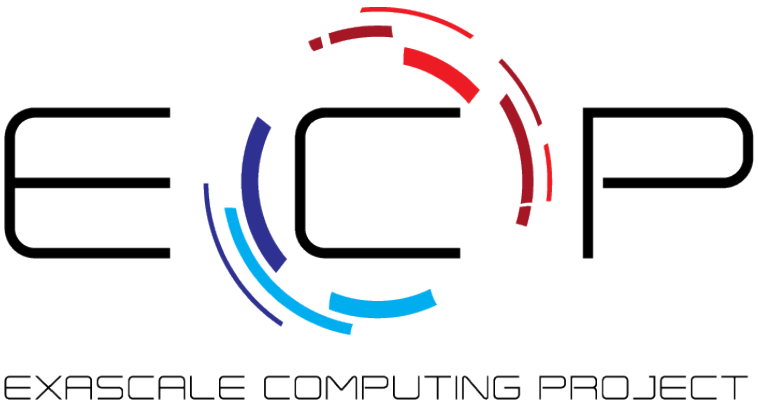
# FY18 Proxy App Suite Release

Milestone Report for the ECP Proxy App Project

David Richards[1], Abhinav Bhatele[1], Omar Aaziz[2], Jeanine Cook[2], Hal Finkel[3], Brian Homerding[3], Peter McCorquodale[4], Tiffany Mintz[5], Shirley Moore[5], and Robert Pavel[6]

[1]Lawrence Livermore National Laboratory, Livermore, CA
[2]Sandia National Laboratories, Albuquerque, NM
[3]Argonne National Laboratory, Chicago, IL
[4]Lawrence Berkeley National Laboratory, Berkeley, CA
[5]Oak Ridge National Laboratory, Oak Ridge, TN
[6]Los Alamos National Laboratory, Los Alamos, NM

October 31, 2018




LLNL-TR-760903

# 1 Executive Summary

The ECP Proxy App Team released version 2.0 of the ECP Proxy App suite on September 28, 2018. The new version includes 5 new proxies and adds or improves coverage in several areas, most notably graph analytics and communication patterns. Proxy apps are used heavily by the PathForward projects, and several of the new proxies are being eagerly adopted. Over the last year we have observed vendors are becoming more sophisticated in their use of proxy apps. However, there is still progress to be made, especially in the area of ensuring that representative problem specifications are used. The Proxy App Team is currently working on such specifications and will be publishing them on our website in the near future.

## 2 The Proxy App Suite v2.0

The ECP Proxy App Team released version 2.0 of the ECP Proxy App Suite on October 1, 2018. For this release, our focus was to improve coverage in areas with few proxies, such as graph analytics, and to provide more proxies with direct links to ECP application and co-design projects. To this end, we have added five new proxies to the suite: Ember, miniQMC, miniVite, PICSARlite, and thornado-mini. We also removed miniFE and miniTri. CoMD was replaced by ExaMiniMD in a previous interim release of the suite.

It is impossible to fully represent all aspects of the DOE or even the ECP workload with a small collection of proxies. However, as shown in the list below, the 15 proxies we have selected provide representative samples of a broad range of data structures, data access patterns, algorithms, numerical methods, and simulation techniques. The Proxy App Team is also prepared to recommend other proxies from our catalog of more than 50 proxies for specific modeling needs.

Release 2.0 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.

**CANDLE Benchmarks** Machine learning benchmarks from the ECP CANDLE project.

**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.

We also continued to improve the ease of use and testing of the Proxy App Suite. This was primarily done through a combination of Spack and continuous integration. We added Fedora as a tested platform, as it is closer to TOSS than the existing CI on Ubuntu. When testing uncovered build or correctness issues we worked with the proxy developers to debug and improve the spackages as well as the apps themselves. Through our work we have ensured that each proxy can be installed with a simple spack command. This also guarantees that the build system of each proxy app is documented and set up in a manner that ensures that dependencies are easily identified and provided.

