

Component-Based Integration of Chemistry and Optimization Packages: Molecular Geometry Optimization

Overview

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 through reuse and composition of existing components of computing. The Common Component Architecture (CCA) is being developed as a component model specifically designed for the needs of high-performance scientific computing.

In this work, we use the CCA to facilitate the creation of interoperable software components for use in quantum chemistry. We demonstrate the advantages of the component-based software development approach, and use the resulting system to compare traditional approaches to molecular geometry optimization with state-of-the-art algorithms developed by experts in optimization.

Sources of Software Complexity

- Increasing physical fidelity of models yields more expensive and more complicated methods
- Push to larger problems requires more complicated implementations
- Increasing hardware capability increases the complexity of the hardware, which is almost always exposed to the programmer and must be explicitly managed for best performance
- Above sources are *intrinsic* in the modern software development environment. There is often additional *available* complexity due to design and implementation decisions

Dealing with Complexity

- Eliminate avoidable complexity
 - Raise the level of abstraction of the programming model
 - Put more responsibility on programming model/execution environment
 - e.g. Global Array model vs. message passing (MPI) for parallel correlated electronic structure codes
 - e.g. Tensor Contraction Engine (see posters and workshop)
 - Develop & use tools to manage remaining complexity
 - Can often help identify and eliminate avoidable complexity too
 - e.g. Scripting languages, makefiles, "standard" libraries, object models, component models

Basic Concepts of Component-Based Software Engineering (CBSE)

- Component**
 - A unit of software deployment/reuse (i.e. has interesting functionality)
 - Interacts with the outside world only through well-defined interfaces
 - Implementation is opaque to the outside world
- Interface (a.k.a. Port on CCA)**
 - Defines how components interact, distinct from implementation
 - Generally, a procedure interface
 - Some component-like environments are based strictly on data flow (e.g. AVS, Data Explorer, etc.)
 - Leverage C++ abstract class, Java interface
- Framework**
 - Holds components during application composition and execution
 - Controls the "exchange" of interfaces between components (while ensuring implementations remain hidden)
 - Provides a small set of standard, ubiquitous services to components

Advantages of CBSE

- Software complexity**
 - Components encapsulate much complexity into "black boxes"
 - Plug and play approach simplifies applications & adaptation
 - Model coupling is natural in component-based approach
 - Facilitates more scalable software development processes
 - Group comes together to define overall architecture, interfaces
 - Individual teams groups separately design & build components to spec.
- Software reuse and delivery**
 - Provides a "plug and play" application development environment
 - Many components available "off the shelf"
 - Abstract interfaces facilitate reuse and interoperability of software
 - Allows researchers to focus on parts of the problem of particular interest to them, utilizing components developed by others for the rest
 - "The best software is code you don't have to write" [Jobs]
- Software performance (indirect)**
 - Plug and play approach can "mix off the shelf" component library simplify changes to accommodate different platforms

Components vs. Traditional Scientific Programming Techniques

- Components can be viewed as a logical extension of both libraries and object-oriented programming concepts
- Components are typically discussed as objects or collections of objects
 - Interfaces generally designed in OO terms, but...
 - OO languages are not required
 - Component interfaces need not be OO
 - 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 can not
- Libraries are often hard to compose together – components easier to compose
- Components must include some code to interface with the framework/component environment – libraries and objects do not
- Components only exist within a component environment (framework)

Pacific Northwest National Laboratory
Operated by Battelle for the
U.S. Department of Energy



NATIONAL
LABORATORY
UNIVERSITY OF OREGON



Pacific Northwest National Laboratory

Theresa L. Windus
Yuri Alexeev
Carl Fahlstrom
Elizabeth Jurrus
Manojkumar Krishnan
Jarek Nieplocha

Oak Ridge National Laboratory

David E. Bernholdt

Argonne National Laboratory

Jason Sarich
Steve Benson
Lois Curfmann McInnes

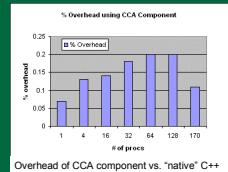
Sandia National Laboratory

Joseph P. Kenny
Curtis L. Janssen



The Common Component Architecture (CCA)

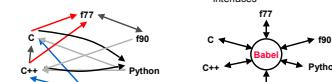
- CBSE has been developed and is now widespread primarily in non-technical areas
- CBSE has not yet had much uptake in high-performance scientific computing
 - Largely due to deficiencies of "commodity" component models for HPC
- The Common Component Architecture is tailored specifically to the needs of the high-performance scientific computing community
- Supports both parallel and distributed computing
- Designed to be implementable with minimal performance impact
- Minimalist approach makes it easier to incorporate existing code into CCA
- Provides language interoperability for important languages for HPC



Overhead of CCA component vs. "native" C++ implementations of a parallel Lennard-Jones molecular dynamics simulation

Language Interoperability with Babel

- Existing language interoperability approaches are "point-to-point" solutions
 - Babel provides a unified approach in which all languages are considered peers
 - Babel used primarily at interfaces

