DOE RFI: "Stewardship of Software for Scientific and High-Performance Computing"

Andy Gallo
agallojr@518computing.com (also andrew.gallo@ge.com)
(518) 618-2667

Introduction

Thank you for the opportunity to respond. Given the timelines of this DOE call for input, it was not possible for my current team at GE Research to prepare a comprehensive peer-reviewed response. Thus, I am responding personally and these comments, while informed by my work experience including at GE Research, are not formally representative of a GE position.

With that said, for context my own background is tersely:

- MS Computer Science in the early '90s with experience as a research programmer at a time when object-oriented was just being introduced to that community
- Starting in the mid-90's and basing loosely on the DOD-sponsored CMM model, helped many companies – engineering firms and other businesses – to mature their software engineering processes
- Three decades of experience as a software engineer, architect, and leader of development, QA, and operational teams using a wide variety of technologies. This included long stints at engineering firms such as GE Power and Boeing.
- Five years at GE Research, including the last two years immersed in HPC software. Prior projects involved distributed workflows for surrogate model training, and IoT data ingest and visualization.
- Adjunct professor of Computer Science including undergraduate software engineering

Dwelling on the last two years with HPC software at GE Research, efforts have included:

- Working with a CFD solver application which originates at an upstream research university, is touched by collaborators from two competing GPU vendors, and is then further customized by GE Research.
- Adapting the build to use Spack for package management and using it to build on an increasing number of platforms (6-10 systems now). Implementing a CI/CD build and test pipeline using GitLab CI.
- Attempting to integrate the CFD physicists and their coding workflows into the above SE process and tooling and for non-technical social reasons failing.
- Experimenting with and performance testing ADIOS2 for enabling in-situ analysis, visualization, simulation control, and coupling for multi-physics and ML surrogate training applications. After refactoring we ultimately cut out ADIOS2 from the implementation and used straight MPI wrapped by our own use case layer.

- Acted as the principal or supporting investigator on several engagements with the national laboratory computing centers, including on: Summit, Spock, Perlmutter, Marconi-100 @ CINECA, and machines at Navy DSRC.
- Discussing long-term strategies at GE for staffing HPC software development and operational teams in a sustainable manner which promotes inclusion and diversity and concluding that the process is best begun as early as possible – in our schools.

The broader motivation for my current projects is twofold:

1) Maintain the readiness of *M* applications (the CFD app and potentially other GE apps) on *N* platforms in support of a diverse and multi-disciplinary team of GE researchers[1] who themselves actively pursue granted time on leadership-class machines, while exploring challenges to the same.

2) Experiment with approaches to workflow orchestration for: pre-/in-/post-processing, multi-physics, ML applications, etc. around high-performance heterogeneous computing.

1 - Software Dependencies

The following software packages are currently used in the projects for which this author is involved. It's likely not an all-inclusive list.

- Spack – used by the computer science team but not yet widely adopted by the general research group or collaborators
- Cmake
- Python and many libraries including PyBind, NumPy, SciPy, TensorFlow
- GCC, Clang; Java on non-HPC nodes involved in wider macro-workflows
- MPI – OpenMPI, MPICH, Spectrum; AMQP implementations for macro-workflows
- ParaView, VTK – also use commercial tools in this visualization space
- HDF5, CGNS
- PETSc
- HIP – for CUDA to AMD GPU port
- Git, GitLab CI

What do these projects need? We're tracking a few things:

- Programming interfaces which provide consistent interfaces over a growing set of heterogenous computes – CPU, GPU, and more. Such interfaces need to balance

[1] The scope of GE Research, which can also be considered the scope of the scientific work referred to in this paper, is broad but is centered around the three core missions of the company as it is currently constituted: the future of flight, clean energy, and precision medicine. https://www.ge.com/research/

abstract generality and the ability to drill into vendor-specific details when needed in a manner which promotes good software engineering practices (encapsulation, reuse, etc.).
- Given that new greenfield development is not the norm, there needs to be a plausible migration path to deploy existing code to new hardware using the above – total rewrites are generally not plausible.
- More portable build and execution environments (e.g., containers) to lighten the operational burden of *M* apps x *N* platforms, ideally as a shared service.
- Similarly, high-level standardized scheduler interfaces to further streamline differences between specific scheduler vendors.
- Secure mechanisms to handle the multi-factor authentication boundary to the national lab machines, which currently inhibits automated CI/CD pipelines.

