Stewardship of Software for Scientific and High-Performance Computing

Ryan M. Richard[*]
email: rrichard@ameslab.gov

Zachery Crandall[*‡]
email: zachcran@iastate.edu

Hubertus van Dam[†]
email: hvandam@bnl.gov

Niranjan Govind[Δ]
email: niri.govind@pnnl.gov

Karol Kowalski[Δ]
email: karol.kowalski@pnnl.gov

Theresa L. Windus[*‡]
email: twindus@iastate.edu

[*]Ames Laboratory. Ames, IA 50011
[†] Brookhaven National Laboratory. Upton, NY 11973
[Δ] Pacific Northwest National Laboratory, Richland, WA 99352
[‡]Iowa State University. Ames, IA 50011

December 13, 2021

1 Organization Background

The present response is the collective view of several NWChemEx team members. Our team is culturally diverse, spread out across a number of Department of Energy (DOE) National Laboratories and academic universities, and has expertise in a number of academic fields (theoretical and computational chemistry, computer science and engineering, and applied mathematics). The NWChemEx package is developed for the domain of computational chemistry, initially focusing on linear-scaling, high-performance computing (HPC) implementations of electronic structure methodologies. Of note, NWChemEx was chosen to be one of the initial exascale software applications developed as part of the DOE-funded Exascale Computing Project (ECP).

Since the original NWChem package, and continuing now with NWChemEx, our team has demonstrated success for developing HPC software. Our success has relied, and continues to rely, heavily on collaborations with other DOE-funded HPC software packages (*e.g.* Global Arrays), as well as access to DOE Leadership Computing Facility resources (both hardware and personnel). We anticipate the future successes of NWChemEx to continue to hinge on the funding, support, and maintenance of DOE-funded HPC-software and Leadership Computing Facilities.

2 Software dependencies and requirements for scientific application development and/or research in computer science and applied mathematics relevant to DOE's mission priorities

2.1 Software Packages

We have built NWChemEx to take advantage of excellent libraries that are currently available and those that will be coming available in the future by taking an approach that separates concerns and uses modules as much as possible. This means that we have quite a few dependencies, and this will likely increase as we develop the software further. This first list contains dependencies that are broadly used outside the computational chemistry community and that we require be available and operating with high performance and portability. We anticipate needing these libraries for the lifetime of the project. These all need to evolve as the hardware and software ecosystems evolve:

- BLAS
- Boost
- LAPACK

- MPI
- ScaLAPACK
- UMPIRE
- PARSEC (This is a future need for the project, and we anticipate needing it in the next two years or so for the ability to build execution graphs and execute them in an efficient, high performance manner.)
- Doxygen
- Sphinx

In addition, we depend on libraries that are developed and/or used primarily within the computational chemistry community, since most of the libraries in the previous list are very low level and are not easy for most domain scientists to use. The highest risks with these libraries are that they will not continue to be supported and improved for new architectures. Our mitigation approaches have been to have APIs for these so that other appropriate libraries can be incorporated if they have better performance, portability, or support. However, TA and TAMM are harder to isolate and so support for that software is essential to our work. The following list is the libraries used primarily within the computational chemistry community that we depend on:

- Global Arrays (we are currently working to also use UPC++)
- MADNESS (this is a futures based runtime environment)
- Tiled Array, TA (this is a distributed tensor array library for dense and sparse matrices on CPUs and GPUs)
- Tensor Algebra for Many-body Methods, TAMM (also a distributed tensor array library for dense and sparse matrices on CPUs and GPUs, but with a different approach)
- A variety of on-node CPU and GPU tensor libraries that include very fast tensor transposes (cuTT, hipTT, LibreTT, and vendor available libraries)
- CMakePP (this is an object-oriented build library that sits on top of CMake to facilitate large project compilation)
- Cppyy (this is an alternative to Pybind11 and Boost Python)

2.2 Languages and APIs

We rely on multiple software languages and associated tools:

