Sustainable Software for Computational Chemistry and Materials Modeling

Daniel Crawford (Virginia Tech), Robert Harrison (U. Tennessee), Anna I. Krylov (U. Southern California), Theresa Windus (Iowa State), Emily Carter (Princeton U.), Edmund Chow (Georgia Tech), Erik Deumens (U. Florida), Mark Gordon (Iowa State U.), Martin Head-Gordon (U. C. Berkeley), Todd Martinez (Stanford U.), David McDowell (Georgia Tech), Vijay Pande (Stanford U.), Manish Parashar (Rutgers U.), Ram Ramanujam (LSU), Beverly Sanders* (U. Florida), Bernhard Schlegel (Wayne State U.), David Sherrill (Georgia Tech), Lyudmila Slipchenko (Purdue U.), Masha Sosonkina (Iowa State U.), Edward Valeev (Virginia Tech), Ross Walker (San Diego Supercomputing Center)
*Corresponding author: sanders@cise.ufl.edu

***Abstract*—Computational chemistry underpins a broad array of vital scientific applications and is one of the pillars of computational science. However, the field also faces some of the most daunting software challenges — perhaps greater than any field of computational science. This position paper describes some of the main challenges inhibiting the development of sustainable and reusable software in computational chemistry and materials modeling. The authors are participants in the recently established Scientific Software Innovation Institute for Computational Chemistry and Materials Modeling ($S^2I^2C^2M^2$).**

I. Introduction

Computational chemistry underpins a broad array of vital scientific applications such as designing more efficient combustion systems, drug design and development, understanding biological systems, semiconductor design, water sustainability, advanced materials, and CO_2 sequestration. It is one of the pillars of computational science, and its impact reaches well beyond chemistry into biomolecular and polymer physics, materials science, and condensed-matter physics. Over its long history, which stretches back more than half a century, computational chemistry has achieved a much-deserved status as a full partner with experiment in scientific discovery, yielding simulations of such high accuracy that its predictions of a variety of molecular properties may be considered "computational experiments," often with greater reliability than laboratory measurements for many chemical properties. However, computational chemistry also faces some of the most daunting software challenges — perhaps greater than any field of computational science.

The history of computational chemistry endows the field not only with great experience, but also with a legacy of diverse and complex code stacks. Many molecular dynamics and quantum chemistry programs involve hundreds of thousands to even a million lines of hand-written code in a variety of languages, including Fortran-77, Fortran-90, C, and C++. While this complexity has arisen naturally from the intricacy of the problems these programs were designed to solve, it also presents a crucial obstacle to the long-term sustainability and extension of the software on ever-changing high-performance computing hardware. For example, while quantum chemistry has produced some of the most accurate and reliable computational models of molecular properties, such highly accurate models have yet to take full advantage of modern high-performance computing architectures. Code complexity also hinders the education of the next generation of computational chemists. Many incoming chemistry graduate students have little programming experience, and they frequently lack advanced math skills. This often makes it difficult for them to get up to speed in theory and code development.

The authors of this position paper are participants in the recently established Scientific Software Innovation Institute for Computational Chemistry and Materials ($S^2I^2C^2M^2$), one of the goals of which is to overcome these obstacles of both algorithms and culture and change the fundamental nature of computational chemistry software development. The team includes computational chemists, computer scientists, applied mathematicians, and computer engineers to attack the fundamental problems of software complexity and education. A year-long conceptualization phase is expected to be funded by NSF.

II. Major Challenges

Computational chemistry faces a broad array of software challenges. Advanced molecular models exhibit astonishing levels of complexity that can easily eclipse the computing resources of even the world's most powerful supercomputers. Faithful and robust molecular dynamics simulations of physically realistic, large-scale chemical problems in heterogeneous catalysis, drug design, or carbon capture will involve exabytes of output data that must be quickly mined and visualized. The high-order polynomial scaling of the current generation of hyper-accurate electronic structure models required for the design of advanced materials such as future organic photovoltaic cells can demand ultrafast computing and communication resources, such as the exascale parallel architectures anticipated in the coming decade. The level of accuracy and scaling

needed for computational chemistry to face these scientific Grand Challenges requires innovative algorithms, software development techniques, and a new approach to education that are far beyond the current state of the art.

A. Current State of Computational Chemistry

The history of computational chemistry software development reaches back to the earliest days of computing, and the legacies of many modern molecular dynamics and quantum chemistry packages span decades. These programs[1], which include both open-source and commercial licenses, involve many hundreds of thousands to even a million lines of hand-written code in languages such as Fortran-77, Fortran-90, C, and C++ (some using several such languages). Such diverse code stacks require vast numbers of human hours and teams of many programmers to develop and maintain. Their effort has resulted in an array of new software technology, including both domain-specific code such as multi-dimensional integral engines and general-purpose algorithms such as the Davidson method for computing eigenvalues of large matrices (some with ranks in the tens of billions).

