Component-Based Integration of Chemistry and Optimization Software

Joseph P. Kenny,[1] Steven J. Benson,[2] Yuri Alexeev,[3] Jason Sarich,[2]

Curtis L. Janssen,[1] Lois Curfman McInnes,[2] Manojkumar Krishnan,[4]

Jarek Nieplocha,[4] Elizabeth Jurrus,[4] Carl Fahlstrom,[3]

Theresa L. Windus[3]

[1]*High Performance Computing and Networking Department, Sandia National Laboratories, MS*

9915, P.O. Box 969, Livermore, CA 94551-0969

[2]*Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass*

Avenue, Argonne, IL 60439

[3]*Environmental Molecular Sciences Laboratory, Pacific Northwest National Laboratory, MS-IN:*

K8-91, P.O. Box 999, Richland, WA 99352

[4]*Computational Science and Mathematics, Pacific Northwest National Laboratory, MS-IN:*

K1-85, P.O. Box 999, Richland, WA 99352

Abstract

Typical scientific software designs make rigid assumptions regarding programming language and data structures, frustrating software interoperability and scientific collaboration. Component-based software engineering is an emerging approach to managing the increasing complexity of scientific software. Component technology facilitates code interoperability and reuse. Through the adoption of methodology and tools developed by the Common Component Architecture Forum, we have developed a component architecture for molecular structure optimization. Using the NWChem and Massively Parallel Quantum Chemistry packages, we have produced chemistry components that provide capacity for energy and energy derivative evaluation. We have constructed geometry optimization applications by integrating the Toolkit for Advanced Optimization, Portable Extensible Toolkit for Scientific Computation, and Global Arrays packages, which provide optimization and linear algebra capabilities. We present a brief overview of the component development process and a description of abstract interfaces for chemical optimizations. The components conforming to these abstract interfaces allow the construction of applications using different chemistry and mathematics packages interchangeably. Initial numerical results for the component software demonstrate good performance and highlight potential research enabled by this platform.

Key words: electronic structure, component, software development, optimization

1

1 Introduction

Computational chemistry codes are becoming increasingly complex, not only in the types of computation available, but also in the software architectures used in code development. Today's production software may encompass millions of lines of code with an infrastructure many years (sometimes decades) old. Incorporating into legacy codes algorithms that rely on new mathematics or computer science can be a challenge. This challenge frustrates collaboration; scientists addressing the same problems are often unable to benefit easily from advances made by colleagues, and development efforts are repeatedly duplicated. For example, several classic methods for unconstrained optimization have been described in the literature.[1-8] While some of these methods are well known and easily implemented, subtle techniques that improve efficiency and robustness are usually employed only by experts in the community. Most computational chemistry codes use one or more of these methods to optimize wavefunctions or molecular geometries. Developing proper interfaces with optimization solvers enables chemists to leverage current research in the area of optimization without duplicating the efforts of experts in the field.

As the complexity of scientific models continues to increase, today's problems in computational chemistry benefit greatly by collaborations among chemists, mathematicians, and computer scientists. These collaborations can increase the level of expertise and new science that can be applied to these problems. Toward that end, we are participating in a project designed to enable multiple research groups to build multidisciplinary software for high-performance computational science. This project, the Common Component Architecture (CCA),[9-11] is dedicated to the use of component technology to enable new and efficient high-performance software capabilities.

In this paper, we present our current research into developing component-based applications in chemistry. In the next section, we provide a brief overview of CCA concepts and tools that are used. We then describe the chemistry, optimization, and linear algebra components used to solve a sample chemical problem: molecular structure optimization. We examine several optimization examples and draw conclusions from our experience in component development.

2 The Common Component Architecture

The Common Component Architecture specification[9–11] has arisen in response to the need for a modular programming model that addresses the requirements of high-performance scientific codes. In particular, software interoperability and reuse are facilitated in an environment that emphasizes performance and parallel execution. The CCA community, of which we are a part, is developing a software suite, including support tools, components, and frameworks for management of these components, that enables researchers to build high-performance programs conforming to component standards.

The design of suitable interfaces that link different software packages is often complicated by the use of many programming languages, such as Fortran 77, Fortran 90, C, C++, Python, and Java. Differences, for example in namespaces and memory access, make the compilation and subsequent linking of heterogeneous codes an onerous task, especially when multiple architectures must be supported. The Scientific Interface Definition Language (SIDL)[12] and a code generation tool called Babel[12,13] address many of the difficulties of language interoperability.

SIDL provides a set of fundamental data types and an object-oriented programming model. A SIDL *interface* declares a set of methods using these data types, and a SIDL *class* is an imple-

mentation of methods that may belong to one or more interfaces. Scientists can implement a class using any of the supported programming languages and can use applications written in other languages by employing code generated from Babel. The production of the required SIDL interface definitions effectively separates design and implementation tasks, encouraging focus on high-level design concepts during initial stages of development. The use of Babel is not specifically required by the formal CCA specification, but it has been widely adopted by CCA developers.

The CCA specification itself is defined as a set of SIDL interfaces. In this context, a component is a SIDL class that implements the `Component` interface provided in the CCA specification and usually one or more specialized interfaces. Component interactions take place between interfaces, termed *uses ports* and *provides ports*. Instantiation of components, mediation of interactions between ports, and composition of a working application are the primary tasks of a framework.

