated as the scaled sum of two fp16 matrices

$$A_{32} \approx a1_{32} * A1_{16} + a2_{32} * A2_{16} \tag{1}$$

where $a1_{32}$ and $a2_{32}$ are in fp32. Scaling by $a1_{32}(a2_{32})$ ensures $A1_{16}(A2_{16})$ is within the limited dynamic range of fp16. Conceptually, $A1_{16}$ roughly captures the 3 most significant decimal digits, and $A2_{16}$ the next 3 lower significant decimal digits. If A_{32} is already well scaled, then one can expect $a1_{32} = 1$ and $a2_{32} = 2^{-11} * a1_{32}$. Matrix multiplication of two fp32 matrices can be approximated by 3 matrix multiplications of fp16 matrices on Tensor Cores, which may give a theoretical speedup of 8/3× on Volta GPU. For another perspective, Eq. (1) suggests the form of operations in DNN are theoretically able to well approximate operations of fp32 using mixed-precision operations in fp16.

4 Applications

4.1 Materials Science and Microscopy

There are multiple opportunities to exploit machine learning techniques in materials science. While many applications have concentrated on materials discovery—as exemplified by the use of ML in conjunction with databases such as The Materials Project [39], AFLOW [40], or OQMD [41]—a tightly coupled integration of traditional simulation techniques has great promise for bridging the accuracy and computational cost divide between first principles calculations and effective models. One promising combination for a tight coupling is for first principles statistical mechanics of materials to calculate the temperature dependence of materials. This requires the calculations of many possible atomic configurations within these materials using a Monte Carlo approach, where the probability of the individual states would be evaluated using an expensive density functional theory calculation [42]. Thus, we will utilize ML to construct surrogate models as an intermediate step to link the first principles calculations to the Monte Carlo simulation. Here, the calculation of ordering phase transitions in alloys might serve as an example [43]. In a solid solution alloy, different chemical species can occupy the sites of an underlying lattice structure (note that the total number of states grows exponentially with the number of atoms in the system). Each of these possible configurations has a different energy that determines the probability of the system being found in this state at a given temperature. To build a model for these interactions, density functional calculations for representative configurations O(1,000–10,000) will be performed to train a surrogate model that can replace the expensive (in the order of multiple node hours per data point) first principles calculation within the Monte Carlo sampling of possible configurations. While this approach can be conducted in a linear workflow, DFT $\rightarrow$ ML $\rightarrow$ MC, we envision a tighter coupled workflow, which augments the original training set with new points from important regions of the phase space discovered during the MC simulation, and retrains the model to improve the quantitative predictive power of this approach.

A long-standing inverse problem in atomic imaging is the loss of phase information during measurement (a.k.a., the phase problem). Given the sparse data collection of scanning transmission electron microscopic (STEM) images on different types of materials, a comprehensive database is needed for the community to study the subject. State-of-the-art electron microscopes produce focused electron beams with atomic dimensions and capture of convergent beam electron diffraction (CBED) patterns. In this dataset [44], we use newly developed electron scattering simulation codes to generate CBED patterns from over 60,000 materials (solid-state materials) from a material project database, representing nearly every known crystal structure. A data sample from this data set is given by a 3D array formed by stacking 3 CBED patterns simulated from the same material at 3 distinct material projections (i.e., crystallographic orientations). Associated with each data sample in the data set is a host of material attributes or properties which are, in principle, retrievable via analysis of this CBED stack. These consists of the crystal space group to which the material belongs, atomic lattice constants and angles, and chemical composition, to name but a few. Of note is the crystal space group attribute (or label).

This dataset could be augmented with experimental data in the future. The generated dataset, based on simulations, will emphasize the scalability aspect of the model with the use of very large images ($\sim$10 GB per image).