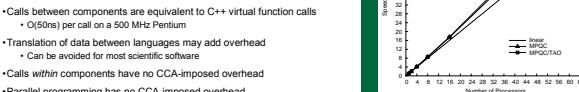


High Performance and Parallelism

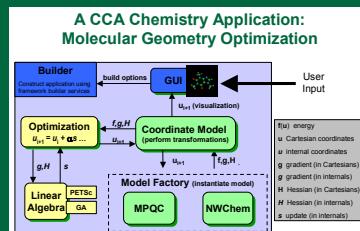
- Single component multiple data (SCMD) model is component composed of widely used SIMD model components
- Each process loaded with the same set of components wired the same way
 - Different components in same process "talk to each other" via ports and the framework
- Same components in different processes talk to each other through their favorite communications layer (i.e. MPI, PVM, GA)
- Framework: Gray
- MCMRD/IMPMD also supported
- Other component models ignore parallelism entirely

CCA Performance

- Calls between components are equivalent to C++ virtual function calls
 - 0.50ns per call on a 500 MHz Pentium
- Translation of data between languages may add overhead
 - Can be avoided for most scientific software
- Calls within components have no CCA-imposed overhead
- Parallel programming has no CCA-imposed overhead
- Advice: be aware of costs and take them into account in design
 - In practice, overheads are negligible



Naphthalene HF6-31G(2df,2pd) Speed-up in MPQC-based Applications



Project Goals

- Move from "proof of concept" stage toward real component-based end-user applications
- Performance evaluation of optimization components
 - Examine efficiency of algorithms in TAO for quantum chemistry
- Further development of optimization capabilities
 - Provide internal coordinate generation, constrained optimization, configurable convergence control
- Graphical user interface to assemble and run applications
 - Provide user-friendly front-end visualization
- Future plans: Exploring chemistry package integration through hybrid calculation schemes and sharing of lower-level intermediates such as integrals and wavefunctions

Underlying Software Packages

- Quantum Chemistry
 - NWChem (PNNL)
 - MPQC (SNL)Optimization
 - Toolkit for Advanced Optimization (TAO, ANL)Linear Algebra
 - Global Arrays (PNNL)
 - PETSc (ANL)

Benchmarking Optimization Methods

- Compare TAO's limited memory variable metric (LMVM) method to traditional chemistry methods (i.e. BFGS)
 - BFGS updates approximate Hessian at each step using current correction vector pair
 - Quadratic in number of variables for both operation count and memory usage
 - LMVM uses guess Hessian and multiple correction pairs (up to 20 in these experiments)
 - LMVM is linear in number of variables for both operation count and memory usage

Number of energy/gradient evaluations required with various approached to converge the structure of four different molecules at the HF6-31G level from a HF/STO-3G starting point.

Electronic Structure Package	MPQC	MPQC	MPQC	NWChem	NWChem
Optimizer	MPOC (BFGS) (LMVM) (BFGS)	TAO (BFGS) (BFGS) (LMVM) (BFGS)	MPQC (BFGS) (LMVM) (BFGS)	NWChem (LMVM) (BFGS)	TAO (BFGS) (LMVM) (BFGS)
Coordinate System	Cartesian	Cartesian	Internal	Cartesian	Cartesian
Initial Guess Hessian	Unit matrix transformed from internals	0.2 * Unit matrix			
Stepsize	4/16	4/4/4	7/5/5	3/3/17	30/30
Glycine (10 atoms)	22/22	26/26	25/13	19/19	59/30
Isoprene (16)	18/18	44/45	78/75	33/37	91/46
Phosphosine (19)	45/45	67/67	85/85	71/36	68/88
Acetylsalicylic acid (21)	56/56	91/97	120/120	36/39	105/105
					196/96

Conclusions

- Optimization
 - LMVM provides modest performance improvements over similar BFGS calculations
 - Having a good initial guess Hessian outweighs LMVM/BFGS differences
 - Cartesian unit matrix (used in most calculations) is not a good guess Hessian
 - Impact greater on LMVM because it is not a Hessian update method

Future plans include
 - Allow LMVM to accept guess Hessian provided by chemistry model
 - Expand to larger problems
 - Benchmark other optimization methods available in TAO

Components
 - Demonstrated benefits of component approach in chemistry applications
 - Interoperability of chemistry, linear algebra packages
 - Ability to easily utilize software written by experts in other areas (TAO, GA, PETSc)
 - Future plans include
 - Hybrid computational schemes, integrating multiple packages
 - Deeper levels of interoperability (integrals, wavefunctions, etc)

CCA
Funding
This work has been supported by the US Dept. of Energy's Scientific Discovery through Advanced Computing Program, the Office of Science, and the Center for Component Technology for Terascala Simulation Software. This research was performed in part using the Molecular Science Computing Facility (MSCF) in the William R. Wiley Environmental Laboratory at the Pacific Northwest National Laboratory. The MSCF is funded by the Office of Biological and Environmental Research in the U.S. Department of Energy. PNLL is operated by Battelle for the U.S. Department of Energy under contract DE-AC06-76RL18375. Oak Ridge National Laboratory is managed by UT-Battelle, LLC for the U.S. Dept. of Energy under contract DE-AC-05-00OR22725.