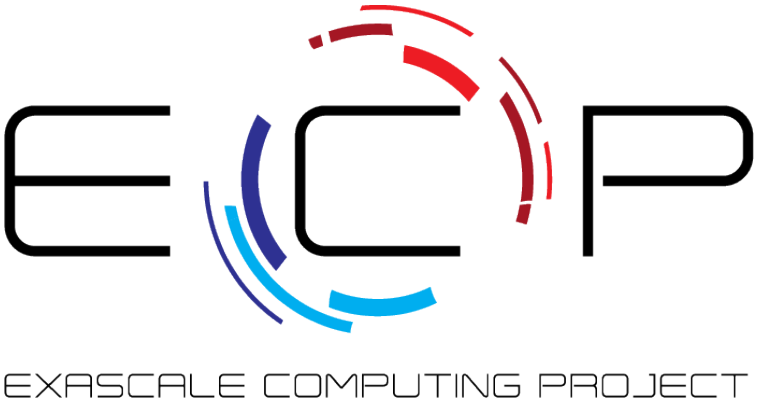
FY22 Proxy App Suite Release

Report for ECP Proxy App Project Milestone ADCD504-14

Jeanine Cook[2], Omar Aaziz[2], Yuri Alexeev[3], Ramesh Balakrishnan[3], Graham Fletcher[3], Christoph Junghans[4], Youngdae Kim[3], 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

December 2022



SAND2023-11615R

Contents

1 Executive Summary

The FY22 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 10–15 proxies that balance breadth of coverage with ease of use and quality of implementation. Also designate approximately 6–10 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 6.0 of the ECP Proxy App Suite has changed very significantly compared to Version 5.0. Although the proxies included in the last suite were useful and have been thoroughly studied, moving forward, we chose to curate a proxy suite that is suitable for system architectures that rely heavily on GPUs to provide the majority of performance. GPU-based systems are becoming prevalent and we need a proxy app suite that can support co-design and acquisition of these systems. Section 2 contains a description of the current contents of the suite.

We have also updated the ML Proxy App Suite to Version 6.0. We have not only added proxies, but have updated some proxies to include additional capabilities. Section 4 describes these new and updated proxies.

We have added a several new proxies to the catalog this year. We maintained this catalog of ECP-and DOE-relevant proxy applications through this final year of the project. This catalog now contains over 80 entries. Brief descriptions of the proxies that have been added to this collection since our last release are provided in Section 3.

Finally, the team has continued to develop the proxy apps that were initially developed and reported on in FY21. Many of these proxies have been updated with scientific simulation capabilities and interfaces to enable them to run on newer GPU architectures and have been added to the suite and/or to the catalog. These efforts are described in Section 5.

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

2 The Proxy App Suite v6.0

The ECP Proxy App Team released version 6.0 of the ECP Proxy App Suite on November 1, 2022 (https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/) .

Release 6.0 of the ECP Proxy App Suite includes the following proxy apps:

CabanaPIC Based on the Cabana library, it does explicit PIC in Cartesian geometry solving relativistic Vlasov-Maxwell equations.

E3SM-kernels Climate modeling kernels originating from the Energy Exascale Earth System Model (E3SM).

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

ExaMPM Material point method built with the Cabana library for fluid and solid mechanics.

GAMESS_RI-MP2_MiniApp Computes a quantum chemistry method, the RI-MP2 energy, for estimating the electron correlation error.

Goulash An interoperability proxy and test suite for heterogeneous programming models and compilers.

HyPar Compressible flow, hyperbolic-parabolic PDE solver using WENO and CRWENO spatial discretizations.

IAMR A parallel, adaptive mesh refinement (AMR) code that solves the variable-density incompressible Navier-Stokes equations and uses the AMReX framework.

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

SNAP Discrete ordinates neutral particle transport application. Mimics the computational workload, memory requirements, and communication patterns of PARTISN.

Quicksilver Represents the key elements of the Mercury workload by solving a simplified dynamic Monte Carlo particle transport problem.

As previously noted, these proxies were chosen to cover a wide scientific domain and because their implementations are compatible with contemporary GPU architectures. Our desire was also to include proxy apps and libraries that were developed within the ECP project. If users desire CPU-based proxy apps, Version 4.0/5.0 is likely more suitable and remains available.

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 80 entries. This section contains brief descriptions of the proxies that have been added to this collection since our last release.

3.1 SHAW

This code simulates the generation and propagation of elastic seismic shear waves in an axisymmetric domain. The name "SHAW" is an acronym built from "SHeAr Waves".
Languages: `C++, Kokkos, Python`
Institution: Sandia National Laboratories
Repo: https://github.com/Pressio/SHAW

3.2 PGAS-FMO Proxy Application

This proxy application simulates the compute load and data-movement of the kernel of the Fragment Molecular Orbital (FMO) method in quantum chemistry. The proxy implements a parallel global address space (PGAS) in order to support FMO.
Languages: `fortran-90+ / MPI`
Institution: Argonne National Laboratory
Repo: https://github.com/gdfletcher/pgas-fmo

3.3 FlexFlow - CandleUno

FlexFlow implementation of Candle Uno benchmarks.
Languages: `C++, C, Cuda, Python`
Institution: Los Alamos National Laboratory
Repo: https://github.com/flexflow/FlexFlow/tree/r22.05

3.4 HyPar

The Hyperbolic-Parabolic 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.
Languages: `CUDA, DPC++, Kokkos`
Institution: Argonne National Laboratory
Repo: https://bitbucket.org/deboghosh/hypar/src/master

3.5 IMEXLBM

Weakly Compressible Lattice Boltzmann solvers on structured grids.
Languages: `C++, Kokkos, DPC++`
Institution: Argonne National Laboratory
Repo: https://github.com/argonne-cps/imexlb;//https://github.com/Maccchiatooo/Cylinder-flow

3.6 IAMR

A parallel, adaptive mesh refinement (AMR) code that solves the variable-density incompressible Navier-Stokes equations and uses the AMReX framework.

