2020 LLNL HPC Application Survey

Developed as part of the *RADIUSS* ISCP Project
in the LLNL Computing Directorate

LLNL-TR-821536
(For internal LLNL distribution only)

Prepared by LLNL under Contract DE-AC52-07NA27344.

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Revision	Date	Comment
0.9	4/10/21	First version sent to survey responders and a few key personnel on the RADIUSS project for feedback
1.0	4/30/21	Final version for broad release – internal LLNL and LFO distribution only

Introduction and Motivation

This report represents a snapshot in time of HPC application development activities at LLNL in 2020. It is the first time a survey of this magnitude has been done since 2004, and we believe the first of this scope at LLNL. It was led out of the RADIUSS project, an LLNL institutionally-funded project managed out of the Computing directorate. RADIUSS aims to promote a common well-supported HPC open-source software stack consisting of products that are developed and maintained here at LLNL. We recommend policies and practices for open-source development and promote these open-source products for use in the scientific application development community at the lab and beyond. The goal is no less than to ensure LLNL is recognized as and remains a world-class institution leading in all areas of HPC application and software development.

The survey data was collected in early 2020 in two phases using Microsoft Forms. Phase I aimed at identifying all of the application projects at the lab and a point-of-contact. Phase II then reached out to those POCs with a detailed survey of questions. The survey took 30-40 minutes to complete on average, and the data was gathered over a two-month window of time. See Section 7 for the complete list of codes and the developer points-of-contact who responded.

In all, 73 software projects are represented, and each project only had a single responder and are thus equally weighted. When statistics are given in the charts and data, these are relative to the 73 projects unless otherwise specified where a subset of the projects were represented.

The survey questions were broken into the following sections:

- Application Description, Team Makeup, and Licensing
- Platforms and Cloud Computing
- Development Infrastructure
- Languages and Programming Models
- Dependencies
- Pain points

This report is likewise broken into sections mirroring the survey. Each subsection will provide a survey question and list of the possible choices that were presented, a short ***basis*** of what the question was intended to learn, a chart to visualize the aggregate responses, and a brief ***analysis*** of the results from the perspective of the author/editor. Some questions were naturally grouped, so some subsections will forego the basis and analysis, instead combining it with subsequent questions.

We used multiple choice questions where possible and appropriate so that we could easily compile statistics on the responses in what are hopefully easy to understand charts and graphs. When free-form answers were offered, the raw data is generally available, and attempts were made to draw broad conclusions and summaries from that data. For the choices, a list using the ○ symbol implies only one choice could be selected, and the □ symbol implies multiple choices were available.

This report does not associate any individual responses with answers in order to ensure anonymity, and the data is either presented as a statistical summary for each survey question. When open-ended questions were answered that required a textual response, the responses will not be attributed to responders in this report, and attempts have been made to either remove any phrases or names that might allow one to infer the responder.

The report concludes with a summary the open-source products developed at the lab that are foundational to many of the HPC applications developed here at LLNL and in the external community. We encourage readers to explore adopting these hardened and proven software technologies in their own applications.

For more information, to provide feedback, request the raw data, or ask about how you can get involved in RADIUSS either as a developer or a user, shoot an email to radiuss-request@llnl.gov.

Rob Neely
RADIUSS Project Lead and Report Author/Editor
neely4@llnl.gov

Executive Summary

The information gathered in this report is designed to provide information that can guide both the RADIUSS efforts and other activities around LLNL – including strategic planning and research and development plans related to scientific computing, software development, and software engineering.

Sufficiently similar survey data from other laboratories or organizations are not available for direct comparison, but one can generally conclude that LLNL software and application teams find a supportive environment for development here at the lab, and that the reader should conclude that LLNL has a mature set of practices overall for doing HPC development that matches its excellence in deploying top tier HPC systems in Livermore Computing.

When a question elicited a particularly interesting piece of data or conclusion, the author generated a "Key Takeaway" for that question, which are then highlighted as the last bit of content for each subsection. These are meant to be observations based on the data collected and not editorial in nature, advisory, or subjective. When the author wanted to highlight specific recommendations, those were usually captured in sidebar box – often consisting of a pointer to additional resources at the lab or on the web.

The Key Takeaways are summarized in the table below and form the basis of an overview of this report.

Of these takeaways, a summary of the key areas the lab should look at making improvements include:

1. Continue to support (through both software and expertise) projects looking to port to **GPU-based platforms**, with an eye toward future-proofing their software against future architecture innovations and vendor-specific programming models

2. Identify key use-cases for public (and private) cloud usage to address the needs expressed by application developers and help advance **cloud technology for HPC**

3. Continue to promote **Continuous Integration (CI) in HPC** with the GitLab deployment in LC

4. Encourage open-source products to **prioritize user documentation, developer documentation, and coding standards**

5. Promote the use of **issue trackers** (e.g., built-in to tools like are available in GitHub and GitLab)

6. Encourage **continuous performance analysis** (e.g., via Caliper and SPOT) for performance-sensitive codes

7. Continue to emphasize a culture of **sustainability in open-source software**, as the adoption of reusable software requires users to be confident in its continued support

8. Develop a **culture of software-sustainability** by funding teams to prioritize efforts that are important to long-term software health once their product has transitioned from research code into a production capability

Parsing through the free-form responses provides the reader with a long list of potential areas for strategic development, but common requests that were not necessarily captured above include, but are not limited to:

1. Provide a central organizational structure where recommended **best practices** are captured, promoted, taught, and continuously updated at LLNL

2. Provide a pool of **software engineering experts** who work with teams on a temporary or short-term basis to help get started in adopting best practices and new tool use

3. Encourage better **user and developer documentation**, and perhaps a centralized repository of pointers to documentation

4. Support **common tools** across the institution and a long-term commitment to provide and support them

5. Promote methods and tools (and perhaps institutional funding) to help **reduce technical debt** in software projects, which can act as a boat anchor on developing new innovations

6. Support **migration of codes from Fortran to C++** for those who wish to do so, both for access to better support tools, compilers, and portability layers – as well as a way to draw in new developers who may not be attracted to working in Fortran

We encourage the reader to draw their own conclusions by reading the full report and comparing their experience to that of the lab HPC software development community.

Section 1 Application Description, Team Makeup, and Licensing

Question 1: THIS PIECE OF SOFTWARE IS BEST DESCRIBED AS…

Choices:

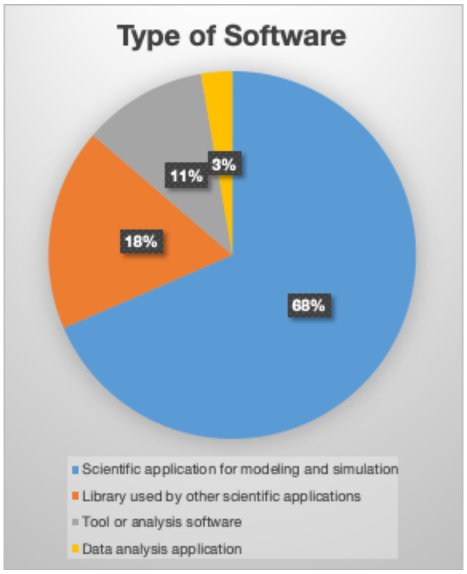
- ○ Scientific application for modeling and simulation
- ○ Data analysis application
- ○ Library used by other scientific applications
- ○ Tool or analysis software
- ○ Suite of related applications
- ○ Other [ ]

Basis: This question was intended to understand the scope of project types represented in the survey.

Analysis:

While the survey largely targeted HPC simulation applications, there were also a large number of library and analysis projects who responded, and throughout the rest of the report there is usually no distinction made.

As anticipated, most of the projects fell into the traditional category of modeling and simulation applications, or libraries that support those applications. The "Tools" and "Analysis" categories capture a broad set of functionalities, some of which arguably could belong in other categories as well. The survey also provided an option for "Other", but the editor took the liberty of categorizing those in one of the existing groups for simplicity and consistency.



Question 2: HOW MANY PEOPLE SUPPORT YOUR PROJECT?

Choices:

	0-1	2-4	5-8	9-14	15-24	25+
# FTEs	○	○	○	○	○	○
# Contributors	○	○	○	○	○	○

Basis: This question aimed at understanding several aspects of a project. The question was first broken down into two related questions:

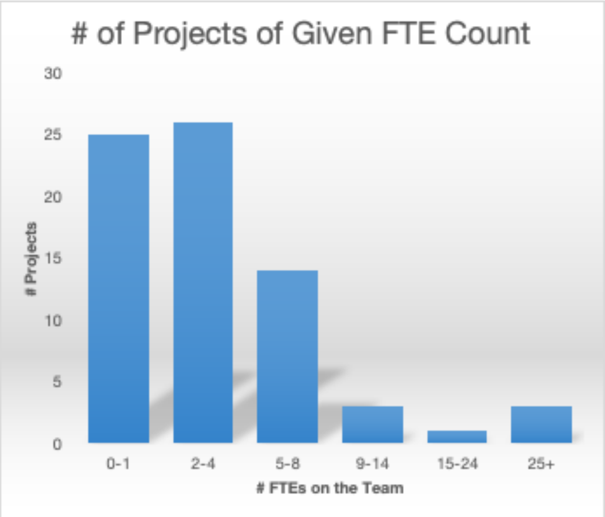
a) Contributor count (e.g., how many people are considered “on the team”), and
b) FTE (Full-Time Equivalent) count, which corresponds to the project funding for staff (the majority of most software project expenses).

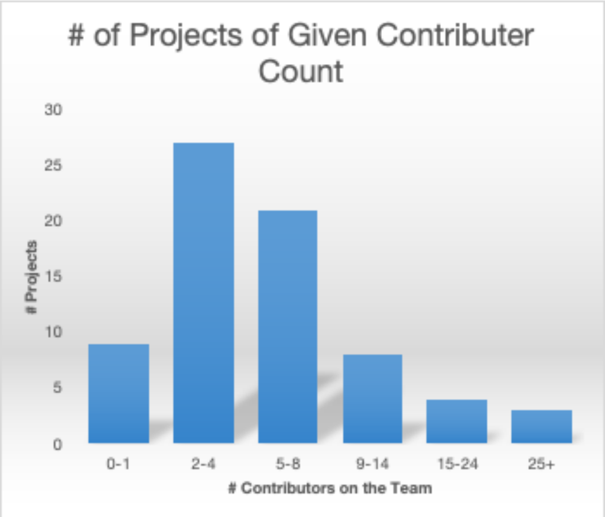
It is assumed that the number of contributors would be equal to or greater than the FTE count. If it is much greater, one might assume that many of those contributors are part-time and have other assignments as well.

The survey also asked for approximate data in a range of predefined bins:

- 0-1 (single)
- 2-4 (small)
- 5-8 (medium)
- 9-14 (large)
- 15-24 (x-large)
- 25+ (2x-large)

The labels are not as important as the numerical values and are relative to our laboratory environment. E.g., a software team of 20 people in many industries would not be considered “x-large”.





Analysis: Most HPC software projects fall into teams of 2-8 contributors at LLNL. The above graph shows relatively how many projects fell into each of the size categories and is does not attempt to draw any correlations between individual project contributors and FTE counts.

To attempt to draw correlations, we show in the table below the number of projects that reported a size in each bin category. The diagonal (in light gray) indicates projects that likely have largely fully-captured staff, as the number of contributors is roughly equal to the FTE count. Those above the diagonal represent teams that probably have some or possibly many members assigned to multiple projects. (Those below the diagonal may have misinterpreted the question, as the FTE count should not outnumber the contributor count).

# Contributors

25+	0	0	0	1	1	2
15-24	0	0	1	2	0	1
9-14	0	1	7	0	0	0
5-8	0	14	6	0	1	0
2-4	19	8	0	0	0	0
0-1	6	3	0	0	0	0
	0-1	2-4	5-8	9-14	15-24	25+

# FTE

In general, one can conclude that the larger a team, the more likely it is that the staff is dedicated to that project. Conversely, smaller projects often have a larger number of contributors relative to their size. This is not particularly surprising, and both attributes of a project have pros and cons, and a healthy organization will promote what's best for the team. A team with a much larger number of contributors than its FTE count can be viewed as a positive (lots of free energy and contributions flowing from other areas), or as a negative (contributors have little or no time to dedicate to the project and may be time sliced too thin to focus on long-term maintenance, for example).

Key Takeaway #1 Smaller projects (by FTE count) tend to have more part-time contributors, versus large projects which tend to have dedicated developers.

Question 3: WHERE DO YOUR DEVELOPERS RESIDE?

Choices:

- ○ At LLNL only (including LLNL staff working remotely)
- ○ Primarily at LLNL, with team members staffed elsewhere
- ○ Primarily elsewhere, with LLNL staff contributors

Analysis merged with next question

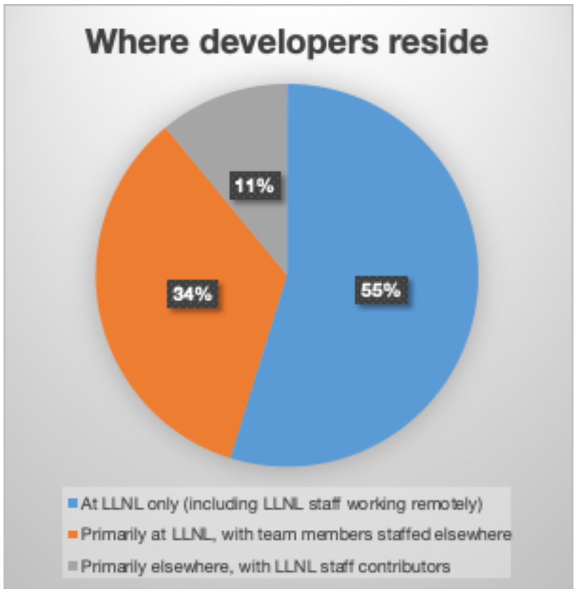
Question 4: LLNL DIRECTORATE(S) SUPPORTING DEVELOPMENT

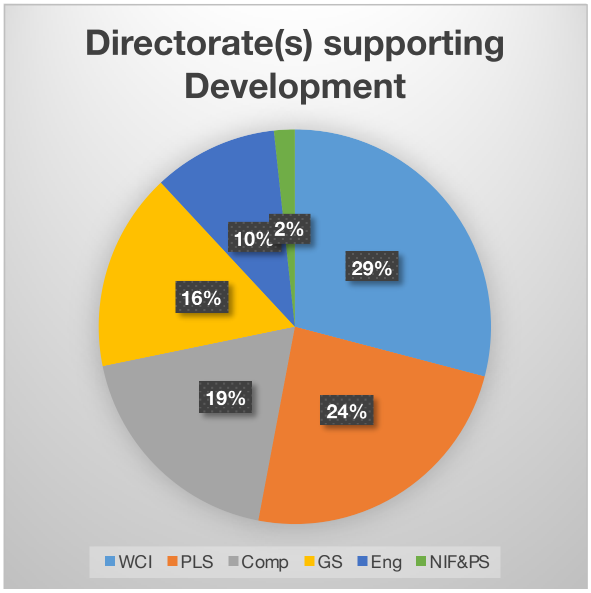
Choices:

- □ Weapons and Complex Integration (WCI)
- □ Global Security (GS)
- □ NIF and Photon Sciences
- □ Engineering
- □ Physical and Life Sciences (PLS)
- □ Computing

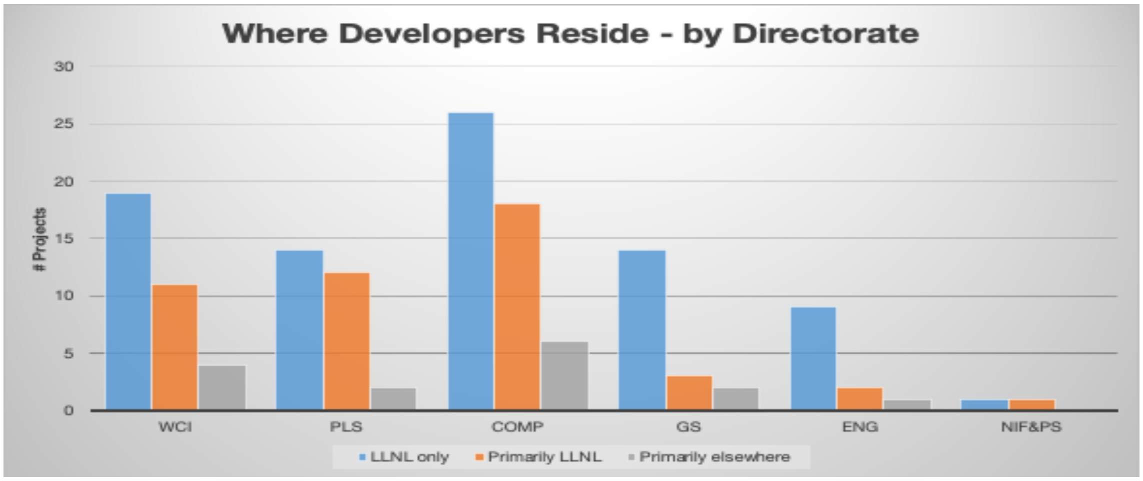
Basis: Software teams are becoming increasingly distributed – sometimes globally. Question 3 aims to identify the breakdown of projects that are exclusively or primarily developed at LLNL versus those that LLNL contributes to, but which are primarily developed elsewhere. Question 4 breaks down the support by directorate – not taking into account the project size or funding – just the number of projects. Responders were allowed to choose more than one directorate as appropriate.

Analysis: Over half of the responses were representing teams that have their developers entirely on site. But a full 1/3 of teams have developers residing elsewhere as well. Only a small fraction indicated that they were part of a team that primarily resides elsewhere, but one must suspect that there are many more of those developers at the lab contributing to applications primarily developed elsewhere and were not captured in our survey.





Looking at the data by directorate reveals a remarkably consistent breakdown.



Key Takeaway #2 Just over half of the software represented in this survey is developed exclusively at LLNL. Conversely, almost half of the software includes developers staffed at other institutions.

Question 5: Approximately what percentage of your overall project effort (by FTE count) is covered by someone with formal training (e.g. degree) in computer science or software engineering? (i.e. versus those trained in the domain science who are code developers)

Choices:

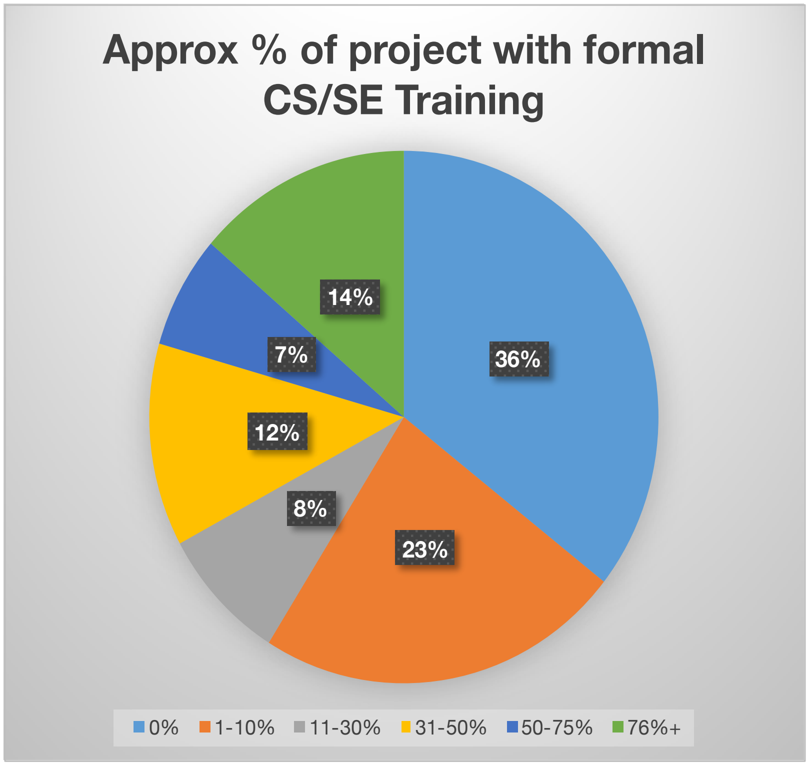
- ○ 0%
- ○ 1-10%
- ○ 11-30%
- ○ 31-50%
- ○ 50-75%
- ○ 76%+

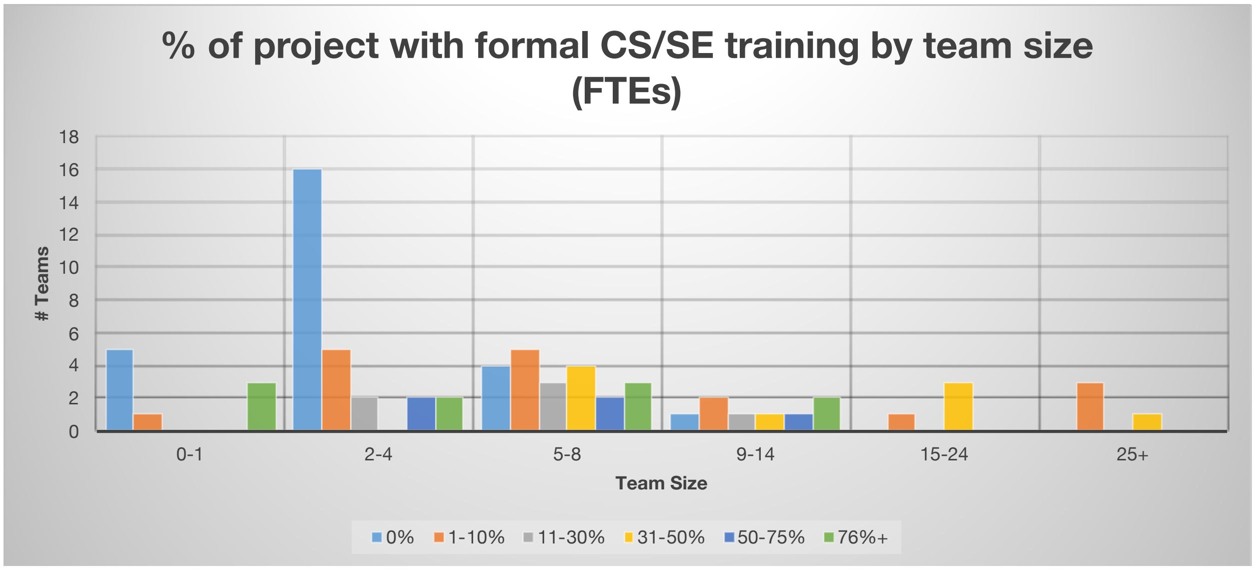
Basis: While software engineering and computer science as a discipline can be learned on-the-job without a formal degree (just like a domain science can be learned by someone with a straight CS degree), a team that includes someone dedicated to improving Software Engineering is presumably more likely to focus on improving software quality and adopting best practices – a major focus of this survey.

Analysis: Nearly 2/3 of LLNL teams have some level of formal CS or Software Engineering expertise on their team. And 21% of the projects had a majority of the team members in that category.

One might assume that the larger a project, the more likely they are to be multi-disciplinary – so we looked at the data by project FTE count as well. But equally suprising, there were no obvious correlations with even x-small and small teams incorporating that expertise in a majority of cases. Medium sized teams appear to have the highest percentage of CS/SE expertise.

It would be very interesting to see these data from other organizations and enterprises that are similar in their scientific and technical staff to LLNL.





Key Takeaway #3 LLNL overall values multi-disciplinary teams with software engineering expertise, but 36% of teams do not have a team member with formal training in Software Engineering.

Question 6: What is the release status of the code?

Choices:

○ Open Source
○ LLNL Proprietary or OUO (i.e., not export-controlled, but also not released as open source)
○ Export-controlled (e.g., ITAR, EAR99)
○ Other []

Analysis merged with next question

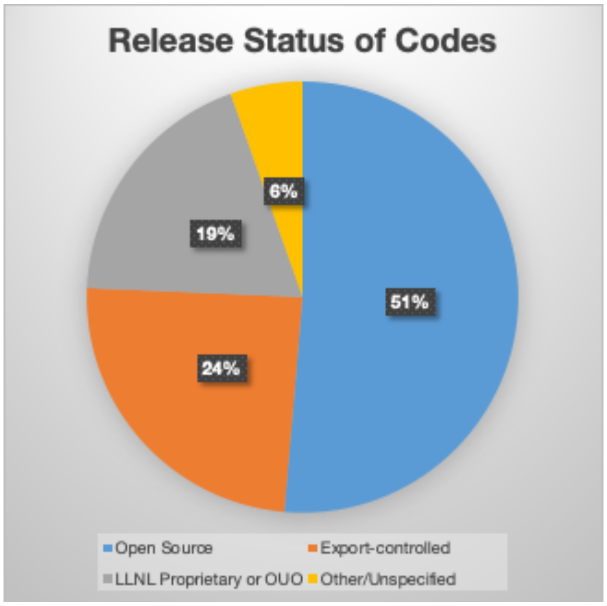
Question 7: What OPEN-SOURCE license is it released under? (Check all that apply)

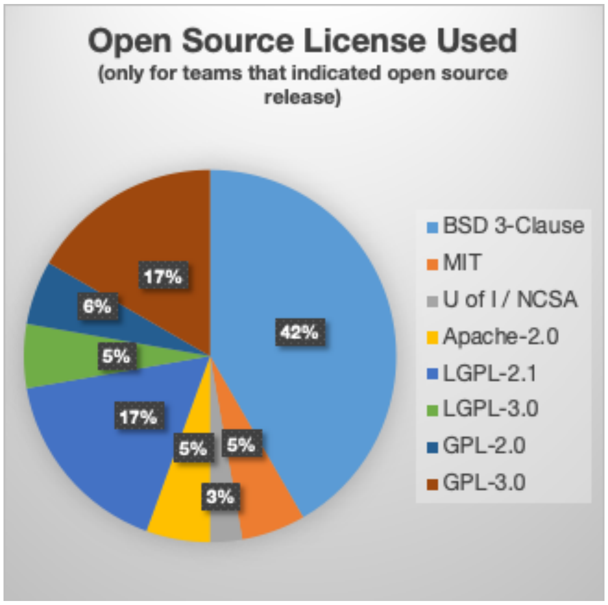
Choices:

□ MIT
□ BSD 3-Clause
□ BSD 2-Clause
□ Apache-2.0
□ Mozilla MPL-2.0
□ LGPL-2.1
□ LGPL-3.0
□ GPL-2.0
□ GPL-3.0
□ Affero AGPL-3.0
□ Other [ ]

Basis: These two questions are aimed at getting statistics about how much of LLNL's unclassified applications and software is freely available, how much we could release but choose not to, and how much falls under some level of export control that prevents open-source release. For those who selected open source as their answer – an additional survey question was presented to them that asked which license they use. The options offered were taken from guidance at the Industrial Partnerships Office, but if no one indicated some of the more obscure licenses (e.g., Affero AGPL-3.0, or Mozilla MPL-2.0) we do not include them in our data. The question was also configured to allow multiple selections, in the event that a given piece of software has multiple licenses, or multiple related projects were included in the same response that happen to have different licenses.

Note: For the question about release status, the selection including "LLNL Proprietary" had the added caveat in the survey description that the code was not export-controlled but also not open sourced – which would imply we could release the software but choose not to for reasons such as competitive advantage. However, the question also included Official Use Only (OUO) designation, which would prevent open source – so the statistics for that category may be artificially high. In retrospect, OUO should've been included in the same category as Export Controlled so as to imply "not openly releasable".





Analysis: Just over half of the software developed at the lab is open source, and just about ¼ of the software is protected under US Export Control laws. The remaining 19% are not released as open source but can presumably be shared with close partners. See the caveat in the section above about how the wording for this category may have inadvertently skewed data that should be considered in the same category as Export Controlled (i.e., unreleasable by statute).

In general, the lab is trying to move projects toward more "permissive" open-source licenses, which put few or no restrictions on how the software can be used. While this may seem counterintuitive, the overall goal of that policy is to lower the barriers to collaboration with others and capitalize on our expertise, not trying to restrict how others might use our software. Of course, there are exceptions where that policy may not be preferred.

Just over half of our open-source projects are released under what is generally considered a permissive license, with about half of those remaining released under LGPL, which while not considered permissive is intended to avoid some of the "viral" aspects of the traditional GPL based licenses. Almost a full ¼ of our projects are still released under GPL 2.0 or 3.0 licenses, which can pose barriers to adoption when the adoptees licenses are not compatible, or they are developing proprietary applications. However, GPL can sound attractive to developers for their language that offers protections against the software being used in ways potentially against our will (e.g., embedded in a commercial product).

That said, for many in the community, GPL (and by association LGPL) has become an indication to avoid using the software, and the lab is moving toward a policy of encouraging permissive licenses. The Industrial Partnerships Office (IPO) will help teams navigate this complex space during the software release process.

Looking for guidance on what open-source license to use? Or how to get your software approved for external release?

Check out https://dev.llnl.gov/opensource/ for a continuously growing set of resources and recommendations.

Key Takeaway #4 About half of the software developed at LLNL is released as open source, and just over half of that open-source software is released under a permissive open-source license.

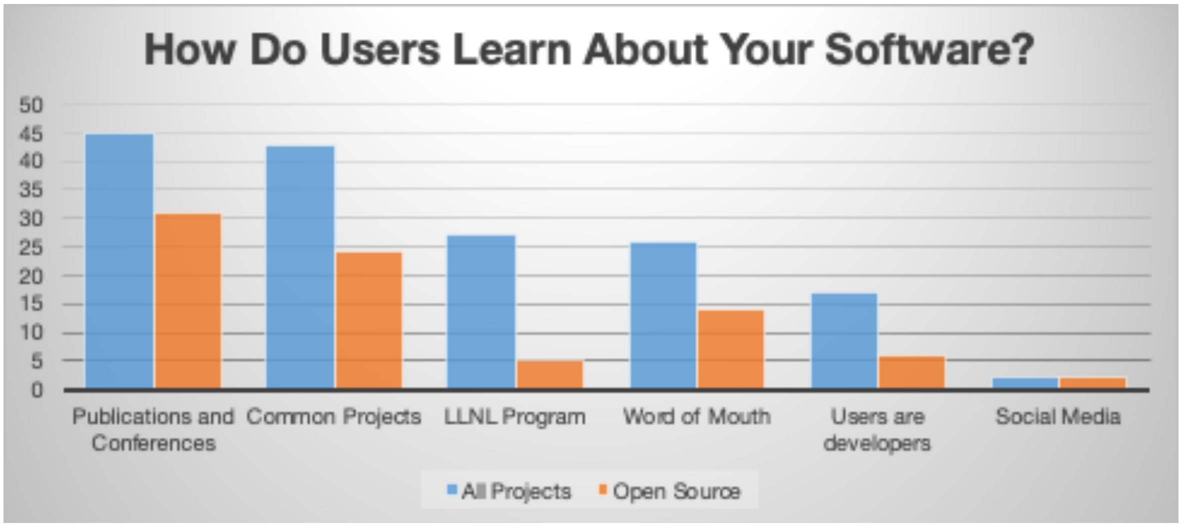
Question 8: How do users learn about your software?

Choices:

- □ My users are mostly the people who develop the software
- □ They're in my LLNL Program, and are expected to use it
- □ They work with us on common projects
- □ Open-source community - e.g., GitHub
- □ Publications, conferences
- □ Social media (e.g., twitter, LinkedIn)
- □ Word of mouth
- □ Other [ ]

Basis: Understanding how LLNL software is discovered provides important information for us to continue to grow our presence, and also shows us where we could be doing more outreach to build awareness.

Analysis: Not surprisingly for a science lab, conferences and publications are a top choice, followed closely by users who work together on common projects. We also narrowed this question down to those who answered that they support open-source software – and as anticipated the open-source community and social media were relatively higher while the other categories trended similarly to non-open source. The exception was the somewhat surprising drop in the users within LLNL Programs who are expected to use open-source software projects, which leads one to believe that LLNL Programs perhaps are not driving open-source development as a priority – and open-source developers are building their own funding sources and communities outside of LLNL Programs.

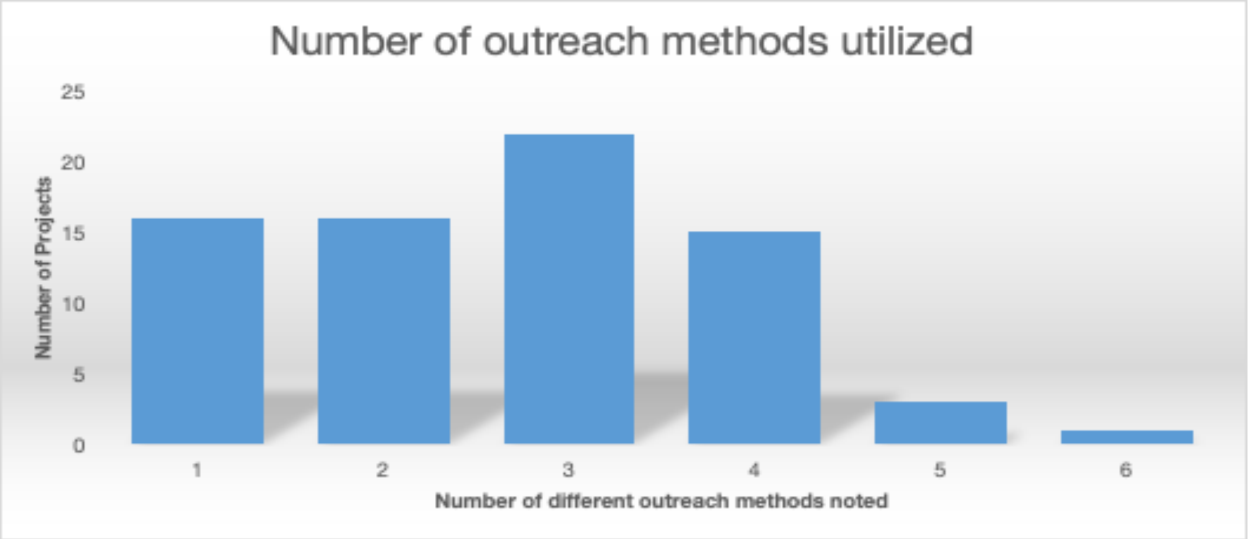


Other answers that were included in “Other” but are not represented in these charts because they were largely one-off include “Workshops”, “Schools”, “WFO (Work for Others)”, and “Inclusion in other codes”. A future survey might consider including some or all of these options as well.

Finally, since this question allowed users to select more than one answer, we looked at how many selections each project selected as a way to judge the diversity of methods typically used.

One of the major goals of the RADIUSS project is to help LLNL open-source teams reach a broader audience and community. Contact radiuss-request@llnl.gov if you’re interested in learning techniques to improve your GitHub presence, refine your documentation, and/or harden your software to encourage broader adoption.

And follow @LLNL_OpenSource on Twitter, there may be more people learning about your software from there than you realize!



Key Takeaway #5 Publications and Conferences are the primary means of building awareness of our open-source software.

Question 9: What are your major funding sources for HPC compute CYCLES?

