FY20 Proxy App Suite Release

Report for ECP Proxy App Project Milestone ADCD-504-10

David F. Richards[1], Yuri Alexeev[2], Xavier Andrade[1], Ramesh Balakrishnan[2], Hal Finkel[2], Graham Fletcher[2], Cameron Ibrahim[2], Wei Jiang[2], Christoph Junghans[3], Jeremy Logan[4], Amanda Lund[2], Danylo Lykov[2], Robert Pavel[3], Vinay Ramakrishnaiah[3], and The ECP Proxy App Team[6]

[1]Lawrence Livermore National Laboratory, Livermore, CA
[2]Argonne National Laboratory, Chicago, IL
[3]Los Alamos National Laboratory, Los Alamos, NM
[4]Oak Ridge National Laboratory, Oak Ridge, TN
[5]https://proxyapps.exascaleproject.org/team

Sept 2020

LLNL-TR-815174

Contents

1 Executive Summary

Version 4.0 of the ECP Proxy App Suite is practically unchanged from the previous release. The current set of proxies has proven useful for many aspects of benchmarking and co-design and we see little reason to alter the suite.

Although there have been few changes to the ECP suite, the team has been hard at work in other areas. In the area of Machine Learning (ML) we have now created a separate proxy suite dedicated to this scientific applications of ML. The suite includes

miniGAN (Generative Adversarial Networks)
miniRL (Reinforcement Learning)
CRADL (inline inference)
Cosmoflow-Benchmark (Convolutional Neural Network)
MLPerf-DeepCam (Climate Segmentation Benchmark)

Section 3 contains more information about these proxies as well as the principles that are guiding the development of the suite.

We have surveyed available proxies for several application domains including Computational Fluid Dynamics, Quantum Chemistry, Quantum Computing Simulation, Molecular Dynamics, Monte Carlo Transport, and Density Functional Theory to identify gaps in proxy coverage. Several new proxy apps are either already available or will be released soon to fill these gaps. Section 4 provides full details.

Finally, section 5 reports on our continued collaboration with the ECP Continuous Integration (CI) effort to use proxy apps to help identify problems and roadblocks to cross-lab CI. We have also assisted the El Capitan Center of Excellence (COE) to stand up a CI system that will be used to test releases of HPE and AMD software stacks. We hope that the COE effort can serve as a model for ECP by showing how the Proxy App Team can work with AD and ST teams identify critical features, kernels, patterns, etc. and incorporate them into a CI system that will help ensure that Frontier and Aurora will provide those needed capabilities.

2 The Proxy App Suite v4.0

The ECP Proxy App Team released version 4.0 of the ECP Proxy App Suite on October 1, 2020 (https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/) . This release is practically unchanged from the previous release. The current set of proxies has proven useful for many aspects of benchmarking and co-design and we see little reason to alter the suite.

Release 4.0 of the ECP Proxy App Suite includes the following proxy apps:

AMG Boomer AMG linear solver from Hypre. Includes both setup and solve phase. Sparse matrices.

Ember A collection of communication patterns specifically designed for use with network simulators.

ExaMiniMD Classical Molecular Dynamics from the CoPA co-design center. Features both a simple Lennard-Jones potential and the computationally expensive SNAP potential.

Laghos High-order finite element Lagrangian hydro in 2D and 3D. Supports multiple orders for thermodynamics and kinematics.

MACSio Generates complex I/O patterns consistent with multiphysics codes.

miniAMR 3D stencil with Adaptive Mesh Refinement.

miniQMC Simplified implementation of real space quantum Monte Carlo algorithms.

miniVite Louvain classification of a large distributed graph. Includes challenging communication patterns as well as a non-trial amount of computation.

NEKbone Conjugate gradient solver for high-order spectral elements. Emphasizes small dense matrix algebra.

PICSARlite Particle-In-Cell (PIC) kernels from the ECP WarpX project.

SW4lite High-order finite difference stencils on a structured grid.

SWFFT 3D distributed FFTs extracted from the HACC cosmology code.

thornado-mini Finite element, moment-based radiation transport; uses a semi-implicit, discontinuous Galerkin method for a two-moment model of radiation transport.

XSBench Continuous energy cross section lookup from Monte Carlo neutron transport. Features unpredictable memory access patterns into large data tables to stress memory system latency.

Because it is not possible to fully represent all aspects of the DOE or even the ECP workload with a small collection of proxies, the Proxy App Team is also prepared to recommend other proxies from our catalog of more than 60 proxies for specific modeling needs.

Our Proxy App Suite page now also includes a link to a page which lists proxies that are likely to be of significant interest to the community. We have also provided pointers to a set of representative problem inputs and parameters to help investigators run proxy apps with inputs that represent production use cases.

Last, but by no means least, We are continuing our efforts to identify and highlight proxy apps for scientific machine learning (ML) workloads. Version 3.0 of the suite introduced miniGAN, but that was from from sufficient to represent the wide variety of ML use cases. We have now identified enough ML proxies to form a small but growing specialized suite dedicated to this important emerging area. Section 3 provides a complete description of the ML suite.

3 Toward a Machine Learning Proxy Application Suite

Over the last few years we have witnessed explosive growth in the use of machine learning (ML) techniques in practically every field of computing. Applications to problems as diverse as self-driving cars, classifying videos, or online shopping recommendations have been widely publicized. The HPC research community has also been active in seeking ways to use ML to benefit simulation workflows, by improving simulation performance, accelerating workflows, improving model accuracy, or optimizing resource allocation for simulations.

Because of the emerging importance of ML to scientific workflows, the ECP Proxy App Team has recognized the need to collect a suite of proxy apps that represents how ML is applied to problems of interest in DOE. We have observed that ML as applied to DOE science has unique characteristics that are different from finding cat videos or following dotted lines. Capturing these unique requirements in a proxy suite will help represent the ML workload for tasks such as co-design, benchmarking, and algorithmic exploration.

Because ECP is focused on delivering production quality applications that will be ready to run when exascale machines are delivered, there is less of a focus on ML techniques than there is the more forward-looking DOE research community. This means that some if not most of the proxies collected for this suite will be not produced by ECP-funded efforts. We think it is entirely appropriate to reach out into the DOE research community to collect appropriate proxies and help build a suite that will represent how ECP applications may evolve in the post-exascale era. We appreciate the willingness of members of the DOE ML community to work with the ECP Proxy App Team on this effort.

3.1 Principles of the Machine Learning Proxy Suite

Proxy applications are models that capture important features of a large application such as performance critical kernels, programming models, communication patterns, etc. In the case of HPC modeling and simulation applications, this typically translates to simplified codes that are intended to be representative of large production codes. Proxies for ML applications require a somewhat different approach than has been typical for DOE proxy apps. For example, ML problems frequently present a data-centric approach rather than an algorithm-centric one. Therefore, it may be necessary to perform simplification on the data set instead of the algorithm to derive an ML proxy app.

ML proxies are different from traditional proxies in other important ways. One obvious difference stems from the fact that ML workloads often rely on a fairly complex set of third-party dependencies such as PyTorch or TensorFlow. This is contrary to the usual principle of creating simple-to-build proxies by avoiding dependencies. Because such dependencies are practically unavoidable in the ML space, it is fortunate that tools such as Spack are making it much easier to manage dependencies. Another difference comes from the rapid pace of innovation and discovery in ML. Algorithms and methods that are popular today can be easily discarded tomorrow, and the search for better training methods or model representations is a major focus of effort in the field. Hence, good proxies must be nimble to adapt to such changes, participate in the innovation process, and help understand how such changes impact science workflows.

With these differences in mind, our intent is to select ML proxies for the suite based on two sets of criteria. First, the suite must embody the core purposes and uses of proxy apps. Those purposes are:

- Hardware and software co-design;
- Programming model exploration and innovation;

- Development of numerical methods and algorithms;
- Optimization and benchmarking;
- Education.

Second, the suite must cover the breadth of the ML application space. The ML suite will encompass different ML proxy apps chosen to represent the three basic learning paradigms associated with ML:

- Supervised Learning;
- Unsupervised Learning;
- Reinforcement Learning.

The goal of developing the ML proxy suite is to develop proxy apps that represent different paradigms and cover different areas of research in DOE such as classification, regression, detection, etc. Different neural networks associated with these paradigms such as Deep Neural Networks, Generative Adversarial Networks, Variational Autoencoders, etc. will also be included. Along with this, standard data sets for training and inference will be provided.

The following sections describe the proxies in the initial version of the ML suite. We expect this list to grow and evolve as additional proxies appear and as we develop a better understanding of the needs and use cases for ML proxies.

3.2 miniGAN

miniGAN [11] is a Generative Adversarial Network (GAN) proxy application that has been developed as part of ExaLearn and released through the ECP Proxy Application Suite. GANs [9,16,23] are Deep Neural Networks (DNNs) that simultaneously train two models: a generator **G** and a discriminator **D**. As the network trains, **G** produces increasingly accurate synthetic data, while **D** attempts to distinguish the synthetic data from the original training data. Important for miniGAN's use as a proxy application, GANs test a greater variety of layer types and training conditions than standard convolutional or feedforward neural networks.

Relating to specific ECP applications, miniGAN aims to be a proxy application for related machine learning applications in cosmology, such as CosmoFlow [20] and ExaGAN [21], and in wind energy, such as ExaWind [29]. miniGAN models the performance for training generator and discriminator networks. The GAN's generator generates plausible 2D/3D maps, and its discriminator identifies fake maps. miniGAN is built on top of the PyTorch [22] and Horovod [28] packages and has been developed so that optimized mathematical kernels (e.g., kernels provided by Kokkos Kernels or vendor libraries) can be plugged into the proxy application to explore potential performance improvements. A generator is provided to generate a data set (series of images) that are inputs to the proxy application.

GitHub: `github.com/SandiaMLMiniApps/miniGAN`

3.3 miniRL

Reinforcement learning (RL) is a type of machine learning where an agent interacts with an environment with the objective of maximizing a quantifiable notion of cumulative reward. An agent in this context is a software process that provides an action, either explorative or exploitative, to the environment. An environment in this context is usually a simulation that performs the action provided by the agent and returns a new state, a reward, and a status that are together referred to as an experience. At any step, an agent can choose to either explore (sample the action space using various techniques) or exploit (query a trained machine-learned model) in order to better understand the environment's behavior and maximize reward. Any new experiences are incorporated back into the machine-learned model (policy), thus helping the agent get better at the task over time.

miniRL is a reinforcement learning (RL) proxy application derived from the Easily eXtendable Architecture for Reinforcement Learning (EXARL) framework, which is being developed by the ExaLearn Control project. The EXARL framework is designed to be used by researchers interested in using RL for control and optimization of their applications or experiments without worrying about the details of the RL implementations. Any RL problem consists of an agent (controller) and an environment (system to be controlled), and EXARL uses an extension of the OpenAI Gym [7] framework, which not only allows existing benchmark environments in Gym to be used but also provides easy integration of new scientific environments. The agent is nothing but a collection of RL algorithms with a state table or associated neural network architectures. EXARL also includes distributed learning workflows, which define how the agent and environment interact with each other.

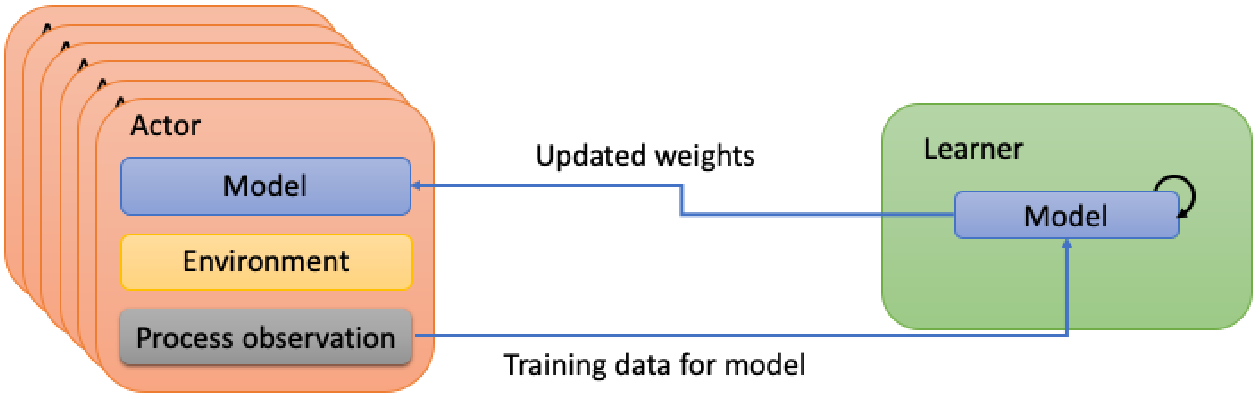


Figure 1: Overview of EXARL architecture. Each actor has a copy of the model, which is updated in every step and used to infer a new action given a state. The actor uses this action to compute the next environment state and runs the Bellman equation to generate updated data. These data are sent back to the learner for training the target model.

