© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

C. Makrides, J.A. Ellis, S. Abbott, L. Roskop, M.E. Rowan, R. Pankajakshan, J.J. Elliott, A. Pietarila Graham, J. Hill, D.F. Richards, and B.R. de Supinski, “The Design and Organization of the El Capitan Center of Excellence for Application Readiness,” in SC26: International Conference for High Performance Computing, Networking, Storage and Analysis, 2026.

https://doi.ieeecomputersociety.org/10.1109/SC51908.2026.00030

The Design and Organization of the El Capitan Center of Excellence for Application Readiness

Constantinos Makrides
Hewlett Packard Enterprise
Spring, TX, USA
makrides@hpe.com

J. Austin Ellis
Advanced Micro Devices Inc.
Santa Clara, CA, USA
austin.ellis@amd.com

Stephen Abbott
Hewlett Packard Enterprise
Spring, TX, USA
stephen.abbott@hpe.com

Luke Roskop
Hewlett Packard Enterprise
Spring, TX, USA
luke.roskop@hpe.com

Michael E. Rowan
Advanced Micro Devices Inc.
Santa Clara, CA, USA
michael.rowan@amd.com

Ramesh Pankajakshan
Lawrence Livermore National Laboratory
Livermore, CA, USA
pankajakshan1@llnl.gov

James J. Elliott
Sandia National Laboratories
Albuquerque, NM, USA
jjellio@sandia.gov

Anna Pietarila Graham
Los Alamos National Laboratory
Los Alamos, NM, USA
annap@lanl.gov

Judith Hill
Lawrence Livermore National Laboratory
Livermore, CA, USA
hill134@llnl.gov

David F. Richards
Lawrence Livermore National Laboratory
Livermore, CA, USA
richards12@llnl.gov

Bronis R. de Supinski
Lawrence Livermore National Laboratory
Livermore, CA, USA
desupinski1@llnl.gov

***Abstract*—El Capitan is the first US exascale system for national security. Its Center of Excellence (COE) enabled co-design and application readiness. The COE's central goal is to prepare applications to use the AMD Instinct™ MI300A APU and Cray Slingshot interconnect effectively on the first day of delivery, and continuously improve after delivery. Application engagements are driven by a team of eight engineers from HPE and AMD who support approximately twenty applications from three national laboratories. Quarterly hackathons had attendances reaching upwards of fifty participants.**

We describe the technical scope, engagement model, and organizational structure that supported COE activities, emphasizing the key components of a high-performance computing COE. We highlight five select case-studies that demonstrate technical challenges in addressing functionality and performance across many software and hardware updates, often utilizing experimental software to overcome issues. We summarize lessons learned and AI impacts that may benefit future COEs.

***Index Terms*—High performance computing, Exascale computing, Graphics processing units (GPUs), User experience**

I. Introduction

Institutions across many industries use Centers of Excellence (COEs) to deploy dedicated teams to focus on specific tasks or challenge problems. In the context of Department of Energy (DOE) supercomputing, COEs frequently support application readiness on upcoming supercomputers [1]–[4].

A prior COE significantly helped transition Advanced Simulation and Computing (ASC) applications to NVIDIA GPU acceleration on the Sierra system [2]. The move to GPU computing was part of the DOE initiative to modernize applications for the exascale era of supercomputing. As the deployment of exascale systems, such as Frontier and El Capitan quickly approached, that effort gained even greater urgency.

AMD GPUs were chosen for both Frontier and El Capitan. From the formation of the COE in 2019, COE and ASC staff worked to port codes from CUDA to HIP and to test them on systems with AMD GPUs. This effort highlighted the first objective of the El Capitan COE, to ensure applications build and run correctly on AMD GPUs using ROCm™, AMD's open source stack for GPU programming, often for the first time.

To properly test both export-controlled codes and open-source proxies, several restricted and open access systems were made available. Figure 1 shows an approximate timeline of the relevant systems used by the El Capitan COE. Each new system brought the application effort closer to the final El Capitan hardware with increasingly more sophisticated AMD GPU capabilities. The final El Capitan networking hardware arrived in 2021, deployed on Livermore Computing's (LC's) unclassified Restricted Zone (RZ) with the RZNevada system, which already used HPE Slingshot [5] 200 Gbps "Rosetta" switches, and was later upgraded from Mellanox CX-5 NICs ("Slingshot-10") to HPE Slingshot 200 Gbps "Cassini" NICs ("Slingshot-11").

MI300A-based nodes became available in late 2023, approximately one year before completion of El Capitan deployment. Distinct from discrete GPU systems, this Accelerated Processing Unit (APU) features a unified physical memory design

SC26, November 15-20, 2026, Chicago, Illinois, USA
979-8-3195-4789-7/26/$31.00 ©2026 IEEE

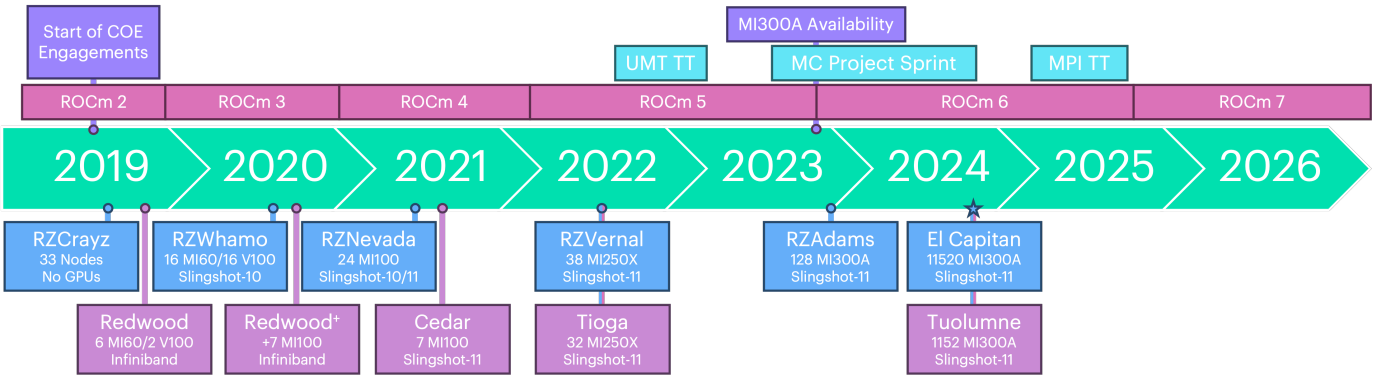


Fig. 1. Timeline of Available Systems, ROCm Version Dates, and Major Events from the onset of the Center of Excellence. Systems are listed below the time line with restricted-access and collaborative-access systems depicted by the blue and violet colors, respectively. Major ROCm versions and their approximate boundaries are listed right above the timeline. The start of COE activities with applications teams and initial availability of MI300A APUs are indicated by the purple boxes. The application engagements highlighted in Section IV are depicted by the teal-colored boxes.

that eliminates the need for expensive memory copies and simplifies porting by removing the distinction between host and device memory spaces [6]. Subsequent work has demonstrated it provides large performance uplifts; the ability to optimize application codes on early access systems based on prior AMD GPUs (MI60, MI100, MI250) was essential for realizing the full benefits of the MI300A.