Languages: C++
Institution: Los Alamos National Laboratory
Repo: https://github.com/AMReX-Codes/IAMR

4 Machine Learning Proxy App Suite

ML has become prevalent in HPC not only for data analysis and discovery but also as part of scientific simulation workflows. This lead the ECP proxy apps team to collect and curate different ML proxies relevant to scientific applications that represent 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. In this FY, we have extended some of the previously developed ML proxies and we have added new proxies to the suite. We report on this work below.

4.1 Additions to miniRL

As described in the previous milestone reports, miniRL is a proxy app for the Easily eXtendable Architecture for Reinforcement Learning (EXARL). miniRL uses a simple augmentation to the well-known Cartpole problem, called ExaCartpole, and uses a Deep Q-Network (DQN) agent. A DQN agent and its variants work with discrete action spaces where the actions are finite and discrete. During every step of the environment, the DQN agent either chooses a random action (exploration) or a best possible action using the learned policy (exploitation) to go from one state to the other. However, this strategy does not work for environments that operate in a continuous action space or becomes nonviable for very large action spaces. For such cases, using a policy network that can act like a function approximator is one possible solution.

The Deep Deterministic Policy Gradient (DDPG) [9,13] is one such algorithm that concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the q-function, which in-turn is used to learn the policy. In this approach the maximum Q value is approximated as $Q(s, \mu(s))$ to determine the policy, $\mu(s)$.

The DDPG agent has been added to miniRL and it uses the continuous Pendulum [4] environment where p, a frictionless pendulum, starts in a random angle from $[-\pi, \pi]$ with a random velocity between $[-1, 1]$, and the objective is to keep the pendulum standing up. The DDPG agent in miniRL reaches a reward (Figure 1) that plateaus around -125, which closely matches the OpenAI Gym leaderboard [3].

The miniRL distribution can be found on Github at `https://github.com/lanl/minRL`.

4.2 Open Catalyst

The Open Catalyst Project [2] is a collaborative research effort between Fundamental AI Research (FAIR) at Meta AI and Carnegie Mellon University's (CMU) Department of Chemical Engineering. The aim is to use AI to model and discover new catalysts for use in renewable energy storage to help in addressing climate change.

Scalable and cost-effective solutions to renewable energy storage are essential to addressing the world's rising energy needs while reducing climate change. As we increase our reliance on renewable energy sources such as wind and solar, which produce intermittent power, storage is needed to transfer power from times of peak generation to peak demand. This may require the storage of power for hours, days, or months. One solution that offers the potential of scaling to nation-sized grids is the conversion of renewable energy to other fuels, such as hydrogen. To be widely adopted, this process requires cost-effective solutions to running chemical reactions.

An open challenge is finding low-cost catalysts to drive these reactions at high rates. Through the use of quantum mechanical simulations (density functional theory), new catalyst structures can

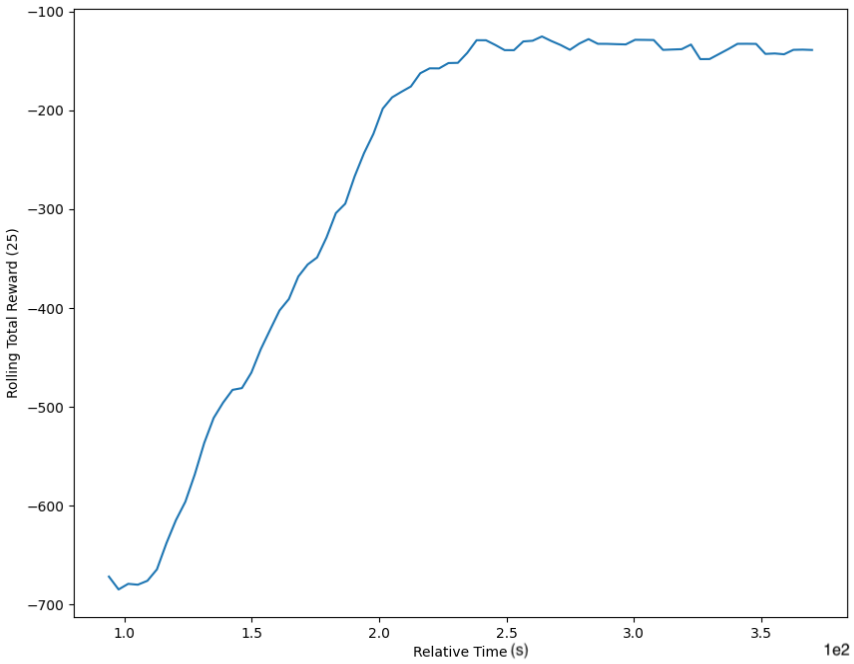


Figure 1: Reward plot of the DDPG Reinforcement Learning agent with the Pendulum environment.

be tested and evaluated. Unfortunately, the high computational cost of these simulations limits the number of structures that may be tested. The use of AI or machine learning may provide a method to efficiently approximate these calculations, leading to new approaches in finding effective catalysts.

The open catalyst proxy app provides implementations of state-of-the-art ML algorithms for catalysis that take arbitrary chemical structures as input to predict energy / forces / positions. The Open Catalyst Project website is https://github.com/mlcommons/hpc/tree/main/open_catalyst.