The architecture of EXARL is separated into learner and actors, as shown in Figure 1. A simple round-robin scheduling scheme is used to distribute work from the learner to the actors. The learner consists of a target model that is trained using experiences collected by the actors. Each actor consists of a model replica, which receives the updated weights from the learner. This model is used to infer the next action given a state of the environment. The environment can be rendered/simulated to update the state using this action. In contrast to other architectures such as IMPALA [13] and SEED [12], each actor in EXARL independently stores experiences and runs the Bellman equation to generate training data. These training data are sent back to the learner, once enough data is collected. By locally running the Bellman equations in each actor in parallel, the load is equally distributed among all actor processes. The learner distributes work by parallelizing across episodes, and actors request work in a round-robin fashion. Each actor runs all of the steps in an episode to completion before requesting more work from the learner. This process is repeated until the learner gathers experiences from all episodes.

miniRL uses the well known inverted pendulum or 'CartPole' [15] environment along with a DQN agent available in EXARL. In addition, it uses the asynchronous workflow distribution scheme to collect experiences from multiple environments running in parallel. miniRL not only acts a benchmark application for the RL algorithms (agents), but also for different workflow distribution schemes. With the EXARL framework, it is easy to swap different environments, agents, as well as learning and workflow distribution schemes for testing the performance. miniRL also had CANDLE [31] functionality built-in, which allows for hyperparameter optimization using the CANDLE Supervisor.

GitHub: Coming Soon

3.4 CRADL

Concurrent Relaxation through Accelerated Deep Learning (CRADL) performs inference, with a trained machine learning algorithm, on mesh geometry data from hydrodynamics simulations. This application models what the workflow would look like in hydrodynamics codes if manual relaxation strategies were replaced with inline inference. The mini app provides many user options, and optimizations, so they can easily determine the most optimal runtime configuration (batch size, NVIDIA optimizations for PyTorch, parallelization strategy, etc.) for their chosen accelerator. CRADL captures timing for the essential components that will be used in a full-scale application:

Timing is delivered for:

- Loading the machine learning model onto the accelerator;
- Loading mesh quality metrics data onto the accelerator;
- Performing Inference.
  - Includes inference per cycle, and inference per cycle per node.

In addition to this, metrics are given to tell users whether their model's computational cost can be hidden for problems with either simple or complex physics in hydrodynamics codes.

Future versions of the miniapp will incorporate novel accelerators, like the CS-1 at Lawrence Livermore National Laboratory. Options will also be introduced for users to load their own machine learning models and data for profiling.

GitHub: https://github.com/LLNL/CRADL

3.5 Cosmoflow-Benchmark

Cosmoflow-Benchmark is an implementation of CosmoFlow 3D convolutional neural network for benchmarking. It is written in TensorFlow [3] with the Keras API and uses Horovod [28] for distributed training. Deep learning is a promising tool to determine the physical model that describes our universe. To handle the considerable computational cost of this problem, Cosmoflow is implemented as a highly scalable deep learning application built on top of the TensorFlow framework. CosmoFlow uses efficient implementations of 3D convolution and pooling primitives, together with improvements in threading for many element-wise operations, to improve training performance on Intel® Xeon Phi™ processors.

The CosmoFlow project aims to process large 3D cosmology datasets on modern HPC platforms. The specific characteristics are as follows:

- The deep learning network described by Ravanbakhsh et al. [25] is adapted to a scalable architecture for a larger problem size of 1283 voxels, and three cosmological parameters are predicted. Simulations are performed to generate the cosmology dataset with the 3 parameter variations.

- Efficient primitives are implemented in MKL-DNN [2] for 3D convolutional neural networks, which are used in an optimized TensorFlow framework for CPU architectures.
- The Cray Programming Environment's Machine Learning Plugin (CPE ML Plugin) is utilized to efficiently scale deep learning training on supercomputers via MPI.
- The single node and scaling performance improvements enables processing of a large 3D dark matter distribution and prediction of the cosmological parameters Ω^M, σ_8 and n_s with unprecedented accuracy.

Github: https://github.com/sparticlesteve/cosmoflow-benchmark/

3.6 MLperf-DeepCam

MLperf-DeepCam is a PyTorch [22] implementation for the climate segmentation benchmark, based on the Exascale Deep Learning for Climate Analytics codebase (https://github.com/azrael417/ClimDeepLearn).

Climate change poses a major challenge to humanity in the 21st century. Tropical Cyclones have caused the US economy over $200B worth of damage in 2017, and a range of stakeholders are interested in a more careful characterization of the change in number and intensity of such extreme weather patterns in the coming decades. In order to address these important questions, climate scientists routinely configure and run high-fidelity simulations under a range of different climate change scenarios. Each simulation produces tens of terabytes of high-fidelity output that requires automated analysis.

DeepCam uses deep learning (DL) methods to extract high-quality, pixel-level segmentation masks of weather patterns. The following describes the characteristics of the MLperf-DeepCam:

- The state-of-the art Tiramisu [19] and DeepLabv3+ [8] architectures are adapted to solve segmentation problems on high-resolution, multi-variate scientific datasets.
- A number of system-level innovations are featured in data staging, efficient parallel I/O, and optimized networking collectives to enable DL applications to scale to the largest GPU-based HPC systems in the world.
- A number of algorithmic innovations are made to enable DL networks to converge at scale.
- Good scaling on up to 27360 GPUs can be seen, obtaining 999.0 PF/s sustained performance and a parallel efficiency of 90.7% for half precision. The peak performance of 1.13 EF/s is obtained at that concurrency and precision.
- The code is implemented in TensorFlow and Horovod; Therefore, the performance optimizations are broadly applicable to the general deep learning + HPC community.

GitHub: https://github.com/azrael417/mlperf-deepcam/

4 New Proxy Applications: Closing Gaps

One of the missions of the ECP Proxy App Project is to identify gaps in the available proxies relative to the overall ECP and DOE science workload. In connection with version 4.0 of the ECP Suite, we have also examined a number of application areas and made plans to deliver proxies that fill in missing modeling capabilities. This section describes those evaluations, the proxies that have been created, and plans to deliver additional proxies in the future.

4.1 Computational Fluid Dynamics: ECP/FlowApps

The Department of Energy (DOE) supports a wide range of research activities in computational flow physics, which, among other things, includes software development for massively parallel simulations using Eulerian (mesh-based) and Lagrangian (particle-based) methods. After having supported (and advised) researchers, over the years, on matters related to code development and performance improvements on homogeneous computing platforms, we anticipate that the transition to a heterogeneous computing platform will not be smooth for the DOE scientific community. A recurring question, related to code porting, that we are going to have to answer is "How do I port my existing code that runs on a CPU-only platform, to a heterogeneous computing platform?" And, having answered that, there is the next question: "Can you help improve the performance of our code by getting it to utilize the GPUs better?" In an effort to prepare for this transition and educate the DOE science base, we have conducted a survey of existing proxy apps in the flow physics space.

The following mini-apps were built on Summit to assess the functionality that is usable in them, and their suitability to serve as examples for converting codes to heterogeneous architectures.

1. CabanaPIC: a C++/Kokkos mini-app for particle in cell (PIC) computations which has a structured mesh framework that can be used for other mesh-based spatial discretizations.
2. CloverLeaf3D: a Fortran mini-app, which is the 3D extension to the CloverLeaf (2D) mini-app, that solves the compressible Euler equations on a structured mesh.
3. miniAero: a C language based compressible Euler/Navier-Stokes finite-volume solver where the equations are discretized on an unstructured mesh.
4. miniAMR: provides a limited C framework for stencil-based computations, on a structured mesh, for problems that can make use of adaptive mesh refinement.
5. miniFE: a C++-based MPI/CUDA/OpenMP code base for implicit finite-element and finite-volume codes.
6. NEKbone: a Fortran 77 (and C) mini-app that solves the Poisson equation for pressure in the incompressible Navier-Stokes (NS) equations. The NS equations are solved using a continuous Galerkin spectral-element method.

It is important to note that none of the proxies listed above solves a complete flow physics problem that is representative of the capability flow simulations that researchers carry out at the Office of Science and NNSA DOE laboratories. We believe that a proxy that solves a complete flow problem will be much more effective as an example of write and optimized flow codes for heterogeneous architectures. In an effort to prepare for this transition and educate the DOE science base, we propose the development of ECP/FlowApps to show case the application of a discretization technique applied to a canonical flow simulation on a heterogeneous computing platform. The ECP/FlowApp is intended to be the smallest code (with minimal dependencies) that is representative of a target flow physics simulation, with a specific discretization method, and with MPI implementations that additionally use one or more shared-memory parallelization paradigms (e.g., Kokkos, OpenMP 4.5+, DPC++, etc.).

The primary goal of the Flow-Apps that we propose is to demonstrate improved parallel performance, and portability of CFD codes via the use of the Kokkos and OpenMP 4.5+ parallelization paradigms, in conjunction with MPI (i.e., MPI+Kokkos and MPI+OpenMP 4.5+). They also will serve as examples to demonstrate the modifications that are needed to transform existing MPI codes (for homogeneous platforms) to take advantage of shared-memory parallelism on the various DOE exascale platforms. While ideally, a code developer should be able to use/modify these ECP/FlowApps to do their own larger scientific simulations, the aim of developing FlowApps is to also educate the users on how to use one of these Apps that is closest to, say, the discretization they are using, and understand what changes need to be made to transform their own codes to run on a heterogeneous computing platform.

4.1.1 Proposed ECP/FlowApps

Since many flow physics projects aim to do direct numerical simulations (DNS) and/or large eddy simulations (LES) of compressible and incompressible flows, and since the geometry of the computations' domains are simple enough to be created within the solvers, we propose the following ECP/FlowApps for simulating compressible/incompressible turbulent flows:

1. Incompressible Navier-Stokes DNS code for simulating forced isotropic turbulence in a triply periodic box. In many ways, such a code is a driver for a 3D FFT library (such as FFTW). There are quite a few Fortran language based, MPI parallelized, forced isotropic 3D turbulence (FIT3D) codes based on the slab (1D) and pencil (2D) domain decompositions that are required for parallelizing this problem. However, there is no openly available version of FIT3D that can run on GPUs. So, the intent is to create such a FlowApp that can run on heterogeneous computing platforms, with one or more shared-memory parallelization options. There is no existing ECP/mini-app for a pseudo-spectral code that can run on a heterogeneous computing platform. Hence, this Flow-App will be created from scratch. Timeline for completion: March 2021.
2. Incompressible higher-order Navier-Stokes DNS code for simulating wall-bounded flows. This FlowApp is a proxy for open-source higher-order CFD solvers, such as Nek5000, PHASTA (incompressible solver).
   The NEKbone ECP/mini-app that represents the Poisson solver for the pressure equation in an incompressible flow solver, makes use of only MPI. There is no GPU capability in NEKbone, and it cannot run on a heterogeneous computing platform. Hence, this Flow-App will be an incompressible flow solver (for velocity and pressure) that will be based on NEKbone. Timeline for completion: September 2022.
3. Structured-grid Finite-Difference/Finite-Volume Solver for Compressible Flows with Shocks. This FlowApp is a proxy for some export-controlled codes (e.g. FDL3DI from AFRL/NASA), and other non-open-source codes that are used by a variety of researchers for simulating hypersonic flows.
   This FlowApp will be based on CloverLeaf3D. Timeline for Completion: September 2021.
4. Unstructured Finite-Element/Finite-Volume Solver for Compressible Flows with Shocks. This FlowApp is a proxy for the existing open-source codes, PHASTA and SU2.
   We will base this code on miniFE and miniAero. Timeline for Completion: March 2022.

4.2 Proxy Applications for Quantum Chemistry

Quantum chemistry has long been a prime consumer of compute cycles at DOE HPC facilities. Methods such as Hartree-Fock (HF) and density functional theory (DFT) place high demands on floating-point computation while requiring comparatively little in terms of input and output. Thus, quantum chemistry seems inherently suited to exploiting the increased computational power available on new architectures. However, the memory requirements of HF and DFT are quadratic in the problem size and so their replicated data forms are limited to perhaps a few thousand processors before the node memory is exceeded and distributed data forms incur significant communication overheads that limit the parallel efficiency. On the other hand, many-body approaches in which a problem is subdivided into 'fragments', typically along chemical lines, before treating at the desired level of theory, are inherently scalable and have gained popularity for their ability to scale to much larger compute partitions. Examples include the fragment molecular orbital (FMO) method. Nevertheless, FMO can encounter communication and load-balancing issues at the limits of its application. Other methods such as coupled cluster (e.g., CCSD(T)) become memory-bound at far smaller applications than even HF or DFT.

In terms of software, the field of quantum chemistry is dominated by large packages—GAMESS, NWChem, Q-Chem, MolPro, Gaussian, to name a few—with millions of lines of code implementing many methods in complex execution modes. Over the years a wide array of programming models and techniques has arisen in response to the emerging trends in the computing environment. As computers become larger, the gap between the compute performance and the storage (memory, disk) and bandwidth (communication and/or disk speed) capabilities becomes ever wider. Many fields have responded by strategically re-computing certain factors where they would previously have stored them. In quantum chemistry, the so-called "direct" approaches involve re-computing the electron repulsion integrals on demand as they became too numerous to store for applications that are large enough to consume the compute capability. Other common practices include the re-ordering of compute loops and ignoring the exploitation of permutational symmetries to yield even-sized tasks, in order to expose parallelism. Parallel global address space (PGAS) models have also been used to implement distributed data algorithms that exploit the large aggregate memory available with massively parallel computers. PGAS models are especially successful when they can be combined with sophisticated asynchronous messaging capabilities that enable the communication overheads to be overlapped with computation. Again, the large compute load of quantum chemistry is an advantage, behind which sizable communication overheads can be hidden.