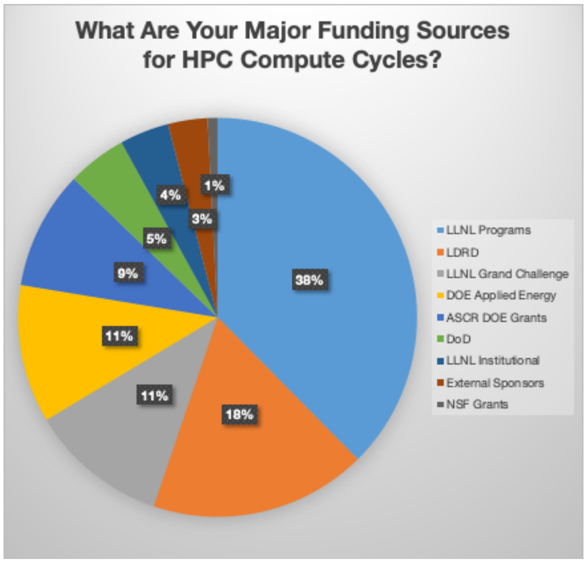
Choices:

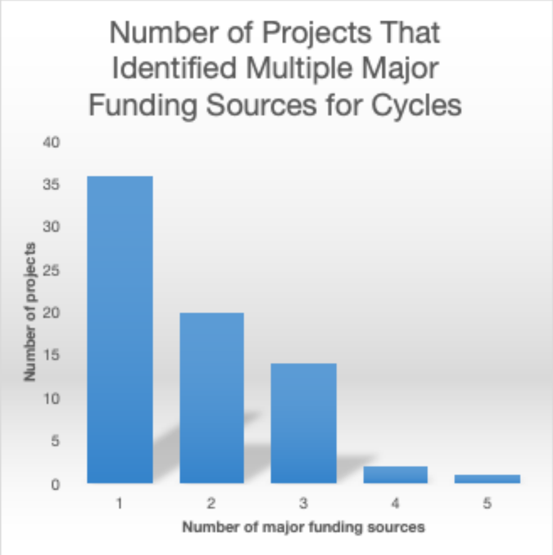
- □ LLNL Programs (e.g., ASC, GS, NIF)
- □ LLNL Grand Challenge Awards
- □ LLNL LDRD
- □ Other LLNL institutional
- □ ASCR Grants at DOE User Facilities (e.g., INCITE, ALCC)
- □ DOE Applied Energy Offices (FE, NE, OE, EREE, ...)
- □ DoD
- □ NSF Grants
- □ External sponsors
- □ Other [ ]

Basis: HPC applications require big and expensive computers to run on, and there are a variety of methods available to lab staff, including our own Livermore Computing, other DOE sites, other agencies, and other sponsor resources. Projects often have multiple means at their disposal which this question aims to uncover.

Analysis: A full ¾ of projects stated that they rely on LLNL-provided funding as a major source of compute cycles – including Programmatic, LDRD, Grand Challenge, and other Institutional. The next largest amount was listed as the DOE Applied Energy Offices, which most likely are run on the NERSC systems at Berkeley Lab, although that wasn't made clear in how the question was asked. Combining the DOE Applied Energy with the ASCR DOE Grants which run on the Leadership Computing Facilities at Oak Ridge and Argonne, a full 20% of our applications take advantage of those other open science DOE HPC resources.

Finally, we looked at how many different sources projects specified as a measure of how many different funding sources each team identified. Of the teams that identified only a single source for cycles, 70% of those identified LLNL Programmatic funding as their sole source of HPC Cycles.





Section 2 Platforms and Cloud Computing

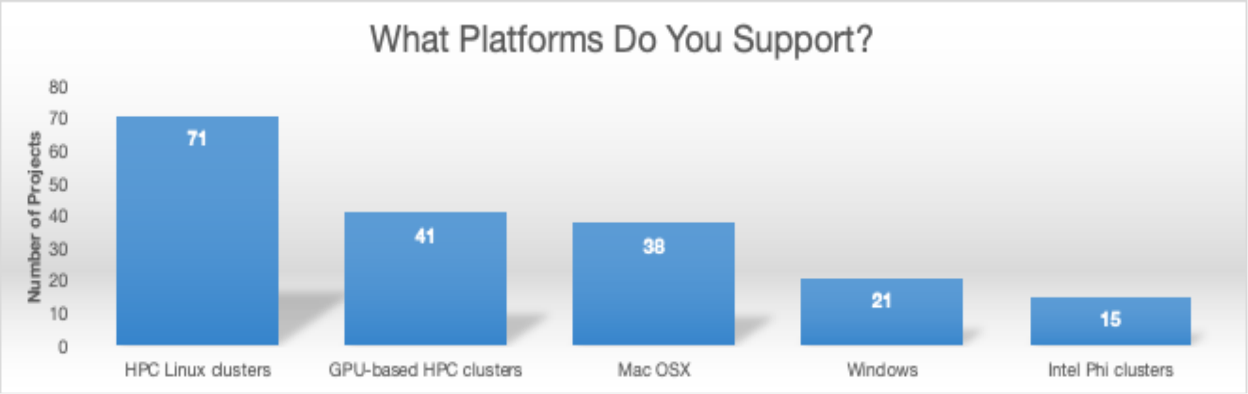
Question 10: What platforms do you support?

Choices:

- □ HPC Linux clusters
- □ GPU-based HPC clusters
- □ Intel Phi (KNL) based clusters (e.g., Trinity, Cori, Theta)
- □ Mac OSX
- □ Windows
- □ Other

Basis: We speculated that HPC applications are generally quite portable across Linux CPU-clusters which make up the bulk of HPC center offerings. This question was aimed at understanding how many applications are supported on GPU-based systems (e.g., Lassen, Sierra, Summit), other DOE systems based on the now defunct Intel Phi, and desktop systems such as MacOS and Windows.

Analysis:



It was encouraging to see how many applications support GPU-based HPC clusters, as this has been an important part of the ASC and LC strategy. Mac OSX support outpaces Windows support on laptop/desktop systems by almost a 2:1 ratio, likely due to the underlying Unix OS and available compilers.

Key Takeaway #6 Over half of the projects represented are ported to GPU-based HPC clusters. Conversely, almost half do not yet support those systems.

Question 11: Have you ever used cloud-based resources (e.g. AWS, Azure) for testing, building, or other development?

Choices:

- ○ Yes, it's now part of our development cycle
- ○ Yes, we tried it but don't use it regularly
- ○ No

Analysis combined with the next 3 questions

Question 12: Have you ever used cloud-based HPC resources for large-scale, multi-node simulations?

Analysis combined with the next 2 questions

Choices:

- ○ Yes
- ○ No

Question 13: Do you anticipate using HPC cloud resources in the future, or are you interested in exploring the option? (Note: assume unclassified but restricted codes are safe to run in GovCloud or similar)

Analysis combined with the next question

Choices:

- ○ Yes
- ○ No
- ○ Maybe

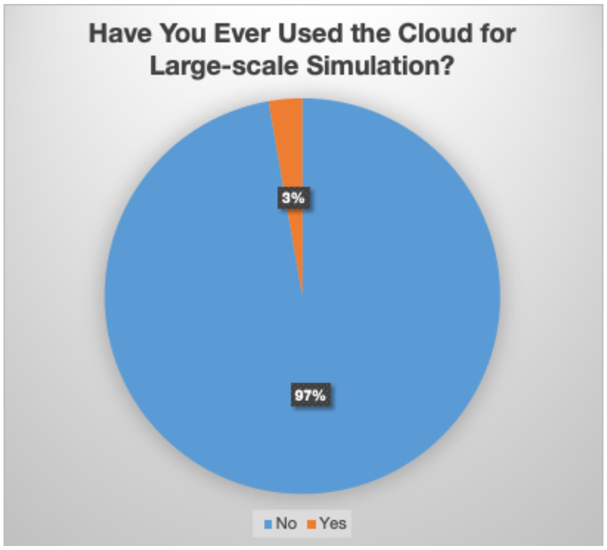
Question 14: How do you think you'd use the cloud?

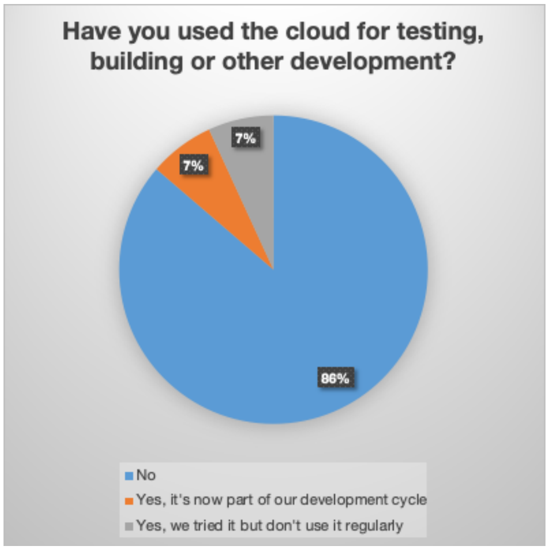
Choices:

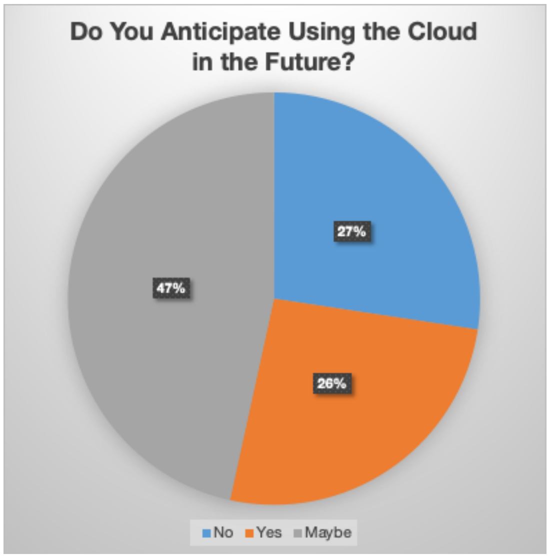
- □ To get HPC cycles for large calculations I can't get through DOE or NSF HPC systems
- □ To test my application on platforms I don't otherwise have access to
- □ To build executables for users who run on different OS's
- □ To use software or services not allowed on my current HPC platforms
- □ To let users try my software in a controlled environment

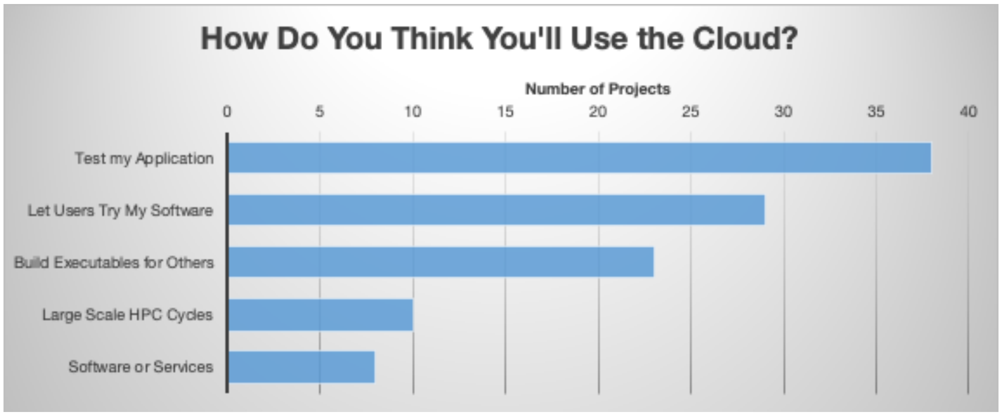
Basis: This series of questions was aimed at getting an understanding of one of the potential disrupters for HPC that has already had huge impact in computing outside of the HPC world. Cloud resources are becoming increasingly popular, particularly for teams looking for resources to build and test using virtual machines, containers, and continuous integration. But while the cloud is getting traction for basic testing, there are also cloud resources that could potentially provide teams with great numbers of cycles in lieu of using time on large government HPC systems. The first two questions were aimed at understanding current usage.

Analysis: Not too surprisingly, most applications have not yet adopted the cloud for large-scale HPC simulations. Whether this is because the payment model is inconvenient for government researchers, they are just not aware of it, or feel they have sufficient resources available to them – this would be a good follow-up question for a future survey.









Key Takeaway #7 Public cloud is not yet standard practice, but there is growing interest in using the cloud for testing and streamlining collaborations with external users.

Section 3 Development Infrastructure

Question 15: What type of version control system do you use?

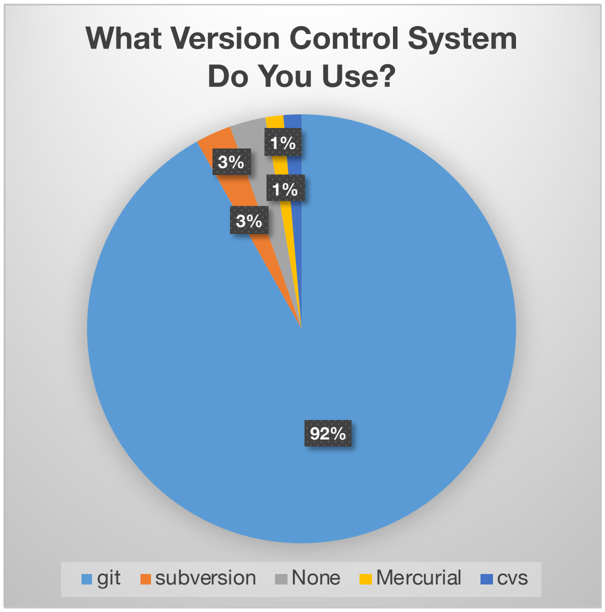
Choices:

- □ git
- □ subversion
- □ None - we don't use/need version control
- □ Other (specify)

Basis: Understanding the extent to which projects are converging on common solutions is important, as it makes it easier to share process improvements, and for developers to quickly come up to speed on developing within a project.

Analysis: As expected, git has come to completely dominate the version control system space. With its excellent features for distributed development, branching, and hosting sites such as GitLab which integrate addition features such as pull requests (code reviews) and continuous integration, anyone not yet using git is in the deep minority at LLNL.

This is not surprising, as GitHub, GitLab, and Bitbucket all default to (or require) git.



Key Takeaway #8 Git has become the de facto standard repository management tool

Question 16: If you have a regression test suite, what percentage of your application does it cover? (by lines of code, functions, features... give us your best guess). Include all of your test suites if you have multiple.

Choices:

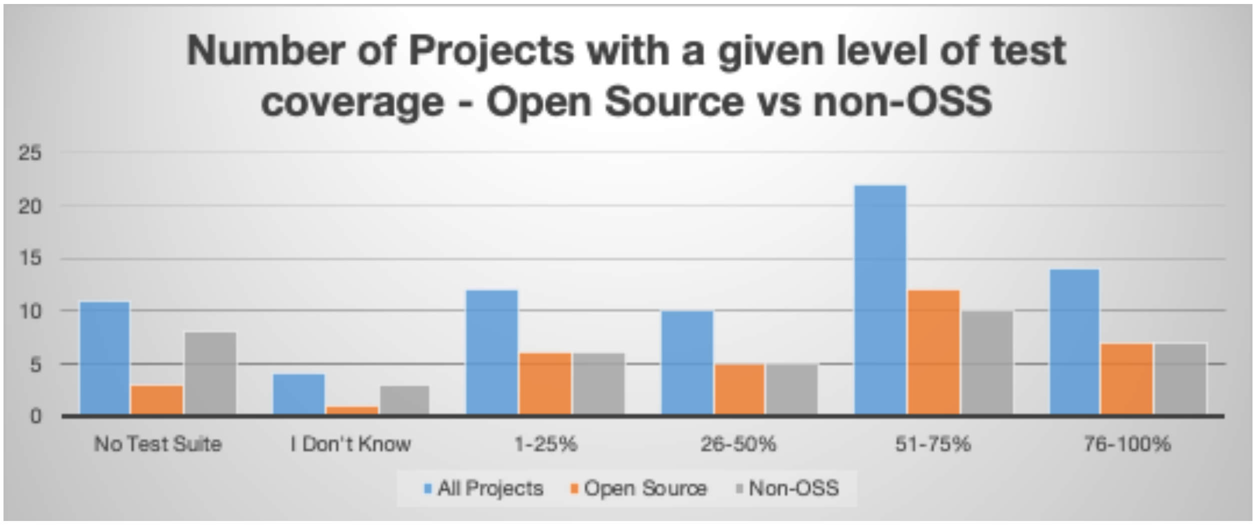
- ○ We don't have a test suite
- ○ I don't know
- ○ 1-25%
- ○ 26-50%
- ○ 51-75%
- ○ 76-100%

Basis: Testing, and specifically regression testing, are an important tool for teams to track and reduce bugs. A regression test suite forms the basis for more integrating testing techniques such as continuous integration.

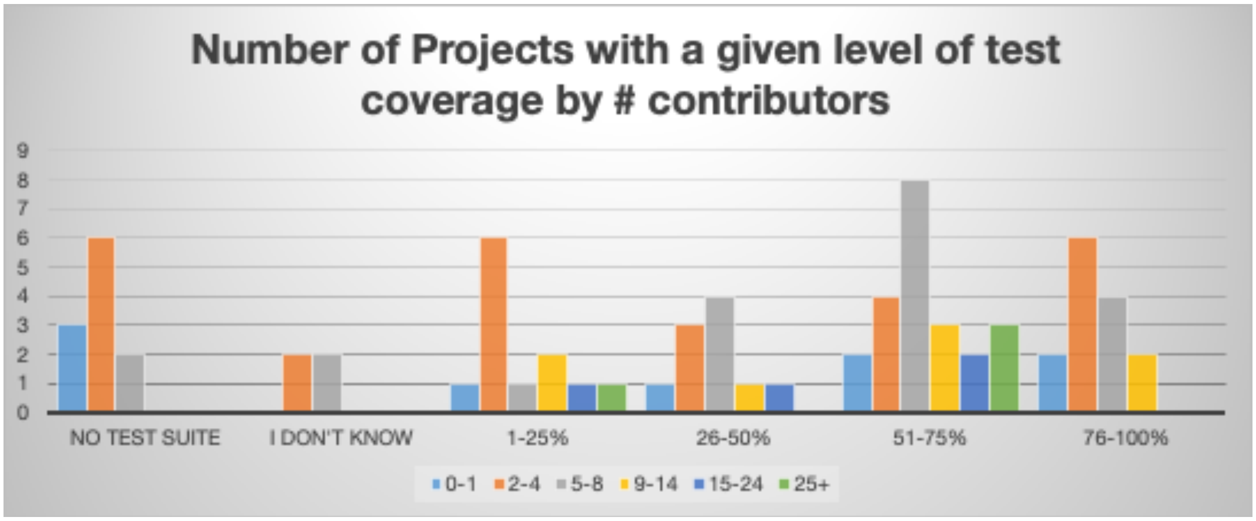
Analysis:

When data was analyzed across all projects, it was hard to draw conclusion with a full 80% of software projects have some sort of regression test suite, and a majority of those having good test coverage of 50% or greater. Projects with 75%+ coverage are doing an excellent job of testing, as that metric is hard to reach, especially as software gets large and complex. But there were likewise 20% of projects with no test suite or relatively poor coverage metrics.

To find out if there was trend, we grouped the data first by whether it was an open-source project or not which is shown in the first chart below. Other than relatively fewer open-source projects reporting no test suite, the coverage metrics were virtually identical between Open Source Software (OSS) and non-OSS.

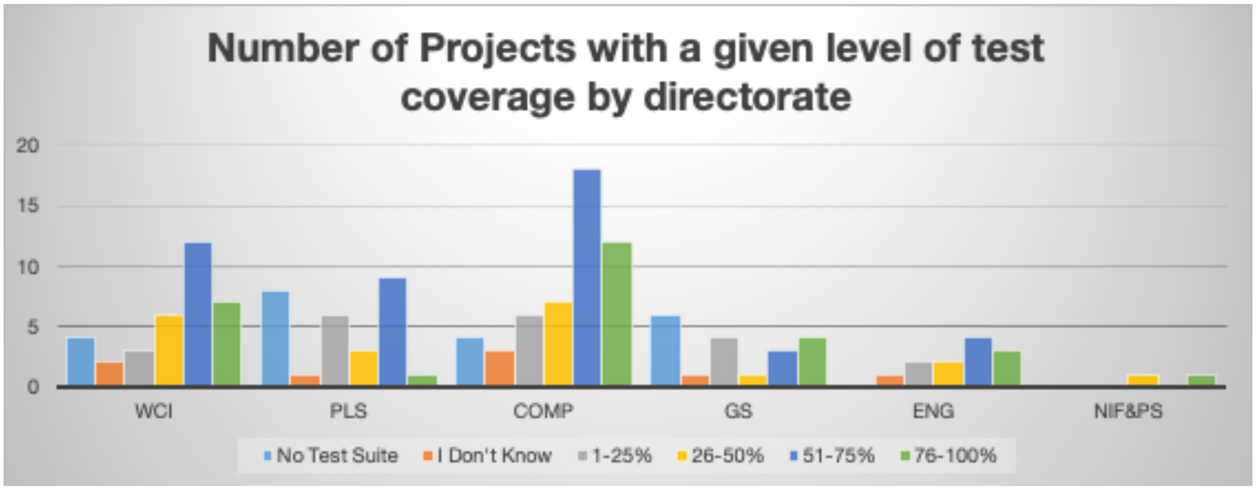


We then looked at the data as function of the team size (# contributors) from Question 2:



Again, no clear correlation emerged, although it is interesting to note that smaller projects (< 8 contributors) were the only ones who indicated they don't have a test suite or didn't know. And likewise, the larger projects (> 15 contributors) were not able to reach the highest coverage percentages – perhaps because they are on large codes where it gets difficult to achieve those higher coverage numbers.

Finally, we tried slicing the data by directorate. WCI, Computing, and Engineering had some of the better metrics, but no obvious correlations jump out.



Projects with a lower percentage of test coverage should consider setting goals to increase coverage with every release. Projects with no test suite should seriously consider adding one, as testing is a fundamental practice of good software development that can prevent bugs from be introduced or reintroduced. There are a number of excellent testing frameworks available.

Key Takeaway #9 Regression test coverage ranges from none to excellent – with a majority of projects providing good test coverage of 50% or greater

Question 17: Does your project use continuous integration to test your software?

Choices:

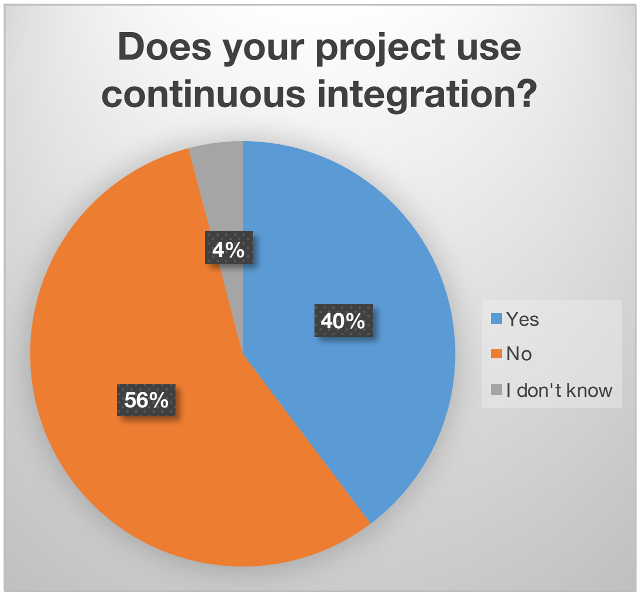
- □ Yes
- □ No
- □ I don't know

Basis: Continuous Integration (CI) is the simple concept of testing your software regularly and automatically, generally before changes are merged into a main development branch. There are tools built into GitHub to support CI, and LC is in the process of replacing the existing Bamboo CI with GitLab CI due to its better interface and HPC-friendly features that have recently been added.

*Livermore Computing is providing **GitLab CI** as a standard HPC-friendly tool for continuous integration. The RADIUSS project is working to provide software teams with templates to make it as quick and painless as possible to begin using continuous integration in LC. See the Section 10 of this document and https://dev.llnl.gov/opensourrce/radiuss for developer guides and CI templates.*

Analysis:
While these numbers are encouraging, over half of the projects reporting still do not use continuous integration. There is ample opportunity for projects to adopt new CI technologies available in the cloud (e.g., at GitHub) and now using GitLab CI within the LC. RADIUSS has been defining templates for teams to adopt and use GitLab CI and can help projects get started if they have a basic test plan set up.

Once a set of tests have been defined that can be easily run and a pass/fail result determined, setup is relatively straight-forward and has big benefits, especially for projects which are accepting a lot of changes.



Key Takeaway #10 Use of Continuous Integration is still not adopted by over half of the projects

Question 18: Does your project have user documentation?

Choices:

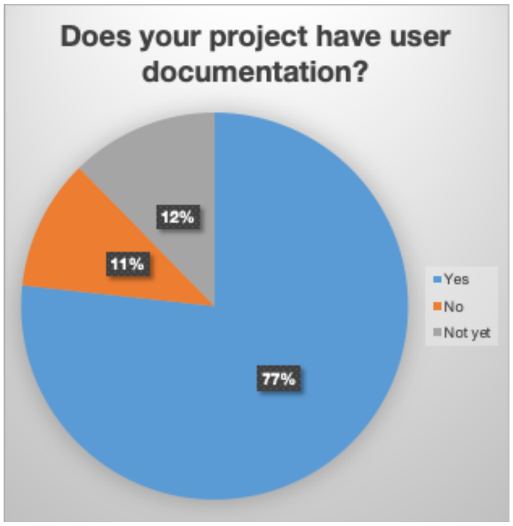
- □ Yes
- □ No
- □ Not yet, but we plan to

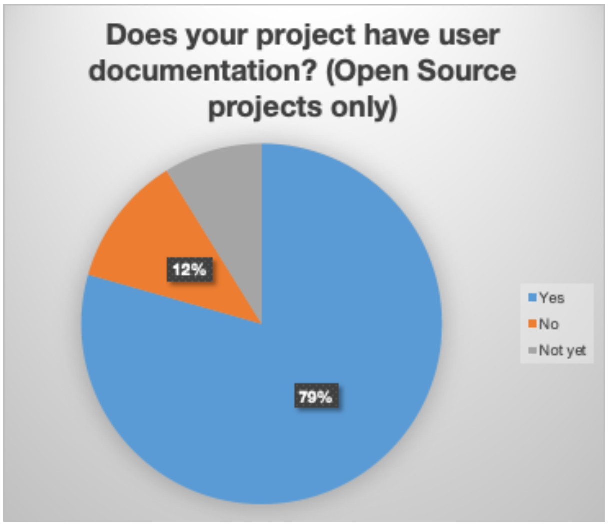
Basis: User documentation is intended to instruct users of the software in the operation and can range anywhere from a few pages to hundreds of pages.

Analysis: Over ¾ of the project's indicated that they have user documentation, with another 12% indicating that don't have it yet, but are planning on developing it. The breakdown was almost unchanged when counting only projects that identified as open source.

The lack of any user documentation can be an indication that the users are also the developers who have intimate knowledge of how it works. It could also be an indication that the software is in the very early stages of development, although best practices are to develop user documentation alongside development of source code.

Regardless, documentation is absolutely necessary if your project is trying to attract new users.



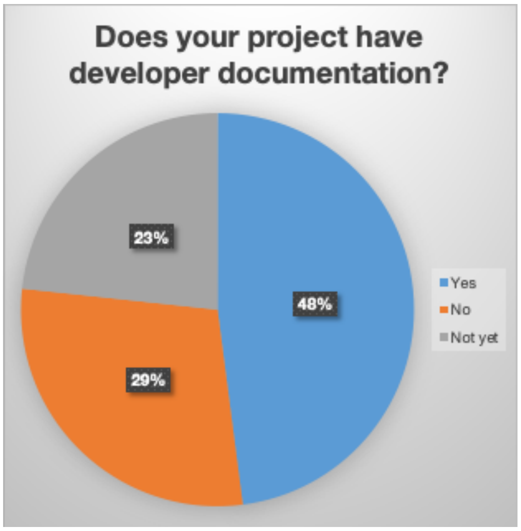


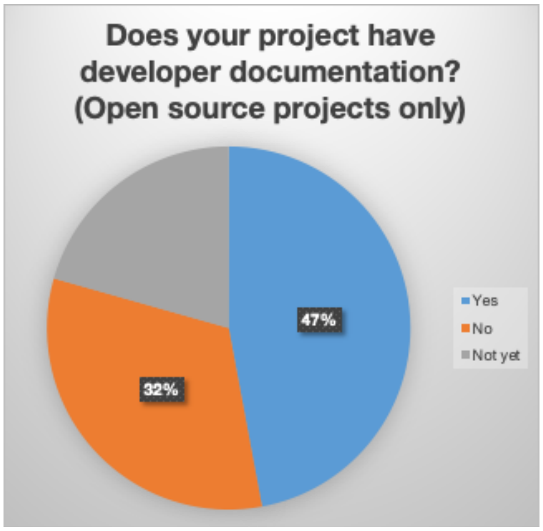
Question 19: Does your project have developer documentation?

Choices:

- □ Yes
- □ No
- □ Not yet, but we plan to

Basis: Developer documentation is intended to instruct other developers on the design, structure, APIs, and any other concerns specific to development of the software. While many smaller projects can probably get away with developers learning by examining the code and adopting the established style and learning other processes through direct mentoring, developer documentation can help a new project member become productive more quickly, as well as establish a common set of practices that leadership can point to keep the code and processes from drifting into inconsistent approaches.





Analysis: Less than half of projects have developer documentation, although almost ¼ of the respondents indicated that they plan to develop some.

Similar to the previous question about user documentation, when we sliced the data by open source projects the breakdown was almost identical.

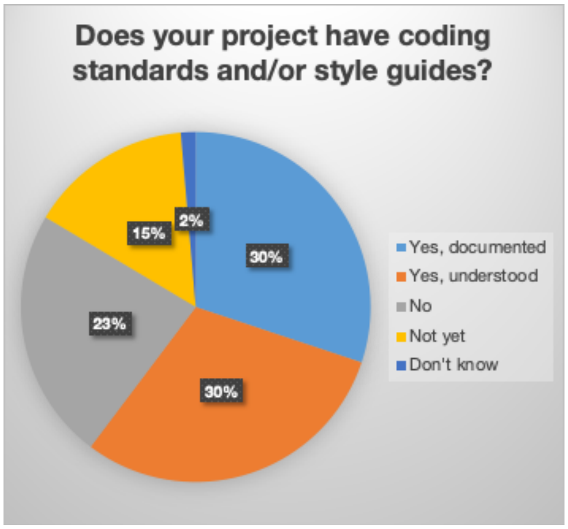
Key Takeaway #11 Whether or not a project is open source has little impact on whether or not they develop user and developer documentation

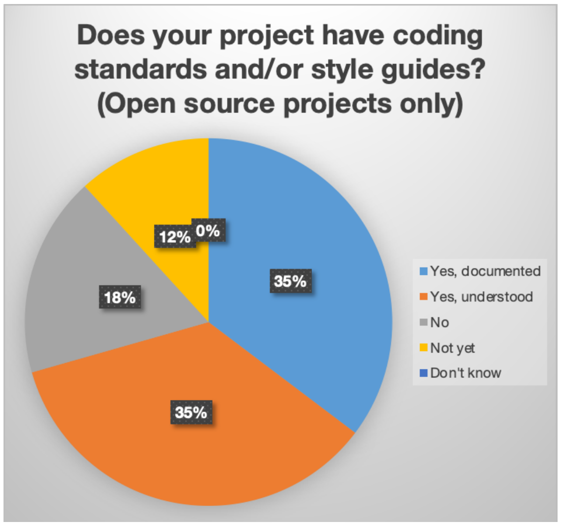
Question 20: Does your project have coding standards and/or style guides?

Choices:

- □ Yes, and they are documented and accessible to all team members
- □ Yes, they are not documented, but enforced (e.g., through code review, prettify tools, etc...)
- □ No
- □ Not yet, but we plan to
- □ I don't know

Basis: Coding standards and style guides are another form of developer documentation aimed at enforcing both how things should be implemented, which features of a language should be used, and a consistent look-and-feel for how the source code should be formatted. This is particularly important with open-source projects where teams are naturally distributed, and non-core developers may be contributing to the code base.





Analysis: Increasingly, formatting standards can be automatically enforced with tools such as *clang-tidy*, which can be automatically run over code before its committed to a shared developer branch in the repository. However, it is still important to document other aspects of coding standards. In particular, teams may choose to hold back on adopting the newest features in updated language standards until such a time where they are more universally adopted and understood.

Interestingly, of this series of questions about documentation – this was the only question that had some signficiant breakdown differences when calling out just open source projects. As can be seen in the pie charts above, 10% more of those open source projects have documented or understood coding standards.

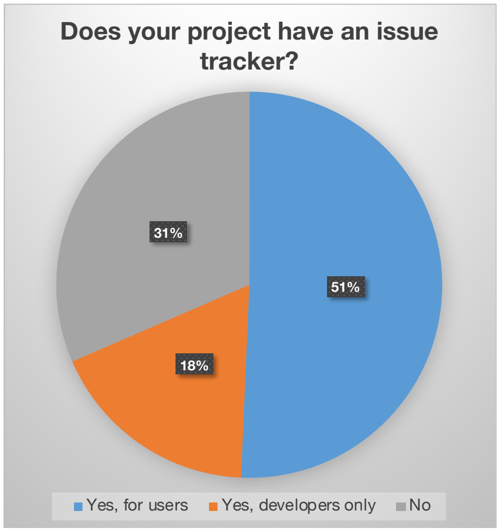
This is likely attributable to the need for such standards when projects are encouranging contributions or creating community health files (e.g. CONTRIBUTING) in their GitHub repository.

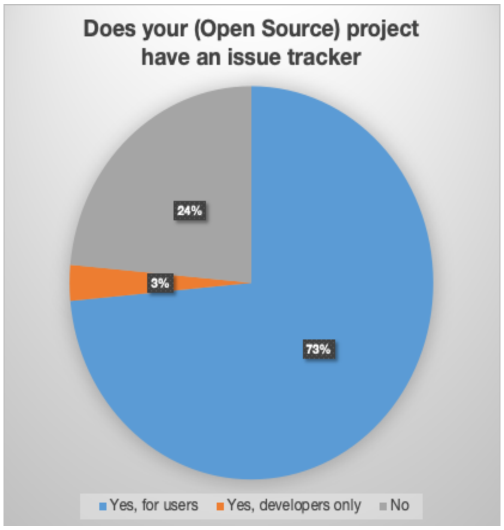
Question 21: Does your project have an issue tracker?

Choices:

- □ Yes, and any user can submit and track an issue
- □ Yes, developers can submit issues on behalf of users
- □ No

Basis: Issue trackers are an important tool for tracking tasks that must be performed, and documenting if and how they were resolved. Some teams allow any user to submit issues and/or add their comments or votes to existing issues to help raise their priority with the developers. This model is common on GitHub, which integrates a basic issue tracker with each project. Other teams choose to retain more control over who can submit issues and will submit them on users' behalf after receiving communication other ways (usually email or discussion).





Looking for an issue tracker to use? For projects hosted in LC, Atlassian's Jira tool is a common solution. GitLab also has a basic issue tracker, and will likely find increased usage within LLNL once GitLab is hosting more repositories locally. GitHub also supports a basic issue tracker for projects hosted there.

Analysis: Just over half of the projects had an issue tracker accessible to users, with another 18% maintaining an issue tracker, but not providing direct access to their users.

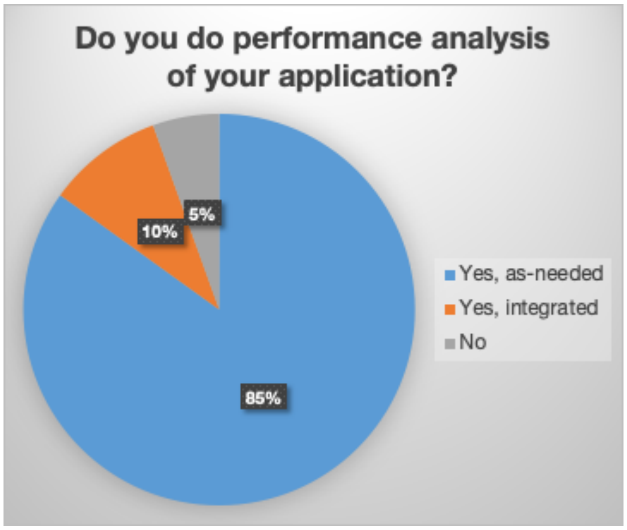
More concerning, almost 1/3 of the projects did not use an issue tracker. For very small teams, this can be acceptable as the number of tasks and/or bugs that must be tracked can be managed and/or centralized using simple techniques. But some form of issue tracker is recommended for all projects, as it provides a trail of documentation for developers who follow, as well as a list of tasks that new developers or team members can begin to tackle when introduced to the project.

Key Takeaway #12 Nearly 1/3 of all projects and 1/4 of open source projects are not utilizing an issue tracker

Question 22: Do you do performance analysis of your application?

Choices:

- □ Yes, it's integrated into our testing process
- □ Yes, on an as-needed basis, or ad-hoc activity
- □ No



Basis: For anyone doing development in the field of High-Performance Computing, performance analysis is critical, at least if your program is going to be consuming HPC resources. There are a number of tools available to assist with performance analysis, and generally developers pick a favorite and used it to identify code performance bottlenecks on occasion.

