In chemistry, as in other domains, the complexity of the software is growing rapidly, driven by a variety of factors. Component-based software engineering is an approach to help manage software complexity and increase productivity by developing software in modular, interchangeable components. The Common Component Architecture (CCA) project addresses this by providing a framework specifically designed for the needs of high-performance scientific computing.

**Sources of Software Complexity**
- Increasing size of mature software yields more and more complex dependencies.
- Doubling the size of existing software requires more and more complex queries.
- Increasing physical complexity of new components leads to more and more complex integration.
- Increasing hardware capability increases the complexity of the hardware, which in turn drives an exponential increase in software complexity.

**Dealing with Complexity**
- Eliminate available complexity.
- Reduce the level of abstraction of the programming model.
- Control the complexity of new components in the programming model.
- Use a variety of design patterns to manage complexity.

**Basic Concepts of Component-Based Software Engineering (CBSE)**
- A component is a software deployment unit. Components have well-defined interfaces.
- CBSE is a software engineering paradigm that supports the development of complex systems.
- CBSE is a way of thinking about software design and development.

**Advantages of CBSE**
- Components may be viewed as a logical extension of both libraries and objects.
- Components encapsulate much complexity into “black boxes”.
- Components may be used without knowledge of the implementation details.
- Components can hide much of the complexity of a deep object hierarchy.
- Component environments can enforce the use of published interfaces (prevent access to internals) – libraries and objects do not.

**Components vs. Traditional Scientific Programming Techniques**
- Components can be used as a logical extension of both libraries and object-oriented programming concepts.
- Components are not typically designed as objects or collections of objects.
- Interface generation/design in C/C++ is not typically considered.
- Libraries are often hard to compose together – components easier to compose.
- Components must indicate how to interface with the external component environment – libraries and objects do not.

**The Common Component Architecture (CCA)**
- CCA has been developed and is now widespread primarily in non-adoption settings.
- CCA has not yet had much uptake in high-performance scientific computing.
- CCA is a component-based software development approach.
- The Common Component Architecture is tailored specifically to the needs of high-performance scientific computing.
- Supports both parallel and distributed computing.

**Language Interoperability with Babel**
- Uses a unified approach to interference.
- Supports user-friendly front-end visualization.
- There is often additional overhead due to design and implementation decisions.

**High Performance and Parallelism**
- Specially designed for the needs of high-performance scientific computing.
- Employs parallel and distributed computing.
- Designed to be implementable with minimal performance impact.

**CCA Performance**
- Cells between components are equivalent to C++ virtual function calls.
- C++ libraries call C++ libraries.
- Translation of data between languages may add overhead.
- Calls within components have no CCA-imposed overhead.

**Conclusions**
- CCA provides modest performance improvements over similar BFGS implementations.
- BFGS-based methods have been shown to perform well in a variety of scientific applications.
- Further development of CCA capabilities is needed to fully realize its potential.
- Component-based approaches are expected to have significant benefits.

**Underlying Software Packages**
- Quake Chemistry MPQC (PNNL) - PETSc
- Optimization: Toolkit for Advanced Optimization (TAO, ANL)
- Linear Algebra: FFTPACK, ScaLAPACK

**Benchmarks Optimization Methods**
- Compare TAO’s limited memory variable metric (LMVM) method to traditional chemistry methods.
- BFGS uses ScaLAPACK without parallelism on each step using current correlation potential.
- Quadratic in number of variables for both operation count and memory usage.
- LMVM uses ScaLAPACK with parallelism (up to 20 in these experiments).

**Number of energized evaluations required with various approaches to converge the structure of four different molecules at the HF/6-31G level from a random start**

**Fundings**
- The work has been supported by the U.S. Department of Energy's Office of Science, Office of Basic Energy Sciences, and Office of Early Career Research (Grant DE-SC0008868). The work has also been supported by the D. H. Dreyfus Foundation Fellowship (T.L. Windus), the National Science Foundation (Grant DMS-1100003), and by the U.S. Department of Energy, Office of Scientific Computing (Grant DE-FG02-99ER25777).