4.3 FlexFlow

FlexFlow is a DNN framework that automatically discovers fast parallelization strategies for distributed DNN training. FlexFlow generalizes and goes beyond today's manually designed parallelization strategies (e.g., data and model parallelism) for distributed DNN training by exploring parallelization opportunities across different Samples, Operators, Attributes, and Parameters.

FlexFlow includes a novel execution simulator to evaluate the runtime performance of different strategies and uses an automated search algorithm to discover highly optimized strategies, which generally outperform today's manually designed strategies.

FlexFlow provides the following key features:

- Flexible Parallelization: FlexFlow supports parallelizing DNN training through combinations of the Sample, Operator, Attribute, and Parameter dimensions, and guarantees that different parallelization strategies maintain the same model accuracy by design.

- Performance Auto Tuning: To accelerate DNN training on a specific parallel machine, FlexFlow uses guided randomized search to automatically find fast parallelization strategies while requiring no manual effort.
- Keras Support: FlexFlow offers a drop-in replacement for TensorFlow Keras and transparently accelerates existing Keras programs by discovering faster parallelization strategies.
- Large-Scale GNNs: FlexFlow enables fast graph neural network (GNN) training on large graphs (e.g., billion-edge) by distributing GNN computations across multiple GPUs (potentially on multiple compute nodes) using attribute parallelism.

The FlexFlow-CandleUno proxy app is the FlexFlow version of the Candle-Uno [1] benchmarks (https://github.com/flexflow/FlexFlow/tree/r22.05).

4.4 ML Proxy App Suite Release 6.0

Release 6.0 of the ECP ML Proxy App Suite includes the following proxy apps:

miniGAN Models performance for training generator and discriminator networks. The GAN's generator and discriminator generate plausible 2D/3D maps and identify fake maps, respectively.

miniRL A reinforcement learning (RL) proxy application derived from the Easily eXtendable Architecture for Reinforcement Learning (EXARL) framework, that is designed to use RL for control and optimization of applications or experiments.

CRADL Captures performance metrics during inference on data from multiphysics codes, specifically ALE hydrodynamics codes.

Cosmoflow-Benchmark Involves training a 3D convolutional neural network on N-body cosmology simulation data to predict physical parameters of the universe.

MLperf-DeepCam Trains a deep learning segmentation model for identifying extreme weather phenomena in climate simulation data.

Open Catalyst Provides implementations of state-of-the-art ML algorithms for catalysis that take arbitrary chemical structures as input to predict energy / forces / positions.

FlexFlow-CandleUno FlexFlow version of the Candle-Uno benchmarks. FlexFlow is a DNN framework that automatically discovers fast parallelization strategies for distributed DNN training.

5 Continued Development of New Proxy Applications

Members of the ECP Proxy App Team have been involved in the development of a number of new proxy applications. Some of these proxies are complete and already in distribution (see the FY21 Proxy App Suite Release report on our project website). However, several of these proxies have been initially developed and released, but have been extended and enhanced during this FY. This section describes our development efforts.

5.1 FFTX Library and a General Proxy for Verifying Correctness of 3D FFTs

Fast Fourier Transforms are used in a broad range of DOE science applications, including ones represented in the exascale application space. In 2017, the ECP Application Development leadership team surveyed the application projects regarding the use of FFTs in their codes. Of the 25 projects, 12 reported current significant use of FFT, and four 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.

Currently, most scientific applications that call FFTs use either proprietary libraries from vendors, such as MKL from Intel, ACML from AMD, cuFFT from NVIDIA, and LibSci/CRAFFT from Cray, or the free portable FFTW library [7]. FFTW applies methods of autotuning and combining calls to one-dimensional FFT kernels to generate code for multidimensional FFTs. However, FFTW does not support modern accelerators or advanced architectural features in a way that provides high performance, and the library is no longer being actively developed.

Hence, ECP has developed a new FFT library called FFTX [11] 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 on multiple platforms.

5.1.1 Implementation of FFTX

FFTX is based on SPIRAL [12], an open-source symbolic analysis and program generation system that generates platform-tuned implementations of signal processing transforms, as well as combinations of these transforms with other operations such as linear transformations.

Generally, there are multiple ways of performing an FFT on a particular size, depending on the factorization of the size, with different radices for different factors, and different ways of combining sub-algorithms for the different radices [14]. Similar to FFTW, the SPIRAL backend of FFTX builds a directed acyclic graph of FFT algorithms and performs algebraic simplifications on it, pruning the graph in order to find an efficient overall algorithm for the particular platform. For this algorithm, SPIRAL generates C code that is compiled and run.

The fork of the FFTX repository used in this report is freely available at:
`https://github.com/petermcLBL/fftx`
Instructions are given there for cloning the repo, installing SPIRAL, and generating and compiling examples that are included.

5.1.2 A General Proxy for Verifying Correctness of 3D FFT Calculations

In the FFTX repo, the directory `examples/verify` has code to check that the answers produced by a 3D FFT function are correct. The implementation is not just for FFTX but also for FFTW, cuFFT, rocFFT, and MKL.

According to Ergün [6], a sufficient set of tests to verify that an FFT routine is returning correct answers is the following:

1. Linearity: Test that $\text{FFT}(\alpha X + \beta Y) = \alpha\text{FFT}(X) + \beta\text{FFT}(Y)$ for random inputs X and Y, and constants α and β.
2. Impulses: Test that the FFT of a unit impulse is a constant function, and vice versa. Also test that $FFT(R) + FFT(I - R) = FFT(I)$ for random input R, and I either a unit impulse or a constant function.
3. Shifts: Test that the FFT of a random array shifted in any of the dimensions in the spatial domain transforms to the FFT of the original array shifted appropriately in the frequency domain, and vice versa.