The Gordian complexity of computational chemistry programs is a natural reflection of the intricacy of the problems they are designed to solve. Modern high-accuracy *ab initio* quantum chemical simulations, for example, involve vast numbers of wave function parameters that must be computed and stored, with computational scalings of $\mathcal{O}(N^7)$ (or even higher), where N represents the size of the molecular system (e.g., numbers of atoms, electrons, or basis functions). As a result, much of the computational chemistry software development in the last 30 years has focused on shared- and distributed-memory parallel architectures, with varying levels of success. Programs such as NWChem, MADNESS, and MPQC were originally designed for parallel computing systems, while most others have been adapted or extended over time to try take advantage of new hardware as it has become available to the developers. A few development groups have intentionally made extensive software modifications to take advantage of high performance computing, with GAMESS, MPQC, ACESIII, and Molpro serving as prime examples.

In addition, the majority of the software development in computational chemistry has been carried out by graduate students and postdoctoral associates, some of whom plan their own careers in academia and therefore are committed to the long-term success of their efforts. Although few of the students involved in this work over the decades have had significant formal education in algorithm development and software engineering, the training of this students within or among theoretical chemistry research groups can be exceedingly productive, since this learning process occurs in the context of the research to be performed. On the other hand, the expectation that such vital education be carried out primarily by individual chemistry research groups — many of which are relatively small — places a heavy burden on their shoulders. A community-oriented approach to the training of new students in computational science has the potential to yield great rewards for both students and software.

[1]Examples include Gaussian, AMBER, Q-Chem, GAMESS, PSI, MPQC, NWChem, CHARMM, COLUMBUS, Molpro, Jaguar, Turbomole, CFOUR, ACES3, Terachem, and many more.

III. Overcoming the Barriers

In this section, we discuss the highest priority issues for overcoming the barriers to sustainable software in computational chemistry and materials modeling.

A. Portable Parallel Infrastructure

Software developers will need to navigate the next decade of rapid change in computer technology that impacts everything spanning from laptops in the classroom, through college clusters, to extreme scale supercomputers. The relevant technology trends to consider are:

- Massive concurrency within a single socket evidenced by multi-core processors (e.g., currently 128 x86 thread units or more on single chip) with increasingly long vector units (e.g., 512 bytes on Intel MIC), and many-core light weight processors (e.g., currently over 10k+ on a chip).
- Massive numbers of sockets in the largest supercomputers implying that bleeding-edge science must coordinate over 10^9 threads possibly with applications needing for the first time to tolerate faults in the system.
- Complex memory hierarchies including limited coherence, partitioned address spaces, and software managed memories.
- Memory and communication bandwidth and capacity (not FLOP/s) being the primary constraints on performance, cost, and energy consumption.

Current programming models in computational chemistry are mostly designed for machines of the past and have not kept pace with the underlying technology, and more recent models such as CUDA are suitable only for a small subset of applications with appropriate massive, uniform parallelism. Powerful new concepts for asynchronous distributed computation, such as found in the DARPA HPCS languages (Chapel, X10, Fortress), are often poorly understood by the campus computational community and, moreover, are only part of the required solutions. Work is needed to frame and answer key questions concerning sustainable programming models and tools in three different contexts

- *Sustainability of large and widely disseminated chemistry codes.* These codes that enable most computations in chemistry and likely will run on leadership class machines.
- *Sustainability of software developed by smaller research groups.* We need to understand appropriate programming models and tools, and how the community and its software can be better organized to accelerate the testing of new ideas and techniques at sufficient scale to determine their worth, e.g., by being able to write code or integrate into existing software to enable facile motion up the Branscomb pyramid.

- *Education* How we should be preparing our students for 2020 and beyond? Certainly this must include the programming tools and models, but more generally must factor in a multidisciplinary foundation to computational science that will enable future researchers to reason about and manipulate software with greater confidence and facility that is presently the case.

B. General-Purpose Tensor Algebra Algorithms

Tensor algebra is ubiquitous in many areas of science and engineering. Whether computational chemistry delivers on the promise of *chemically-predictive* simulation hinges on finding new approaches to computing with high-dimensional tensors. Current software limits tensors to eight or fewer dimensions while emerging methods will require functions to be modeled in $3N$-dimensional Hilbert space, where N (the number of electrons) may be $\mathcal{O}(100)$ or more. Direct storage of and computation using such high-dimensional objects leads to unphysical scalings that are high-degree polynomial (e.g. $\mathcal{O}(N^7)$ of accurate coupled cluster methods) or even exponential. Thus, tackling emerging scientific questions with such computational approaches is not feasible.

To develop electronic structure methods of the necessary scope and fidelity we must tackle several challenges:

- Take advantage of low-rank representations for functions and operators using various known and emerging techniques.
- Develop robust implementations of these algorithms on massively parallel platforms;
- Standardize data structures and algorithms, APIs, software elements, and frameworks;
- Automate the derivation, transformation, and implementation of tensor expressions.

Developing a common framework of reusable software elements for tensor computation will require not only efforts of computational chemists, but also engagement of computer scientists, engineers, and applied mathematicians. Such cross-disciplinary collaboration is the only way to address the critical software issues of a general-purpose tensor algebra infrastructure, such as composability and sustainability (via domain-specific languages, runtimes, and compilers) as well as robustness (via static and dynamic algorithm analyses).