Additionally, we have improved the ECP Proxy Apps website. We have added more metadata to ensure that the provenance and sponsors of each proxy app are easily found. Furthermore, we have improved synergy with our proxy application assessment teams by providing a mechanism to directly publish our assessment findings on the proxy apps website. Additional material on representative problem sizes and comparisons to parent applications will be published in the near future.

# 3 Areas of Improved Coverage

## 3.1 Graph Analytics: miniVite

Version 1.0 of the ECP Proxy App Suite included miniTri as an attempt to capture graph analytics patterns. However, several experts we contacted felt that miniTri was not representative of typical graph workloads. To resolve this problem, we worked with the ExaGraph Co-Design Center to produce a new graph analytics proxy app. The result of those efforts is miniVite.

MiniVite implements a single phase of the Louvain community detection method in distributed memory. Because of its speed and ability to yield high-quality communities, the Louvain method is one of the most widely used tools for community detection. A full implementation of Louvain requires multiple phases and multiple iterations; however, the single phase captured in miniVite contains a large fraction of the work that would be done in a full multi-phase implementation.

Like its parent, Vite, miniVite can read any real-world graph data as input. For efficient I/O, such graph data is required to be in a binary format. The code for binary conversion from a variety of common graph formats is packaged separately with Vite. Because it can be cumbersome to manage the large files typically associated with real graph data, miniVite can also generate a Random Geometric Graph (RGG) in parallel.

## 3.2 Ember: A Communication Pattern Proxy

Multi-node communication patterns underpin the scalability and parallel performance of the Department of Energy and broader HPC workloads. Modeling of these patterns is an important aspect of extreme scale supercomputing systems. To date, many vendors have relied on communication traces to evaluate network designs. However, traces can be difficult to obtain at scale, and take significant I/O storage. For interconnect simulators, the reading and replay of traces requires high-performance I/O subsystems that are often expensive and may be unavailable. To this end, the Ember suite provides communication patterns in a simplified setting where the application calculations, control flow, etc, are replaced by parameterized delays. This enables traces to be captured more efficiently, and network simulators such as the Structural Simulation Toolkit (SST, http://sst-simulator.org), can use the Ember/SST motif library to directly implement these patterns thus eliminating the need for traces.

The intention of Ember is to enable much larger scale modeling of high-performance interconnects to achieve DOE's goal of scalable Exascale computing systems. The motifs contained in the suite are intentionally simplified, and by design, do not capture every permutation of the basic patterns within the DOE workload. When used collectively, our experience working with leading industry vendors has been that the motifs capture pertinent aspects of the network interconnect. Ember code components represent highly simplified communication patterns that are relevant to DOE application workloads. In most cases, the patterns are parameterized to allow for the study of sensitivities to message sizes, message rates, rank placement, etc. The initial implementation

of Ember was written within the SST simulation framework to permit evaluation of the communication patterns within the simulator removing the need for large trace files. In order to allow for these to be run outside of SST, these patterns are now being extracted and written in MPI or SHMEM so they can be used with a broader tool portfolio and run on supercomputing platforms.

### 3.3 Quantum Monte Carlo and miniQMC

miniQMC is a proxy for QMCPack, which is an electronic structure code for periodic 2D and 3D solid-state systems that implements several Quantum Monte Carlo algorithms, including diffusion Monte Carlo (DMC), Variational Monte Carlo (VMC), and several other advanced QMC methods. miniQMC is based on bulk nickel oxide (NiO) and computes the total energy of a quantum system composed of mobile electrons and immobile atoms, using both DMC and VMC algorithms. For the DMC algorithm (currently our focus), particles are propagated through 3D space, using Monte Carlo to generate a sequence of electron positions (walkers) that are randomly sampled with some probability ($P$); their local energy is computed using an integral estimation. These walkers either reproduce, remain the same, or kill themselves based on their energy. The computationally expensive parts of the algorithm are generating the particle positions and computing the local energy; the stochastic behavior of the walkers may cause periodic computational imbalance which the algorithm attempts to mitigate. MiniQMC contains the key computational kernels of QMCPack, but their relative importance in terms of execution time is not necessarily maintained. Currently, miniQMC only implements a single walker and is intended for single-node performance studies. It is not intended to represent the communication characteristics of its parent, QMCPack.