The build process for FFTX generates the executable `testverify_lib` that runs the tests on *all* the 3D sizes that are built in the FFTX library, on four FFT functions: complex-to-complex forward and inverse, real-to-complex, and complex-to-real. The user specifies the number of iterations of random data at runtime.

The `examples/verify` directory also contains code for running the same four FFT functions in other libraries besides FFTX. At runtime, the user specifies not only the number of iterations of random data, but also a 3D array size. The executables, for libraries that are available, are in:

- `testverify_device` for NVIDIA's cuFFT if the `cmake` build line contains `-D_codegen=CUDA`;
- `testverify_device` for AMD's rocFFT if the `cmake` build line contains `-D_codegen=HIP` (yes, same executable name as for cuFFT);
- `testverify_fftw` for FFTW if the `cmake` build line contains `-DUSE_FFTW3=1`;
- `testverify_mkl` for Intel's MKL if the `cmake` build line contains `-DUSE_MKL=1`.

5.2 PGAS-FMO: A Proxy for Fragment Molecular Orbital (FMO) Method in Quantum Chemistry

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, as shown below.

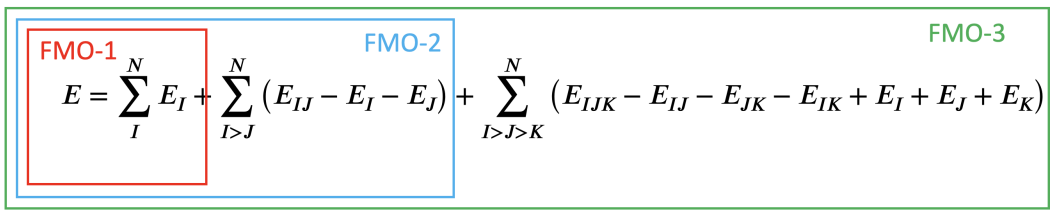


Figure 2: The total energy E in FMO is expanded as a series in the fragments starting with the monomers, then the dimers, and so on.

The FMO 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.

5.2.1 Work done in FY22

The development of PGAS-FMO was completed in March 2022. Shown below is a scaling plot of the ProxyApp:

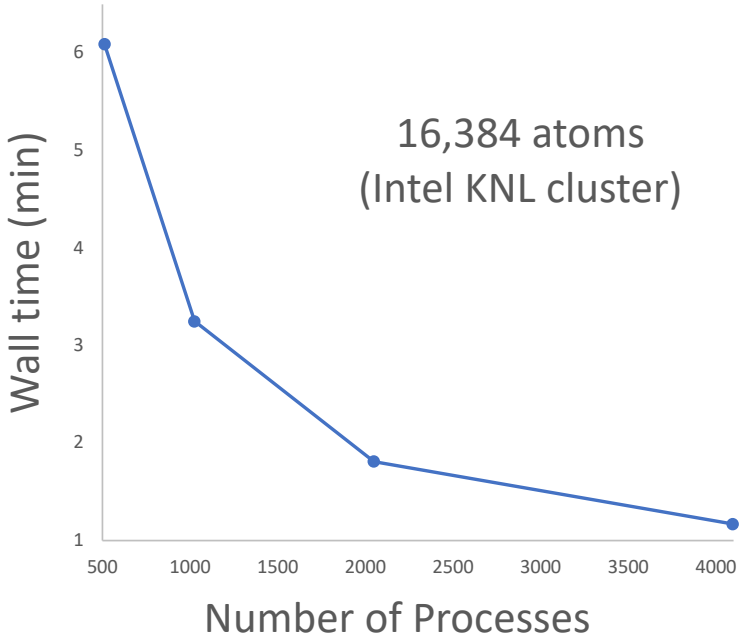


Figure 3: Strong scaling of the PGAS-FMO proxy application on Theta KNL at ALCF using MPI+OpenMP. The scaling was done on a system consisting of 16,384 helium atoms, atomic spacing: 1.2 Å, Coulomb cutoff: 10.0 Å, point charge cutoff: 20.0 Å, and system dimension: 38.4 Å. Each atom/fragment is modeled using ten gaussian basis functions.

The finalized implementation of PGAS-FMO proxy application may be obtained from the PGAS-FMO repository, which has been updated, and it also includes performance data and regression tests.

5.2.2 Software Deliverables

In addition to being available on the public PGAS-FMO repository, the application was submitted to the ECP ProxyApps Suite, and was accepted on April 7th.

5.3 miniGAP: A Proxy for Developing Machine Learning-Based 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. However, 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 dependent on the parametrized forms of the force potential, and existing potentials hardly neither include high order energy terms nor do they 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.

In particular, non-linear and non-parametric approaches, such as Gaussian Progress Regression (GPR) and Neural Network models 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 it requires fewer 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.

5.3.1 Work done in FY22

The project has made significant progress in the implementation of GPR, where we have used Tensorflow and OpenMP for the descriptor generation.