Risks to maintaining dependencies and new adoption? Without a doubt, the single greatest issue is the immaturity of internal software engineering processes, and a general inability to come to terms with their importance. This is often reported in other scientific organizations. How are scientific contributors measured and rewarded? Usually not for the maturity of their SE processes, and while such maturation might improve the science, the gratification is potentially non-obvious and at minimum delayed. We'll not belabor the point further here, but it's one which needs addressing as it's a significant inhibitor to productivity, technology adoption, reproducibility of scientific results, etc. This gap in capability, which one could see as a national strategic imperative, is best addressed sooner rather than later in the careers of new scientists.

Secondly, further risks to adoption of tools which originate from incubating scientific organizations like the ECP E4S are their need to be hardened, have at least community-grade documentation and support, plausible product roadmaps, etc. This is discussed further below.

2 - Data Security

What security technologies are used today?

- We mentioned the national lab MFA perimeter, and GE has similar.
- At CINECA, we have been given a secure VM to use in building. Thus, we copy our critical IP software codes to a secure VM to which only we have access, we build, and then copy just the executables to the main machine (Marconi-100) for execution.
- At US and EU facilities, it's common for GE engineers to obfuscate the geometry to protect that IP as well.

As for provenance and reproducibility, the above referenced SE immaturity is of course an inhibitor – there are currently multiple versions of the same code running on those 6-10 machines referenced above, all of which are producing results on which decisions are being

made. Differences in *n*th decimal places are not uncommon for a whole host of build time and runtime reasons. Regression testing using a corpus providing comprehensive coverage is unfortunately not the norm.

Regarding provenance of results outside of HPC applications, this development team has also built a workflow system for heterogeneous distributed computing employing commodity hardware working typically on embarrassingly parallel problems. Applications are referenced in a metadata-rich repository and are versioned, as are the inputs and the outputs and all are recorded in a hierarchically navigable audit trail. While the runtime is not containerized and thus variations in platform can happen over time and therefore do impact reproducibility, from the application tier's perspective, provenance is being tracked.

It is our intention to bring this same provenance capability to HPC workflows in the coming year.

3 - Infrastructure Requirements

What infrastructure is required to develop scientific software?

- Access to diverse hardware. Unfortunately, this is not a Java world, builds are not portable, and maintaining MxN application readiness requires a multitude of boxes which are likely often not 100% utilized for this purpose, but when they are used, would prefer to be *fast*. Build environments could be maintained as a shared service.
- Access to scaled hardware. Systems work differently at scale. A DevOps team wanting to maintain MxN readiness will occasionally need to perform tests at scale to ensure the science team can obtain performance at their intended run size. Large scale performance tests do not typically get executed as frequently at normal CI/CD testing, which prefers to be as frequent as practical.
- These infrequent scaled tests might include big data as well.
- When arriving on a shared machine for build and execution, it helps to have consistent access to the same *a priori* tooling – compilers, package managers, libraries, SE tools, and the like. We're encouraged by the new "reuse" capability in Spack v0.17 which looks to handle both builds from source as well as hopefully easier installs from binary. With perhaps the occasional cutting-edge differentiator, when arriving on a shared platform it would be ideal to just build my app, not all the (in many cases) routine dependencies too.

4 - Developing Community Software

As a software engineer, I have on several occasions produced an SDK or API as the product – some layer upon which we intend other programmers to build. This is akin to writing software for community use. At some level, there is no extra effort – all code of all kinds should be

version controlled, requirements tracked, test cases produced, peer reviewed, documented, etc. Scientific software, even that produced for internal use, doesn't earn a free pass on this for many reasons, scientifically chiefly among them being the reproducibility question.

Code produced by community does come with some additional challenges it seems. A well-maintained community package would have many contributors, and team size increases communication complexity. Requirements and new features get decided by committee not fiat, and while that might add resiliency, it also adds time and churn.

