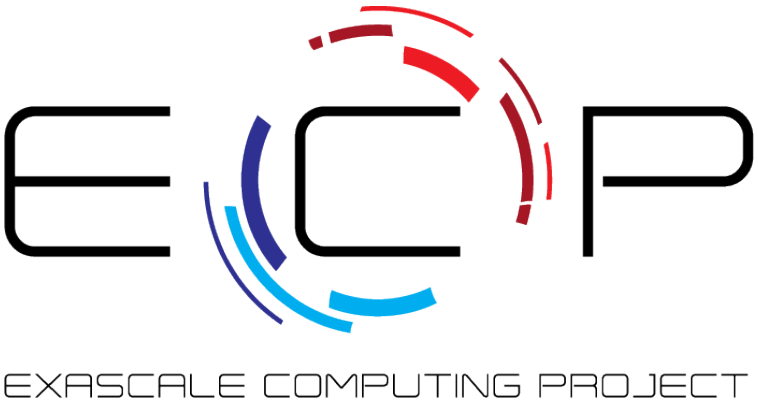
FY21 Proxy App Suite Release

Report for ECP Proxy App Project Milestone ADCD504-12

David F. Richards[1], Omar Aaziz[2], Yuri Alexeev[3], Ramesh Balakrishnan[3], Jeanine Cook[2], Graham Fletcher[3], Christoph Junghans[4], Youngdae Kim[3], Jeffery Kuehn[4], Nevin Liber[3], Geng Liu[3], Amanda Lund[3], Alvaro Mayagoitia[3], Peter McCorquodale[5], Robert Pavel[4], Vinay Ramakrishnaiah[4], Courtenay Vaughan[2], and The ECP Proxy App Team[6]

[1]Lawrence Livermore National Laboratory, Livermore, CA
[2]Sandia National Laboratories, Albuquerque, NM
[3]Argonne National Laboratory, Chicago, IL
[4]Los Alamos National Laboratory, Los Alamos, NM
[5]Lawrence Berkeley National Laboratory, Berkeley, CA
[6]https://proxyapps.exascaleproject.org/team

Sept 2021



LLNL-TR-827482

Contents

1 Executive Summary

The FY21 Proxy App Suite Release milestone includes the following activities:

> Curate a collection of proxy applications that represents the breadth of ECP applications, including application domains, programming models, supporting libraries, numerical methods, etc. Identify gaps in coverage and work with application teams to commission or develop proxies to cover gaps. From within this collection, designate the "ECP Proxy Application Suite" of 12–15 proxies that balance breadth of coverage with ease of use and quality of implementation. Also designate approximately 8–10 proxies to form the "ECP Machine Learning Proxy Suite". The ML suite will represent algorithms, use cases, and programming methods typically used by ECP science workloads to incorporate machine learning into their workflows.

Version 5.0 of the ECP Proxy App Suite is 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. Section 2 contains a description of the current contents of the suite.

Although we have elected not to make any changes to to the ECP suite, the team has been hard at work in other areas:

- We continue to maintain a catalog of ECP-and DOE-relevant proxy applications. This catalog now contains over 75 entries. Brief descriptions of the proxies that have been added to this collection since our last release are provided in Section 3.
- We have expanded the Machine Learning Suite by adding MiniRL. Section 4 describes miniRL.
- We are using cosine similarity metrics to evaluate proxy applications. Section 5 presents recent work in this area including how to identify gaps and redundancies within a proxy suite.
- Finally, the team is involved in the development of seven new proxy applications covering a variety of areas. These efforts are described in Section 6.

Our project website is `https://proxyapps.exascaleproject.org`

2 The Proxy App Suite v5.0

The ECP Proxy App Team released version 5.0 of the ECP Proxy App Suite on October 1, 2021 (https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/) . This release is 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 5.0 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 75 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.

3 Additions to the ECP Proxy App Collection

We maintain a catalog of ECP-and DOE-relevant proxy applications at https://proxyapps.exascaleproject.org/app/. This catalog now contains over 75 entries. This section contains brief descriptions of the proxies that have been added to this collection since our last release.

3.1 miniMDock

A GPU-accelerated performance portable particle-grid based protein ligand molecular docking tool that can be used for virtual drug discovery compound screens based on a molecular recognition model, that analysis a three-dimensional model of an interaction between a protein and a small molecule (ligand).
Languages: `C, C++, Cuda, HIP, Kokkos`
Institution: Oak Ridge National Laboratory
Repo: https://github.com/ORNL-PE/miniMDock

3.2 Goulash

The goulash project will contain a set of proxy codes that test compiler and linker interoperability features that are needed to build complex applications with multiple libraries that are written in a variety of languages and programming models. Tests will be assembled from published fundamental algorithms, basic numerical methods, and the tools and scripts needed to launch the suite and verify results. Tests may include material from previously released proxy apps but will not include full applications or libraries. LLNL-CODE-795383
Languages: `C, C++, Fortran, Python`
Institution: Lawrence Livermore National Laboratory
Repo: https://github.com/LLNL/goulash

3.3 miniRL

The architecture of EXARL is separated into learner and actors. 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 and SEED, each actor in EXARL independently stores experiences and runs the Bellman equation to generate training data. These training data are sent back to the leaner, 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' 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 functionality built-in, which allows for hyperparameter optimization using the

CANDLE Supervisor.
Languages: `Python, C, C++`
Institution: Los Alamos National Laboratory
Repo: https://github.com/lanl/minRL/

3.4 ExaCMech

The miniapp itself acts as a multi-point simulator of a simple uniform deformation problem and outputs the average Cauchy stress tensor from either a user supply of unit quaternion grain orientations or N randomly generated grain orientations. The user can choose between two different material models to run the miniapp under. One model contains simpler physics and is therefore cheaper to run over the more realistic physics model. Next, the user is able to swap between the various RAJA backends such as CPU, OpenMP, CUDA, or in a future release HIP in order to compare performance between different backends and platforms. Finally, the portions of the miniapp that drive the constitutive model follows a similar design to the code used within ExaConstit, an open-source solid mechanics/crystal plasticity FEM code used within the ExaAM project. So, the miniapp provides a basis for how one might integrate ExaCMech within an application code. Note: the ExaCMech miniapp lives in the same repo as the ExaCMech library rather than a separate repo.
Languages: `C++, RAJA`
Institution: Lawrence Livermore National Laboratory
Repo: https://github.com/LLNL/ExaCMech

3.5 miniERIs

This mini-app computes the electron-repulsion-integral (ERIs) using the rotated axis algorithm for s,p and d basis functions and then build the Fock matrix by contracting the density matrix with the ERIs. Both the ERI evaluation kernels and Fock building kernel are offloaded. Currently, the mini-app assumes the inputs are water clusters. The code will be generalized to other types of physical systems in the future.
Languages: `Fortran 90, OpenMP`
Institution: Argonne National Laboratory
Repo: Contact gamess@iastate.edu for access

3.6 CabanaMD

A molecular dynamics proxy app built with Kokkos and Cabana.
Languages: `C++, MPI, Kokkos`
Institution: Oak Ridge National Laboratory
Repo: https://github.com/ECP-copa/CabanaMD

3.7 ExaMPM

Material point method proxy app built with Cabana and Kokkos for fluid and solid mechanics.
Languages: `C++, MPI, Kokkos`
Institution: Oak Ridge National Laboratory
Repo: https://github.com/ECP-copa/ExaMPM

3.8 exawind-virtuals-app

This application contains a minimal example of device virtual functions used in Exawind/Nalu-Wind codebase. This tests virtual function capability on device. The core physics kernels are implemented as sub-classes of Kernel and Algorithm is the driver that holds a vector of user-registered computational kernels as pointers to the base class. These instances are created on device using placement new and run on device.
Languages: `C++, CMake`
Institution: Sandia National Laboratories
Repo: https://github.com/Exawind/exw-virtuals-app

3.9 hypre-mini-app

A mini-app to use HYPRE solvers and preconditioners on linear systems generated by the Nalu-Wind low-Mach code. This app reads the linear system matrix, RHS, and reference solution data in MatrixMarket and HYPRE IJ formats.
Languages: `C++, CMake`
Institution: Sandia National Laboratories
Repo: https://github.com/Exawind/hypre-mini-app

3.10 E3SM-kernels

These kernels originate from the Energy Exascale Earth System Model (E3SM). They are intended for sharing with interested parties for co-design purposes.
Languages: `Fortran`
Institution: Oak Ridge National Laboratories
Repo: https://github.com/e3SM-Project/codesign-kernels

3.11 basic-hf-proxy

This proxy application simulates the compute load and data-movement of the kernel of the Hartree-Fock method in quantum chemistry. The proxy features a simplified algorithm for computing electron repulsion integrals that is easily offloaded to GPUs, together with integral screening to avoid small terms.
Languages: `Fortran-90`
Institution: Argonne National Laboratories
Repo: https://github.com/gdfletcher/basic-hf-proxy

3.12 minTally

A mini-app that abstracts the key components of the tallying system from a full Monte Carlo neutron transport application
Languages: `C, OpenMP`
Institution: Argonne National Laboratory
Repo: https://github.com/amandalund/minTally

4 Machine Learning Proxy App Suite

About 77% of the total data in the world was created in the last five years, which has forced us to develop automatic techniques to understand it. Machine learning (ML) is at the forefront of technologies used to process vast amounts of data as it puts the onus on the machine to learn new techniques to process data rather than manual development. This paradigm shift is necessary to overcome data explosion and we have seen machine learning gain traction in practically every field of computing.

New machine learning techniques are being developed everyday that have profound implications for computational and experimental science and engineering. This lead the ECP proxy apps team to collect and curate different ML workflows relevant to scientific applications that represents how unique ML techniques are applied to DOE science that is different from typical use cases of academia and industries.

In the previous report (Milestone ADCD-504-10), we identified how ML proxies are different from traditional proxies and also laid out the criteria to select them. The first criterion is to embody the core purposes and uses of proxy apps in general and the second is for the suite to cover the breadth of the ML application space. We have been working on identifying and developing ML proxies based on the above mentioned criteria and have used some of the ML proxies for testing software stacks for Aurora and Frontier as described below:

- Candle P1B1, P3B1, and Uno benchmarks were used to test the status and operations coverage of the Alpha 1.x versions of TF on Gen9 Iris, and DG1 Yarrow (JLSE).
- Candle P1B1, P3B1, and Uno benchmarks were used to test the status and operations coverage of Alpha 2 version of TF on Gen9 Iris, DG1 Yarrow and ATS Articus nodes (JLSE).
- OLCF effort as part of the CANDLE project to port Microsoft's DeepSpeed to Frontier COE systems
- Initial version of miniRL was used to test additional operations coverage required for reinforcement learning on JLSE and Frontier COE systems.
- CANDLE benchmarks were used to develop the CI workflows used to monitor changes on early access platforms.

Candle benchmarks implement deep learning architectures that are relevant to problems in cancer. These architectures address problems at different biological scales, specifically problems at the molecular, cellular, and population scales. miniRL is a reinforcement learning proxy application derived from the Easily eXtensible Architecture for Reinforcement Learning framework. At the time of our FY20 milestone report ADCD-504-10, miniRL was under development. It has since been completed and released. The next section provides details about the implementation of miniRL and some preliminary results.

4.1 miniRL: A Proxy App for the Easily eXtendable Architecture for Reinforcement Learning

Reinforcement Learning (RL) is an elegant formalization of the problem of intelligence and deep RL has led to many groundbreaking discoveries. These advancements in RL come at a cost of added complexity and scale of the underlying algorithms and are hindering researchers to rapidly prototype ideas. The Easily eXtendable Architecture for Reinforcement Learning (EXARL) [12] framework is an attempt to alleviate this hindrance and advance the use of RL for scientific applications. The goal of EXARL is exploration of online learning of RL systems using Exascale machines for generating data and training the neural network in real time.

In Reinforcement Learning, an agent interacts with an environment with the objective of maximizing a quantifiable notion of cumulative reward. An 'agent' is a software process that provides an action, either explorative or exploitative, to the environment. An 'environment' 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 proxy application derived from the EXARL framework, which is being developed by the ExaLearn Control project (ADCD08). It captures the important communication and computational aspects of EXARL, which is demonstrated by using a Deep Q-Network (DQN) agent and a synthetic environment called ExaCartPole. 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.

In EXARL, an agent is a collection of RL algorithms with a state table or associated neural network architecture that controls an environment. EXARL includes distributed learning workflows which define how the agent and environment interact with each other to improve scalability. Additionally, EXARL uses an extension of the OpenAI Gym [1] framework, which not only allows existing benchmark environments in Gym to be used but also provides easy integration of new scientific environments.