- C++
  - Key capabilities: Object-oriented, high-performance language with good ECP support
  - Missing capabilities: Standardized build system, standardized package manager, language reflection
  - Time frame: Currently being used and will continue to be used for the foreseeable future.
  - Risks: Libraries for more niche functionalities can be hard to find, build system complexity can deter users, and language complexity can dissuade developers. We are mitigating build system complexity with CMakePP and language complexity through a Python API.
  - Scope: NWChemEx is primarily written with C++.
- CMake
  - Key capabilities: Most used C++ build system, cross-platform support, feature-rich
  - Missing capabilities: Not user-friendly, verbose, complexity leads to fragile build systems
  - Time frame: Currently being used and will continue to be used for the foreseeable future.
  - Risks: Official support is slow (mitigating with CMakePP).
  - Scope: Via CMakePP, CMake is ultimately the build system used by all of NWChemEx.

- Python
  - Key capabilities: One of the most used languages in the world, many workflows rely on it, user-friendly, reasonably performant, and many modules
  - Missing capabilities: Performance and usability with complex software stacks on HPC systems
  - Time frame: Currently being used and will continue to be used for the foreseeable future.
  - Risks: Limited HPC support
  - Scope: Public API of NWChemEx is fully exposed via Python.
- CUDA, HIP, and SYCL
  - Key capabilities: Enables use of GPUs (some are vendor-specific while others are not, but we currently need all of them for performance portability)
  - Time frame: Currently being used; in the future they will still likely be used, but ideally as a backend to a more unified programming API.
  - Risks: Compared to CUDA, HIP and SYCL are relatively new; this means there are performance issues and missing features. Committing to one of these solutions can be problematic when the performance is not there for different architectures.
  - Scope: NWChemEx is dependent on these languages for the most time-consuming operations of the software and likely will be for the foreseeable future.

3 Infrastructure requirements for software development for scientific and high-performance computing

Most NWChemEx developers develop the majority of their code locally, requiring no special infrastructure aside from a standard workstation. Once code has been tuned locally, development continues at DOE Leadership Computing Facilities, which we have access to as part of the ECP. The remainder of the infrastructure needs of the NWChemEx project are satisfied by GitHub and GitHub's underlying infrastructure. The following list describes the infrastructure needs of the NWChemEx project in more detail.

- GitHub
  - Key capabilities: version control, project management resources, source-code hosting, documentation hosting, continuous integration (CI) on standard hardware (although with cost), social coding features (pull requests, issue tracking, discussions, wikis, etc.)
  - Missing capabilities: Easier integration with DOE infrastructure
  - Time frame: Currently being used and will continue to be used for the foreseeable future.
  - Risks: Could remove features we use or raise costs (mitigation: could switch to GitLab or BitBucket).
  - Scope: All the NWChemEx project's CI and project hosting (source code and documentation) is done through GitHub.
- Validation tests on standard hardware
  - Key capabilities: Ensures our software continues to compile and run on standard platforms (Mac, Windows, Linux) for research that does not require HPC capabilities.
  - Missing capabilities: Currently have low hardware coverage in our tests. This is largely because of the cost of running CI on non-Linux hardware, and a lack of personnel time to implement the tests.
  - Time frame: Currently being used and will continue to be used for the lifetime of the software.
  - Risks: Rising CI costs (mitigation: could use local hardware), time and money costs associated with maintenance
  - Scope: Currently have CI tests, through GitHub, on hardware running Linux. We intend to add additional hardware coverage as time and money permit.
- Validation tests on HPC hardware

  - Key capabilities: Ensures our code continues to compile and run on the actual HPC hardware. The hardware should include the hardware of all current primary architectures. At the present this means CPUs and GPUs from: Nvidia, AMD, Intel, and IBM, as well as ARM-based architectures.
  - Missing capabilities: Ability to easily integrate these tests with our current CI infrastructure and have results available to the whole team to fix issues.
  - Time frame: Currently being used and will continue to be used for the lifetime of the software, but the exact tests will change as the hardware changes.
  - Risks: losing access (mitigation: could procure individual workstations to continue access when a genus of hardware disappears from the primary hardware pool)
  - Scope: Currently, we manually run such tests on very select algorithms, but we ideally would like to automate these tests with better test coverage.