What are the impediments to contributing to community software? Working at a corporate facility, security and IP concerns are probably the largest factor – the preference to keep software for self as a competitive advantage versus sharing it with others.

5 - Workforce Sustainment

What challenges exist to recruiting and retaining scientific computing talent?

- Scientific computing – formally as an engineering discipline – is not generally taught in schools. Knowing Fortran or Python does not make you a software engineer.
- Software engineering itself, even to computer science majors, is often an elective as are courses such as numerical methods, parallel computing, and the like. Thus, finding early career people with the right skills is a challenge.
- The tooling and approaches to scientific software are vastly different – harder, denser, less reliable – than those used in general industrial software development. The productivity of even experienced general software engineers when they initially make the transition to scientific software can be poor. The bar and barrier to entry are very high.
- Scientists, as a breed of persons, are distinct from for example business salespersons. The culture of science and scientists can be foreign to people who are not native scientists. Finding a good scientific software developer to fill a role includes many unique non-technical aspects.

Regarding hiring from a diverse population – the above issues are only compounded among the traditionally underserved – the pool of available qualified candidates even further reduced.

The educational system owns this part of the problem – it must produce candidates with the right resumes. On another side of the issue, employer consumers of these resumes must make it clear – via compensation, acknowledgements, career trajectories, etc. – that scientific software done well is valuable, encouraging new and transitioning careerists to invest themselves in the available training.

Government, understanding the strategic imperative, can help in the form of educational and business grants, imposing standards on new government contracts including time grants on

leadership-class machines, and helping get the message out to the young people about the bright futuristic career prospects for those skilled in scientific software, especially among those traditionally overlooked.

6 - Technology Transfer

The above referenced SE process deficiencies and inhibitors limit the breadth and speed of technology adoption. Thus, after some years of observation, and a year of actual hands-on experimentation, the number of technologies which have been newly adopted from the ECP E4S and other similar community sources has been limited to: Spack, ParaView etc., HIP, GitLab CI. If we also consider the list of packages which were used prior to this project's exploration (listed above), in balance we see a few things in common:

- Open source, or at least open core, allowing free initial experimentation
- An active communicating team of developers, in diverse numbers
- Good documentation
- The option of purchasing enterprise features and support

Spack is perhaps not yet at the level of enterprise-supported, but that might be the current exception. Even HIP, while open source, has the backing of a major company. Without the support of financial sustainment, to fill in the gaps which the community does not produce for itself, interest can wane. Alternatives arise, the prospective of new work becomes more interesting, and packages become stale. So, early adopters beware.

7- Overall Scope

The section in the RFI "Proposed Scope" seems reasonability comprehensive and we'll not alter much other than to say the training topic might emphasize the gap in software engineering process.

8 - Management

The ECP E4S standards are a good starting point. Packages which carry this "government seal of approval" adhere to certain guidelines – they use source control, they have documentation, etc. We'll not list the guidelines here, nor suggest they are complete as currently formulated.

Perhaps even better is when the organization bestowing the seal of approval becomes a self-regulating consortium, one which can perhaps respond to the needs of the market in a manner more quickly than government, and yet, government can play an important role in initiating, engaging, and supporting the activity.

Tangentially, this author has argued in other places that the government might *require* users of government compute facilities to utilize (generally, in an agile way) certain basic SE processes be utilized as part of any grant of compute time.

9 - Assessment

What kinds of metrics are useful for measuring success of the stewardship effort? Metrics have always been challenging in the domain of software engineering process improvement. Many organizations find it best to defer formal measurement until after some initial iterations of improvement. Even after time, choosing the right denominator, the in "wins / defects per opportunity" sense, is difficult. What is an opportunity?

In early stages of a process maturity campaign, the first step is often to cheerlead and form a coalition of the willing, to then make early gains in areas which are broadly agreed to be most important and impactful – 80% of the benefit might be gained by focusing on the obvious low-hanging 20% of the problems. Afterward, to go after more latent or difficult improvements, measurement can be used as a guide, to add scientific formalism to the art of process improvement. Walk then run.

Conclusion

The author thanks the committee for the opportunity to comment and hopes the content has been on point and useful for its intended purpose.