Quantum Chemistry packages face further challenges at the exascale. Despite the already complex array of programming models, the next generation of supercomputers will present paradigms such as heterogeneous computing (GPUs) and multi-threading that the community has yet to fully embrace. In this respect, proxy applications would be extremely useful for studying the computational aspects of the applications without the complexity of the entire package. Proxies can also be used to isolate and study algorithmic motifs that can suggest how a given application could be implemented.

While it is clear that a great many such proxies are possible for a field such as quantum chemistry, the current suite has relatively few, and none represent a PGAS model. At present, the ECP Proxy Applications catalog has three proxies that come under the heading of quantum chemistry: miniQMC and the recently added GAMESS-HF-Proxy and GAMESS_RI-MP2_MiniApp. The former is based on QMCPACK, where 'QMC' refers to quantum Monte Carlo—a sub-field of quantum chemistry in which the energy expression is integrated stochastically. Thus, QMC is algorithmically quite different from the mainstream quantum chemistry where the integrals are computed analytically. The latter two proxies are derived from the GAMESS package, as the names suggest, the first of which is briefly discussed below. The second computes the Møller-Plesset second order (MP2) perturbation theory energy using a resolution of the identity (RI)—a type of approximation that needs far less memory than the conventional MP2 algorithm. RI-MP2 theory is used to estimate the electron correlation energy, providing a systematic improvement over the Hartree-Fock model. RI-MP2 is often used in conjunction with methods such as HF and FMO.

To make further headway into this space, we have identified two more proxies to help ensure that the current suite covers ECP workloads.

4.2.1 Basic Hartree-Fock Proxy

For computation, there is general agreement that Hartree-Fock—the entry-level method of mainstream quantum chemistry—provides the most obvious starting point. HF is a central part of most quantum chemistry packages and the starting point for more advanced approaches in molecular orbital theory. HF is algorithmically similar to DFT. The kernel of HF is dominated by the evaluation of electron repulsion integrals (ERI), whose overall cost scales as the fourth power of the problem size. As stated above, the ERI are usually not stored but contracted into the elements of a matrix subsequently input to a generalized eigenproblem. The high cost of ERI has prompted the development of many methods for computing them (for example, Rotated Axis, Rys Polynomial, Obara-Saika, PRIZM, and so on), almost as numerous as the packages themselves. Indeed, the ERI code is often the key distinguishing characteristic of a quantum chemistry package from the software perspective, into which its primary functionality is interfaced. An example is the GAMESS-HF-Proxy mentioned above, which makes use of the LibInt ERI code.

When it comes to assessing computational performance, that of the ERI code can be a complex function of the input molecular system (variables include the degrees of contraction and angular momentum of the Gaussian basis functions, and the interatomic distances). In terms of software, the ERI code itself can be large and, in the context of a proxy, it is debatable whether it constitutes a "minimal dependency". However, concerns over both the compute performance and the software dependencies can be resolved if the HF proxy is based on computing ERI over the basic 's-type' Gaussians for which there is just a single formula. Such an approach provides:

1. A generic starting point that is not tied to any particular quantum chemistry package or ERI code, thus complementing the existing GAMESS-HF-Proxy.
2. Versatility: input cases can be tuned to model different computational loads.
3. Simplicity: the single formula can be easily re-implemented for different programming models, languages, and hardware.

This 'Basic HF Proxy' would quickly allow measurements of floating-point efficiency, GPU speedup, offloading, and other performance data, to inform the community and vendors. Further validations of fidelity can be made by comparison with more general codes and, if necessary, the proxy suite can be further augmented with more representative ERI code(s).

Reduced to its minimal functionality, the Basic-HF-Proxy computes the two-electron term of the so-called Fock matrix from a guessed (or input) density matrix corresponding to a system of atoms whose coordinates are read in at the start. Note that this is the compute kernel of the popular DFT method. Furthermore, the proxy could also be used as the compute kernel of other quantum chemistry proxies, such as that described next.

4.2.2 PGAS-FMO Proxy

A qualitatively different aspect of performance is concerned with the communication overheads, and the overlap of communication with computation, that occur in distributed fragmentation methods such as the FMO implementation in the ECP code, GAMESS. As stated above, FMO is a popular method on supercomputers owing to its inherent scalability/concurrency. However, in its entirety the FMO method is typically highly complex. The implementation in GAMESS has approximately 200 input settings and options. Thus, one aim of a proxy would be to minimize this complexity by reducing FMO to its most basic computation and communication tasks. In this PGAS-FMO proxy, as with the Basic HF proxy, a set of atomic coordinates corresponding to the fragments is input at the start along with a density (or densities) which subsequently populates the PGAS array. Following this, individual compute-processes (or groups) obtain the densities of remote fragments via one-sided messages (GET/PUT) and calculate the contribution to the electron repulsion potential of the corresponding fragment at the desired level of theory. By choosing HF as the level of theory, the Basic HF Proxy can be used as the compute kernel. Since the potential energy for separated monomer fragments, corresponding to the first term of the many-body expansion, is computed, an FMO-1 calculation is performed. Levels higher than 1 successively improve the N-body approximation but are algorithmically similar, the main difference being the (larger) fragment definitions.

In GAMESS, FMO relies on the Distributed Data Interface (DDI)—a Parallel Global Address Space (PGAS) model akin to Global Arrays in NWChem—to implement the PGAS scheme. In the proxy, a minimal library based on the same data-server model as DDI, but independent of DDI, and sufficient just to service the needs of the proxy, is implemented. This minimal PGAS library could be instructive to those considering using such an approach to extend their applications to the high levels of concurrency available at the exascale, as the rest of the proxy could be.

4.2.3 Summary of proxy application current development status and timelines:

1. Basic-HF-Proxy (compute kernel)
   (a) Coding is complete and validated.
   (b) Needs MPI parallelism, GPU offload, and benchmarking data.
   (c) Time to delivery: 3 months.
2. PGAS-FMO-Proxy
   (a) PGAS (parallel) mini-library, coding is done.
   (b) Basic FMO code is complete and tested. The number of fragments per rank/group is currently fixed, needs extending to variable fragments per group.
   (c) Needs offloading, benchmarking.
   (d) Time to delivery: 6 months.

Looking ahead, additional proxies could be developed in the future to cover the workloads of other popular quantum chemistry methods such as coupled cluster (CCSD(T)) and configuration interaction (CI), which both have memory-bound kernels.

4.3 Simulation of Quantum Computing: QTensor-mini

Quantum Computing has attracted significant research interest in DOE as a technology that will help carry us beyond Moore's Law. Because qubits are difficult to build, simulations of the behavior of quantum computers provide an important contribution to the research in this field.

The Quantum Approximate Optimization Algorithm (QAOA) is the most studied quantum optimization algorithm and is considered to be the prime candidate for demonstrating quantum advantage. There is a worldwide race underway amongst top quantum information science researchers to find combinatorial optimization problems, and their instances, that run efficiently and faster on quantum devices rather than on classical computers—the so called *quantum advantage*. A demonstration of this would be a significant achievement in computational science.

The Argonne-developed quantum simulator QTensor is written using a tensor network contraction technique, which is exceptionally well suited for simulating short quantum circuits like QAOA quantum circuits. For example, QTensor is much faster than simulators provided by vendors like IBM and Google, which are using an older state-vector simulation approach. It is worth noting that QTensor aims to be able to perform effectively on a wide array of HPC hardware (especially exascale machines), to best suit the needs of anyone studying QAOA circuits.

We surveyed available proxy apps and found that proxies capable of modeling the memory and time costs of tensor network contractions are lacking. Tensor networks provide an abstract representation of higher order tensors which effectively reduces their often prohibitive memory requirements for storage. However, this reduction in storage cost potentially requires prohibitive costs in time and space for evaluation, depending on the structure of the network representation. We are creating QTensor-mini to fill this gap.

Although QTensor-mini is primarily intended to represent aspects of QTensor, it is important to note that tensor networks have widespread applications in both Scientific Computing and Machine Learning. In Machine Learning and Statistical literature, one example of their use is found in Probabilistic Modeling. Tensor networks are the key to the evaluation of probabilistic graphical models, which are used to reason about probabilistic systems with complex dependency structures. Another example may be found in Constraint Satisfaction literature, where tensor networks may be used as efficient solvers for some CSP instances.

QTensor-mini will help achieve QTensor's goal of widespread success on HPC hardware (especially for upcoming exascale supercomputers Aurora and Frontier). The proxy application enables efficient exploration of the relevant design space, and in addition, serves as a compact, sharable demonstration of tensor network based algorithms. To target both homogeneous and heterogeneous computing platforms, we are developing our proxy app to target both CPU and mixed CPU/GPU platforms. QTensor-mini requires few dependencies, needing only a BLAS library for each of the targeted processors.

QTensor-mini will also be representative of ExaTn, a high performance tensor library in development at Oak Ridge National Lab. The QTensor team is working closely with ExaTn developers to target upcoming exascale supercomputers.

4.3.1 Formulation of the Problem

QTensor represents quantum circuits as tensor networks. As an example, we take a circuit and convert to a graph using a graphical representation of gates. We then contract the graph vertex-by-vertex. Vertex contraction removes a vertex from graph and connects all its neighbors.

The cost of each vertex contraction operation depends exponentially on number of neighbors of the vertex. This number is known in graph theory as the contraction width and the minimum value of it over all possible contraction orders is known to be treewidth + 1.

Our goal is to minimize resource requirements for the contraction of the full circuit to maximize the size of the simulated quantum circuits in terms of both qubits and gates. The simulation is effectively a memory-bound problem, since we have to store large complex-valued tensors that represent intermediate states of the circuit. The memory required to store a tensor with r indexes is $M = 2^{r+4}$.

The simulation consists of many elementary steps, each step dependent on the previous. Each step is a contraction operation of some set of tensors over some index i_0. This operation can be represented as

$$R = \sum_{i_0, i_1, \ldots, i_K} T^1_{i_0 \ldots i_{r_1}} T^2_{i_0 \ldots i_{r_2}} T^m_{i_0 \ldots i_{r_m}} \tag{1}$$

where our set of tensors has m tensors.

The size of resulting tensor R_j at each step varies from a handful of bytes to gigabytes in a single simulation. Computation cost scales as 2^r where r is the rank of a tensor, and implementing advanced optimization techniques provides benefit only for large tensors with rank approximately of $r \geq 20$.

4.3.2 Objectives of QTensor-mini

We identified the most time-consuming and representative steps in QTensor (like pairwise-tensor contraction in the bucket (Eqn. 1). We extracted the corresponding code and packaged it as the proxy app QTensor-mini. Our initial objective was to evaluate the performance and optimize the code for CPUs. This work is done and it will be released in October 2020.

The next step will be to port QTensor-mini to various GPU platforms like NVIDIA GPUs, AMD GPUs, and Intel GPUs with the aim to get ready QTensor working on existing supercomputers and upcoming exascale supercomputers. This work will be done over the course of 2021. In 2021 and 2022, we will introduce refined solvers for determining optimal contraction order in the network.

QTensor-mini is currently under review by the Argonne legal team, and we expect it to be officially released in October of 2020.

QTensor-mini will be available in Github.

4.3.3 Timeline

March 2021: The official release of the proxy app will be in October of 2020. By March 2021, we will implement at least two backends to run on existing GPU platforms (NVIDIA and AMD). The GPU backends in consideration will include CuBLAS, ArrayFire, and, further down the line, Intel's in-beta OneAPI. We will also optimize and further benchmark existing backends.

September 2021: We will develop the backend for Intel GPU (Xeon Xe). Our prime candidate is MKL version using OneAPI interface. Once XLA support arrives for Intel GPUs, we will create further backends using popular tensor packages such as JAX and PyTorch.

4.4 Molecular Dynamics and Biological Applications

Classical Molecular Dynamics (MD) is an important component of the DOE workload, and several proxy apps have been created to represent common MD codes. However, all of the the most widely used MD proxies including CoMD, miniMD, and exaMiniMD are based on use cases drawn from Condensed Matter Physics and Material Science. In contrast, use cases from Biology are not as well represented in the proxy app space.

Because ExaMiniMD is a member of the ECP Proxy App Suite, and because most MD proxies are similar in design, we made exaMiniMD the focus of our evaluation. ExaMiniMD is designed to represent major computational features of all-atom MD simulation, that is the floating-point intensive pairwise atom-atom nonbonded interactions. When applied to biological problems, MD typically requires bonded interactions and long range interactions and has tended to employ force/energy lookup tables. We decided to evaluate whether it would be beneficial to add these additional features to exaMiniMD to better support modeling of biological MD workloads.

