

# Extended Abstract: Accessible Peta-scale Computing for the Life Sciences

Christopher Mueller and Andrew Lumsdaine  
Open Systems Laboratory  
Indiana University  
Bloomington, IN 47405  
{chemuell,lums}@osl.iu.edu

The adoption of high-end computational tools in the life sciences creates unique and synergistic opportunities for the life science and computational science research communities. At the same time that chemists and biologists are acquiring powerful new research tools that are fundamentally changing the nature of their work, computational science researchers are presented with new problems that are different from traditional PDE-based computations. In this extended abstract we present an outline for the future of peta-scale computing in life science applications. Based on the authors' experience working with chemists and biologists, we discuss the challenges of integrating traditional approaches to high-performance computing into data-intensive, user-driven environments. In these environments, the scientist's focus is on science, not computing, making it important for the high-performance community to develop tools and techniques to aid and enhance the usage of high-end resources. We see this as an opportunity for the computational science community to combine and extend many of the approaches used in high-performance computing and contemporary software engineering to deliver the applications that will drive the next generation of discovery in the life sciences.

## 1 INTRODUCTION

While many physical science and engineering disciplines have had strong ties to the high-performance research and development communities for years, the life sciences, specifically chemistry and biology<sup>1</sup>, have only recently begun to integrate high-end computational tools into their environments. This has created a unique opportunity for both sides. Chemists and biologists are quickly acquiring powerful new research tools that are fundamentally changing the nature of their work. Computational science researchers and developers are being confronted with a new set of challenges that are different from previous pushes to integrate science and computation. The challenges arise at all levels, from the sheer size and types of the data to the social differences between the 'wet' and 'dry' sciences.

For example, it is well known that the human genome project, probably the most visible large-scale biological computing project, has identified over three billion base pairs in the human genome [?]. Combined with the estimated 30,000 genes and the numerous ways genes interact with the genome and their environment, the cost of running even simple analyses on the full data collections is currently prohibitive.

At the other end of the spectrum, consider the makeup of the research and development communities. For the first time, high-performance computing is faced with a user group with no formal training in programming techniques and algorithm design. The life science's heavy focus on laboratory and experimental methods combined with the dearth of useful computational tools has left little time or reason for computer training. On the other side, computational scientists and developers have very little exposure to the life sciences, with most graduate and undergraduate programs encouraging physics over biology for science electives.

In this extended abstract, we present an outline for the future of peta-scale applications for computational life sciences, with a focus

<sup>1</sup>In this paper, we use the term 'life sciences' to refer to chemistry and biology.

on the challenges of integrating high-performance computing into a data intensive, user-driven environment. Based on the authors' experience working with chemists, biologists, and computational scientists in both industry and academia, we explore what makes working in this field different from traditional high-performance domains. We also present a series of research and development opportunities for bringing the potential of large-scale computation to the life sciences.

## 2 SMALL MOLECULES AND PETA-BYTES

To understand the differences between traditional scientific computing and computing for the life sciences, it helps to take a step back and understand the data, methods, and cultures of chemistry and biology. In a reductionist worldview, mathematics gives the formal methods for describing physics, which in turn is used to model chemistry, which leads directly to biology. Chemistry and biology are connected via the sub-field of molecular biology, which focuses on the chemical processes at the sub-cellular level. Molecular biology is the heart of most modern life science applications, including the various \*-mics fields and the biological side of drug discovery.

Molecular biology is driven by the so-called 'central dogma' [?]. The central dogma states that genomes encode genes which give rise to the proteins that form the basic molecular components on which all other components of life are built. Cells are simply dynamic collections of proteins and small organic molecules. Proteins are the engines that drive cellular construction and communication and use other proteins and small molecules to aid in their tasks. Small molecules provide the connection back to chemistry, where organic chemistry focuses on the study and synthesis of small compounds containing carbon.

At first glance, this appears to be a rather straight-forward set of data types to model. However, between the central dogma and a set of tools for effectively managing, modeling, and mining data, there is the complexity of life and the methods for understanding it. Biological systems are difficult to study. Experiments are difficult to design, are expensive, and can produce tera-bytes of data in a single run. The time scale for scientific research is months and years, and a large amount of effort may yield no results. In addition, only recently have data collections been moved to computers. Traditionally, the literature record has been the primary source of data, with the primary sources being the lab notebooks kept by bench scientists.