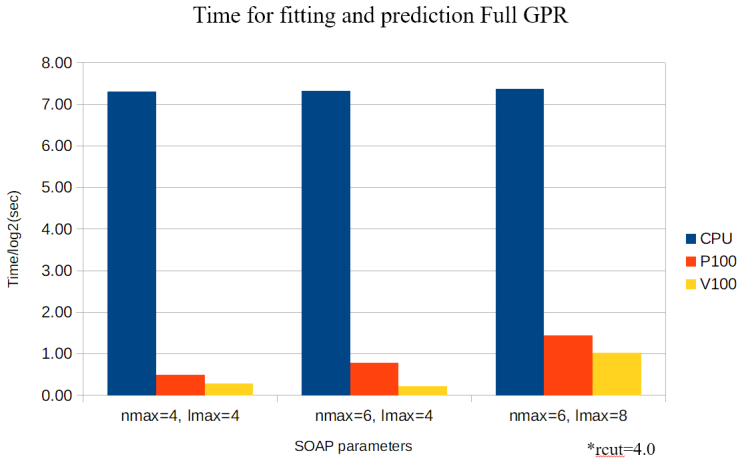


Figure 4: Performance of fitting and prediction of minigap code of 3000 molecules from QM7b database. These performance tests were run on the JLSE test platform at Argonne on A100 Nvidia GPUs, where the offloading to the GPUs was done using the CUDA specific APIs in TensorFlow

The release updated miniGAP ProxyApp, available from (https://git.cels.anl.gov/vama/minigap), consists of the improvements:

1. Developed the preprocessing data and regression tests on test beds, such as Intel Xe and AMD GPU.
2. Tested I/O and inference for extended systems.
3. Added OpenMP GPU to descriptor generation

5.3.2 Software Deliverables

In ProxyApps is available on the public miniGAP repository. The ProxyApp is in the process of being included in the ECP ProxyApps Suite.

5.4 QTensor-mini: A Proxy for Quantum Tensor Contraction Simulators

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.

5.4.1 Work done in FY22

The project ended in spring 2022. During the period from January to April, our work focused on finding optimal tensor network contraction sequences. This work led to the development of new algorithms and the submission of the paper to the prestigious Institute of Electrical and Electronics Engineers (IEEE) High-Performance Extreme Computing (HPEC) conference, where the paper won the best student award.

5.4.2 Publications

- Cameron Ibrahim, Danylo Lykov, Zichang He, Yuri Alexeev, Ilya Safro, *Constructing Optimal Contraction Trees for Tensor Network Quantum Circuit Simulation*, https://doi.org/10.48550/arXiv.2209.02895

5.4.3 Deliverables

The QTensor-mini ProxyApp available for download. We have initiated the process of spackifying the code and having it included in the ECP ProxyApps suite.

5.5 HyPar: A Proxy for Compressible Navier-Stokes Solvers on Structured Grids

HyPar, which stands for Hyperbolic-Parabolic 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 rationale for the creation of HyPar was to provide a variety of spatial discretizations, and temporal integration schemes, as well as a parallel implementations with various device (GPU) abstraction layers, to evolve a system of PDEs on heterogeneous computing platforms. 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.

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 WENO/CRWENO schemes, in particular, have the twin advantages of being robust, minimally dissipative schemes, that also one have a compact stencil.

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 several device abstraction layers, such as CUDA, DPC++, and Kokkos (with the SYCL backend) that are available in the Nvidia NVHPC, and Intel oneAPI programming environments. The science problem that was identified, to demonstrate the code transformation of the MPI-only code to MPI+X (with "X" being CUDA, DPC++, Kokkos) is that of decaying isotropic turbulence in compressible flows. We believe that by using the WENO and CRWENO schemes in HyPar to simulate the model problem on heterogeneous computing platforms (e.g., Summit, Aurora, Frontier), HyPar would be

an effective ProxyApp for compresssible flow solvers. We envisage HyPar being used either 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 complex geometries on heterogeneous computing platforms.

5.5.1 Work done in FY22

In FY22, we continued the developments we had initiated in FY21 by focusing on the performance of MPI+CUDA implementation in Nvidia GPU based computing platforms. Our objective was to establish a baseline against which subsequent comparisons could be made with DPC++ and Kokkos implementations. Therefore, our kernel design principles to efficiently implement HyPar with CUDA focused on optimizing its memory access patterns. Since many core operations of HyPar are memory-bound, it is crucial for computational performance to avoid slow memory transactions, and to reduce as many wasted compute resources and memory transactions as possible. The steps we took to optimize computational performance on GPUs are as follows:

- Completed refactoring the MPI-only HyPar code by moving up the code stack to implement the remaining routines on GPUs to avoid costly data transfer of intermediate results between the host and device.
- Implemented lexicographic thread configuration.
- Implemented coalesced memory access.
- Implemented distributed (MPI) computation between compute nodes using GPU-aware MPI. Analyzed the impact of communication and communication frequency on the total computation time and convergence, respectively.

We present the computational performance of our MPI+CUDA implementation on the isotropic turbulence decay of compressible flow. We compare the CPU+GPU computation time with that of the MPI-only based (CPU) implementation, and also demonstrate the scalability of our implementation using multiple GPUs. The tests to assess the performance of HyPar were performed on OLCF/Summit where each compute node has 6 NVIDIA Volta V100 GPUs and 2 POWER9 CPU sockets with 22 physical cores each and 4 hardware threads on each physical core.

5.5.2 Unit Performance: One GPU vs. one CPU core

In the case where multiple GPUs are employed as in Section 5.5.3, our observation provides a guideline for determining the appropriate number of grid points to be assigned to each GPU to prevent resource under-utilization.