Analysis: As one would hope, a full 95% of our respondents indicated that they do performance analysis. (A subsequent survey on the preferred tools would be of interest). But only 10% have indicated thus far that it is integrated into their regular testing regimen. Given that it was probably closer to 0% just 5-10 years ago, this is an improvement – but we should be striving for a much higher percentage using this technique. Our HPC systems are too valuable to allow untuned applications to use significant cycles.

LLNL is a pioneer in enabling ubiquitous performance analysis, where lightweight tools (e.g., Caliper) can be integrated as a library into your application, and tools tied into your standard testing (e.g., Spot) can track performance over time at granularities that are neither too course (e.g., the wall clock time of the entire calculation) or too fine (e.g., every function point). This model of ubiquitous performance analysis allows performance changes to be identified quickly, as well as monitor progress across a range of test problems as performance optimizations are performed.

Key Takeaway #13 While almost all (95%) of projects do performance analysis, only 10% are doing it as an integrated and regular part of their testing cycle

Section 4 Languages and Programming Models

Question 23: WHAT PROGRAMMING LANGUAGES DOES YOUR PROJECT CURRENTLY USE? (DON'T INCLUDE 3RD PARTY LIBRARIES - ONLY WHAT YOU DEVELOP ON YOUR TEAM)

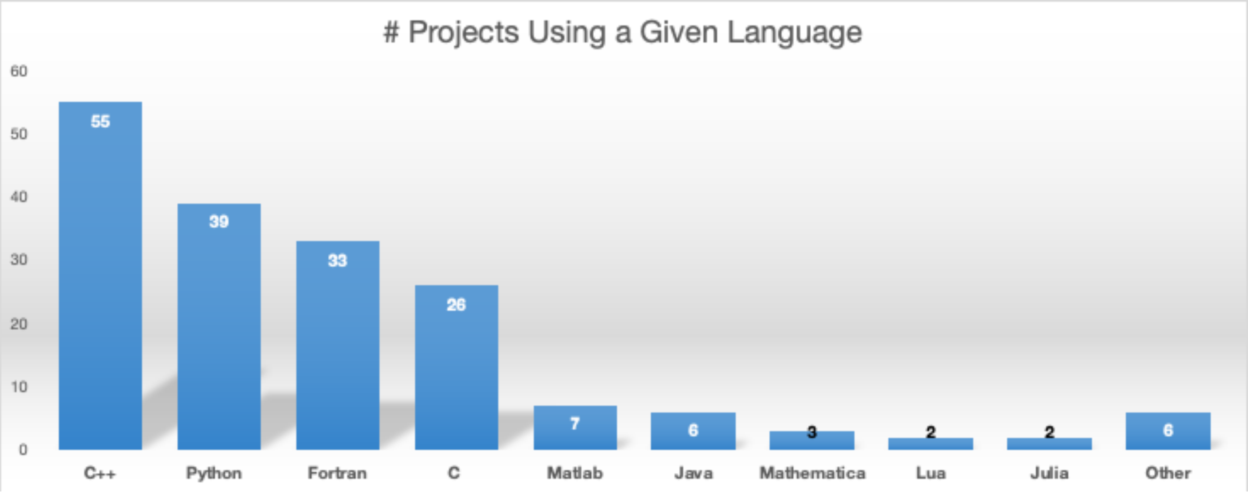
Choices:

- □ C++
- □ C
- □ Fortran
- □ Python 2
- □ Python 3
- □ Lua
- □ Java
- □ Julia
- □ MatLab
- □ Mathematica

Basis: The choice of languages in HPC is interesting to note, as there has been a slow but steady shift from Fortran to C to C++ over the years, with Python also firmly establishing itself in scientific computing for non-performance critical sections of code.

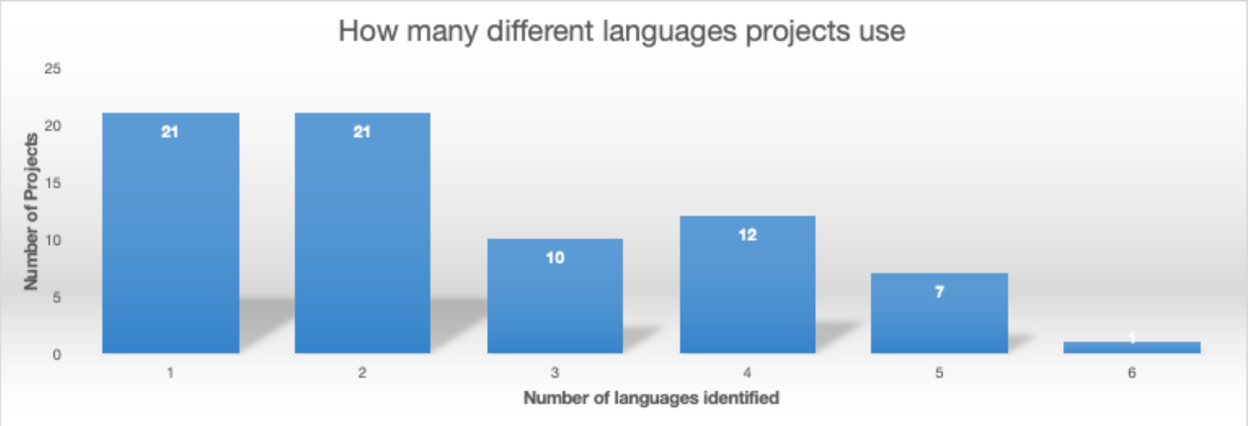
Analysis: Ask just about anyone in scientific HPC computing what the dominant languages in use are, and you'll probably get an answer that looks very similar to LLNL, perhaps with the order changed (many science labs still claim Fortran as the dominant scientific language). But in the last 10-15 years, C++ has become the lingua franca of HPC computing, and this data verifies that this is indeed the case at LLNL.

Note that for this data, we combined Python 2 and Python 3 into a single category. A further breakdown of Python usage is included in a chart at the end of this section.



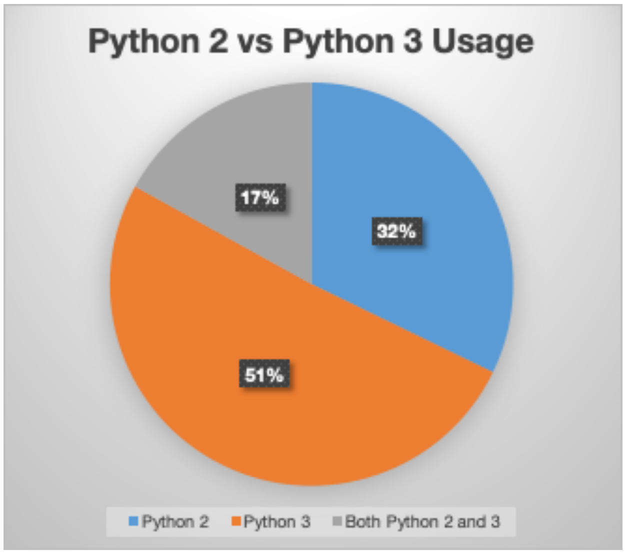
We used the same survey data to get an idea of how many different languages applications tended to use, since the survey allowed them to pick all that apply. The data presented is in histogram style and shows that indeed mixed-language application development is indeed common.

Note that the data may be skewed a bit on the high side because 10 projects mentioned both Python 2 and Python 3 as languages.



Since we called out Python 2 and Python 3 as different language options (although they were reported as a single language in the first chart), it's worth highlighting that data. Python 2 is officially deprecated as an open-source project and will no longer receive any updates. While major security holes will likely be patched by the community if found, there is high incentive for projects to convert to Python 3. The difficulty in doing that ranges widely between projects.

Languages noted as "Other" included: Yorick, TCL, Perl, Groovy, and IDL.



Key Takeaway #14 C++ and Python dominate as HPC languages, but Fortran and C continue to make a strong showing. Many projects are multi-language.

Question 24: WHAT PARALLEL PROGRAMMING MODELS DOES YOUR APPLICATION OR LIBRARY CURRENTLY USE?

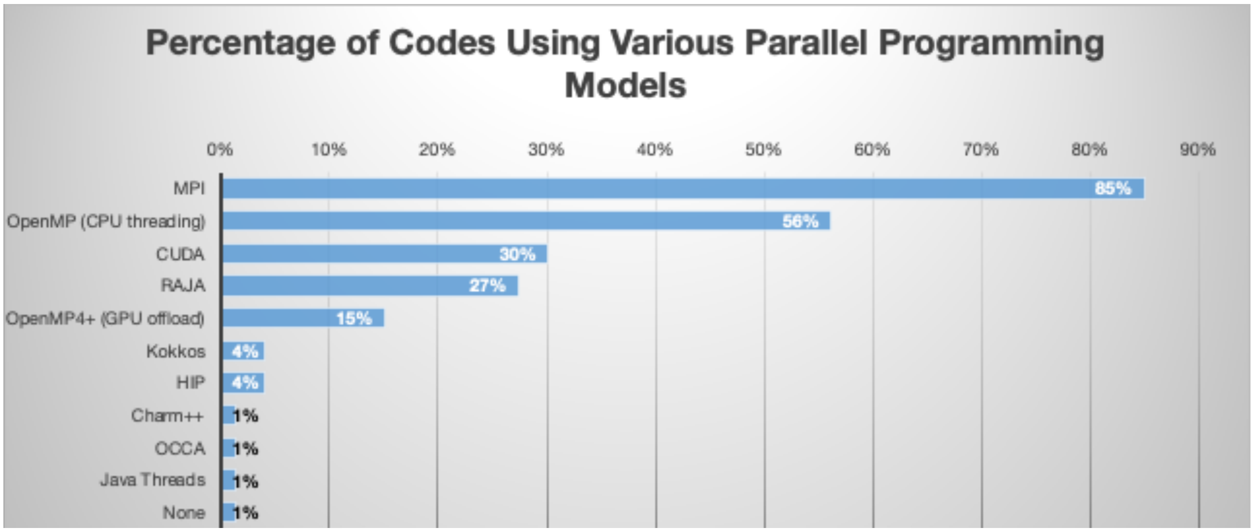
Choices:

□ MPI
□ OpenMP (CPU threading)
□ OpenMP4+ (GPU offload)
□ RAJA
□ Kokkos
□ CUDA (directly in source, not behind an abstraction layer)
□ HIP (directly in source, not behind an abstraction layer)

□ None (e.g., my app is single core, but I run large ensembles)

Basis: Parallel programming models have become increasingly important as on-node parallelism is exploding – especially with the introduction of GPGPUs.

Analysis: While MPI is still pretty much the de facto standard for HPC simulation, on-node models such as OpenMP, RAJA, Kokkos, CUDA, and HIP are emerging as necessary to exploit modern architectures. RAJA is a popular choice for performance-portability at LLNL as it is primarily developed here. We would expect HIP to slowly eat into the CUDA numbers as the deployment of El Capitan approaches, since HIP will pass-through directly to CUDA on NVIDIA GPUs while also supporting AMD GPUs natively.



Question 25: If you are thinking of making any changes to your programming language and parallel programming model choices in the future, please describe them here (E.G., "Rewriting Fortran to C++", or "Adopting RAJA for GPU offload")

Choices:
(Free form response)

Basis: This question was the first of our open-ended questions and was aimed at getting some insight into how projects are approaching languages and programming model choices outlined in the prior several questions.

Analysis:
The answers provided are presented below with light editing for syntax or punctuation errors. The names of the codes or respondents is not provided but can be upon request and with permission of the respondent.

While answers varied, many were aimed at preparing for GPUs. It was heartening to see many of the open-source tools developed in ASC and promoted under RADIUSS mentioned as well.

Common answers included:

- Rewriting Fortran to C++
- Targeting GPUs with performance-portable abstraction layers or directives
- Updating Python2 to Python3

I may be forced to convert to GPU's if there are no CPU's available in the future on LC
Rewriting Fortran to C++ and/or OpenMP threading
Porting aspects of our model to use HIP.
Rewriting Fortran to an object-oriented language (e.g., C++)
If RAJA was available via something like LLVM I would be interested in looking at integrating that using GraalVM. Realistically though, if we start using GPUs it will be using a project already in the JVM ecosystem like Aparapi or JOCL.
We plan to begin a port of our steering language from Yorick to Python 3 later this year.
Beginning to add support for accelerators. University contributors are exploring directly using CUDA. Desperately trying to promote use of an abstraction layer (Kokkos/RAJA). Moving from C to C++.
Current FY20 effort to rewrite using RAJA. Almost finished Another code under this project is implemented in Fortran. Is fairly lightweight and could be rewritten as mini-app in C++ using RAJA.
* Implementing OpenMP4+ for CPU-intensive routines (on-going) * Using cuBLAS for linear algebra (on-going) * Possibly rewriting CPU-intensive routines in C++ (pending support)
Some GPU abstraction layer (Kokkos, RAJA, OpenMP4+, etc.)
Adding Kokkos-based structures, adding support for HIP and DPC++ either through direct use or through abstraction layers.
Want to update for GPUs. Not sure how we will do this.
Rewriting Fortran to C++, adopting HIP for GPU offload. Considering using Umpire.
Developing new core engine using Charm++ for parallel execution.
Using TensorFlow for GPU calculation of selected kernels
For a vectorized rewrite of a subset of the library, we are looking into use of RAJA
We'll be updating to Python 3 this year.
GPU deployment, most likely through OpenMP or RAJA.
No changes at this time.
Legacy code. Migrating capabilities to new framework.
Adopting RAJA for GPU offload
Adopting RAJA for GPU offload
Adopting RAJA for GPU offload
GPU offload
We are exploring performance portable solutions, including Kokkos and DPC++.
May consider rewriting some loops for new architecture computers
The <name deleted> project is focused on replacing Fortran in the atmosphere model with C++/Kokkos code. Ocean, ice, etc seem to be moving to CUDA.
Planning to use the Chombo Proto GPU framework.
The communication layer is modular, so can be developed to include other medium, which we are currently looking into.
Replace CUDA with HIP. Eventual usage of RAJA. More earnest use of C++11.
Porting to GPU, following the lead of <developer>, who is working on GPU strategies for <code>
using python to post-process data
Integrate with Kokkos in LAMMPS
Moving to OpenMP4+ when Fortran support is sufficiently capable
Adopting RAJA for GPU offload
Adopting RAJA for GPU offload.
Potentially adopting RAJA if/when it makes sense for us.
We are porting zfp to HIP and will explore OpenMP 5. We may partner with RAJA/Umpire and/or Kokkos to support zfp via their APIs, but we do not have plans to rewrite zfp based on those programming models.

Aiming at GPU support for most computationally intensive parts of the code.
Incorporation of GPU support.
We are actively integrating more RAJA
We are trying to find free energy on our project to integrate Umpire, which will be essential for GPU memory sharing with our host codes Help with Spot might be nice to get integrated performance tracking Will be transitioning this physics to the <code name withheld> code base, which will be C++ with RAJA. This transition will take a long time.
We plan on moving more of our Fortran to C++. Also, we plan on being early adopters of the asynchronous/stream features being developed in RAJA/CHAI/Umpire
Exploring how to support a more diverse set of accelerator systems
Investigating CHAI
Migrating to Python 3 from Python 2
We have recently rewritten Fortran codes to C++ and are implementing RAJA for GPU-based computing.

Section 5 Dependencies

Question 26: About how many 3rd party dependencies does your application have? Do not include those provided by the vendor OS (e.g. libm)

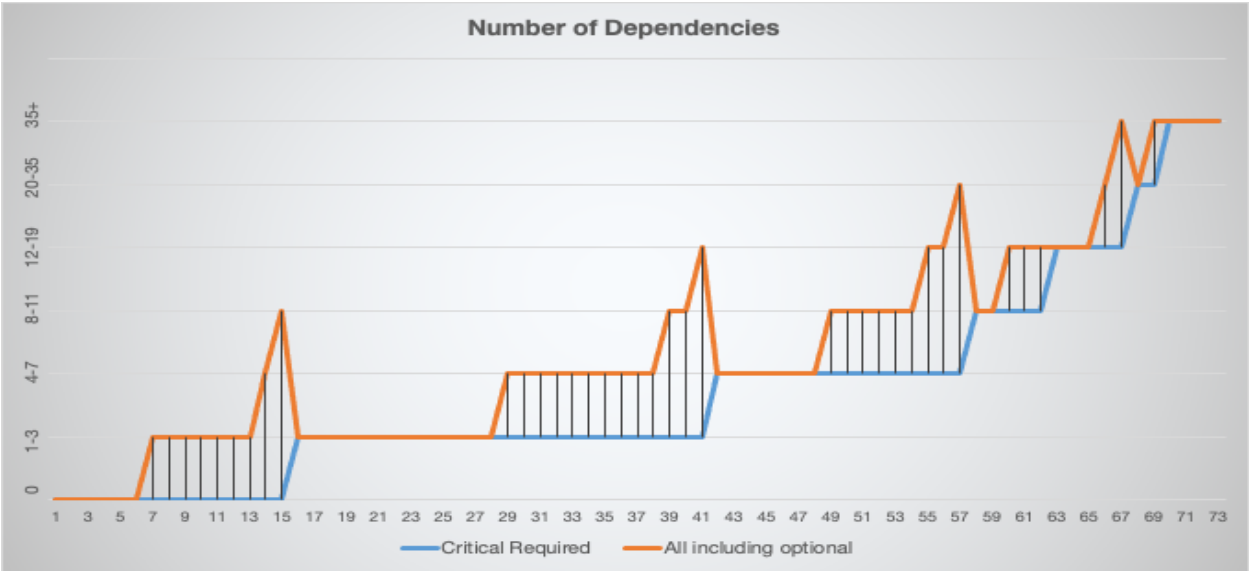
Choices:

	0	1-3	4-7	8-11	12-19	20-35	35+
Critical to core functionality - part of our base build	○	○	○	○	○	○	○
All libraries - including those that are optional for occasional features	○	○	○	○	○	○	○

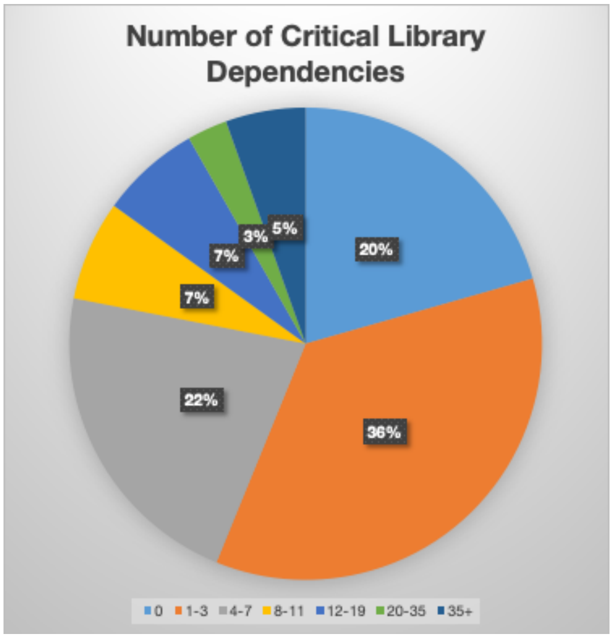
Basis: As software grows more sophisticated, it relies on 3rd party libraries which must be built and/or provided as part of the overall functionality or which can provide optional functionality. This question aimed at getting a broad understanding of how many 3rd party libraries teams were incorporating and managing as part of their build process.

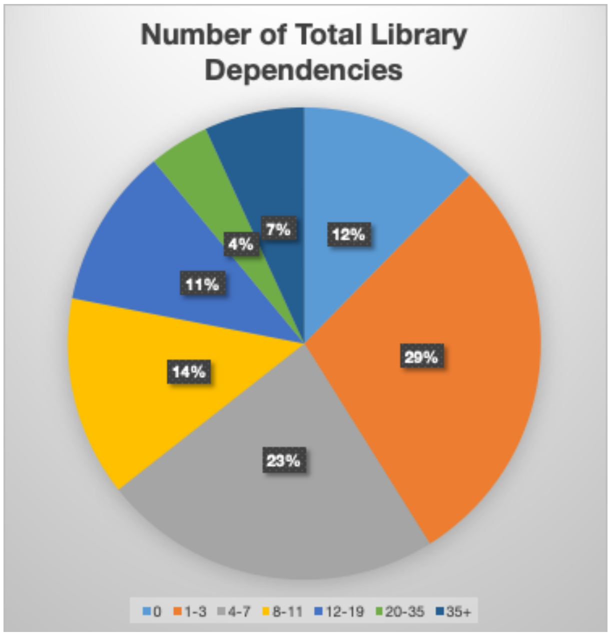
Analysis: The chart below sorts the responses such that each project is represented by one tick on the x-axis. At each point, the blue line represents the number of critical dependencies by range, and the orange line represents the total number of dependencies. A large gap between the blue and orange line indicates a high percentage of optional libraries for that project.

Note: The question was apparently misunderstood by some responders as there were some projects (~10%) that listed the total number of libraries (including optional) as less than the number of required/critical libraries – which is by definition not possible since one includes the other. Instead of trying to guess their meaning (i.e., maybe they thought the optional question was "in addition to…") we simply set the values equal, otherwise the graph was difficult to visualize



An alternate view of the data is available in the charts below





The data shows that 15 of the 74 (~20%) projects can be built with no external dependencies. Most projects had a small number of dependencies (see next question for the most common), and at the other extreme about 15% of the projects had a dozen or more critical dependencies.

While some projects apparently try hard to reduce the number of library dependencies they have (presumably for simplicity in porting), many projects are aggressively pursuing library use, with over half of the projects claiming more than 8 total (including optional) dependencies. This is likely partially due to the vigorous pursuit of highly modular development in the ASC program at LLNL, primarily through the applications supported in Weapons Simulation and Computing's (WSC) Computational Physics program.

A large number of dependencies can add complexity to builds, but also demonstrates software reusability. Many of those modular libraries are forming the basis for the products promoted under the RADIUSS project.

Key Takeaway #15 About 4 out of every 10 software projects report having 8 or more dependencies. Conversely, 1 out of 5 report having 0 critical dependencies.

Question 27: Of the dependencies/libraries you have that are critical to your core functionality (if you answered >0 above), what are the libraries and their capabilities you rely on? Limit to just the top 3. (E.g., "Hypre - AMG solvers", or "HDF5 - Parallel I/O checkpoint/restart")

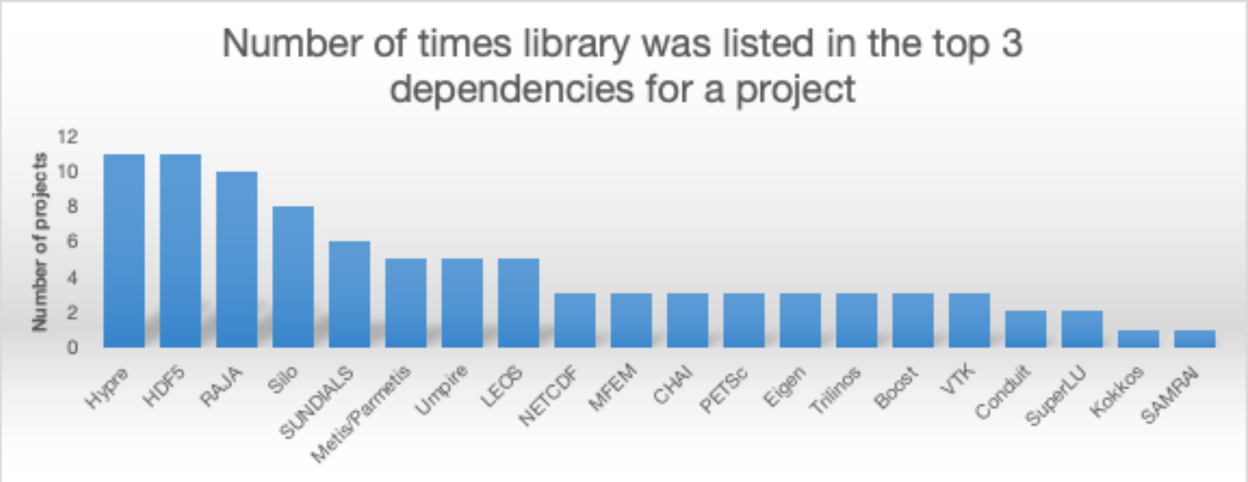
Choices:
Free form answers

Basis: This question was aimed at trying to understand which libraries (dependencies) are the most important to each project's core functionality. We limited the responses to the top 3. See comments in the analysis section for how this question (and the previous question) could've been clarified and should be better spelled out in future surveys.

Analysis: The intent of this question was to understand how much software a project needs to bring along when going to a new machine where it may not be installed, and thus the project is left to build it themselves. This was not clear in the question (nor in the previous question), and so a number of projects answered with dependencies such as MPI, BLAS, and LAPACK – all of which any reputable HPC cluster will have pre-installed. Many HPC centers will also prebuild HDF5 and various DOE-funded math libraries, so the ambiguity in this question is clear, and likely skewed the results.

For this first graph, libraries that were commonly mentioned in the top 3 dependencies were counted to see how often (out of 74 total) they appeared. But we manually removed the set of vendor libraries above (MPI, BLAS, and LAPACK) since we're speculating that a number of projects would have also listed those had the question been clarified. ScaLAPACK and OpenMP are similar in spirit and should be excluded from a future survey. However, despite this reinterpretation of the data in the graph below, the raw responses are left intact in the table that follows.

Note that for the almost half of the projects that listed 4 or more critical dependencies (see previous question), this forces them to leave out a large number of dependencies, so this data is not fully representative and subject to some discretionary prioritization for those projects with a large number of dependencies.


Number of times library was listed in the top 3 dependencies for a project

The raw data is presented below. If a project had no dependencies, they were removed from the table. For the complete list of projects in this survey, see Section 7.

AEOLUS	netCDF
ALE3D	silo - parallel checkpoint/restart RAJA+CHAI+UMPIRE - portability mslib+leos - material models
Ardra/Armus/[Radar]	RAJA, CHAI, Umpire Conduit BLAS/LAPACK Google Test
Ares	RAJA Silo (w/HDF5) LEOS

BOUT++ code suite	Hypre/PETSc SUNDIALs FFTW;
CCT	Spring 5.0 Hibernate Project Reactor
COGENT	Chombo - adaptive mesh refinement library Hypre - solver library HDF5 - parallel I/O
ddcMD	FFTW
DFTNESS	BLAS LAPACK - Most linear algebra operations
Diablo	WSMP (and backup packages PasTIX, MUMPS) - direct linear solvers HYPRE (and backup package LIS) - iterative linear solvers METIS/ParMeTIS (partitioning) Mili, Silo - visualization HDF5 - restart files Exodus - mesh database input ARPACK - eigenvalue extraction
Eiger and EMSolve	EIGER : ScaLAPACK -> solver EMSolve : Hypre -> CG, AMG, AMS solvers), silo, (hdf5 for silo hdf5) -> checkpoint/restart, parmetis (metis) partitioning partitioning
Energy Exascale Earth System Model (E3SM) Simple Cloud-Resolving E3SM Atmosphere Model (SCREAM)	Kokkos NetCDF Trilinos Sundials
ExaCMech	SNLS for non-linear solvers RAJA
ExaConstit	MFEM - Finite element library and solvers ExaCMech - material constitutive modelling library RAJA - parallelization abstraction library
FUSION (Fundamental Unified Structure and Interactions of Nuclei)	CUDA LAPACK Eigen Note: the last two are critical but can be replaced (i.e., LAPACK->CUBLAS).
Geocentric	Deal.II Trilinos PETSc
GEODYN	BoxLib / SAMRAI CGAL mpqc
GEODYN material library (a.k.a. gmlib) and 'snowninja' particle packing code	gmlib: RAJA, LEOS, BLT snowninja: hdf5, silo, mpi
GEODYN-L, geodyn material library	silo library, metis, mesquite
GEOSX	RAJA, CHAI/Umpire Hypre/SuperLU/Trilinos
GridDyn	Sundials – solvers Boost- general libraries KLU - sparse direct solvers
HELICS	ASIO- networking and timing ZeroMQ(optional but most people use it) jsoncpp-json parsing,
HYDRA	HYPRE

	Overlink LEOS
Integrated Yield Determination Tool	NASA WorldWind - 3D Earth Viewer Jackson Project - JSON Library for Java Apache Commons Libraries (Math3, CLI, Lang) Apache Batik Project - Batik Swing Components,
Kull	*Transport packages (Teton, IMP) *Material properties (LEOS, Opac) *Python
lalibe	QDP++, Chroma, Libxml2
LEOS	RAJA umpire cereal - serialization
Livermore Design Optimization (LiDO)	MFEM Hypre Metis
Livermore Metagenomic Analysis Toolkit (LMAT)	perm-je - persistent memory allocation library
LLNL UQ Pipeline	surrogate modeling capabilities - scikits, internally developed we also use scipy and other common Python libraries
MARBL	Axom -- Computer science infrastructure MFEM-- Finite element discretization hypre -- scalable linear algebra solvers
Mercury and Imp	MCAPM/GIDI - Monte Carlo particle transport collision physics RNG - LLNL random number generator library Overlink/Carter - geometry
MFEM	Hypre - AMG solvers METIS - Graph Partitioning
MSlib	MatProp for parameter input. SNLS for non-linear equation solvers.
MuMMI	Flux, scheduler GROMACS, molecular dynamics engine MDAnalysis molecular dynamics python analysis suite
nn-scattering, TensorPWE	TensorFlow X-Tensor NumPy
ns-3 contrib	Documentation: doxygen, sphinx, latex I/O: libxml2, SQLite
NUFT (Nonisothermal Unsaturated-Saturated Flow and Transport model)	PETSc
Overlink	MPI Silo
ParaDyn	HDF5 - affords a fast and standard method to share linking files between codes
ParFlow	Hypre - MGSemi, PFMG, SMG, PFMGOctree Solvers NetCDF - Parallel I/O Sundials - non-linear solver
pF3D	FFT. pF3D uses spectral methods for a number of operators. Due to the small angle approximation, we use 2D FFTs over xy-planes. pF3D passes MPI messages until it has complete rows or complete columns in the memory of a single MPI process, then calls 1D FFT functions from the library. We plan to switch to batched 1D FFT calls as we port to GPUs.
PMesh	Qt - cross-platform GUI, ACIS - CAD solid modeler, VisIt - Material Interface Reconstruction
Qbox/Qball	FFT, linear algebra, eigensolver
QMCPACK	HDF5 - Parallel I/O FFTW Boost (although we are trying to eliminate the Boost dependency). CUDA/ROCm for GPU based systems. Eventually oneAPI/DPC++ for Intel.

SAMRAI	HDF5 - parallel I/O for visualization and checkpoint/restart RAJA - threading support for CPUs and GPUs Umpire - memory management for GPUs
SCUMM	Integrators
Spheral	Python, pybind11, Boost
SW4	MPI OpenMP RAJA
Teton	PhysicsUtils Conduit/Blueprint HDF5/Silo
Topanga and KIM3D	HDF5 - Parallel I/O checkpoint/restart
TopoMS	VTK Qt
TransFort	VTK - parallel I/O MKL (BLAS, LAPACK) HYPRE - (under construction)
USER-EPH	CUDA runtime
VBL++	Qt for GUI, FFTW, HDF5, Sundials for ODE
VisIt	VTK - Visualization and analysis Qt - Graphical user interface MesaGL - Software rendering
Zero-order Reaction Kinetics (Zero-RK)	SUNDIALS - CVODE integrator SuperLU - Sparse LU factorization
zfp	CMake - Build system googletest - Testing cmocka - Testing

Question 28: About how many of each type of dependency do you have? (Include all 3rd party libraries, including optional ones)

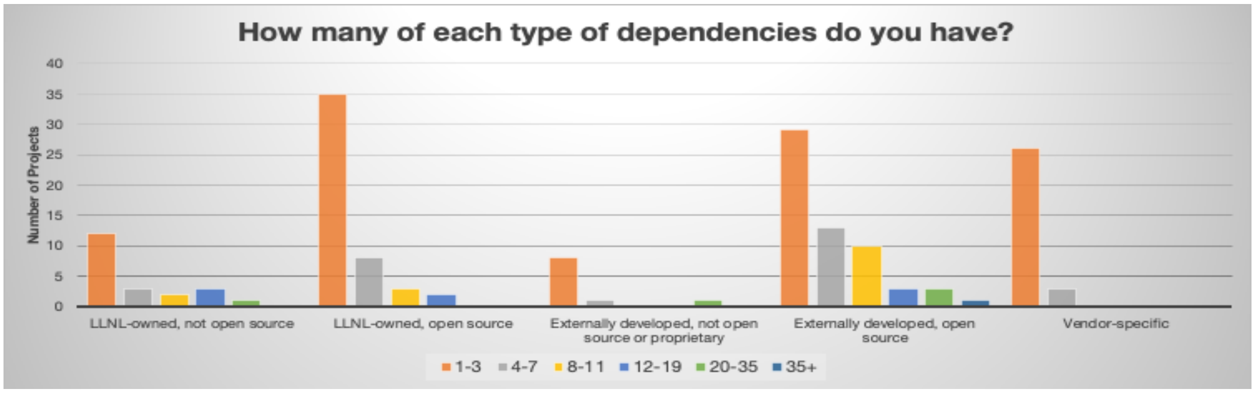
Choices:

	0	1-3	4-7	8-11	12-19	20-35	35+
LLNL-owned, not open source	○	○	○	○	○	○	○
LLNL-owned, open source	○	○	○	○	○	○	○
Externally developed, not open source or proprietary	○	○	○	○	○	○	○
Externally developed, open source	○	○	○	○	○	○	○
Vendor-specific	○	○	○	○	○	○	○

Basis:

This question aimed at understanding where people obtain their dependencies from. i.e., do they primarily rely on in-house developed software, or do they look outside of LLNL as well? And for each of those categories, whether they use open-source software.

Analysis: For purposes of this chart, if a project reported 0 instances of those types of dependencies (e.g. internal vs external, open-source vs proprietary) they were not included. The analysis shows that open-source software is the most popular, with many projects reporting a large number of dependencies on externally developed open-source. But a large number of projects also have dependencies on software developed within LLNL – especially open-source.



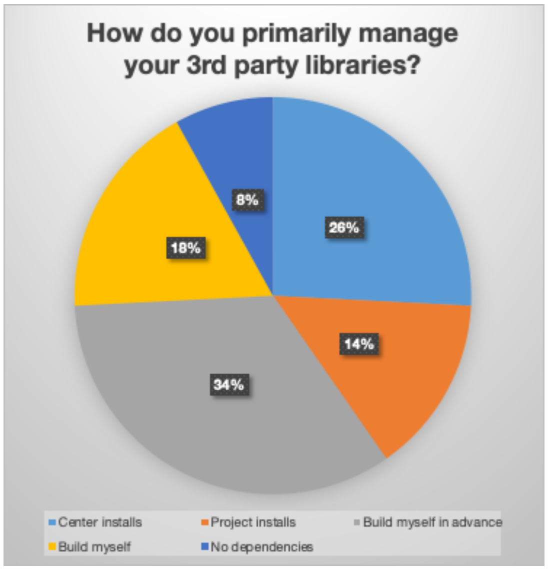
Question 29: How do you primarily manage your 3rd party libraries?

Choices:

- □ I rely on the center (e.g., LC) to install them as part of what they support
- □ I rely on others (e.g., on my team or division) to build them in a standard project directory
- □ I build them myself in advance and only rebuild when they need updating
- □ I build them alongside my project source code with every build or test
- □ I don't have any dependencies (I answered "0" above)

Basis:

This question aimed at understanding whether teams managed their own builds of dependencies, or if they relied on others, such as a dedicated project member or the computing center putting commonly used builds in a public directory to link against.



Analysis: Just over half of the respondents indicated that the build their 3rd party dependencies themselves. While many rely on builds performed by the compute center (26%) or someone else on their project (14%).