- Performance testing on HPC hardware
  - Key capabilities: The occasional (on account of the cost) need to ensure that contributions have not compromised performance.
  - Missing capabilities: A well-defined way to do this without exhausting resources (both human and computing), and an application programming interface (API) to allow this to be interfaced with our CI.
  - Time frame: Currently being performed in a limited capacity and will continue for the lifetime of the software, but a particular test will continue only as long as the hardware is available.
  - Risks: Amount of computational time needed to do at-scale testing (mitigation: settle for performance testing on the largest available cluster)
  - Scope: NWChemEx developers of HPC algorithms currently manually test the performance of their algorithms. For resource cost reasons, once an algorithm's performance is acceptable, further tests are usually not performed. We would like to have CI continue such testing.

4 Developing and maintaining community software

Going from an in-house research code to community software is non-trivial. In-house research code tends to be poorly documented (word of mouth instructions suffice), somewhat buggy (bugs are usually just worked around, rather than fixed), and can lack useful APIs (hacking the code to change parameters/algorithms is often considered acceptable). Even on-boarding a single external developer can require a significant amount of time. In our opinion community software must: have reasonable defaults, be relatively full-featured, relatively easy to use for non-experts, hardened (few bugs and work reliably), have API documentation, have tutorials for common use cases, be well packaged (installation should be as painless as possible), and be well supported (answering user questions, clarify documentation, take bug reports, etc.). The following responses assume our definition of community software.

- How much effort to move beyond specific research/internal usage?
  - Initially transitioning NWChemEx to community software will likely require at least 3 to 5 full time equivalents (FTEs). After the initial effort, we estimate that 2 to 3 FTEs should suffice to maintain the software. These estimates do not include effort to do performance tuning or develop new features, only to interact with the community.

- Which tasks are the largest contributors to the needed effort?
  - Increased human interactions. Whenever multiple people interact, there are philosophical differences, different agendas, etc. which must be worked through.
  - Platform support. The more people use the software, the more hardware setups, and build problems become an issue.
  - CI hardening. As more people become interested in the software, you get more contributions. Automation is essential to deal with influx of contributions, but must be developed.
  - Documentation. With a large community, the primary resource should be the documentation.
  - Tutorials. Essential since you cannot manually onboard everyone.

  - Packaging. If you want widespread adoption, the software must be easy to obtain and install.
  - Connecting frameworks. Development of interfaces between ECP software within established codes in the quantum chemistry community (for example, NWChemEx with NWChem, GAMESS)
  - Workflows. To enable ease of use of the software package(s), clear workflows (for example, Python-based) are needed.
  - Project-management. Coordinate/organize developer activities, tracking bugs, triaging issues, mediating conflicts, etc. requires time.
  - Error checking. Many research codes do not robustly validate user input, or worry about edge cases. With a community code, you need to anticipate bad inputs and catch them.

- Largest non-monetary impediments?
  - There are no publications or recognition for most of the aforementioned tasks.
  - Often times there is coupling between the domain and the APIs, meaning whoever works on extending the software to a broader community, needs to know some complexity of the domain.
  - The ability to understand your community and to be able to lead them and anticipate what drives them.

- How is it currently funded?
  - Very rare for it to be funded directly.
  - Happens as part of new software/theory development, although not necessarily to the level that is required.
  - Realistically, these efforts require multiple full time positions and need to be funded as such.

- How does this compare to the level needed to maximize impact?
  - We currently do not have funding to move from research to community software. Ideally, we need at least three FTEs to approach our maximal impact.

5 Challenges in building a diverse workforce and maintaining an inclusive professional environment

The NWChemEx project has experienced a number of challenges in building and maintaining talented professionals. Briefly these are:

- Funding. Many graduate students in our field end up going to tech as opposed to staying in academia or the National Laboratories. This is primarily because they start at two to three times the salary of a distinguished scientist at a national laboratory.
- Perks. DOE does not offer many of the perks that are commonly associated with large companies in the technology industry. These benefits, such as cafeterias, game rooms, snack bars, bring your pet to work days, day care, maternal/paternal leave, gyms, mental health days, and wellness stipends, are a very successful method to draw in and retain talented young professionals.
- Funding. Academia and the National Laboratories are fundamentally grant-driven. By contrast, industry tends to have secure funding.
- Academia and National Laboratories tend to claim their cutting-edge research portfolio as their main advantage. National Laboratories additionally can appeal to a sense of patriotism. However, as evidenced by ECP, this is no longer always the case as DOE often works hand-in-hand with a number of tech companies.
- Universities have a steady stream of students coming through who participate in research and development as part of their education. DOE laboratories do not have these students coming through by default. Nevertheless, getting students working on/with your code is essential to move this knowledge into the community.
- Diversity challenges can be difficult since by the time they are ready to enter the workforce in postdoc or staff positions, we have already lost a lot of diversity in the candidate pool. Research has shown that connecting younger students (especially middle school and high school) with mentors and opportunities to participate in STEM activities is a viable path toward helping with this issue.