The availability of new hardware was accompanied by advances in the underlying software. Figure 1 overlays the approximate releases of major versions of ROCm. The rapid cadence of ROCm releases, especially in the early days of the COE, ensured continued application progress. While we only depict ROCm major versions, other key software, such as Cray's compilers and MPICH, continually co-evolved with ROCm releases. The speed of these software releases proved challenging as well as beneficial, as regressions did occasionally occur as did incompatibilities between components.

A COE's main purpose is application readiness for the forthcoming system. The composition and organization of a COE needs to be robust enough to adapt to a continuously changing hardware and software landscape, while providing expertise to application teams. Beyond direct support of the application portfolio, the whole software ecosphere must be validated, including resources such as compilers, system libraries, profilers, and debuggers.

The paper is organized as follows: Section II goes through the composition and organization of the El Capitan COE detailing the employed application readiness strategy. Section III details some of the specifics of what COE-application team engagements encompass with select applications-team engagements detailed in Section IV. Finally, we conclude with some outlook and lessons learned in Section V.

II. Organization of the COE

Following the successes of previous COEs [2], a COE for the El Capitan Supercomputer was a mandatory requirement with the selected vendor partners, HPE and AMD, both providing personnel to join a single El Capitan COE in early 2019. The COE's organization and operation have continued to evolve since then. Operational components, for example issue-tracking systems, needed to be created and to evolve based on the feedback from code teams and COE engineers. The current organization of the COE as described here is the culmination of feedback and adjustments made over its lifetime.

A. Composition of the COE

The El Capitan COE consists of individuals from vendors, both HPE and AMD, and staff from each of the participating NNSA laboratories. The vendor portion of the El Capitan COE is comprised of eight full-time equivalent (FTE) engineers, with an additional half of an FTE being reserved for non-technical functions, such as organizing periodic meetings with COE leadership and satisfying reporting obligations. The FTE counts are fulfilled by individuals from a pool of engineers shared mainly with the other Collaboration of Oak Ridge Argonne and Livermore-2 (CORAL2) COE operating at the time, the Frontier COE. This pooling allowed for expertise to be shifted on an as-needed basis between the two COEs, as well as knowledge dissemination over a broader group. While the Frontier and El Capitan architectures are quite different, there are more similarities in the software stack where insights from one organization could translate directly to the other. As a result of the pooling, the eight FTE vendor engineers are ultimately staffed by more than eight individuals from both HPE and AMD.

COE staff members come from a number of unique backgrounds. Almost all COE members have advanced degrees in natural sciences, mathematics, or computer science. Moreover, there are both engineers that have had long careers in HPC and others that come from more academic careers. The ability to draw from different technical perspectives bore fruit later on when encountering challenging problems as differences in interests and backgrounds often lead to specializations in different areas.

The vendor COE is complemented by a lead from each of the NNSA Tri-labs, Lawrence Livermore National Laboratory (LLNL), Los Alamos National Laboratory (LANL), and

Sandia National Laboratories (SNL). The team leads collectively coordinate efforts between vendor personnel and lab staff and are in general liaisons for the respective organizations. The COE leads organize efforts into six-month work-periods where expected engagements are recorded in a work plan. The work plans are written with the intent for flexibility as issues with emerging software stacks may take precedence over regular application readiness efforts. Work plans conclude alternating between a written report and a presentation detailing COE activities over the past work period and a proposal for the subsequent work period.

This six month cadence provides continuous opportunities to evaluate application readiness and also set future expectations. This would normally be a significant burden on individual application teams, as it distracts from normal operations; however, the assignment of a COE engineer as a point of contact (POC) for each team lessens this impact significantly. In practice, these work plan boundaries are often transparent to the individual application teams allowing them to focus on application development while providing a view into their application readiness to their organizational leaders.

B. Application Readiness Strategy

The main drivers of application readiness fall into three broad categories:

1) Direct Application Engagement
2) Application Independent Research
3) Special Events

Direct application engagement are COE engineer interactions with each of applications selected by lab leads in a given work period. A liaison from the COE is designated as the primary POC for each application team and typically drives the engagement for each team. Fifteen to twenty primary application teams were supported at any given time, with the majority of the engagements tracing back to the start of the COE in 2019.

The fifteen to twenty application suite translates to approximately two or three applications for each vendor COE staff. This was found to be a reasonable workload for the COE engineers with many of the early engagements being exclusively driven by the POC. There would be more fluid engagements between application teams and many COE engineers outside of their POC as the project matured. The nature of these direct application team engagements are further discussed in Section III.

The COE primarily interacts with the developers who write and maintain the applications that run on El Capitan. Other users on the system may directly raise issues for the COE to investigate via the issue tracking system described in Section II-C, through the associated code developers of their application, or via other institutional support structures.

The next category of application readiness is application independent research. The software or libraries in this category are generally shared across many individual applications. Prime examples of this are the performance portability libraries RAJA and Kokkos, which were heavily used by the tri-lab applications. These engagements are typically many-to-many and have well defined needs. The software that typically fall into this category often use experimental or cutting-edge support from languages and tool-chains, which in turn may experience breakages or regressions in performance with new releases.

Lastly, the COE leadership organizes and participates in a number of events. Presentations and webinars are sought on roughly a monthly cadence. Topics can vary from software and hardware updates, findings on key system software, or case studies highlighting guidance and best practices for optimal performance. Presentations on newly available features or hardware are often scheduled shortly before availability to the user group to maximize effectiveness.

The COE schedules four hackathons a year alternating locations between the NNSA laboratories. COE staff and application team work together over a three- or four-day period with the emphasis on either focused development or progress towards research questions. There are optional jump-start days where code teams may meet with COE engineers early so that any issues that would impede direct engagement with the COE on the days of the hackathon could be resolved early. Hackathons also present an opportunity for users outside the portfolio of COE-supported applications to obtain dedicated support from the COE.

Hackathons were found to be important drivers for application readiness. Specialized help can be brought per request from application teams, and issues can be iterated on in an expedited fashion. Hackathons provide a clear venue for guided help with specialized tools such as parallel debuggers or performance characterizing tools. This is especially true for tools like debuggers, where the typical developer rarely uses such tools, while COE engineers encounter many problems that require heavy-weight tools thereby developing expertise that can be shared with application teams. Hackathons were expected to start shortly after the COE kickoff meeting in 2019, but the COVID pandemic forced the first hackathon to be conducted in late 2022 shortly after the availability of AMD Instinct™ MI250X-based systems.

Beyond the COE, there were a series of CORAL2 working groups that intersected general areas of interest for HPC applications. COE involvement mainly focused on four working groups: Compilers, Messaging, Tools, and System Software Packaging. In each of these forums, COE engineers could advocate on behalf of their application teams to developers and project leads for the respective groups. These meetings were another avenue for application teams to raise concerns or issues to developers of system software. Moreover, tickets filed in the bug tracking system can be highlighted and prioritized directly to the developers responsible for fixing the underlying issues.