***Spack** is a tool developed at LLNL and supported by RADIUSS aimed at simplifying dependency management for software teams. Learn more at https://spack.io and in Section 8*

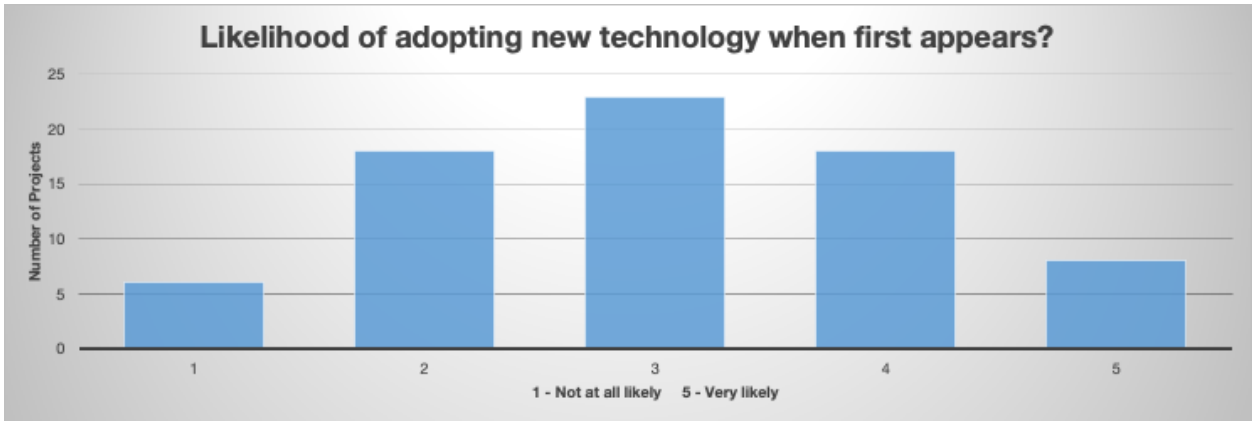
Key Takeaway #16 Over half of the responding developers and 2/3 of their projects report building their 3rd party libraries from source. 1/4 of projects depend on them being built in advance by the computing center

Question 30: How willing are you or your team to adopt new technologies/tools/libraries developed by others when they first appear? (1 - Not at all likely, 5 - Very likely)

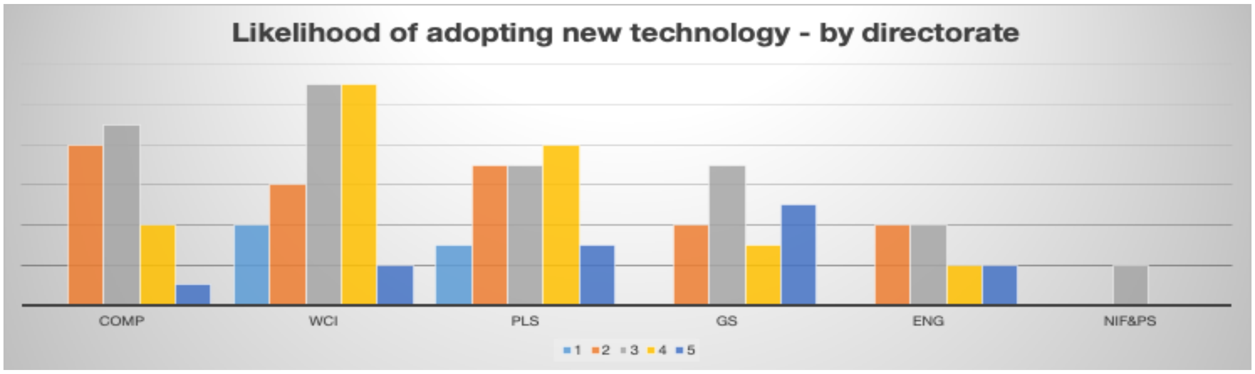
Choices:
1 (Not at all likely) – 5 (Very likely)

Basis: With new software being developed and released at an increasingly rapid pace, there are benefits and perils to being an early adopter. This question was aimed at understanding the risk level teams are willing to adopt.

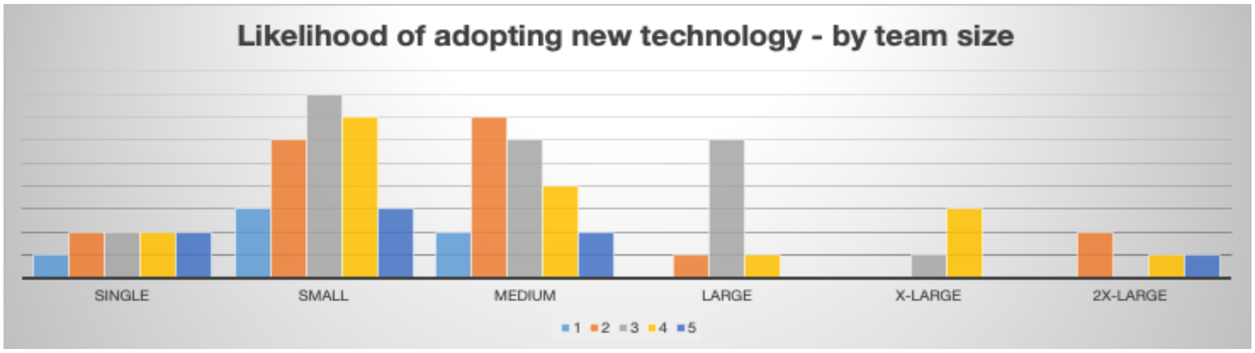
Analysis: An almost perfect gaussian distribution of risk taking at LLNL.



A deeper analysis of the data broken down by the directorate primarily supporting development reveals a someone different distribution, with GS and WCI skewing toward the "more likely to adopt new technologies" categories, and somewhat surprisingly the Computing directorate skewing toward a more conservative approach.



We also analyzed the data to see if team size played a factor in the spread, speculating that perhaps small teams would take more risk in adopting new software as they may not have resources to develop software themselves, and that larger teams as well might be more likely due to their ability to perhaps cover the risk a bit more. However, the following chart indicates no clear signal along those lines.



Key Takeaway #17 Teams are generally conservative about adopting new technologies

Question 31: HAVE YOU ADOPTED EXTERNAL DEPENDENCIES THAT LATER WERE LEFT UNSUPPORTED/UNMAINTAINED BY THE DEVELOPERS?

Choices:

- □ Yes
- □ No
- □ Not sure

Analysis combined with the following question

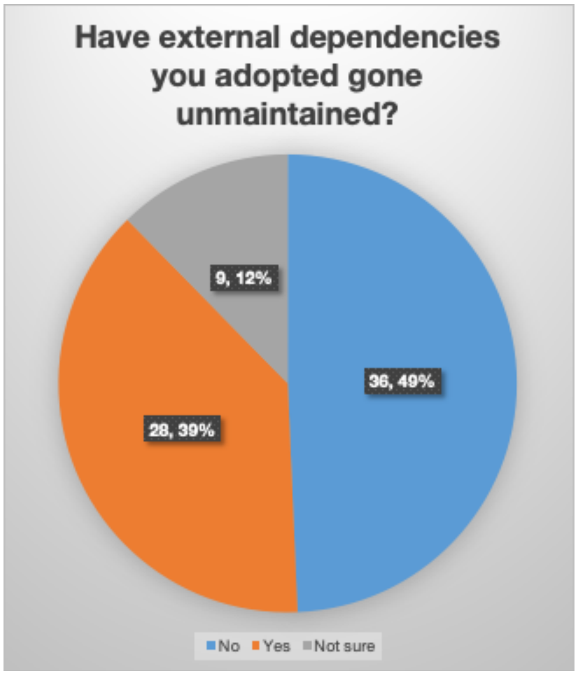
Question 32: WHAT DID YOU DO WHEN THAT SUPPORT STOPPED?

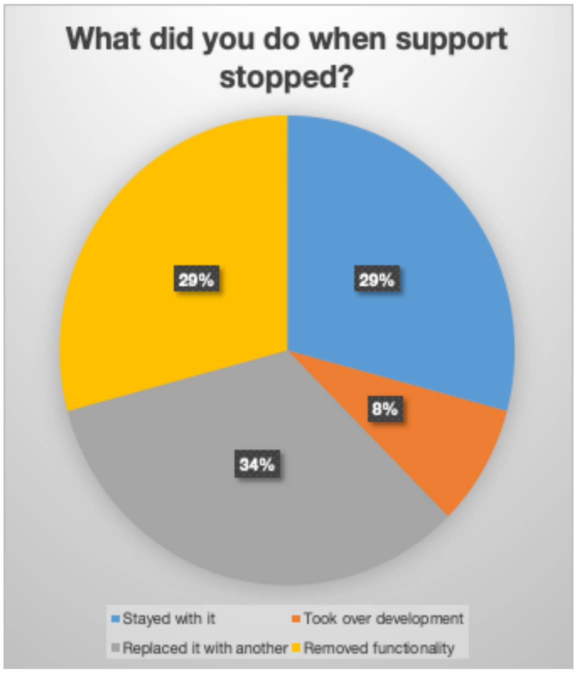
Choices:

- □ Stayed with old version that worked, making minor fixes and porting as needed
- □ Took over development of the library - we are now the primary maintainers
- □ Replaced it with another similar library
- □ Removed that functionality from our software

Basis: A common stated reason for not adopting external dependencies is that the project has no control over whether or not that external dependency will remain funded and/or maintained, thus leaving the project with a capability that they may have to take over themselves.

Analysis: Somewhere between 1/3 and 1/2 of all projects reported that this has happened to them. Of those, a follow-up question was asked about what they then did, with a very even distribution of responses between three choices: continuing to use it as is, replacing it with other software, and removing the functionality entirely from the code. Not surprisingly, only a small number of projects (8%) took advantage of having access to the source code to take over maintenance of the project.





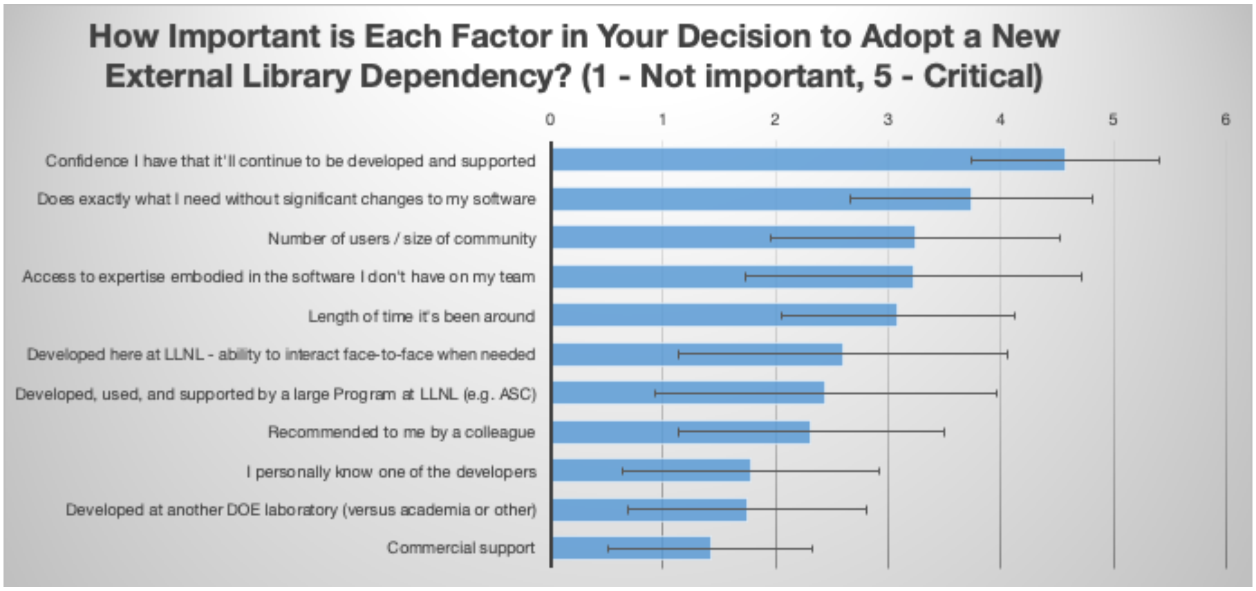
Question 33: Rate on a scale of 1-5 how important each factor is in your decision to adopt a new external library dependency (1 - not important, 5 - critical)

Choices:

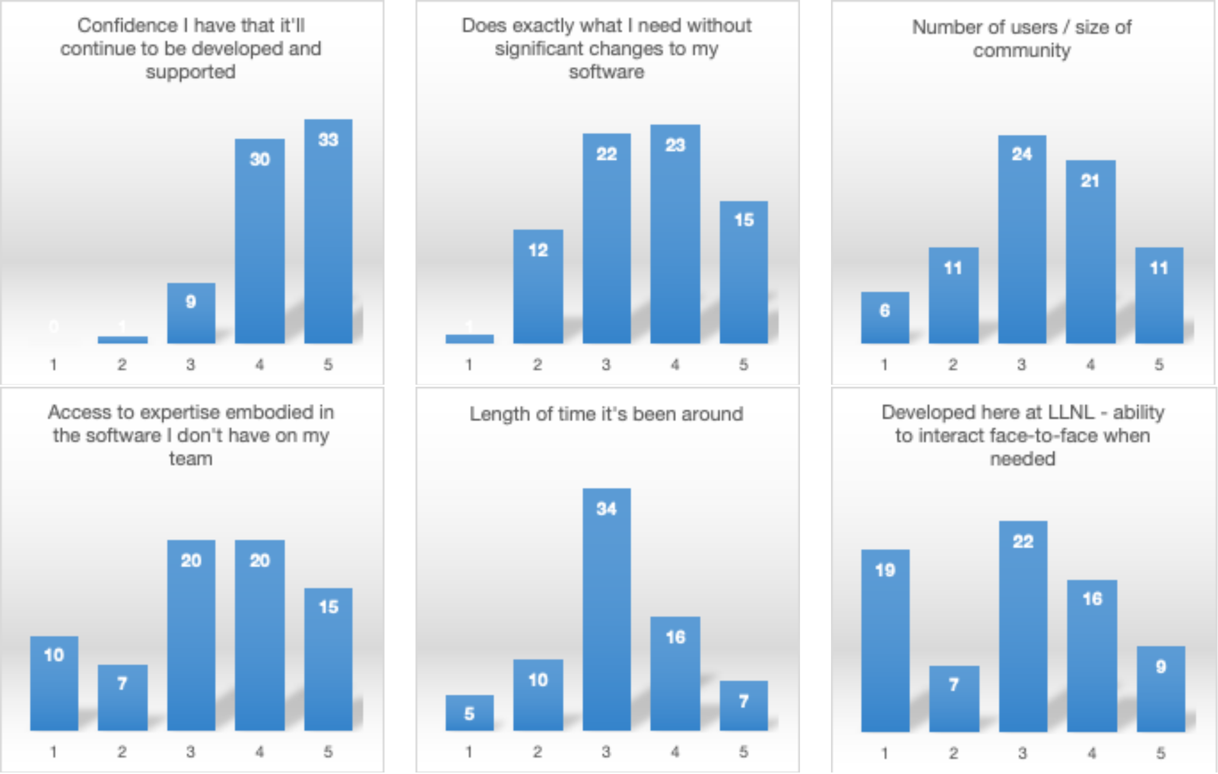
- □ Developed, used, and supported by a large Program at LLNL (e.g., ASC)
- □ Developed here at LLNL - ability to interact face-to-face when needed
- □ Developed at another DOE laboratory (versus academia or other)
- □ Access to expertise embodied in the software I don't have on my team
- □ Length of time it's been around
- □ Confidence I have that it'll continue to be developed and supported
- □ Number of users / size of community
- □ Does exactly what I need without significant changes to my software
- □ Recommended to me by a colleague
- □ I personally know one of the developers
- □ Commercial support

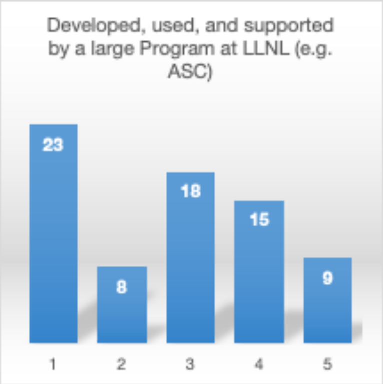
Basis: Lowering the barriers to adoption of reusable libraries is an important strategy for RADIUSS and the laboratory. This question aimed to understand what some of the most important issues are in that consideration. The follow-up question after this asked for open ended answers to this question, which were enlightening in that our initial choices clearly missed some common reasons.

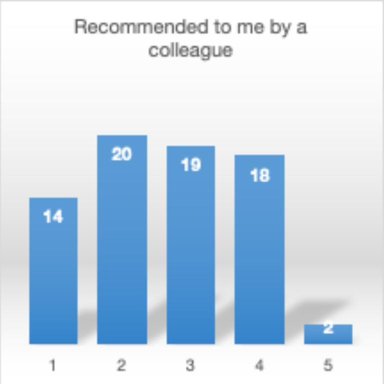
Analysis: The first chart below shows the average response across all projects, with the error bars indicating a standard deviation. Not surprisingly, confidence that the software will continue to be maintained stood out as the most important factor. Perhaps somewhat surprisingly, where that support came from (e.g., ASC or commercial) was not as important.

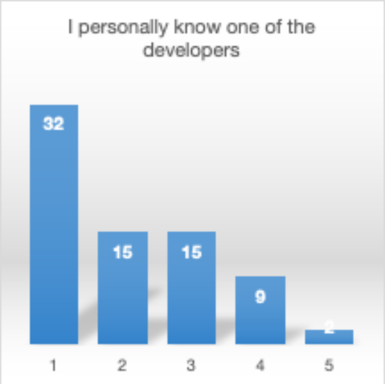


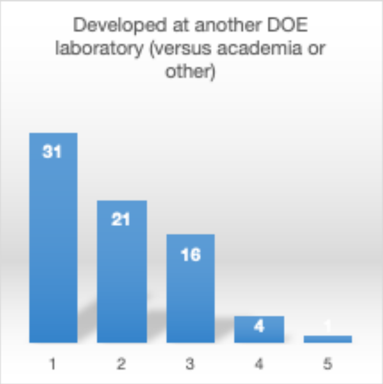
The detailed breakdown for each question is below with each bar representing the number of responses, and 1 – Not important, 5 - Critical

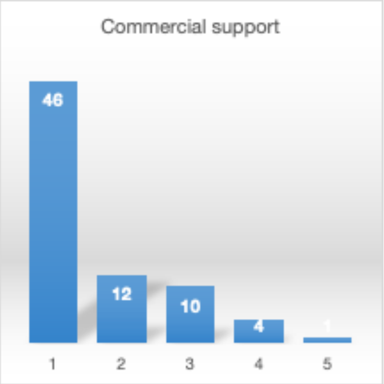



Developed, used, and supported by a large Program at LLNL (e.g. ASC)


Recommended to me by a colleague


I personally know one of the developers


Developed at another DOE laboratory (versus academia or other)


Commercial support

Key Takeaway #18 Confidence that a dependency will continue to be supported is the top reason for teams to adopt and continue to use open-source projects

Question 34: Any other important factors that influence your decision on external library or dependency adoption that were missed above? Where would you rank them?

Basis: While the choices in the previous question were extensive, they were certainly not exhaustive, and thus we offered the opportunity for free form response. Just shy of half of the projects responded.

Analysis:
While all of the answers were insightful, if we were to give this survey again in the future – we would consider adding the following choices based on these responses:

- License compatibility
- Scalability
- Difficulty of implementing the functionality ourselves, either due to lack of *expertise* to develop the capability in-house, or the lack of *resources* (time, developers, funding) to do so
- A local "champion" to drive inclusion in our product
- Availability of quality documentation and example usage
- Availability of a Spack package
- Portability

> We are scientist and as long as the software produces valuable scientific results with reasonable amount of software development effort, we are fine with that. You can read it from the other direction: if we are forced to reduce scientific activity due to too much work in software development we will need to rethink if it is worth doing it.

License compatibility and track record of security reporting/response
The above answers apply to FFTs. In the case of <our project>, it was a big help that <developer’s> office was down the hall and that he was willing to support the needs of HPC codes (e.g., a 3 million MPI process job). We are moving to Python because <he> retired and because Python has wide adoption and support in the scientific community. We do have some concerns about whether MPI support for Python will deliver good performance for very large jobs. Python has caused a lot of users’ problems when importing packages in large node counts runs. We think that Spindle will deal with that. Yorick did an excellent job of delivering high performance I/O in very large runs. We plan to shift from Yorick's internal PDB package to HDF5 as part of our port to Python. This adds a new external dependency, but HDF5 has wide support in the scientific community. We think HDF5 will perform well if we call it in the right way, but that is not yet proven at scale.
Documentation
People on my team should have some time to devote to the adoption process (Very important)
Manpower required to implement the new tools.
Long term support, packaged for standard distros,
Given that <our library> is part of the wider ASC code ecosystem, how things will be accepted and interact in that ecosystem matters significantly.
It's important to us to have a developer that is willing to champion a new library and take a large part of the burden of interacting with it and its developer team. If someone is enthusiastic, we're much more likely to take a risk on a new library, assuming they have the time.
That the external tool is properly supported with documentation and at least some availability for someone to answer questions. Also, how willing they are to listen to requirements and the ability to support new requirements.
Documentation and Good Examples
(5) Reliability; (5) Speed; (5) easy to use
Presence or absence of a Spack package!
Based on need. In general, we will avoid dependencies unless they have widespread use/support, this is essential.
It's nice if the software isn't too much of a "black box" if possible (3).
The main limitations we have run into on using external libraries are availability to build on all the different platforms in use and the license must be compatible with BSD-3.
Ability to build on all the used platforms, and license compatibility.   Both of these are critical if it doesn't meet them, we can't use the library.
The library fulfills an important need, such as performance analysis, or a better inline interface to my code.
Some other factors would be-- Libraries that are also used by our closest on-site collaborators and customers. Libraries that do not cause our customers to see bloat in their external dependencies.
Yes, prefer libraries to frameworks. (frameworks imply all or nothing adoption).
If the library has a test suite, and way to report issues, and has been making frequent patch releases. If not, we would adopt if it is a simple library. We use commercial when we have to - i.e., Qt and HDF5.
We generally try to avoid dependencies to the extent possible to lower the bar for experimenting with and adopting zfp.
Portability.
If the Open-Source software is owned or developed by well-established company, companies, or open-source foundation such as Linux.
Commonality among multiple code teams within larger program (4).
We avoid this dependency as much as possible, but it is becoming much more difficult as the project requirements and computing requirements have become much more complex in the past 5 years.
Does it solve a problem that would be too expensive to develop or maintain ourselves? I would rank this a 5+.
Willingness to add features. Willingness to allow our developers to add features.
Try to minimize external dependency to ease porting to next generation hardware. If external dependency adopted need to be confident that the library will be quickly ported to next generation hardware or a replacement will be available on the new hardware (very important)
Broad use in external community - 3
LLNL program support/interest and active developer support are the major factors
For me, it is important to have compatibility with different OS and running environments
Is the external library on our critical code path or is it an optional feature that we can easily disable, e.g., on unsupported platforms.

Section 6 Pain points – long form answers

This final section consists of entirely free form responses, and the answers are provided with minimal editing. Common themes are called out before each detailed table of responses

Question 35: WHAT ARE YOUR BIGGEST PAIN POINTS WHEN BRINGING ON NEW USERS OF YOUR CODE?

Basis: New users of your software are almost always desired, but also almost incur more work on the developers to support them, which can take away time from developing new features or exploring new research directions. Lowering the bar for new users is important to build a case for continued and/or increased project funding.

Analysis:
Common answers included:

- Training in proper use of the code
- Lack of documentation in how to use the code
- Porting to all the systems user's demand
- Easing their ability to build the code and its dependencies themselves

Training them to understand the code
Training
Compiling on a different platform could become pain.
Limited number of team members available for deep engagements
Our documentation is critically lacking, and I am perpetually fighting to convince funders that they should be willing to give up new features for documentation and more tests
Installing/compiling - User base does not have developer background, so compiling is a problem even with a CMake configuration process. We don't have resources to provide binaries for every platform. Not even clear how to provide binaries for HPC systems. Have created Docker image and setup automated Docker deployment which is helping some. No Windows support - Many users want a Windows support. Currently using Docker approach to help address; in progress not sure this will address need.
model complexity misused by overconfident users. Asking for material fits urgently with very little knowledge of their own needs. not reading the manual
Installation of software + libraries; enforcing version control.
No particular pain points: since the code is open source, it is sometimes downloaded by people in 'exotic' institutions who don't know what they are doing.
Building on new systems
User code written without third party libraries in mind.
The format used for user input can be confusing owing to multiple options that are not well documented. The lack of clear documentation that spans all possible use cases for the code, coupled with the fact that one needs to be an expert nuclear theorist to understand what the codes do leads to a significant lag time until a new user can be productive and confident in their results.
Lack of domain knowledge, either Discrete Event Simulation or communications networks and systems.
Trying to make input format simple
Interpreting users as host-codes: our documentation is not great, and most code physicists who are asked to engage do not actually have the relevant training.
It's probably between building all the dependencies and our limited documentation on how to run everything.
The biggest pain point for bringing in a new user is typically related to their knowledge of the application space. All users have a learning curve to come up with how things are done with the code, but helping users come up that curve at the same

time their coming up the application space curve is the biggest challenge. The code team shoulders some of the application space education because the user is making mistakes based on their ignorance of what matters in the application space and code is not behaving well because of it. People tend to notice problem setup and ALE has stumbling points because they're obvious activities, but the space of user mistakes in the application area is much larger and time consuming to correct.
Our lack of documentation. We need to develop both user and developer documentation.
lack of documentation - steep learning curve
Standardization of install process on *all* architectures and environments.
The code is installed in /collab/usr/gapps so access straightforward.
It’s a very complex system with a lot of dependency
Lack of documentation
Getting them to adopt new codes vs extremely old unsupported FORTRAN codes, this is a nuclear physics culture problem. Ease of access outside LLNL.
Build the code a new platform
Our code has a huge user interface, over 1500 function points. Users are all very occasional users. Nobody, anywhere, goes to school to study meshing, which seems like it "should" be easy, so it's often hard to convince them to take it seriously.
Documentation
Code requires HPC resources, not readily accessible to broader bioinformatics community; having cloud options for users would make the software more readily accessible to collaborators.
Complicated method and complex workflow. Users in the community are used to simpler/ approximate approaches with much simpler workflows.
We have a very specific set of users that do not change very often.
Training
Our build system is complex and opaque (partly because we support a large number and variety of machines).
We haven't been able spend as much time on documentation as we would like, so it's hard for a new user to learn problem setups without significant help from the developers. We maintain a set of sample inputs, but they are not comprehensive and are hard to keep up to date.
Documentation and examples
Documentation.
Tutoring usage of an ALE code
User errors in the input file (documentation), user mishandles the problem data (does not rename a variable as needed, or neglects to include a units conversion factor, even though these capabilities are available), user difficulties in plotting the Silo files in VisIt.
Teaching them how to use it, especially when they are used to "point and click" GUI type codes. But usually, we try to find another user to act as a mentor. We also have a really big user manual
There is a significant learning curve for a new user. We have good code documentation but not much in terms of teaching someone how to get started other than to begin with simple examples and expand from there. When <our framework> is incorporated into an existing code, it requires significant refactoring of things in the old code such as the data management, parallel decompositions, and timestepping in order to align things into a model that works with <our framework>. <Our framework> has its own input file syntax that isn't directly compatible with the input syntax used by any other code. Assuming a user code doesn't want to completely adopt our syntax, it usually requires a layer of in-memory translation to get things working.
Bugs, untested parameters, compilation
The biggest pain points are (1) a lack of "how-to" documentation with clearly explained examples, and (2) many new users are actually learning much of the specific physics along with the code. That leads to complaints - primarily from users that don't use our rad-hydro codes - that the code is too hard to use.
There is no direct funding support and time to support new-user training.
Having our software work out-of-the-box on a wide range of platforms. Getting access to a wide range of platforms to develop and test our software on. Getting the latest versions of Mac OS are currently the biggest issue.
We have developed a training course that we sporadically offer, which is an effective way of training new users. Teaching that training course does require a significant investment of time. In addition, new users often have questions that require time to answer.
Common environment setup. We write helper scripts in bash, sometimes they prefer tcsh. Or they prefer windows and don't understand Linux.

1) Our code has no user interface. 2) Both source code documentation and user documentation are lacking. 3) Our code has many "knobs" that need to be set just right, i.e., we don't have intelligent or automated or adaptive default values. There is a big difference between "grad student code" and "production code", our code is in between.
This fortunately tends to not be an issue, as our users often get started on their own with little help from our team. The biggest challenges arise when the users have already adopted <our library> and want to make it do more advanced things
Getting up, running, and tested on new platforms for them.
Porting to new platforms for them.
We have extensive documentation and examples. But they were all created in the early days of the project. The code and interface have been changing more rapidly than we can update all the old samples. Training is not formal, but mentor-based.
Lack of resources to help them or support common software stacks.
Our input deck syntax has a steep learning curve.
installation of third-party libraries
The biggest pain point is the use of internal and external meshing tools to create the problem. Then, users struggle with learning how to run the code properly and the code's old input syntax.
Not knowing that somebody new is using our code because it is a TPL for much larger codes. If you consider the host codes as users, biggest problem is that decisions are made by host codes that only work for them and not for other host code customers.
- Gaps in understanding the physics (most of our users aren't domain experts in transport). - Code output that is easy to ingest into people and other codes. We have a lot of text output (with all kinds of result information) and Silo output. A lot of our data is hard to visualize in VisIt, and hard to get into things like Excel.
Documentation is still incomplete.
Documentation
Matching host code infrastructure to library API. They may think about the domain in a different manner than we have previously. Integrating new library features to eliminate redundant host-code work.
having relevant demonstration examples that a prospective user can easily relate to and can demonstrate capabilities and thus value.
Get them to read the documentation
Educate lots of basic usage of the code.
Since we're at the early stages of bringing on users, it takes a lot of effort to help each user get up to speed with our workflow. E.g., using Lua to drive the code, setting up meshes with PMesh, describing features of our code, ... We've put in a lot of thought and effort into providing users with a streamlined environment for using our code. E.g., visualization support via GLVis, visit, ascent and devil ray, easy output extraction using ultra/pdv curves, integration with Jupyter and merlin. Each of those supported features have to be maintained, and we sometimes only find out that things are broken when a user tries to use the feature in an unexpected way. (This is also helpful since it improves the robustness of our software/tool).

Question 36: WHAT ARE YOUR BIGGEST PAIN POINTS WHEN BRINGING ON NEW DEVELOPERS OF YOUR CODE?

Basis: As development teams grow by adding new developers, it is critical to bring those developers up to a productive state of contributing as soon as possible. But that takes time and energy of the existing project developers.

Analysis:

Common answers include:

- Training them in the design, algorithms, and usage of the code
- (Steep) learning curve for the above activities
- Lack of developer documentation

- Understanding the underlying physical models
- Learning the programming language
- Difficulty of attracting developers to use Fortran

training them to understand the code
Bringing up to speed
Getting up to speed with content of large application. Getting up to speed with processes.
Learning curve for programming language, and HPC framework. Also lack of documentation/ developer manual.
Not sure. It all depends. When things do not work the reason could be as many or more than number of developers. Difference in opinion probably is?
Combination of domain knowledge in both math and cs
Also, no documentation
Lack of developer documentation is an issue.
Lack of code documentation - Code does not have enough documentation for methods, data-structures. Was an extremely bad decision to not force original developers to document their code. Have put in place new standards to enforce adding documentation. Old code base - Code is 27 years old, is C based. Could use refactoring to use C++ and newer practices
code complexity (for non-CS majors), model complexity (for CS majors) It's relatively straightforward. However, people don't like Fortran dev so it decreases enthusiasm.
Getting new developers up to speed on heterogenous code base.
* Complexity of underlying physics models: DFTNESS implements state-of-the-art nuclear density functional theory methods. * Complexity of the methods: actual implementation of DFT involve a lot of linear algebra, spin algebra, tensor contractions, group theory, etc. * Complexity of the code: the code is large, and it takes months/years to be able to navigate it
Not frequent enough to know
Lack of developer documentation, lack of on-boarding documentation.
Understanding what is being calculated.
The code is completely undocumented when it comes to the types of ad hoc algorithms used to make things faster. Equations that are presented simply on the relevant scientific papers are computed using various strategies that are not referenced and thus a new developer cannot quickly understand what is being done. Comments are quite sparse even in the more newly developed parts of the code, which means that in order for one to find out exactly what something does they have to either spend a lot of time on it or ask the person who wrote it. The use of Fortran for the bigger part of the code further makes it difficult to interface with the newer GPU accelerated nodes.
Lack of C++ experience LC's uneven willingness to install new packages, new versions, multiple versions of packages. LC appears to do that for some programs, not others. Requires more work from us to get the right version of a fairly standard tool installed in a project directory. Harder to test other versions.
It is a big code, with many developers and many modules it can be a steep learning curve for new developers who don't have a computer sciences degree. Our developers are mainly hands-on geophysicists.
Understanding the legacy Fortran methods
Finding people with both good coding skills and sufficient subject area knowledge.
The code was written as a replacement to Fortran coding that performed well on CPU-based HPC but that was a nightmare to maintain and that was not going to get ported for GPU usage. The code has many vestiges of that history, in part to facilitate migration of more functionality from the old Fortran code. So it is not as readable as it could be.
It would be getting them up to speed on MFEM on how it works, since we rely heavily on that library to do a lot for us.
Bringing new developers up to speed on how to navigate a large code base and educating them on the code architecture so that they can successfully read and modify the code. It's also difficult to bring new developers up to speed on the application space and how to use the code. For physicists, they need more in-depth knowledge of this, but usually have some degree of prior training. For computer scientists, they need to become familiar with it, but may have little to no background.
Same as above, lack of documentation. We also have a single point of failure (myself) for explaining aspects of the code, which needs to expand.
lack of documentation - steep learning curve
Perhaps, documentation.

It’s a very complex system with a lot of dependency
Building TPLs
Funding
To get a good understanding of the whole code
Our coding standards are not as well documented as I'd like, and we don't have a sanitizer.
Documentation
Users that are good software developers and familiar with the domain are less common.
Steep learning curve, complicated code, advanced C++.
Complexity of the code as it covers a wide range of features in order to be a standalone application.
Training
Staff are often physicists with poor software engineering skills, while software engineers often don't grasp the use case necessary for completing the scientific experiment. Basically, communicating across disciplines is hard.
Helping them understand the structure of the code, given an ever-growing increase in options for solving different types of problems.
Developer documentation and standard, and examples
Documentation
Familiarizing with code and requirements
Initialization, including initial clone, compilation, and running the test suite. Learning the data structures of the code. Developers have been able to work effectively since we have an API interface to the Overlink and Carter libraries.
We don't have huge pain points, but Fortran is a troublesome language to a lot of people. Realistically, **our** code is pretty large and it's not always easy to know when some feature already exists than can be re-purposed for a new requirement. So just showing people "where stuff is".
Again, the learning curve is a big factor.
Learning curve
This has only happened once - moving from 1 to 2 developers - so it hardly qualifies as a pain point. However, that experience did point out the lack of any documentation explaining how the code is structured and how it is intended to function.
There is no direct funding support to bring on new developers.
Complete developer documentation for all portions of the software.
The breadth of physics and software in our project requires a significant learning curve (our project is similar in this respect to other projects in WSC, however). Effective mentoring of a new employee takes dedicated one-on-one time. New avenues for communicating software processes such as Confluence are a significant benefit in getting new developers up to speed. But we still need to communicate a lot of information verbally.
Again, setting up the common environment. Usually takes a few days to get all the tools. We have most of the steps documented, but sometimes we've missed one or two.
1) It seems impossible to find new employees who have BOTH domain knowledge AND good programming skills. 2) Our test suite is fragile, hence folks don't use it, and code gets broken.
Educating them on compression and a very diverse and sometimes complex piece of code. Knowing all of zfp requires familiarity or preferably expertise with C/C++, Python, Cython, NumPy, Fortran, OpenMP, CUDA, CMake, googletest, cmocka, Sphinx, git, GitHub, TravisCI, AppVeyor, ReadTheDocs, and Linux/macOS/Windows compilers. zfp also relies on some challenging and unusual programming concepts such as C templates and namespaces, and support for multiple concurrent instances/versions of zfp.
Students who need significant training in the importance of automated memory access and leak checks -- thinking "if it works in this case on this platform, everything must be fine."
Teaching students the importance of automated memory access and leak checking.
We have extensive documentation and examples. But they were all created in the early days of the project. The code and interface have been changing more rapidly than we can update all the old samples. Training is not formal, but mentor-based. The size of the project, and what it needs to do is a challenge to fully understand.
Learning curve can be steep, especially with regards to new technology: containers, Kubernetes. We have mitigated this to some extent by having training material and reference examples they can work through to understand concepts.
We rely on the dependencies from the USQCD software stack. As such, there is a lot of templating and enumeration that makes lalibe seem like its own language. This can take a while to catch up on.