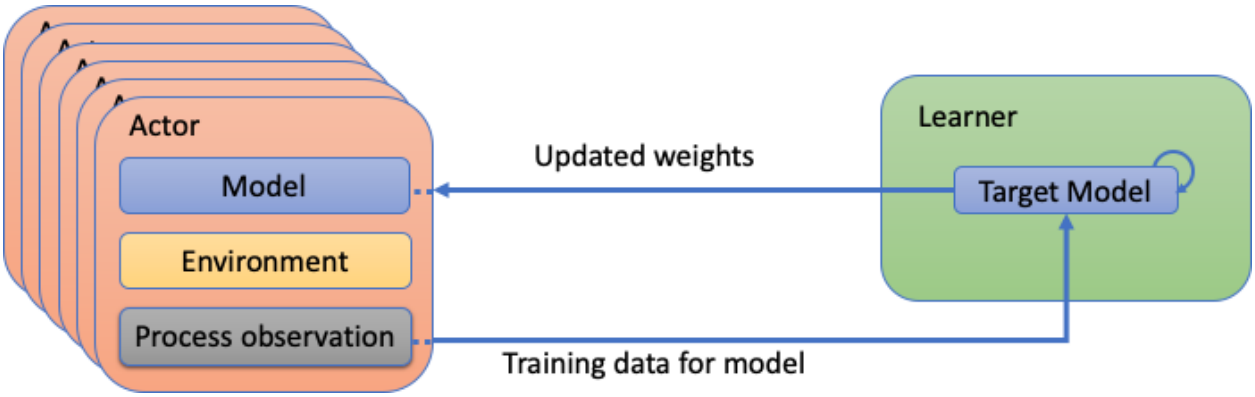


Figure 1: Overview of EXARL architecture. Each actor has a copy of the model, which is updated in every step and used to infer a new action given a state. The actor uses this action to compute the next environment state and runs the Bellman equation to generate updated data. This data is sent back to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

The architecture of EXARL is separated into learners and actors as shown in Figure 1. The learner consists of a target model that is trained using training data (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 [4] and SEED [3], each actor in EXARL independently stores experiences and runs the Bellman equation to generate training data. The Bellman equation is used to estimate the optimal expected reward at a state and provides the necessary condition for optimality of a control with respect to the loss function. In the DQN agent, the computation of Bellman equations take about 80% of the total agent time. The training data is sent back to the learners 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 after completing their current task. 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 ExaCartPole environment, which is a synthetic environment comprised of the well known inverted pendulum or 'CartPole' [5] environment with the added complexity of computing PI in a distributed manner. A DQN agent is trained on this environment and learning behavior can be analyzed. Note that the distributed computation of PI does not affect convergence of training but is rather introduced to add additional computational complexity and test the ability of EXARL to support complex, multi-process environments. With the EXARL framework, it is easy to swap different environments, agents, as well as workflows for testing the performance. miniRL incorporates this along with the built-in CANDLE [14] functionality, which allows for hyperparameter optimization using the CANDLE Supervisor.

miniRL uses MPI for distributed memory parallelism. The training and generation of experience (training data) can run on multiple nodes. The framework inherently supports complex scientific simulation environments that has to be run in parallel for all practical purposes. The RL software agents can be classified into actors and learners. The actors are responsible for running the environments and generating training data and the learners utilize this data to train the policy. The interaction between RL actors and learners is defined as a workflow in this context. miniRL currently supports three workflows:

- sync
- async
- rma

4.1.1 `sync` workflow

The synchronous (`sync`) workflow is designed to support multiple environment groups, each running multi-process environments in parallel as shown in Fig. 2. The launched MPI processes are divided into MPI groups, each with their own sub-communicator. A fixed number of processes can be assigned to each environment group and multiple environment groups can generate experiences in parallel for training. The generated experiences are gathered in the rank 0 process of each environment group (actors), which in-turn acts as learner processes. The learner processes train the optimal policy by utilizing the generated training data by the actors.

RL agents can either take a random action or utilize the learned policy for action, while gradually leaning towards exploiting the learned policy as the training progresses. As a result, actors can get different control parameters at each step and can take different paths toward episode completion. In addition, some environment simulations are inherently stochastic. All these factors can lead to a load imbalance and waiting for all environment groups to complete, stalls some actor processes and results in wasted cycles. This is one of the drawbacks of the sync workflow.

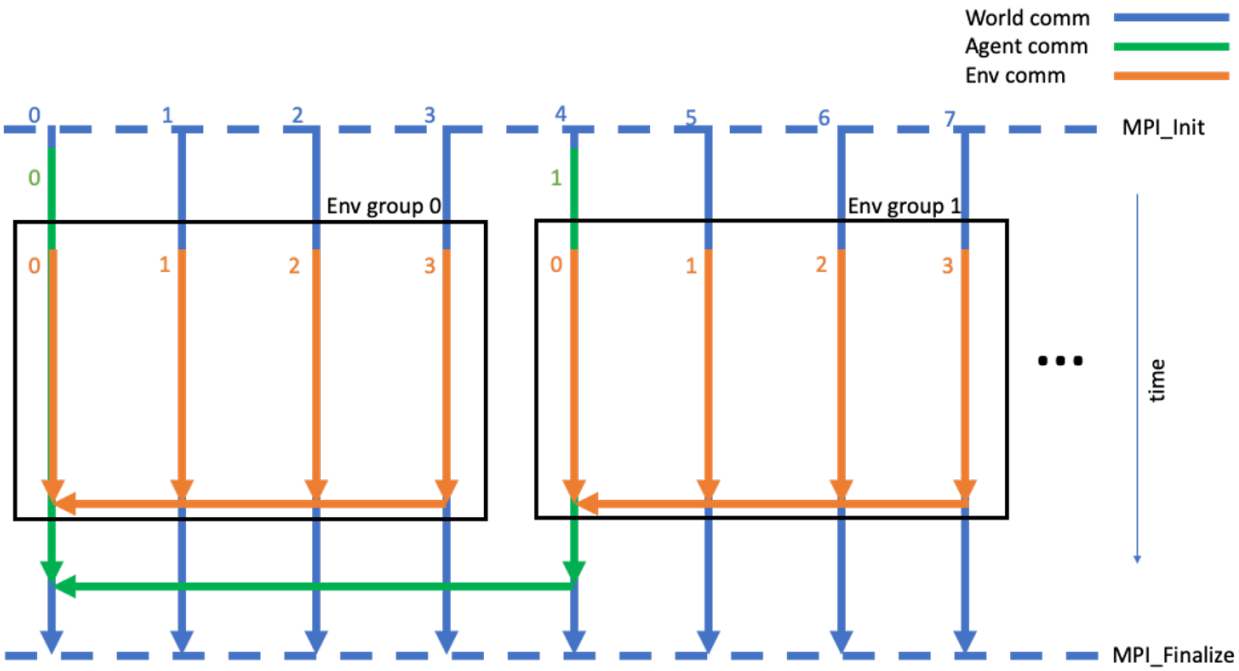


Figure 2: sync workflow.

4.1.2 `async` workflow

To alleviate load balancing issues, the asynchronous (async) workflow is developed, which is illustrated in Fig. 3. In the async workflow, the actors and learners are decoupled and communicate only when necessary to exchange updated information. Each actors receive an updated episode count and target weights from the learner. The episode count is used for the termination criteria and the target weights are used for inference. Using the updated target weights, an action (control parameter) is generated, which is used to traverse the Markov model (step). The current state, action, reward, and new state are pushed into memory. The value function is updated using the Bellman equation by sampling from memory. The resulting training data is sent back to the learner for training the policy.

The learner updates the episode count and uses the received training data for training the policy and the updated weights are sent to the actors. The training of the policy can either be done using a single thread/process or in parallel using multiple threads/processes using gradient sharing.

Even though the learners and actors are decoupled in the async workflow, there is a momentary lock step between them because of the use of two-sided MPI communication, which can result in a bottleneck when scaling the application on a large machine.

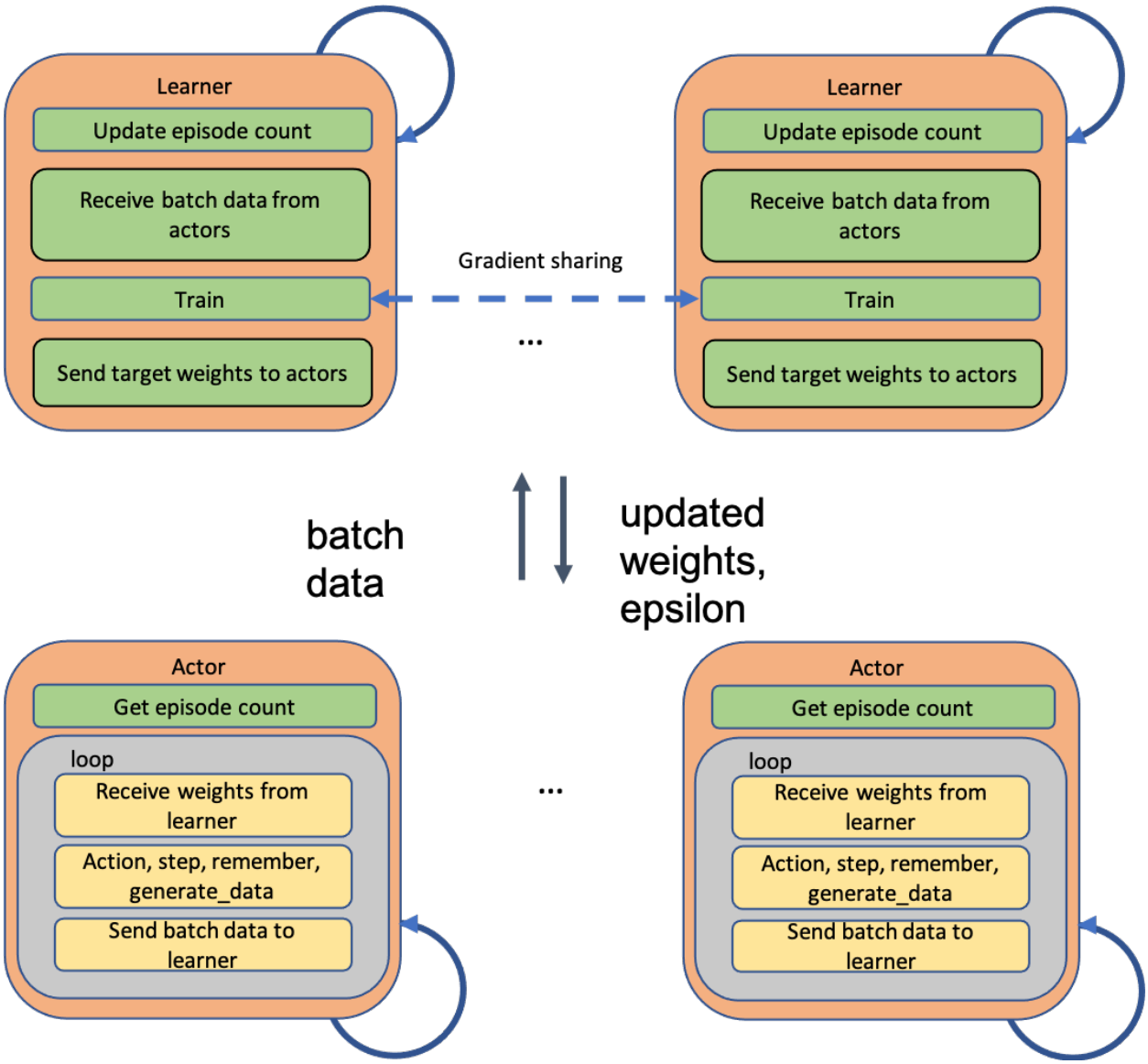


Figure 3: async workflow.

4.1.3 `rma` workflow

The rma workflow is essentially the same as the async workflow with the only change being the use of one-sided MPI communication for exchanging data as shown in Fig. 4. The data is written into an RMA window or "memory pool" and the learners and actors can read/write from this pool, independent of each other. Even though this improves scalability, it introduces some inadvertent algorithmic issues. If either the actors or the learners go faster than the other depending on the complexity of environment simulations/training, it can result in model or experience shear when either the model or experience lags behind the other. This can result in convergence issues but can be mitigated using ordered data structures such as stacks and queues.

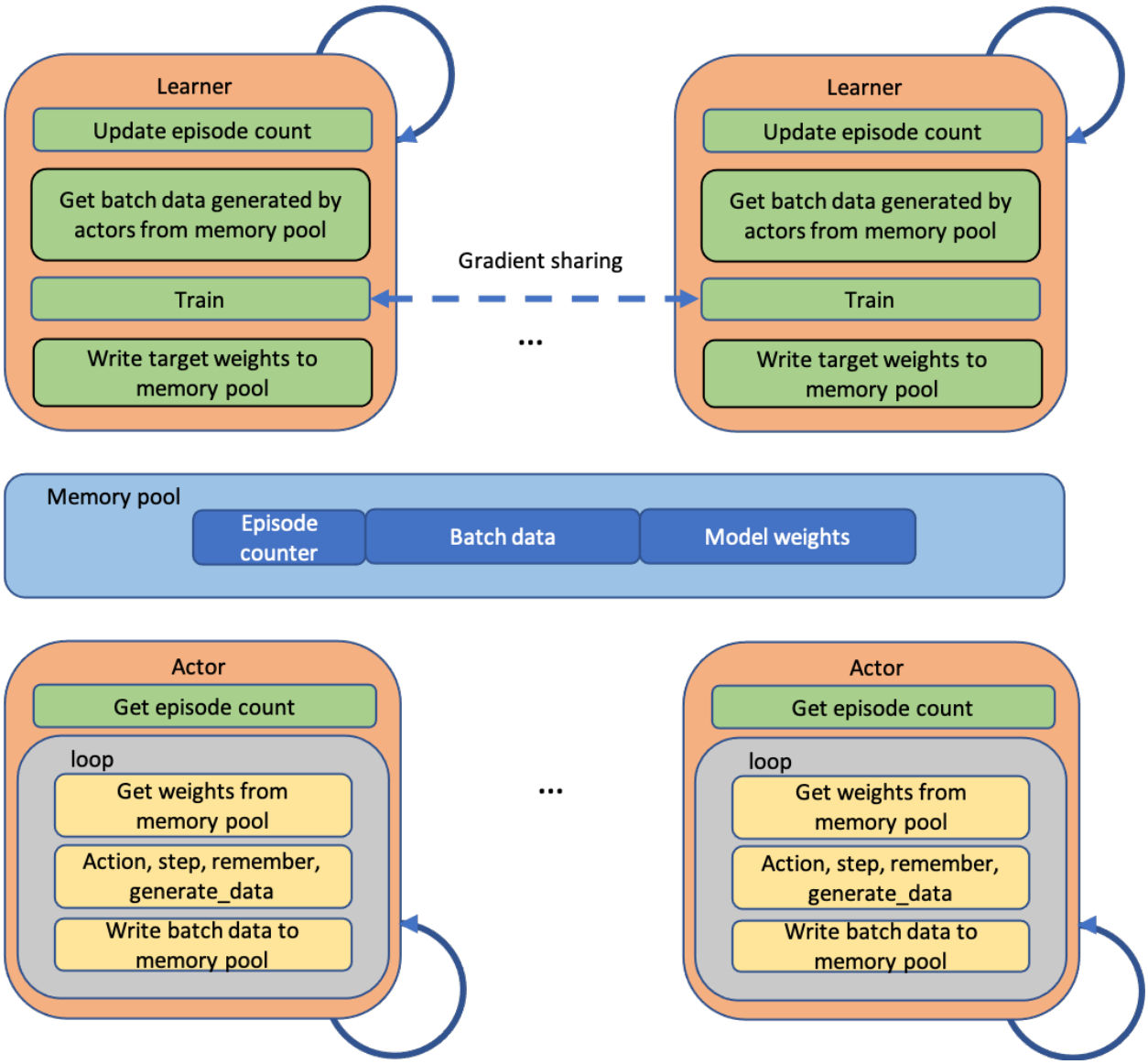


Figure 4: rma workflow.