C. Issue Tracking

To encourage reporting of any issue encountered by application developers, regardless of how minor, the COE established a dedicated issue tracker with a customized workflow. Figure 2 shows a workflow for bug reports made to the issue-tracking

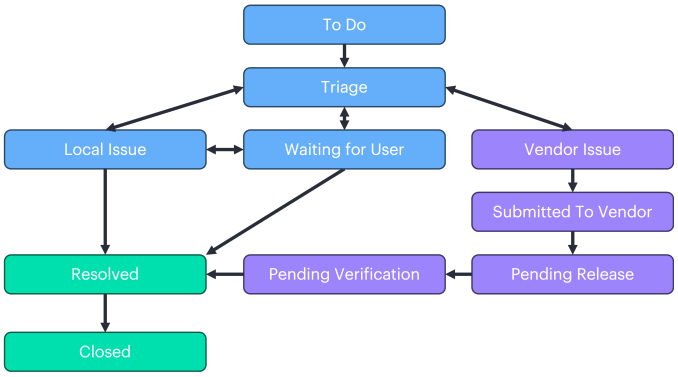


Fig. 2. Bug Tracking Workflow.

system. New tickets are distinguished from tickets in triage in order to track active investigations by engineers and prioritize efforts. Triaging of tickets is the most time consuming step for the COE as it often involves creating reproducers that can be shared to system software developers. The creation of reproducers was often a challenging problem as most of the supported application codes are export-controlled. Moreover, as the software stacks matured, more nuanced issues emerged including small performance regressions which often have additional complexities in creating a tractable reproducer. Section III-A provides a more in-depth description of the triage process and how performant workarounds are sought for application teams.

Tickets were placed into one of three statuses following triage:

- Vendor Issue: Resolution will need lab-external vendors to fix an underlying issue
- Local Issue: The diagnosis suggests an issue with a local software component
- Waiting for User: More information or a discussion is needed with the submitter

Most issues follow the "Vendor Issue" track depicted by purple boxes in Figure 2, where fixes will be implemented by external vendor research and development (R&D). This branch of the workflow involves COE engineers adding the issue to the vendor's issue tracking system and providing updates and additional information to and from the vendor teams. COE members often work towards providing performant workarounds while formal solutions are developed. Once R&D provides a software release version for the fix, the issue is placed in the "Pending Release" state and tagged with a software release version to validate against. Issues are only resolved once releases with fixes are tested against the original application. Verification is an important step as non-performant workarounds may have been implemented.

The shepherding of tickets between different organizations was performed by COE leadership until more staff were brought in to support issue resolution, highlighting an important aspect of a COE in which support gaps are temporarily handled by COE engineers. While necessary, the stopgap behavior is not optimal for longer periods of time as time is taken away from other critical COE activities such as performance optimization.

III. COE-Application Team Engagements

In this section we describe in detail the interactions between COE engineers and applications teams. The exact engagements between the application team and the assigned COE liaison naturally varies from team-to-team and changed in character as the delivery of El Capitan approached. Application readiness was initially quantified into three levels:

- Level 1: Basic Building and Testing: Can perform at least a limited build with at least one compiler set and running smoke tests
- Level 2: Complete Build and Performance Model: All build options are validated, comprehensive testing passes, a performance baseline can be established
- Level 3: Continuous Optimization and Testing at Scale: Continuously improving and evaluating performance.

These categories were designed to be high-level overviews where progress could be evaluated by leadership teams and draw attention to applications that could make use of more resources.

El Capitan is the first system for many of the COE engagement applications where AMD GPUs and the Heterogeneous-Compute Interface for Portability (HIP) programming language are used. There were early successes in running utilities such as `hipify`, which automatically processes source files and makes the appropriate CUDA to HIP ports. Modifying build systems in contrast was a larger issue that the COE needed to address. For example, CMake is a common build system used by many of the application teams, with HIP support officially being added in CMake-3.21 in July 2021, a full year and a half after the start of the porting efforts by the COE.

OpenMP offload is the GPU-offload strategy for Fortran codes. Most Fortran applications had previously used CUDA-Fortran, OpenACC, or OpenMP-4.5 as their offloading strategy following the previous systems at LLNL. Direct ports of OpenACC and OpenMP 4.5 to the OpenMP-5.0 (later OpenMP-5.2) compliant Cray Compiler Environment (CCE) Fortran compiler were less of an issue. Issues were typically encountered where applications relied on vendor-specific or undefined behavior.

Most application teams reached Level-2 of application readiness relatively quickly, although progress towards Level-3 was much more nuanced and subjective. Many of the applications have numerous options and features that can be enabled, thus complicating the success criteria towards Level-3 of application readiness. Furthermore, establishing performance baselines for fair comparisons was complicated, in part, by the application teams' ongoing code development. There were several cases where performance tuning and optimization on AMD GPUs also translated to other systems and programming models.

It was eventually more useful to further decompose applications by options or features and proceed with supporting high-priority components. Nevertheless, this three-level application

readiness set a good precedent as to the expectations of application readiness as the final system approached.

A. Hierarchy of issues discovered

Understanding the hierarchy of debugging complexity is essential to build an effective triage practice. There are three dimensions to issues that largely determine difficulties in resolution:

1) Determinism: Failure modes consistently or intermittently occur
2) Scale: The minimum number of nodes needed to trigger the failure mode.
3) Failure mode: Crashes, hangs, or silent failures

These dimensions combine and form a spectrum of issue difficulties. A deterministic crash on a single node sits at one end of the spectrum, while a non-deterministic hang at thousands of nodes sits at the other. Assessing these components for a particular issue dictates the debugging strategy, tools, and expertise required.

Deterministic crashes at small scale provide clear signals, manageable reproduction environments, and lend to fast iteration cycles. For example, a single process that exhibits a segmentation fault on a GPU can typically be triaged and reproduced in a short period of time. Creating reproducers for such cases is typically straightforward.

Compiler code-generation issues frequently fall into this category of tractable issues. Optimization passes can introduce correctness bugs or crashes that can be isolated to a specific region of source code. An effective workaround strategy is through the use of directives, such as `#pragma clang optimization off`, to guard the offending code section. This technique allows application teams to remain productive while a fix is released. Fixes integrated into the vendor software stack can require multiple release cycles to reach users thus highlighting the COE's emphasis on finding performant workarounds for application teams. Workarounds are best shared through a combination of formal documentation and deliberate word-of-mouth to high-productivity users who organically share knowledge across a large user base. Both are important although social propagation often proves more effective than documentation alone, in which case hackathons can amplify successes to a large audiences.

Issues that only manifest at higher node counts pose more challenges in debugging, largely because reproduction itself becomes expensive and constrained. Scale thresholds tend to introduce discrete jumps in difficulty based on nodes size. The COE recognized the following thresholds:

- 2 Nodes: Multi-node threshold
- 64 Nodes: Queue constrained
- 128 Nodes: System constrained
- 1000+ Nodes: Only approachable on El Capitan

The one to two nodes threshold is the least restrictive on approachability with application local issues being the most common culprit. The next threshold at approximately 64 nodes is where queue wait times lengthen, thus prolonging iterative tests. Issues at 128 nodes exceed sizes for smaller production systems necessitating investigations on larger systems such as Tuolumne and El Capitan. Lastly, any issues that only appear on over 1,100 nodes can only be approached on El Capitan.

Hangs are particularly challenging at the larger thresholds. Unlike crashes, hangs produce no stack trace or error signal. The root cause frequently lies in race conditions within the GPU runtime or network communication corner cases that appear only under specific timing and concurrency conditions. Non-deterministic hangs at scale represent the most challenging class of problems. Reproduction is uncertain, the debugging surface is enormous, and root-cause analysis may require coordinated inspection of GPU state, network state, and runtime internals. Tools such as `rocgdb`, LLNL's Stack Trace Analysis Tool (STAT) [7], and other platform-specific debuggers are essential for these investigations, however, these issues tend to be sensitive to any changes such that failure modes, including rates of failures, can be influenced by simply using debuggers. This particular case also highlights the need to co-develop tooling that reliably operates at large scales so that solutions can be delivered in a timely manner.