The scale of the data and the types of algorithms can be intoxicating for computational scientists. While the genome is well known, a much larger and diverse type of data exists in the databases of pharmaceutical companies. Pharmaceutical companies develop small molecules, commonly called targets, into drugs. For every successful drug, there are hundreds, if not thousands of targets that are studied but ultimately rejected. Modern biotech and pharmaceutical companies have begun collecting the data from all the experiments performed during the drug discovery process, and many academic and industrial labs have recently begun relating that data back to the data collected from genome projects. These data collections tie together all known information about small molecules, genomes, gene interactions, protein interactions, and the toxicity and potency

of many of the targets. It is hoped that these large, integrative data sets can form the basis of the future of drug discovery and development [?]. However, the techniques for processing and interpreting these heterogeneous data collections are in their infancy.

### 3 OPPORTUNITIES FOR SCIENTISTS

Even with a brief introduction to the types of data, it is easy to see that there are significant opportunities for changing the way chemistry and biology are approached using computational methods. However, the nature and scale of the data has made it difficult to provide useful tools. Large data sets take time to process and many of the algorithms used for mining these data sets are still experimental. If it takes a day to process a data set, both scientists and algorithm developers will only perform a small number of runs. Additionally, the current algorithms are heavily parametrized, which makes it difficult to search for good combinations of parameters.

Consider a simple example: clustering chemical compounds based on feature vectors. A compound can be represented by many different features, some continuous and some categorical. It is common practice to roll up features into a bit vector, with a bit on or off if a feature is present or meets a certain threshold. To put some numbers on things, bit vectors, or *chemical fingerprints*, are often thousands of bits long. Compound databases contain hundreds of thousands to tens of millions of compounds and there are dozens of comparison algorithms used for comparing fingerprints. On a modest library of compounds with 100,000 entries and 1000 features per compound, computing the triangular similarity matrix generates  $(n-1)(n)/2 \simeq 10^{15}$  values in a similar number of operations. For clustering algorithms, this step may need to be repeated  $n$  times or more. While clustering is a popular topic for computational research papers [?], the cost of clustering real data and the number of outcomes depending on parameter settings limits its utility in practice.

As integrative data sets become more common, these costs are becoming more evident. However, even traditional high-performance techniques have been demonstrated to bring the performance of some of the analytical kernels from hours to minutes. Peta-scale resources offer the opportunity to bring these times down to microseconds. Instead of waiting overnight for one set of results, a scientist using a peta-scale resource could try out multiple parameter configurations in minutes. The ability to interact with data on such scales will enable scientists to gain a deeper understanding of their data and the algorithms.

Fast, interactive data analysis will not only benefit biologists and chemists, but also more traditional computational scientists modeling systems in-silico. Many computational approaches to chemistry and biology lack the accuracy to be useful by practicing scientists. Even common models, such as molecular docking simulations (e.g., [?], must be backed up by experimental data for scientists to accept them. With new tools for studying experimental data, computational scientists benefit by having large pools of information to develop models and provide real-time feedback steering into computational systems.

### 4 OPPORTUNITIES FOR HPC

Interactive, large data analysis represents one of the best opportunities for high-performance computing in recent history. Industry and application consolidation in the past ten years has focused HPC on benchmark computers capable of performing numerical simulations at near peak performance. These systems are a natural evolution of the original success in HPC — large numerical simulations — and represent the success of the many HPC efforts. However, chemistry and biology present different data and user profiles that will force a shift in the way resources are used and applications developed.

The most fundamental difference between traditional scientific applications and life science applications is that the former are char-

acterized primarily by solving systems of partial differential equations, while the latter are more data driven, with simple computational kernels applied to large data sets. For example, instead of regular structure, life science data collections are often unordered and the integrative data sets take on complicated graph structures. Unordered data sets are tricky in that many algorithms assume a natural order and the results are different based on the order of the data, leading to interpretation problems. Graphs present a challenge in that they are referential structures that make it difficult to exploit cache hierarchies to achieve high performance. Often, the main memory or disk bus, rather than processor capability, determines the effective performance for an algorithm [?, ?].