4.1.4 Results

In the ExaCartPole environment, a pole is attached by an unactuated joint to a cart, which moves along a frictionless track. The cart can move either left or right in a one-dimensional setting and is controlled by applying a force of +1 or -1 to the cart. The pole starts upright (inverted pendulum) and the goal is to prevent it from falling over. A reward of +1 is provided for every time step the pole remains upright. The episode ends when the pole is more than 12 degrees from vertical, or the cart moves more than 2.4 units from the center.

We ran experiments to test the convergence of reward on the Darwin cluster at the Los Alamos National Laboratory and the results are shown in Fig. 5. The `async` workflow was used in this experiment and the figure shows the reward increasing and reaching convergence as the policy is trained. The policy was trained using 1000 episodes, with 200 steps per episode. miniRL does not employ a stopping criteria as of now and runs through all episodes. Note that the y-axis shows

the rolling average of reward, averaged over 25 time steps. There are momentary drops in the reward while training because of the use of ϵ-greedy approach for training, where the agent takes random action for exploration based on the value of epsilon, which is decreased over time to favor exploitation of the learned policy for actions over exploration (random actions). Training the policy for a longer period of time should result in better convergence.

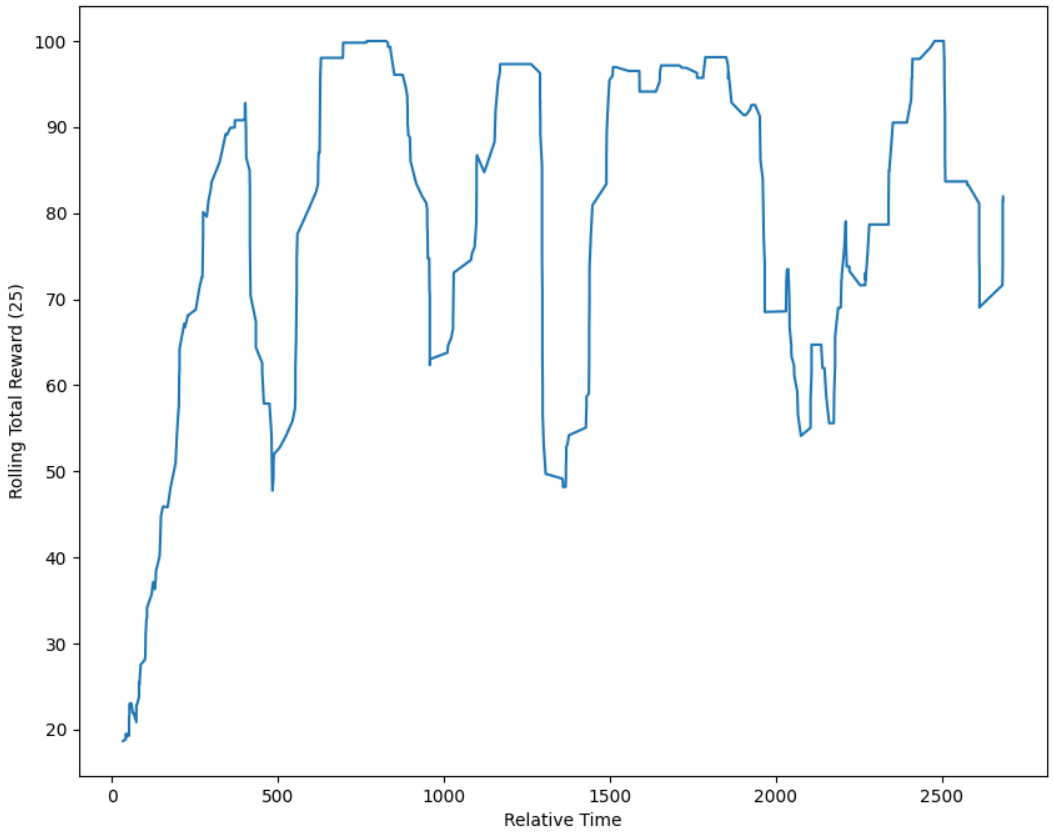


Figure 5: Reward plot shows convergence.

We also conducted scaling tests on the scaling partition of Darwin and the results are shown in Fig. 6. The `rma` workflow was used in scaling studies. miniRL follows a producer-consumer model as the actors are responsible for generating experiences for training. miniRL currently uses a single learner for processing all the experiences (training). Therefore, by increasing the number of MPI processes, we are increasing the number of actors that generate training data and eventually the single learner responsible for utilizing all this data for training becomes the bottleneck. That is why we see diminishing returns when we scale beyond 256 processes. This can be mitigated by using multiple learners.

In the future iterations of this proxy app, we plan to include additional agents and environments. Furthermore, some algorithmic improvements such as V-trace [4] along with ordered data structures would help alleviate the model/experience shear in the `rma` learner.

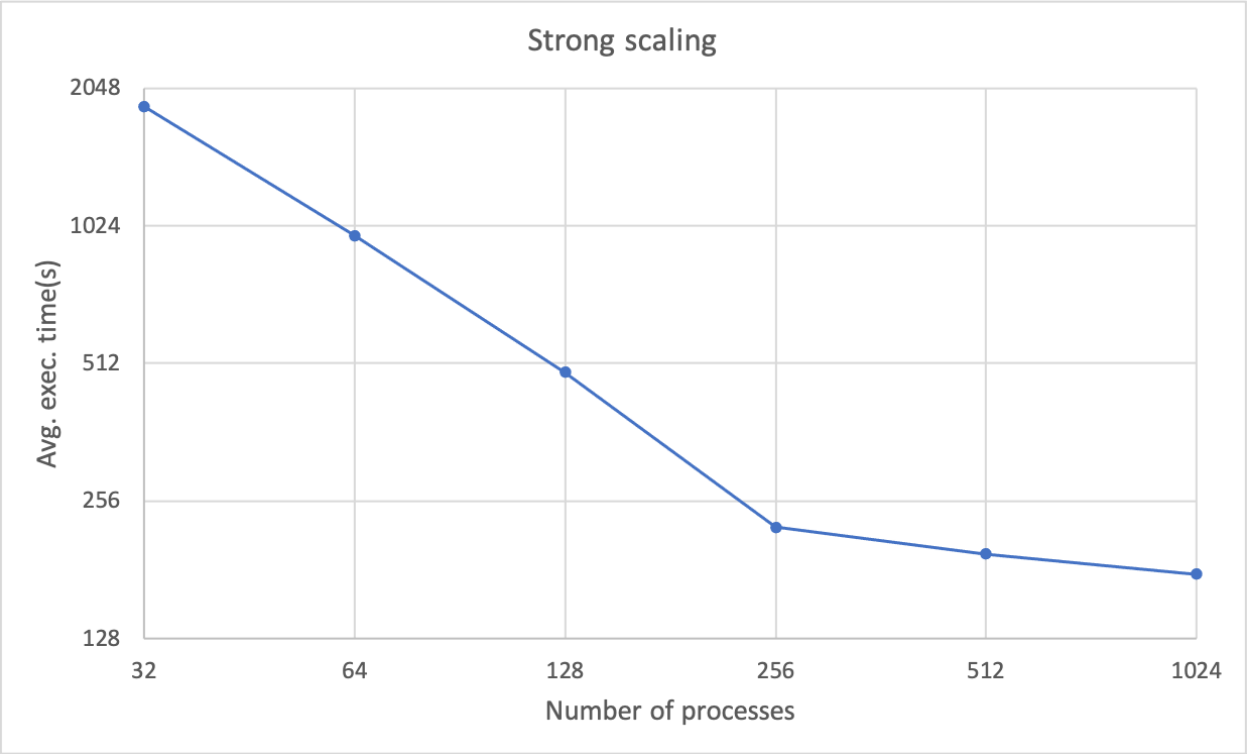


Figure 6: Strong scaling plot of miniRL run on the scaling partition of Darwin.

GitHub: https://github.com/lanl/minRL

5 Gap and Redundancy Analysis

5.1 Overview

One of the most critical aspects of an effective set of proxy applications, is the set's ability to maintain a high fidelity representation of the original workload at a reduced level of complexity. In this gap analysis effort we evaluate a set of 20 applications (mixed parent and proxy) spanning several domains to understand the parent-proxy correspondence and to identify any gaps or redundancies within the proxy set.

5.2 Experimental Methodology

In this section, we present our methodology for collecting data and how we use that data to produce a cosine similarity (COS) matrix. We use two system platforms with different architectures to collect data on 20 proxy and parent applications that span several scientific domains.

5.2.1 Proxy/Parent Application Suite

In this work, we use a suite of 20 total applications that are a combination of proxy/parent pairs with some extra proxies and parents that are not paired. Not all applications we use are proxy/parent pairs because several of the proxies have export-controlled parents, which has complicated data collection, delaying their addition to our suite. Also, Castro is the parent of a proxy called Thornado, but at this point, we are not using Thornado because of I/O issues that caused problems with data collection; we intend to resolve this in future work.

We use the vendor-specific compiler on each platform to compile all of the applications excOpenMC. On the Intel Skylake system, we used icc 20.0.2.254 and OpenMPI 4.0.3, on the IBM Power9 system we used xl V16.1.1 with IBM Spectrum MPI. OpenMC required us to use GNU compilers, 8.3.1 on Skylake and 8.2.1 on Power9.

For each proxy/parent pair, we use the same input problem and/or parameters where possible. In cases where we cannot run the same problem, we use the closest matching problem available and we size both proxy and parent application problems in all cases to use about 50% of the available memory.

Descriptions of the Application and Proxy Codes Tested and Analyzed in This Study
AMG2013 is a proxy application for BoomerAMG and is a parallel algebraic multigrid solver for linear systems arising from unstructured grid problems. We ran the default Laplace problem with a custom resizing. We did not run BoomerAMG because we did not have the expertise to ensure that we treated it fairly, but we intend to use it in the future.

LAMMPS is a classical molecular dynamics code, with particles ranging from a single atom to a large composition of material. It implements mostly short-range solvers, but does include some methods for long-range particle interactions. ExaMiniMD, which is a proxy for LAMMPS, implements limited types of interactions, and only short-range ones.

Laghos is a proxy application that is a high-order Lagrangian hydrocode meant to represent several compressible shock hydrocodes, including BLAST. We did not run BLAST because it is export controlled and we are still working with LLNL to run its experiments on their systems.

QMCPACK is a quantum Monte Carlo package for computing the electronic structure of atoms. MiniQMC covers QMCPACK's essential computational kernels. The computational themes of miniQMC and QMCPACK are particle methods, dense and sparse linear algebra, and Monte Carlo methods.

Vite is an implementation of the Louvain method for (undirected) graph clustering or community detection. MiniVite is a proxy application for Vite that implements a single phase of the Louvain method in distributed memory for community detection.

Nek5000 is a spectral element computational fluid dynamics solver while its proxy application Nekbone solves the Poisson equation with a spectral element multigrid preconditioned conjugate gradient solver.

PENNANT serves as a proxy application for rad-hydro physics-based algorithms on an unstructured mesh, modeling the computation and memory access patterns typical to rad-hydro applications. It is modeled on, and thus serves as a proxy for, an unreleased LANL code.

PICSAR is Particle-In-Cell solver, while its proxy application, PICSARlite, is a subset of the actual codebase.

SNAP serves as a proxy application for discrete ordinates neutral particle transport, modeling the computation and memory access patterns typical to neutral particle transport applications. It is modeled on, and thus is a proxy for an unreleased LANL code..

SW4 is a geodynamics code that solves 3D seismic wave equations with local mesh refinement. SW4lite is a scaled-down version of SW4 that has limited seismic modeling capabilities, but does solve the elastic wave equation and uses some of the same numerical kernels as those implemented in SW4.

The Hardware Accelerated Cosmology Code (HACC) is an N-body framework that simulates the evolution of mass in the universe, with both short and long range interactions. The long-range solvers implement an underlying 3D FFT. SWFFT is the 3D FFT that is implemented in HACC. Since this FFT accounts for a large portion of the HACC execution time, SWFFT serves as a proxy for HACC.

Castro is an adaptive mesh, astrophysical radiation hydrodynamics simulation code.

OpenMC is a Monte Carlo particle transport code.

XSBench is a proxy application for OpenMC and represents the continuous energy macroscopic neutron cross section lookup kernel, which is a key computational kernel of Monte Carlo particle transport.

5.2.2 System Platforms

We want to look at proxy/parent behavior similarity on significantly different platforms, so we chose an Intel Skylake and an IBM Power9 system located at Sandia National Laboratories. The IBM platform does implement graphics processing units (GPUs) on each socket. However, since this work focuses on CPU behavior, we do not include these characteristics. The Intel system runs the RHEL7.8 operating system (OS); RHEL7.6 OS runs on the IBM system.