Understanding of the math behind the solver
The code is Fortran: CS developers don't learn Fortran - most graduate engineering students do. The code contains a large numbers features and core capability that just require time to learn.
Lack of understanding of numerical methods, code base shifts rapidly so new users find it hard to keep up. The changes based on OMP and CUDA for GPUs is staggering, no one person is expert in deterministic transport and using these programming models, so teamwork is key
- We lack a developer's guide, that would go a long way. However, we have started down this path. - Complexities of building on HPC clusters. New developers often think our build systems are overly complicated (an opinion that fades quickly) - Sheer complexity of transport can make daunting code for new people to understand
Unfamiliarity with modern **C**++
Code standards
Coming up to speed in domain. Broad scope of technology/skills needed. Knowledge transfer.
1) Learning the HPC is a big curve 2) Physical distance between CS people seated in Q only areas and the new developer located in the unclassified areas
Help them to understand the physics model underlying the codes if they are general software developers. If they are scientists, it takes time for them to understand the coding structures for a certain scientific calculation.
There is a lot to learn! Our team has made an effort to streamline the build process so that all developers can build the code with all TPLs from scratch and update the build system, when necessary, but there is still a lot to keep track of and learn about. We often run into issues with LC group permissions. It is difficult to test which group permissions are necessary since our current developers already have the proper permissions. We try to update our documentation every time a developer gets onboarded.

Question 37: WHAT ARE YOUR BIGGEST PAIN POINTS REGARDING LONG-TERM MAINTENANCE OF YOUR CODE?

Basis: This question aimed at understanding where projects felt they were going to have to put a lot of energy to keep their project sustained going forward.

Analysis:

Common answers include:

- Porting to GPUs
- Sustained funding
- Responding to external feature requests

I'm worried about switching to GPU's because I am the only developer and I write new algorithms constantly and porting them to GPU would severely limit my ability to progress at a reasonable rate
Upgrades to GPU and OpenMP
Lots of technical debt in core infrastructure developed 10-15 years ago. Poor encapsulation practices then have infected 1M lines of code, making refactoring to remove major undertakings. Sufficient resources to do exhaustive testing for combinatorics of compilers and hardware. Cultural difficulties in upstreaming in-house changes to externally maintained libraries.
The code is mostly written in a procedural fashion. As the code got bigger, the need for an object-oriented framework became apparent.
Lack of sustaining funding support. Architecture change of HPC system that is not in favor of our software.
Balancing capabilities with code complexity. More comprehensive testing.
The recent Java transition to six-month release cycles has been causing (hopefully) temporary problems where my support matrix has exploded as some of my customers at funding agencies are lagging far behind. For instance, I have customers with glib-c < 2.10 and Java 6 still.

<our code> has multiple versions of some packages (e.g., hydro). We also have a lot of physics flags in our packages. The GPU port looks like it will result in separate (but related) GPU and CPU versions of each function that needs to be run on the GPU. The net result is that we have too many lines of code to maintain. We hope to remove some old and rarely used packages by moving their features into the "mainstream package". Another pain point is that <our code> does not have unit tests. We developed a set of <code> kernels as a tool for porting to OpenMP and tuning to obtain high performance. The kernels have correctness checks for the answers from individual functions. We hope to find a way to use the testing suite develops for the kernels to improve our checking of <code>, but we probably won't be able to work on that until after the GPU port is complete.
Lack of SE funding - Project has gone through several multi-year periods without CS support.   Trained CS support is 0.4 FTE's, and most contributions are from domain scientists. Lack of SE staff - Most of the project are domain scientists with limited SE backgrounds/training. Contributions frequently need work to maintain code architecture, standards, testing practices, etc.
library is a relatively new addition to big codes and so management of user needs vs. staff availability is changing.
it should probably be rewritten in C++ for longevity and to utilize HPC tools. I have written the same code 3 times now so it's not too cumbersome for an early career person to take on.
Libraries becoming outdated and incompatible; dealing with multiple programming languages in the project; issues with new architecture support.
* Legacy code: <code> was originally single-source file, legacy Fortran 77: transitioning to a modular structure with more modern and flexible Fortran takes precious time that is not rewarded in the nuclear theory community * Workforce: nuclear theory community, especially DFT, is too small in the US
Lack of testing practice
Maintaining funding (although the ISCP funding has been very appreciated and helpful) for keeping test and build systems up to date. Also, our funding usually covers large-scale systems and deployment, but we have many users who have only small math models and serial systems, and we have no ability to support them. Lastly, we have outside developers who contact us wanting to do interfaces from other languages/environments (such as Python, Julia, R, etc.) and we have little support to work with them to ensure the produced interfaces are of high quality and using our code correctly.
Changing architect and supporting software (e.g., compilers).
The lack of unit tests is really the biggest problem. It is made worse by the lack of small functions which can be tested separately.
We're slow to respond to bug reports, patches, improvements from upstream.
Reworking for GPU functionality. And very few people have the skills to debug tricky edge cases that can arise in production simulations.
Making the code, and its use through the ExaConstit **MFEM**-based finite element code, more user-friendly will be a challenge given effort available. We would like to get the code to the point where it can become a community code and much more widely utilized, but supporting that kind of effort is a challenge, and may become even more challenging at the end of the current ECP (DOE office of science) activity.
It'll probably be supporting the various accelerators out there as runtime backends for the code.
The size and complexity of the code is pretty high. Much of the code has grown organically over the years, and we've been afforded some time here and there to clean it up into something scalable. This process of tidying up tends to be the first thing to go when there's a high demand on the code, so if we ignore it too long, the code will become too unwieldy to add new features to. The explosion of new libraries and interdependence of them also increase the code's complexity. There's a large coordination aspect to them, especially when using common libraries, like RAJA. This really requires every library to have somewhat active development if the common pieces are changing over time so that they can keep up. Like most code teams, we're also stretched pretty thin with multiple single points of failure for the code. The build system is probably the most critical of these.
Probably expertise/knowledge of how the code works, and further expansion of the testing system (both in terms of coverage and automation).
funding/support
- New: Porting to new architectures, which are now accelerator based and difficult to program. - New: Compilation across different platforms, which is much more painful than before, mainly due to architecture variation and the need to support C++ (which is less portable than C). - Maintaining expertise on the different components of the code as developers leave the project.
Funding.
Currently it is the changing of Hypre interface, presumably this will stabilize after the GPU integration.
Keeping it up to date with the newest version of all dependencies

This is a legacy code, so we do not have long term maintenance plans. The project is only active while we migrate capabilities to new platform.
Finding time to write documentation and comprehensive examples
Because of multiple teams around world, (1) they may setup different different repos; (2) their changes may **have** broken the code
500,000 line code base. Spack helps a lot with libraries but it's still a ton of dependency management. GLX (OpenGL over X) support is slowly rotting away, so we don't have a path forward for interactive graphics. There's no decent open-source CAD kernel so we are doing a huge port away from the one we like but whose source code is unavailable.
documentation
We need to be able to have some targeted software development support. Our sponsor funding is focused on the application not the software unfortunately.
New architectures required code redesign. Lack of stable/adequate solutions for device support in modern C++.
Not having enough Unit tests to automate finding dependency behavior breakage.
compiler changes, new architecture for computers
Evolving architectures require continuous updating for good performance, but our code is so huge that such updates aren’t possible very often.
Ever increasing complexity and concentration of knowledge about how certain parts of the code work by the specific individuals who developed them. The research deliverables keep pushing us forward to make the code do new things, but we never get a chance to go back clean things up like we'd like to. The continued existence and support of the Chombo and Hypre libraries upon which our code heavily depends is also a long-term potential concern.
Funding for maintenance. Aside from that. One thing we have encountered on occasion is that the oldest version of compilers support is driven by the newest version that the HPC platforms support.   So long term support means continuously testing new compilers and libraries as it comes out and still maintaining support on the older versions.
Funding for maintenance.
Porting for use on GPUs. Occasional need to switch to new third party libraries.
Having the rug pulled out from under us (CUDA being replaced by HIP/ROCm, Autoconf needs to be replaced with CMake, Spot 1 will be discontinued in favor of Spot 2, Subversion was replaced with Git, etc.).
code bloat. Trying to keep code from getting duplicated, re-created. We have our own testing and building infrastructure, git, bitbucket, Jira. There will be pain when we move to **BLT**, whatever testing framework is finally decided upon. We are a Fortran code, that was decided long ago. It isn't the easiest language to keep "clean". It isn't the easiest code to adapt to new technologies like GPU's.
lack of version control system :-)
Our continuous integration is not as good as it should be. We need to make better use of these resources to make sure we are testing a sufficient set of systems and configurations with and without our optional dependencies. Also there remain sections of the code that need better test coverage.
evolution of main code (LAMMPS), for example we developed for CUDA while they were transitioning to Kokkos. Kokkos version of the code was hard to understand.
There is no direct and stable funding support for long-term maintenance of the code.
Developing and testing on a wide range of platforms.
Our project is fortunate in that we have had a consistent core of staff that have been on the project longer-term. However, there are some specific capabilities in which the person who originally developed those capabilities is no longer on our project or at the Lab. As a somewhat smaller project, it can be difficult to have multiple people to cover all areas. As a result, when someone leaves, we may have capabilities not well-supported.
We haven't been around all that long yet. Eventually it will be backwards compatibility with prior versions I imagine.
1) code maintenance is not valued by management ,2) hard to find people who enjoy this type of work, 3) libraries evolve in non-backward-compatible ways
Having to support a huge feature space resulting from the Cartesian product of external scalar types (IEEE half/float/double/quad, bfloats, posits, integers, ...), array dimensionality (1D, 2D, 3D, 4D), programming models (serial, OpenMP, CUDA, HIP, AVX, FPGA, ...), programming languages (C, C++, Python, Fortran, System C), low/mid/high level APIs, compression/decompression features, ...
Need to maintain backward compatibility of input files to the fullest extent possible.
Maintaining backward compatibility of input files to the extent possible. Updating older input files when necessary.
Over the decades, team turn-over has meant some critical parts of the code are now black boxes. The methods as well as the code design are lost when developers leave. Part of this is because we don't have enough written documents on new methods. Milestones and user features are highly rewarded. But often to get those done on time, you incur huge amounts of technical

debt. We seldom have the time to go back and clean up/refactor the code to be easier for long-term maintenance after the rush to get the feature is done.
It doesn't use any architecture agnostic paradigms such as raja, Kokkos, OpenACC, etc. Therefore, some kernels have to be rewritten when HPC systems change.
lack of available software developers at the lab (and of an internal "market" to find such folks)
While the code is only a half million lines, it takes time to keep it "modern". There is no automated why to incorporate new language standards and coding methodology into it.
Code base shifts rapidly to eke out performance. Testing of code base is not nearly good enough and features we have agreed to develop fall into disrepair quickly
We have made great inroads on performance portability abstractions (RAJA, CHAI, Umpire). Our code already had a pretty good architecture, and we are continuing to improve that through extensive refactoring. The biggest issue we have is maintaining performance across our platforms of interest. We seem to have ups and downs in performance... some explainable and some not. We also don't have (yet) a good, automated performance monitoring system in place.
multi-platform testing.
consistence funding for code development
Increasing rate of technology changes. The proliferation of machines and packages, and the need to build and test the combinatorial problem. Hiring new developers. Knowledge transfer.
adoption of new libraries and new tools. Example is migration to GitLab for regression testing/continuous integration.
1) Finding people who are interested in doing long-term maintenance 2) Stable funding
Keep the compatibility with new or update libraries, particularly for some core libraries like MPI, OpenMP, and compilers.
We've automated a lot of CI tests, including regression, performance, memory, common build configurations, visit tests. We've also automated a lot of code checks via clang format, clang-tidy. These have been very helpful in catching problems early. However, it is difficult/impossible/unreasonable to test all code configurations. Due to the size of our team, we're often one-deep in a lot of the codebase and build/workflow processes. As a representative example (there are numerous others) most of the [CS] team understands the build system, but a lot of the magic has been developed by a single developer who is retiring in the near future and who will be very difficult to replace. On the other hand, this has clearly been beneficial with respect to unity of vision/execution. On top of that, due to the extreme modularity of our codebase, a lot of expertise is external to our team.

Question 38: What can LLNL do to mitigate your pain points?

Basis: The RADIUSS project which spearheaded this report is one important potential solution to many of the pain points discussed in the earlier questions. This question is aimed at understanding where an institutional project like RADIUSS can help, and where we maybe need to be thinking about other strategic directions beyond RADIUSS.

Analysis:

Common answers include:

- Help with porting to GPUs
- Common tools across the institution with long-term commitment to provide them
- Stable funding
- Help with staffing (providing, recruiting, identifying)
- Developer training – new skills

RADIUSS can help with the first two bullets – porting to GPUs using the software developed as part of the RAJA Portability Suite has already helped several applications get onto the GPU-based systems at LLNL. RADIUSS does not currently provide long-term funding to projects but can provide some short-term funding and resources to help integrate LLNL open source into applications and provide support for those new dependencies for the long-term. Staffing remains an issue, as the lab does not maintain a "pool" of software developers designed to quickly and easily move between projects on an as-needed basis. However, LLNL's matrix organization and the Computing directorate in particular can provide needed expertise and hiring if funding looks stable.

Have someone available to port algorithms to GPU
More support
Invest in additional hardware for testing, especially on heterogeneous systems. The need is currently met (sort of) but will become short once higher percentage of our application runs on GPUs.
Have more frequent communications
Help with testing. Help with finding people with the right skills that are available to join the team.
I have to maintain my own Jenkins CD pipeline right now but having build agents available for a variety of OSes would be very nice.
Providing a bit more support from OpenMP "gurus" would help. Tom Scogland helps some, but he has limited time set aside for that purpose. There are times when it would be good to have him look directly at coding in pF3D and the pf3d kernels.
Reduce overheads - Staff is very expensive which reduces effort levels Part time developers - Difficult to get developers for 25% time allocations or limited time efforts (3-6 months). Small projects have a hard time staffing efforts.
LLNL folks have been really great. Interested parties in the program have been very helpful offering funding and/or personnel to make changes necessary for next-gen platforms.
If programs continue to depend on this code, I may need help with conversion. The speed of conversion with help from ASQ members has been amazingly fast for gmlib and so it would be a better use of time to get help than do it myself.
* Make sure on-line documentation on systems and software is comprehensive, accessible and up-to-date * Ideally, provide on-site computer scientist to serve as point-of-contact with decent fraction of FT (say: 0.2) to help adapting the code to LLNL systems as they are evolving (on the model of INCITE projects at ORNL for instance) * Please keep supporting Fortran for a few more years at least...
Continue the ISCP funding that allows us to maintain and mature our testing and build systems. Provide access to expertise in developer documentation development and web-based documentation systems. Develop and propagate code development standards (to help improve design of application codes that will allow them to more readily work with third party libraries). Provide software carpentry training at LLNL (especially to our app developers - but all of us could benefit). Continue and improve valuing code DevOps in performance reviews. Continue and improve valuing interfacing work between libraries and applications - work that falls in neither the application nor the math/CS side but is necessary for effective use of libraries.
Maintain proper funding.
LLNL can potentially help by offering classes on productivity tools and good coding practices. For example, while unit testing sounds like a good practice, it requires one of us to start from scratch and potentially make numerous mistakes while trying to implement it in a part of the code. Having some good guidelines would definitely help. Similar to the training sessions that happened when the HPC architecture changed to GPU-accelerated nodes, this would be really beneficial to most of the people developing this suite of codes.
LC's uneven willingness to install new packages, new versions, multiple versions of packages.
The main code developers are in academia and we (LLNL) are contributors. LLNL can make sure/assist LLNL developers that the code can run on any new LC platforms.
It is kind of late now, but open-source release on this one was challenging. Took way longer than desired and had to involve significant engagement from management. Someone not as well-connected might have given up or failed to get the release to go through. In general, LLNL could improve the release process. The deal wherein utility/library software got a somewhat streamlined process left out 'application' codes.
LLNL just need to continue to provide easy abstraction layers any new accelerators that come available through RAJA. We could probably work with the SPACK team members at LLNL to come up with SPACK packages for both ExaConstit and ExaCMech to make the entire build process easier for our end-users.
Training classes for developing and using simulation software could potentially help. We have several classes to talk about background for our user's application space, but less specifically about simulation. Developing and encouraging people to follow some guidelines on library development would probably help. Simple things like versioning, backward compatibility and interface changes would be helpful if they were more consistent for people who have to deal with a lot of libraries. Additional people would always be nice, although everyone is stretched pretty thin.

I'm not sure. I need to find time to write up the needed documentation, but that's on me.
provide funding/support
- Additional funding opportunities for maintaining codebase. - Additional s/w development staff.
Better transition support for major API changes
Long term stable realizes (or stable interfaces) for LLNL dependencies
Provide a more extensive suite of precompiled libraries on LC. Efforts like RADIUSS are also useful. At the end of the day, I suspect most codes written at the lab have infrastructure capabilities that are 90% identical, while 10% is dedicated to specific physics or algorithms. To the extent we can provide common infrastructure for mesh handling, discretization, memory management, portability, I/O, visualization, linear and nonlinear solvers, etc. it will be a huge help.
Access to a docker swarm, with Linux, Windows, and Mac hosts, for testing and building releasable executables and packages
We will need a professional staff with computer science training for long-term maintenance of your code
Supporting Spack earlier would have helped a lot. Continuing excellent procurement is essential, especially if we can guarantee the machine is not the last of its kind like Sequoia.
Maintain some targeted / limited institutional support for software maintenance tasks. Improve support for deployment of software on cloud resources for broader community use.
help with direction for the next generation computing
Not sure – being such a large multi-lab project, software solutions are a bit of a political issue. LLNL leads the broad direction of the project, but Sandia, Argonne, LANL, and to some extent ORNL drive the computer science.
Not really sure. Hire good developers. Continue to maintain good software development environments and tools on LC.
Documentation standards and templates and supporting tools. Easier to use CI systems for HPC.
Documentation standards
COE is helpful, particularly information about El Capitan and Sierra issues. Help with compiler and performance issues from vendors and LC people.
Good documentation and a transition period when an old tool is replaced with a newer and better tool.
Help perhaps with adapting our existing build system (based on scons) and testing system (we rolled our own) to the new lab-blessed technologies. Help with porting to GPUs somehow. Fortran-based "prettify" routines (don't know if those exist, haven't looked yet).
Help with continuous integration and test coverage. Support ongoing efforts to address code portability and I/O issues using tools such as conduit and Axom.
1) Actual training on technologies (rather than short/half day demonstrations), also oriented to just not computer scientistic. 2) Providing development-oriented login nodes (persistent connections, graphical connections).
Offer tutorials and/or assistance with automatic testing and continuous integration tools.
Providing a small but stable funding support to maintain and update the code to better meet the program needs.
Provide access to Mac OS systems with the latest versions of the operating system. Access to Microsoft Azure for testing.
Support to increase staffing, which we do currently have from WSC. Divisional support in recruiting staff, which I also feel we have.
My biggest pain point is JIRA. Since the last update in March, it takes me freaking 6 clicks per action to get through the system. I've been working with Neil O'Neill, but aside for telling me to use a fresh browser every day in a private window, I have had no success. Eventually I get the action completed, but man am I frustrated!! Before the update - it worked perfectly. So my Atlassian tools. Also, Bamboo. We could really use GitLab or something like that to have more build agents. That really slows down development because we really did write a ton of tests.
1) institutionally supported GUI, input deck format, parser, etc. The fact that every code has its own input deck format and parser is absurd. 2) institutionally supported regression test software. Not overly complicated, keep it simple, a non-CS major should be able to use it. 4) COMP needs to hire people and create a pool (a "bench") of practitioners who can help new software projects get off the ground. Example 1: I need an input deck and parser for my code, COMP's solution is to hire an external no-experience full time recent graduate for me, which will take a year. I only need the person for a few months! This is not optimal. Example 2: My project needs an automated test suite, COMP's solution is to hire an external no-experience full time recent graduate for me, which will take a year. I only need the person for a few months! This is not optimal. SOLUTION: COMP needs to hire a pool a programmers who jump from project to project, with 6-month maximum term with any project. If we don't have a pool of available experienced programmers the Lab should halt all WFO proposals, HPC for energy/manufacturing proposals, LDRD proposals, etc. there is nobody available to sling code.

Our current problem is not funding but access to good developers with the right expertise who are available to jump in and help implement a feature over a short time span (3-6 months). We also sometimes lack expertise; e.g., no one on the team knows HIP and we have very limited CUDA expertise.
(1) Ensure every new platform has robust, well-optimized core libraries as soon as possible: MPI, OpenMP, BLAS, LAPACK, ScaLAPACK, FFTW. (2) Provide expertise to help with compiling and linking with all core libraries as new platforms come online.
Ensure that all core libraries on new platforms are robust and well optimized as soon as possible: MPI, OpenMP, GPU libs, BLAS, LAPACK, ScaLAPACK, FFTW.
Encourage more publications. (Now, you might get a gold star on your PA if you write something. It should be that you get a demerit if you don't.) Encourage the team's plans to be realistically scoped, factoring in things like user support and paying down technical debt. Somehow allow this work to be recognized as important.
More funding or effort for centralizing this type of capability. Currently, there are multiple groups/programs leveraging this kind of tech.
Most of our pain points come from design choices and the previous need to quickly have calculations done in time for LDRD results. The fast approach is often not the best strategy for user-friendliness or maintainability. Our pain points don't really come from anything related to LLNL.
Maintain a pool of software developers various projects can use
Advocate with language standard committees and compiler vendors on behalf of all code teams at LLNL.
Help us transition this physics to the shared code base; make sure we keep this code going while this transition occurs with as small of a support staff as possible. This code should be a key app for El Capitan. We will need help porting and testing. We are already behind on this effort. Help us with recruiting and hiring talented staff Recognize how important this code is for LLNL success on El Capitan -- I think this is the essential piece of our El Capitan story
We have zero support in the RAJA team to address performance (which is quite odd). I think it would go a LONG way to invest in a "performance guy" that would try to address performance issues inside of RAJA, and performance issues in codes that use RAJA.
Provide docker images that are identical to LC platforms. Allow offsite services such as TravisCI to run jobs on LC systems.
Ensure procurement contracts include vendor support of latest compiler standards. If the cloud is linked to the RZ gitlab, it could be useful in testing commits across a wide variety of platforms. Spack will help with this.
I think the adoption of common tools (Git, Confluence, the Atlassian toolset in general has been a big help). The adoption of common tools has eliminated many of those one-off tools.
1) Provide a pool of competent maintainers 2) Provide funding for them
Provide a lecture or training for software developers whose expertise is close to other sciences (e.g., mathematics, physics) instead of computing science and software engineering.
We have a lot of features planned that we are really excited about providing to our users and to the program. At the same time, all of our developers have a lot on their plates. We feel that we would be able to deliver more of these features to our users and to the program if we had a larger team. This goes for the Axom project as well (Axom, BLT, conduit, shroud, raja, umpire). For our critical dependencies, LLNL should ensure that the expertise is spread over a team, i.e., it is not 1-deep. Specifically, management should invest in having the project lead mentor new developers for their libraries. This effort on the project lead's side should be rewarded. Compatible releases of the RADIUSS stack, i.e., versions that are tested together and are compatible will likely help with dependency conflicts (see 5-year challenges below), especially as the individual RADIUSS projects mature/stabilize. Obviously, this will not solve all problems due to the perennial need for bugfixes/features from individual packages. On the developer side, consistent documentation / training /build processes for RADIUSS packages will help developers understand / work effectively with the modular CS packages.

Question 39: What *can't* LLNL do to mitigate your pain points?

Basis: Perhaps a difficult question to answer, but this was meant to get a feeling for what application teams feel the institution can't solve for them.

Analysis:

Common answers include:

- Write their documentation
- Control software directions of things not developed here
- Provide funding to every project that needs/wants it

Address existing technical debt and cultural difficulties - that's on us. But they can get behind us when we make a pitch to devote a large amount of time to address that technical debt.
I don't think we have people to write documentation for me or twist my funder's arms to cough up the dough to pay for that. :)
Rewrite my code...
Provide unlimited funding to our project. :)
Develop software that is platform independent.
The reason behind almost all of these pains is the fact that we tend to focus on science and not writing good code. This basically means that people who contribute to the code base will do whatever is needed to obtain the scientific result and documentation is deferred to the future indefinitely. LLNL cannot reasonably require (or fund) people to write code documentation or refactoring code.
Suppress LC machines without making sure our code can run on new installation.
LLNL really can't help us on the documentation front of things.
There's only so much background an institution can give before you really have to dive into the details of a code, and no amount of training is going to take someone all the way there. Also, given that our job is to create these massive, very capable codes, there's a limit to how far down you can take the complexity, even with infinite resources.
Any influence on outside dependencies
There's a lot LLNL can't do since other labs lead the computer science aspects of <our code>.
Clone key developer's brains.
Make for time to do the supporting stuff.
Micromanage their solutions and approaches to overcome
LLNL "can't" stop external freight trains such as the advent of CMake.
Re-write the whole thing in C++.
maintain my code for me :-)
The complexity and learning curve of <our code> is baked in.
No direct funding support to maintain and update the code.
Stop the Corona virus. Seriously, we don't have any spare cycles to work on performance. So that is a future pain point when we are embarrassed that things are slow.
All it takes is money and will....
Limit its machine selections to those that coding approaches currently in widespread use at LLNL.
slowdown the requirements: Next Gen codes, El Capitan
We can do >>anything<<, is this a trick question? Oh, I guess it all comes down to funding. We don't have unlimited funds.
A lot of the above pain points are intrinsic to developing a large complex modular codebase. Turning the question around -- there are a lot of issues that are not pain points for us because of the great work that LLNL is already doing. For example, we really appreciate the efforts of LC in providing us with consistent dependable environments on the various platforms and in providing us with great tools, like Spot and Caliper to understand our code’s performance!

Question 40: WHAT ARE THE BIGGEST SOFTWARE DEVELOPMENT CHALLENGES YOU'RE FACING IN THE NEXT YEAR?

Basis: Similar to earlier free-form questions, this was aimed at understanding the shorter-term tactical demands facing software applications right now

Analysis:
Common answers include:

- Porting and optimizing for GPUs
- General code maintenance – removing technical debt
- Not enough developers or time to do what's needed

designing algorithms
Open MP
Preparing for El Capitan.
Including Hypre solvers to replace the existing solvers.
If we can beat the memory bandwidth of the current NVIDIA GPU is not clear. Note: **CUDA** FFT runs 1/10 of the peak flops with which I have little hope hitting good performance.
Working with MFEM-based applications to help them achieve significant performance on GPU architectures.
Much of my current computation, specifically N-dim optimizations for model fitting, are (speed) bounded by certain aspects of the way the domain problems are formulated. To fix this will require sitting down with SMEs for the domain and re-formulating the approaches they have been using for 20+ years in some fashion. Also, my test harnesses need considerable attention for coverage.
Accelerator (GPU) development is coming from university contributors which is causing issues since university students lack a long-term perspective and are focused on near term performance hacks. We also have multiple universities doing incompatible changes.
bring model developers up to speed on how to put in new models given that it is being converted to support GPU offloads.
Funding for code clean-up and performance optimization (as opposed to funding that covers initial "physics" development). This should be one and the same and good software development design and planning should do it, but in practice this is very hard with the way funding comes in.
Developing a prototype capable of fully exploiting new GPU-based architectures
Lack of developer time
Supporting uncertain architectures and programming environments. Supporting users who are migrating to new architectures. Continuing to encourage new application users to adopt our library.
Having funding to port code to GPUs
In the next year we have to further extend the code suite's usage of GPUs. We have laid out the steps necessary and we will be performing extensive benchmarking and profiling to get the best performance. Furthermore, switching from CUDA to HIP will also be done in the next couple of years.
Ability to account for big data, which are used for our inverse problems.
Using GPU kernels
Keeping code base integrated and maintainable while we explore a partial rewrite for vectorization down the call stack, motivated in part by GPU performance considerations.
Our biggest software challenges in the next year will be writing more performant compute kernels to run on the GPU. We'll also need to start testing and profiling our performance on AMD hardware to prepare for Frontier and El Capitan.
Keeping all the development moving forward without incurred too much additional technical debt. We have a lot of features requested and have to get moving onto the CORAL-2 preparation. Additionally, we have a lot of users to support. I don't think this is much different than most years since we moved into the accelerator era though.
Probably deployment to GPU's and any other upcoming hardware changes. We also need to transition to Python 3, which will affect both developers and users since Python is the code interface.
updating all dependencies
Machine heterogeneity and poor programming models and tools for them. In particular, GPUs are a huge disruption with complex programming models (especially memory models) and little consensus on which approaches to use, making interoperability of libraries an extremely challenging task.
In general, many softwares and tools that I develop via research funds tend to die down due to a lack of funding and time for maintaining, continuing, and/or promoting the tool.

N/A - no development for this code, only maintenance.
Adding in new features
Platform portability.
Time
Make code work on GPU efficiently
Porting away from our closed-source CAD kernel.
GPU offload
We will need to be re-engineer some of the software to better scale to the increasing size of data that needs to be processed.
Performance portability and lack of adequate solutions.
Java versions release cycle is significantly faster than just a couple of years ago.
compile the code for new machines
Getting <our code> running end-to-end on GPUs is the biggest challenge, but we are in the middle of a complete rewrite to do that (using Kokkos). Better code testing coverage is always on my mind as well but is hard to get funding for. We are currently using a junky/donated testbed for continuous integration testing and it would be nice to have a more reliable machine... but we'll probably just buy our own with project funds and put it at Argonne or PNNL.
Refactoring what started as a research code to enable further development and better use as a production code by our physics collaborators.
Standardizing ways of starting up multinode jobs in different environments from HPC to AWS to a small local cluster.
Integrating modelica model libraries
Porting more packages to GPUs, improving GPU performance.
Creating a performant implementation for AMD GPUs.
GPU's. Dealing with a much larger user and development base as the code grows
I'm adopting a LBNL developed code (WARPX) that will run on GPU machines. Porting my own code will be too painful.
We are bringing our GPU support for umpire/RAJA into our mainstream releases. I anticipate as this is adopted by users there will be much feedback to address regarding ease of use and performance bottlenecks.
Time, software development for PLS is patchy and gives short periods of funding. Software development is done "along the way" with research, resulting in low research productivity and bad quality software.
Coming up with a portable solution for the code to run efficiently on El Capitan. Doing this with a single source for the pieces common to Cretin and the rad-hydro codes is currently impossible.
Adapting the code to the GPU platforms.
Continuing to port to new architectures (CUDA, AMD)
We are continuing to develop and evolve our software base to achieve improved performance on GPUs. At the same time, we are working to deploy new physics capabilities into production. Balancing these two goals is an on-going challenge.
I think it is really going quite well, but this last update to the Atlassian tools is killing me. Also, we tend to get our interns laptops which aren't full development machines. That really slows down progress. I am not taking an intern this summer but for the next summer I do take, I plan to try to get more help here.
1) personnel (we have two new hires, both recent college grads, which means they will need a lot of hand-holding) 2) we want to open-source (part of...) our code, which means we need documentation, an easy-to-use build system, a functional test suite. 3) our code is implicit, we user hypre, so until hypre runs efficiently on GPU's we can't use the new big iron.
Completing CUDA support and porting zfp to HIP while ensuring reasonable performance. Part of this also means writing a large number of unit tests and making sure the code is well tested on GPUs (perhaps via GitLab CI).
Porting to new platforms and incorporating GPU support.
Incorporation of GPU support.
WSC's new focus on modularity means a lot more meetings and coordination at a bigger scale, and less time doing work.
Hiring/training and retaining people with skill sets needed for this type of work, especially when some may require Q or TS. Some idea of skills needed: DevOps, Security with focus on container tech, Data Management Engineers/Developers
A novel method involving tensor contractions needs to be ported over to this code.
lack of programming manpower
Migration to GPU architectures since it requires visiting nearly every single routine in the entire program.
Making progress on new testing on Sierra, finding a fixing complicated OMP bugs on Sierra, implementing Umpire, utilizing Conduit in a reasonable way and setting appropriate requirements for Conduit

We aren't 100% ported to RAJA. This is a major factor going forward, especially for 1D and 2D problems running on GPUs. I think a major challenge for us is to understand/improve our performance on GPU architectures so we have a better opinion of more broadly adopting GPU architectures (such as CTS2 or beyond). Adopting asynchronous models in RAJA/CHAI is going to have a large reward but will be a very challenging software engineering problem.
Linear solver packages on GPU.
Design and restructure <code> to support more CPU and GPU hardware in a generic way. Port more potential model to Sierra/Lassen.
El Capitan technology changes. Lack of manpower slowing development of updated tools for new platforms. Host code demands on library as early adopters.
Engaging with a wider set of potential users and with that comes with new use cases along with deploying/implementing new tools that replace the existing capabilities is going to be the biggest challenge this current fiscal year that will continue into the next fiscal year.
1) Finding more people that can implement new algorithms in the code. The basic problem is that most computer scientists do not understand how to translate the math into an algorithm that can be coded. 2) Efficient saving and retrieval of checkpoint data.
We are planning to release the developed software to public as open source. It should be challenging to track issues and bugs and to respond to user demand.
Porting our codebase to BlueOS in a manner that will be compatible with El Capitan. Using RAJA and Umpire and MFEM's portability abstraction layers have been instrumental to this process and it is now mostly a matter of working on the code and improving the memory movement. Now that we've wrapped most of our kernels in MFEM/RAJA macros, we need to start thinking about effective, readable and maintainable abstractions that are catered specifically to high order hydro loops, which can have complex triply, or quadropoly nested loops (per-element, per-material, per-quadrature, per-species), possibly with interspersed code between kernel calls at each level. The MFEM/CEED team has been thinking about this, but it would be nice to build support for these abstractions into RAJA. Improved GPU support from our third-party libraries, e.g., MFEM, hypre, Axom. In particular, GPU-accelerated AMG matrices.

Question 41: WHAT ARE THE BIGGEST SOFTWARE DEVELOPMENT CHALLENGES YOU'RE FACING IN THE NEXT FIVE YEARS?

Basis: Similar to the previous question but attempting to get projects to think beyond the immediate needs.

Analysis:

Common answers include:

- Porting/optimizing to GPUs and new architectures
- Sustained funding to keep software robust
- Scaling improvements

porting to GPU's if I find there are no CPU's available at some point
GPU
Fully fleshing out heterogeneous computing paradigms throughout our code base while also investing in new features and addressing technical debt.
Use the parallel programming to efficiently harness the heterogeneous architecture i.e., combining MPI with GPU programming.
if our code does not perform well on El Capitan, I'm not sure DOE BES will continue funding us.
Performance on AMD and Intel GPUs.
<code> is a GUI based application and very nearly literally every GUI framework in existence is being abandoned for web-based tech right now. I will have to overhaul the tool to move with the times as updates eventually will stop happening for the thick client libraries we are using.