As mentioned in the introduction, the social structure of the life science community puts more pressure on computational researchers to work effectively with and understand the data and needs of their users. Whereas physicists often have basic programming skills, biologists and chemists do not. Their time is better spent in the lab or designing experiments, not tracking down bad memory accesses (the same could be said for physicists, but as a community they have made the decision to handle a portion of their development needs).

Given these opportunities and constraints, we now present our ideas for what the HPC community can do to become relevant to and thrive with the life science community.

#### 4.1 Hardware

At the lowest level, peta-scale hardware systems need to be designed to work on relational and distributed data, with an emphasis placed on fast interaction between the compute and storage resources. Life science research labs and pharmaceutical companies are characterized by many teams working at different locations on related problems. Hardware systems must support rapid access to data but also provide a good level of client side performance for smaller visualization and analysis tasks. However, current hardware is rarely utilized to its full potential, but for computational tasks and data movement. Thus, as hardware scales to support peta-scale systems, it is much more important that the tools and methods developer's use are effective for delivering high-performance code that fully utilizes available resources.

#### 4.2 Software

The most opportunities come at the software level. Because of the number of algorithms and types of data and the fact that the experts — chemists and biologists — will be the customers rather than the developers, a shift in the way high-performance software is developed is necessary. Current high-performance applications are combinations of algorithms developed in low-level languages and executed by scripts using complicated schedulers. Because the developers are often the scientists, this is a perfectly acceptable and efficient solution. However, as the size of the development teams grow to include at least one member who is not an expert, the development, execution and deployment environments must adapt to all participant's needs.

We suggest focusing on four major areas:

- High-level languages and language features
- Programming models
- 'Micro' development environments
- Developer education

In the next few paragraphs, we describe how each of these can be used to effectively integrate HPC into the life sciences.

**Languages and Features** Scientific computing has been slow to adopt new languages and language features. The conservative approach has paid off by allowing numerical applications to steadily evolve and adapt to new hardware while still keeping the original code bases intact using a limited number of developers. This

approach will not work for life science applications. While most scientific applications are built around a few common kernels, the number of different data types, formats, and algorithms used in the life sciences is already larger and already fragmented. Many of the implementations use high-level scripting languages, mirroring the general industrial trend away from compiled languages for most application development. Rather than continue to require expertise in C or FORTRAN, the HPC community has a unique opportunity to start adapting new languages and techniques for high-performance applications. Techniques such as component systems, object-oriented programming and generic programming have proven to be useful for abstracting different parts of applications. While they all come with well-known performance or usability limitations, they are indispensable tools of modern software engineering. The research challenge for the HPC community is to integrate these techniques into their culture in a way that performance is preserved where it is needed.

As an example of how these techniques can benefit HPC, consider generic programming in C++. While an advanced technique, generic scientific libraries have been shown to achieve performance on par with FORTRAN libraries while handling many more types of data than the original libraries [?, ?]. Recent work with the Parallel Boost Graph Library [?] has even begun to discover techniques for simplifying the parallelization of complex graph algorithms. This allows the application developer to focus on the application, not the parallelization.

Features that effectively separate concerns are especially important for life science applications, where the complexity of the data and algorithms already places cognitive pressure on the developers. As the shift from compiled to scripting languages for application glue has demonstrated, working at appropriate abstraction levels pays dividends in the types of applications developed.

**Programming Models** Language features provide tools for application developers, but high-performance computing, especially data intensive applications, places demands on both the application and developer that tools alone cannot solve. In other fields, higher level programming models, often captured as design patterns or system architectures, provide the intellectual structures necessary to manage complex applications.

To illustrate, consider two models that we have found essential for success in the life sciences: mixed language models and component models. Mixed language development is not new but is especially important for delivering high-performance applications for non-technical users. Scripting and interpreted languages have become the language of choice for user interface developers and have strong support libraries for rapidly building applications. However, their performance suffers on compute intensive tasks. Using high-level languages for interface design and application data flow and compiled languages for compute kernels gives developers a fair trade-off between performance and productivity while enabling them to develop high-performance applications with good user interfaces [?].