Silent correctness issues, where the program completes without error but produces an incorrect answer or result, occupy a uniquely difficult position in the hierarchy. These issues emerge from use-cases that exercise the GPU software stack in ways that standard test suites do not cover. Failure modes in these cases are often detected indirectly. For example, a small numerical error can propagate until a more visible secondary error is emitted originating elsewhere in the simulation. Investigation into root causes can be challenging due to large differences in the corresponding code base from where the visible errors occur and the original silent error.

Complex application codes will inevitably exercise corner cases that no test suite anticipates. The challenge compounds when the affected code is export-controlled as source cannot be easily shared. This highlights the important role in having long-term relationships between COE and application staff. These relationships developed over time lower barriers in having experienced staff in the COE work with application teams on issues that can span weeks or longer.

A well-functioning COE invests in developing engineers who are experts in approaching the challenging issues described in this section. Simultaneously, newer team members can gain experience on the more tractable issues in order to develop expertise through mentorship and progressively challenging assignments. The strategic goal is to both solve the problems at hand and build the institutional capacity to solve future problems more efficiently. Future COEs can benefit from recognizing early that non-determinism and scale largely dictate the difficulty of the issue, and maximize long-term effectiveness by strategically structuring their teams, tooling investments, and vendor relationships.

B. Obfuscation and Creating Reproducers

The creation of tractable reproducers is an important function of the COE, although there are challenges in providing

reproducers for export-controlled codes. Source code from these applications cannot be shared with vendor engineering teams and bringing expertise, such as compiler or runtime engineers, on these systems is not always possible. The COE developed a set of techniques and tooling to expose issues from these applications to vendor engineering teams, ranging from lightweight diagnostic collection to systematic code reduction.

A useful first step in triaging issues at scale is to collect diagnostic information in parallel across all ranks of the failing job. The COE developed several scripts to extract key metrics from jobs at scale. These scripts include obtaining backtraces from each rank, `dmesg` collection, wrapping the debugger `rocgdb` to attach each rank and capture various application-specific quantities, and so on. Once a debugger can be attached to all processes in a job, very narrow cases can be triggers to specific actions. For example, capturing a user-generated signal to collect backtraces in a hanging-application.

Patterns can often be found in specific backtraces distributed across a subset of the total threads in a simulation. These backtraces may implicate a particular library or even a section in the user's code and may be sufficient in guiding the COE engineer on the next part of their investigation.

COE engineers have made use of several workflows that utilize information from intermediate compiler steps or reports from the compiler. For example, one workflow attempts to create an obfuscated reproducer to a compilation error through the use of intermediate LLVM bitcode (`.bc`) files generated by the `--save-temps` compilation flag. Tools such as `llvm-reduce` can then minimize the IR while preserving the exact failure mode, producing a compact reproducer. The resulting reproducer may be sufficiently obfuscated from the original source code such that it is then shareable to vendor teams. This exact workflow is further described in Section IV for the Monte Carlo Transport code and has proved effective in isolating and ultimately addressing compiler issues in many other applications. We note that some of these automatic tools can take excessive amounts of time to produce a compact reproducer, in which case other approaches would be more effective.

Another such workflow utilizes the set of compiler flags `--save-temps` and `--Rpass-analysis=kernel-resource-usage` to inspect register pressure and spill behavior of GPU kernels. Such information can be instrumental in identifying and fixing code-generation bugs and regressions in regard to performance, or in providing insight to resource requirements of particular kernels.

These workflows highlight some of the techniques used by COE engineers in obtaining diagnostic information or automatically generating reproducers. COE engineers use these techniques to connect issues in export-controlled codes to vendor engineering teams and ensure timely fixes for application teams by creating tractable reproducers.

IV. Success Story Case Studies

The following section highlights five case studies that showcase the value of having a COE and the benefits in resolving key issues. Three of these engagements can be seen in reference to the timeline of the COE in Figure 1, with each engagement represented by a teal box. A summary of COE impact is found in Table I. The case studies described here are all subsets of the full suite of longer-term engagements with the application teams.

TABLE I
Summary of COE Impact Across the El Capitan Readiness Campaign

Metric	Value
Applications supported	20+ across 3 labs
Early access and production systems supported	11 systems
Hackathon events held	18+
Cumulative hackathon participants	∼200+
Representative Performance Improvements	
MC kernel latency	300 ms → 15 μs
Atomic add intrinsic (compiler codegen)	3.5–4.1×
UMT GPU kernel (threading improvements)	∼46×
Linker throughput (lld optimization)	3.5–4.5×
Empire throughput	12×

A. LLNL Monte-Carlo Transport Project

Engagements with the Monte Carlo Transport Project (MC) team at LLNL focused on their suite of particle and implicit thermal x-ray photon transport codes. The MC team maintains a large code base with multiple variations and features. One can decompose the project along three axes: history-based versus event-based [8], with or without nuclear data libraries, and accelerator versus CPU [9]. The nature of Monte Carlo methods, and the amount of options supported by the MC team in particular, result in a high code complexity for the main computation routines. At the most extreme edge lies history-based with nuclear data libraries on the accelerator. This use case involves a small number of extremely large kernels with many branches, but even the simpler event based kernels are more complex and more branching than the majority of GPU accelerated code. The MC team deploys an extensive test suite to deal with this complexity.

The MC team started from existing code infrastructure for running on the Nvidia V100 GPUs of the previous Sierra HPC system at LLNL [9]. This allowed the developers and COE to focus on platform-to-platform and on device-to-device differences. The COE engagement with the MC team fell broadly into two categories: compiler functionality and performance tuning. For this case study, we focus on the former and briefly summarize the latter.

1) Compiler Functionality Issues: When MI250X hardware arrived with RZVernal (Fig. 1) the MC team's large CPU test suite mostly passed while their extensive GPU test suite mostly failed. In the most extreme cases, such as when the nuclear data library was used, compilation with either AMD or HPE's HIP compilers would fail. These compiler failures are characterized as compiler crashes or as impossibly long link times. The COE consulted with the application team, LLNL management, and the vendor engineering teams to arrive at the consensus that a

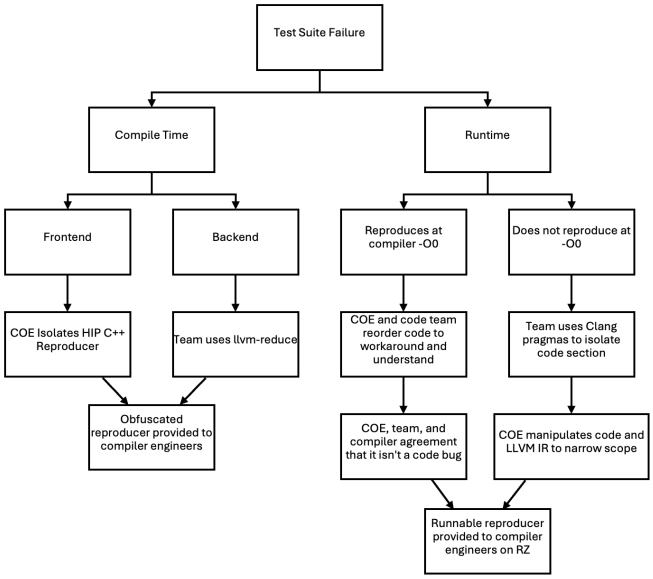


Fig. 3. Workflow used to address compiler issues encountered by the Monte Carlo Transport Project. Test suite failures believed to be compiler issues are triaged to either compile time faults (the compiler crashes) or runtime faults (the code crashes). Compile time issues typically resulted in obfuscated C++ or LLVM IR reproducers provided to AMD, whereas runtime issues are isolated as far as possible and examined by AMD compiler engineers on secure LLNL systems.

high-touch COE engagement was required to ensure the MC team could run on the final system.