4.2 Super-Resolution for HPC Simulations

Image compression is a very active field of study, with new methods being constantly generated [45]. The need for improvements in image compression quality is growing in the field of HPC simulations because of the exponential trend in data generation. There exists an untapped potential in this situation due to the nature of simulated data that is not currently exploited. Simulation data from numerical systems of partial differential equations exist on a solution manifold [46]. Thus, the manifold hypothesis in machine learning—which states that real-world, high-dimensional data lie on low-dimensional manifolds embedded within the high-dimensional space—is concrete for simulation data. We can therefore expect that identifying this map to the low-dimensional manifold will provide ideal compression for HPC Simulations. In the next paragraph we describe the basic setup that allows researchers to test in situ machine learning for climate simulations.

The shallow water equations on a sphere is a well-known model used in climate science to simulate basic dynamics of the Earth's atmosphere. There exist many test case scenarios for this model [47–50], and in this paper we use the challenging test known as the barotropic instability test [50], which involves perturbing the atmospheric flow with a localized bump to the balanced height field. We simulate the shallow water equations on the sphere by using a fourth-order Runge-Kutta method [51] for time-stepping and a discontinuous Galerkin (DG) spatial discretization with a Legendre basis on a cubed-sphere mesh [52,53].

The data structure for this simulation consists of a time series of images for each of the six faces of the cube, which for analysis can be converted to a single series lat-lon gridded images. Arguably, the most commonly used form of lossy compression is the discrete cosine transform (DCT) method [54]. ECP has two projects on the development of FFTs and related transformations, one of which is heFFTe [55], already integrated in the ECP software stack, delivering high-performance (close to 90% of the theoretical roofline peak), and very good weak and strong scaling on systems like Summit at ORNL. This common compression method fills out the setup of the in situ machine learning super resolution method. At each time step of the barotropic instability simulation, in situ machine learning methods are exposed to both the compressed image data and the original image. As the simulation progresses, only the compressed image data is stored to disk, and the machine learning method adaptively learns correct super-resolution transformation of the compressed data. Final stored data contain all compressed image data with the trained machine learning method. It has been demonstrated in [56] that, using this setup, it is possible to train in situ networks to reduce the error in lossy compression—obtaining multiple orders of improvement. Going forward, it will be necessary to further improve in situ compression and analysis in order to maximize discovery with HPC simulations.

5 Meeting Exascale

Exascale systems will appear in 2021. These machines incorporate accelerators from NVIDIA, AMD, and Intel. Efforts in preparing the software stack for exascale systems have been a major focus in the DOE ECP program [57]. We believe a native DNN framework such as MagmaDNN will pave a unique path to meet the challenges of exascale platforms. Assisted by the openDIEL parallel workflow engine, which admits a diverse spectrum of applications as functional modules, we will be able to exploit the full capabilities of exascale machines. In-situ data augmentation will be incorporated with compute-intensive simulations, leading to discovery in multi- and inter-disciplinary, systems-wide, real-time recommendation systems. From instrumentation calibration, experimental and observable results, theoretical simulations, to validation and analysis—exascale computing will be brought to bear on health, transportation, environment, and social sciences. For example, climate and weather recommendation systems will be able to integrate models in storms prediction, rainfall, vegetation, farm growth, pollution, power usage, traffic flow, etc. [58–62]. As inputs and outputs from many sensor devices become ubiquitous, the importance of a scalable AI framework and an extensible workflow tool will grow.

Challenges will rise from algorithmic approaches as HPC systems continue to expand and evolve. Multiple-precision implementation will be unavoidable. Setting the basis and preparing for product diversity from different vendors will be an important consideration of an AI framework, as well. One challenge in performance is to reduce the impact of communication while maintaining good convergence of the algorithm, such as SGD, in the DNN framework. In the Horovod framework [26], for example, the solution is to distribute the training of neural network on large machines using an efficient gradient reduction strategy referred to as ring-allreduce. Notably, in the 2018 Gordon-Bell award winning paper [63], Horovod was used in the context of TensorFlow to train two DNN models on large systems, including the Summit machine, to detect extreme weather patterns from climate data. Similarly, Horovod was used in [64] to train a large DNN model for tackling inverse problems in imaging. Algorithmic improvements towards individual or combined synchronous, asynchronous, and pipeline approaches are essential to improve resource usage as well as convergence.