Bonded interactions are general features of both material science and biological MD simulations. Although typically they affect overall performance by only a few percent and therefore not an HPC concern, including these terms presents a more realistic picture of biological modeling. To investigate how bonded force calculation terms affect the overall ExaMiniMD code structure, size, and performance we recycled the bonded force classes of LAMMPS to create a version of exaMiniMD that included both particle domain decomposition and MPI.

As the work to add bonded terms to ExaMiniMD progressed, we soon found that the code became unnecessarily complex in terms of both code size and inadvertent influence on the existing nonbonded force calculation. Generally bonded terms include bonds, angles, dihedrals and improper dihedrals. The corresponding neighbor list construction involves numerous array/pointer operations and generates complex atom connectivity. Moreover, due to the complex atom-atom exclusion relationships generated by bonded lists, the existing nonbonded force calculations have to be modified significantly. Without bonded terms, the periodic neighbor-list construction of nonbonded calculations only depends on atom-atom distance and therefore is programmed in a few simple arrays/pointers; however, the 1-2, 1-3, 1-4 and modified 1-4 exclusions due to bonded terms generate multiple branches in nonbonded neighbor-list construction and involve large number of array/pointer operations. Extra filters of exclusion have to be applied to the original simple atom distance evaluation. These atom-atom exclusions also are force field dependent, further complicating their implementation in ExaMiniMD.

It soon became evident that our prototype implementation of bonded terms threatened to double the size of ExaMiniMD. This significant increase of code size contradicts the original design purpose of a proxy app—code compactness and focus on the major performance concerns. Taking into account the significantly expanded code size and complexity and the modest contribution of bonded terms to total run time, we decided to abandon the effort.

Long-range forces in MD have typically been calculated using the Particle Mesh Ewald (PME) or Particle-Particle-Particle-Mesh (P^3M) algorithm. These algorithms involve many-to-many communications and intensive floating-point operations. They also require an optimized FFTW library and efficient SIMD/SIMT implementations. However, such long-range terms account for only about $\sim 5\%$ of run time. Moreover, in recent years there have been several scalable algorithms designed to replace PME/P^3M and further reduce the cost of long-range computations. Many MD codes designed for biological applications like NAMD, Gromacs, etc., have adopted these novel but economic algorithms to improve code speed. Based on these considerations, as well as our negative experience with bonded terms, we elected not to pursue adding long-range terms to exaMiniMD.

Finally, we evaluated the prospects to add a force/energy lookup table to exaMiniMD. Based on discussions with several leading MD code developers, we determined that there has been a trend toward utilizing the SIMD/SIMT programming to directly compute nonbonded force/energy instead of relying on tables. Once again, it appears that there is no need to add features to exaMiniMD.

Overall, we have determined that while most MD proxies do not explicitly contain terms and algorithms that are needed for biological applications, the current ExaMiniMD strikes a good balance between representing the major performance concerns while maintaining compact code. Adding capabilities to exaMiniMD is likely to expand code size and/or increase complexity while offering only minor or negligible performance implications for future architecture study.

4.5 Monte Carlo Particle Transport for Reactor Analysis

The use of Monte Carlo (MC) methods for the design and analysis of nuclear reactors has become increasingly promising in recent years. MC methods offer an extremely accurate approach to reactor simulation since they can represent the geometry and physical phenomena nearly exactly; however, they come with significant computational challenges.

In the ECP ExaSMR project, the goal is to carry out coupled MC neutronics and CFD simulations of a 3D full small modular reactor (SMR) core. Two MC neutron transport applications are used in the project, Shift and OpenMC. There are several proxy apps that model important characteristics of the full MC transport applications in the context of reactor analysis. These proxies mainly represent two computational challenges: the calculation of the macroscopic cross section, and the random branching control flow of the applications.

4.5.1 Continuous-Energy Cross-Section Lookups

One of the key performance bottlenecks in Monte Carlo reactor simulation is random memory accesses into large read-only tables of tabulated cross-section values. These cross sections determine the probabilities of various interactions and are sampled at each step in a particle's trajectory to determine the next event that the particle will undergo. Cross-section data is needed for each nuclide in the problem. So-called continuous-energy cross sections are tabulated at a huge number of energy points to resolve the detailed structure of resonances.

One of the characteristics of reactor transport is a large nuclide inventory. When the fuel depletes, hundreds of nuclides build up in the fuel region. Continuous-energy cross sections for all nuclides at a single temperature can require a few gigabytes of memory. During the simulation, accesses to this data are extremely frequent (every time a particle changes energy or moves to a new material) and essentially random. Because of this, reactor simulation becomes a memory latency bound problem. Because of the importance of cross-section lookups in MC simulations, multiple proxy apps have been developed to represent the lookups.

XSBench XSBench represents continuous-energy cross-section lookups but ignores particle tracking. Even though XSBench uses synthetic cross-section data and generates energies and materials randomly, the data access patterns faithfully recreate those found in production applications, and the multicore scaling efficiency and floating-point calculation rates are very similar. For Shift and OpenMC, XSBench is a highly representative proxy since it successfully models the most computationally expensive kernel of the full simulation.

RSBench RSBench represents the multipole method of calculating continuous-energy cross sections. This method has a much lower memory footprint, only requiring megabytes of data (instead of gigabytes), and improved data locality. However, memory savings come at the cost of significantly increased floating-point computation. Because multipole cross sections are more compute intensive and less memory intensive, they are potentially promising for next-generation architectures.

With both RSBench and XSBench, the macroscopic cross-section calculation, the key bottleneck of Monte Carlo reactor simulation, is very well represented.

EBMS EBMS is another proxy in the ECP extended catalog that addresses the challenge of doing frequent random memory accesses from large cross-section tables. It is a prototype that implements a proposed algorithm for parallel Monte Carlo neutron transport codes that would significantly reduce the on-node footprint of cross-section memory. Rather than replicating the large cross-section tables on every node, a memory-server model is employed in which the data resides on a remote set of disjoint processors. The cross-section data is partitioned into energy bands and distributed across nodes. Taking advantage of the fact that most neutrons move from high to low energies during their lifetime (with only occasional upscattering), a new tracking algorithm is adopted in which each processor starts by loading the highest energy band and tracking each particle until it leaves that band. The next-highest band is then loaded, and the tracking proceeds in this way until all the particles have been absorbed. While this algorithm demonstrates a possible path forward for these memory-bound codes, it has not actually been implemented in any of the full applications, and there are not currently any efforts to do so.

4.5.2 Particle Tracking

Another challenge to getting good Monte Carlo performance on next-generation systems is the complex branching execution path associated with tracking the particles. MC neutronics codes have traditionally used a 'history-based' approach where each particle is tracked through randomly selected interactions until it is absorbed or escapes from the system. Because individual neutrons do not interact with each other, the algorithm is straightforward to parallelize. Each thread tracks a particle from birth to death as it undergoes a series of collisions and surface crossings until it is eventually killed. Unfortunately, this fundamentally MIMD approach doesn't map well to SIMT parallelism in GPUs because of the high branching and high latency.

'Event-based' methods are an alternative to the history-based approach which attempt to exploit the vectorization capabilities of the GPU. Instead of tracking each particle from beginning to end, particles experiencing the same event type are processed together. Event-based processing helps avoid thread divergence on the GPU, but suffers from higher overheads associated with sorting particles by event type. While the history-based approach typically leads to a single large GPU kernel, an event-based approach will have multiple specialized kernels with less complexity which may be able to achieve higher occupancy. Most of the recent work on adapting MC transport to GPUs has focused on event-based algorithms.

At least two proxies have been designed to capture the branching control flow of MC applications, with the main motivation being to explore different tracking algorithms on the GPU. Both of these proxies employ multigroup cross sections to reduce the memory footprint and computational cost of cross-section lookups.

ProfugusMC ProfugusMC is a multigroup Monte Carlo solver that was designed to mimic the algorithms and design of Shift. It uses a limited geometry and small number of tallies, and runs on one predetermined transport problem.

ProfugusMC has been used to explore the tradeoffs of history-based and event-based approaches on the GPU. With this proxy, the best performance on the GPU is achieved using a modified history-based method where Particle histories are truncated after a fixed number of collisions and the surviving particles are consolidated and launched in a new kernel. Curiously, performance in Shift is quite different. With continuous-energy cross sections the event-based algorithm far outperformed a history-based approach. This indicates that while ProfugusMC is very useful for exploring algorithmic details, not all the characteristics carry over from multigroup to continuous-energy, and the performance issues can be very different between the proxy and the full application.

Quicksilver Quicksilver is a proxy intended to approximate the performance of the production code Mercury. It was created to help inform refactoring for GPUs and to allow for collaboration since Mercury is export-controlled. Like ProfugusMC, it aims to capture the random branching nature of the control flow. Quicksilver uses multigroup cross sections, has simplified physics, geometry and meshing, and uses a reduced number of tallies. Unlike ProfugusMC, Quicksilver does capture Mercury's domain decomposition, using MPI for inter-node communication. It also allows for flexible inputs. Quicksilver takes a history-based approach for running on the GPU, where the entire tracking loop is converted into a 'big' GPU kernel.

4.5.3 Possible Gaps

One challenge that is not addressed by existing proxy apps is the large memory footprint of tallies. Tallies are running sums of events such as collisions or particle tracks that are used to compute physical quantities of interest such as the flux or reaction rates. Design calculations for a reactor core can involve 200–300 fuel pins in a typical fuel assembly and around 200 assemblies in a reactor core. with 100 axial and 10, several reaction rates and hundreds of nuclides in the fuel, tallies can reach hundreds of gigabytes or even terabytes of memory.

One possible compromise when dealing with the limited on-node memory when tallying is the use of NVRAM to store tally data. Unlike cross-section data, tally data is not needed for determining the random walk of the particle at each step, so it could be stored remotely. It could be worth investigating the feasibility of using NVRAM to enable design calculations for reactor cores while avoiding domain decomposition.

4.5.4 Progress and Next Steps

Currently, there are no proxy apps that isolate the tallying component of a Monte Carlo transport reactor simulation. It could be useful to have a simple tally proxy to explore problems that arise when handling the large, fine-grained tallies that are required for nuclear reactor design and analysis. Such a proxy could help to assess whether it is possible to use NVRAM for tallies or give early insight to potential performance bottlenecks such as the tally synchronization among nodes at the end of each cycle.

A simple code to mimic the tallying component of Monte Carlo transport is being implemented. The goal is to extract the basic tally loop and capture the volume and frequency of tally data written. The relevant tally bins are updated in a manner similar to the full application. Further work and verification ongoing to ensure the performance characteristics of the full application are captured. The current version is CPU-only, but a future version using OpenMP 4.5 for GPU offload is planned. The time line for completing this next version of the code is May 2021.

An analysis of the tally write pattern for detailed full-core reactor models has been performed using the OpenMC Monte Carlo particle transport code to gain an understanding of the volume and frequency of the data written. The distribution of the number of collisions and tracks per particle in both fuel and non-fuel materials has been studied in order to quantify the number of times each particle scores to a tally, and the amount of data written at each tallying event has been calculated for the physical quantities that would typically need to be tallied in this type of simulation. By characterizing the tally write pattern, we can try to capture it outside the full code in a simple way. We can also use this information to gauge whether the use of NVRAM is a viable option given the requirements of tallying in these calculations.

The next step in exploring the potential of NVRAM would be to do some simple testing of the write performance on NVRAM. Following this, we would perform some write benchmarking in the context of the highly simplified MC tally proxy, using data sizes and patterns that approximate those of a real simulation. By comparing this to the equivalent on RAM, or to the simulation time for other parts of the code, we can decide whether it is worthwhile to pursue this idea further. The time line for completion for this work is February 2021.

4.6 Density Functional Theory: minq

Computational material science is key for the understanding and discovery of materials in a large number of energy applications, from the development of batteries and solar cells, to characterizing matter under extreme conditions. Because of this, the simulation of materials is one of the main applications in supercomputing workloads.

When accurate simulations of materials are required, it is necessary to solve the Schrödinger equation of Quantum Mechanics under some level of approximation. The most popular of such approximations is density functional theory (DFT) [10] due to its moderate computational cost and reasonable accuracy, in comparison with other methods like Quantum Monte Carlo or many-body perturbation theory. This means that each year, a significant fraction of the DOE supercomputing time is spent in DFT codes like VASP, Quantum Espresso, or Gaussian.

Despite its importance, DFT codes have been slow to develop support for modern GPU architectures. GPU implementations of DFT are scarce, usually not much faster than their CPU counterparts [18, 26], and normally are limited to a single node or small number of GPUs [5, 30]. This is explained in part by the complexity of DFT, but also to the lack of library support. DFT codes rely on parallel linear algebra routines to run efficiently in parallel, a role that ScaLAPACK has provided quite efficiently for CPUs [17]. For GPUs, SLATE [14], under development at the University of Tennessee and sponsored by the Department of Energy, promises to fulfill that role. However SLATE is in the development phase and offers only basic implementations of the operations required for DFT.