### 3.4 Thornado: A Proxy for Radiation Transport in Astrophysical Systems

Thornado is a proxy app developed in the ExaStar ECP AD project. The ExaStar project targets multiphysics simulations of astrophysical explosions. Their target application, Clash, will be a component-based multiphysics toolkit, built on capabilities of current astrophysics simulation codes Flash and Castro and on the massively parallel adaptive mesh refinement framework AMReX. The Clash code suite will have modules for hydrodynamics, advanced radiation transport, thermonuclear kinetics, and nuclear microphysics. In this code suite, the radiation transport kernel is expected to consume 80% of the full application runtime.

Thornado is a proxy app for finite element, moment-based radiation transport. The proxy application code solves the equation of radiative transfer using a semi-implicit, discontinuous Galerkin (DG) method for the two-moment model of radiation transport. Thornado currently implements deleptonization radiation transport (i.e the loss of leptons in a protoneutron star over time through isotropic radiation of neutrinos). The deleptonization problem is representative of both the FLOPS and memory access patterns that are expected in the exascale problem.

### 3.5 PICSARlite: An Improved WarpX Proxy

PICSAR is the Fortran 90 library of particle in cell (PIC) subroutines that are used in the WarpX particle accelerator modeling application. The PICSAR library includes a stand-alone application that runs PICSAR subroutines without the AMReX library that is required by the full WarpX application. Because PICSAR has 47,039 lines of code, not counting comments and blank lines, it is larger than is usually desirable for a proxy app. Following consultations with the WarpX team, we extracted a subset of PICSAR called PICSARlite with 9,036 lines of code (also excluding comments and blank lines) that includes the most essential components and, generally, only the

simplest option when there are multiple options available. This version of PICSARlite is in the `PICSARlite` branch of the PICSAR repository.

In an effort to make PICSARlite more adaptable as well as track future changes in the PICSAR library, the WarpX team added a script to the PICSAR repository that takes the full library and generates a subset that contains only the user's specified choices for Maxwell solver, particle pusher, particle depositions, and optimization. The choices `fdtd`, `boris`, `direct`, and `off`, respectively, generate the smallest version of this customizable PICSARlite, with 21,339 lines of code, not counting comments and blank lines. The script to generate a customized PICSARlite is in the branch `generate_miniapp` of the PICSAR repository. We are working with the WarpX team to add options to the script that can generate an even smaller version, down to the length of the mini-application in the `PICSARlite` branch.

# 4 Identified Gaps: Coverage and Other Issues

Although we have made progress in improving the coverage of the ECP Proxy App Suite, some gaps remain. From the perspective of algorithms and numerical methods, the most obvious gap is the lack of any proxies that cover deterministic transport or the particle tracking part of Monte Carlo transport (XSBench covers only cross section lookup). This gap is mostly mitigated by the fact that the CORAL-2 procurement suite contains both Kripke and Quicksilver. Hence potential DOE vendors are well exposed to proxies in those areas.

The Proxy App Team also receives a fairly steady stream of requests for proxies that exercise certain hardware features or programming models and methods. The requests that currently cause the most difficulty are for proxies that use atomics or are written in task-based styles. To some degree, this is a chicken and egg problem. Codes haven't used atomics and tasks because implementations have traditionally performed poorly. However, better implementations haven't been a priority since the features were lightly used. Proxies that utilize certain advanced memory or network features and proxies that model workflow and scheduling concerns are also in demand. More proxies with GPU ports and OpenMP 4.5 versions are also needed.

Overall, we see clear signs that vendors are becoming much more aware of the limitations of proxy apps. It is now common to hear questions about whether optimizations found for proxies will actually still apply in more complex proxy apps. We have also seen a trend toward using realistic problem specifications in evaluations and reporting the problem specifications that were used. However, this latter trend seems to be most strongly associated with proxies in the CORAL-2 suite. Because those proxies were supplied with specific problem descriptions, it was far easier for vendors and other users do to the right thing. Now that vendor attention is shifting away from the CORAL-2 procurement, we are seeing a return to old patterns of failing to specify how proxies are configured or run. The Proxy App Team is moving to address this problem by publishing standardized problem descriptions on our website.