There is a trend in increasingly larg model sizes (especially in NLP—the latest model [65] has 17 Billion parameters), and a need in scientific applications (e.g., geospatial imaging) to process larger-dimension inputs. Although there exist exascale deep learning applications [63,66] on pre-exascale system such as Summit thanks to the Tensor Core technology, those use cases push the limit on large-batch training for data parallelism and are not generally applicable to exascale learning challenges. Early efforts [9,67] on model parallelism have made progress, but are not yet mature or generic. An AI framework that can efficiently exploit various level of parallelisms (data parallel, model parallel, hyperparameter search, etc.) will be in demand.

6 Conclusion

Exascale systems are an important investment in the US. With exascale, we envision a stronger economy and improved quality of life. It will also lead to important scientific discovery and resolution of complex issues related to national security. Development in AI software and tools that scale well on these systems is important, and even more critical for AI frameworks that also work well across the existing spectrum of exascale applications. Although many challenges exist, a primary roadblock is the lack of direct collaborative effort, and a software platform that values performance as the foremost priority. In this paper, we present a unique set of AI tools and algorithms, as well as efforts between collegiate and ORNL researchers, demonstrating that there is a pathway to integrate and deploy machine learning to those ends—with emphasis on two major applications on exascale systems.

Acknowledgments. This material is based upon work supported in part by the Laboratory Directed Research and Development program at the Oak Ridge National Laboratory, which is operated by UT-Battelle, LLC., for the U.S. Department of Energy under Contract DE-AC05-00OR22725. The work was partly supported by the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, with specific thanks to the FASTMath Institutes.

This work was conducted at the Joint Institute for Computational Sciences (JICS) and the Innovative Computing Laboratory (ICL), sponsored by the National Science Foundation (NSF), through NSF REU Award #1659502 and NSF Award #1709069. This work used hardware donations from NVIDIA as well as the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by NSF grant number ACI-1548562. Computational Resources are available through a XSEDE education allocation award TG-ASC170031.

References

1. Ise, T., Oba, Y.: Forecasting climatic trends using neural networks: an experimental study using global historical data. Front. Robot. AI **6**, 32 (2019)
2. Wang, J., Balaprakash, P., Kotamarthi, R.: Fast domain-aware neural network emulation of a planetary boundary layer parameterization in a numerical weather forecast model. Geosci. Model Dev. **12**(10), 4261–4274 (2019)
3. Agrawal, A., Choudhary, A.: Deep materials informatics: applications of deep learning in materials science. MRS Commun. **9**(3), 779–792 (2019)
4. Feng, S., Zhou, H., Dong, H.: Using deep neural network with small dataset to predict material defects. Mater. Des. **162**, 300–310 (2019). Citation Key: FENG2019300
5. Ye, W., Chen, C., Wang, Z., Chu, I.-H., Ong, S.P.: Deep neural networks for accurate predictions of crystal stability. Nat. Commun. **9**, 3800 (2018)
6. Ben-Nun, T., Hoefler, T.: Demystifying parallel and distributed deep learning: an indepth concurrency analysis. ACM Comput. Surv. **1**, 1–37 (2019)
7. Han, J., Xu, L., Rafique, M.M., Butt, A.R., Lim, S.: A quantitative study of deep learning training on heterogeneous supercomputers. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER), pp. 1–12 (2019)
8. You, Y., Zhang, Z., Hsieh, C.-J., Demmel, J., Keutzer, K.: ImageNet training in minutes. In: Proceedings of the 47th International Conference on Parallel Processing, ICPP 2018. NY, USA. Association for Computing Machinery, New York (2018)
9. Shazeer, N., et al.: Mesh-TensorFlow: deep learning for supercomputers. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 31, pp. 10414–10423, Curran Associates Inc. (2018)
10. Geng, J., Li, D., Wang, S.: ElasticPipe: an efficient and dynamic model-parallel solution to DNN training. In: Proceedings of the 10th Workshop on Scientific Cloud Computing, ScienceCloud 2019, New York, USA, pp. 5–9. Association for Computing Machinery (2019)
11. Huang, Y.: GPipe: efficient training of giant neural networks using pipeline parallelism. In: Advances in Neural Information Processing Systems, vol. 32, pp. 103–112. Curran Associates Inc. (2019)
12. Gholami, A., Azad, A., Jin, P., Keutzer, K., Buluc, A.: Integrated model, batch, and domain parallelism in training neural networks. In: Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, New York, USA, pp. 77–86. Association for Computing Machinery (2018)
13. Chen, C.-C., Yang, C.-L., Cheng, H.-Y.: Efficient and robust parallel DNN training through model parallelism on multi-GPU platform. arXiv abs/1809.02839 (2018)