In order to study how SLATE works for DFT, we have developed minq. Minq is a proxy app to model the computationally intensive parts of a DFT code based on SLATE. In particular, it implements the linear algebra operations that usually dominate the computational time for large systems. These same operations are usually the bottleneck when DFT is parallelized over a large number of processors. The objective is to provide a simple code, with a simple interface, that can serve as a sample of what are the matrix sizes and operations that are relevant for DFT to library and hardware developers.

Minq is based on the inq DFT code currently developed at LLNL, which is based on the experience gained in the development of the Octopus [6] and Qball [27] codes. As such, the operations that minq implements are quite universal and appear in most plane-wave or real-space DFT codes. While based on SLATE, minq is easily adaptable to other libraries, and it is the objective to extend it to other linear algebra implementations. We plan to include support for the ELSI [32] library that offers several alternatives for some parallel linear algebra operations.

4.6.1 Usage and problem sizes

Minq is designed for simplicity of use: it only needs one parameter to run, the number of atoms N_A, and it does not need any external data. N_A is the significant value used in the community to describe the scale of a calculation. The actual simulation parameters, the number of wave-functions N_W and the size of the basis set N_B are derived by minq from N_A based on a typical material (crystalline silicon).

The computational cost of minq is $\mathcal{O}(N_\mathrm{A}^3)$ and the memory one is $\mathcal{O}(N_\mathrm{A}^2)$. The typical number of atoms for DFT codes range between 50 atoms for small calculations, to around 10,000 atoms for very large hero-type calculations.

4.6.2 Design and algorithms

The objective of a DFT simulation is to calculate a set of wave-functions that describes the electrons in a material. Each wave-function is represented by a set of coefficients on a given basis set; this basis set is usually a set of plane waves, but other basis sets are used by some codes. The wave-functions are obtained by calculating the first N_{W} eigenvectors of the Hamiltonian operator H. Minq includes the orthogonalization of the wave-functions and the subspace diagonalization of the Hamiltonian. The cost of these operations scales as $\mathcal{O}(N^3)$, so they dominate for large systems. At present minq does not include the fast Fourier transforms (FFTs) that appear in plane-wave-based DFT, but this might be implemented in the future.

The orthogonalization and diagonalization algorithms required for DFT are traditionally provided by BLAS and LAPACK. Minq implements these procedures calling SLATE. A particularity of the implementation has to do with how the matrices are distributed. In the standard parallelization strategy for DFT [4], the wave-function matrix is distributed in blocks using a 2D decomposition. Usually ScaLAPACK and SLATE work with several blocks of the matrix per processor. In DFT, however, only one block is assigned per processor since this is the most efficient approach for the FFT operations that require contiguous data. One of the objectives of developing minq is to understand how much of this limitation affects performance.

Minq is a new proxy app to represent the data decomposition and linear algebra in plane-wave DFT codes. It is designed for simplicity of development and use. As such we expect it to be a valuable tool to help library and hardware developers to understand the nature of workloads that appear in quantum material simulations.

GitHub: `https://github.com/LLNL/minq`

4.7 Skeletal Proxy Apps for I/O and Communication

Many proxy apps are designed to model the computation phases of an application. However, the I/O and communication phases are becoming increasingly performance-critical as increases in compute performance continue to outstrip improvements in network and I/O bandwidth.

To facilitate modeling of communication and I/O, we have been working on Skel-Proxy, a set of tools for automatic generation of proxy apps that target I/O and communications of AI workloads. This toolset is based on Skel, which has been widely used for generating I/O benchmarks and other codes.

The Proxy App project has focused on modeling applications' I/O and communication behaviors by capturing traces of POSIX I/O and MPI communication calls. We collaborated with members of the Tau team to adapt the previously existing Tau tracing mechanism to provide sufficient information to allow modeling of I/O and communication behavior. The models are then used to instantiate Skel templates to produce skeletal proxy applications that mimic the modeled behaviors of the original application. This process is shown schematically in Figure 2.

The proxy applications generated by Skel-Proxy consist of generated C++ files, along with a tailored build mechanism that uses CMake to support building and installing the proxy application on other platforms without significant user intervention. This results in a realistic code that exercises the I/O and communication capabilities of a target platform without a need to build the original application on that platform, and without most of the library dependencies that the original application would have required.

The main goal of this work is to create the capability to quickly create proxies for arbitrary applications that capture their I/O and communication behaviors. These proxies will provide a means to test new hardware with realistic workflows, support a range of research questions while

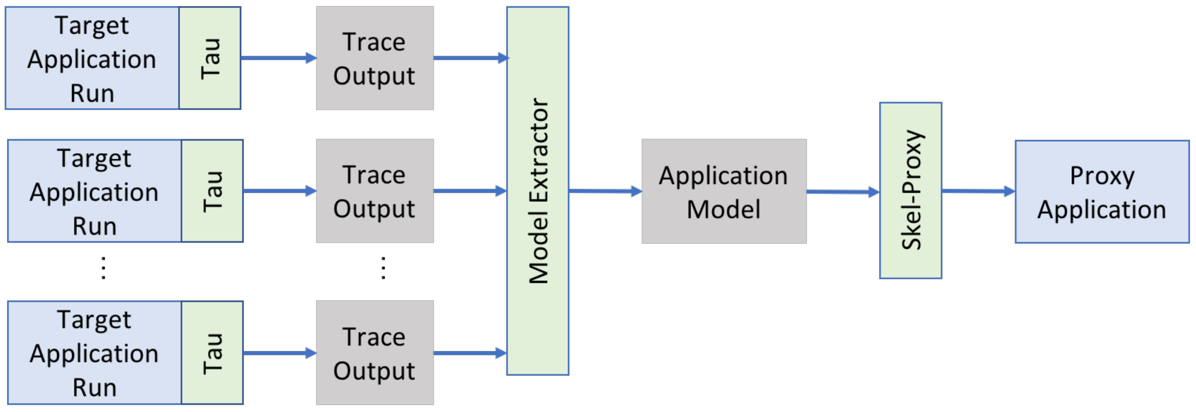


Figure 2: The process of generating a skeletal proxy application.

minimizing complexity (as the proxies have few library dependencies and a simple, consistent build mechanism), and promote wider understanding of codes that cannot be widely released.

4.7.1 Completed activities

The I/O proxy application generation work began in January 2020. Since then, we have focused on the I/O behavior of applications, and so far we have accomplished the following:

- Define a schema for the I/O models
- Collaborate with the Tau project to obtain needed trace information in a usable format
- Create a set of filters to flexibly extract I/O models from Tau traces
- Create a set of templates designed to produce the set of files that makes up a complete proxy application (including source, CMake, and the synthetic files needed to support read behavior).

4.7.2 Current and future work

As we complete and refine the Skel-Proxy toolchain, we are beginning to turn toward gathering a collection of AI applications with both typical and extreme I/O and communication behaviors, in order to create and release a suite of living proxies that target a changing set of interesting applications. We have identified one application, MENNDL, to use for the initial evaluation of these tools. We are working with the MENNDL team to obtain traces, and will work toward the release of this first generated proxy application, which is planned for November, 2020. We will continue to add other proxies to the suite during FY2021, aiming for 5-10 AI/Deep Learning proxies, while refining Skel-Proxy to support a broader set of cases. This will put us on track for a release of the Skel-Proxy tools in August 2021, which will make it easy for users to build their own I/O skeletons as proxies for a wide range of applications.

5 Continuous Integration

Continuous Integration (CI) testing has proven useful for many open- and closed-source projects and has become a standard good practice for codes ranging from small projects to operating systems developed by Fortune 500 companies, all the way to tightly-coupled codes supported by multiple DOE laboratories.

Implementing CI within ECP is a significant challenge due to the need to test against a variety of hardware and environments across multiple laboratories. As the Exascale Computing Project (ECP) is meant to prepare application codes to run on a recently solidified target representing the exascale High-Performance Computing (HPC) platforms, the proxy apps have been vital to determine the viability of programming environments, and hardware and software tool chains. This has manifested as a need to run on a wide range of Linux-based operating systems with a range of compilers and libraries. Additionally, it is of vital importance to ensure that the various build systems for these support the slightly different HPC environments on each machine at different laboratories.

5.1 ECP CI Project

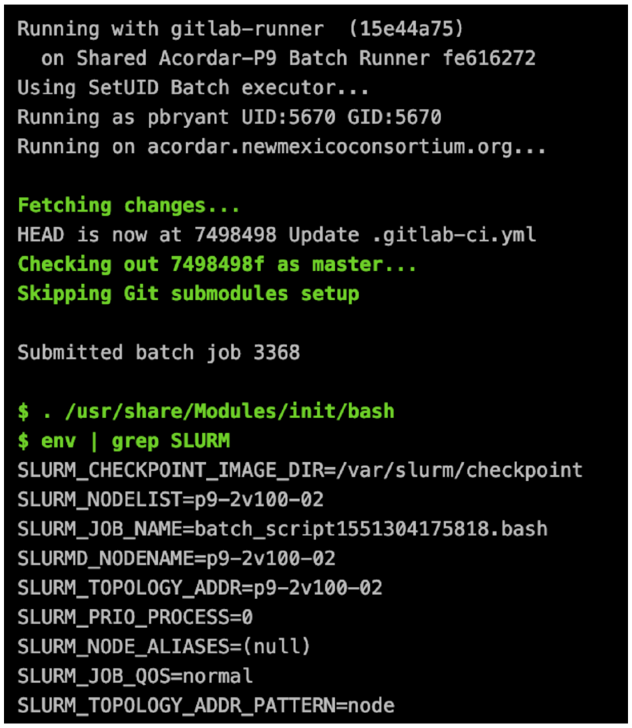
Recent developments in the ECP CI project have helped make it possible to test proxy apps on more platforms. While the ECP CI project had many goals, the two most pertinent to the ECP Proxy Apps project were the batch runner and general efforts to provide shared infrastructure between laboratories.

The batch runner and its associated SetUID capabilities were vital from a security and reproducibility standpoint. Prior to these efforts, all CI jobs required specialized containers or dedicated resources for execution. The batch runner provides the ability to run CI jobs via the actual job scheduler (batch runner) on HPC platforms, and the SetUID capabilities ensure that jobs are executed using the developer's account that pushed changes to the repository. These combined allow for CI jobs to run on the actual hardware and platforms of interest as the developer with authorization. A simple example of a batch runner configuration can be seen in Figure 3.

Similarly, the shared infrastructure between labs has been vital. While the workflows are still a work in progress, this allows developers at one site to run their tests on almost all sites' platforms in a mostly automatic way.

5.2 Proxy Apps Team Methodology

We started with build tests for most of the proxy apps in our catalog using Travis CI. We need to ensure the apps actually build on standard Linux distributions before testing on non-standard hardware. Due to their popularity, we picked Ubuntu and Fedora as baselines. As we recommend Spack as the mechanism of choice to deploy proxy apps, we based the CI on Spack as well. While this limits us to testing proxies that actually have Spack packages, it has the advantage that it enables testing of Spack and the proxies' dependent packages as well. We don't test all apps of the catalog as some apps, such as the CANDLE benchmarks, have missing dependencies in Spack. Where possible, the ECP Proxy Apps team reaches out to the application teams to improve spackages, but due to limited resources, triage is often necessary.



```
Running with gitlab-runner  (15e44a75)
  on Shared Acordar-P9 Batch Runner fe616272
Using SetUID Batch executor...
Running as pbryant UID:5670 GID:5670
Running on acordar.newmexicoconsortium.org...

Fetching changes...
HEAD is now at 7498498 Update .gitlab-ci.yml
Checking out 7498498f as master...
Skipping Git submodules setup

Submitted batch job 3368

$ . /usr/share/Modules/init/bash
$ env | grep SLURM
SLURM_CHECKPOINT_IMAGE_DIR=/var/slurm/checkpoint
SLURM_NODELIST=p9-2v100-02
SLURM_JOB_NAME=batch_script1551304175818.bash
SLURMD_NODENAME=p9-2v100-02
SLURM_TOPOLOGY_ADDR=p9-2v100-02
SLURM_PRIO_PROCESS=0
SLURM_NODE_ALIASES=(null)
SLURM_JOB_QOS=normal
SLURM_TOPOLOGY_ADDR_PATTERN=node
```