Another language commonly used in the life sciences is SQL. Most data is stored in relational databases and accessing it requires working knowledge of SQL. Using a flexible mixed language model also opens up opportunities for using database features for high-performance computations. While common in industry, this specific type of mixed language development is rarely seen in academia.

Component models have demonstrated their utility for general application development and are starting to make waves in the high-performance community [?]. Component models allow developers to deliver application components at the appropriate level of abstraction and often span multiple languages. High-performance components can separate algorithms, resources, and data and have uses not only in application development but also

infrastructure development. Open MPI, an evolution of FT-MPI, LA-MPI, LAM/MPI, and PACX-MPI uses a low-level component system to abstract hardware resources, making it easy to port Open MPI to new hardware with minimal changes to the code [?, ?]. Performance studies have shown this to be an effective strategy, with no performance impact for the added flexibility [?].

**Micro Development Environments** Development environments such as Visual Studio and Eclipse have changed the way developers develop and maintain code, abstracting many of the details of build systems and directory structures from the user and allowing them to focus on the application itself. Recently, special purpose development environments have emerged for specific types of applications. In contrast to the ‘problem solving environments’ of the past that attempt to provide a full solution for all development in a given domain [?], these new ‘micro’ development environments are targeted at a specific development problem and work with existing development processes. For instance, GPU programming has evolved around micro development environments that target fragment and shader programming, algorithms targeted at pixel and geometry data, respectively, and integrate into the developer’s toolbox. Another example is the myriad plug-ins available for the Eclipse development platform.

With more processors sporting multiple cores and SIMD units and many including processor support for threading, there is an opportunity to develop analogous tools for different resource and application domains. For instance, a tool could include thin abstractions for manipulating molecular data using SIMD instructions and generate high-performance object code that can be linked into larger applications. In a recent, unpublished project, we have demonstrated the feasibility of building systems in this fashion, using Python to generate machine code for chemical similarity calculations.

**Developer Education** To fully succeed in the life sciences, the HPC community must start to increase its outreach and training to include developers that generally do not utilize high-performance tools and development models. The pharmaceutical industry is already heavily vested in systems built around databases and high-level languages such as Visual Basic, Python, Perl, and Java. With high-performance techniques developed by expanding research into these languages, the HPC community can gain access to large developer bases. By educating developers on programming models that enable high-performance applications, more developers will architect applications with these models in mind.

This is not simply a matter of ‘build it and they will come’, however. Instead, just as developers writing software for chemists and biologists must work with them, the HPC community must embrace a broader class of developers and provide proven tools and techniques for integrating applications into large-scale computing resources. There is already a strong bias towards productivity over performance in the industry and only through education will developers have the knowledge to know how to access high-performance systems with little impact on their productivity.

As a start, high-performance techniques need to be introduced into not only the computer science undergraduate curriculum, but also informatics programs and business computing programs. Rather than teach HPC as a series of tricks for laying out matrices in memory and arranging calculations, general strategies applicable to a broad class of languages should be taught first, with the focus shifting to the languages used at the institution for advanced topics. For instance, Java is a popular language, but there are performance trade-offs that are made with overly complicated object hierarchies. To turn this into a teaching opportunity, a professor could encourage students to experiment with different methods for modeling the same system and study the performance and programability of each approach for different tasks. This will begin to introduce them to the value of designing for performance when necessary, without

overwhelming them with details of algorithms they will most likely never use in practice.

## **5 CONCLUSION**

Peta-scale computing for the life sciences offers one of the greatest opportunities to the high-performance computing community for expanding its influence. However, real differences between traditional scientific computing and computing for the life sciences conspire to limit the impact HPC infrastructure and techniques will have in the field. Shifting the focus of HPC research and development to more closely match the needs of developers working with chemists and biologists, along with efforts to educate the broader development community on high-performance computing will help HPC transition into the new application areas presented by the life sciences.