C. Protocols for Information Exchange and Code Interoperability

One of the greatest impediments for the sustainable, innovative software development in the computational chemistry community is the lack of information exchange and code interoperability. Chemistry software developers have, in the past, been more competitive than collaborative, and data and software sharing has therefore been very limited. However, growing software code stacks and ever more complicated theoretical methods make maintaining and developing software more intricate, and it is difficult for individual research groups or even small collaborations to make significant progress without substantial duplication of effort. For example, the development of a new orbital-localization method in one program that specializes in coupled cluster theory could also benefit a different code that focuses on multi-reference methods. However, due to lack of compatibility between the programs, the new method may need a completely new implementation to be used in the second code. Not only does this impede the users of these codes from accomplishing new science, it is also a waste of human labor — a limited commodity in today's funding environment.

While these two areas — information sharing and code interoperability — are intimately linked, there are significant challenges associated with each separately as well. For example, information sharing is not only associated with the data that needs to be exchanged between codes, but is also a broader topic for storage of results that users need. With the explosion of computational resources, huge amounts of data are being produced with only limited data mining efforts. The lack of standards (or methods to convert between different standards) for the storage and limited availability of the data hampers the use of tools that work with data from multiple sources.

From previous work in data and software exchange, it is clear that various protocols will need to be examined and discussed for their use together as well as a universal architecture. For example, several of the software codes have already developed semantics for their data and the data provenance (the metadata that describe how the data was produced, in what format, by whom, and when). While others have not developed a formal methodology, implicit semantics are often in place. It is unrealistic to require all of the software codes to use the same formats, yet too much transformation of data can lead to errors, computational inefficiencies and complexity in the interfaces. The same is also an issue with the code interoperability. Different codes use different languages (mostly Fortran-77, Fortran-90, C, and C++) with different execution models that can be incompatible. In addition, the level of interoperability between the different codes must be established. For example, one could compose coarse-grained applications that use the Hartree-Fock code from one program and the MP2 code from another. One could also compose relatively fine-grained applications that calculate most of the one electron contributions to a Fock matrix in one program, and relativistic and solvent terms in another. A universal architecture that will overcome these and other challenges is needed.

D. Education and Training

Code stacks in computational chemistry programs are exceptionally complicated, with 100k-1M lines of hand-written code and intricate functional logic, sometimes engaging multiple programming languages such as Fortran-77, Fortran-90, C, C++, and python. Such code diversity often seems daunting to new graduate students.

The challenge of introducing students to software development in computational chemistry is made even more difficult by the chemical-education culture in the U.S. Unlike many

other STEM fields, Ph.D. programs in chemistry require only modest coursework in order to introduce students to research as quickly as possible. This approach is reasonable for most chemical subfields, such as synthetic chemistry, for which the undergraduate curriculum produces students well prepared for hands-on laboratory research almost as soon as they arrive on campus. However, such students often have little to no background in programming, leaving many of them relatively unprepared for research in computational chemistry. As a result, research groups that depend on the development of molecular dynamics or quantum chemistry software are forced to adopt *ad hoc* approaches to indoctrination of new students in programming techniques and algorithms, such as localized short-courses taught by more senior graduate students or postdocs. While this approach has a strong advantage in its domain-specific approach, it also can strain the resources of the numerous smaller computational chemistry research programs. Furthermore, in quantum chemistry software development, for example, postdocs educated in other STEM areas — and even in different subfields of computational chemistry — sometimes require a year or more of additional training beyond the Ph.D. before they bear substantial fruit.

Clearly one of the most important activities of the $S^2I^2C^2M^2$ will be the community-wide training and education of students in the fundamental algorithms of computational chemistry and software best-practices. To get around institutional issues, a summer school for teams of graduate students from across the country is planned.

IV. Conclusion

The software and cultural obstacles to sustainable software are formidable, but if $S^2I^2C^2M^2$ is successful, the computational chemistry community — broadly defined — will enjoy open access to robust new software tools and infrastructure, as well as training and education for students and postdocs in algorithms, code standards, and software best-practices. These new tools will enhance and accelerate progress toward atomic-scale modeling of larger and more significant chemical problems. The impact of the $S^2I^2C^2M^2$ will stretch well beyond chemistry, however. If chemistry is the "central science," then computational chemistry is at the core of computational science. Improvements in chemical software and algorithms will yield benefits in biomolecular and polymer physics, materials science, and condensed-matter physics — indeed, all areas of science that rely on atomistic simulations of physical phenomena.

It is *not* the goal of the $S^2I^2C^2M^2$ to produce a monolithic computational chemistry software package to compete with or to replace existing programs. Healthy scientific competition is beneficial and should continue to be encouraged. However, new sets of robust and properly validated software components that can be shared among all computational chemistry programs will serve as — in the words of the late President John F. Kennedy — "a rising tide [that] lifts all boats." This new software will enable the entire spectrum of chemistry codes to take full advantage of existing and emerging parallel computing hardware, to share data and to interact more easily, and to solve complex problems faster and more efficiently.