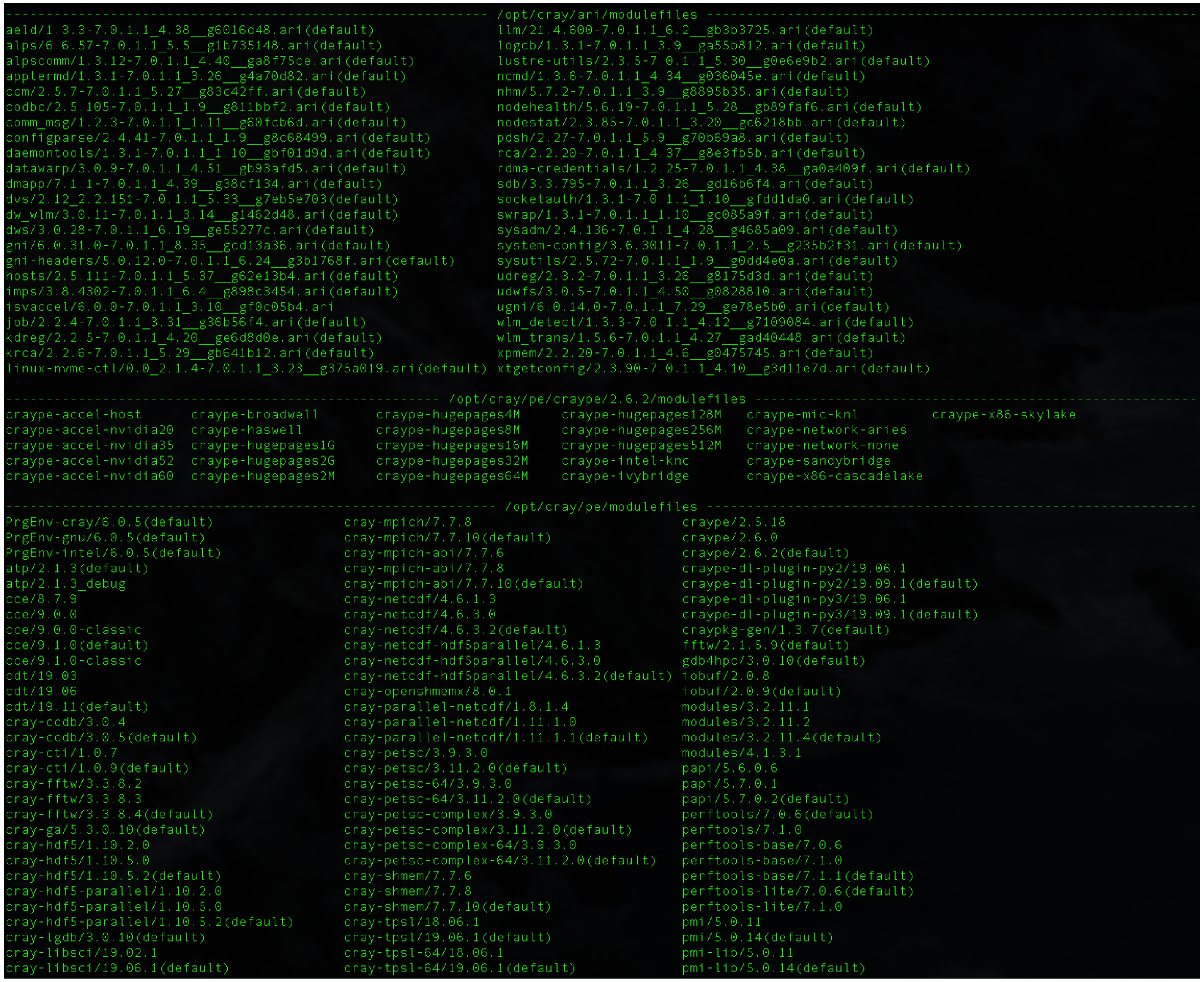
Figure 3: Example of Batch Runner Configuration [1]

5.3 Lessons Learned

5.3.1 Tool Chain Issues with Proxy Apps

In the initial phase, we focused on fixing minor and major issues in the build systems and spackages of different proxy apps. During this process, we found a number of minor issues in the build systems and spackages of different proxy apps. Most of these issues were mainly due to hard-coded flags and ambiguous dependency declarations. By working with the Spack team and the proxy apps developers we were able to resolve most of these issues.

This hardcoding of paths and requirements is problematic as different sites and platforms have vastly different environments. As an example, NERSC's Cori is heavily influenced by its use of Cray hardware and software, as seen in Figure 4, whereas Oak Ridge's Summit has a module system more heavily influenced by the underlying IBM Power9 nodes, as in Figure 5. Reconciling these differences was primarily accomplished by using Spack to provide a somewhat common environment.

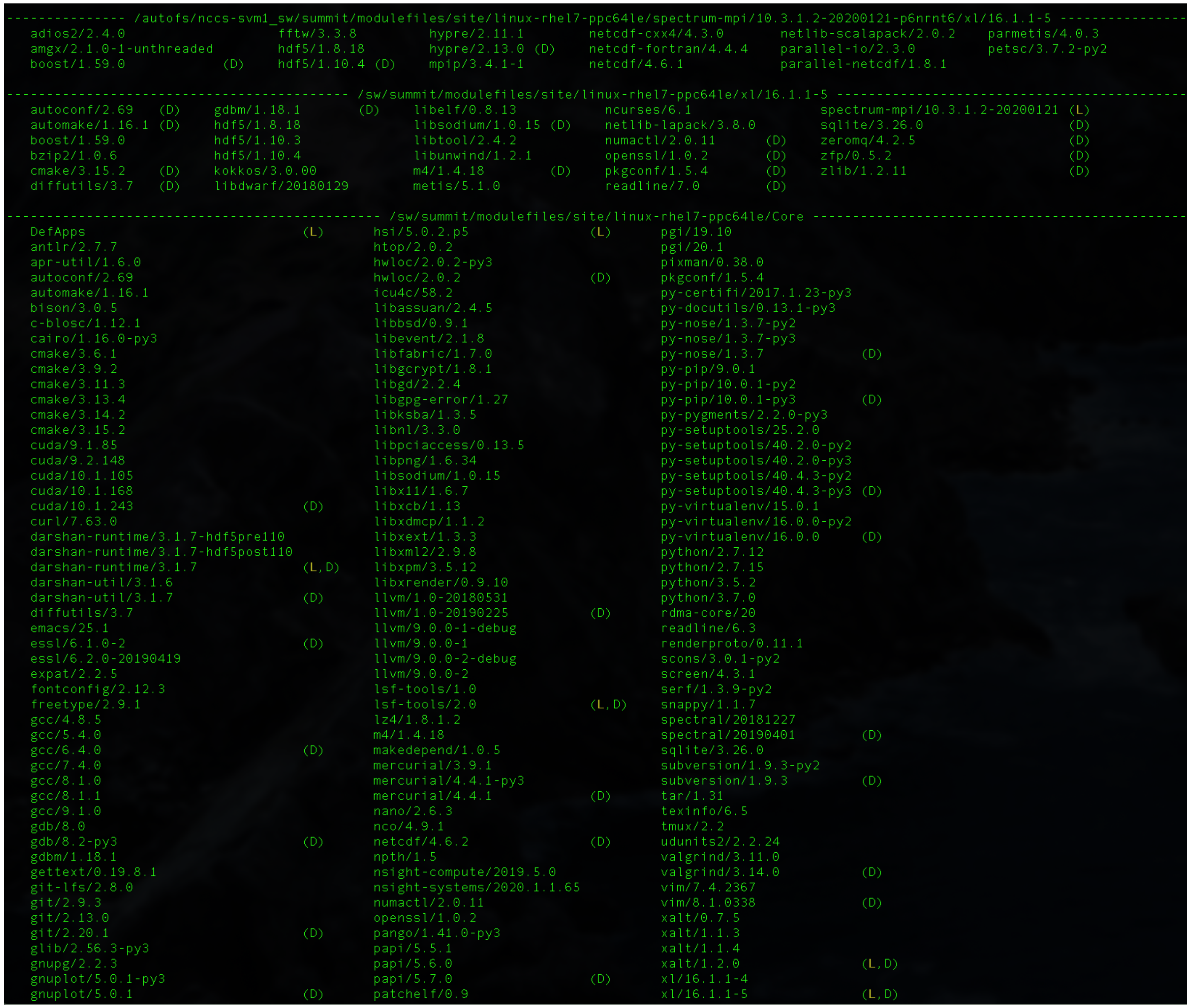


```
------------------------------------------------------------ /opt/cray/ari/modulefiles ------------------------------------------------------------
aeld/1.3.3-7.0.1.1_4.38__g6016d48.ari(default)               llm/21.4.600-7.0.1.1_6.2__gb3b3725.ari(default)
alps/6.6.57-7.0.1.1_5.5__g1b735148.ari(default)              logcb/1.3.1-7.0.1.1_3.9__ga55b812.ari(default)
alpscomm/1.3.12-7.0.1.1_4.40__ga8f75ce.ari(default)          lustre-utils/2.3.5-7.0.1.1_5.30__g0e6e9b2.ari(default)
apptermd/1.3.1-7.0.1.1_3.26__g4a70d82.ari(default)           ncmd/1.3.6-7.0.1.1_4.34__g036045e.ari(default)
ccm/2.5.7-7.0.1.1_5.27__g83c42ff.ari(default)                nhm/5.7.2-7.0.1.1_3.9__g8895b35.ari(default)
codbc/2.5.105-7.0.1.1_1.9__g811bbf2.ari(default)             nodehealth/5.6.19-7.0.1.1_5.28__gb89faf6.ari(default)
comm_msg/1.2.3-7.0.1.1_1.11__g60fcb6d.ari(default)           nodestat/2.3.85-7.0.1.1_3.20__gc6218bb.ari(default)
configparse/2.4.41-7.0.1.1_1.9__g8c68499.ari(default)        pdsh/2.27-7.0.1.1_5.9__g70b69a8.ari(default)
daemontools/1.3.1-7.0.1.1_1.10__gbf01d9d.ari(default)        rca/2.2.20-7.0.1.1_4.37__g8e3fb5b.ari(default)
datawarp/3.0.9-7.0.1.1_4.51__gb93afd5.ari(default)           rdma-credentials/1.2.25-7.0.1.1_4.38__ga0a409f.ari(default)
dmapp/7.1.1-7.0.1.1_4.39__g38cf134.ari(default)              sdb/3.3.795-7.0.1.1_3.26__gd16b6f4.ari(default)
dvs/2.12_2.2.151-7.0.1.1_5.33__g7eb5e703(default)            socketauth/1.3.1-7.0.1.1_1.10__gfdd1da0.ari(default)
dw_wlm/3.0.11-7.0.1.1_3.14__g1462d48.ari(default)            swrap/1.3.1-7.0.1.1_1.10__gc085a9f.ari(default)
dws/3.0.28-7.0.1.1_6.19__ge55277c.ari(default)               sysadm/2.4.136-7.0.1.1_4.28__g4685a09.ari(default)
gni/6.0.31.0-7.0.1.1_8.35__gcd13a36.ari(default)             system-config/3.6.3011-7.0.1.1_2.5__g235b2f31.ari(default)
gni-headers/5.0.12.0-7.0.1.1_6.24__g3b1768f.ari(default)     sysutils/2.5.72-7.0.1.1_1.9__g0dd4e0a.ari(default)
hosts/2.5.111-7.0.1.1_5.37__g62e13b4.ari(default)            udreg/2.3.2-7.0.1.1_3.26__g8175d3d.ari(default)
imps/3.8.4302-7.0.1.1_6.4__g898c3454.ari(default)            udwfs/3.0.5-7.0.1.1_4.50__g0828810.ari(default)
isvaccel/6.0.0-7.0.1.1_3.10__gf0c05b4.ari                    ugni/6.0.14.0-7.0.1.1_7.29__ge78e5b0.ari(default)
job/2.2.4-7.0.1.1_3.31__g36b56f4.ari(default)                wlm_detect/1.3.3-7.0.1.1_4.12__g7109084.ari(default)
kdreg/2.2.5-7.0.1.1_4.20__ge6d8d0e.ari(default)              wlm_trans/1.5.6-7.0.1.1_4.27__gad40448.ari(default)
krca/2.2.6-7.0.1.1_5.29__gb641b12.ari(default)               xpmem/2.2.20-7.0.1.1_4.6__g0475745.ari(default)
linux-nvme-ctl/0.0_2.1.4-7.0.1.1_3.23__g375a019.ari(default) xtgetconfig/2.3.90-7.0.1.1_4.10__g3d11e7d.ari(default)

------------------------------------------------------ /opt/cray/pe/craype/2.6.2/modulefiles ------------------------------------------------------
craype-accel-host      craype-broadwell       craype-hugepages4M     craype-hugepages128M   craype-mic-knl         craype-x86-skylake
craype-accel-nvidia20  craype-haswell         craype-hugepages8M     craype-hugepages256M   craype-network-aries
craype-accel-nvidia35  craype-hugepages1G     craype-hugepages16M    craype-hugepages512M   craype-network-none
craype-accel-nvidia52  craype-hugepages2G     craype-hugepages32M    craype-intel-knc       craype-sandybridge
craype-accel-nvidia60  craype-hugepages2M     craype-hugepages64M    craype-ivybridge       craype-x86-cascadelake

------------------------------------------------------------- /opt/cray/pe/modulefiles -------------------------------------------------------------
PrgEnv-cray/6.0.5(default)                cray-mpich/7.7.8                          craype/2.5.18
PrgEnv-gnu/6.0.5(default)                 cray-mpich/7.7.10(default)                craype/2.6.0
PrgEnv-intel/6.0.5(default)               cray-mpich-abi/7.7.6                      craype/2.6.2(default)
atp/2.1.3(default)                        cray-mpich-abi/7.7.8                      craype-dl-plugin-py2/19.06.1
atp/2.1.3_debug                           cray-mpich-abi/7.7.10(default)            craype-dl-plugin-py2/19.09.1(default)
cce/8.7.9                                 cray-netcdf/4.6.1.3                       craype-dl-plugin-py3/19.06.1
cce/9.0.0                                 cray-netcdf/4.6.3.0                       craype-dl-plugin-py3/19.09.1(default)
cce/9.0.0-classic                         cray-netcdf/4.6.3.2(default)              craypkg-gen/1.3.7(default)
cce/9.1.0(default)                        cray-netcdf-hdf5parallel/4.6.1.3          fftw/2.1.5.9(default)
cce/9.1.0-classic                         cray-netcdf-hdf5parallel/4.6.3.0          gdb4hpc/3.0.10(default)
cdt/19.03                                 cray-netcdf-hdf5parallel/4.6.3.2(default) iobuf/2.0.8
cdt/19.06                                 cray-openshmemx/8.0.1                     iobuf/2.0.9(default)
cdt/19.11(default)                        cray-parallel-netcdf/1.8.1.4              modules/3.2.11.1
cray-ccdb/3.0.4                           cray-parallel-netcdf/1.11.1.0             modules/3.2.11.2
cray-ccdb/3.0.5(default)                  cray-parallel-netcdf/1.11.1.1(default)    modules/3.2.11.4(default)
cray-cti/1.0.7                            cray-petsc/3.9.3.0                        modules/4.1.3.1
cray-cti/1.0.9(default)                   cray-petsc/3.11.2.0(default)              papi/5.6.0.6
cray-fftw/3.3.8.2                         cray-petsc-64/3.9.3.0                     papi/5.7.0.1
cray-fftw/3.3.8.3                         cray-petsc-64/3.11.2.0(default)           papi/5.7.0.2(default)
cray-fftw/3.3.8.4(default)                cray-petsc-complex/3.9.3.0                perftools/7.0.6(default)
cray-ga/5.3.0.10(default)                 cray-petsc-complex/3.11.2.0(default)      perftools/7.1.0
cray-hdf5/1.10.2.0                        cray-petsc-complex-64/3.9.3.0             perftools-base/7.0.6
cray-hdf5/1.10.5.0                        cray-petsc-complex-64/3.11.2.0(default)   perftools-base/7.1.0
cray-hdf5/1.10.5.2(default)               cray-shmem/7.7.6                          perftools-base/7.1.1(default)
cray-hdf5-parallel/1.10.2.0               cray-shmem/7.7.8                          perftools-lite/7.0.6(default)
cray-hdf5-parallel/1.10.5.0               cray-shmem/7.7.10(default)                perftools-lite/7.1.0
cray-hdf5-parallel/1.10.5.2(default)      cray-tpsl/18.06.1                         pmi/5.0.11
cray-lgdb/3.0.10(default)                 cray-tpsl/19.06.1(default)                pmi/5.0.14(default)
cray-libsci/19.02.1                       cray-tpsl-64/18.06.1                      pmi-lib/5.0.11
cray-libsci/19.06.1(default)              cray-tpsl-64/19.06.1(default)             pmi-lib/5.0.14(default)
```