The CCA `Component` interface includes only the `setServices()` method, which the framework uses to provide `Services` objects to instantiated components. The `Services` object provides access to a set of framework services, which includes the capability to register ports that a component provides or uses. Beyond employing our existing codes to provide implementations of the SIDL interfaces that we have developed, the only additional programming work involved the use of a small set of these services methods to handle the description and acquisition of ports. In this work, we have used Ccaffeine,[14] one of several frameworks[14–18] that comply with the CCA specification. The Ccaffeine framework currently supports both single-program, multiple-data (SPMD) and multiple-program, multiple-data (MPMD) parallel models and is available for multiple platforms.

The Common Component Architecture is distinguished from other component technologies in its avoidance of both added performance overhead and interference with parallel programming

models. Low overhead in component interaction is achieved by loading components from dynamically loaded libraries into the same process. The overhead associated with a method call over a connected port is similar to that of a C++ virtual function call. Parallel execution within individual components proceeds without hindrance from the framework.

3 Chemistry Optimization Architecture

In nearly all cases of chemical interest, the large differences between the masses of nuclei and electrons allow the treatment of electronic motion independent of nuclear motion. Under this simplification, termed the Born-Oppenheimer approximation, the electronic wavefunctions and, therefore, energies are parameterized by the nuclear coordinates.[19] The resulting potential energy surface (PES) for nuclear motion is determined by summing the nuclear and electronic energies for a given nuclear configuration. Local minima and saddle points on the PES correspond to possible stable chemical species and reaction transition states, respectively, and are therefore of great interest to chemical researchers.

In the following sections, we describe the interfaces and components that we have devised to perform geometry optimization, wherein the total molecular energy is minimized. The software packages that we have integrated to perform this task include two quantum chemistry packages, Massively Parallel Quantum Chemistry (MPQC)[20–22] and NWChem,[23,24] an optimization solver package, the Toolkit for Advanced Optimization (TAO),[25,26] and two parallel data management packages, Global Arrays (GA)[27,28] and the Portable Extensible Toolkit for Scientific Computation (PETSc).[29,30] All these packages are designed for high performance on parallel hardware.

The general architecture for our component based application is shown in Figure 1. The application is composed of several components that together accomplish the task of molecular geometry optimization. Each of the components in the architecture implements ports that are based on SIDL interface specifications. The interface specifications will be partially described in this paper; the full descriptions are available from the authors.

1. Although not a component, the *model* object is a Babel class implementation of the quantum chemistry `Model` interface for the particular code of interest. The model provides the core quantum chemistry capabilities: evaluation of molecular energies and energy derivatives. NWChem and MPQC implementations are currently available.

2. The *model factory* component uses information concerning the molecular system and calculation options to instantiate a model object. Connected components request a model via the provided `ModelFactory` port.

3. The *coordinate model* component provides a `ModelPort` and uses the `ModelFactory` port to obtain a model object. This additional chemistry layer performs optimization tasks that either are specific to the chemical domain or require knowledge of operational details for the chemistry models.

4. The *solver* component implements various optimization algorithms. It provides a `SolverPort` to applications, and it uses an `LAFactoryPort` to instantiate required linear algebra objects and a `ModelPort` to compute energy, gradient, and Hessian information.

5. The *linear algebra* component, currently available as either a PETSc or a GA implementation, supports basic and advanced linear algebra computations by providing an `LAFactoryPort`.

6

Each of these components is further described in the following sections.

4 Chemistry Components

The fundamental task in adapting quantum chemistry packages to provide components is implementation of the `Model` interface. A model contains a molecule class, which maintains molecular structure information and method implementations for molecular energy and energy derivative evaluations. As derivative properties are calculated in Cartesian coordinates, the arguments and return values for chemistry models are restricted to Cartesian quantities. Provided the chemistry models return quantities in the same reference frame as the given coordinates, the details of symmetry implementation and molecule reorientation will not cause conflicts between model implementations. Thus, fault-tolerant applications can be constructed by using redundant chemistry models as back-ups. Though the specific implementation of a model will vary based on the design of the particular quantum chemistry package used, the initialization of a model will generally require the collection of input parameters such as the level of theory, choice of atomic orbital basis set, and molecular structure. An implementation of the `ModelFactory` interface must be provided to collect input parameters and provide them to the initialization member of the model class, returning the initialized model via its provides port.

We have implemented models and model factories for both NWChem and MPQC, and it should be a straightforward task to add components based on other electronic structure programs. NWChem is a computational chemistry code created for massively parallel computations. It has many theoretical and algorithmic methods available for electronic structure calculations with Gaussian and plane-wave basis sets as well as classical molecular dynamics capabilities.[23,24] While

NWChem uses object-oriented principles, it is implemented in a combination of Fortran, C, and Python (with Python used mostly for prototyping capabilities). Therefore, the model factory instantiates essentially the same NWChem object for all initializations. This is in contrast to the MPQC implementation, where the model factory instantiates a different object depending on the input parameters. The first implementation of the NWChem model factory and model functionality used C++ and C++ wrappers because the Ccaffeine framework initially worked only with C++ components. However, the latest versions of the framework are Babel aware and allow the component developer to use languages supported by Babel. We have now implemented these functionalities with native Fortran, which makes our development much more straightforward. This approach will also enable other components that rely on lower-level routines to be implemented easily.