6 Requirements, barriers, and challenges to technology transfer, and building communities around software projects, including forming consortia and other non-profit organizations

First, it is useful to address some successful examples:

- MPI. There is a large research activity around the parallel programming that MPI supports, including research implementations of the recommendations in MPICH, OpenMPI, and MVAPICH. This research activity is paired with industry-based activities. The industry partners need MPI to support their products. This motivates them to contribute to the standardization efforts.
- Linux. Around Linux, there is an active research community looking at various operating systems issues and moving that knowledge forward. From industry, there is a need to provide robust operating systems on Linux servers. This encourages industry partners to engage in the Linux development.
- CMake. CMake started as the build system for VTK being developed by a single research company for that sole purpose. However, it met the need for a more general multi-platform build system. It has since been adopted by a number of industry partners (not the least Microsoft's Visual Studio) as a key development tool.

Clearly, there is a path to success when there is a research activity that directly feeds into an immediate business need. The issue with computational chemistry (and other HPC application domains) is that such a direct connection does not obviously exist. This means that industry support for maintenance work and basic training is limited. One could try to change this. One model would be an analogy to services like GitHub. GitHub provides a collection of services that together provide an effective software development platform. With sufficiently large numbers of users, this platform can be provided as a service at a modest cost such that duplicating these capabilities is not worth the effort. Would it be possible to provide computational chemistry services in such an integrated way that trying to locally replicate such services from open source software would not be worth the hassle? Some services are trying to do this (e.g. Crunchyard, https://www.crunchyard.com/), and some have been doing this for years. However, the computational chemistry market is small, and many research institutions have access to cheap labor (e.g. students) to make replicating services affordable. A key change in this situation might the emergence of machine learning models. These approaches need vast amounts of training data, and computational chemistry models can be deployed to generate this data. However, most of the research is currently focused on the machine learning models.

Investment in setting the computational chemistry infrastructure up is also increasingly a burden without a reward, thereby making the case for buying this service rather than developing it in house. This could be a commercially viable path to providing computational chemistry as a service. In such an ecosystem, the commercial efforts could be a source of funding to take care of the software maintenance aspects that research funding does not support. Additionally, training for how to use computational chemistry infrastructure most effectively could also be a market. Currently, there are various platforms for online training that could be used (e.g. Udacity, https://www.udacity.com/us) to provide education to a large community at a competitive price.

7 Management and oversight structure of the stewardship effort

Most scientific software applications are domain specific. It is infeasible for a single person, or even a few people, to manage a huge portfolio of unrelated scientific software packages. For this reason, management needs to be done using some sort of tiered structure. One approach might to have some centralized support for computer engineering tasks such as build systems, CI, etc with dispersed support from the expert software developers who know the science andd algorithms in detail. These two support systems would need to work together in partnership to enable stewardship of the software.

8 Assessment and criteria for success for the stewardship effort

- Perhaps the simplest metric is a "customer" satisfaction survey that focuses on: ease-of-use, documentation, and support quality. This provides immediate feedback from the community about how well a package is being maintained.
- The number of active developers. Generally speaking, projects with more active developers tend to be more heavily used.

- Downloads and other user metrics. These can be spoofed, so they may be deceptive. When viewing these metrics, the expected audience size should also be considered; 100 active users is a more impressive audience size for a chemistry code than it is for something more widely used, like the entire C++ language.
- Activity in repositories showing improvements in the software.
- New theoretical and algorithmic implementations in. the software.
- Science papers, both the papers themselves and citations to those papers, that use the software.