Figure 4: Module Configuration on NERSC's Cori



```
--------------- /autofs/nccs-svm1_sw/summit/modulefiles/site/linux-rhel7-ppc64le/spectrum-mpi/10.3.1.2-20200121-p6nrnt6/xl/16.1.1-5 ----------------
   adios2/2.4.0                   fftw/3.3.8         hypre/2.11.1        netcdf-cxx4/4.3.0       netlib-scalapack/2.0.2    parmetis/4.0.3
   amgx/2.1.0-1-unthreaded        hdf5/1.8.18        hypre/2.13.0 (D)    netcdf-fortran/4.4.4    parallel-io/2.3.0         petsc/3.7.2-py2
   boost/1.59.0            (D)    hdf5/1.10.4 (D)    mpip/3.4.1-1        netcdf/4.6.1            parallel-netcdf/1.8.1

-------------------------------------------- /sw/summit/modulefiles/site/linux-rhel7-ppc64le/xl/16.1.1-5 --------------------------------------------
   autoconf/2.69   (D)    gdbm/1.18.1       (D)    libelf/0.8.13           ncurses/6.1                spectrum-mpi/10.3.1.2-20200121 (L)
   automake/1.16.1 (D)    hdf5/1.8.18              libsodium/1.0.15 (D)    netlib-lapack/3.8.0        sqlite/3.26.0                  (D)
   boost/1.59.0           hdf5/1.10.3              libtool/2.4.2           numactl/2.0.11      (D)    zeromq/4.2.5                   (D)
   bzip2/1.0.6            hdf5/1.10.4              libunwind/1.2.1         openssl/1.0.2       (D)    zfp/0.5.2                      (D)
   cmake/3.15.2    (D)    kokkos/3.0.00            m4/1.4.18        (D)    pkgconf/1.5.4       (D)    zlib/1.2.11                    (D)
   diffutils/3.7   (D)    libdwarf/20180129        metis/5.1.0             readline/7.0        (D)

------------------------------------------------ /sw/summit/modulefiles/site/linux-rhel7-ppc64le/Core ------------------------------------------------
   DefApps                           (L)      hsi/5.0.2.p5               (L)      pgi/19.10
   antlr/2.7.7                                htop/2.0.2                          pgi/20.1
   apr-util/1.6.0                             hwloc/2.0.2-py3                     pixman/0.38.0
   autoconf/2.69                              hwloc/2.0.2                (D)      pkgconf/1.5.4
   automake/1.16.1                            icu4c/58.2                          py-certifi/2017.1.23-py3
   bison/3.0.5                                libassuan/2.4.5                     py-docutils/0.13.1-py3
   c-blosc/1.12.1                             libbsd/0.9.1                        py-nose/1.3.7-py2
   cairo/1.16.0-py3                           libevent/2.1.8                      py-nose/1.3.7-py3
   cmake/3.6.1                                libfabric/1.7.0                     py-nose/1.3.7            (D)
   cmake/3.9.2                                libgcrypt/1.8.1                     py-pip/9.0.1
   cmake/3.11.3                               libgd/2.2.4                         py-pip/10.0.1-py2
   cmake/3.13.4                               libgpg-error/1.27                   py-pip/10.0.1-py3        (D)
   cmake/3.14.2                               libksba/1.3.5                       py-pygments/2.2.0-py3
   cmake/3.15.2                               libnl/3.3.0                         py-setuptools/25.2.0
   cuda/9.1.85                                libpciaccess/0.13.5                 py-setuptools/40.2.0-py2
   cuda/9.2.148                               libpng/1.6.34                       py-setuptools/40.2.0-py3
   cuda/10.1.105                              libsodium/1.0.15                    py-setuptools/40.4.3-py2
   cuda/10.1.168                              libx11/1.6.7                        py-setuptools/40.4.3-py3 (D)
   cuda/10.1.243                     (D)      libxcb/1.13                         py-virtualenv/15.0.1
   curl/7.63.0                                libxdmcp/1.1.2                      py-virtualenv/16.0.0-py2
   darshan-runtime/3.1.7-hdf5pre110           libxext/1.3.3                       py-virtualenv/16.0.0     (D)
   darshan-runtime/3.1.7-hdf5post110          libxml2/2.9.8                       python/2.7.12
   darshan-runtime/3.1.7             (L,D)    libxpm/3.5.12                       python/2.7.15
   darshan-util/3.1.6                         libxrender/0.9.10                   python/3.5.2
   darshan-util/3.1.7                (D)      llvm/1.0-20180531                   python/3.7.0
   diffutils/3.7                              llvm/1.0-20190225          (D)      rdma-core/20
   emacs/25.1                                 llvm/9.0.0-1-debug                  readline/6.3
   essl/6.1.0-2                      (D)      llvm/9.0.0-1                        renderproto/0.11.1
   essl/6.2.0-20190419                        llvm/9.0.0-2-debug                  scons/3.0.1-py2
   expat/2.2.5                                llvm/9.0.0-2                        screen/4.3.1
   fontconfig/2.12.3                          lsf-tools/1.0                       serf/1.3.9-py2
   freetype/2.9.1                             lsf-tools/2.0              (L,D)    snappy/1.1.7
   gcc/4.8.5                                  lz4/1.8.1.2                         spectral/20181227
   gcc/5.4.0                                  m4/1.4.18                           spectral/20190401        (D)
   gcc/6.4.0                         (D)      makedepend/1.0.5                    sqlite/3.26.0
   gcc/7.4.0                                  mercurial/3.9.1                     subversion/1.9.3-py2
   gcc/8.1.0                                  mercurial/4.4.1-py3                 subversion/1.9.3         (D)
   gcc/8.1.1                                  mercurial/4.4.1            (D)      tar/1.31
   gcc/9.1.0                                  nano/2.6.3                          texinfo/6.5
   gdb/8.0                                    nco/4.9.1                           tmux/2.2
   gdb/8.2-py3                       (D)      netcdf/4.6.2               (D)      udunits2/2.2.24
   gdbm/1.18.1                                npth/1.5                            valgrind/3.11.0
   gettext/0.19.8.1                           nsight-compute/2019.5.0             valgrind/3.14.0          (D)
   git-lfs/2.8.0                              nsight-systems/2020.1.1.65          vim/7.4.2367
   git/2.9.3                                  numactl/2.0.11                      vim/8.1.0338             (D)
   git/2.13.0                                 openssl/1.0.2                       xalt/0.7.5
   git/2.20.1                        (D)      pango/1.41.0-py3                    xalt/1.1.3
   glib/2.56.3-py3                            papi/5.5.1                          xalt/1.1.4
   gnupg/2.2.3                                papi/5.6.0                          xalt/1.2.0               (L,D)
   gnuplot/5.0.1-py3                          papi/5.7.0                 (D)      xl/16.1.1-4
   gnuplot/5.0.1                     (D)      patchelf/0.9                        xl/16.1.1-5              (L,D)
```

Figure 5: Module Configuration on ORNL's Summit

A greater issue was that many Department of Energy scientists treat the GNU toolchain as their primary and often only supported toolchain. This is problematic as many flags are hardcoded (e.g., `-march=native`) in a way that GNU can understand but not other compilers. Adapting the build systems to provide this information in a more vendor-agnostic fashion was a more involved task but also greatly increased the readiness of these applications for exascale platforms.

5.3.2 Tool Chain Issues with Vendor Specific Workarounds

Once we verified that the proxy apps could be built with general toolchains, we focused on vendor specific ones. Initially we used the ORNL Power9-based ASCENT machine.

Once these were fixed, we added CI on ORNL's Power9-based Ascent machine. During this step, we hit two new kinds of issues in Spack. The first one was due to the fact that the GNU compiler on Ascent is actually an IBM XL compiler in disguise. This vendor-provided workaround resulted in Spack, and the proxy apps team, making false assumptions about the compiler. Once we informed Spack that this was the XL compiler in disguise, these issues were resolved. The second issue was again hard-coded flags, that this time could be not processed by the IBM XL compiler. We fixed those issues in Spack as well.

5.3.3 Automated Workflows

The CI pipelines are triggered automatically for each push to a main repository, which maps well to pull requests and development. Travis CI and Gitlab CI cron jobs are planned once we got all proxy apps to build ASCENT, which is required for nightly and weekly regression tests.

5.3.4 Credentials and Workflows

Currently the process still requires us to request an account on each individual site, but with the federalization of ECP CI this will hopefully improve.