The MPQC package provides parallel implementations of Hartree-Fock, density functional, and various perturbative methods. The object-oriented design is implemented in C++ and runs efficiently on a wide range of computer architectures.[20–22] Adaptation of the MPQC code to provide the core quantum chemistry components primarily involved utilization of the MPQC wavefunction class. The model contains an MPQC wavefunction object, which is constructed using an input string provided by the model factory. Property evaluation methods in the model simply call corresponding methods of the MPQC wavefunction object.

Implementation of the `Model` and `ModelFactory` interfaces provides the basic functionality of quantum chemical codes. The paradigm followed here is to build higher-level application-specific component systems on top of these core chemistry components, using existing general-purpose components when possible. The optimization architecture additionally includes mathematics components, which provide optimization capabilities, and a component that handles chemistry domain-specific tasks. The chemistry-specific coordinate model hides operational details of

the chemistry model, such as the updating of molecular structures via the molecule class, from the mathematics layer. Inclusion of a chemistry-specific optimization layer also allows internal coordinate implementations to be provided independent of the chemistry model, maintaining model interchangeability.

The coordinate model also supports run-time molecular visualization via a viewer interface. Combining viewer and builder interfaces, we have created a prototype graphical user interface (GUI). The GUI uses framework services to construct and run the appropriate component applications based on user input. This prototype front-end demonstrates that component technology can provide a familiar and user-friendly interface to all quantum codes that adopt a component architecture.

5   Numerical Optimization Component

The Toolkit for Advanced Optimization (TAO)[25,26] provides optimization software for the solution of scientific applications on high-performance architectures. These applications include minimizing energy functionals that arise in differential equations[31] and molecular geometry optimization. As surveyed by Moré and Wright,[32] various software packages are available for solving these problems; however, their portability, versatility, and scalability are restricted within parallel environments. Even on uniprocessor architectures, numerical software has traditionally made rigid assumptions about the data structures used for mathematical objects such as vectors and matrices.

Use of optimization software within complex applications involving several research groups and multiple software packages demands robust and flexible solution strategies. TAO includes a variety of algorithms, and its implementations of these algorithms allow scientists to customize

solvers to improve their performance. In particular, the choice of data structures and linear solvers can significantly affect the usage of optimization solvers and their performance.

Within the Common Component Architecture, TAO provides the `SolverPort`, which allows applications to use its optimization solvers. Its interface allows applications to specify the algorithm and set a variety of options. The applications under consideration have the following form: minimize $F(v)$ subject to $l \leq x \leq u$. The objective function F is assumed to be differentiable, and the lower and upper bounds on the variables, l and u, may be set to negative and positive infinity to make the application unconstrained.

The following optimization solvers are implemented in TAO for unconstrained and bound constrained minimization problems.

1. *Newton methods*,[33] which use the first- and second-derivative information from the objective function, are well known for their fast local convergence properties. Global convergence of these methods is enforced using either a *line search* or a *trust region*.[3]

2. *Variable metric methods* use the first-order derivatives at several points to approximate the Hessian matrix of second derivatives. These methods are also known as quasi-Newton and preconditioned conjugate gradient methods. Each iteration uses relatively little time and memory compared to full Newton methods. The methods have proven successful for unconstrained problems and have been adapted to bound-constrained optimizations.[7,8] These methods are particularly useful when the matrix of second derivatives is fully dense and expensive to compute.

A similar interface to each of these solvers allows applications to switch the algorithm used without restructuring their code. The implementation of these algorithms in TAO reduces to a handful of well-defined operations on the objective function and the vectors and matrices.

The TAO *solver* component uses the `ModelPort` for operations on the objective function. A component that provides the `ModelPort` must implement the energy function, provide derivative information, and define bounds when they exist.

The vector and matrix operations used in the optimization solvers involve sums, inner products, and a variety of more specialized linear algebra operations. Objects that implement these operations can be instantiated through the `LAFactoryPort`. TAO uses this port to create objects suitable for its algorithms. Both Global Arrays and PETSc provide this port and offer implementations of these data structures for the solvers in TAO.

The TAO design philosophy is to use object-oriented techniques of data and state encapsulation, abstract classes, and limited inheritance to create a flexible optimization toolkit. The algorithms in the toolkit strongly emphasize the reuse of external tools where appropriate. Its design enables bidirectional connection to lower-level linear algebra support provided in toolkits such as PETSc and Global Arrays as well as higher-level application packages such as MPQC and NWChem.

6   Linear Algebra Components

Parallel implementations of mathematical operations involving vectors and matrices can be found in several packages which can be used as components.

6.1 Global Arrays Component

The Global Arrays (GA) toolkit[27,28] provides a shared-memory programming interface allowing distributed-memory computers to access dense multidimensional arrays. An array created with the GA toolkit is a distributed array that can be accessed in a fashion similar to accessing an array in shared memory. The GA model exposes to the programmer the hierarchical memory of modern high-performance computer systems and, by recognizing the communication overhead for remote data transfer, it promotes data reuse and locality of reference. Global arrays allow the programmer to flexibly decompose and distribute the arrays among the processors, as well as to determine distribution and locality information through library operations. Since GA is compatible with MPI,[34] programmers can use full MPI functionality in addition to GA functionality.