5.5.3 Parallel Performance: Multiple GPUs using MPI

5.5.4 Publications

- Youngdae Kim, Debojyoti Ghosh, Emil M. Constantinescu, Ramesh Balakrishnan, *GPU-accelerated DNS of compressible turbulent flows*, Computers & Fluids, Volume 251, 30 January 2023, Elsevier, https://doi.org/10.1016/j.compfluid.2022.105744

5.5.5 Deliverables

The MPI+CUDA implementation of the HyPar ProxyApp has been verified and is available for download. We had initiated the process of spackifying the code, and it is in the process of being

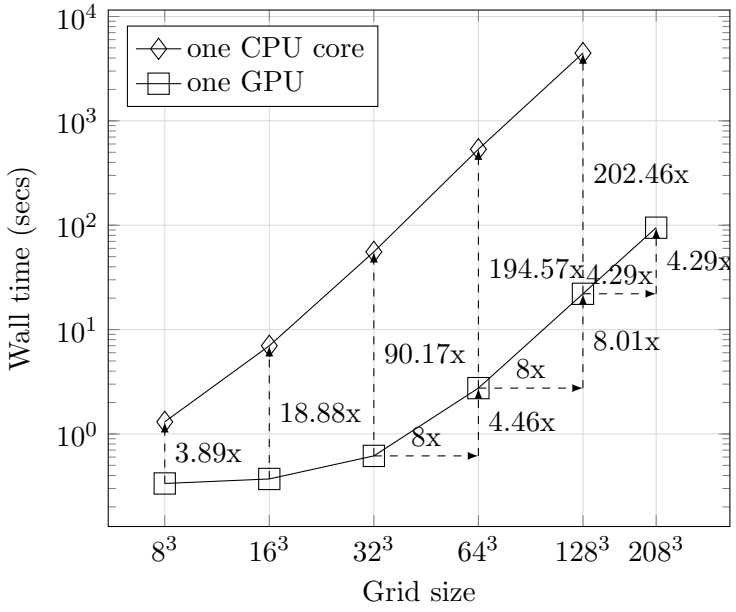


Figure 5: The wall times of one GPU and one CPU core, respectively, for executing 100 time steps over varying grid sizes. Once we reach a grid with 64^3 points, the computational resources on GPUs start to saturate. This has been verified by comparing the computation time with grids of sizes 64^3 and 128^3, where the wall time increases exactly at the same rate as the increase of the grid size in this case.

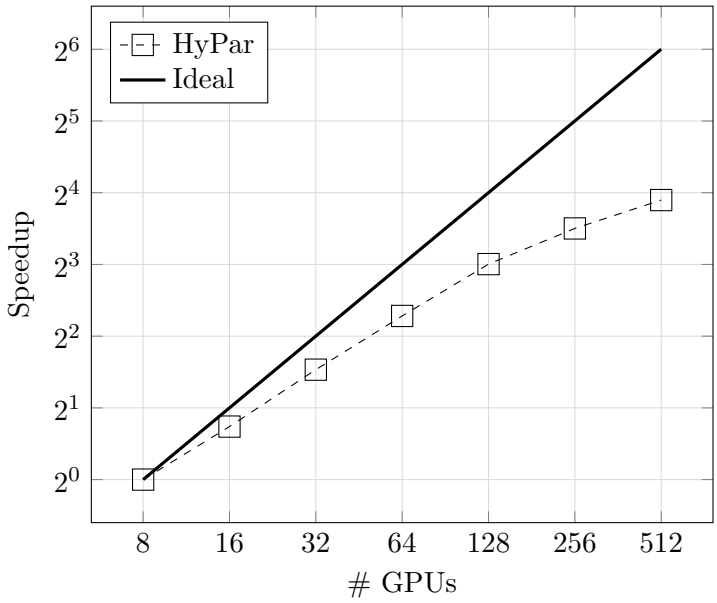


Figure 6: Strong scaling over a grid with 256^3 points. We note that a grid with 208^3 points is the maximum size that HyPar can run on one GPU because of the 32 GB memory limit on V100. For V100 with 16 GB of memory, the maximum size is around 160^3. Communication between GPUs was performed via CUDA-aware MPI, which allowed us to directly communicate between memories of multiple GPUs on the same or different nodes without staging them through CPUs.

included in the ECP ProxyApps suite. We will continue to update the publicly accessible Github repository with MPI+Kokkos and MPI+DPC++ implementations.

5.6 IMEXLBM: A Proxy for Weakly Compressible Lattice Boltzmann Solvers on Structured Grids

The Lattice Boltzmann Method (LBM) [8] is a relatively novel approach to solving the Navier-Stokes equations (NSE) in the low-Mach number (weakly compressible) 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 [10] 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 (or modification) of other full-fledged LB codes (e.g. Palabos), and is a much needed building block for simulating multiphase flows on exascale computing platforms at Argonne National Lab and the other DOE laboratories.

5.6.1 Work done in FY22

FY22 began with the development of IMEXLBM for C++ on the host (i.e CPU). The parallel model employed here was message passing interface (MPI)-only. It was developed, verified and validated for a common CFD benchmark, flow past a sphere (3D) [15]. Following this, our team began to develop the code to use the performance portable library, Kokkos, to enable execution on the device (i.e. GPU) of heterogeneous platforms. Much of the work in FY22 was focused on optimizing the MPI+Kokkos version of IMEXLBM on the ThetaGPU platform at ALCF [5].

5.6.2 Scaling and Performance: Single Phase Flow over a 3D Sphere and a 3D Taylor-Green Vortex

In IMEXLBM the strong scaling and weak scaling were performed for 3D sphere flow and the 3D Taylor-Green Vortex simulation. All the performance and simulation data in this document were generated on the ThetaGPU cluster at the Argonne Leadership Computing Facility [5].

The validation of the GPU acceleration is performed by the simulation of flow over a 3D sphere. The simulation results from the MPI+Kokkos code CPU+GPU was compared with the pure MPI code and the differences were determined to be about 10^{-16} which means the simulation results from GPU are consistent with the results calculated from CPU. The results of the 3D scaling and performance study are shown in FIG.7 (a) and (b). The GPU cases have slightly better performances than the CPU cases when the number of processors are small. When we increase the devices number to 8, GPU cases performance is twice speed to the CPU performance.