Software Lead Backup/Transition - Need to find/train a backup or potentially new software lead for the project. Maintaining funding for SE activities - Currently have single source of funding for the software engineering efforts on the project; maintaining the limited support is crucial for development.
Some level of consistent programmatic support to cover new model development, bug/feature/manual upkeep.
New architectures.
Developing a production-ready version of <our code> capable of fully exploiting new GPU-based architectures
if not converted to C++ it may become obsolete on new machines.
Hardware platform evolution
Supporting a variety of architectures (3 different GPU vendors, for instance). Users are likely to change the way they are using newer GPU-based systems causing reworking of interfaces to our code and changing needs for how to use our codes. Continuing to encourage new application users to adopt our library.
Having funding to port code to GPUs
Re-structuring the codes to take advantage of Frontier/El Capitan-like architectures will be the biggest challenge. Parts of the code that are for now left to run on CPUs will need to be reworked from scratch in order to perform calculations efficiently.
Balancing demands for performance and features/flexibility, while trying to modernize the code base.
If we end up needing to go to a vectorized call stack for performance, then that will be a significant challenge.
Since our codebase is rather young, it's hard to tell. However, it's most likely going to be related to optimizing our code in order to take full advantage of the new CORAL2 machines. Also, we'll probably need to convert several of our python post-processing scripts over to C++, so they can be run on the fly with the simulations.
Getting our entire code into a usable state for the GPU clusters. If GPUs can't cover everything, then we need to determine a strategy on how to best use our various platforms to get their work done with our code.
Programming machines with ever increasing heterogeneity will not be manageable with current approaches.
Staying abreast with the latest s/w technology can often a challenge for folks with primary focus on research. Having access to new skills (e.g., courses) and staff (e.g., s/w developers) is essential to continue.
Generalizing the code for new science applications and porting to other machines/architectures and across machines.
Platform portability. Maintaining a funding base to support code maintenance and new development, not just application runs.
Time
Make <our code> code working on exascale computers
Possibly dealing with dependency and API issues as we simultaneously split into libraries and integrate those into codes.
GPU offload
Increasing access to broader user community.
Potential for more radical changes in hardware/software.
Evolution of Software is moving more rapidly, example UI components being obsoleted, and we are forced to 'redo' to keep existing functionality.
improving multi-threading, GPU computing
<code> places a big bet on very high resolution in order to make good use of upcoming exascale computers, but CFL restrictions leave us with short timesteps and resultingly short simulations despite being able to parallelize our grid. Parallel-in-time, implicit solves, and other tricks for efficient timestepping will be valuable. Additionally, we will be running at a large variety of resolutions, so online regridding of input data (rather than having separate input data for each resolution) will be important.
Incorporating GPUs
Scaling to millions of independent processes coordinating together.   So, figuring out how to startup and locate of HPC millions of different process that are different sizes, with different communication patterns, and connectivity.   And do this in a fashion that is portable to different machines at LLNL and other national Labs
Making the software more usable and flexible.
Porting to El Capitan. New algorithm developments.
Creating a performant implementation for AMD GPUs.
GPU's. Trying to absorb Conduit, AXOM, MFEM technologies where possible.
Learning to implement a code that could be relatively simple like massively parallel PIC, on a GPU.

Iterating on the GPU support will be a multi-year effort. Addressing performance issues that will be discovered as applications exercise our functionality at large scale will be a main driver of our development. I anticipate significant revisions to our parallel communication model and our support for load balancing.
Moving architectures. Fragmentation of vendor specific tools. Lack of good quality C++ libraries.
Modernizing the code - which is >30 years old - while making sure it remains efficient and portable to the new architectures. We also need to move to more robust and flexible automatic testing which can cover the various output options and formats.
Adapting the code to the new HPC platforms or infrastructures.
Continuing to port to new architectures (CUDA, AMD).
We are continuing to develop and evolve our software base to achieve improved performance on GPUs. We will also be tackling the same performance portability challenges as other projects related to Sierra, El Capitan, CTS machines, etc. Finally, deploying new physics capabilities results in additional customers to support.
Back to common environments and tools. Worried about transition to gitlab, but I think it will be worth it if I get more build agents.
1) the only technical challenge is the trend toward GPUs, most engineering codes are implicit and unstructured and need unstructured linear solvers, and these solvers don't yet run on GPU's. 2) Other challenges are more personnel and management related. COMP needs to hire people, creating a pool of programmers who work on short-term assignments, who have *experience* with HPC and software development best practices.
Developing and implementing new algorithms to fully leverage exascale resources.
El Capitan. Code transitions.
Working with security and getting this approved on classified network(s).
Maintenance and portability to AMD GPUs.
Lack of programming manpower
Incorporating "institutional" developed software that is not essential to our code's functionality but allows linking and limited commonality with other LLNL codes. As a Fortran and engineering codes, these packages are difficult to utilize as they were developed primarily for C++ with data organized differently.
Transition all of this to a new framework and recover the same level of performance in a C++ RAJA +MPI code base compared to Fortran + OMP +MPI
Getting our code "stable". We have a software architecture we are "moving" towards, but we have a lot of refactoring still to go. It will be a great challenge to get our code there, while still working on new features/physics and porting our code to El Capitan.
Solving linear systems on GPU.
Support a more diverse set of accelerator environments.
El Capitan technology changes. Host code demands on library as early adopters. Likely retirement of most of the development team.
El Capitan. The move to exascale computing and in my arena the move to exascale class ensembles and UQ studies is going to be the biggest challenge.
Some of the main developers are reaching retirement age.
Large-scale simulation fitting in the next-generation exascale-performance computing. Dealing with the large data throughput and fitting to exascale environment (e.g., updated or new libraries) should be challenging.
Managing the complexity of our codebase as our project’s scope continues to grow. Managing code development time with user support time as users begin to adopt our code. Paying down technical debt that has accrued during our code's rapid development over the past few years Interdependent modularized packages: Modularity has brought many benefits, but it has increased the interdependence of these packages. For instance, we’ve run into cases where an apparently simple change was blocked due to conflicts between dependencies in other packages, which required incompatible changes to other packages. As an example: I needed to update M**ARBL**’s Axom, which seemed like a trivial change. However, this had a change related to semantics of zero-sized allocations, causing an incompatibility between Umpire and Conduit. In the end, we were able to get bugfixes into Axom and Umpire, improving both codebases, but this process took about 2-3 weeks of careful coordination. I suspect some, but certainly not all of these issues will reduce as the individual modular libraries mature.

Section 7 List of Codes and Responders

Application/ Library Name	Responder POC	Description
ACE/ACES	Ted Stirm	Provides compute environment for running mixed workloads leveraging containers(AI/ML, Database, Web apps, Batch jobs...etc.) in the same cluster leveraging Kubernetes.
AEOLUS	Akshay Gowardhan	Computation fluid dynamics model to simulate flow and dispersion in urban areas and complex terrain
ALE3D	Peter Robinson	multi-material hydrodynamics.
Ardra/Armus/ [Radar]	Adam Kunen	Discrete ordinates [DO or SN] solution of neutral particle transport. Ardra handles neutron and gamma ray physics. Armus is a framework/library that provides general SN data structures and linear solvers (such as transport sweep algorithms). Radar is a to-be-developed thermal radiative transfer (TRT) code that is planned to be developed on the Armus framework.
Ares	Brian Ryujin	Ares is a multiphysics code that has a rad-hydro core. It is used to model experiments, including high explosives, ICF, and pulsed power. Each package makes its own decisions about what methods and algorithms to use, so there's a large range.
BOUT++ code suite	Xueqiao Xu	BOUT++ is a modern, highly adaptable, object-oriented C++ code for performing sophisticated 3D plasma and fluid simulations.
CCT	Justin Barno	Coda Calibration Tool (CCT) is an application for calibrating 1D shear wave coda seismic measurement models to observed data using a much smaller set of reference calculations from other means (waveform modeling, etc.).
COGENT	Milo Dorr	COGENT simulates the edge plasma region of a tokamak fusion reactor. It utilizes a high-order discretization of a continuum gyrokinetic model of multiple plasma species together with a gyrokinetic Poisson model of the electrostatic potential. Multiple collision models are also implemented.
Cretin	Howard Scott	Simulation code for non-LTE atomic kinetics and radiation transfer. The atomic kinetics is also embedded within rad-hydro codes (under the name DCA) to provide non-LTE capability.
ddcMD	Jim Glosli	problems: material science, plasma physics, biological systems. Methods: Molecular Dynamics
DFTNESS	Nicolas Schunck	DFTNESS contains 2 kernels called HFBTHO and HFODD that both solve the Hartree-Fock-Bogoliubov (HFB) equation. This is a non-linear, dense, complex Hermitian eigenvalue problem with dimensions of the order of 5000 which is solved iteratively.
Diablo	Jerome Solberg	Multiphysics, primarily solid mechanics and heat transfer, with an emphasis on implicit methods (e.g., requiring a linear solve at each step)
Eiger and EMSolve	Joe Koning	These two codes are used to solve Electromagnetic simulations. Eiger is an MPI parallel frequency-domain solver and EMSolve is a MPI parallel time and frequency domain collection of codes (all wave simulations).
ElAc	Keehoon Kim	It solves coupled seismacoustic wave propagation in the solid Earth and atmosphere. A high-order finite difference method is used.
(E3SM) (SCREAM)	Peter Caldwell	E3SM (Energy Exascale Earth System Model) is a global climate model, including representations of the atmosphere (think weather forecasting model), land, ocean, and sea ice. SCREAM is an atmosphere model aimed at extremely high global resolution.
ExaCMech	Nathan Barton	See https://github.com/LLNL/ExaCMech ECMech is a GPU-friendly library of constitutive models, so far mostly of the crystal-mechanics-based variety. Algorithms are based in part on solving small-dimensional (5D or so) systems of strongly non-linear equations.
ExaConstit	Robert Carson	The purpose of this code app is to determine bulk constitutive properties of metals. This is a nonlinear quasi-static, implicit solid mechanics code built on the MFEM library based on an updated Lagrangian formulation (velocity based). It makes use of standard FEM techniques to discretize the PDEs, a newton-Raphson solver to deal with the non-

		linearities of the PDEs, and then Krylov solvers for solving the linearized system of equations.
FEusion	Mike Puso	The library handles the interaction of overlapping grids. The primary application is the coupling of a Lagrangian solid in the form a finite element mesh or SPH particle discretization to an ALE background mesh in the form of a fluid or solid. The library also handles the coupling of a Lagrange FE model in ParaDyn with an ALE3D model.
FRESCOX	Ian Thompson	DIRECT REACTION PROBLEMS IN NUCLEAR PHYSICS. Uses coupled-channels method, for local and non-local interactions
FUSION (Fundamental Unified Structure and Interactions of Nuclei)	Sofia Quaglioni	The FUSION code package performs first-principle calculations of the wave function of protons and neutrons bound inside a nucleus as well as in a low-energy nuclear reaction. It is used to model the properties of light nuclei (ground state, size, spectrum of energy levels, etc.) and their reaction observables (cross sections). Most of the algorithms used are specifically tailored to the adopted physics model, with the exception of the Lanczos algorithm which is used to obtain the extremal eigenvalues and eigenvectors of large sparse matrices.
Geocentric	Joshua White	Single and multiphase poromechanics. Finite element and finite volume discretization, implicit timestepping.
GEODYN	Efrem Vitali	Simulation code to solve shock physics /continuum mechanics problems Massively parallel cell-centered Godunov Eulerian method
gmlib and snowninja	Eric Herbold	gmlib – (GEODYN material library) large-deformation material models for hard rock, soils, concrete in hydrocodes snowninja - spherical and ellipsoidal particle packing using a MC algorithm in parallel
GEODYN-L, geodyn material library	Oleg Vorobiev	Lagrangian FD/FE hydrocode for explicit modeling of wave propagation in discontinuous media caused by explosions and impact. Works with various material libraries (geodyn, geodyn-l), models contacts with common planes and embedded discontinuities, has some remap capabilities from Eulerian solvers to handle large deformations
GEOSX	Randy Settgast	Equation of motion, Compositional multiphase flow, Fracture Mechanics, Coupled physics
GridDyn	Philip Top	Open-Source power system dynamic simulation
HELICS	Philip Top	Co-simulation, allowing multiple simulation software packages to interact in a coordinated time synchronous fashion.
HiOp	Cosmin Petra	HiOp solves mathematical optimization problems, also known as mathematical programming problems or nonlinear optimization problems. It implements interior-point methods and dedicated HPC linear algebra kernels specialized to the computational nature of the underlying optimization problem.
HYDRA	Marty Marinak	HYDRA is a multiphysics 2D/3D radiation hydrodyanmics code that is used to simulate a broad range of ICF and HED targets.
hypre	Rob Falgout	Parallel library of multigrid linear solvers
Integrated Yield Determination Tool	Douglas Knapp	Inverts for Nuclear Explosion Yield using Seismic and Airblast signals collected at local distances. The algorithm was developed by LLNL Research Staff. This not a HPC application.
K2	Mark Sherlock	A vlasov-fokker-planck code coupled to MHD and ion vlasov with a variety of electromagnetic field solvers and extra physics packages such as EoS and ray tracing. This is used to study transport processes relevant to laser plasma interactions and ICF.
Kull	Tom Brunner	Kull is a large, multi-physics code for simulating ICF capsules and other HED experiments.

lalibe	Arjun Gambhir	Lalibe is a suite built on top of the USQCD/Chroma software stack. Its purpose is to compute hadronic observables through the method of Lattice QCD. It contains a collection of novel algorithms/features including the Feynman-Hellmann method, hierarchical probing, sequential sources with coherent sinks, and full hdf5 support.
LEOS	Dale Slone	The Livermore Equation of State (LEOS) Project provides EOS data, access software, and tools for LLNL applications that require accurate global material-specific EOS data. This data is an essential component in hydrodynamic simulations.
Livermore Design Optimization (LiDO)	Dan White	A suite of codes for design optimization, aka generative design or PDE constrained optimization. Physics solvers (elasticity, maxwell, thermal,...) are finite element based, and these are "wrapped" with a nonlinear optimizer to optimize the topology and/or shape of a part. Some components of LiDO employ machine learning using sklearn and pytorch.
Livermore Metagenomic Analysis Toolkit (LMAT)	Jonathan Allen	Identify organisms and genes present in a biological sample using the sequenced genetic material as input.
LLNL UQ Pipeline	David Domyancic	The UQ Pipeline provides an umbrella of capabilities that include ensemble generation capabilities, sampling methods, and uncertainty quantification methods. Taken together, these capabilities provide scientists an ability to make predictions with uncertainty of untested experiments that are informed by previously completed experiments.
MARBL	Kenneth Weiss	MARBL is a next-generation multi-physics code for simulating high energy density and focused physics experiments driven by high-explosive, magnetic or laser-based energy sources. MARBL is based on high-order numerical algorithms and modular CS infrastructure (Axom project) emphasizing robustness, scalability and flexibility.
Merced	Bret Beck	Calculate multi-group transfer matrices from nuclear data as needed for nuclear transport.
Mercury and Imp	Patrick Brantley	Mercury: Monte Carlo Particle Transport Imp: Implicit Monte Carlo Thermal Photon Transport
MFEM	Tzanio Kolev	High-order finite element library
MSlib	Nathan Barton	Mechanical response (e.g., constitutive) modeling. Including strength, equation of state, and damage and failure aspects. Models typically involve iterative solve of 1D or small multidimensional systems of strongly nonlinear equations.
MuMMI	Helgi Ingolfsson	A Multiscale Machine-Learned Modeling Infrastructure, for coupling macro and micro scale simulations using machine learning and concurrent simulation execution on a massive scale.
nn-scattering, TensorPWE	Kyle Wendt	These are tools for fitting microscopic nuclear models to scattering data and electroweak observables
ns-3 contrib	Peter Barnes	LLNL developed contributions to the open-source ns-3 network simulation toolkit.
NUFT (Nonisothermal Unsaturated-Saturated Flow and Transport model)	Yue Hao	NUFT is a flexible multipurpose computer software package for modeling multiphase, multi-component heat and fluid flow and reactive transport in porous media. It solves the continuum equations for the conservation of mass and energy with using the integrated finite-difference spatial discretization and fully implicit time integration method. Several sub-modules are implemented to simulate various flow and reactive transport processes in porous media. The NUFT code is capable of running large-scale complex simulations on high-performance parallel computing systems. It has been widely used for many applications such as CO2 geologic storage, geological disposal of nuclear waste, geothermal energy exploitation, groundwater remediation, subsurface hydrocarbon production and so on.
Overlink and Carter	Jeffrey Grandy	Overlink transfers zone-centered and node-centered quantities, such as temperature, represented on one mesh, to a different, unrelated mesh using geometric interpolation. The Carter code, which is part of the Overlink project, assumes Cartesian meshes, in order to take advantage of specialized optimizations.

ParaDyn	Ed Zywicz	ParaDyn is a nonlinear, explicit, finite element code for analyzing the transient dynamic response of three- dimensional solids and structures. It is used to simulate the dynamic structural response of complex multi-body systems (e.g., watches, cars, aircraft carriers and dams) under normal operation and to short duration loadings from drop events, blasts and other excitations. It combines central difference time-integration with low order finite elements and a lumped mass idealization, and has advanced contact detection and enforcement algorithms to accurately model the interaction of multiple components.
ParFlow	Steve Smith	Parallel Flow (ParFlow) is an application for integrated hydrology modeling that simulates spatially distributed surface and subsurface flow, as well as land surface processes including evapotranspiration and snow. ParFlow simulates the hydrologic cycle from the bedrock to the top of the plant canopy, integrates three-dimensional groundwater flow with overland flow and plant processes using physically-based equations to rigorously simulate fluxes of water and energy in complex real-world systems.
pF3D	Steve Langer	pF3D simulates the coupling between a high intensity laser and a plasma. It propagates the laser through the plasma and can include coupling to SRS and SBS backscattered light waves. The light propagation algorithm assumes the light makes a small angle with the z-axis (the nominal laser propagation direction).
PMesh	Walt Nissen	Geometric setup, broadly defined. Define simulation geometry using a CAD engine, (optionally) discretize, and output block-structured, unstructured, Constructive Solid Geometry (CSG) and NURBS meshes. There is no science in PMesh.
PSC	Andreas Kemp	Particle-in-Cell FDTD Maxwell solver with Boris particle push
Qbox/Qball	Tadashi Ogitsu	Quantum mechanical electronic structure of materials and molecules based on plane wave pseudo potential density functional theory algorithm
QMCPACK	Miguel Morales-Silva	Calculates the electronic structure of atoms, surfaces and solids using stochastic approaches based on quantum Monte Carlo. The method performs stochastic sampling of high dimensional spaces (>1000) through the propagation of an ensemble of "walkers", which at long time sample the desired distribution.
SAMRAI	Noah Elliott	SAMRAI is an object-oriented C++ software library that enables structured adaptive mesh refinement (SAMR) technology in large-scale parallel application development. SAMRAI's tools can be used to incorporate SAMRAI into applications that involve coupled physics models and sophisticated numerical solution methods, and which are intended to run on large HPC systems.
SCUMM	Vincenzo Lordi	Quantum dynamics solver for superconducting qubit circuits, using Brune circuit synthesis and Langevin quantum dynamics to compute input/output dynamics for finite-element device models.
SPARC	John Pask	SPARC solves the Kohn-Sham equations of density functional theory using a real-space grid approach for massive parallelizability. A main application is large-scale quantum molecular dynamics simulations.
SPECFEM3D_GLOBE	Christina Morency	SPECFEM3D_GLOBE simulates global and regional (continental-scale) seismic wave propagation based on a spectral-element method. Effects due to lateral variations in compressional-wave speed, shear-wave speed, density, a 3D crustal model, ellipticity, topography and bathymetry, the oceans, rotation, and self-gravitation are all included. It supports CUDA and OpenCL GPU accelerators. Git versioning system (on github) to support user contributions. The code also allows the calculation of frequency dependent sensitivity kernels by adjoint method to be used for inversion problems.
Spheral	Mike Owen	Spheral is a research code for meshfree numerical methods, mostly hydrodynamics. We use it to try out a variety of existing and novel hydrodynamic methods and develop new methods. Spheral is also applied to a number of areas now, such as astrophysics, planetary defense, and modeling solid material breakup due to fracture. Spheral is currently deployed as a standalone code, as well as a library for use in larger efforts such as ALE3D.
SQDFT	John Pask	SQDFT solves the Kohn-Sham equations of density functional theory in a real-space representation using the O(N) scaling Spectral Quadrature method, especially well-suited to massive parallelization and high electronic temperatures.

SUNDIALS	Carol Woodward	SUNDIALS provides initial value problem integrators for ODE systems using multistep methods, ODE integrators with additive and multirate Runge-Kutta methods, including support for IMEX methods, an initial value problem solver for differential-algebraic equations (DAEs), ODE and DAE systems integrators including sensitivity analysis (forward and adjoint), and nonlinear algebraic systems solvers.
SW4	Anders Petersson	SW4 solves the time-dependent seismic wave propagation problem for regional 3-D domains using an advanced summation-by-parts finite difference method on structured grids with mesh refinement.
Teton	Teresa Bailey	Deterministic thermal radiative transfer, using on-node sweep algorithms on unstructured grids, utilizing Bi-CG Stab solvers for acceleration schemes, MPI + OMP on Toss, using OMP-4.5 with target offload for GPU architectures in the hopes that OMP is portable to other types of GPUs. Exploring the use of UMPIRE soon.
Topanga and KIM3D	David Larson	KIM3D is a hybrid (kinetic ions/fluid electron) plasma simulation code for modeling the early time (t<~30 sec) debris motion and generation of beta-decay electrons from a high-altitude nuclear explosion. Topanga is a modern version of KIM3D using a spherical mesh and including the motion of the neutral atmosphere, fully electromagnetic wave propagation in non-plasma regions, and ionospheric chemistry. Topanga is used to model the late-time EMP pulse (t>~3 sec-600 sec), also known as E3, generated by a high-altitude nuclear explosion.
TopoMS	Harsh Bhatia	TopoMS is a computational tool for detailed topological analysis of molecular and condensed matter systems, including the computation of atomic volumes and charges through the quantum theory of atoms in molecules (also known as Bader analysis), as well as the complete molecular graph. With roots in techniques from computational topology, and using a shared-memory parallel approach, TopoMS provides scalable, numerically robust, and topologically consistent analysis.
TransFort	Pratanu Roy	TransFort is a code for solving fluid flow (both single phase and multi-phase), heat transfer and reactive transport problems. It is currently being used to model moisture transport in polymeric/ceramic materials, which is funded by material aging and compatibility project from the WCI. TransFort uses a finite volume method to solve the transport equations. A semi-implicit pressure linked equation (SIMPLE) algorithm is used to link the pressure and velocity to solve the incompressible Navier-Stokes equations. An implicit Euler or RK-2 method is used for time-stepping. A level set method (higher order WENO with TVD RK-3) is used to capture the interface for two-phase flow simulations.
USER-EPH	Alfredo Correa Tedesco	LAMMPS extension (LAMMPS "fix") to capture electron-ion interaction and electronic stopping.
VBL++	Kathleen McCandless	VBL++ is the latest generation in a family of laser propagation codes using split-step Fourier transform optics simulation techniques. The predecessor code written in the Java programming language, VBL-Java, was itself based on the Fortran code, Prop. The name VBL++ comes from the fact that the simulation code was ported from Java to C++ to allow operation of the code on high-performance computing environments. Key features supported by VBL++ are: * Paraxial beam propagation through non-linear materials, including the Kerr effect * Frequency conversion through Type I doubler crystals and Type II tripler crystals * Non-linear Frantz-Nodvik amplification and lower-level lifetime effects.
VisIt	Eric Brugger	VisIt is a distributed, parallel visualization and analysis tool.
XBraid	Rob Falgout	Non-intrusive parallel-in-time multigrid solver library
Zero-order Reaction Kinetics (Zero-RK)	Russell Whitesides	Simulates chemically reacting systems efficiently with sparse, adaptive, preconditioned methods.
zfp	Peter Lindstrom	zfp is a compressed representation of multidimensional numerical arrays. It provides C++ compressed-array classes for random access using inline compression and a C/Fortran/Python API for compressing whole arrays.

Section 8 Recommendations for a Follow-on Survey

Developing a set of survey questions that balances simplicity with clarity is an art, and admittedly we didn't always pose our questions in a way that elicited the response we intended. Throughout this survey we identified those issues in the Analysis section for each survey question, and briefly summarize them here. The hope is that if/when a future iteration of this survey is performed, a large number of the same questions can be used so as to begin to track trends over time, while also continuously improving the questions (and adding to them) offered to responders.

Question 6: What is the release status of the code?
Remove OUO from the second option so that it's clear the choice to not open-source is not because of government policy restricting it. Add OUO to the next option instead: "OUO or Export-controlled (e.g., ITAR, EAR99).

Question 8: How do users learn about your software?
Add "workshops", "schools", "work for others", and "inclusion in other codes" as additional multiple-choice options.

Question 13-14: Does you anticipate using HPC cloud resources in the future…
If users answered Yes or Maybe, they were presented with a followup question asking about how they might plan on using the cloud. If they answer No (almost half of our responses), we should also follow up with a "Why not?" question. Potential multiple-choice options might include "No advantages over LLNL/DOE resources", "Payment model is too complex", "Don't know where to start", etc.

Question 22: Do you do Performance Analysis of your Application?
Add a new followup question to those who answer "yes" to understand what tools they use. E.g., Caliper, HPC Toolkit, OpenSpeedShop, ScoreP, Vampir, etc…

Question 23: What programming language(s) does your project use?
While they still have yet to have deep penetration in production codes at LLNL, Asynchronous Task Models such as Legion, HPX, Charm++ and others might deserve some mention by then – perhaps as generic group. Also, SYCL is emerging as a potential competitor through Intel's OneAPI push and should probably be included in a future survey.

Question 26-27: About how many 3rd party dependencies do you have… What are they?
Word the survey to exclude software that one can usually rely on being installed. Question 26 asked to include those provided by the vendor OS and used libm as an example, but that should probably be expanded to include vendor optimized math libraries like BLAS, LAPACK, ScaLAPACK. HDF5 is a tougher one, as many projects build their own version, but there are also vendors who supply it as part of the vendor software stack.

Question 27: Of the dependencies you have, what are the top 3?
Limiting the answers to just the top three likely skewed the results as larger projects with perhaps dozens of dependencies were forced to somewhat arbitrarily prioritize their top 3. Instead, leave the number of libraries they respond about as open-ended.

Question 33: How important are these factors in your decision to adopt a new dependency?
To the multiple-choice options, add "License compatibility", "Scalability", "Difficulty of implementing the functionality ourselves", "A local 'champion' to drive inclusion in our product", "Availability of

quality documentation and example usage", "Availability of a Spack package", and "Portability" – as these were all identified multiple times in the open-ended followup question #34.

New Topic areas:

ML/AI - At the time this report was written we are entering an era where machine-learning (ML) and AI are increasingly becoming important components of the application ecosystem. It would be useful to add a section to the survey that addresses how developers are approaching this paradigm shift, and how application teams taking advantage of ML/AI. E.g., Do you currently have a workflow that includes ML/AI? If yes, is it embedded in the main simulation or used for pre- or post-processing? Are your sponsors requesting it? Do you use 3rd party ML/AL apps or services; if so, which? How much FTE does your team dedicate to this expertise? What are the pain points? If not, why not? Not relevant, no resources, no expertise, no sponsor buy-in?

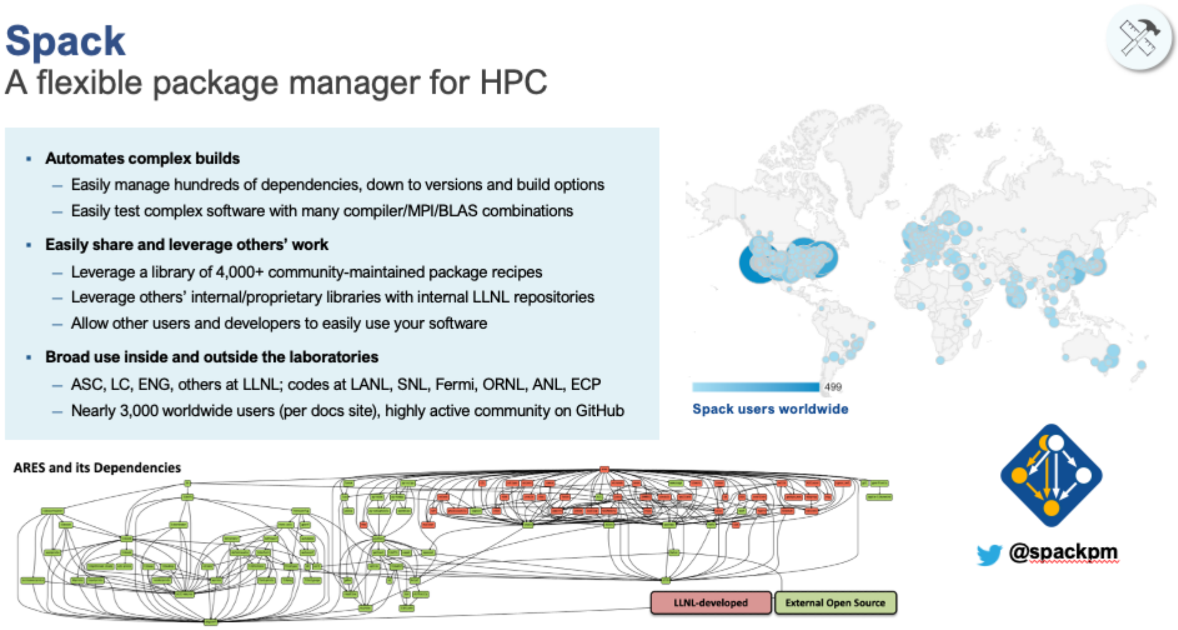
New Technology – Exploring what users are thinking about in general as future technology could be useful as a set of open-ended questions. This was explored in this current survey specifically about cloud services, but questions along the lines of "What new technology are you interested in learning more about and potentially adopting?", perhaps broken down by topic areas (development tools, I/O, programming models, databases, hardware acceleration, …) could provide some insight into what the developer community at the lab is thinking about.

Section 9 RADIUSS Software Products

The RADIUSS project aims to highlight and support existing open-source libraries and products developed at LLNL and to encourage their adoption in applications both inside and outside of the lab. Many of the sidebars throughout this document highlight particular products aimed at addressing certain issues software developers are likely to experience where LLNL open-source products can help. Here is a more complete list of products using a presentation provided at the 2020 Computing Road Show.

Build Tools

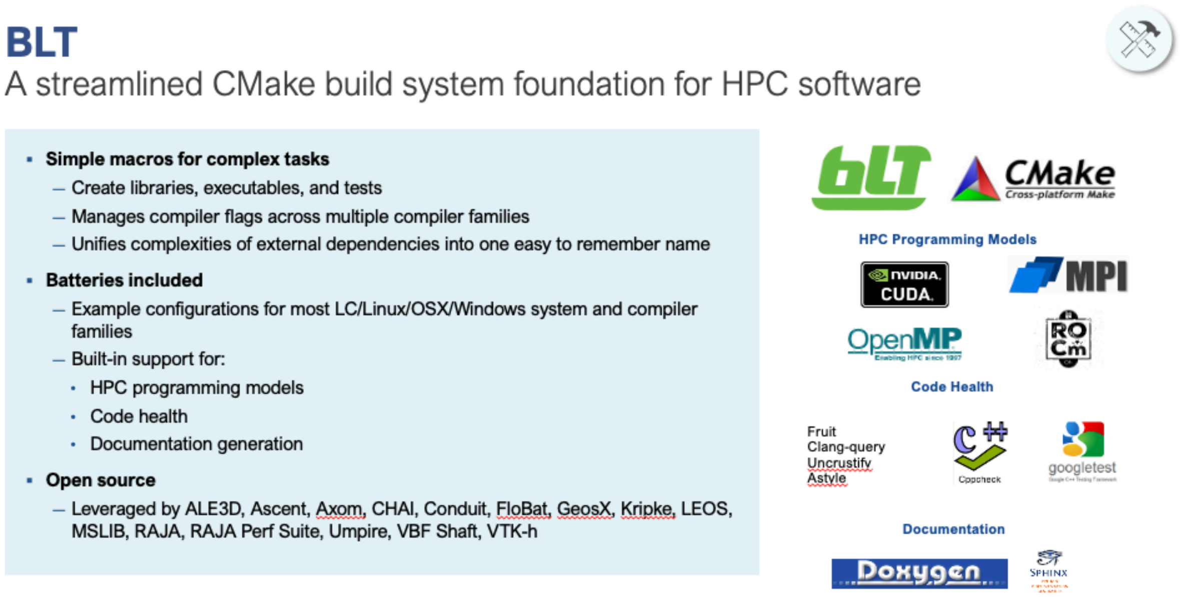
Project	Description	Website	Repository
Spack	A flexible package manager for HPC	spack.io	github.com/spack/spack
BLT	A streamlined CMake build system foundation for HPC software	llnl-blt.readthedocs.io	github.com/LLNL/blt
Shroud	Easily create Fortran, C and Python interfaces for C or C++ libraries	shroud.readthedocs.io	github.com/LLNL/shroud



Spack

A flexible package manager for HPC

- **Automates complex builds**
  - Easily manage hundreds of dependencies, down to versions and build options
  - Easily test complex software with many compiler/MPI/BLAS combinations
- **Easily share and leverage others' work**
  - Leverage a library of 4,000+ community-maintained package recipes
  - Leverage others' internal/proprietary libraries with internal LLNL repositories
  - Allow other users and developers to easily use your software
- **Broad use inside and outside the laboratories**
  - ASC, LC, ENG, others at LLNL; codes at LANL, SNL, Fermi, ORNL, ANL, ECP
  - Nearly 3,000 worldwide users (per docs site), highly active community on GitHub



BLT

A streamlined CMake build system foundation for HPC software

- **Simple macros for complex tasks**
  - Create libraries, executables, and tests
  - Manages compiler flags across multiple compiler families
  - Unifies complexities of external dependencies into one easy to remember name
- **Batteries included**
  - Example configurations for most LC/Linux/OSX/Windows system and compiler families
  - Built-in support for:
    - HPC programming models
    - Code health
    - Documentation generation
- **Open source**
  - Leveraged by ALE3D, Ascent, Axom, CHAI, Conduit, FloBat, GeosX, Kripke, LEOS, MSLIB, RAJA, RAJA Perf Suite, Umpire, VBF Shaft, VTK-h

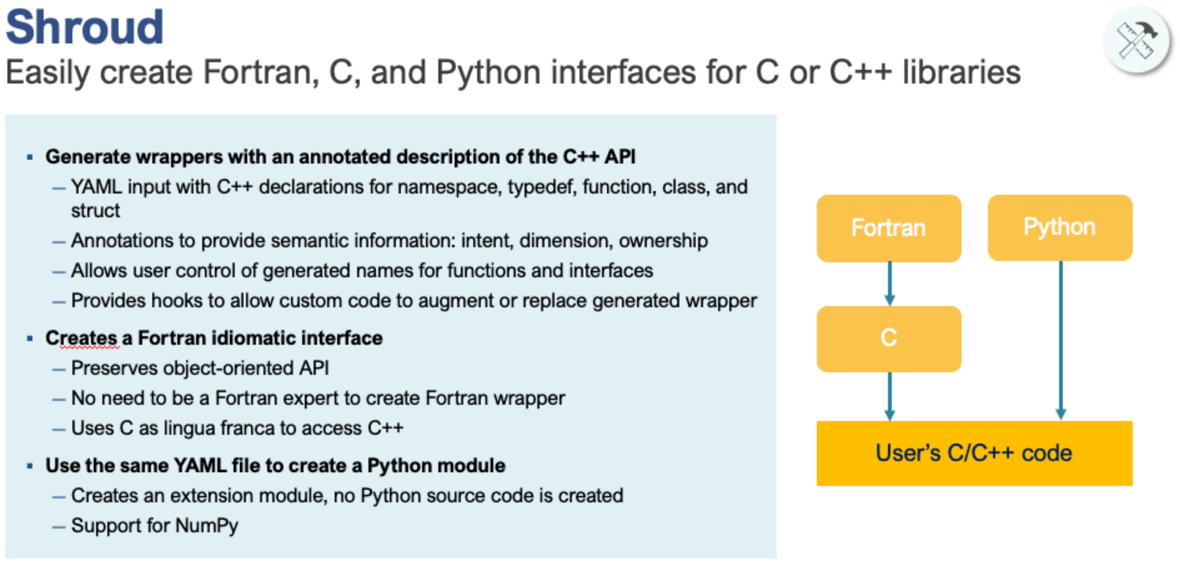
HPC Programming Models

Code Health

Fruit
Clang-query
Uncrustify
Astyle

Documentation

BLT
CMake
Cross-platform Make
NVIDIA CUDA
MPI
OpenMP
ROCm
Cppcheck
googletest
Doxygen
SPHINX



Shroud

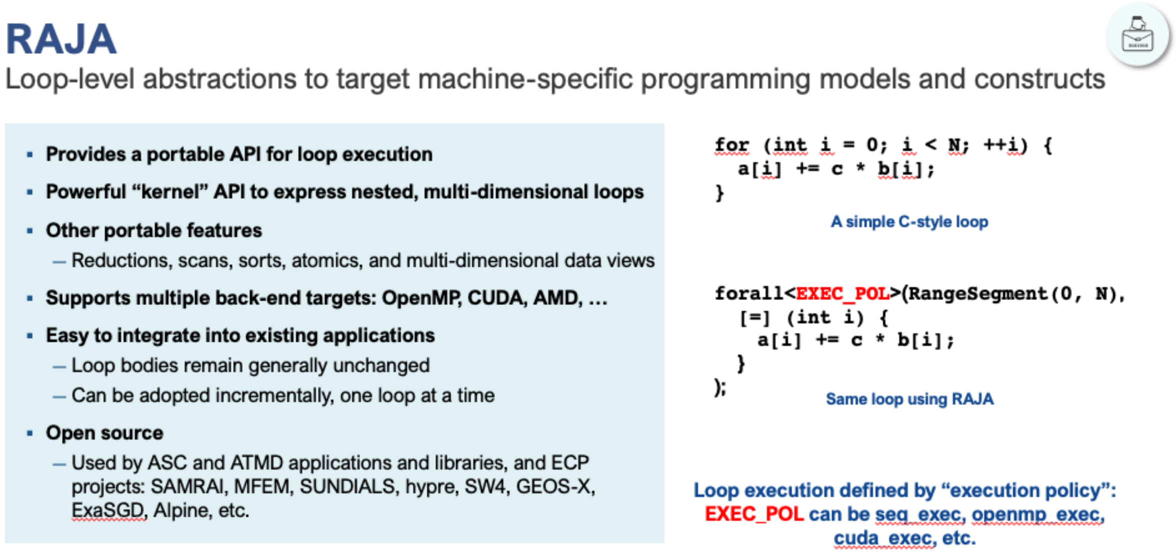
Easily create Fortran, C, and Python interfaces for C or C++ libraries

- **Generate wrappers with an annotated description of the C++ API**
  - YAML input with C++ declarations for namespace, typedef, function, class, and struct
  - Annotations to provide semantic information: intent, dimension, ownership
  - Allows user control of generated names for functions and interfaces
  - Provides hooks to allow custom code to augment or replace generated wrapper
- **Creates a Fortran idiomatic interface**
  - Preserves object-oriented API
  - No need to be a Fortran expert to create Fortran wrapper
  - Uses C as lingua franca to access C++
- **Use the same YAML file to create a Python module**
  - Creates an extension module, no Python source code is created
  - Support for NumPy

Portable Execution and Memory Management

Project	Description	Website	Repository
RAJA	Loop-level abstractions to target machine-specific programming models and constructs	software.llnl.gov/RAJA	github.com/LLNL/RAJA

CHAI	Optional add-on to RAJA for automating data motion between memory spaces	software.llnl.gov/CHAI	github.com/LLNL/CHAI
Umpire	An application-focused API for memory management on NUMA & GPU architectures	software.llnl.gov/Umpire	github.com/LLNL/Umpire
LvArray	Array classes for high-performance simulation software	lvarray.readthedocs.io	github.com/GEOSX/LvArray



RAJA

Loop-level abstractions to target machine-specific programming models and constructs

- **Provides a portable API for loop execution**
- **Powerful "kernel" API to express nested, multi-dimensional loops**
- **Other portable features**
  - Reductions, scans, sorts, atomics, and multi-dimensional data views
- **Supports multiple back-end targets: OpenMP, CUDA, AMD, …**
- **Easy to integrate into existing applications**
  - Loop bodies remain generally unchanged
  - Can be adopted incrementally, one loop at a time
- **Open source**
  - Used by ASC and ATMD applications and libraries, and ECP projects: SAMRAI, MFEM, SUNDIALS, hypre, SW4, GEOS-X, ExaSGD, Alpine, etc.