The GA toolkit offers support for both task and data parallelism. Task parallelism is supported through the one-sided (noncollective) copy operations that transfer data between global memory (distributed/shared array) and local memory. The data-parallel computing model is supported through the set of collectively called functions that operate on either entire arrays or sections of global arrays. The set includes BLAS-like operations such as copy, addition, transpose, dot product, and matrix multiplication. The set of data-parallel operations has been enlarged to support the requirements of the `LAFactoryPort`, which includes elementwise operations on arrays (e.g., elementwise addition of two arrays or shifting a diagonal). The GA toolkit also extends its capabilities in the area of linear algebra by offering interfaces to third party libraries, for example, standard and generalized real symmetric eigensolvers (PeIGS[35]) and linear equation solvers (ScaLAPACK[36]).

The GA component provides the `GlobalArrayPort` and `LAFactoryPort`, of which only the latter is used in this application. The GA ports are the set of public interfaces that can be used by other components. The `GlobalArrayPort` provides public interfaces for creating and accessing distributed arrays. The `GlobalArrayPort` offers a set of operations for one-sided data transfer operations (get, put, scatter, gather, etc.), collective array operations, and supportive operations for data locality control, data distribution, and queries. The `LAFactoryPort` provides TAO (and other components) with core linear algebra support for manipulating vectors, matrices, and linear solvers. While GA already has interfaces for Fortran 77, C, Python, and C++, developing a CCA component version of GA further increases its usability with other programs.

6.2 PETSc Component

PETSc[29,30] is a suite of data structures and routines for the scalable solution of scientific applications. The software includes linear and nonlinear equation solvers as well as distributed-memory implementations of vectors and matrices that use MPI[34] for interprocess communication. Efficient methods for the assembly of these structures hide many details of MPI, and a large library of operations involving these structures facilitates their use within large-scale applications.

Given an objective function and derivative information, many nonlinear solvers are defined as a sequence of vector and matrix operations similar to the ones found in the BLAS and LAPACK libraries. The parallel implementations of these methods in PETSc can be used by these solvers to achieve scalable performance on large-scale applications.

Used within a component framework, PETSc provides the `LAFactory` port. This port consists of an interface for creating vector and matrix objects of appropriate size and distribution over processors.

7   Optimization Examples

The chemistry, optimization, and linear algebra components described in the preceding sections comprise a toolkit for large-scale quantum chemistry computations on high-performance architectures. A review of our early numerical results demonstrates the flexibility and potential for future research made possible by component applications.

Figure 2 illustrates speedup for the first five energy and gradient evaluations of an RHF/6-311G(2df,2pd)[37–39] optimization of isoprene performed with MPQC-based applications using up to 64 processors. The strong correlation between speedup for the stand-alone and component software indicates that parallel performance and scalability do not suffer from component overhead, thus meeting one of the goals of this work.

Since most software designs make rigid assumptions regarding programming language and data structures, the ability to compare packages has traditionally been very limited. A significant benefit of component software is that developers can more easily compare different techniques for solving the same problem. By offering a more uniform interface to some of their functionality, NWChem and MPQC can be used interchangeably within the component framework. Similarly, different optimization solvers implementing the SIDL interface can be tested and benchmarked.

As an example of this flexibility, we compared the optimization solvers in NWChem, MPQC, and TAO. These optimization solvers each implement variations of a quasi-Newton algorithm.[3,7]

Quasi-Newton algorithms use differences in coordinates and gradients between consecutive iterations, termed correction pairs, to create a quadratic model. The minimizer of this model determines the step direction. Several features, however, distinguish the three solvers. The MPQC and NWChem solvers perform the traditional Hessian update of Broyden, Fletcher, Goldfarb, and Shanno (BFGS),[3] where the approximate Hessian is explicitly updated at each optimization step using the current correction pair. The limited-memory variable-metric (LMVM) method[7] in TAO implicitly updates an initial approximation to the Hessian using a limited collection of correction pairs. The operation count and memory requirement of this method scale linearly with the number of variables. The component-based optimizations presented here used the TAO solver component with the LMVM algorithm storing at most twenty correction pairs. For a steplength, MPQC always uses a unit multiple of the step direction, whereas NWChem may minimize a second quadratic model, and TAO applies cubic interpolation to adjust the steplength when the unit size does not achieve sufficient reduction in the energy function.

The initial Hessian approximations also differ among the three solvers. Whereas TAO uses a multiple of the identity matrix to approximate the initial Hessian matrix, MPQC can transform a diagonal matrix in internal coordinates, for which force constants can be more accurately approximated, into a full matrix in Cartesian coordinates. While NWChem can also approximate the Hessian in internal coordinates, a diagonal scaled by 1/2 in Cartesian coordinates was used for this research. The LMVM algorithm used in the TAO package requires a guess Hessian as input for each iteration.

The performance of the available solvers on different classes of chemical structures contrasts the strengths of each solver. Tables I and II provide the number of function and gradient evaluations required to determine a converged structure for glycine ($C_2H_5NO_2$), isoprene (C_5H_{10}),

15

phosphoserine ($C_3H_8NO_6P$), acetylsalicylic acid ($C_9H_8O_4$), and cholesterol ($C_{27}H_{46}O$) using the various software programs. Each molecule was optimized at the RHF/6-31G level of theory[37,40,41] using a starting structure obtained at the RHF/STO-3G level of theory.[37,42,43] Since the purpose of these computations is to show the differences between optimization methods, this level of theory provides reasonable results without using significant resources. Energies were computed to an accuracy of 10^{-8} Hartrees in MPQC. For NWChem the value for the SCF convergence criterion (the norm of the orbital gradient) was set to 10^{-10}, yielding an accuracy of at least 10^{-7} Hartrees. The NWChem integral screening threshold was set to 10^{-12}. The optimizations were performed in Cartesian coordinates with convergence tolerances of 0.00045 (0.00030) and 0.00180 (0.00120) for maximum (RMS) gradients and displacements, respectively (atomic units).