5.3.5 A Moving Target targeting Moving Targets

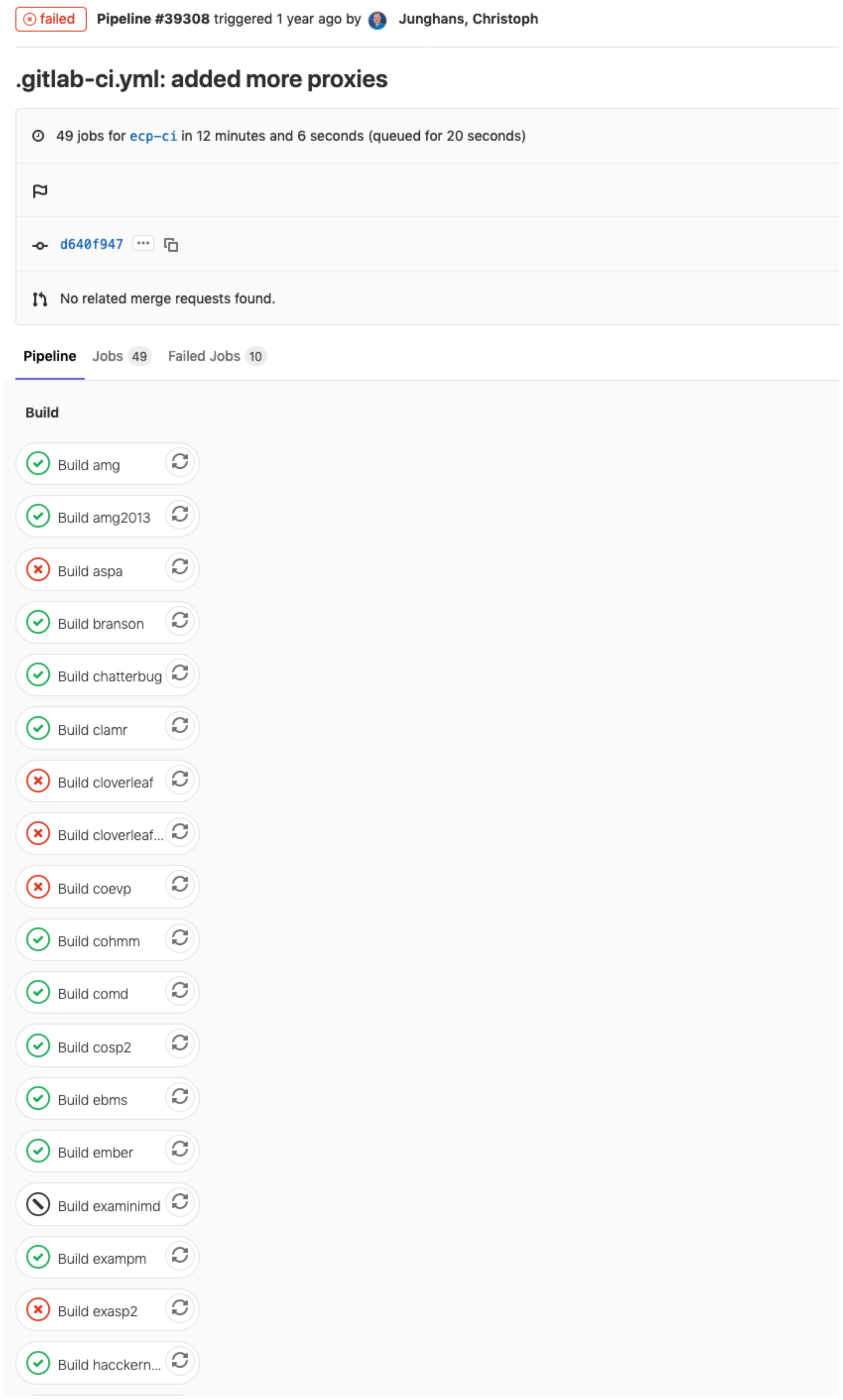
Much of this work has been used to stress-test and improve the Gitlab CI work. Because of very responsive support from Onyx Point, we frequently encountered needs for software updates and the need for facilities to update the Gitlab installations.

And, as previously mentioned, the Gitlab CI installations primarily focused on early testing platforms to evaluate hardware and software for pre- and post-exascale platforms. In many of these cases the platforms simultaneously were used by friendly users, facilities, and the vendors themselves. This often resulted in toolchain and environment changes occurring on a weekly basis.

The combination of environment updates from all of these directions at once meant that a script that worked one day could mysteriously fail the next, with no clear indication of the cause. This resulted in a lot of debugging to determine what changed, and what needs to change to support this. An example of such a failure can be seen in Figure 6.

5.3.6 Export Control and NDAs

The Exascale Computing Project is an incredibly diverse project with members coming from a wide range of backgrounds, employment classifications, and nationalities. While vital to develop solid software and engage in good science, this proved somewhat problematic for the purpose of debugging issues on many of these platforms. Much of the information on these early testing platforms was



failed Pipeline #39308 triggered 1 year ago by Junghans, Christoph

.gitlab-ci.yml: added more proxies

49 jobs for ecp-ci in 12 minutes and 6 seconds (queued for 20 seconds)

d640f947

No related merge requests found.

Pipeline Jobs 49 Failed Jobs 10

Build

- ✓ Build amg
- ✓ Build amg2013
- ✗ Build aspa
- ✓ Build branson
- ✓ Build chatterbug
- ✓ Build clamr
- ✗ Build cloverleaf
- ✗ Build cloverleaf...
- ✗ Build coevp
- ✓ Build cohmm
- ✓ Build comd
- ✓ Build cosp2
- ✓ Build ebms
- ✓ Build ember
- ⊘ Build examinimd
- ✓ Build exampm
- ✗ Build exasp2
- ✓ Build hacckern...

Figure 6: Example of a Partial Failure in the CI Pipeline

still under NDA, and not every organization was under the same shared NDA, and identifying what information could and could not be shared proved difficult. Similarly, some organizations interpreted the export control determinations in different ways, which further restricted who could and could not be brought in to debug and troubleshoot.

This generally resulted in the ECP Proxy Apps team needing to identify issues independently without any support, and trying to isolate the aspects of these codes, which had issues with certain aspects of the early testing platforms. This was clearly sub-optimal for many reasons.

5.4 Future Work

Members of the ECP Proxy App Team have recently provided assistance to vendor partners in the El Capitan Center of Excellence (COE) who were working with the Spack team to stand up a CI system on the Redwood system at HPE. This system is being used to test the HPE and AMD software stacks against open-source libraries and proxy apps of importance to LANL, LLNL, and SNL. The COE plans to use this system to provide early detection of compiler and other toolchain issues that will impact NNSA applications. Ideally, both functionality and performance regressions will be detectable.

The Proxy App Team has proposed to facilitate a similar CI system running on early hardware for Frontier and Aurora. Such a system will be most successful if AD and ST teams will identify critical features, kernels, patterns, etc. and work with the Proxy App Team to find or create proxies that exhibit these critical needs. Having such a collection in a CI system will help keep these issues firmly in view and help assure that system vendors focus on providing the solutions needed by ECP applications. The system will also help ECP developers stay abreast of the status of various needed features.

6 Acknowledgments

We thank the following individuals for their authorship, expertise, and assistance throughout all aspects of the ML proxy app release and for their help in writing the ML section (3) of this report:

- Siva Rajamanickam[1], J. Austin Ellis[1]: MiniGAN
- ExaLearn Control Team [24]: miniRL
- Kristofer James Ekhart Zieb[2]: CRADL
- Steve Farrell[3]: Cosmoflow-Benchmark
- Thorsten Kurth[4], Steve Farrell[3]: MLperf-DeepCam

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

Lawrence Livermore National Laboratory is operated by Lawrence Livermore National Security, LLC, for the U.S. Department of Energy, National Nuclear Security Administration under Contract DE-AC52-07NA27344.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Oak Ridge National Laboratory is managed by UT-Battelle, LLC, for the U.S. Department of Energy Under contract DE-AC05-00OR22725.

Los Alamos National Laborary is operated by Triad National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy under Contract No. 89233218CNA000001.

Argonne National Laboratory is managed by UChicago Argonne LLC for the U.S. Department of Energy under contract DE-AC02-06CH11357.

The Lawrence Berkeley National Laboratory is a national laboratory managed by the University of California for the U.S. Department of Energy under Contract Number DE-AC02-05CH11231.

[1]Sandia National Laboratories, Albuquerque, NM
[2]Lawrence Livermore National Laboratory, Livermore, CA
[3]Lawrence Berkeley National Laboratory, Berkeley, CA
[4]NVIDIA Corporation

References

[1] ECP CI Documentation. URL: https://ecp-ci.gitlab.io/docs/admin.html#runners.

[2] Intel(r) math kernel library for deep neural networks (intel(r) mkl-dnn). URL: https://github.com/intel/mkl-dnn.

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[4] Xavier Andrade, Joseba Alberdi-Rodriguez, David A Strubbe, Micael JT Oliveira, Fernando Nogueira, Alberto Castro, Javier Muguerza, Agustin Arruabarrena, Steven G Louie, Alán Aspuru-Guzik, et al. Time-dependent density-functional theory in massively parallel computer architectures: the OCTOPUS project. *Journal of Physics: Condensed Matter*, 24(23):233202, 2012.

[5] Xavier Andrade and Alán Aspuru-Guzik. Real-space density functional theory on graphical processing units: computational approach and comparison to gaussian basis set methods. *Journal of chemical theory and computation*, 9(10):4360–4373, 2013.

[6] Xavier Andrade, David Strubbe, Umberto De Giovannini, Ask Hjorth Larsen, Micael JT Oliveira, Joseba Alberdi-Rodriguez, Alejandro Varas, Iris Theophilou, Nicole Helbig, Matthieu J Verstraete, et al. Real-space grids and the Octopus code as tools for the development of new simulation approaches for electronic systems. *Physical Chemistry Chemical Physics*, 17(47):31371–31396, 2015.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[8] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[9] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative Adversarial Networks: An Overview. *IEEE Signal Processing Magazine*, 2018. arXiv:1710.07035, doi:10.1109/MSP.2017.2765202.

[10] Reiner M Dreizler and Eberhard KU Gross. *Density functional theory: an approach to the quantum many-body problem.* Springer Science & Business Media, 2012.

[11] J.A. Ellis and S. Rajamanickam. miniGAN. https://proxyapps.exascaleproject.org/app/minigan/, 2020.

[12] Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. SEED RL: Scalable and efficient deep-RL with accelerated central inference. *arXiv preprint arXiv:1910.06591*, 2019.

[13] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-RL with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[14] Mark Gates, Jakub Kurzak, Ali Charara, Asim YarKhan, and Jack Dongarra. SLATE: design of a modern distributed and accelerated linear algebra library. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–18, 2019.

[15] Shlomo Geva and Joaquin Sitte. A cartpole experiment benchmark for trainable controllers. *IEEE Control Systems Magazine*, 13(5):40–51, 1993.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. 2014. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[17] Francois Gygi, Erik W Draeger, Martin Schulz, Bronis R De Supinski, John A Gunnels, Vernon Austel, James C Sexton, Franz Franchetti, Stefan Kral, Christoph W Ueberhuber, et al. Large-scale electronic structure calculations of high-Z metals on the BlueGene/L platform. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, pages 45–es, 2006.

[18] Mohamed Hacene, Ani Anciaux-Sedrakian, Xavier Rozanska, Diego Klahr, Thomas Guignon, and Paul Fleurat-Lessard. Accelerating VASP electronic structure calculations using graphic processing units. *Journal of computational chemistry*, 33(32):2581–2589, 2012.

[19] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 11–19, 2017.

[20] Amrita Mathuriya, Deborah Bard, Peter Mendygral, Lawrence Meadows, James Arnemann, Lei Shao, Siyu He, Tuomas Kärnä, Diana Moise, Simon J. Pennycook, Kristyn Maschhoff, Jason Sewall, Nalini Kumar, Shirley Ho, Michael F. Ringenburg, Prabhat, and Victor Lee. CosmoFlow: Using deep learning to learn the universe at scale. In *SC'18*. IEEE Press, 2018.

[21] Mustafa Mustafa, Deborah Bard, Wahid Bhimji, Zarija Lukic, Rami Al-Rfou, and Jan Kratochvíl. CosmoGAN: Creating high-fidelity weak lensing convergence maps using generative adversarial networks. *Computational Astrophysics and Cosmology*, 6:1–13, 2017.

[22] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[23] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, pages 1–16, 2016. arXiv:arXiv:1511.06434v2, doi:10.1007/s11280-018-0565-2.

[24] Vinay Ramakrishnaiah, Malachi Schram, Jamal Mohd-Yusof, Sayan Ghosh, Yunzhi Huang, Ai Kagawa, Christine Sweeney, and Shinjae Yoo. Easily eXtendable Architecture for Reinforcement Learning (EXARL). https://github.com/exalearn/ExaRL, 2020.

[25] Siamak Ravanbakhsh, Junier Oliva, Sebastian Fromenteau, Layne Price, Shirley Ho, Jeff Schneider, and Barnabás Póczos. Estimating cosmological parameters from the dark matter distribution. In *International Conference on Machine Learning*, pages 2407–2416, 2016.

[26] Joshua Romero, Everett Phillips, Gregory Ruetsch, Massimiliano Fatica, Filippo Spiga, and Paolo Giannozzi. A performance study of Quantum ESPRESSO's PWscf code on multi-core and GPU systems. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, pages 67–87. Springer, 2017.

[27] André Schleife, Erik W Draeger, Yosuke Kanai, and Alfredo A Correa. Plane-wave pseudopotential implementation of explicit integrators for time-dependent Kohn-Sham equations in large-scale simulations. *The Journal of chemical physics*, 137(22):22A546, 2012.

[28] Alexander Sergeev and Mike Del Balso. Horovod: Fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

[29] M A Sprague, S Ananthan, G Vijayakumar, and M Robinson. ExaWind: A multifidelity modeling and simulation environment for wind energy. *Journal of Physics: Conference Series*, 1452:012071, jan 2020. doi:10.1088/1742-6596/1452/1/012071.

[30] Alexey V Titov, Ivan S Ufimtsev, Nathan Luehr, and Todd J Martinez. Generating efficient quantum chemistry codes for novel architectures. *Journal of chemical theory and computation*, 9(1):213–221, 2013.

[31] Justin M Wozniak, Rajeev Jain, Prasanna Balaprakash, Jonathan Ozik, Nicholson T Collier, John Bauer, Fangfang Xia, Thomas Brettin, Rick Stevens, Jamaludin Mohd-Yusof, et al. CANDLE/Supervisor: A workflow framework for machine learning applied to cancer research. *BMC bioinformatics*, 19(18):59–69, 2018.

[32] Victor Wen-zhe Yu, Carmen Campos, William Dawson, Alberto García, Ville Havu, Ben Hourahine, William P Huhn, Mathias Jacquelin, Weile Jia, Murat Keçeli, et al. ELSI – an open infrastructure for electronic structure solvers. *Computer Physics Communications*, 256:107459, 2020.