5.6.3 Two-Phase Simulation: Single Droplet inside a 3D Taylor-Green Vortex

We extended our work by adding physical complexity. In particular we sought to understand the performance and feasibility of performing a two-phase, sharp interface calculation. We focus the dynamics to be non-mixing hence a binary fluid where properties may change over a small distance (i.e. the interface). These simulations are difficult from a modeling perspective as it requires a solver of another particle distribution function (to resolve the interface) along with a surface tension force which involves first and second order derivatives. Resolving derivatives adds further burden to computation through communication with sending and receiving messages in ghost-layers. To better understand the effect of computing a derivative on the performance, we implement two numerical derivative methods: the local derivative method [15] and the isotropic finite difference method [15], to the model, and study the short-time shape evolution of a single droplet inside of Taylor-Green Vortex.

The weak scaling and strong scaling of this case is shown in Figure 7 (c), (d). From the weak scaling, because we introduce another particle distribution function (PDF), the average time per simulation step for single droplet evolution is nearly twice of the single phase case. When we increase the number of the processors, the average time per step for the local derivative case is less than isotropic finite difference case. Since the local derivative is only calculated on the device i.e. GPU without MPI data transfer, it saves a lot of time. The strong scaling in Figure 7 (d) also proves that the local derivative method is more efficient compared to the isotropic finite difference method. However, when we choose the local derivative method, we need to increase the interface thickness to obtain a consistent result with isotropic finite difference. For a small scale problem, the isotropic finite difference is able to gain an accurate result with similar performance. When we consider a large scale problem, the local derivative method can be chosen to improve the efficiency of the code.

5.6.4 Publications

- Zhao, C., Patel, S., Lee, T., & Balakrishnan, R. *IMEXLBM: A GPU-accelerated, lattice Boltzmann solver for exascale machines.* (In preparation)
- Zhao, Chunheng. *Ternary flow simulations based on the conservative phase field, lattice Boltzmann method.* 2022. The City College of New York (CUNY). PhD Dissertation (in preparation).

5.6.5 Deliverables

- IMEXLBM C++ Suite for single phase flows with Kokkos directives on GPUs

  URL: https://github.com/argonne-cps/imexlb

- IMEXLBM C++ Suite for single phase flows with DPC++ directives on GPUs

  Current URL: https://github.com/Maccchiatooo/Cylinder-flow
  Target URL: https://github.com/argonne-cps/imexlb