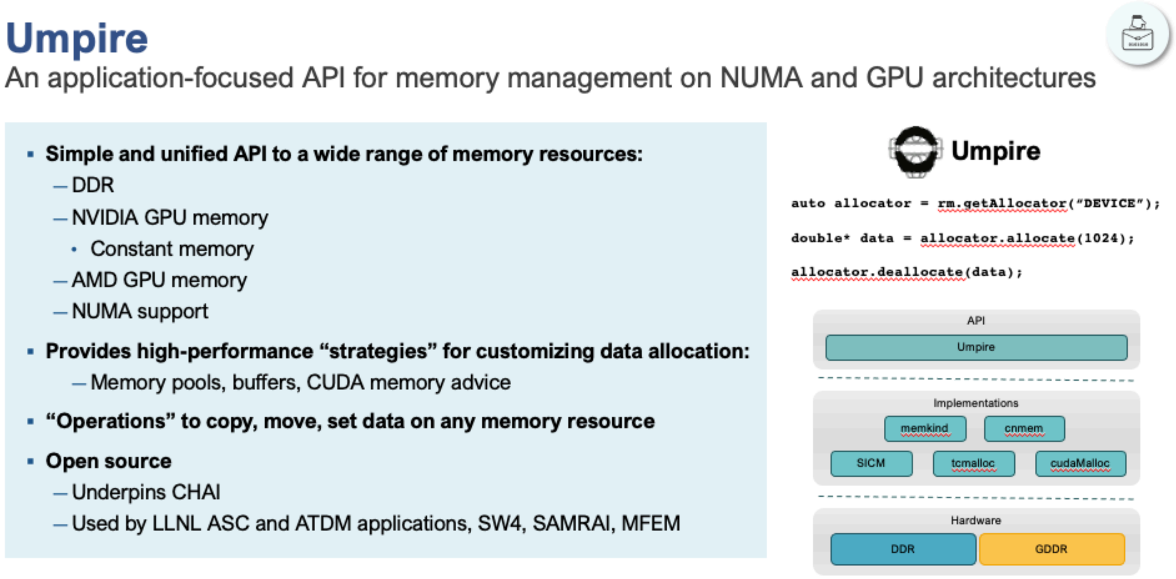
```
for (int i = 0; i < N; ++i) {
  a[i] += c * b[i];
}
```

A simple C-style loop

```
forall<EXEC_POL>(RangeSegment(0, N),
  [=] (int i) {
    a[i] += c * b[i];
  }
);
```

Same loop using RAJA

Loop execution defined by "execution policy": EXEC_POL can be seq_exec, openmp_exec, cuda_exec, etc.



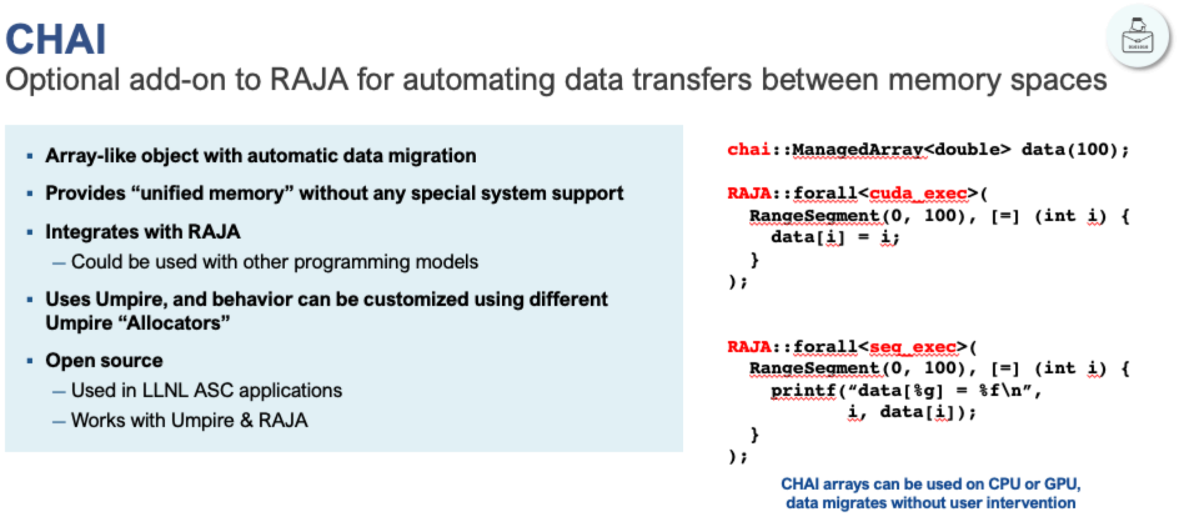
Umpire

An application-focused API for memory management on NUMA and GPU architectures

- **Simple and unified API to a wide range of memory resources:**
  - DDR
  - NVIDIA GPU memory
    - Constant memory
  - AMD GPU memory
  - NUMA support
- **Provides high-performance "strategies" for customizing data allocation:**
  - Memory pools, buffers, CUDA memory advice
- **"Operations" to copy, move, set data on any memory resource**
- **Open source**
  - Underpins CHAI
  - Used by LLNL ASC and ATDM applications, SW4, SAMRAI, MFEM

Umpire

```
auto allocator = rm.getAllocator("DEVICE");

double* data = allocator.allocate(1024);

allocator.deallocate(data);
```



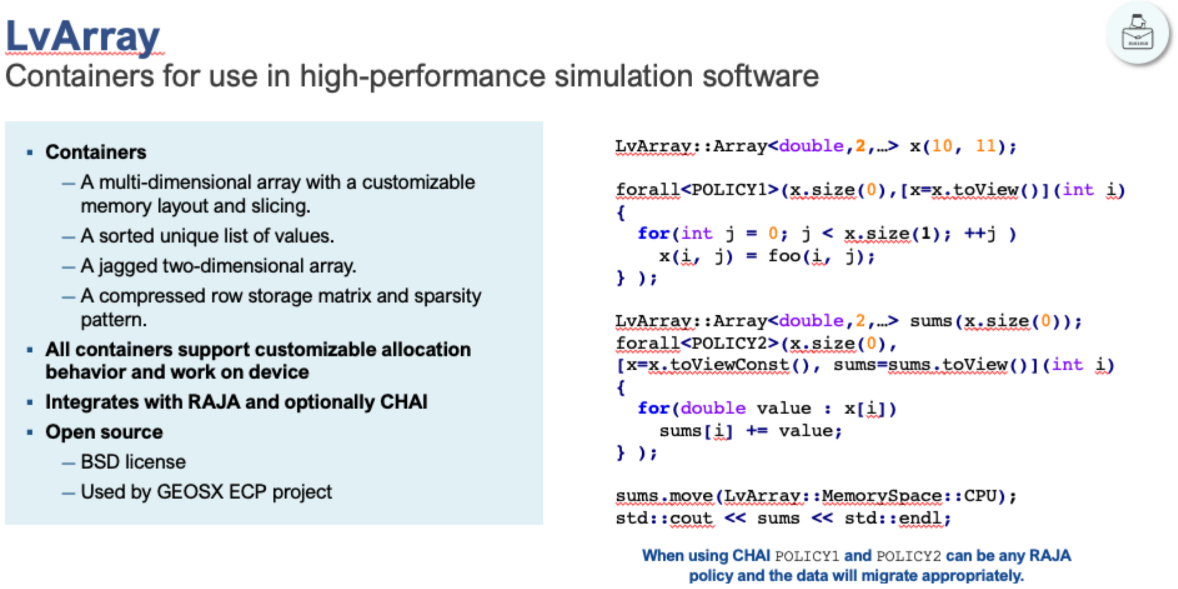
CHAI

Optional add-on to RAJA for automating data transfers between memory spaces

- **Array-like object with automatic data migration**
- **Provides "unified memory" without any special system support**
- **Integrates with RAJA**
  - Could be used with other programming models
- **Uses Umpire, and behavior can be customized using different Umpire "Allocators"**
- **Open source**
  - Used in LLNL ASC applications
  - Works with Umpire & RAJA

```
chai::ManagedArray<double> data(100);

RAJA::forall<cuda_exec>(
  RangeSegment(0, 100), [=] (int i) {
    data[i] = i;
  }
);


RAJA::forall<seq_exec>(
  RangeSegment(0, 100), [=] (int i) {
    printf("data[%g] = %f\n",
           i, data[i]);
  }
);
```

CHAI arrays can be used on CPU or GPU, data migrates without user intervention



LvArray

Containers for use in high-performance simulation software

- **Containers**
  - A multi-dimensional array with a customizable memory layout and slicing.
  - A sorted unique list of values.
  - A jagged two-dimensional array.
  - A compressed row storage matrix and sparsity pattern.
- **All containers support customizable allocation behavior and work on device**
- **Integrates with RAJA and optionally CHAI**
- **Open source**
  - BSD license
  - Used by GEOSX ECP project

```
LvArray::Array<double,2,…> x(10, 11);

forall<POLICY1>(x.size(0),[x=x.toView()](int i)
{
  for(int j = 0; j < x.size(1); ++j )
    x(i, j) = foo(i, j);
} );

LvArray::Array<double,2,…> sums(x.size(0));
forall<POLICY2>(x.size(0),
[x=x.toViewConst(), sums=sums.toView()](int i)
{
  for(double value : x[i])
    sums[i] += value;
} );

sums.move(LvArray::MemorySpace::CPU);
std::cout << sums << std::endl;
```

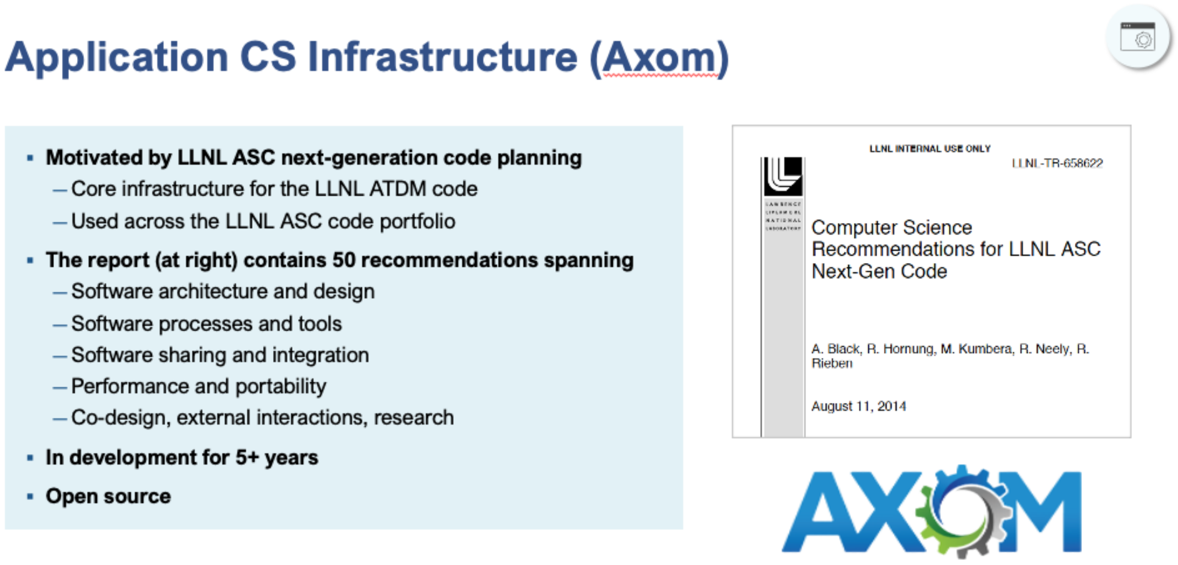
When using CHAI POLICY1 and POLICY2 can be any RAJA policy and the data will migrate appropriately.

Application CS Infrastructure

Project	**Description**	**Website**	**Repository**
Axom	Flexible software infrastructure for the development of multi-physics applications and computational tools	software.llnl.gov/axom	github.com/LLNL/axom

Note: Axom is managed as a single project and repository, but consists of a number of smaller components described below:

Axom Component	Description
Sidre	In-core hierarchical key-value data management, plus parallel file I/O (restart, viz. files), support for heterogeneous memory systems, etc.
Quest	Spatial point/surface queries; in-out, signed distance, point containment, point-in-cell, etc.
Primal	Geometric primitives (point, vector, triangle, etc.) and operations (distance, intersection, closest point, etc.)
Spin	Spatial acceleration data structures; octree, kd-tree, R-tree, BVH, etc.
Mint	Mesh data model; structured, unstructured, particles.
Slam	Set, relation, map abstractions.
Slic/Lumberjack	Unified/shared inter-package message streams, parallel logging, and filtering.



Application CS Infrastructure (Axom)

- **Motivated by LLNL ASC next-generation code planning**
  - Core infrastructure for the LLNL ATDM code
  - Used across the LLNL ASC code portfolio
- **The report (at right) contains 50 recommendations spanning**
  - Software architecture and design
  - Software processes and tools
  - Software sharing and integration
  - Performance and portability
  - Co-design, external interactions, research
- **In development for 5+ years**
- **Open source**

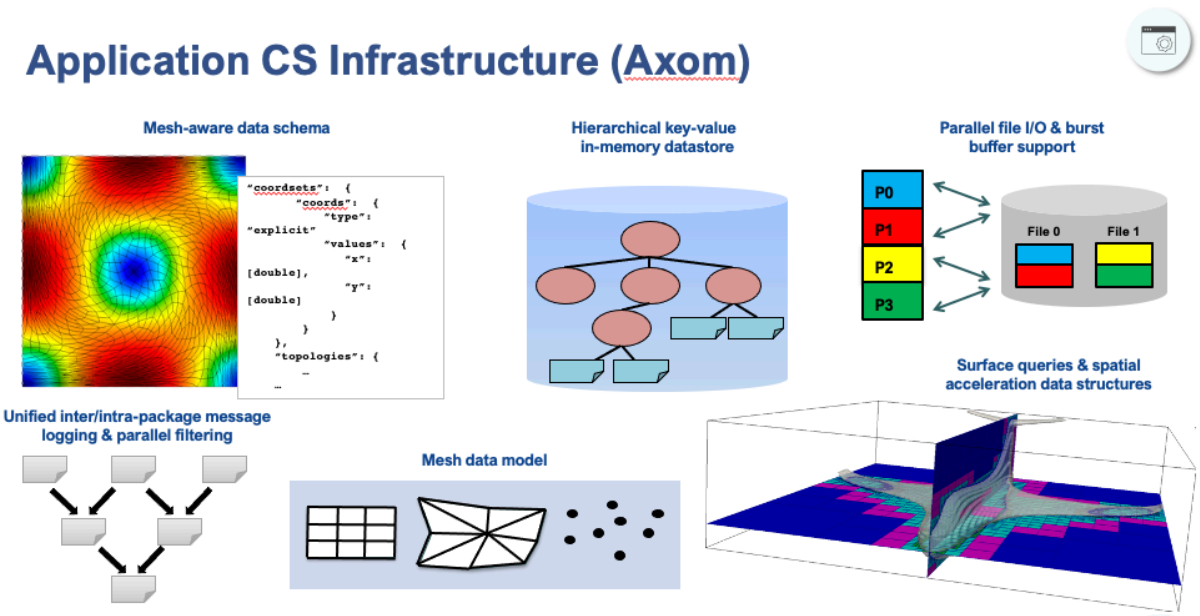
LLNL INTERNAL USE ONLY

LLNL-TR-658622

Computer Science Recommendations for LLNL ASC Next-Gen Code

A. Black, R. Hornung, M. Kumbera, R. Neely, R. Rieben

August 11, 2014



Application CS Infrastructure (Axom)

Mesh-aware data schema

```
"coordsets":  {
        "coords":  {
            "type":
"explicit"
            "values":  {
               "x":
[double],
               "y":
[double]
             }
         }
    },
    "topologies": {
         …
    …
```

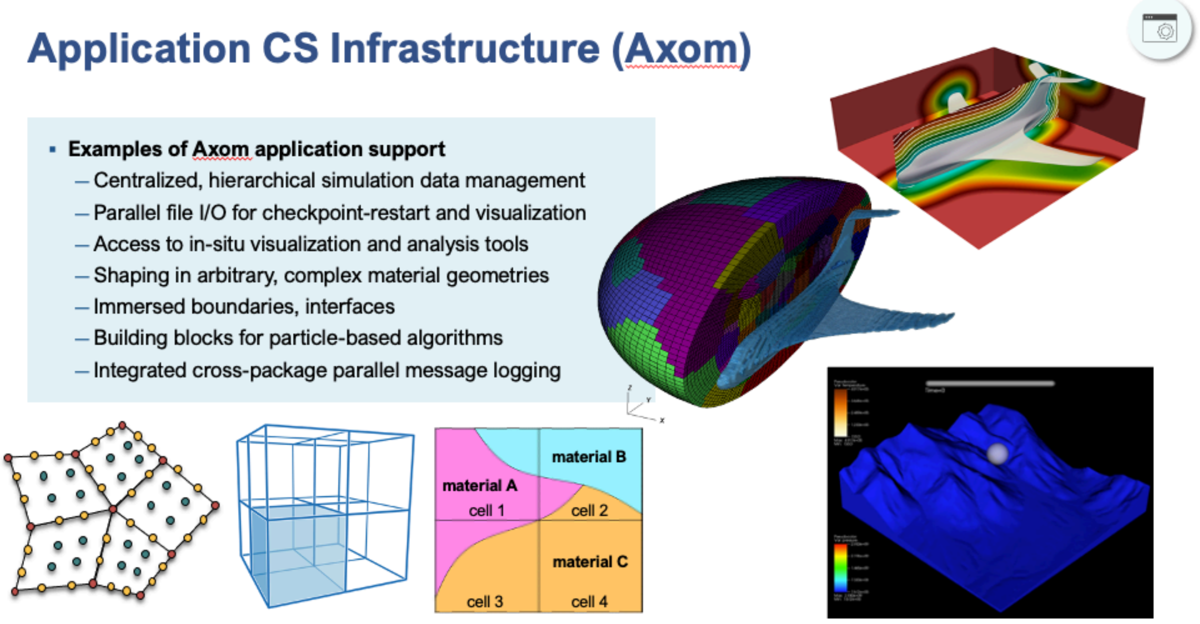
Hierarchical key-value in-memory datastore

Parallel file I/O & burst buffer support

Surface queries & spatial acceleration data structures

Unified inter/intra-package message logging & parallel filtering

Mesh data model



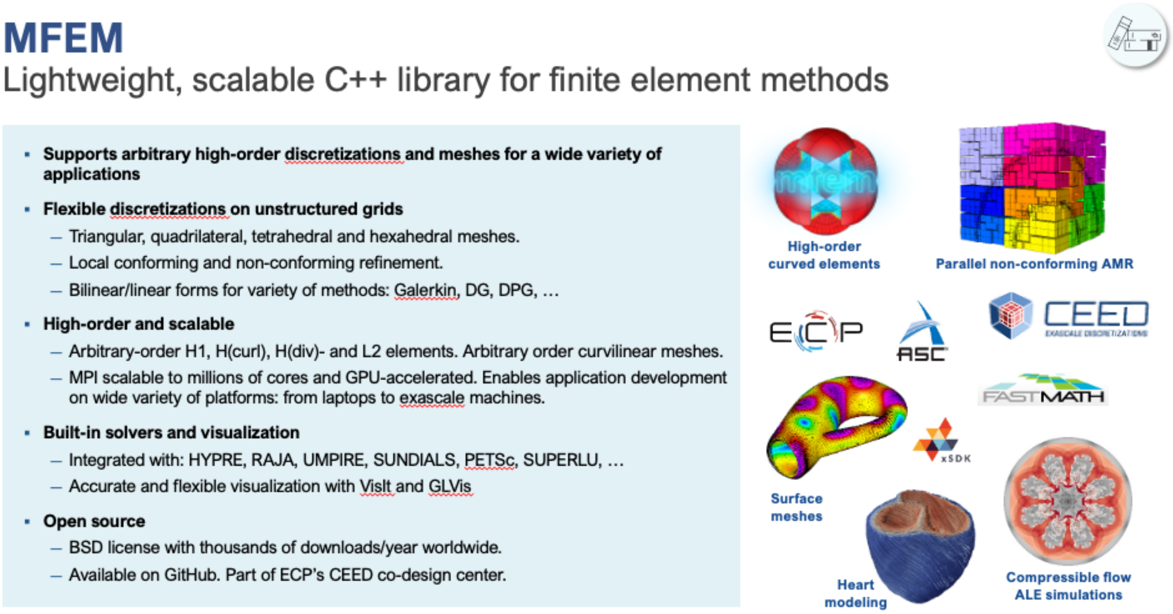
Application CS Infrastructure (Axom)

- Examples of Axom application support
  - Centralized, hierarchical simulation data management
  - Parallel file I/O for checkpoint-restart and visualization
  - Access to in-situ visualization and analysis tools
  - Shaping in arbitrary, complex material geometries
  - Immersed boundaries, interfaces
  - Building blocks for particle-based algorithms
  - Integrated cross-package parallel message logging

Math and Physics Libraries

Project	Description	Website	Repository
MFEM	Unstructured high-order finite element library	mfem.org	github.com/mfem

hypre	Preconditioners and solvers for large-scale matrices	www.llnl.gov/casc/hypre	github.com/hypre-space
SUNDIALS	Nonlinear and differential/algebraic equation solvers	www.llnl.gov/casc/sundials	github.com/LLNL/sundials
SAMRAI	Structured Adaptive Mesh Refinement framework	computation.llnl.gov/projects/samrai	github.com/LLNL/SAMRAI
XBraid	Lightweight support for multigrid Parallel-in-Time	www.llnl.gov/casc/xbraid	github.com/xbraid

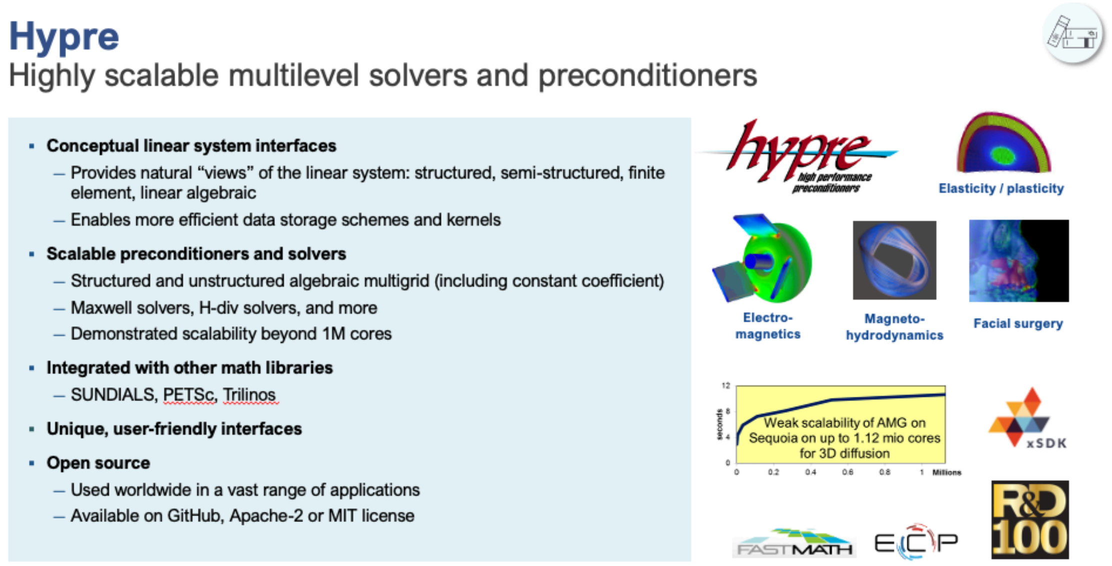


MFEM

Lightweight, scalable C++ library for finite element methods

- **Supports arbitrary high-order discretizations and meshes for a wide variety of applications**
- **Flexible discretizations on unstructured grids**
  - Triangular, quadrilateral, tetrahedral and hexahedral meshes.
  - Local conforming and non-conforming refinement.
  - Bilinear/linear forms for variety of methods: Galerkin, DG, DPG, …
- **High-order and scalable**
  - Arbitrary-order H1, H(curl), H(div)- and L2 elements. Arbitrary order curvilinear meshes.
  - MPI scalable to millions of cores and GPU-accelerated. Enables application development on wide variety of platforms: from laptops to exascale machines.
- **Built-in solvers and visualization**
  - Integrated with: HYPRE, RAJA, UMPIRE, SUNDIALS, PETSc, SUPERLU, …
  - Accurate and flexible visualization with VisIt and GLVis
- **Open source**
  - BSD license with thousands of downloads/year worldwide.
  - Available on GitHub. Part of ECP's CEED co-design center.

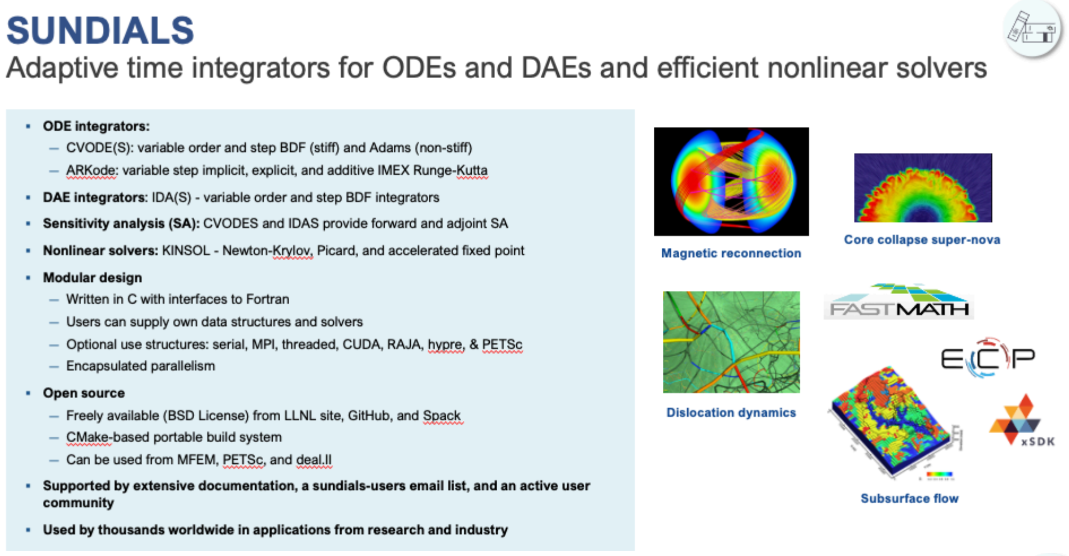
High-order curved elements
Parallel non-conforming AMR
ECP
ASC
CEED
EXASCALE DISCRETIZATIONS
FASTMATH
xSDK
Surface meshes
Heart modeling
Compressible flow ALE simulations



Hypre

Highly scalable multilevel solvers and preconditioners

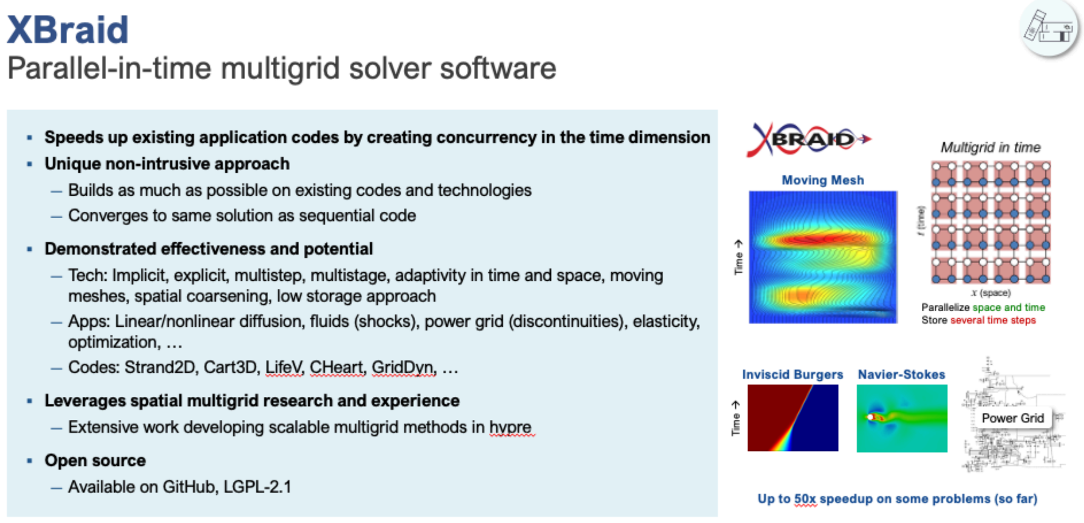
- **Conceptual linear system interfaces**
  - Provides natural “views” of the linear system: structured, semi-structured, finite element, linear algebraic
  - Enables more efficient data storage schemes and kernels
- **Scalable preconditioners and solvers**
  - Structured and unstructured algebraic multigrid (including constant coefficient)
  - Maxwell solvers, H-div solvers, and more
  - Demonstrated scalability beyond 1M cores
- **Integrated with other math libraries**
  - SUNDIALS, PETSc, Trilinos
- **Unique, user-friendly interfaces**
- **Open source**
  - Used worldwide in a vast range of applications
  - Available on GitHub, Apache-2 or MIT license



SUNDIALS

Adaptive time integrators for ODEs and DAEs and efficient nonlinear solvers

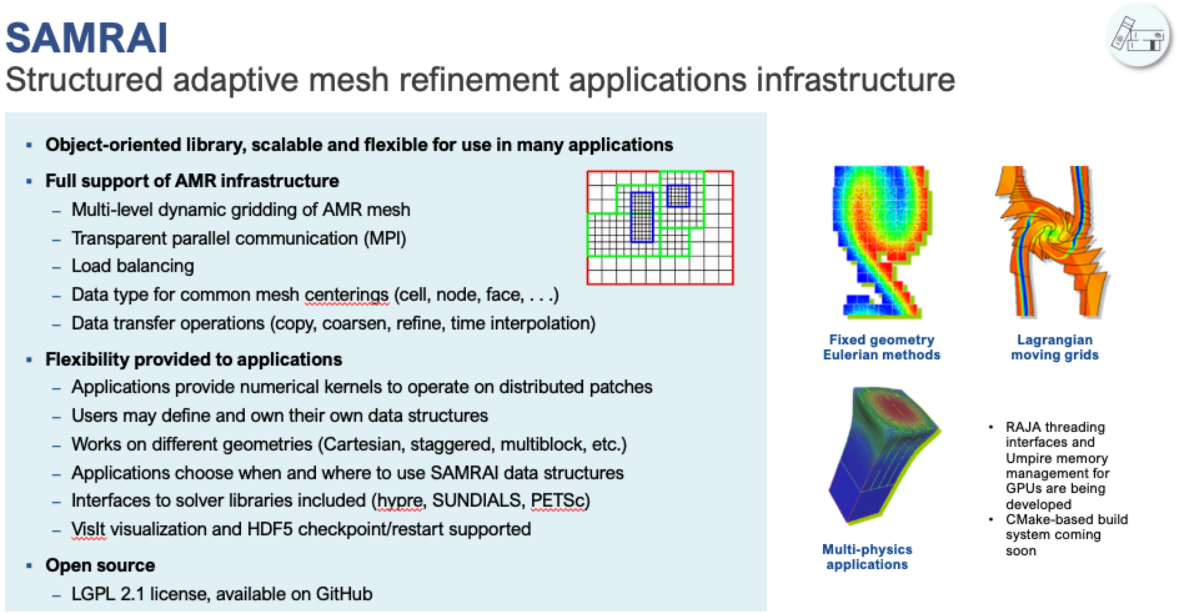
- **ODE integrators:**
  - CVODE(S): variable order and step BDF (stiff) and Adams (non-stiff)
  - ARKode: variable step implicit, explicit, and additive IMEX Runge-Kutta
- **DAE integrators:** IDA(S) - variable order and step BDF integrators
- **Sensitivity analysis (SA):** CVODES and IDAS provide forward and adjoint SA
- **Nonlinear solvers:** KINSOL - Newton-Krylov, Picard, and accelerated fixed point
- **Modular design**
  - Written in C with interfaces to Fortran
  - Users can supply own data structures and solvers
  - Optional use structures: serial, MPI, threaded, CUDA, RAJA, hypre, & PETSc
  - Encapsulated parallelism
- **Open source**
  - Freely available (BSD License) from LLNL site, GitHub, and Spack
  - CMake-based portable build system
  - Can be used from MFEM, PETSc, and deal.II
- **Supported by extensive documentation, a sundials-users email list, and an active user community**
- **Used by thousands worldwide in applications from research and industry**