Finally, the Proxy App Team has observed a trend towards "miniapps" that contain 10's or even 100's of thousands of lines of code, or that depend on large and complex libraries such as Trilinos or HDF5. We have struggled to produce a new proxy to represent adaptive mesh refinement because of the size and complexity of the AMReX library. To some degree, large and complex proxies may be unavoidable, but the creation of PICSARlite shows that progress in this area can be made. We have also been working with proxy developers to find ways to reduce library dependencies without substantially altering the fidelity of their proxies.

## 5 Proxy App use in the ECP Community

To date, there are several communities within the overall ECP project that are using various proxies from the current ECP suite in their project workflows. Through the ECP Proxy App Suite website and through team member attendance at PathForward and other vendor meetings, various ECP projects are finding the suite and the information that accompanies each proxy useful to their work.

The PathForward vendor community is actively using proxies from both the past and present (v2.0) suite. Proxy App Team members are actively engaged with HPE, Cray, AMD and Intel to advise them on problem inputs/parameters and execution details, guide them to appropriate proxies for their various architectural exploration, and review the results they obtain. In general, the PathForward vendors use the information on our proxy web page to choose, download and build the proxies they use in their work. They have requested more information on building, running, and performance of the proxies, which we are presently trying to provide through new fields for each proxy on the site.

The ECP Hardware Integration and Evaluation team is using proxies in some of their sub-projects. The memory system evaluation team requested advice on which proxies to use in their studies (i.e., memory intensive behavior) and they used these in their evaluation of various HBM memories. The node evaluation team wanted to understand node-level behavior characteristics of the V1.0 proxy suite apps in order to choose the right proxies for their studies. We pointed them to the first assessment milestone report (AD-CD-PA-1110; available on our proxy app web site) to get this information.

A number of AD teams are developing proxies and using them for testing and prototyping in their development processes. In most cases those proxies are too immature to be shared outside the projects. Where appropriate we will work with AD teams and co-design centers to help them expose their proxies to a wider audience.

Some ST projects are using proxies for testing and evaluation of their tools. We have provided assistance to both the ROSE compiler team and the flang team to select and use proxies.

## 6 Conclusion and Future Work

Since the inception of the project, we have released the second suite of ECP proxy apps that comprises applications that cover algorithms in primary application areas including stencil computation, adaptive mesh refinement, molecular dynamics, machine learning, particle-in-cell, graph computation, FFT, neutron transport, and I/O and communication patterns. We have a standard process for distribution and build, a website that provides basic information and easy download, standards for developing, documenting, and contributing a proxy, and we have some metadata with each proxy, soon to include performance data generated from our proxy/parent pair assessments.

Although we've established some fundamental infrastructure, we are focusing on improving some primary aspects of the suite and the method and manner in which it is curated:

- Proxy build and run information and performance data: We are working on a standard method of posting early build/run information, problem size and scaling information for each proxy/parent pair on the suite web page. This will serve as a primary communication mechanism between this project and the PathForward vendors, who frequently request this information. We are also working on a method to deliver the performance information we have generated for proxies in the suite and their respective parents. This will be helpful in choosing appropriate proxies for specific hardware architecture design studies and in understanding how well the proxy represents its parent with respect to hardware performance characteristics.

- Algorithmic coverage: Develop or adopt a set of computational dwarves that are applicable to ECP applications to better track algorithmic coverage in the proxies.
- Mechanism for submission and standards-checking of new proxies: We really need an automated mechanism for development teams to submit and standards-check their proxies. In terms of standards, we have developed a set for proxy development, but really need to further develop these standards, making them more comprehensive, then determine an efficient means of communicating these to the application development teams.
- GPU-capable proxies: We need to identify and adopt more GPU-enabled proxies into the suite. We also need to modify the webpage classification to include GPU in a similar way that we classify proxies by implementation language so a user can quickly identify those proxies that are GPU-capable.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.