The MC team developers, COE, and AMD compiler and runtime engineers met weekly for approximately two years beginning in 2023. Some features of the combined MC code stack were deemed high risk and the collaboration recommended they be eliminated. For example, the nuclear data library GIDI+ [10] and the MCGIDI interface featured runtime polymorphism and recursion. Supporting virtual functions on the GPU is complex and had only recently been supported by AMD's compiler, so the application team refactored the library to eliminate recursion and be header-only, compacting the runtime polymorphism into compile time. [11] Some challenging attributes, such as the kernel complexity intrinsic to the code, could not be removed even though they caused significant compiler issues.

This latter class of irreducible issues showcases the true strength of the COE model. Over the course of approximately two years, COE staff worked with both the MC team and the compiler engineers to isolate, workaround, report, and fix AMD HIP compiler issues affecting the MC team. The close collaboration between compiler developers, the COE, and the MC team allowed broad deployment of techniques normally known only in the compiler development community. For example, the MC team became quite adroit at exercising `llvm-reduce` on their code to generate their own compact LLVM intermediate representation (IR) reproducers. In rare cases in which the compiler would crash before IR generation, then the COE would manually isolate and obfuscate C++ code sections.

Cases in which the compiler generates incorrect code that crashes at runtime were significantly more difficult to resolve. The COE showed the application team how to use the Clang directive `#pragma optimize off/on` to toggle optimization and inlining for specific functions or code sections. Using this technique the application team was able to selectively disable optimization until their test suite passed. COE staff would then take the full application code and, working in both high level C++ and the LLVM IR, narrow the scope of the deoptimization directives until specific regions of IR were implicated. As several AMD compiler engineers had access to the LLNL RZ systems, the COE then handed off the isolated problem to be fixed by these compiler engineers. This full triage workflow is shown in Figure 3. Note that here we loosely use the terms "Frontend" to mean any error that occurs before temporary IR can be written and "Backend" to mean any error that can be reproduced from a temporary IR file. Backend, in our usage, is sufficiently broad to include both optimization and code-generation. In all cases the workflow begins with JIRA tickets marking a test suite failure and ends with reproducers being provided to the compiler team, although sometimes newer compilers would solve the problem and cut the workflow short. A vital part of this effort was the presence of AMD compiler engineers both in the weekly meetings and on the LLNL RZ systems. AMD compiler engineers on LLNL systems were especially important for compiler errors that caused runtime failures. Internal compiler errors are, in general, easier to isolate and obfuscate. Runtime errors require data and code that cannot be provided externally, and many of these problems would not have otherwise been solved.

2) Additional Performance and Correctness Efforts: As the compiler issues resolved, the MC team turned to performance analysis. Very complex kernels often spill many registers and require large program stacks, which must be allocated in HIP scratch memory. In profiling, the MC application team found that these stack allocations could result in kernel launch delays between 600μs (for stack sizes below the 140 MB default threshold) and 300ms (above the threshold). While this threshold value is adjustable, the shorter 600μs still remained a significant bottleneck and the COE staff worked with the ROCm runtime team to redesign the stack allocation mechanism, using the MC codes as one of the tests. The new default mechanism as of ROCm 6.4.0 is that the runtime will asynchronously reclaim the dynamic stack to minimize the extra latency and reduce it to less than 15μs.

The COE also helped refine a hierarchical atomic add for 64-bit integers that performs 3.5–4.1$\times$ faster than the naive implementation [12], worked with the HPE MPI developers to improve `MPI_Reduce` performance, and resolve correctness issues in GPU-aware `MPI_Allreduce`. Since El Capitan entered production the COE and the LLNL MC team meet every other week rather than weekly, and compiler engineering joins the calls when needed. The initial combined COE, application, and compiler engineering collaborative sprint produced significant and meaningful results, with each compiler

release from ROCm 6.1.0 to 6.4.0 being able to compile and run substantially more of the MC team's test suite.

B. Teton

Teton solves thermal radiative transport equations using the discrete ordinates (Sn) method. It is written primarily in Fortran but has some C++ for memory management and library interfaces. GPU acceleration is achieved primarily through OpenMP Offload directives. UMT [13] is an open-source proxy application for Teton and reflects the structure of the key computational kernels of interest for this case study. We will focus the following discussion on UMT rather than Teton, except where the distinction is meaningful.

Some aspects of UMT's computational approach presented challenges to be overcome by the COE, the application team, and the compiler developers. First, UMT primarily uses user-defined derived types and uses `pointer` over `allocatable` for allocatable variables. Second, the key computational kernels use hierarchical parallelism rather than collapsible flat parallelism. For this case study, we focus on this second set of challenges and how they were addressed through a "UMT Tiger Team," although we will briefly discuss the first and how the APU design mitigated it later in this section.

1) Hierarchical Parallelism: The Upstream Corner Balance scheme implemented in UMT decomposes the spatial domain into zones (elements), sub-zonal corners, angles, and frequency groups. [14] In addition to this standardized physical discretization, many applications add logical decompositions to allow flexible domain decomposition and partitioning. To organize this complexity most developers use compound data types, such as C/C++ structures or Fortran derived types. The organization of these data structures informs the parallel algorithm and level of parallelism. UMT orders the computational degrees of freedom into hyper-planes of zones, with zones temporarily grouped into *sets*.

The sketch in Algorithm 1 shows a characteristic computational kernel (`getPhiTotal` in UMT) where the impact of this data hierarchy can be seen. When offloaded to the GPU with OpenMP, the outer loop over $ZoneSets$ at line 2 will be distributed across teams of threads. This outer loop contains multiple groupings of tightly nested loops. Just prior to the first nested loop, the initial thread of each team calculates values common to all zone sets through convenience pointers. The nested loop at line 5 over the corners in the zone set can be fully collapsed with the loop over energy groups at line 6 and is partitioned across the threads in the team. The loop at line 11 is not partitioned across threads in the offloading region but the inner loops at lines 14 and 15 are. Each iteration of the combined loops is fully independent.

The content of the first nested loop is a simple assignment with no significant computational work. The benefit of combining it in the target region with the second nested loop is to improve cache locality, as $\phi_{total}^{ZoneSet}$ is accessed in both nested loops, and to offset the launch overhead for a relatively small computation.