Table I presents results for the stand-alone NWChem package and the NWChem/TAO component-based package. The stand-alone NWChem results, generated both with and without line searching, demonstrate that line minimization frequently produces moderate decreases in iteration counts (the number of iterations corresponds to the number of gradient evaluations for the NWChem solver). Though the NWChem solver with line searching does at times yield the lowest number of iterations, the high number of function evaluations required for the line searches causes total execution times to exceed those obtained without line searching for all examples.

While the accompanying differences in Hessian approximations do not allow direct comparison of line searches in NWChem and TAO, the backtracking line search[7] implemented in TAO is able to avoid undue overhead and is therefore an appropriate algorithm to use in quantum chemistry optimization. Line searching is not currently available with the stand-alone MPQC package. From the standpoint of algorithms, the availability of a modern, efficient line search is one clear advantage that quantum chemistry packages may gain through adoption of the CCA architecture.

16

The results for the chemistry and TAO solvers using diagonal guess Hessians (Tables I and II), show that performance benefits are seen with the LMVM solver. As Cartesian coordinates are strongly coupled, Hessians in these coordinates contain large off-diagonal elements and are poorly approximated by diagonal matrices. Regardless, the TAO solver shows considerable performance enhancements in several instances. Factors including the Hessian update and line search schemes contribute in varying degrees to the increased performance observed with the TAO solver component.

The results in Table II emphasize the importance of Hessian approximation quality. Comparing the results obtained with stand-alone MPQC using the transformed internal guess with those for the MPQC/TAO component software, it is apparent that the traditional BFGS algorithm with a good guess Hessian significantly outperforms the current LMVM algorithm. Whether the BFGS or LMVM method is preferred when both use a good guess Hessian is an interesting question that is driving further code development. To improve performance in the chemistry discipline, TAO developers plan to modify the LMVM algorithm to use a dense guess Hessian provided via the `ModelPort`. Since the number of stored correction pairs can be varied, the LMVM method may produce better Hessian approximations, particularly for long optimizations in which BFGS Hessian contributions accumulated in early iterations could be detrimental.

The integration of software packages brought about by this project has enabled a range of observations. Use of the LMVM method is relatively new in computational chemistry, but preliminary results show that further exploration of the algorithm should be pursued. Beyond the examination of solver algorithm performance, valuable insight has been gained into the sensitivity of quantum calculations to numerical issues. Surprisingly, we have observed that the path to a minimum can

be extremely sensitive to not only Hessian updates and line searches, but also SCF convergence criteria and even round-off error.

8 Conclusions

The optimization architecture and example calculations described in this work demonstrate several benefits of component technology. The component concept naturally encapsulates useful software entities and encourages generic interface definitions. With standard component architectures in place, the programming language issue is of concern only at the well-defined component boundaries where tools such as Babel can be employed. As illustrated by our optimization examples, component frameworks allow developers to easily compare techniques. The example calculations, demonstrating performance enhancements and suggesting routes to further improve the TAO solver component for chemical applications, show that component toolkits offer the potential for new research and collaborations between otherwise isolated researchers.

Although work remains to provide the optimization capabilities and efficiency expected by typical users, a component geometry optimization architecture has been developed that allows the leveraging of external solver packages. The greatest potential for benefit lies in more complicated applications of constrained optimization, transition state searching, and reduced density matrix optimization. Solution strategies for such problems are not as well understood, and the complexity of these strategies implies considerable effort in their implementations. The availability of robust and efficient implementations of these strategies in a component framework will provide greater incentive for chemists to use specialized mathematics packages. With the imminent developments in TAO that will allow it to use additional information from chemists and other application groups,

chemical scientists who adopt component technology will be provided a solver that leverages up-to-date optimization technology. In addition to building further capabilities using the existing model components, we will explore the potential for low-level integration within chemistry models.

The flexibility of component technology does come at a cost. Software development best practices are still emerging as the community gains experience, and a fair investment of time is required for software developers wishing to adopt CCA technology. Building functioning component-based applications is considerably complex, requiring installation not only of each software package contributing components, but also of several middleware packages. Currently, the users of component software are primarily those who develop it. Adoption of component technology by the wider chemical community will depend on improvement of the deployment process.

The components that we have developed form the foundation of a flexible toolkit for quantum chemical computations. Through the adoption of technologies developed by the CCA community, our component applications are able to seamlessly incorporate software packages spanning multiple scientific disciplines, development groups, software architectures, and programming languages. Comprising software and middleware designed with a focus on performance, these applications take full advantage of modern parallel computing capabilities. This component platform eliminates barriers that discourage code reuse and collaborative development, thereby facilitating interaction among research groups.

Acknowledgements

References

[1] Császár, P.; Pulay, P. J Mol Struct 1984, 114, 31-34.

[2] Banerjee, A.; Adams, N.; Simons, J.; Shepard, R. J Phys Chem 1985, 89, 52-57.