XBraid

Parallel-in-time multigrid solver software

- **Speeds up existing application codes by creating concurrency in the time dimension**
- **Unique non-intrusive approach**
  - Builds as much as possible on existing codes and technologies
  - Converges to same solution as sequential code
- **Demonstrated effectiveness and potential**
  - Tech: Implicit, explicit, multistep, multistage, adaptivity in time and space, moving meshes, spatial coarsening, low storage approach
  - Apps: Linear/nonlinear diffusion, fluids (shocks), power grid (discontinuities), elasticity, optimization, …
  - Codes: Strand2D, Cart3D, LifeV, CHeart, GridDyn, …
- **Leverages spatial multigrid research and experience**
  - Extensive work developing scalable multigrid methods in hypre
- **Open source**
  - Available on GitHub, LGPL-2.1



SAMRAI

Structured adaptive mesh refinement applications infrastructure

- **Object-oriented library, scalable and flexible for use in many applications**
- **Full support of AMR infrastructure**
  - Multi-level dynamic gridding of AMR mesh
  - Transparent parallel communication (MPI)
  - Load balancing
  - Data type for common mesh centerings (cell, node, face, . . .)
  - Data transfer operations (copy, coarsen, refine, time interpolation)
- **Flexibility provided to applications**
  - Applications provide numerical kernels to operate on distributed patches
  - Users may define and own their own data structures
  - Works on different geometries (Cartesian, staggered, multiblock, etc.)
  - Applications choose when and where to use SAMRAI data structures
  - Interfaces to solver libraries included (hypre, SUNDIALS, PETSc)
  - VisIt visualization and HDF5 checkpoint/restart supported
- **Open source**
  - LGPL 2.1 license, available on GitHub

Fixed geometry Eulerian methods

Lagrangian moving grids

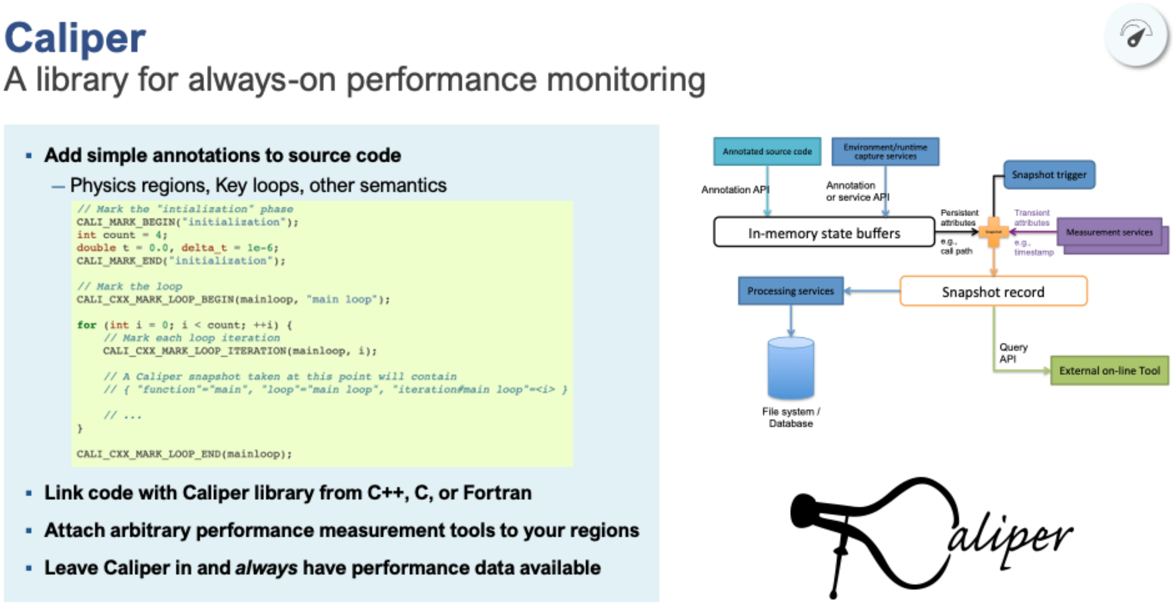
Multi-physics applications

- RAJA threading interfaces and Umpire memory management for GPUs are being developed
- CMake-based build system coming soon

Performance and Workflow Tools

Project	Description	Website	Repository
Caliper	Always-on performance measurement library	llnl.github.io/Caliper/	github.com/LLNL/Caliper
SPOT	Performance history tracking	computing.llnl.gov/projects/caliper	github.com/LLNL/Caliper
Flux	Resource management and scheduling	flux-framework.org	github.com/flux-framework
MaestroWF	A tool and library for specifying and conducting general workflows	maestrowf.readthedocs.io	github.com/LLNL/maestrowf

Spindle	Library loading and program start-up at scale	computing.llnl.gov/projects/spindle	github.com/hpc/spindle
LBANN	Machine learning training and inference at extreme scale	lbann.readthedocs.io	github.com/LLNL/lbann



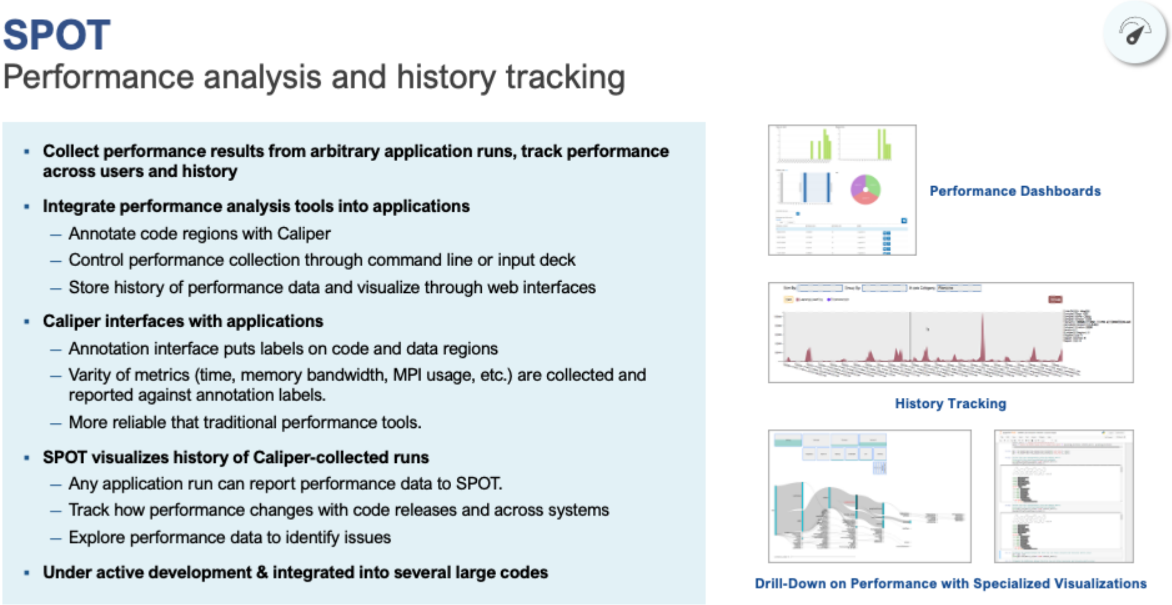
Caliper

A library for always-on performance monitoring

- **Add simple annotations to source code**
  - Physics regions, Key loops, other semantics

```
// Mark the "intialization" phase
CALI_MARK_BEGIN("initialization");
int count = 4;
double t = 0.0, delta_t = 1e-6;
CALI_MARK_END("initialization");

// Mark the loop
CALI_CXX_MARK_LOOP_BEGIN(mainloop, "main loop");

for (int i = 0; i < count; ++i) {
    // Mark each loop iteration
    CALI_CXX_MARK_LOOP_ITERATION(mainloop, i);

    // A Caliper snapshot taken at this point will contain
    // { "function"="main", "loop"="main loop", "iteration#main loop"=<i> }

    // ...
}

CALI_CXX_MARK_LOOP_END(mainloop);
```

- **Link code with Caliper library from C++, C, or Fortran**
- **Attach arbitrary performance measurement tools to your regions**
- **Leave Caliper in and *always* have performance data available**



SPOT

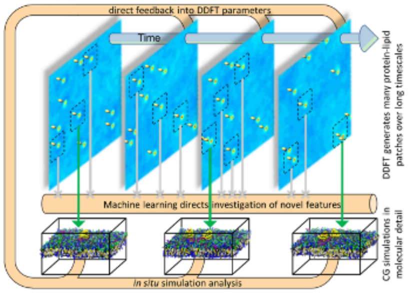
Performance analysis and history tracking

- **Collect performance results from arbitrary application runs, track performance across users and history**
- **Integrate performance analysis tools into applications**
  - Annotate code regions with Caliper
  - Control performance collection through command line or input deck
  - Store history of performance data and visualize through web interfaces
- **Caliper interfaces with applications**
  - Annotation interface puts labels on code and data regions
  - Varity of metrics (time, memory bandwidth, MPI usage, etc.) are collected and reported against annotation labels.
  - More reliable that traditional performance tools.
- **SPOT visualizes history of Caliper-collected runs**
  - Any application run can report performance data to SPOT.
  - Track how performance changes with code releases and across systems
  - Explore performance data to identify issues
- **Under active development & integrated into several large codes**

Flux

Next-generation resource management and scheduling framework to address emerging challenges

- **Workflow challenges**
  - Modern Workflows are increasingly difficult to schedule
  - Cancer Moonshot Pilot2, Machine Learning LDRD Strategic Initiative, …
- **Resource challenges**
  - Changes in resource types are equally challenging
  - GPGPUs, Burst buffers, under-provisioned PFS BW, …
- **Fully hierarchical approach for job throughput/co-scheduling**
- **Graph-based resource model for resource challenges**
- **Rich APIs for workflow communication and coordination**
- **Consistent APIs for workflow portability and reproducibility**

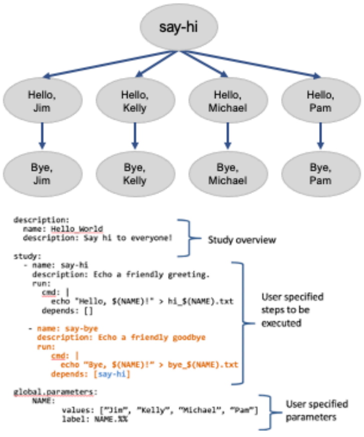

direct feedback into DDFT parameters
Time
DDFT generates many protein-lipid patches over long timescales
Machine learning directs investigation of novel features
CG simulations in molecular detail
In situ simulation analysis

@FluxFramework

MaestroWF

A standard framework to make simulation studies easier to manage, run, and expand

- **Consistent study specification definition**
  - Specify multi-step workflows in a human-readable and self-documenting YAML specification.
  - Studies can be linear or parameterized, are easily shareable between users, and can be software generated.
  - Easily repeat studies simply by launching an existing specification.
- **Lightweight workflow automation and monitoring**
  - Studies are parsed, expanded based on parameters, and monitored automatically.
  - Workflows are expanded into DAGs, with workflow steps being launched as their dependencies allow them.
- **Easy for users to specify and launch workflows**
  - Specifications being shareable allows existing studies to serve as templates for new ones (making both set up and knowledge sharing easier).
  - A study specification allows users to build standard infrastructure to generate the necessary YAML to run larger collections of studies.



```
description:
  name: Hello_World
  description: Say hi to everyone!

study:
  - name: say-hi
    description: Echo a friendly greeting.
    run:
      cmd: |
        echo "Hello, $(NAME)!" > hi_$(NAME).txt
      depends: []

   - name: say-bye
     description: Echo a friendly goodbye
     run:
       cmd: |
         echo "Bye, $(NAME)!" > bye_$(NAME).txt
       depends: [say-hi]

global.parameters:
    NAME:
          values: ["Jim", "Kelly", "Michael", "Pam"]
          label: NAME.%%
```

Study overview

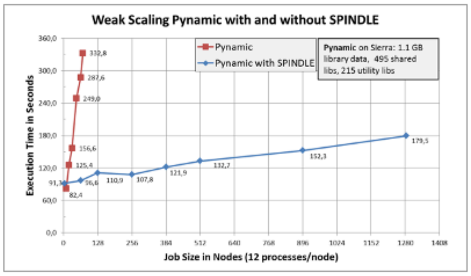
User specified steps to be executed

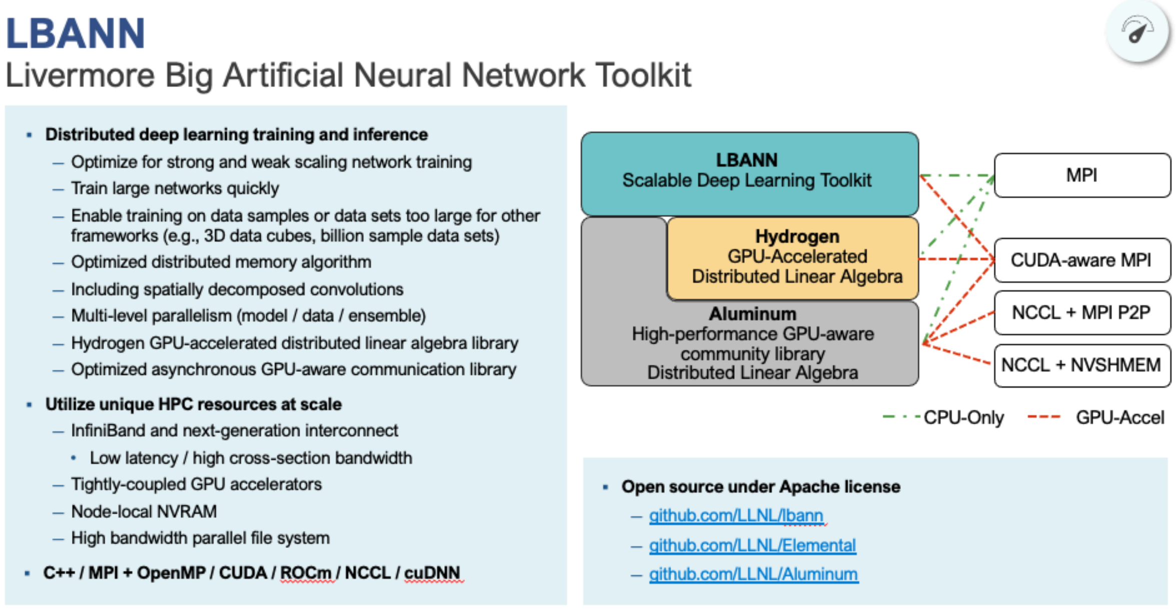
User specified parameters

Spindle

Scalable application start-up

- **Job launch not scalable with many libraries or Python**
  - Solves start-up issues from loading libraries and Python modules at scale
  - Nodes hammer shared file systems when searching and loading libraries
  - Impacts users across whole center
- **Spindle makes job launch scalable**
  - Single node loads libraries/python-modules.
  - Broadcasts libraries to other nodes over high-bandwidth communication network.
  - Run by: `% spindle srun -n 512 ./myapp`
- **Open source**
  - LGPL-2.1 with thousands of downloads/year worldwide
  - Available on GitHub





LBANN

Livermore Big Artificial Neural Network Toolkit

- **Distributed deep learning training and inference**
  - Optimize for strong and weak scaling network training
  - Train large networks quickly
  - Enable training on data samples or data sets too large for other frameworks (e.g., 3D data cubes, billion sample data sets)
  - Optimized distributed memory algorithm
  - Including spatially decomposed convolutions
  - Multi-level parallelism (model / data / ensemble)
  - Hydrogen GPU-accelerated distributed linear algebra library
  - Optimized asynchronous GPU-aware communication library
- **Utilize unique HPC resources at scale**
  - InfiniBand and next-generation interconnect
    - Low latency / high cross-section bandwidth
  - Tightly-coupled GPU accelerators
  - Node-local NVRAM
  - High bandwidth parallel file system
- **C++ / MPI + OpenMP / CUDA / ROCm / NCCL / cuDNN**

LBANN
Scalable Deep Learning Toolkit

Hydrogen
GPU-Accelerated
Distributed Linear Algebra

Aluminum
High-performance GPU-aware
community library
Distributed Linear Algebra

MPI

CUDA-aware MPI

NCCL + MPI P2P

NCCL + NVSHMEM

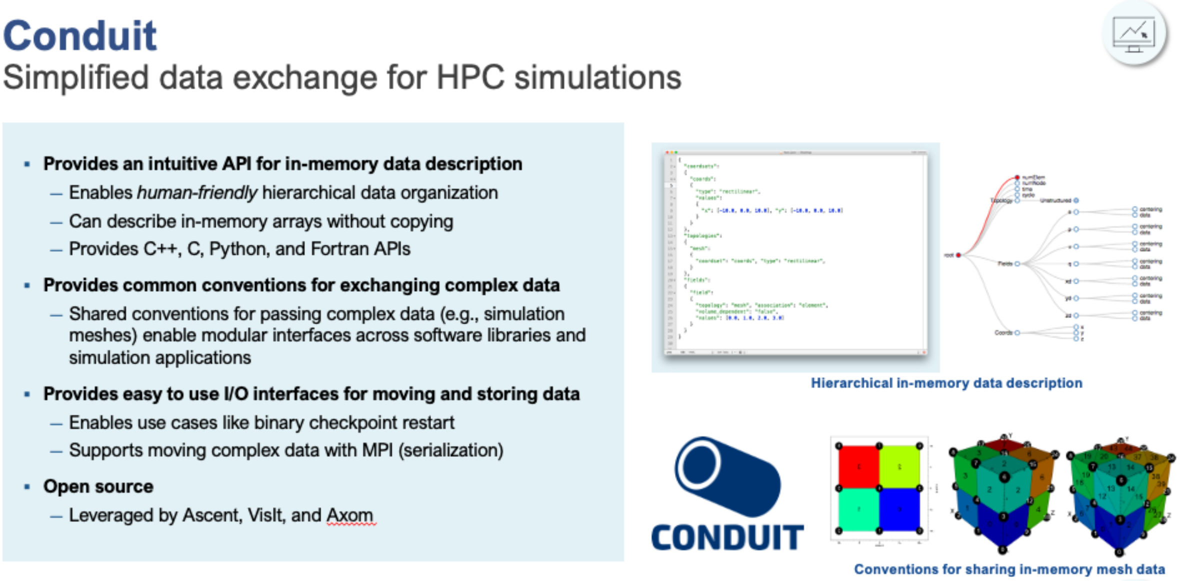
CPU-Only GPU-Accel

- **Open source under Apache license**
  - github.com/LLNL/lbann
  - github.com/LLNL/Elemental
  - github.com/LLNL/Aluminum

Data Management and Visualization

Project	Description	Website	Repository
Conduit	Simplified data exchange for HPC simulations	software.llnl.gov/conduit	github.com/llnl/conduit
Ascent	Flyweight in situ visualization and analysis for HPC simulations	ascent-dav.org	github.com/alpine-dav/ascent
zfp	In-memory compression of floating-point arrays	zfp.readthedocs.io	github.com/LLNL/zfp
SCR	Multilevel checkpointing support and burst buffer interface	scr.readthedocs.io	github.com/LLNL/scr/
VisIt	Feature-rich mesh-based visualization and analysis platform	visit.llnl.gov	github.com/visit-dav/visit

GLVis	Lightweight high order visualization for MFEM	glvis.org	github.com/GLVis/glvis



Conduit

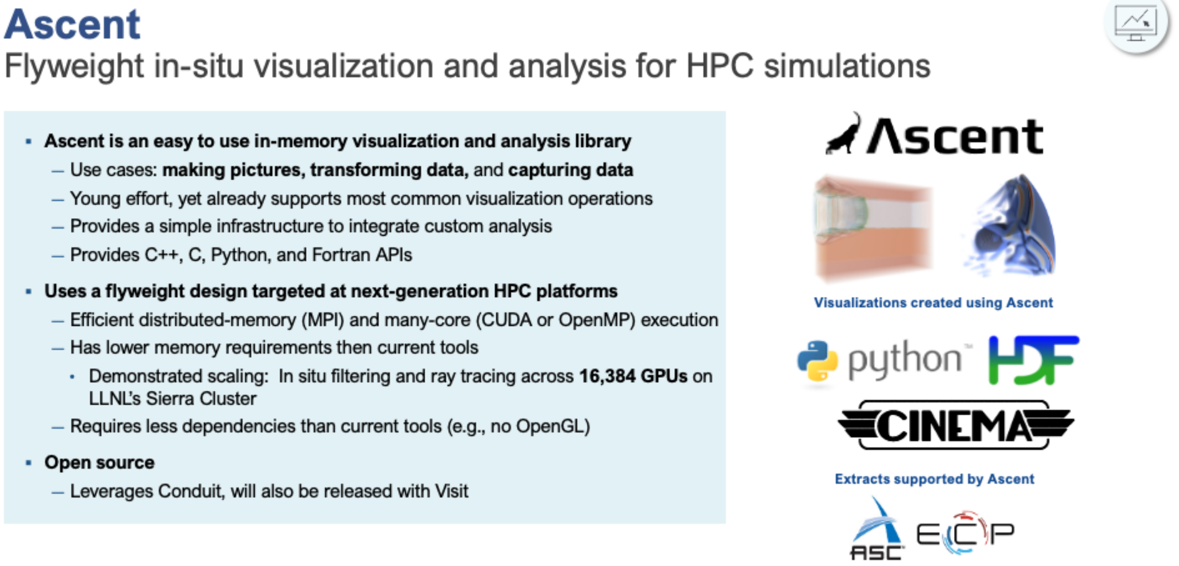
Simplified data exchange for HPC simulations

- **Provides an intuitive API for in-memory data description**
  - Enables *human-friendly* hierarchical data organization
  - Can describe in-memory arrays without copying
  - Provides C++, C, Python, and Fortran APIs
- **Provides common conventions for exchanging complex data**
  - Shared conventions for passing complex data (e.g., simulation meshes) enable modular interfaces across software libraries and simulation applications
- **Provides easy to use I/O interfaces for moving and storing data**
  - Enables use cases like binary checkpoint restart
  - Supports moving complex data with MPI (serialization)
- **Open source**
  - Leveraged by Ascent, VisIt, and Axom

```
{
  "coordsets":
  {
    "coords":
    {
      "type": "rectilinear",
      "values":
      {
        "x": [-10.0, 0.0, 10.0], "y": [-10.0, 0.0, 10.0]
      }
    }
  },
  "topologies":
  {
    "mesh":
    {
      "coordset": "coords", "type": "rectilinear",
    }
  },
  "fields":
  {
    "field":
    {
      "topology": "mesh", "association": "element",
      "volume_dependent": "false",
      "values": [0.0, 1.0, 2.0, 3.0]
    }
  }
}
```

Hierarchical in-memory data description

Conventions for sharing in-memory mesh data



Ascent

Flyweight in-situ visualization and analysis for HPC simulations

- **Ascent is an easy to use in-memory visualization and analysis library**
  - Use cases: **making pictures, transforming data,** and **capturing data**
  - Young effort, yet already supports most common visualization operations
  - Provides a simple infrastructure to integrate custom analysis
  - Provides C++, C, Python, and Fortran APIs
- **Uses a flyweight design targeted at next-generation HPC platforms**
  - Efficient distributed-memory (MPI) and many-core (CUDA or OpenMP) execution
  - Has lower memory requirements then current tools
    - Demonstrated scaling: In situ filtering and ray tracing across **16,384 GPUs** on LLNL's Sierra Cluster
  - Requires less dependencies than current tools (e.g., no OpenGL)
- **Open source**
  - Leverages Conduit, will also be released with VisIt

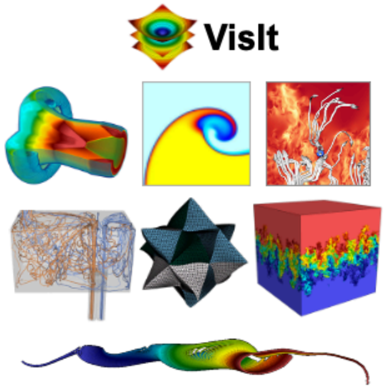
Visualizations created using Ascent

Extracts supported by Ascent

VisIt

Full-featured visualization and analysis for HPC simulations

- **Production end-user tool supporting scientific and engineering applications**
  - Use cases: ***data exploration***, ***quantitative analysis***, ***visual debugging***, ***comparative analysis*** and generation of ***presentation graphics***
  - Provides a rich feature set and a flexible data model suitable for many scientific domains
  - Includes more than 100 file format readers
  - Provides GUI and Python interfaces, extendable via C++ and Python
- **Provides parallel post-processing infrastructure that scales from desktops to massive HPC clusters**
  - Uses MPI for distributed-memory parallelism on HPC clusters
  - Development underway to leverage on-node many-core (CUDA or OpenMP) parallelism
- **Open source**
  - Used as a platform to deploy research from the DOE visualization community
  - Initially developed by LLNL to support ASC, now co-developed by several organizations

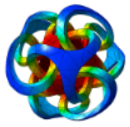


Visualizations created using VisIt

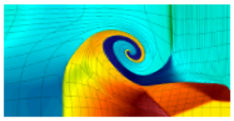
GLVis

Lightweight OpenGL tool for accurate and flexible interactive finite element visualization

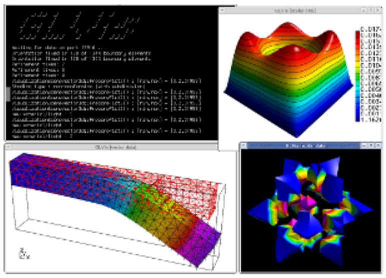
- **Accurate visualization**
  - 1D/2D/3D, volume/surface, triangular/quad/tet/hex, low/high-order meshes
  - Arbitrary high-order, scalar and vector finite element and NURBS solutions
  - Visualization of parallel meshes and solutions
- **Lightweight and interactive**
  - Unlimited number of refinement and de-refinement levels
  - Support for antialiasing, accurate cutting planes, materials, lighting, and transparency
  - Processor and element shrinking for better visualization of 3D mesh interiors
- **Flexible server support**
  - Simultaneous visualization of multiple fields/meshes in separate GLVis windows
  - Local visualization for remote parallel runs with secure socket connections
  - Persistent visualization of time-evolving fields
- **Open source**
  - LGPL-2.1. Available on GitHub
  - Based on the MFEM finite element library
  - Used in MFEM, MARBL/BLAST, LiDO, and more



Supports general meshes and fields



Visualization of a time-dependent high-order BLAST simulation

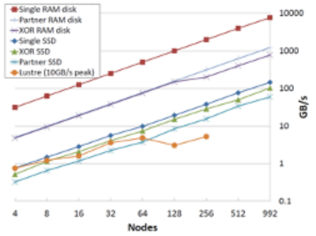


GLVis server sessions with multiple windows

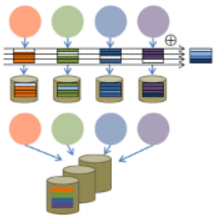
Scalable Checkpoint/Restart (SCR) Library

Enables fast, portable I/O to burst buffers across HPC systems

- **SCR provides fast, scalable I/O performance for LLNL applications**
  - SCR caches output data in node local storage like RAM disk or burst buffer, which can be as much as 1000x faster than the parallel file system
  - SCR hides the complexity of different burst buffer systems and storage architectures
- **Easy integration into application codes**
  - Simple wrapper API around existing checkpoint/restart code
  - Full featured scripting tools wrap existing job launch commands, e.g. srun → scr_srun
- **SCR now enables fast I/O for general output from applications**
  - SCR can now cache visualization dumps or other output to node local storage and drain data to the parallel file system in the background
  - Applications can output data more frequently without the overhead
- **Open source**
  - Available on GitHub with BSD license



SCR's I/O strategies scale with the number of nodes in an HPC job

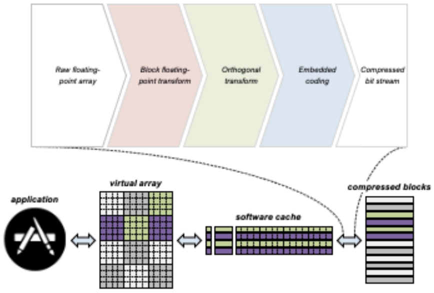


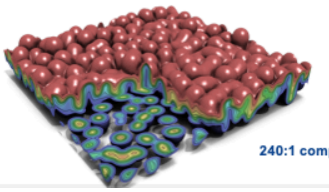
SCR protects your data and manages the complexity of HPC storage hierarchies for performance portable I/O

ZFP

In-memory compression of floating-point and integer arrays

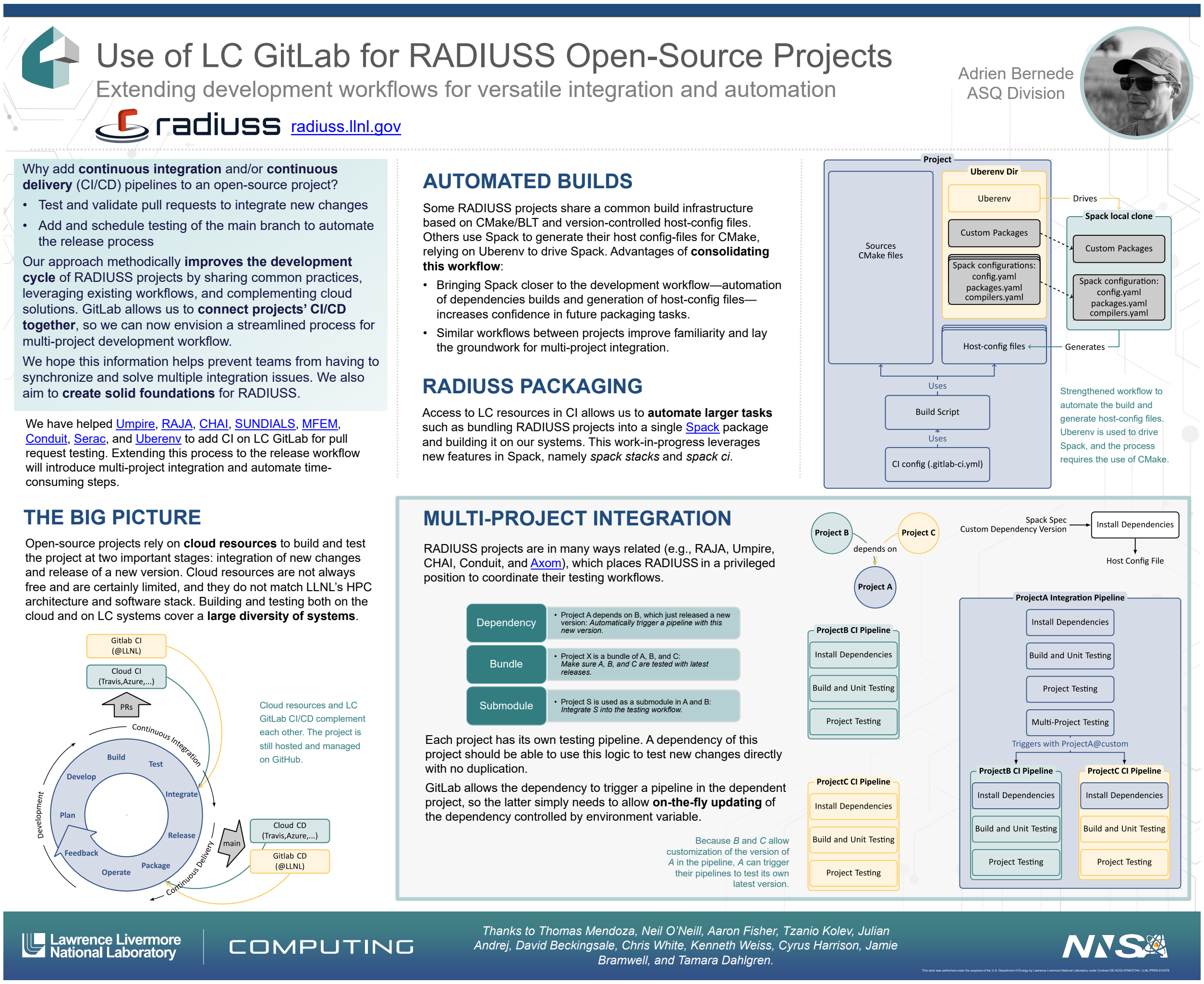
- **Provides a conventional array interface for multidimensional scalar fields**
  - Supports constant-time read & write random access to any array element
  - Hides complexity of (de)compression via C++ operator overloading
  - Provides efficient data access via iterators, views, proxy references and pointers
  - Supports thread safe access and STL algorithms
- **Provides a simple API for (de)compression of whole arrays**
  - Supports prescribed error tolerance or precision, exact storage, lossless compression
  - Supports OpenMP and CUDA parallel (de)compression at up to 150 GB/s throughput
  - Provides C++, C, Python, and Fortran APIs
  - Suitable for compressing checkpoints, viz dumps, MPI messages, CPU-GPU transfers
- **Open source**
  - BSD licensed and available via GitHub, Spack, and Fedora RPM
  - Supported by Intel IPP, HDF5, Silo, ADIOS, VTK-m, LEOS, E4S, …





240:1 compression

Section 10 Getting Started with GitLab CI



Use of LC GitLab for RADIUSS Open-Source Projects

Extending development workflows for versatile integration and automation

radiuss radiuss.llnl.gov

Adrien Bernede
ASQ Division

Why add **continuous integration** and/or **continuous delivery** (CI/CD) pipelines to an open-source project?

- Test and validate pull requests to integrate new changes
- Add and schedule testing of the main branch to automate the release process

Our approach methodically **improves the development cycle** of RADIUSS projects by sharing common practices, leveraging existing workflows, and complementing cloud solutions. GitLab allows us to **connect projects' CI/CD together**, so we can now envision a streamlined process for multi-project development workflow.

We hope this information helps prevent teams from having to synchronize and solve multiple integration issues. We also aim to **create solid foundations** for RADIUSS.

We have helped Umpire, RAJA, CHAI, SUNDIALS, MFEM, Conduit, Serac, and Uberenv to add CI on LC GitLab for pull request testing. Extending this process to the release workflow will introduce multi-project integration and automate time-consuming steps.

THE BIG PICTURE

Open-source projects rely on **cloud resources** to build and test the project at two important stages: integration of new changes and release of a new version. Cloud resources are not always free and are certainly limited, and they do not match LLNL's HPC architecture and software stack. Building and testing both on the cloud and on LC systems cover a **large diversity of systems**.

Cloud resources and LC GitLab CI/CD complement each other. The project is still hosted and managed on GitHub.

AUTOMATED BUILDS

Some RADIUSS projects share a common build infrastructure based on CMake/BLT and version-controlled host-config files. Others use Spack to generate their host config-files for CMake, relying on Uberenv to drive Spack. Advantages of **consolidating this workflow**:

- Bringing Spack closer to the development workflow—automation of dependencies builds and generation of host-config files—increases confidence in future packaging tasks.
- Similar workflows between projects improve familiarity and lay the groundwork for multi-project integration.

RADIUSS PACKAGING

Access to LC resources in CI allows us to **automate larger tasks** such as bundling RADIUSS projects into a single Spack package and building it on our systems. This work-in-progress leverages new features in Spack, namely *spack stacks* and *spack ci*.

Strengthened workflow to automate the build and generate host-config files. Uberenv is used to drive Spack, and the process requires the use of CMake.

MULTI-PROJECT INTEGRATION

RADIUSS projects are in many ways related (e.g., RAJA, Umpire, CHAI, Conduit, and Axom), which places RADIUSS in a privileged position to coordinate their testing workflows.

Dependency	• Project A depends on B, which just released a new version: *Automatically trigger a pipeline with this new version.*
Bundle	• Project X is a bundle of A, B, and C: *Make sure A, B, and C are tested with latest releases.*
Submodule	• Project S is used as a submodule in A and B: *Integrate S into the testing workflow.*

Each project has its own testing pipeline. A dependency of this project should be able to use this logic to test new changes directly with no duplication.

GitLab allows the dependency to trigger a pipeline in the dependent project, so the latter simply needs to allow **on-the-fly updating** of the dependency controlled by environment variable.

Because *B* and *C* allow customization of the version of *A* in the pipeline, *A* can trigger their pipelines to test its own latest version.

Lawrence Livermore National Laboratory | COMPUTING

Thanks to Thomas Mendoza, Neil O'Neill, Aaron Fisher, Tzanio Kolev, Julian Andrej, David Beckingsale, Chris White, Kenneth Weiss, Cyrus Harrison, Jamie Bramwell, and Tamara Dahlgren.

NNSA

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-PRES-814278