14. Stevens, R.: Exascale computing: the coming integration of simulation, data and machine learning. In: European Technology Platform for High-Performance Computing (ETP4HPC) ISC'18 Workshop (2018)
15. Abadi, M., et al.: Tensorflow: large-scale machine learning on heterogeneous distributed systems. CoRR, vol. abs/1603.04467 (2016)
16. Paszke, A., et al.: PyTorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, vol. 32, pp. 8026–8037. Curran Associates Inc. (2019). Citation Key: NIPS2019_9015
17. Chen, T., et al.: MXNet: a flexible and efficient machine learning library for heterogeneous distributed systems. CoRR, vol. abs/1512.01274 (2015)
18. Nichols, D., Wong, K., Tomov, S., Ng, L., Chen, S., Gessinger, A.: MagmaDNN: accelerated deep learning using MAGMA. In: Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning), PEARC 2019. NY, USA. Association for Computing Machinery, New York (2019)
19. Nichols, D., Tomov, N.-S., Betancourt, F., Tomov, S., Wong, K., Dongarra, J.: MagmaDNN: towards high-performance data analytics and machine learning for data-driven scientific computing. In: Weiland, M., Juckeland, G., Alam, S., Jagode, H. (eds.) ISC High Performance 2019. LNCS, vol. 11887, pp. 490–503. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34356-9_37
20. Nichols, D., Febbo, R., Lopez, F., Wong, K., Tomov, S., Dongarra, J.: MagmaDNN (Version 1.2), July 2020. https://doi.org/10.5281/zenodo.3972406
21. Tomov, S., Dongarra, J., Baboulin, M.: Towards dense linear algebra for hybrid GPU accelerated manycore systems. Parallel Comput. **36**, 232–240 (2010)
22. Ozik, J., Collier, N., Wozniak, J., Spagnuolo, C.: From desktop to large-scale model exploration with Swift/T. In: Proceedings of the 2016 Winter Simulation Conference (2016)
23. Wong, K., Trzil, Z.: Tuple space implementation in a parallel workflow engine, OpenDIEL. In: Student Paper, PEARC 2018 (2018)
24. Betancourt, F., Wong, K., Asemota, E., Marshall, Q., Nichols, D., Tomov, S.: openDIEL: a parallel workflow engine and data analytics framework. In: Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines Learning, Student Paper, PEARC 2019, NY, USA. Association for Computing Machinery, New York (2019)
25. Niu, F., Recht, B., Re, C., Wright, S.J.: HOGWILD! A lock-free approach to parallelizing stochastic gradient descent. In: Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS 2011, Red Hook, NY, USA, pp. 693–701. Curran Associates Inc. (2011)
26. Sergeev, A., Balso, M.D.: Horovod: fast and easy distributed deep learning in TensorFlow. arXiv preprint arXiv:1802.05799 (2018)
27. Dean, J., et al.: Large scale distributed deep networks. In: Advances in Neural Information Processing Systems, vol. 25, pp. 1223–1231. Curran Associates Inc. (2012)
28. Sallinen, S., Satish, N., Smelyanskiy, M., Sury, S.S., Ré, C.: High performance parallel stochastic gradient descent in shared memory. In: IPDPS, pp. 873–882. IEEE Computer Society (2016)
29. Sa, C.D., Zhang, C., Olukotun, K., Ré, C.: Taming the wild: a unified analysis of HOG WILD! -style algorithms. In: Proceedings of the 28th International Conference on Neural Information Processing Systems, NIPS 2015, Cambridge, MA, USA, vol. 2, pp. 2674–2682. MIT Press (2015)