[3] Fletcher, R. Practical Methods of Optimization: Unconstrained Optimization; Wiley: New York, 1981; Chapter 3.

[4] Moré, J. J.; Sorensen, D. C. Mathematical Programming 1979, 16, 1-20.

[5] Ortega, J. M.; Rheinboldt, W. C. Iterative Solution of Nonlinear Equations in Several Variables; Academic Press: New York, 1970.

[6] Gilbert, J.; Nocedal, J. SIAM J Optimiz 1992, 2, 21-42.

[7] Benson, S.; Moré, J. A Limited-Memory Variable-Metric Algorithm for Bound-Constrained Minimization, Technical Report ANL/MCS-P909-0901, Mathematics and Computer Science Division, Argonne National Laboratory, 2001.

[8] Byrd, R. H.; Lu, P.; Nocedal, J.; Zhu, C. Y. SIAM J Sci Comp 1995, 16, 1190-1208.

[9] Armstrong, R.; Gannon, D.; Geist, A.; Keahey, K.; Kohn, S.; McInnes, L.; Parker, S.; Smolinski, B. in Proceedings. The Eighth International Symposium on High Performance Distributed Computing; IEEE Press: Santa Fe, 1999.

[10] Bernholdt, D. E.; Allan, B. A.; Armstrong, R.; Bertrand, F.; Chiu, K.; Dahlgren, T. L.; Damevski, K.; Elwasif, W. R.; Epperly, T. G. W.; Govindaraju, M.; Katz, D. S.; Kohl, J. A.; Krishnan, M.; Kumfert, G.; Larson, J. W.; Lefantzi, S.; Lewis, M. J.; Malony, A. D.; McInnes, L. C.; Nieplocha, J.; Norris, B.; Parker, S. G.; Ray, J.; Shende, S.; Windus, T. L.; Zhou, S. Intl J High-Perf Computing Appl 2004, submitted.

[11] CCA Forum; CCA Forum homepage; http://www.cca-forum.org/.

[12] Dahlgren, T.; Epperly, T.; Kumfert, G. Babel User's Guide, version 0.9.0, 2004.

[13] Lawrence Livermore National Laboratory; Babel homepage; http://www.llnl.gov/CASC/components/babel.html.

[14] Allan, B. A.; Armstrong, R. C.; Wolfe, A. P.; Ray, J.; Bernholdt, D. E.; Kohl, J. A. Concurrency and Computation: Practice and Experience 2002, 14, 1-23.

[15] Govindaraju, M.; Krishnan, S.; Chiu, K.; Slominski, A.; Gannon, D.; Bramley, R. in 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid; IEEE Press: Santa Fe, 2003.

[16] Zhang, K.; Damevski, K.; Venkatachalapathy, V.; Parker, S. in Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004); IEEE Press: Santa Fe, 2004, to appear.

[17] Grimshaw, A.; Ferrari, A.; Knabe, F.; Humphrey, M. Computer 1999, 32, 29-37.

[18] Lewis, M.; Ferrari, A.; Humphrey, M.; Karpovich, J.; Morgan, M.; Natrajan, A.; Nguyen-Tuong, A.; Wasson, G.; Grimshaw, A. J Parallel and Distributed Comp 2003, 63, 525-538.

[19] Levine, I. N. Quantum Chemistry; Allyn and Bacon: Boston, 1974; Chapter 13.

[20] Janssen, C.; Seidl, E.; Colvin, M. in ACS Symposium Series, Parallel Computing in Computational Chemistry, Vol. 592; American Chemical Society: Washington, D.C., 1995.

[21] Janssen, C. L.; Nielsen, I. M. B.; Colvin, M. E. in Encyclopedia of Computational Chemistry; John Wiley & Sons: Chichester, UK, 1998; Chapter Parallel Processing for ab Initio Quantum Mechanical Methods.

[22] Sandia National Laboratories; MPQC homepage; http://aros.ca.sandia.gov/~cljanss/mpqc.

[23] Kendall, R. A.; Apra, E.; Bernholdt, D. E.; Bylaska, E. J.; Dupuis, M.; Fann, G. I.; Harrison, R. J.; Ju, J. L.; Nichols, J. A.; Nieplocha, J.; Straatsma, T. P.; Windus, T. L.; Wong, A. T. Comput Phys Commun 2000, 128, 260-270.

[24] Pacific Northwest National Laboratory; NWChem homepage; http://www.emsl.pnl.gov/docs/nwchem/.

[25] Benson, S.; McInnes, L. C.; Moré, J.; Sarich, J. TAO User's Manual, Technical Report ANL/MCS-TM-242, Mathematics and Computer Science Division, Argonne National Laboratory, 2004. See http://www.mcs.anl.gov/tao.

[26] Benson, S. J.; McInnes, L. C.; More', J. J. ACM Transactions on Mathematical Software 2001, 27, 361-376.

[27] Nieplocha, J.; Harrison, R. J.; Littlefield, R. J. J Supercomput 1996, 10, 197-220.

[28] Pacific Northwest National Laboratory; Global Array Toolkit homepage; http://www.emsl.pnl.gov:2080/docs/global/.

[29] Balay, S.; Gropp, W. D.; McInnes, L. C.; Smith, B. F. in Modern Software Tools in Scientific Computing; Birkhauser Press: Hunenberg, Switzerland, 1997.

