Parallel programming tools for the future: Perspective from the developers of a big quantum chemistry package

I.A. Kaliman, E. Epifanovsky, A.I. Krylov

Introduction

Continuing advances in computer hardware enable yet bigger systems to be simulated using high-level *ab initio* techniques such as coupled-cluster theory that represents the "gold standard" in quantum chemistry. However, during the past several years increases in single-core computing performance have stalled due to physical limits on transistor sizes and clock speeds achievable at the current manufacturing technology. Even on mobile devices the dominant increase in computational power comes not due to an improved single-core performance but because of increasing number of processing units. Growing interest in energy-efficient computing also contributes to the trend of using many slower but less energy-hungry processors at supercomputing facilities. The state-of-the-art supercomputers have hundreds of thousands of processing cores with projections predicting millions of cores at exascale computing facilities within next five years. Moreover, to achieve even more efficient utilization of computer resources most computationally intensive tasks are offloaded to more energy efficient specialized hardware accelerators. This presents a challenge of adapting high-level quantum chemistry techniques to modern multi-node/multi-core/accelerator-enabled architectures. Here we will highlight some problems of modern parallel tools from the perspective of the developers of a big quantum chemistry package, such as Q-Chem. We will also discuss potential opportunities for future parallel models and environments.

The pace of technology

New technologies are always exciting. But developers of quantum chemistry software are also concerned about the rapid pace of change. Development of a big package take decades and adapting the codes to new technologies like GPGPU or Intel Xeon Phi is not always easy because of the need to learn new programming models, application programming interfaces (APIs), and architecture details of new hardware. The main concern that by the time the code is stable and polished the technology is outdated and one needs to start from scratch. When a new technology arrives many jump on board to use it when the hype is big, only to find out that the code becomes obsolete when the vendor discontinues the product. IBM Cell processor serves as a recent unfortunate example. This innovative CPU was first introduced in the Sony PlayStation 3 console and used later in a number of supercomputers. But in the new generation console Sony opted for a more pragmatic approach with a conventional CPU design which destroys the hope for future widespread use for Cell design. Unfortunately a lot of effort invested in developing new software for this unconventional CPU would not pay off.

Because of the complexity of computational chemistry codes the development of reliable packages takes years. But with current pace the technology can be outdated even before the

implementation is finished. The problem is exacerbated in academic environment where the development cycles are usually much longer than in industrial setting due to funding constraints and personnel (students, PostDocs) turnover. This highlights the need for higher level abstractions for new hardware and parallel programming models. Both for developers and users it is important that the code will work long in the future. We believe that standardized technologies like OpenCL – a framework for writing the programs that execute across heterogeneous platforms – take us in the right direction. The same should be for other parallel tools and environments.

Keeping things simple

One of the major factors contributing to the not-invented-here syndrome in scientific software community is lack of guarantees that new tools will be maintained in future. All too often very promising software is abandoned when the funding runs out or the interests of the developers change to other things. Making the tools open-source with a liberal license and encouraging people from outside to work on them is a possible hedge against the fate of dying. Designing simpler tools which are easy to use and understand should lower the entry barrier for outside developers. The developers of quantum chemistry packages can provide valuable feedback and code to the parallel community.

We believe that having good and concise documentation is as important as having the working tools and standards. Unfortunately, through many years of development both codes and texts become bloated. Message Passing Interface (MPI) standard which is by the time of 3.0 release contains 852 pages and hundreds of functions can serve as an example here. Despite a vast number of functions in the standard for the majority of scientific codes only a handful are truly useful. The focus of the developers of future tools should be not only on cool new algorithms but also on a good yet concise documentation. APIs should be easy to use and understand for a wider audience of scientific application developers. Standard committees should carefully weight how high-level their APIs should be. Low-level APIs can help squeeze every bit of performance from the hardware. Yet they are usually harder to use and are much less stable when the hardware changes. The challenge for developers of parallel tools is to find right balance between the ease-of-use and stability of APIs versus the amount of power the users have in controlling the details.

What should already be there but isn't

All high level quantum chemistry methods rely on manipulation of multidimensional tensors and matrices. Because of huge size of such tensors the naïve in-memory algorithms often cannot be applied directly. The lack of standard library for manipulation of big tensors is apparent. That is why almost every quantum chemistry package has its own in-house tensor library implementation. Q-Chem, PSI, NWChem can all serve as an example. Operations on tensors have no special relationship to computational chemistry and are useful in a number of different fields. A good parallel tensor library could have been developed a long time ago in the spirit of

LAPACK/ScaLAPACK. Yet, no suitable alternative to in-house development is available. We believe that there should be a common library implementing such functionality. But the standard should come not from some governing body but from the implementers themselves, as was the case with FFTW library, which is now a de facto standard for fast Fourier transforms. We want to believe that one of these libraries (or a new one from HPC community) will finally become the standard eliminating the need to maintain in-house functionality that can be shared. In future vendors may provide their optimized implementations which share common interface, as is the case with BLAS/LAPACK.

Truly free license

Software licensing may pose a big problem for commercial application development. "Viral" open-source licenses like GPL force people to share contributions and release the source code back into the community if they want to redistribute software that uses GPL-licensed code. Because of such restrictions GPL-licensed code cannot be used in closed-source packages. Adopting less restrictive licenses like BSD or MIT would benefit the community by enabling commercial software to use future tools and libraries greatly expanding their impact. Even though these licenses do not force people to share the code it does not mean that commercial developers won't give back to the community. For example, many parts developed for Q-Chem, like *libefp* and *libtensor* libraries, as well as IQmol software, are open sourced and available under liberal licenses. As an additional example we may point out to the LLVM/Clang compiler which immensely benefited from being adopted by Apple because of its permissive license.

Fault tolerance from the start

One of the biggest problems of new exascale machines will be inevitable hardware failures during calculations. Software running on tens of thousands of processors may experience multiple hardware and software failures during a single calculation. To our knowledge no widely used quantum chemistry package implements any sort of fault tolerance. That is, if one process crashes the whole calculation has to be restarted from the beginning. Check-pointing might be a good first step towards fault tolerant software, but fine grained checkpoints are not always possible owing to huge tensor sizes in quantum chemistry. This can affect performance negatively as the nodes usually have no local disks and permanent storage can only be accessed on a slow shared file system. Optimal fault tolerant solutions would implement algorithms which are able to restart parts of a calculation in response to a failed process without bringing down the whole simulation. At the same time we believe that scientific software developers should not have to explicitly deal with soft errors like bit-flipping caused by thermal noise or cosmic radiation. These are to be dealt with on a lower hardware and/or software level like it is the case with ECC memory today. Protection against such errors should be extended to other parts and communication channels of the hardware.

Designing fault tolerant quantum chemistry software would require rethinking many algorithms and in some cases radically changing design. The tools and models that will help dealing with this task would be of great importance in future.

Conclusion

In this short article we shared some of the concerns of the developers of a big quantum chemistry package with a broad user base. We hope that this feedback will be valuable to the parallel software community and the points presented here will be taken into account when designing future tools and environments for parallel application development.