These architectures seem relatively similar. The Intel architecture is a CISC (complex instructions set computer) and the IBM is a RISC (reduced instruction set computer) which is fundamentally different. However, this difference does not manifest in a fundamental difference in the execution pipeline. They have similar pipeline depths, numbers of execution units, and issue widths. The differences are primarily in the memory subsystem and in SIMD width. The Skylake processor supports up to 512-bit SIMD where the Power9 only supports 128-bit. For about half of the applications we observed similar execution times on the two platforms. For the other applications, we observed significant slowdowns on the IBM architecture. We believe (and IBM agreed) that these applications benefit from the wide SIMD on Intel Skylake and could not attain the same performance on Power9 given the 128-bit wide SIMD support.

We run all of our applications in MPI-only mode and the collection runs are done using 128 ranks, one rank per core, on four nodes. We chose this configuration because it is small enough

to feasibly run large numbers of experiments relatively quickly, yet it is large enough to capture important communication behavior.

5.2.3 Data Collection and Analysis

We use LDMS as the collection infrastructure in all of our experiments. LDMS implements a plug-in architecture, where plug-ins are often engineered to collect data for a particular component or piece of the system. In this work, we use the PAPI sampler, which implements the PAPI API within the sampler to connect to every process (rank) in each application for collection of node-related performance counter data. We carefully examined all of the available performance events on each hardware platform. We functionally tested to ensure that at a minimum, plausible data was returned for each event. We eliminated events that were clearly returning no data or unstable data (i.e. vastly varying) across application runs. In cases where LDMS cannot be used, we have prototyped a method of collecting the data with a wrapper to MPIRUN, which provides substantially similar capability.

We collect on the order of 700 events on each platform for each application; the set of events that are used as input to the cosine similarity algorithm is called a vector. Several runs are required to collect the complete set of data. Data is collected from each application process (i.e., each rank) and we compute an average for each event across all ranks so that all data reflects a per-rank value. Before an application vector is input into COS, we normalize the event counts by cycles executed, which is also a collected event. So all events input into COS are events/cycle.

We want to understand if a proxy comprehensively models the node behavior of the parent, but we also aim to understand where the proxy is and is not a good model of the parent in terms of node components such as cache, TLB, branch predictor, and pipeline. Therefore, we classified each of these 700 events for both platforms into categories corresponding to behavior of specific components implemented in the processor architecture. The categories we use on both architectures are: (1) cache, (2) branch predictor, (3) instruction mix, (4) pipeline, (5) memory, (6) virtual memory, and (7) transactional memory. We perform cosine similarity analysis on each of these categories individually. The cache, pipeline, and memory categories are further broken down, but only because we limit the number of events collected during a single application execution due to software multiplexing. Multiplexing can affect accuracy if the number of events collected is very large. Although this number is application specific (i.e., depends on application behavior), we experimentally determined that if the number of events collected per experiment was 35 or less, the effect on accuracy is negligible. We perform COS analysis for each of these sets of 35 events within each of the component categories. In some cases, these category subsets are related in some way, for example, they may all be front-end pipeline events. In other cases, these subsets may be a mix of events, for example L1 and L2 cache events may be in the same cache subset on which COS analysis is done. We note these cases in the results with labels that reflect the component to which the majority of events pertains.

Because the processors on these systems have significantly different architectures, the available performance events were expectedly very different. Some commonality exists for instance in events associated with the cache subsystem, translation look-aside buffers (TLBs), and the branch predictor. However, the majority of the events pertain to other parts of the pipeline and are vastly different. We analyze the cosine similarity between all the applications on each platform individually, then we combine the two platform vectors for each application (each application vector comprises about 1400 events) and we perform COS analysis to extract better understanding about how the application behavior differs across platforms. Our analysis tools that compute cosine similarity and principal components are written in python and use several of the standard math

libraries (e.g., math, numpy) and scikit-learn facilities (cosine-similarity). We have also developed an extensive infrastructure to automate data collection experiments.

5.3 Results

Below we present cosine similarity data for the application set on both the IBM Power9 and the Intel Skylake. The first pair, Figures 7 and 8 include the results for all counters, while the subsequent pairs, Figures 9–14, show subsets of the data for the cache usage, instruction mix, and pipeline utilization. The chart bodies contain the distance between the application vector pairs defined by the vertical and horizontal axes, in degrees. In particular, one should note the parent-proxy correspondences and the self-congruence along the diagonals.

5.4 Discussion and Recommendations

The results above demonstrate two key insights: (1) that we already understood, namely that a kernel is not particularly representative of a whole code, and (2) more interestingly that dissimilar algorithms can be similar in how they interact with an architecture or a component of an architecture. We can use these properties to determine how well a proxy represents its target application (gaps) and also to discover cases where one or more dissimilar codes have sufficiently similar performance characteristics that one of them may be used to represent the behavior of the group (redundancies). From the complexity in Figures 7–14 one can see that the result is rarely quite as simple as $A = B$, therefore the following recommendations walk through a variety of cases.

- MiniQMC: Figures 7 & 8 show that MiniQMC differs significantly from QMCPack in its overall behavior. While we can see in Figures 11 & 12, that the instruction mixes are fairly similar, Figures 9, 10, 13, & 14 show that these codes differ significantly in how they utilize the cache and pipeline elements of the Intel and IBM processors.
  Recommendation: Be cautious in using MiniQMC for exercises not centered on instruction mix.
- SWFFT: Figures 7–14 show that the correspondence between HACC and SWFFT is not particularly close in any aspect on either processor. The developers use SWFFT because the 3D-FFT it implements is a significant component of HACC.
  Recommendation: For exercises targeting HACC workloads, prefer HACC to the SWFFT proxy.
- Several of the apps/proxies are somewhat unique in their behavior. Specifically, QMCPack, ExaMiniMD/LAMMPS, XSBench, Laghos/Castro, Pennant, and SNAP are fairly unique in how they stress a system and the caches (particularly Intel), as seen in Figures 7–10.
  Recommendation: The uniqueness of these five codes, suggests that their usage in an exercise should depend on the target workload. If the target workload includes one or more among these five codes, they should be included in any final set representing the workload and never down-selected. On the other hand, if the target workload does not contain a component from this group, that component should not be included in the test set.
- Castro and Laghos are fairly similar in their behavior across both processors and all four groups. Recommendation: Despite some moderate differences in instruction mix on the IBM Power9 (leading to some difference in the merged results also), an exercise could choose to down-select to only one of Castro or Laghos if resource pressures required a reduction. For more focused exercises, if a downselect is required, one should minimize overlap in the groups based on the target processor feature being investigated and the target workload.