Algorithm 1 An example of the nested parallelism in UMT.

```
1:  Distribute across teams
2:  for all ZoneSets do
3:      assign zone set specific pointers and values
4:      Collapse and partition across threads
5:      for c in ZoneSet corners do
6:          for g in groups do
7:              φ_total^ZoneSet(g, c) ← 0
8:          end for
9:      end for
10:     Run sequentially per team
11:     for all Sets do
12:         assign Set specific pointers and values
13:         Collapse and partition across threads
14:         for c in corners do
15:             for g in ZoneSet groups do
16:                 calculate φ_total^ZoneSet(g_0^Set + g, c)
17:             end for
18:         end for
19:     end for
20: end for
```

The hierarchical parallelism as shown in Algorithm 1 challenged early versions of the CCE compiler for several reasons. First, correct handling of Fortran pointers in offload regions took several iterations between the COE and the CCE compiler team to correctly implement. Once the pointer implementation was complete, profiling analysis showed large variations in kernel-to-kernel performance where kernels such as the one represented by Algorithm 1 performed as much as fifty times worse than expected. This performance discrepancy was prohibitive for any meaningful workload and towards the end of 2022 the COE launched a focused effort in collaboration with the HPE compiler team to understand the issues with UMT OpenMP offload performance.

The most significant cause of reduced performance determined by the COE related to how CCE compiler versions 15.0.0 and older handled OpenMP hierarchical parallelism as in Algorithm 1. Specifically, the inner collapsed loops at lines 14 and 15. Examining compiler loop-mark listing files, the COE determined that the compiler was failing to partition the collapsed corners and groups loop over threads in a team, despite an explicit `!$omp parallel do` directive, thus serializing the entire nested loop. Working with the CCE compiler team, the COE determined that the compiler was relying on auto-vectorization analysis that incorrectly detected a loop carried dependence between iterations and refused to partition the loop. The compiler team noted that CCE's OpenACC implementation more strictly followed the programmer's directives, and that they planned to modify the OpenMP implementation to similarly strictly obey directives in CCE-16. In the meantime, the COE ported some of the most poorly performing UMT kernels from OpenMP to equivalent OpenACC and saw significant speedup. A two dimensional sweep kernel went from ~ 300 ms to 6.5 ms on the MI250X.

The CCE compiler uses the same runtime for both OpenMP and OpenACC and allows this combining of non-overlapping directive models.

This relatively simple workaround allowed the Teton team to continue porting to El Capitan early-access hardware using directive languages rather than undertaking a full language port or waiting for fixes to make it to the next CCE release.

2) Additional functional and performance considerations: In this work, we focus mainly on the UMT Tiger Team efforts to exemplify how subject matter experts in a dedicated, cross-functional COE are able to work across barriers and prototype solutions under tight timelines. The Teton-COE engagement spanned nearly the entire lifetime of the El Capitan COE. Much of the early work dealt with functional issues in porting from OpenMP 4.5 code implemented on Sierra [14] to OpenMP 5.2 code. More specifically, the growing pains related to Fortran derived types with pointer components (which the 4.5 standard disallowed in target regions) as the CCE compiler implemented the OpenMP 5.2 specification.

The COE continued to work with the Teton team on performance improvements after the conclusion of the Tiger Team. A key observation during the performance investigation was the realization that the APU design of the MI300A could be leveraged with unified shared memory (USM) support in OpenMP to avoid unnecessary memory copies leading to large performance gains. The COE advised the team to wait for the availability of the MI300A-based systems before proceeding with large algorithmic changes since USM support is most effective on the APU architecture of the MI300A.

Many of the complications around derived types with pointer components and performance costs of excess data movement were resolved with the arrival of OpenMP USM mode on the truly unified MI300A APU, where pointers no longer need to be mapped and data transfers do not need to occur. Waiting to optimize on the MI300A directly with unified shared memory support would be an effective use of developmental time. This highlights the value added by the COE in communicating timelines and expectations to application teams so that informed decisions on resources are made.

The UMT effort demonstrates the flexibility of the COE to maintain both a long, incremental application engagement and a high-touch, intensive collaboration with compiler engineering to solve critical problems.

C. MPI Tiger Team

After El Capitan entered production, several MPI issues emerged that affected both functionality and the performance of critical user applications. To address these high-impact problems, a dedicated MPI Tiger Team was formed with a well defined and finite scope to resolve these issues. The team was comprised of LLNL application team members and COE engineers to identify, reproduce, and resolve problems affecting performance, correctness, and memory-consumption behavior. The formal MPI tiger team engagement started in April 2025 and lasted until September 2025 and is depicted in Figure 1 by the teal-colored box labeled as "MPI TT."

From the outset, the COE served as a central conduit for technical feedback to HPE engineering. Most of the issues discovered were encountered with export-controlled codes and difficult to reproduce failure modes. As such, direct inspection by MPI teams was not possible and support needed to be channeled through COE engineers. Moreover, some of the performance issues would only manifest for large node counts and runtimes exceeding hours of wall clock time.

As MPI fixes were identified and implemented, the COE further collaborated with HPE software packaging to accelerate delivery of evaluation MPI builds, enabling timely testing on LLNL systems. This tight feedback loop spanning reproducer development, engineering updates, packaging, and deployment allowed the MPI Tiger Team to validate fixes quickly and maintain momentum.

Through coordinated testing, the Tiger Team resolved several fundamental MPI issues, contributing directly to the stabilization of the overall software stack. These efforts addressed memory-related failures, inter-process communication (IPC) caching problems, performance degradation over time, and other issues that impacted multi-node GPU-aware MPI workloads. In particular, the IPC issues necessitated a new algorithm for on-node communication, which was accelerated in development as a result of this collaboration [15]. Experimental releases with beta-level support for this new algorithm were exercised by the COE on the problems of interest and feedback was provided to HPE's MPI team for a fully supported implementation in a future release.

Midway through the MPI Tiger Team's lifecycle, substantial progress was achieved on the initial set of goals. More than half of the identified MPI issues were resolved, and the forthcoming MPI releases demonstrated significant improvements across multiple user applications. With the most critical defects addressed and system performance stabilizing, the team began to wind down its dedicated effort. The remaining issues were either fully resolved, awaiting final validation by the application teams, or transitioned into longer-term COE work items better suited for sustained engineering engagement. The MPI Tiger Team's focused effort not only stabilized GPU-aware MPI on El Capitan but also established a clear path for ongoing refinement of the system software stack. Additionally, the improvements to the software stack are an example of how COE engagements can drive product improvements beyond the systems supported by the COE.

D. LBANN Machine Learning Engagements

The COE has a long-standing engagement with the Livermore Big Artificial Neural Network toolkit (LBANN) [16] framework team at LLNL and has collaborated on several machine learning (ML) efforts throughout the El Capitan project. While a number of improvements included ML kernel optimizations and quality of life enhancements for PyTorch from-source builds, there was a concentrated effort on RCCL reliability and performance both before and after machine production. This included debugging hangs during scale-out and improvements to collective communication operations.

The COE provided support by working through several hangs that appeared during scale-out on El Capitan over 1000+ nodes. In particular, techniques described in Section III-B, especially the at-scale collection of backtraces and usage of minimal reproducers, allowed COE engineers to determine the root cause of these issues. The debug effort resulted in new best-practices and hardening of RCCL and the AMD AI software stack, for example: (1) improved understanding of the applicability of HIP runtime debug flags to RCCL, (2) patches to RCCL addressing missing synchronization points, and (3) identification of deadlocks resulting from interactions between RCCL and Cray networking software.