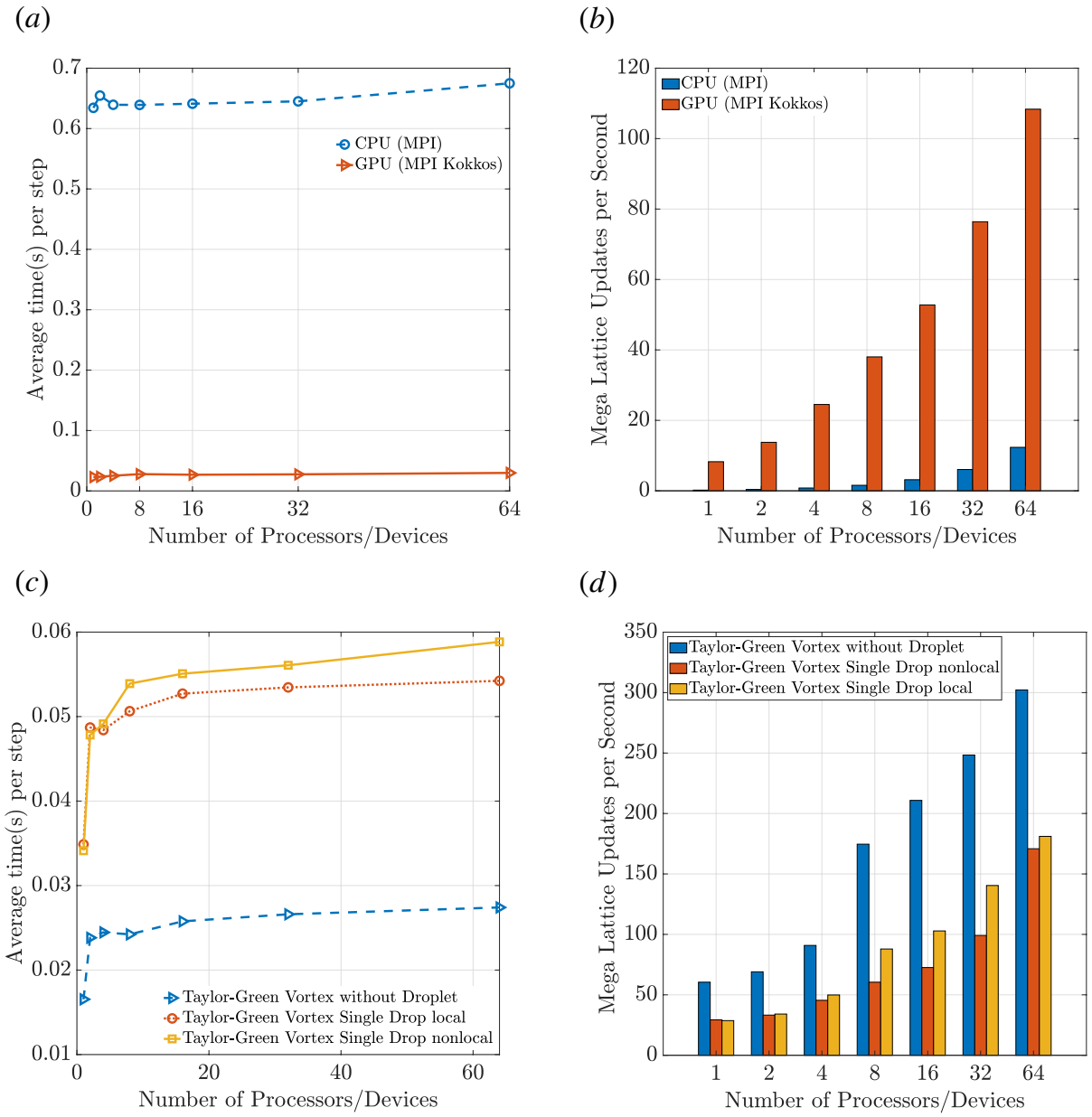


Figure 7: Weak/strong scaling for the 3D sphere flow (a), (b); and for the Taylor-Green Vortex (c), (d). Comparison between the simulation running on CPU and GPU is shown in (a) and (b). When we run the simulation on 64 processors, the speedup ratio nearly reach 10. The speed of single phase Taylor-Green Vortex is almost twice of the multi-phase simulation (single droplet in Taylor-Green Vortex). Results gathered on on ThetaGPU [5]

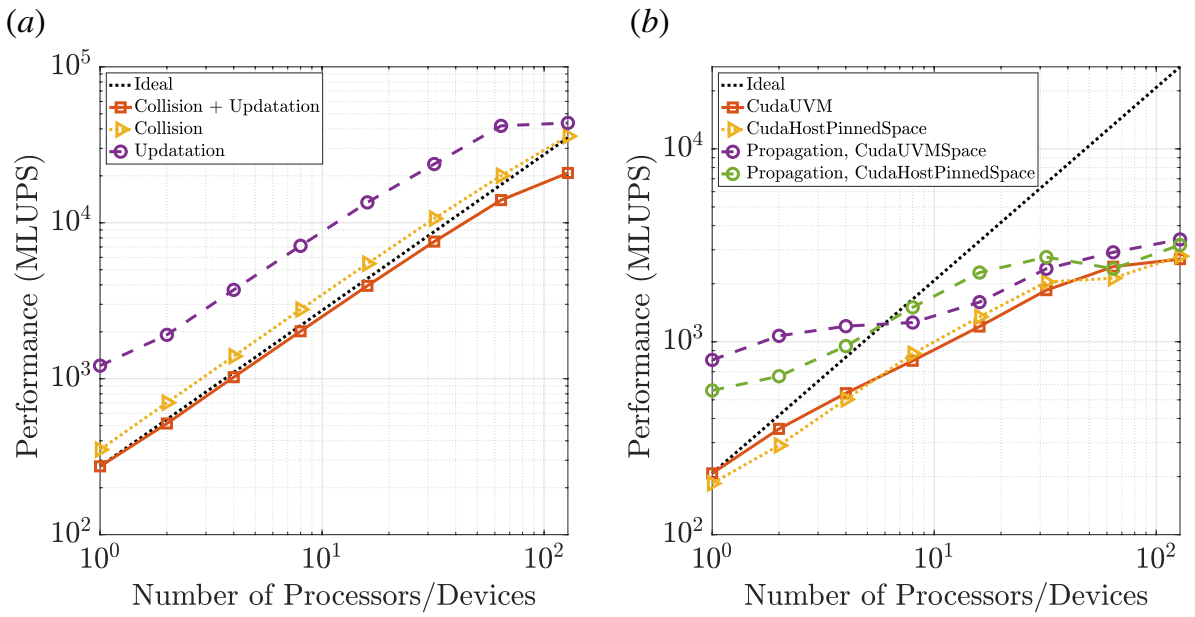


Figure 8: Strong scaling of the function collision, propagation and updation under $GPU = [1-128]$, for 3D Taylor-Green Vortex. The simulation is conducted based on $L/\Delta x = 256$. (a) shows the performance of the collision and updation. It is nearly close to the ideal performance. (b) shows the performance of the full loop (collision+propagation+updation), and the propagation.

6 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] Ecp candle benchmarks. URL: https://github.com/ECP-CANDLE/Benchmarks/blob/master/Pilot1/Uno/.

[2] Open catalyst project. URL: https://opencatalystproject.org/.

[3] openai/gym leaderboard. URL: https://github.com/openai/gym/wiki/Leaderboard.

[4] openai/gym pendulum v1. URL: https://github-wiki-see.page/m/openai/gym/wiki/Pendulum-v1.

[5] Thetagpu. https://www.alcf.anl.gov/alcf-resources/theta. Accessed: 2022-12-11.

[6] F. Ergün. Testing multivariate linear functions: Overcoming the generator bottleneck. In *Proc. ACM Symp. Theory of Computing (STOC)*, volume 2, pages 407–416, 1995.

[7] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[8] 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.

[9] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2015.

[10] 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.

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

[12] Markus Püschel, José M. F. Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan W. Singer, Jianxin Xiong, Franz Franchetti, Aca Gačić, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo. SPIRAL: Code generation for DSP transforms. *Proc. of the IEEE*, 93(2):232–275, 2005. Special issue on *Program Generation, Optimization, and Adaptation*.

[13] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proc. Int. Conf. on Machine Learning*, 2014.

[14] Charles Van Loan. *Computational Frameworks for the Fast Fourier Transform*. Society for Industrial and Applied Mathematics, USA, 1992.

[15] Chunheng Zhao. *Ternary flow simulation based on the conservative phase field Lattice Boltzmann method*. PhD thesis, The City College of New York, 2022.