30. Lian, X., Huang, Y., Li, Y., Liu, J.: Asynchronous parallel stochastic gradient for nonconvex optimization. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 28, pp. 2737–2745. Curran Associates Inc. (2015)
31. Lopez, F., Chow, E., Tomov, S., Dongarra, J.: Asynchronous SGD for DNN training on shared-memory parallel architectures. Technical report, ICL-UT-20-04. Innovative Computing Laboratory, University of Tennessee (March 2020). (To appear in IPDPSW'20 proceedings)
32. Wolfson-Pou, J., Chow, E.: Modeling the asynchronous Jacobi method without communication delays. J. Parallel Distrib. Comput. **128**, 6 (2019)
33. Yamazaki, I., Chow, E., Bouteiller, A., Dongarra, J.: Performance of asynchronous optimized Schwarz with one-sided communication. Parallel Comput. **86**, 66–81 (2019)
34. Courbariaux, M., Bengio, Y., David, J.-P.: Training deep neural networks with low precision multiplications (2014)
35. Intel Corporation: BFLOAT16–Hardware Numerics Definition. White paper. Document number 338302–001US, November 2018
36. Blanchard, P., Higham, N.J., Lopez, F., Mary, T., Pranesh, S.: Mixed precision block fused multiply-add: error analysis and application to GPU tensor cores. SIAM J. Sci. Comput. **42**(3), C124–C141 (2020)
37. Haidar, A., Tomov, S., Dongarra, J., Higham, N.J.: Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, pp. 1–11 (2018)
38. Sorna, A., Cheng, X., D'Azevedo, E., Wong, K., Tomov, S.: Optimizing the Fast Fourier Transform using mixed precision on tensor core hardware. In: 2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW), pp. 3–7 (2018)
39. Jain, A., et al.: Commentary: the materials project: a materials genome approach to accelerating materials innovation. APL Mater. **1**(1), 011002 (2013)
40. Gossett, E., et al.: AFLOW-ML: a RESTful API for machine-learning predictions of materials properties. Comput. Mater. Sci. **152**, 134–145 (2018)
41. Kirklin, S., et al.: The Open Quantum Materials Database (OQMD): assessing the accuracy of DFT formation energies. npj Comput. Mater. **1**, 15010 (2015)
42. Eisenbach, M., Zhou, C.-G., Nicholson, D.M., Brown, G., Larkin, J., Schulthess, T.C.: A scalable method for Ab Initio computation of free energies in nanoscale systems. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC 2009, New York, NY, USA, pp. 64:1–64:8. ACM (2009)
43. Eisenbach, M., Pei, Z., Liu, X.: First-principles study of order-disorder transitions in multicomponent solid-solution alloys. J. Phys. Condens. Matter **31**, 273002 (2019)
44. Laanait, N., Borisevich, A., Yin, J.: A Database of Convergent Beam Electron Diffraction Patterns for Machine Learning of the Structural Properties of Materials (May 2019). https://doi.org/10.13139/OLCF/1510313
45. Sayood, K.: Introduction to Data Compression. The Morgan Kaufmann Series in Multimedia Information and Systems. Elsevier Science (2017)
46. Rheinboldt, W.C.: On the computation of multi-dimensional solution manifolds of parametrized equations. Numer. Math. **53**(1), 165–181 (1988)