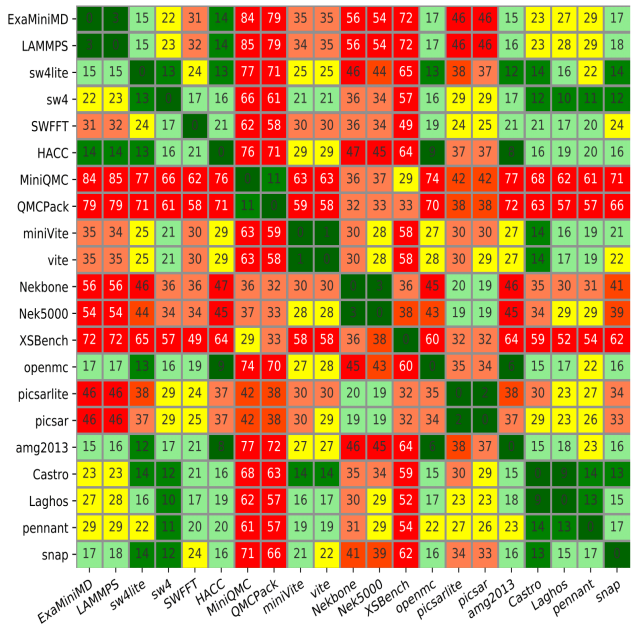


Figure 7: Intel SkyLake merged results

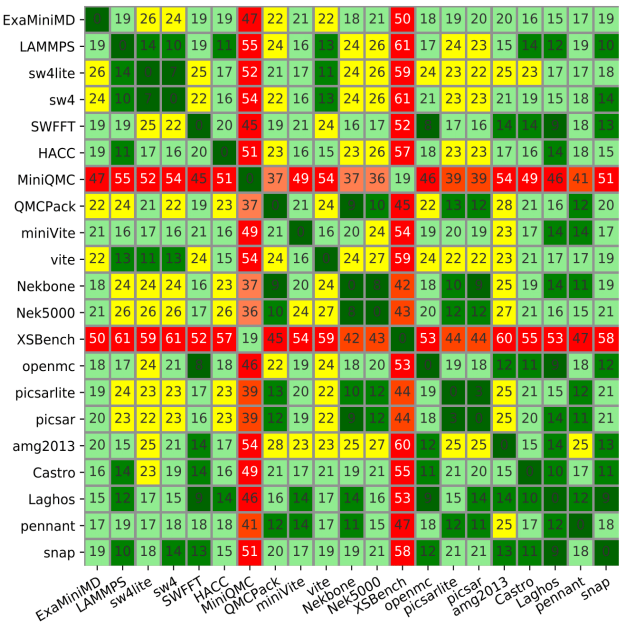


Figure 8: IBM Power9 merged results

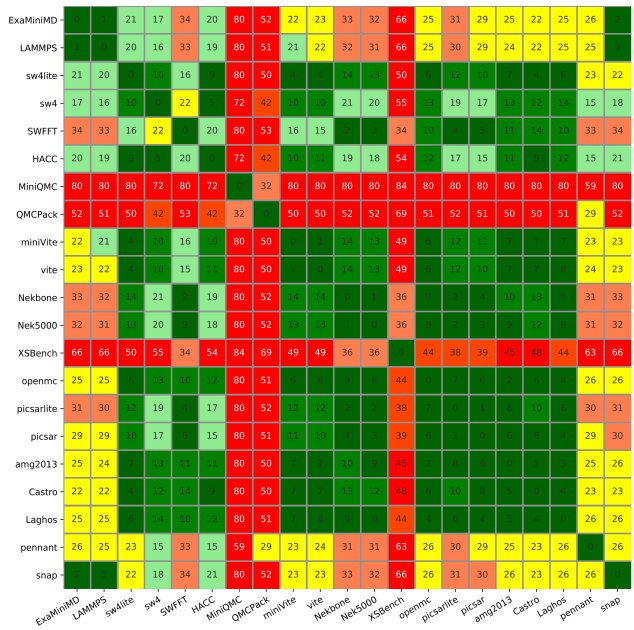


Figure 9: Intel SkyLake cache results

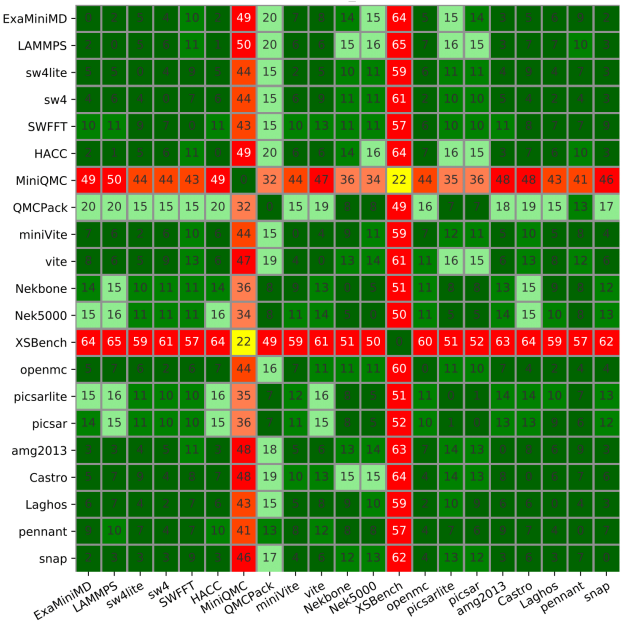


Figure 10: IBM Power9 cache results

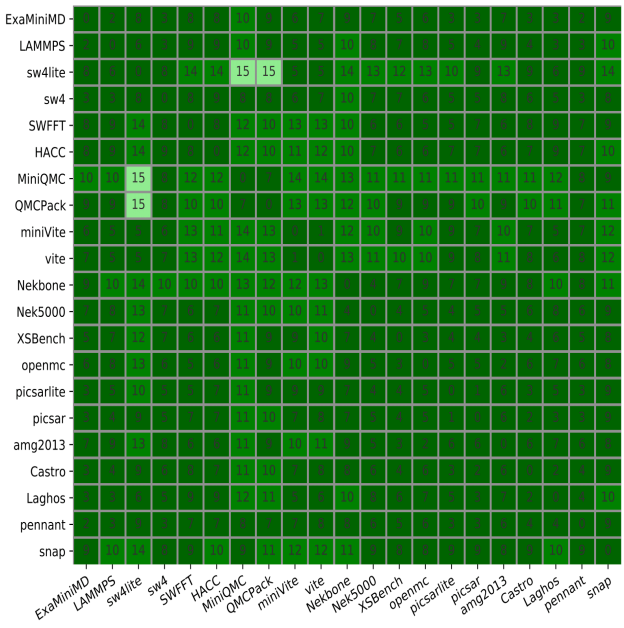


Figure 11: Intel SkyLake instruction mix

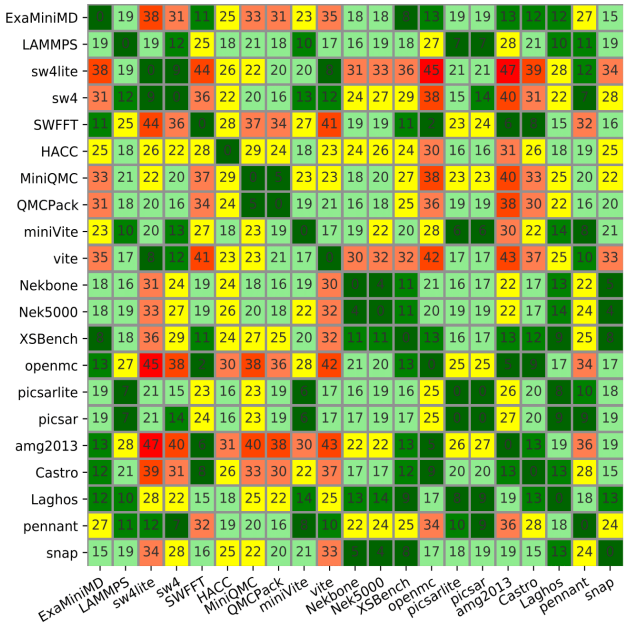


Figure 12: IBM Power9 instruction mix

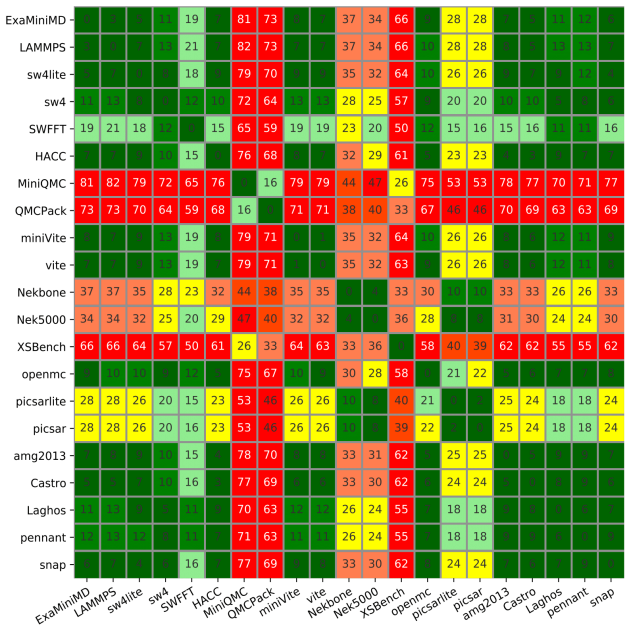


Figure 13: Intel SkyLake pipeline results

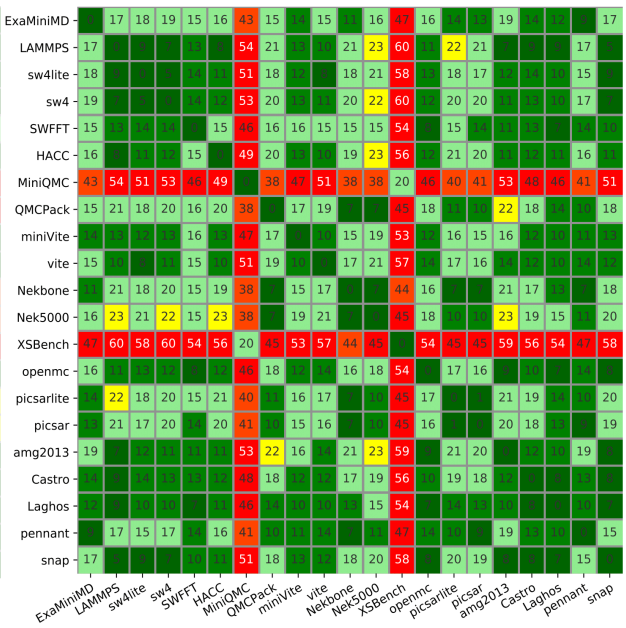


Figure 14: IBM Power9 pipeline results

Recommendation, Cache Testing: For a target workload containing members of one of the 9 line groups below, select at least one member of that line group. Where a line group contains multiple choices, select the one which is most dominant in the target workload (or its proxy) or take the first (which is roughly most representative of the line):

- ExaMiniMD, LAMMPS
- HACC, SW4, SW4lite
- QMCPack
- miniVite, Vite
- picsarlite, picsar, nek5000, Nekbone
- Laghos, AMG2013, Castro
- XSBench
- Pennant
- SNAP

Recommendation, Instruction Pipeline Testing: For a target workload containing members of one of the 6 line groups below, select at least one member of that line group. Where a line group contains multiple choices, select the one which is most dominant in the target workload (or its proxy) or take the first (which is roughly most representative of the line):

- QMCPack
- XSBench
- Nekbone, Nek5000
- SW4lite, SW4, HACC, LAMMPS, OpenMC, miniVite, Vite
- Laghos, SNAP, Castro, AMG2013
- Pennant

Recommendation, Instruction Mix Testing: Because of the variation in instruction mix behavior on the Power9, we recommend against significant down-selecting, except among those applications which are fairly similar along both diagonals (again, most representative first, but workload should override):

- ExaMiniMD
- LAMMPS
- SW4, SW4lite
- HACC
- QMCPack
- Vite
- miniVite
- Nekbone, Nek5000, (XSBench)
- (XSBench)
- OpenMC, (XSBench)
- Picsar, PicsarLite
- Castro, Laghos, AMG2013
- Pennant

– SNAP

5.5 Future Work

This study's results are based on preliminary data collected on each of our applications. Recently, we have increased the volume of data collected to 5 runs per code, and the granularity to single second granularity. While we do not expect major changes in the overall recommendations, the additional datasets will provide some stronger bounds on the uncertainties (currently ∼5%, new estimated closer to 1–2%). The preliminary higher resolution results suggest that we will be able to layer-in additional methodology to provide some automated characterization of kernel-similarity across the entire application set. This analysis already shows some promise for identifying the architectural features that are most determinative for the performance of the aggregate workload, an individual application, or for cross-application similar kernels. Additionally, we anticipate applying these techniques and methods to assist teams in identifying input data sets and problems that are most representative of standard use cases. We also look forward to adding proxies for AI/ML workloads to this study. We hope to update these results in a future report.

A related study for GPU performance is underway for a separate milestone.

6 Development of New Proxy Applications

Members of the ECP Proxy App Team are involved in the developement of a number of new proxy applications. Some, such at the FFT proxies that are being developed in conjunction with the FFTX project are still in the early assessment stage. Others have released initial versions and are now being extended or enhanced. This section describes our development efforts.

6.1 FFT Proxy Applications

Fast Fourier Transforms are used in a broad range of DOE science applications, including ones represented in the exascale applications space. In 2017, the ECP AD leadership team surveyed the applications projects regarding the use of FFTs in their codes. Of the 25 projects, 12 reported current significant use of FFT, and 4 others anticipated it as a possibility in the future. Of the 12 teams that use FFTs, all but two reported that they relied at various levels on community or vendor libraries.

ECP is developing a new FFT library called FFTX [10] to fill in the gap identified in the exascale programming push for a common set of spectral-based algorithms and kernels. FFTX uses symbolic transformation tools, code generation techniques, and autotuning to create exascale-ready high-level FFT packages for multiple applications. To test its implementation, the FFTX team identified four use cases that are representative of FFTs in exascale codes [2]:

1. the Pseudo-Spectral Analytical Time-Domain (PSATD) algorithm to solve Maxwell's equations in WarpX;
2. a plane wave solver for density functional theory, used by NWChemEx;
3. solver for Poisson's equation with periodic boundary conditions, used by LAMMPS and MD-NWChemEx;
4. discrete free-space convolution with Hockney's algorithm, used by RF accelerator modeling and a fast low-communication Poisson solver.

Our plan is to implement a suite of proxies for FFTs that will cover a representative set of use cases beyond these four, from an assessment of ECP and selected non-ECP applications and vendors. Uses cases may vary by dimensionality and size of transforms, real and complex values, and whether or not distributed.

The assessment will cover applications in the ECP projects ExaFEL, WarpX, QMCPack, WDMApp, phase-field modeling and MGmol in ExaAM, CoPA solvers in LAMMPS and HACC, and benchmarking in heFFTe; non-ECP applications such as planewave, BerkeleyGW, and Qbox; and the vendors Intel and AMD.

6.2 Description of FY21 work on the PGAS-FMO proxy

6.2.1 Background

This proxy app represents the Fragment Molecular Orbital (FMO) method in Quantum Chemistry. FMO is a popular method for modeling biological systems, such as proteins and drug-enzyme interactions, owing to its ability to compute thousands of atoms using ab-initio wave functions. This is achieved by first subdividing a system into smaller parts, or fragments. The total energy is then expanded as a series in the fragments starting with the monomers, then the dimers, and so on. This approach derives its efficiency from the neglect of inter-fragment exchange ('nearsightedness' principle)—an approximation managed by careful extension of the series and other model factors. Also relevant in the present context is the inherent parallelism of FMO. A key component of the

implementation is the use of a parallel global address space (PGAS)—a paradigm for distributing data on the fragments over the compute nodes and subsequently providing access. Thus, to the end-user, this proxy provides insights into both the FMO method and the implementation of PGAS concepts. Real FMO applications are highly complex and the present proxy app aims in the first instance to be the simplest possible representation of FMO for the purposes of performance analyses. Among the planned features of the app is the offload of the compute kernel to GPU accelerators using OpenMP directives.

6.2.2 Algorithm Design

The compute kernel comprises the electron-repulsion term of the monomer potential at the Hartree-Fock level, including the medium- and long-range approximations to the coulomb interaction. For the present purposes, only the monomer term is considered as higher terms (dimers, trimers, etc) merely redefine the number and sizes of fragments and are, therefore, algorithmically similar to that of the monomers. The monomer term loops over the fragments assigned to a given compute process, then over all fragments to compute contributions to the local fragment Fock matrices. The density matrix of each remote fragment is obtained via a 'GET' operation to its location in the PGAS. The PGAS implementation employs 'data-server' processes to simulate the one-sided GET with regular two-sided message passing (using MPI communication primitives). In the compute kernel, electron repulsion integrals over gaussian-type functions (GTF) located on different atomic centers are computed and contracted with elements of the density matrix before being summed to elements of the local fragment Fock matrix. Within the integral calculations the gaussian-product factors are tested to avoid small terms. The proxy is written in Fortran-90 and current dependencies include an MPI library.

6.2.3 The Model

Fragments are represented by helium atoms. Thus, the compute load is simulated using a cluster of such fragments. Each fragment/atom hosts a single 'orbital' occupied by a pair of opposite-spin electrons. Each orbital is contracted over a set of s-type GTFs, comprising the fragment basis set. The compute load can be varied through the number and positioning of the fragments, together with the number of GTFs and the various integral screening and approximation cutoffs.

6.2.4 Features of the PGAS-FMO proxy app

- Includes a PGAS API with one-sided 'GET' and 'PUT' capability:
  - Ability to create PGAS array then GET/PUT from/to, resp.
- Regular collective communications:
  - initialize/finalize, global sum (in double precision), broadcast, synchronize.
- Variable number of fragments per compute process.
- Medium- and long-range coulomb approximations
  - Medium-range: intra-fragment coulomb terms (see Eq. 12 of Ref. [8]).
  - Long-range: point-charge model (see Eq. 13 of Ref. [8]).
- Compute kernel is based on the Basic-HF-proxy (includes integral screening).

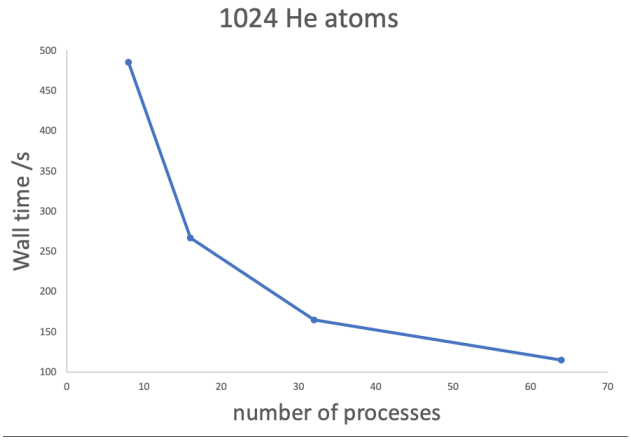


Figure 15: Strong scaling of PGAS-FMO

6.2.5 Status (as of EO FY21)

- First (MPI) version is completed and validated against GAMESS.
- Strong-scaling is demonstrated (see Figure 15).
- GPU offload version is ready, but testing is currently stalled by incompatibilities between the MPI and OpenMP software on our GPU cluster. We are in the process of developing workarounds.

6.2.6 On-going work: variable fragment size capability

This is designed and partly implemented but finalization is on hold while the current software issues (incompatibilities in the MPI and OMP software on our GPU cluster) are being resolved.

6.3 The MinTally Proxy

6.3.1 Background and Motivation

In recent years improvements in computer hardware and algorithms have pushed the boundaries of what is possible using stochastic computational methods. One important example is computing neutron distributions in nuclear reactor cores. Monte Carlo methods are the gold standard for reactor core simulations. However, MC-based neutron flux profiles historically were far too expensive to apply to practical problems. Recent advances, though, have raised the possibility of doing practical, continuous energy Monte Carlo simulations of reactor cores using detailed geometric representations, e.g., for the small modular reactors which are the subject of the ExaSMR project in ECP.

One potential bottleneck of MC is evident in the so-called *Kord Smith Challenge*—a community benchmark that specifies the requirements for a realistic MC reactor core simulation. The Kord Smith Challenge includes approximately 100GB of cross section information and up to 1TB of tally data. These data structures are too large to fit on existing node memory resources, so meeting the challenge requires some form of data decomposition across node. Focusing on the more difficult tally data problem, the most obvious approach is domain decomposition. Domain decomposition, though, adds the complication of requiring particles to be moved in memory when they cross

region boundaries. This is feasible but generally undesirable—it breaks the simplicity of the classic approach and has a poorly understood impact on performance.

An alternative strategy to domain decomposition relies on the large banks of non-volatile memory that have recently become an additional layer of the memory hierarchy on large supercomputers. The Summit supercomputer, e.g., includes 800GB per node of NVRAM, an adequate amount to allow replicated local storage of the tally data structures for most realistic applications. NVRAM is considerably slower than traditional RAM, but write requirements for tallies, e.g. do not require synchronicity, and could conceivably be done efficiently with large, relatively slow write-speed memory. Given the potential benefit of this approach, we have begun a series of numerical experiments designed to estimate its feasibility on realistic reactor benchmark problems. This can in turn serve as a more general case study for the use of NVRAM as a computational approach for accommodating large global data structures locally.