[30] Balay, S.; Buschelman, K.; Gropp, W.; Kaushik, D.; Knepley, M.; McInnes, L.; Smith, B. F.; Zhang, H. PETSc User's Manual, Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2003. See http://www.mcs.anl.gov/petsc.

[31] Rodrigues, J. F. Obstacle Problems in Mathematical Physics; Elsevier: Amsterdam, 1987.

[32] Moré, J. J.; Wright, S. J. Optimization Software Guide; SIAM Publications: Philadelphia, 1993.

[33] Nocedal, J.; Wright, S. J. Numerical Optimization; Springer-Verlag: New York, 1999.

[34] Message Passing Interface Forum, International J of Supercomputer App and High Performance Computing 1994, 8, 159-416.

[35] Fann, G.; Elwood, D.; Littlefield, R. Parallel Eigensystem Solver Manual, version 3.0, 1999.

[36] Blackford, L. S.; Choi, J.; Cleary, A.; D'Azevedo, E.; Demmel, J.; Dhillon, I.; Dongarra, J.; Hammarling, S.; Henry, G.; Petitet, A.; Stanley, K.; Walker, D.; Whaley, R. C. ScaLAPACK User's Guide; SIAM: Philadelphia, 1997.

[37] Roothan, C. C. J. Rev Mod Phys 1951, 23, 69-89.

[38] Krishnan, R.; Binkley, J.; Seeger, R.; Pople, J. J Chem Phys 1980, 72, 650-654.

[39] McLean, A.; Chandler, G. J Chem Phys 1980, 72, 5639-5648.

[40] Hehre, W.; Ditchfield, R.; Pople, J. J Chem Phys 1972, 56, 2257-2261.

[41] Francl, M.; Petro, W.; Hehre, W.; Binkley, J.; Gordon, M.; DeFrees, D.; Pople, J. J Chem Phys 1982, 77, 3654-3665.

[42] Hehre, W.; Stewart, R.; Pople, J. J Chem Phys 1969, 51, 2657-2664.

[43] Collins, J. B.; v. R. Schleyer, P.; Binkley, J. S.; Pople, J. A. J Chem Phys 1976, 64, 5142-5151.

[44] U. S. Dept. of Energy; SciDAC Initiative homepage; http://www.osti.gov/scidac/.

Table I. Numbers of energy and gradient evaluations required for Cartesian structure optimizations at the HF/6-31G level of theory. Results are provided for the stand-alone version of NWChem using a line search and no line search and for the NWChem/TAO component-based application.

Molecule	NWChem (no line search)		NWChem (line search)		NWChem/TAO	
	Energy	Gradient	Energy	Gradient	Energy	Gradient
glycine	33	33	65	33	19	19
isoprene	56	56	89	45	45	45
phosphoserine	79	79	121	61	67	67
acetylsalicylic acid	43	43	83	42	51	51
cholesterol	33	33	>194	>98	30	30

Table II. Numbers of energy and gradient evaluations required for Cartesian structure optimizations at the HF/6-31G level of theory. Results are provided for the stand-alone version of MPQC using transformed internal and unit guess Hessians and for the MPQC/TAO component-based application.

Molecule	MPQC (transformed)		MPQC (unit)		MPQC/TAO	
	Energy	Gradient	Energy	Gradient	Energy	Gradient
glycine	17	17	26	26	19	19
isoprene	18	18	75	75	43	43
phosphoserine	45	45	85	85	62	62
acetylsalicylic acid	24	24	54	54	48	48
cholesterol	25	25	27	27	30	30

Figure Captions:

Figure 1. A schematic representation of the component optimization architecture.