Broader improvements to RCCL occurred over time due to development work by AMD's library teams, leading to better smoothness and consistency of collective communication operations such as `rcclAllReduce`, `rcclAllGather`, and `rcclReduceScatter`.

E. Atomics and Linkers

While prior sections provide deep dives into specific code team engagements in this section, we highlight the breadth of the COE's reach. Early in the AMD ROCm stack, the Sandia Parallel Aerodynamics Reentry Code (SPARC) application failed to compile due to register allocations exceeding the limits on register spillage. This was not a unique error to SPARC and represented a growing pain of a new compiler. The COE successfully provided a reproducer that led to many iterative fixes, and ultimately a compiler that not only compiles but also reports when occupancy requests are not honored.

Another issue that highlights the challenge the COE faced when working with MI250X hardware (ahead of MI300A), is that atomics perform substantially differently on each. The ElectroMagnetic Plasma In Realistic Environments (EMPIRE) application worked with the COE to rework an algorithm that used global atomics to avoid atomics altogether. This optimization provided a substantial $12\times$ performance improvement [17], which enabled novel science at scale.

The COE has also become the vanguard in providing feedback on the compiler itself. Most notable is an ongoing effort to improve link times in codes using relocatable device code (RDC). The COE worked with AMD's compiler team to co-design a linker technology "Module Splitting" that introduces some parallelization into the link time optimization passes which are used during RDC link steps. This co-design effort has led to a 3.5–4.5$\times$ speedup in link times for the EMPIRE code [18], [19], and this collaboration continues as our efforts have fostered R&D that is looking to improve compilers of the future for GPU code generation.

V. Lessons Learned

We conclude with several recurring lessons emerging from COE activities so that future teams may benefit. Compatibility issues across the software stack were a persistent source of disruption throughout the COE engagement, and were further complicated by the fact that different organizations have ownership over individual software components and release cycles were not synchronized. Changes in one component may introduce breakages in another component, and the uncoordinated release cycles led to situations where there was not enough time for COE engineers to find, triage, and report issues before the cutoff for the next software release cycle. Efforts to coordinate release cycles improved over time but with imperfect outcomes.

Although disruptions due to mismatched release cycles should be expected they can be mitigated by investing in integration testing infrastructure that catches incompatibilities early, while also advocating for tighter cross-vendor release coordination. Several attempts were made to create such a testing infrastructure but never materialized into a practical automated testing suite. Part of the difficulty was at the interplay of the maturity of the software stack and timing of such an infrastructure. Integration testing would have maximum value early in the project timeline, however the early immaturity of compilers, build systems, and other system software consumed practically all available support resources. In hindsight, an argument could have been made for delaying application engagements at the start of the COE and focusing on development of such testing infrastructures.

The usefulness of a forward looking testing infrastructure is amplified if pre-release software is readily available allowing issues to be found before official releases are made. However, such an effort would unfortunately require coordination between many groups outside of the COE.

Another stark observation is that legacy code assumptions carried forward from prior GPU platforms introduced subtle issues. Assumptions about things like warp size, implicit synchronization behavior, and build system configurations tuned to different software ecosystems all required careful auditing during porting. The prior use of non-standard conforming code or utilization of vendor-specific workarounds have also been consistent pain points especially in cases where sub-optimal performance was observed. Our experience has been that the short-term gains of such approaches often resurface as painful investigations when issues arise later on. The availability of individuals with ties to language standards committees has often been helpful in providing feedback to the committees and obtaining clarification on gray-areas in standards.

Future COE should also recognize the importance of being robust and having versatile engineers. Section II touched upon having a deep pool of engineers to draw from. This can be critical in resolutions of issues involving engineering across several organizations. The application-centric perspective of COE engineers supports these efforts by providing real application data. System noise on El Capitan was greatly improved over time [20] and the challenges of system resiliency are likely to remain of key importance in future leadership-class computing facilities, as component count and complexity continue to increase [21]. Ensuring that COE engineers can participate in such activities can both benefit issue resolution and bring best practices back to the application teams.

Looking forward, artificial-intelligence-assisted code generation is already providing concrete productivity gains in COE

work. Examples include writing microbenchmarks, investigations of MI300A chiplet partitioning modes, explaining runtime characteristics such as caching behavior and communication patterns, and so on. Using institutionally approved coding tools to generate initial reproducer boilerplate significantly reduced time spent creating new reproducers. Artificial intelligence (AI) tools have also introduced new vectors of misinformation where application developers may receive and act on incorrect hardware or software information from language model chatbots. These issues are not unique to COE engagements, but add a layer of complexity when new hardware or software features are made available. Experienced COEs can serve as safeguards against misinformation. Vendors can mitigate inaccuracies by keeping public documentation current, since that documentation is the primary source of information.

Despite the obvious utility of AI-based tools, they are not substitutes for the personal interactions provided by COE staff. AI cannot take responsibility for decisions, cannot access systems spanning two institutions the way COE staff routinely do, and cannot replace the trust and relationships that underpin effective cross-institutional collaboration. The social and institutional boundaries that COE engineers navigate daily are core functions and not obstacles to be automated away.

A common theme in the COE is increasing engagements and minimizing barriers to support. The organizational structure described in Section II describes the non-technical structures of the COE that resulted in multi-year relationships between application teams and specialized expertise from the vendors resulting in faster turn-around times when problems were encountered later on, such as the ones in Section IV. Having COE engineers with both direct exposure to codes and familiarity with common software frameworks used in these codes resulted in quicker resolutions to both performance and functional issues.

The COE ultimately proved its value by helping ensure multiple applications were ready to run when El Capitan was delivered. As of March 2025, El Capitan's system readiness L2 milestone summary described the major components of system software as operational, highlighting several unclassified ASC simulations running on up to 4096 nodes [22]. The most important keys to this success include sustained user outreach and education, direct engagements with application code teams, relentless issue tracking and prioritization, and expert COE staff with a wide variety of knowledge and skills.

VI. Acknowledgments

Some authors used AI-assisted tools during the drafting process, including for collating notes and editing. All content was reviewed and approved by the authors.

AMD, the AMD Arrow logo, Instinct, ROCm, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Sandia National Laboratories is a multi-mission 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.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
LLNL-CONF-2018586 and SAND2026-25126C

References

[1] J. R. Neely and B. R. de Supinski, "Application Modernization at LLNL and the Sierra Center of Excellence," *Computing in Science & Engineering*, vol. 19, no. 5, pp. 9–18, 2017. [Online]. Available: https://doi.org/10.1109/MCSE.2017.3421556

[2] J. P. Dahm, D. F. Richards, A. Black, A. D. Bertsch, L. Grinberg, I. Karlin, S. Kokkioa-Schumacher, E. A. León, J. R. Neely, R. Pankajakshan, and O. Pearce, "Sierra Center of Excellence: Lessons Learned," *IBM Journal of Research and Development*, vol. 64, no. 3/4, 12 2019. [Online]. Available: https://doi.org/10.1147/JRD.2019.2961069

[3] W. Joubert, R. Archibald, M. Berrill, W. Michael Brown, M. Eisenbach, R. Grout, J. Larkin, J. Levesque, B. Messer, M. Norman, B. Philip, R. Sankaran, A. Tharrington, and J. Turner, "Accelerated application development," *Comput. Electr. Eng.*, vol. 46, no. C, pp. 123–138, Aug. 2015. [Online]. Available: https://doi.org/10.1016/j.compeleceng.2015.04.008