6.3.2 Approach

Our analysis is based on the open source Monte Carlo transport code OpenMC [13] using the community BEAVRS [7] benchmark. BEAVRS (Benchmark for Evaluation And Validation of Reactor Simulations) represents a 4-loop Westinghouse plant with to our knowledge the most details available in the open literature. Core loadings and detector readings were provided by plant operators for the first two cycles of operation with all necessary details (i.e., as-fabricated fuel assembly loadings, burnable absorber pin layouts, operational histories, control rod positions and boron concentration). It provides an excellent demonstration of any computational approach relevant to full core reactor modeling.

The OpenMC code is a large, complex application targeting extreme parallelism and suitable for a wide range of physical phenomena on arbitrary geometric regions. Ultimately, any new approach has to be validated on a realistic benchmark such as BEAVRS using the full OpenMC code. Given the complexity of OpenMC/BEAVRS, though, a key intermediate step is to abstract a simplified version of OpenMC which retains the key performance features of the full code for the components under study—in this case, the performance of the tally system. The proposed proxy app should be informed by data from OpenMC running the BEAVRS benchmark, while still being lightweight and highly simplified, making it far more suitable for testing new hardware features, less mature programming models, and for development by people without expertise in all of the details of neutron physics. Ultimately, the goal is for performance improvements in the proxy app to translate to performance improvements for OpenMC. This requires a careful identification of the abstractions and proper validation relative to the full app.

6.3.3 Develpment of *MinTally*

A new proxy app, *MinTally*, was developed to address the use cases described above. MinTally abstracts the key components of the tallying system from the OpenMC code, allowing a problem to be defined for any user-specified number of tallies, tally filter bins, nuclide inventory, and tally scores. MinTally completely abstracts out the complex tracking physics of the full application, modeling the tallying by looping over a specified number of tally events per particle, randomly choosing cells, and scoring all of the events for each nuclide in the given cell. This is highly simplified compared to the full OpenMC application, which tracks neutron histories through a series of complex interactions and scores tallies in the geometric regions where the collisions or surface crossings occur. For performance modeling, however, it is reasonable to suppose that a random assignment would have similar performance, something which is to directly verified in the following

section. Finally, MinTally is designed to run on a single shared-memory CPU using OpenMP multithreading, a reasonable model for a local, replication based tally system on a distributed memory computer.

6.3.4 Initial Verification

The goal is to use MinTally as a first step in scoping the feasibility of using NVRAM for large tally simulations. Ideally, performance statements about MinTally would roughly translate to performance statements about OpenMC when used with a similar tally configuration, and MinTally optimizations in certain cases could be directly ported to OpenMC. To test this hypothesis we define an initial set of verification benchmarks using the BEAVRS benchmark.

Defining suitable verification benchmarks required modifying BEAVRS to include arbitarily large tallies. To accomplish we 1) added the ability to axially and radially decompose fuel pins to seamlessly increase the number of geometric regions, 2) allowed for an arbitrarily large rather than fixed nuclide inventory, and 3) allowed for the specification of an arbitrary number of scores. This flexibility then allowed us to run OpenMC on BEAVRS with a wide range of tally sizes up to the maximum on-node available memory, enabling the comparison of scaling behavior between the full and mini-app.

A series of numerical experiments were performed using the modified BEAVRS benchmark with OpenMC and instrumenting just the tally operations. A number of simulations were carried out covering a range of tally sizes by adjusting both the number of geometric regions and the number of nuclides in the fuel region. The tally process in the full code is complex and for various reasons its execution time doesn't necessarily scale linearly with the aggregate size of the tally data, so the goal of this comparison is to verify similar scaling behavior (in tally size) between the full app and the proxy.

Figure 16 shows the results of this initial verification study. Simulations are carried out with tally sizes from approximately 1GB up to about 600GB, an appropriate range for realistic application and for the NVRAM use case. The results show a superlinear increase in execution time with tally size for the full application, and a qualitatively very similar behavior for the proxy application for most tally sizes. This is true when both increasing tally size using additional geometric regions (upper two curves) or nuclide inventory (lower two curves). A small but interesting anomaly is the more rapid increase in execution time for the full app at large tally sizes when increasing tallies via nuclide inventories. This and a broader range of performance characteristics will be addressed in forthcoming extended verification exercises.

6.3.5 Ongoing Work

The development and analysis of MinTally is ongoing, and further verification tests are likely required to gain confidence in its usefulness as a performance proxy for large tally OpenMC simulations. After adequate verification and any necessary modifications, the next step is to run MinTally using NVRAM, something which is considerably simpler than doing initial tests and scoping studies with OpenMC. Performance data from these simulations will then guide our strategy for incorporating NVRAM staging directly into OpenMC.

Report EOY 2021 α

EOY Report 2021

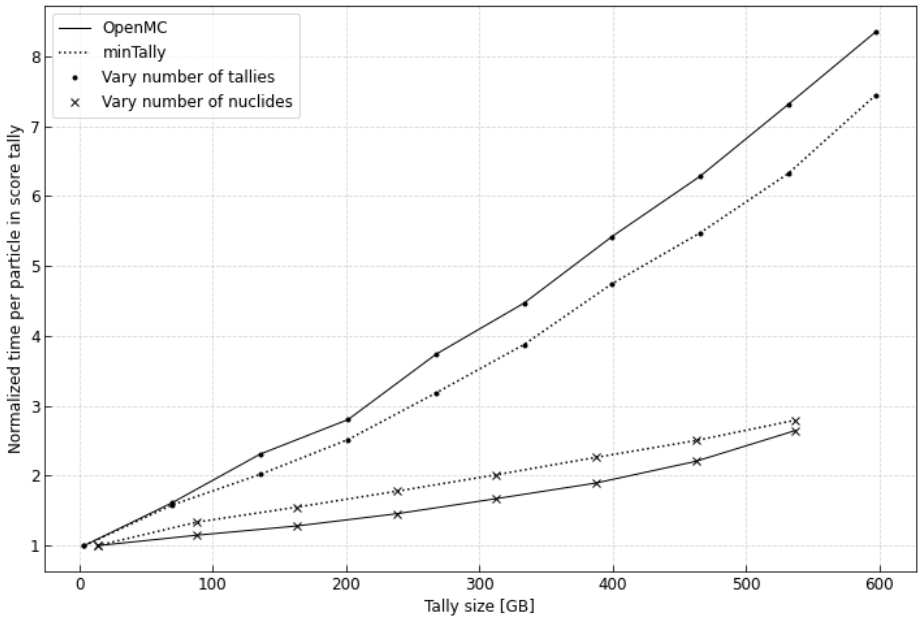


Figure 16: Performance vs. tally size for an initial verification study of MinTally.

6.4 miniGAP: Machine Learning Inter-atomic potentials

Most of the computing cycles in leadership computers are spent in atomistic simulations for materials science and chemistry. The evaluation of energies and forces is ubiquitous in the study of dynamic properties of materials and molecules, although this task is extremely demanding, particularly because long trajectories with a large number of atoms are needed to model realistic scenarios. Reduced scale methods, such as force fields and coarse grain, are useful when several thousands of particles are simulated. These approaches are highly depended on their parametric form and in many cases hardly could include high order energy terms or capture the physics of complex interactions, such as van der Waals interactions or chemical bond breaking. In the last five years, Machine Learning potentials (MLIP) have become very efficient in accurately predicting energies and forces of atomic structures using only a fraction of time of quantum mechanical computations, although their computational cost is higher than for traditional force fields.

Particularly, Gaussian Progress Regression (GPR) and Neural Network models, that are non-linear and non-parametric approaches, have gained traction within adopters of MLIP. GPR is particularly useful when the amount of data is limited. Moreover, GPR could easily estimate the associated error of the prediction, or covariance, which is inherent from the properties of the Gaussian functions. An important advantage of GPR over other regressors is that the former requires a few of hyperparameters for its optimization.

The miniGAP proxy application focuses on GPR and the atomic descriptors that enable prediction of properties and force fields of materials at large scale. Particularly, the Smooth Overlap of Atomic Position (SOAP) descriptor has shown a good performance distinguishing small changes in the 3D structures of molecular systems and bulk materials.

6.4.1 Formulation of the problem

The energy of a material or molecule can be computed as a sum of local energies ε_i of the all the atoms in the system, and written as follows,

$$E = \sum_i \sum_{i \in s} \varepsilon_i^s$$

The individual local energies are composed with the weight factor α_h for each type of atom and the similarity kernel K, which provide the similarity metric between two atoms i and s. This comparison uses a chemical descriptor d_α. This the local energy is defined as

$$\varepsilon_i^s = \sum_s^M \alpha_i K(\mathbf{d}_i, \mathbf{d}_s)$$

In this project we are interested in the SOAP descriptor which is a many-body and takes in account all the possible neighbors within a cutoff radius r_{cut} is defined below as,

$$\rho_i(\mathbf{r}) = \sum_j f_{cut}(\mathbf{r}_{ij}) Exp\left[-\frac{|\mathbf{r} - \mathbf{r}_{ij}|^2}{2\sigma_\alpha^2}\right]$$

SOAP has the property of smooth decayment the after the cutoff distance, and it is defined within the function f_{cut}.

In the practice, the descriptor is approached with a basis set of fixed size. The radial part, $g_n(r)$, could use either Legendre Polynomial or Gaussian Type Orbital functions, and the radial part is taken in account with spherical harmonics Y_{lm}, as

$$\rho_i(r) = c_{nlm}^i g_n(r) Y_{lm}(\hat{r}).$$

In this last equation, the c_{nlm} are the expansion coefficient that regularizes the descriptor.

With the definition of the neighbor density, the SOAP kernel can be estimated as:

$$K(\rho_{soap}\rho'_{soap}) = \sum_{n,n',l} P_{n,n',l} P'_{n,n',l}$$

with the rotationally-invariant power spectrum,

$$P_{n,n',l} = \sum_m c_{nlm}^{i*} c_{nlm}^i.$$

Once the model is trained with the best α_i that reproduce accurately energies and forces, we will apply our model to large systems in molecular dynamic simulations.

6.4.2 Objectives of miniGAP

This mini application will supplement the current portfolio of AI/ML examples for materials and chemistry with an implementation of an accelerated GPR model and SOAP descriptor to enable representative workloads in MLIP training and large scale simulations. One of the most expensive procedures in the training of an MLIP is the evaluation of the descriptors for all the atoms in each structure in a given data set. This is a particularly onerous computation when the derivatives of the descriptor with respect the atom positions are evaluated to include forces in the training. The

second most expensive step is the estimation of the weight factors, which requires algorithms to get eigenvalues of the similarity matrix.

In this project we will implement a fast evaluation of the descriptor, taking as starting point one of the open source implementations of SOAP. We will adopt programming models to enable hardware acceleration and use portable solutions that could help to evaluate performance for more than one architecture. Additionally, we will make use of Tensorflow routines with support for GPU acceleration. A general GPR could be implemented within Tensorflow, and this would be a demonstration for GPR training under exascale hardware.

6.4.3 Proposed Milestones

FY21
1) Competition of mini app for AI/ML. We propose Gaussian Process Regression for properties with a portable programming models such as SYCL and OpenACC.
2) Working in an interface to connect machine learning inference from Gaussian Process Regression model in an exascale mini app such as ExaMiniMD and demonstrate large scale application.

FY22
1) Development of preprocessing data and regression tests in exascale test beds, such as Intel Xe and AMD GPU
2) Testing I/O and inference for extended systems in exascale computers
3) Improvement of building routines.

6.4.4 Current Status

The project has made significant progress in the implementation of GPR as proposed in FY21.1 milestone, we are using Tensorflow and OpenMP for the descriptor generation, FY21.2 is currently delayed. We expect to release a first version of miniGAP (https://git.cels.anl.gov/vama/minigap) during 2021 Q4.

6.5 QTensor-mini: Tensor Contraction Simulator

We developed a proxy app for our specialized Quantum Approximate Optimization Algorithm (QAOA) quantum circuit simulator. 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, our simulator is much faster than ones 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, a particular type of quantum circuit important to Quantum Machine Learning and Optimization problems.

We surveyed the ECP Proxy Apps suite and found that it lacks an application which models the memory and time costs of tensor network contractions. 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.

Our primary goal is to develop a proxy app for QTensor, but it is important to note that tensor networks have widespread application, both in Scientific Computing and Machine Learning. In Machine Learning and Statistical literature, one example of their use is found in Probabilistic Modelling. 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 is vital to achieving the 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, serve 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. Our app requires few dependencies, needing only a BLAS library for each of the targeted processors.

This app should also serve as a soft proxy for ExaTn, a high performance tensor library in development at Oak Ridge National Lab, whom the QTensor team is working closely together with to target upcoming exa-scale supercomputers.

6.5.1 Formulation of the Problem

QTensor is based on representing 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 the vertex from graph and connects all its neighbours.

The cost of each operation of contraction of a vertex depends exponentially on number of neighbours of the vertex. This number is known in graph theory as '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 both in terms of 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 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 optimisation techniques provides benefit only for large tensors with rank approximately of $r \geq 20$.

6.5.2 Objectives of QTensor-mini

We identified the most time-consuming and representative steps in QTensor (like pairwise-tensor contraction in the bucket (eq. 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 the code was released in October 2020.

Since October of 2020, we have worked on porting and optimization of the code for NVidia GPUs with an aim to run it on Summit and upcoming Polaris supercomputers. We also explored different solvers for the optimal contraction order in the tensor network.

The next steps will be to port QTensor-mini to AMD GPUs and Intel GPUs with the aim to get ready QTensor working on upcoming exa-scale supercomputers. This work will be done over the course of 2022.