47. Williamson, D., Drake, J., Hack, J., Jakob, R., Swarztrauber, P.: A standard test set for numerical approximations to the shallow water equations in spherical geometry. J. Comput. Phys. **102**, 211–224 (1992)
48. Nair, R.D., Jablonowski, C.: Moving vortices on the sphere: a test case for horizontal advection problems. Mon. Weather Rev. **136**(2), 699–711 (2008)
49. Mcdonald, A., Bates, J.R., McDonald and Bates: Semi-Lagrangian integration of a shallow water model on the sphere. Mon. Weather Rev. **117**, 130 (1989)
50. Galewsky, J., Scott, R., Polvani, L.: An initial-value problem for testing numerical models of the global shallow-water equations. Tellus **56A**, 429–440 (2004)
51. Abramowitz, M., Stegun, I. (eds.): Handbook of Mathematical Functions, chap. 9. Dover Publications (1972)
52. Sadourny, R.: Conservative finite-difference approximations of the primitive equations on quasi-uniform spherical grids. Mon. Weather Rev. **100**(2), 136–144 (1972)
53. Nair, R., Thomas, S., Loft, R.: A discontinuous Galerkin global shallow water model. Mon. Weather Rev. **133**(4), 876–888 (2005)
54. Ahmed, N., Natarajan, T., Rao, K.R.: Discrete Cosine transform. IEEE Trans. Comput. **C–23**(1), 90–93 (1974)
55. Tomov, S., Ayala, A., Haidar, A., Dongarra, J.: FFT-ECP API and high-performance library prototype for 2-D and 3-D FFTs on large-scale heterogeneous systems with GPUs. ECP WBS 2.3.3.13 Milestone Report FFT-ECP STML13-27. Innovative Computing Laboratory, University of Tennessee (2020)
56. Lee, S., et al.: Improving scalability of parallel CNN training by adjusting mini-batch size at run-time. In: 2019 IEEE International Conference on Big Data (Big Data), pp. 830–839 (2019)
57. Exascale Computing Project. https://www.exascaleproject.org
58. Zhou, K., Zheng, Y., Li, B., Dong, W., Zhang, X.: Forecasting different types of convective weather: a deep learning approach. J. Meteorol. Res **33**(5), 797–809 (2019). https://doi.org/10.1007/s13351-019-8162-6
59. Samsi, S., Mattioli, C., Mark, V.: Distributed deep learning for precipitation nowcasting. In: 2019 IEEE High Performance Extreme Computing Conference (HPEC) (2019)
60. Keaney, M., Neal, T.: Comparing Deep Neural Network and Econometric Approaches to Predicting the Impact of Climate Change on Agricultural Yield. UNSW Economics Working Paper (2020)
61. Yi, X.X., Zhang, J., Wang, Z., Li, T., Zheng, Y.: Deep distributed fusion network for air quality prediction. In: Proceedings of KDD 2018, London, United Kingdom (2018)
62. Chen, K., Chen, K., Wang, Q., He, Z., Hu, J., He, J.: Short-term load forecasting with deep residual networks. arXiv abs/1805.11956v1 (2018)
63. Kurth, T., et al.: Exascale deep learning for climate analytics. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018. IEEE Press (2018)
64. Laanait, N., et al.: Exascale deep learning for scientific inverse problems (2019)
65. Rajbhandari, S., Rasley, J., Ruwase, O., He, Y.: ZeRO: memory optimization towards training a trillion parameter models. arXiv abs/1910.02054 (2019)
66. Laanait, N., et al.: Exascale deep learning for scientific inverse problems. arXiv. abs/1909.11150 (2019)
67. Huang, Y., et al.: GPipe: efficient training of giant neural networks using pipeline parallelism. arXiv abs/1811.06965 (2018)