Figure 2. Parallel speedup for the first five function and gradient evaluations of isoprene RHF/6-311G(2df,2pd) optimizations using MPQC in stand-alone (MPQC) and component (MPQC/TAO) applications.
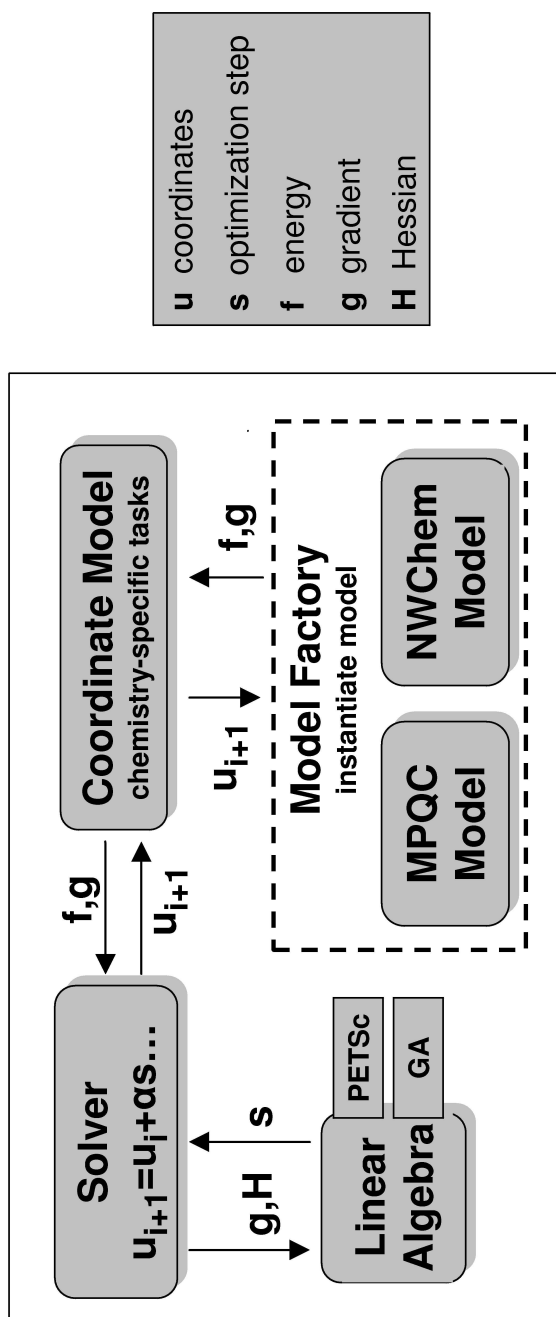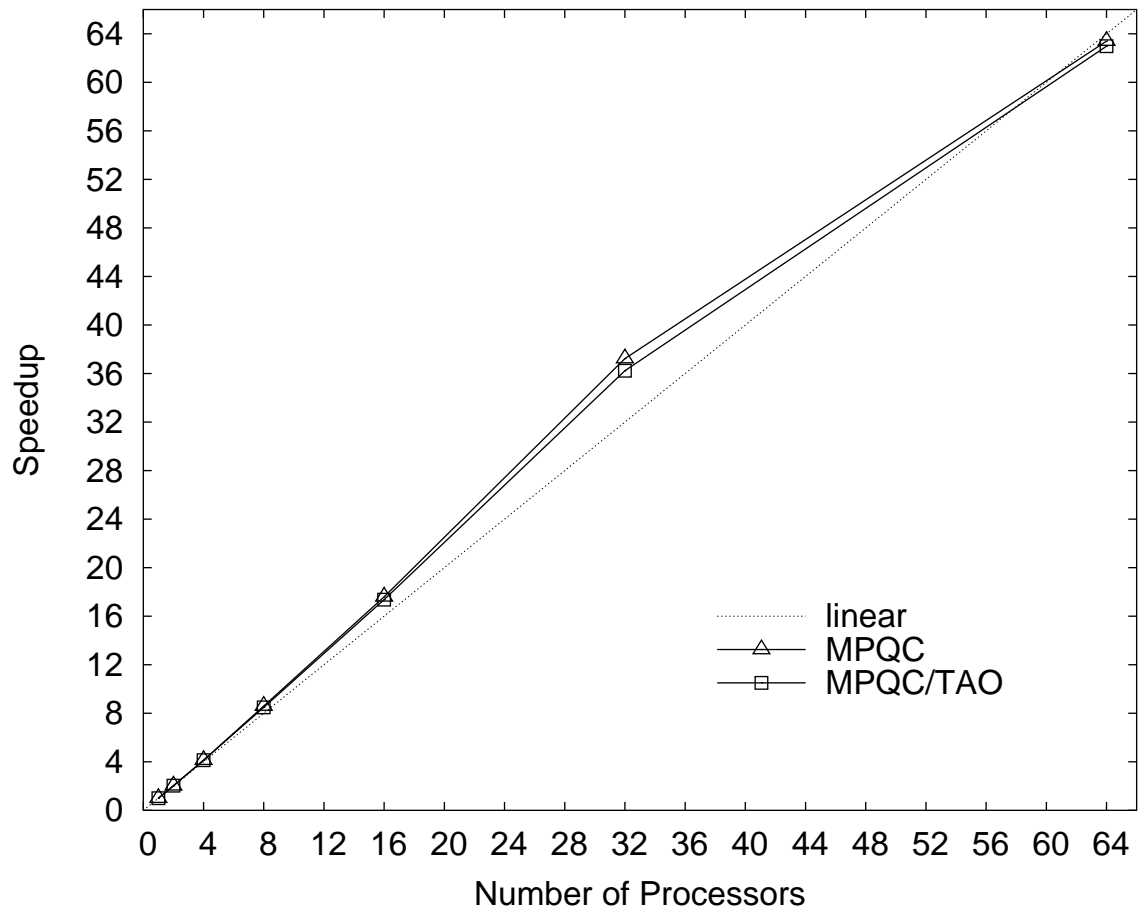
Figure 1.

Figure 2.

Graphical Table of Contents:

Components in Computational Chemistry

Joseph P. Kenny,[1] Steven J. Benson,[2] Yuri Alexeev,[3] Jason Sarich,[2] Curtis L. Janssen,[1] Lois Curfman McInnes,[2] Manojkumar Krishnan,[4] Jarek Nieplocha,[4] Elizabeth Jurrus,[4] Carl Fahlstrom,[3] Theresa L. Windus[3]

[1]*High Performance Computing and Networking Department, Sandia National Laboratories, MS 9915, P.O. Box 969, Livermore, CA 94551-0969*

[2]*Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439*

[3]*Environmental Molecular Sciences Laboratory, Pacific Northwest National Laboratory, MS-IN: K8-91, P.O. Box 999, Richland, WA 99352*

[4]*Computational Science and Mathematics, Pacific Northwest National Laboratory, MS-IN: K1-85, P.O. Box 999, Richland, WA 99352*

Please use abstract from manuscript for graphical TOC abstract.

Please use Figure 1 from manuscript for graphical TOC figure.