6.5.3 Current State of QTensor-mini

QTensor-mini has been reviewed by the Argonne legal team, and has been officially released. QTensor-mini is available on Github at https://github.com/Argonne-QIS/QTensor-mini.

During the summer of 2021, we worked closely with NVidia performance engineering team led by Tom Gibbs work to port and optimize QTensor-mini for NVidia GPUs. As a result of this work, we implemented PyTorch and CuPy backends. We benchmarked the code on both CPUs and GPUs. It allowed us to find the optimal allocation of resources to both CPUs and GPUs, although some load balancing issues are still left to be resolved. As a result of this work, we sped up code 140 times on NVidia GPU V100 over the NumPy baseline on a CPU for the benchmarked quantum QAOA circuits. This work was accepted for the presentation and publication in the Second International Workshop on Quantum Computing Software held in conjunction with SC21: The International Conference For High Performance Computing, Networking, Storage and Analysis (https://sc21.supercomputing.org/presentation/?id=wksp127&sess=sess136). We are currently writing the full paper for this workshop proceedings to be published in IEEE digital library Xplore (https://ieeexplore.ieee.org/Xplore/home.jsp). The next step will be to write the backend for new NVidia tensor network library cuTensor, and optimize code for A100 GPU. Once that is done, we will run large-scale performance studies on the Polaris supercomputer and submit the technical paper for the Supercomputing conference in 2022.

6.5.4 Timeline

We are proposing the following updated timeline for this project:

1. Fall of 2021 and Spring of 2022: We will explore the use of various node elimination algorithms to find optimal tensor network contraction order in the bucket. In particular, we will use a multiscale approach as an effective method to reduce the complexity of the problem by using various coarsening frameworks. We currently have Greedy and Tamaki backend solvers implemented. KaHyPar backend was just implemented. Zoltan+AD/Agg and HMetis backends will be implemented in the Fall of 2021. Working with the Zoltan package is of special importance given that it is also an ECP project.
2. Summer of 2022: We will develop the backends 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.
3. Summer and Fall of 2022: We will explore the use of the developed simulator to advance quantum information science in U.S.. In particular, we will showcase how we can run large-

scale quantum circuits for combinatorial optimization, material design etc. problems on the Exa-scale supercomputers Aurora and Frontier. This will result in a number of Gordon Bell submissions at the Supercomputing conference and high profile peer-reviewed publications.

6.6 HyPar

6.6.1 Introduction

HyPar, which staands for Hyperbolic-Parabolic (with Source) Partial Differential Equations Solver, is a finite-difference framework to solve any general hyperbolic-parabolic set of partial differential equations (with source terms) on Cartesian (structured) grids. The initial rationale for the creation of HyPar was to provide a variety of spatial discretizations and temporal integration schemes, file input/output routines (to handle files in a few different formats), as well as a parallel implementations, to evolve a system of PDEs using MPI. Of the spatial discretizations that are available in HyPar, the WENO and compact reconstruction WENO (CRWENO) schemes have been use as the "solver", or engine, to evolve the compressible Navier-Stokes equations for shock dominated flows. While HyPar is self-contained, it can also be used in conjunction with PETSc.

With the move to using high-order spatial discretizations for direct numerical simulations (DNS) and large eddy simulations (LES), there is a need for a ProxyApp for simulating compressible flows. The WENO and CRWENO schemes have become popular for computing flows with shocks, owing to low dissipation and dispersion properties in these schemes. The CRWENO scheme, in particular, has the twin advantages of being a robust high-order scheme, and also one that has a compact stencil. The CRWENO scheme, however, requires the inversion of a tri-diagonal matrix to determne the coefficients of the spatial discretization. The development of a sparse matrix solver for the tri-diagonal system of equations, on GPUs is a pacing item, with the added bonus that the usability of a scalable (on GPUs) sparse-matrix solver extends beyond simulating the compressible Navier-Stokes equations.

Our aim in this project, therefore, was to extend the usability of HyPar to solve the compressible Navier-Stokes equations on heterogeneous computing platforms (i.e., CPU+GPU) by building into it the ability to offload computations from the CPU to the GPU using SIMD paradigms, such as Cuda and DPC++, that are available in the Nvidia NVHPC and Intel oneAPI programming environments. Two problems have been identified, namely the rising bubble problem, which simulates the rising of an air bubble in a liquid, and a shock boundary layer interaction (SBLI) at high Mach numbers. We believe that by using the WENO and CRWENO schemes in HyPar to simulate the two model problems on heterogeneous computing platforms (e.g., Summit, Aurora, Frontier), HyPar would be an effective ProxyApp for compresssible flow solvers. HyPar could either serve as an engine for other existing finite-difference compressible flow solvers, or one could choose to extend HyPar to be a full-fledged flow solver, for evolving compresssible flows, in commplex geometries on heterogeneous computing platforms.

6.6.2 Equations

HyPar solves the following partial differential equation (PDE) using a conservative finite-difference algorithm on a Cartesian grid:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{F}_{\text{hyp}}(\mathbf{u}) + \mathbf{F}_{\text{par}}(\mathbf{u}) + \mathbf{F}_{\text{sou}}(\mathbf{u}) \tag{2}$$

where $\mathbf{F}_{\text{hyp}}$ is the hyperbolic term, $\mathbf{F}_{\text{par}}$ is the parabolic term, and $\mathbf{F}_{\text{sou}}$ is the source term. Each of these is discretized in space to obtain the following semi-discrete ordinary differential equation