[4] H. A. Nam, G. Rockefeller, M. Glass, S. Dawson, J. Levesque, and V. Lee, "The Trinity Center of Excellence Co-Design Best Practices," *Computing in Science and Engg.*, vol. 19, no. 5, p. 19–26, Jan. 2017. [Online]. Available: https://doi.org/10.1109/MCSE.2017.3421542

[5] D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler, "An In-Depth Analysis of the Slingshot Interconnect," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–14. [Online]. Available: https://doi.org/10.1109/SC41405.2020.00039

[6] A. Smith, G. H. Loh, M. J. Schulte, M. Ignatowski, S. Naffziger, M. Mantor, M. F. N. Kalyanasundharam, V. Alla, N. Malaya, J. L. Greathouse, E. Chapman, and R. Swaminathan, "Realizing the AMD Exascale Heterogeneous Processor Vision : Industry Product," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 876–889. [Online]. Available: https://doi.org/10.1109/ISCA59077.2024.00068

[7] G. L. Lee, "Stack Trace Analysis Tool (STAT)," [Computer Software] https://doi.org/10.11578/dc.20180510.2, jan 2018. [Online]. Available: https://doi.org/10.11578/dc.20180510.2

[8] S. P. Hamilton, S. R. Slattery, and T. M. Evans, "Multigroup Monte Carlo on GPUs: Comparison of history- and event-based algorithms," *Annals of Nuclear Energy*, vol. 113, pp. 506–518, 2018. [Online]. Available: https://doi.org/10.1016/j.anucene.2017.11.032

[9] P. S. Brantley, R. C. Bleile, S. A. Dawson, N. A. Gentile, M. S. McKinley, M. J. O'Brien, M. M. Pozulp, D. F. Richards, D. E. Stevens, J. A. Walsh, and H. Childs, "LLNL Monte Carlo transport research efforts for advanced computing architectures," in *International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering 2017 (M and C 2017)*. Korean Nuclear Society - KNS, April 2017. [Online]. Available: https://www.osti.gov/biblio/1357357

[10] G. Gert, M. Descalle, C. Mattoon, and B. Beck, "The LLNL nuclear data infrastructure for the GNDS data format," in *Proceedings of the international conference on physics of reactors-Physor 2022*, 2022, pp. 123–132. [Online]. Available: https://www.osti.gov/servlets/purl/1869527

[11] M. Pozulp, B. Beck, R. Bleile, P. Brantley, S. Dawson, N. Gentile, E. Gonzalez, J. Grondalski, M. Lambert, M. McKinley, M. O'Brien, R. Procassini, D. Richards, A. Robinson, S. Sepke, D. Stevens, R. Vega, and M. Yang, "Status of Mercury and Imp: Two Monte Carlo Transport Codes Developed Using Shared Infrastructure at Lawrence Livermore National Laboratory," *EPJ - Nuclear Sciences & Technologies*, vol. 10, 2024. [Online]. Available: https://doi.org/10.1051/epjn/2024025

[12] S. A. Dawson, R. C. Bleile, P. S. Brantley, E. S. Gonzalez, M. S. McKinley, M. J. O'Brien, M. M. Pozulp, A. P. Robinson, and M. Yang, "Progress Porting LLNL Monte Carlo Transport Codes to the AMD Instinct MI300A APU," in *Proceedings of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science*

and Engineering (M&C 2025). American Nuclear Society, 2025, pp. 1792–1801. [Online]. Available: https://www.osti.gov/biblio/2563318

[13] B. C. Yee, T. S. Haut, P. F. Nowak, R. C. Chen, and A. C. Black, "UMT," [Computer Software] https://doi.org/10.11578/dc.20211208.4, oct 2021. [Online]. Available: https://doi.org/10.11578/dc.20211208.4

[14] P. Nowak, A. Black, S. Rennich, D. Appelhans, R. Chen, T. Haut, P. Maginot, T. Bailey, P. Brown, A. Kunen *et al.*, "Porting Teton, a Discrete-Ordinates Thermal Radiative Transfer Code, to Sierra," in *International Conference on Mathematics and Computational Methods applied to Nuclear Science and Engineering (M&C)*, 2019, pp. 2128–2137. [Online]. Available: https://www.ans.org/pubs/proceedings/article-46794/

[15] Hewlett Packard Enterprise Development LP, "intro mpi - introduces the message passing interface," 2026, see section on GTL_ENABLE_HSA_IPC_SIGNAL_CACHE. [Online]. Available: https://cpe.ext.hpe.com/docs/26.03/mpt/mpich9/intro_mpi.html

[16] B. Van Essen, H. Kim, R. Pearce, K. Boakye, and B. Chen, "LBANN: livermore big artificial neural network HPC toolkit," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, ser. MLHPC '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2834892.2834897

[17] L. T. Meredith, J. Elliott, C. Glusa, J. J. Hu, D. Sirajuddin, C. Moore, and R. Pawlowski, "Scalability and Performance of the Empire Plasma Physics Code on the El Capitan Platform," Talk presented at the SIAM Conference on Parallel Processing for Scientific Computing (PP26), Berlin, Germany, mar 2026, minisymposium MS53, "High-performance Iterative Solution Strategies for Large-scale Coupled Problems – Part I of II". [Online]. Available: https://meetings.siam.org/sess/dsp_talk.cfm?p=151820

[18] J. J. Elliott, "Effectively Leveraging Multi-Institution and Multi-Vendor Relationships to Address Advanced Software Needs." Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 03 2025, presentation at the SIAM Conference on Computational Science and Engineering (CSE25), Fort Worth, Texas, USA. [Online]. Available: https://doi.org/10.2172/3020979

[19] J. J. Elliott and J. Doerfert, "Build Times on Modern Heterogeneous Systems: Steps towards a faster future." Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 10 2024, presentation at the Trilinos User Group, Albuquerque, NM, USA. [Online]. Available: https://doi.org/10.2172/2999236

[20] E. A. León, J. Glenski, M. Stock, K. McMahon, W. Loewe, C. Snyder, L. Kaplan, S. Vadlamani, T. I. Mattox, T. D'Hooge, B. Behlendorf, N. Hanford, R. Pankajakshan, and M. L. Leininger, "Breaking the system noise barrier at exascale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 411–436. [Online]. Available: https://doi.org/10.1145/3712285.3759793

[21] S. Atchley, C. Zimmer, J. Lange, D. Bernholdt, V. Melesse Vergara, T. Beck, M. Brim, R. Budiardja, S. Chandrasekaran, M. Eisenbach, T. Evans, M. Ezell, N. Frontiere, A. Georgiadou, J. Glenski, P. Grete, S. Hamilton, J. Holmen, A. Huebl, D. Jacobson, W. Joubert, K. Mcmahon, E. Merzari, S. Moore, A. Myers, S. Nichols, S. Oral, T. Papatheodore, D. Perez, D. M. Rogers, E. Schneider, J.-L. Vay, and P. K. Yeung, "Frontier: Exploring exascale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3581784.3607089

[22] M. LeGendre and A. Bertsch, "El capitan system readiness: L2 milestone summary," Lawrence Livermore National Laboratory (LLNL), Livermore, CA (United States), Tech. Rep., 07 2025. [Online]. Available: https://www.osti.gov/biblio/2584749