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 41.55   4135.66  4135.66   420000     9.85     9.85  WENOFifthOrderCalculateWeights
 35.06   7625.99  3490.33  2100000     1.66     1.66  Interp1PrimFifthOrderWENO
  9.44   8565.83   939.84   420000     2.24     2.24  NavierStokes2DUpwindRusanov
  4.90   9054.02   488.19   490000     1.00     1.00  NavierStokes2DPreStep
  2.19   9271.77   217.75   210000     1.04    39.65  HyperbolicFunction
  1.65   9435.90   164.13   420000     0.39     0.39  NavierStokes2DModifiedSolution
  1.38   9573.51   137.61   210000     0.66    46.43  TimeRHSFunctionExplicit
  1.31   9704.18   130.67    70000     1.87   141.16  TimeRK
```

Figure 17: Profiling results of HyPar over 2D RisingThermalBubble example

(ODE) in time:

$$\frac{d\mathbf{u}}{dt} = \hat{\mathbf{F}}_{\text{hyp}}(\mathbf{u}) + \hat{\mathbf{F}}_{\text{par}}(\mathbf{u}) + \hat{\mathbf{F}}_{\text{sou}}(\mathbf{u}) \tag{3}$$

where $\hat{(\cdot)}$ represents the spatially discretized terms. The governing PDE can be of any space dimension.

In this report, we describe our optimization strategy (removing data transfer between the host and the device) to efficiently implement HyPar on GPUs and present initial computational results. We have achieved about 60 times speedup so far by exploiting GPUs. We note that all experiments on GPUs in this report have been performed using Nvidia's V100.

6.6.3 Accelerating HyPar using GPUs

Our main goal is to significantly accelerate the computation time of HyPar using GPUs. To this end, we first profiled the existing CPU-based HyPar code using profiling tool gprof to identify the current computational bottlenecks. This is described in Section 6.6.4 We then implemented the most compute-intensive routines on GPUs as described in Section 6.6.5. Our observations are that i) GPUs can significantly accelerate computation time (up to 160 times faster) and ii) reducing or removing data transfers between the host and the device is critical to obtaining a high speedup of computation time. Based on these observations, we started to implement the entire HyPar routines on GPUs to remove the need for data transfers of intermediate results. Section 6.6.6 presents the current status of our implementation.

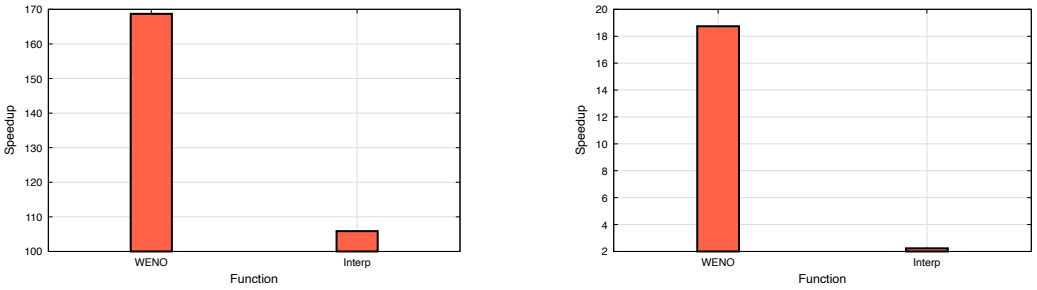
6.6.4 Profiling HyPar

Figure 17 is the profiling results obtained from solving a 2-dimensional RisingThermalBubble example. In Figure 17, we observed that the fifth-order WENO computation and its interpolation routines take more than 76% of the total computation time. The results indicate that if we accelerate those two routines on GPUs, then we will obtain a significant speedup.

The computational procedure for those two routines is suitable for us to rewrite it using SIMD instructions only: they are defined for each grid point; there is no data dependency between grid points in computation; and the same instructions are executed over different grid points. We have implemented these two routines on GPUs as described in Section 6.6.5.

6.6.5 Initial computational improvement on GPUs

Figure 18 shows speedup of computation time of those two routines when they are implemented on GPUs. Figure 18(a) measures the computation time only without including the time for data transfer between the host and the device. We have observed about 165 times and 105 times speedups



(a) Computation only (b) Computation and copy

Figure 18: Speedup of the top 2 routines on GPUs

for the WENO calculation and interpolation routines, respectively. However, if we include data transfer time (the time to copy function values at grid points from the host to the device plus time to return the interpolation results back from the device to the host), then speedups significantly degraded: from 165 and 105 to 18 and 2 for weights computation and interpolation, respectively. These results present that data transfer takes a significant amount of the total computation time. Therefore, reducing or removing it will largely affect the total computation time.

6.6.6 Removing data transfers between CPU and GPU

Although reducing the time for data transfer is critical to obtaining a high speedup, we may not be able to avoid such transfer as long as we have routines that work on the host and require data computed from GPUs. Certainly, we have to copy the initial function values at grid points from the host to the device and return a solution computed from the device back to the host at the end. Except for those cases, we may be able to avoid data transfer for all the intermediate results by implementing the relevant routines on GPUs.

Figure 19 presents the current speedup results as we implement those routines on GPUs. Since we avoid data transfer for intermediate results, we note that the speedups for WENO calculation and interpolation have restored back to 165 and 105, respectively, compared with Figure 18(b) where speedups were 18 and 2 because of the time for data transfer. From the results, we observed that speedup was proportional to the computational intensity of the original CPU code: the top four routines in Figure 17 correspond to the first four routines from the left in Figure 19, where they showed more than 100 times speedup; and less speedup was achieved for relatively easy part. On average, we obtained about 60 times speedup by using GPUs.

6.6.7 Future work for FY22

Based on our initial computational results, we are convinced that reducing data transfer is critical to obtaining a high speedup. Toward this goal, we aim to achieve the following during FY22:

- Implement the remaining routines on GPUs to avoid data transfer of intermediate results between the host and device.
- Implement distributed computation using GPU-aware MPI. Analyze the impact of communication and communication frequency on the total computation time and convergence, respectively.
- Port the Nvidia-based GPU code to use Intel's oneAPI.

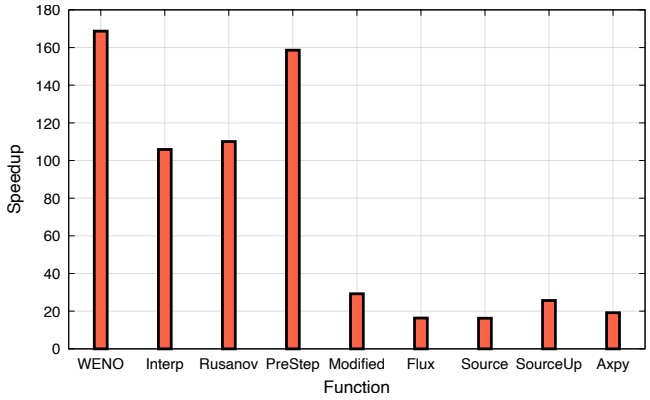


Figure 19: Speedup of computation time

- Investigate a framework that integrates our PDE solver with machine learning (ML) techniques for realizing synergies between ML and PDE solver.

6.7 The IMEXLBM Proxy App Suite

6.7.1 Introduction

The Lattice Boltzmann Method (LBM) [6] is a relatively novel approach to solving the Navier-Stokes equations (NSE) in the low-Mach number regime and its governing equation can be derived from the Boltzmann equation after discretizing the phase space with constant microscopic lattice velocities. One major drive behind the use of LBM in the CFD community is the ease of parallelization, but the increasing popularity of LBM can also be attributed to its unique feature: LBM solves a simplified Boltzmann equation that is essentially a set of first-order hyperbolic PDEs with constant microscopic lattice velocities, for which a plethora of simple yet excellent discretization schemes [9] are available. Furthermore, all the complex non-linear effects are modeled locally at the grid points. Thus, the overall numerical scheme becomes extremely simple and efficient. LBM can deliver better solution quality than a comparably discretized NSE solver at substantially lower computational cost. Furthermore, the linear advection part of LBM can be solved exactly due to unity CFL property. The absence of dispersive and dissipative errors in LBM makes it an ideal choice for the simulation of turbulent flows, interfacial flow, and acoustics, in which conservation of kinetic energy and isotropy play a crucial role. The highly scalable IMEXLBM ProxyApp suite that can be run on a variety of platforms is a first step towards the development of other full-fledged LB codes, and is a much needed building block for simulating multiphase flows on exascale computing platforms at Argonne National Lab and the other DOE laboratories.

6.7.2 Motivation

This project aims to develop a Lattice Boltzmann Method (LBM) proxy application code for heterogeneous platforms (such as Summit, Aurora, and Frontier). A proxy app, by definition, is a proxy for a full-fledged application code that simulates a wider array of problems. The IMEXLB proxy app, therefore, is a small self-contained code unit, with minimal dependencies, that will demonstrate both the science as well as the route to achieving parallel performance on a heterogeneous computing platform for one or more exemplar problems. For this project, in particular, the IMEXLBM proxy app suite will consist of Fortran and C++ codes that will simulate two-phase

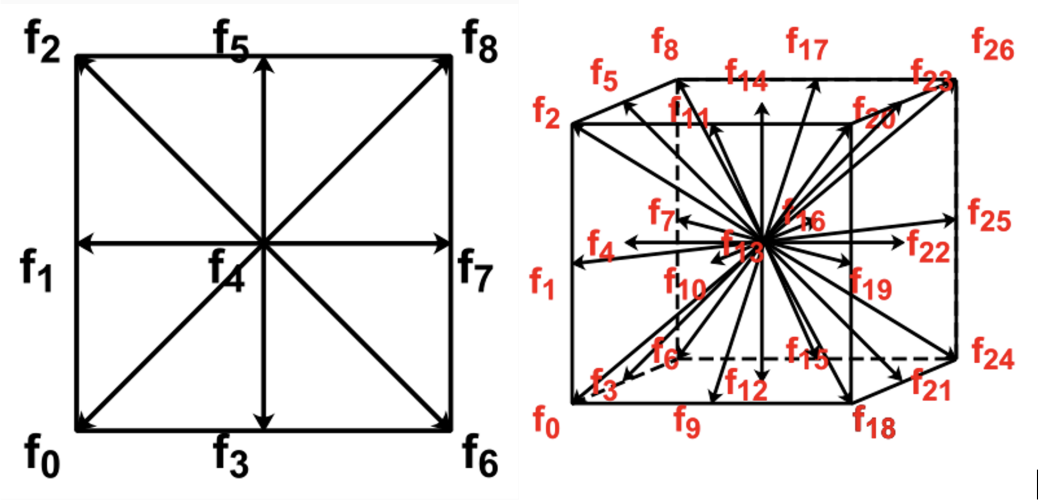


Figure 20: D2Q9 model (left) and D3Q27 model (right)

canonical flows using the MPI+X parallelization paradigm, where X denotes the SIMD parallelization paradigms: OpenMP-4.5+/Kokkos/DPC++/RAJA. This proxy app will serve as a reference for other LB code developers who may wish to either use IMEXLB as a building block, or follow the SIMD parallelization strategy to modify their own code, to do their simulations on platforms like Aurora/Frontier/El Capitan.

6.7.3 Code Structure

IMEXLBM is written in Fortran 90 and employs the 9-velocity lattice model in 2D and the 27-velocity lattice model in 3D [11] (see Figure 20). It is parallelized by the Message Passing Interface (MPI) and OpenMP 4.5 for GPU offload. The MPI setup distributes the computing tasks to multiple CPU processors, and each CPU processor corresponds to a GPU device, therefore 1 MPI-rank is bound to 1 GPU at any given moment in the simulation. With the help of OpenMP offloading directives, the data are offloaded to GPU devices to achieve hybrid parallel computing. Figure 21 depicts the schematic diagram of the parallelization of IMEXLBM.

The Lattice Boltzmann Method (LBM) in the code is implemented by executing the subroutines for collision, streaming, boundary condition setup and post processing at each time-step. At each time-step, the particle distribution functions (PDF) are updated, which in turn recovers macroscopic fluid properties such as pressure and velocities. An additional set of PDF is introduced to minimize the possibility of incorrect data access in non-local parallel operations. Subroutines for outputting data and monitoring the simulation are also called as required by the user.

6.7.4 Components of IMEXLBM

IMEXLBM is capable of handling complex curved boundaries by imposing interpolated bounce-back condition on a background 2D rectangular or 3D cubic uniform grids. The background computational domain is domain decomposed by cutting each dimension into several sub-domains. The number of generated domains is the same as the number of CPU processors. The setup of MPI is done by the subroutines in module “cart_mpi”.

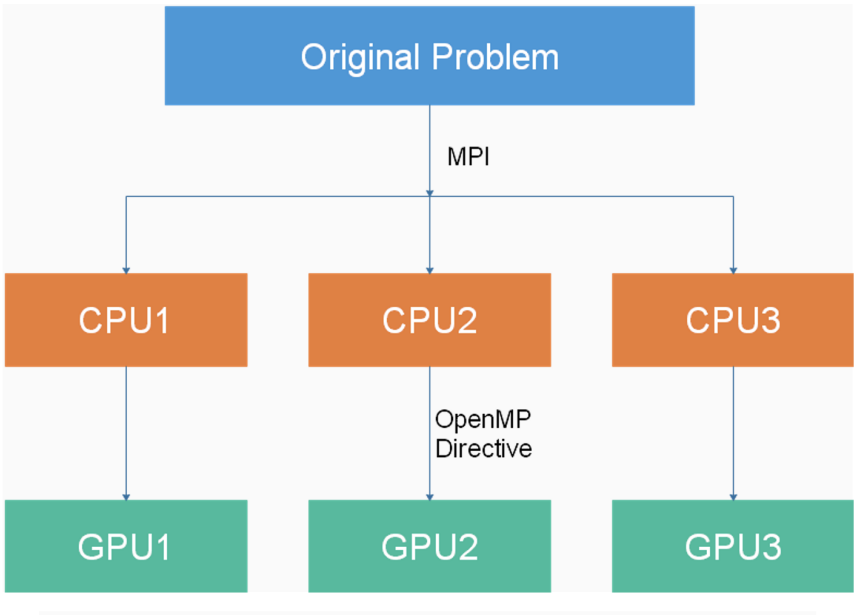


Figure 21: The schematic diagram of the hybrid parallelization

SetCartesian Subroutine This subroutine generates a cartesian MPI communicator. It optimizes the number of processors in each dimension. This subroutine determines the size of arrays defined on each CPU processor. All processors have similar computational loads.

SetLocalLength Subroutine Based on the communicator generated in "SetCartesian" subroutine, this subroutine generates the local array information, including the local array length in each direction, and the positions of starting and ending points of local arrays in the whole domain.

SetNeighbors Subourtine This subroutine defines the neighbors of the current MPI processor and generates sending and receiving subarray datatypes for the communications with these neighbors.

PassF Subroutine This subroutine is called when the communication of the Particle Distribution Function (PDF) Arrays among different CPU processors is needed. The communication is based on the sending and receiving subarray datatypes generated in "SetNeighbors" subroutine.

6.7.5 Data Offloading with OpenMP-4.5+ Directives

IMEXLBM is accelerated by GPU devices. The data offloading and GPU parallelization are achieved by OpenMP directives. These directives perform the following three tasks.

Selection of Device The device number is related to the rank of CPU processor such that each CPU processor offloads data to different GPU devices. The GPU ID is calculated by MOD(RANK, NUM_DEVICES), where NUM_DEVICES is the number of GPU devices on each node of the cluster.

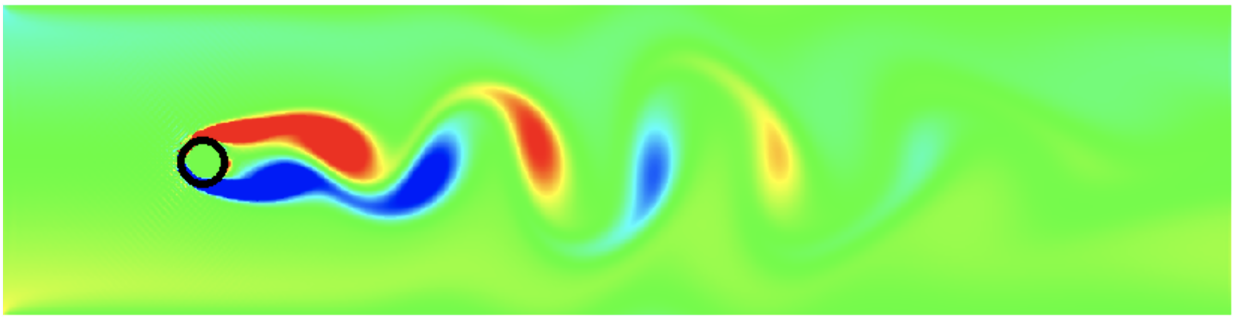


Figure 22: Vorticity field of 2D cylinder flow obtained by IMEXLBM code

Parallelization on GPU Devices The directive `!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO` is used for the Do loop blocks in IMEXLBM subroutines. Reduction operations are required for taking the maximum or the sum. The overall algorithm is local, and thus no data mapping is necessary.

Communication The communication of data is required in the streaming, post processing, and the monitor subroutines. After LBs arrays for PDFs are allocated, they are mapped to the devices by the directive `!$OMP TARGET DATA`. Before the PassF subroutine is called, the PDF data should be updated on CPUs; and after the PassF subroutine is called, the PDF data should be updated on GPUs. The macroscopic properties should be updated on CPUs before data output.

6.7.6 FY21 Outcomes

Both 2D and 3D simulations have been *successfully* implemented, and the proxy app has been tested for correctness (of the simulated results) agains the CPU-only reference version. IMEXLBM can be compiled by Intel OneAPI and Nvidia compilers. The applications can run on Argonne National Lab's Bebop Linux cluster, Ludwig at City College of New York, and Penzias at CUNY HPC center, when the target devices point to CPUs. MPI I/O is employed to accelerate the writing process. The output data format is binary TECPLOT, which can be easily read by visualization software like Tecplot and Paraview (see Figure 22).

6.7.7 Technical Issues and Failure on ThetaGPU Nodes

The code is also tested on the GPU cluster ThetaGPU at LCF, Argonne. However, it is not running as expected when the target devices are GPUs. It turned out that the failure happened in the host-to-device related directives, where memory is not being allocated as intended on the GPU. We have tracked the source of error to the OpenMP directives for allocating and transferring data. A simpler code with functional syntax for GPU offloading directives is currently being tested and will be used to remedy the IMEXLBM code. Data exchange among CPUs will also be replaced by data exchange among GPUs to make the code more efficient.

FY21 Deliverable The Fortran 90 version of the IMEXLBM code has been developed. We plan to release v1.0 of this proxy app in November 2021, together with an ArXiV report detailing the code structure and code performance.

6.7.8 Future Work

Our efforts will continue with fixing the data offloading problem, and simultaneously, we will explore ways to make efficient use of the memory subsystem to ensure fast execution of the so-called LB-streaming and collision steps. Verification and validation of our implementation will be performed on standard CFD benchmark problems. Kernel performance will be assessed on available GPU-based architectures at Argonne. Using knowledge gained through the Argonne-Intel Center of Excellence (COE) partnership, we will identify modifications to the code to ensure optimal performance of the IMEXLBM code. The code will also be rewritten in C++ language. New milestones and scope-of-work will be developed for FY22.

7 Acknowledgments

We are grateful for the guidance and advice of Dr. Emil Constantinescu on the development of the HyPar ProxyApp, and Dr. Saumil Patel and Prof. Taehun Lee on the development of the IMEXLBM ProxyApp. We also wish to recognize the contributions of Angela Chen and Sean Stafford, who worked as summer interns at Argonne, on the development of the ProxyApps QTensor-mini and mini-Gan, respectively.

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.

References

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

[2] Phil Colella, Brian Van Straalen, Andrew Canning, Peter McCorquodale, Doru Thom Popovici, Franz Franchetti, Daniele Spampinato, Anuva Kulkarni, Tze-Meng Low, and Mike Franusich. September 2019 ECP ST Project Review, ECP Project FY20 WBS 2.3.3.07, FFTX. 2019. URL: https://confluence.exascaleproject.org/display/REV2019ST/Sep+2019+ECP+ST+Project+Review+Schedule?preview=/69011020/73401722/Sept2019-Sparse-FFTX-Review.pdf.

[3] 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.

[4] 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.

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

[6] X. He and L. S. Luo. Theory of the lattice boltzmann method: From the boltzmann equation to the lattice boltzmann equation. *Phys. Rev. E*, 56:6811–6817, 1997.

[7] N. Horelik, B. Herman, B. Forget, and K. Smith. Benchmark for evaluation and validation of reactor simulations (BEAVRS), v1.0.1. In *Proc. Int. Conf. Mathematics and Computational Methods Applied to Nuc. Sci. & Eng.*, Sun Valley, Idaho, 2013.

[8] Tatsuya Nakano, Tsuguchika Kaminuma, Toshiyuki Sato, Kaori Fukuzawa, Yutaka Akiyama, Masami Uebayasi, and Kazuo Kitaura. Fragment molecular orbital method: use of approximate electrostatic potential. *Chemical Physics Letters*, 351(5):475–480, 2002. URL: https://www.sciencedirect.com/science/article/pii/S0009261401014166, doi:https://doi.org/10.1016/S0009-2614(01)01416-6.

[9] S. Patel, M. Min, K. C. Uga, and T. Lee. Spectral-element discontinuous galerkin lattice boltzmann method for simulating natural convection heat transfer in a horizontal concentric annulus. *Computers & Fluids*, 95:197–209, 2014.

[10] FFTX project. FFTX Home. 2021. URL: https://commons.lbl.gov/display/FFTX/FFTX+Home.

[11] Y. H. Qian, D. d'Humieres, and P. Lallemand. Lattice bgk models for navier-stokes equation. *Europhys. Lett.*, 17(6):479–484, 1992.

[12] 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.

[13] Paul K. Romano, Nicholas E. Horelik, Bryan R. Herman, Adam G. Nelson, and Benoit Forget. OpenMC: A state-of-the-art Monte Carlo code for research and development. *Ann. Nucl. Energy*, 82:90–97, 2015. doi:10.1016/j.anucene.2014.07.048.

[14] 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.