Institute of Physics Publishing Journal of Physics: Conference Series **16** (2005) 536–540
doi:10.1088/1742-6596/16/1/073 SciDAC 2005

Component-based software for high-performance scientific computing

Yuri Alexeev, Benjamin A. Allan, Robert C. Armstrong, David E. Bernholdt, Tamara L. Dahlgren, Dennis Gannon, Curtis L. Janssen, Joseph P. Kenny, Manojkumar Krishnan, James A. Kohl, Gary Kumfert, Lois Curfman McInnes, Jarek Nieplocha, Steven G. Parker, Craig Rasmussen and Theresa L. Windus, *on behalf of the Center for Component Technology for Terascale Simulation Software (http://www.cca-forum.org)*

Sandia National Laboratories, Livermore, CA 94551 USA
Oak Ridge National Laboratory, Oak Ridge, TN 37831 USA
Dept. of Computer Science, Indiana University, Blookington, IN 47405 USA
Pacific Northwest National Laboratory, Richland, WA 99352 USA
Lawrence Livermore National Laboratory, Livermore, CA 94551 USA
Argonne National Laboratory, Argonne, IL 60439 USA
SCI Institite, University of Utah, Salt Lake City, UT 84112 USA
Los Alamos National Laboratory, Los Alamos, NM 87545 USA

E-mail: gannon@cs.indiana.edu, rob@sandia.gov

Abstract. Recent advances in both computational hardware and multidisciplinary science have given rise to an unprecedented level of complexity in scientific simulation software. This paper describes an ongoing grass roots effort aimed at addressing complexity in high-performance computing through the use of Component-Based Software Engineering (CBSE). Highlights of the benefits and accomplishments of the Common Component Architecture (CCA) Forum and SciDAC ISIC are given, followed by an illustrative example of how the CCA has been applied to drive scientific discovery in quantum chemistry. Thrusts for future research are also described briefly.

1. Introduction

In recent years, two accomplishments have fueled an upsurge in the complexity of scientific simulation software. First, rapid growth in computational capability based on increasingly intricate hardware architectures is driving computational scientists to develop new, more complex algorithms to make best use of the systems. Second, scientific advances are yielding new ways of approaching challenging problems, offering better efficiency, accuracy, or fidelity. Code complexity and reliance on software are increasing as essential consequences of both of these accomplishments. Computational science software is at growing risk of becoming a victim of its own success, increasing in complexity until it becomes unmanageable, unmaintainable, and incomprehensible. This inherent complexity impacts the productivity of developers and, if left alone, ultimately may cap the rate of progress in creating and improving scientific software.

Component-based software engineering (CBSE) is an approach developed in other areas of computing as a means of addressing similar problems of complexity. Units of software functionality are encapsulated as *components* which interact with each other only through well-defined interfaces. The

actual implementation is opaque to other components, and application composition is archived through providing and using these interfaces. This approach reduces complexity by allowing *developers* to focus on the internals of the small set of components on which they are working, while *users* of components need only be concerned with component interfaces. This separation of concerns is useful in the collaborative or community-oriented software development that increasingly characterizes modern high-end simulations. Component-based environments typically offer a "plug and play" approach to composition of components into applications, in which components offering the same interface are interchangeable, allowing easy swapping of components to test new algorithms, tune for performance, and other reasons. To the extent that interfaces for certain functionality are agreed to by communities of users, components can more easily be reused across multiple applications.

2. The CCA in Brief

Since 2001, the SciDAC-funded Center for Component Technology for Terascale Simulation Software (CCTTSS) has led a program of research and development into the formulation, roles, and use of component technologies in high-performance science computing.

A central result of this effort has been the Common Component Architecture (CCA) [1, 2], a specification for a component environment designed to meet the needs of modern high-performance computational science. The basic features of the CCA's design that distinguish it from commodity component architectures widely used in non-scientific applications include:

- **High Performance:** Minimize performance penalties due to componentization and allow HPC-friendly implementations.
- **Parallel and Distributed Computing:** Support local, parallel, and distributed computing seamlessly. However, the CCA does not impose a particular parallel programming model or distributed computing environment on the user.
- **HPC Language Support, Data Types, and Platforms:** In particular, support for Fortran, multi-dimensional arrays and complex numbers, and computer platforms that do not run Java and/or Microsoft Windows.
- **Easy Reuse of Legacy Software:** Minimize the effort required to incorporate existing software into the CCA environment.

CCA components are loaded by a framework and assembled into an application by connecting interfaces that express a requirement for a certain functionality (*uses ports* in CCA terminology) with implementations of that functionality (*provides ports*), and the application can then be executed. Application composition can be scripted or accomplished interactively with a GUI. Instantiation and connection of components can even be done under program control by a main program or a special component. The CCA-specified "BuilderService" interface allows applications to dynamically adapt, changing out components on-the-fly to improve performance or numerics as a simulation progresses.

3. CCA Accomplishment Highlights

Productivity and Performance Improvements. CCA technology is intended to change the way scientific software is developed and used, and to improve the productivity of both developers and users. Early-adopters of the CCA have realized these benefits. The SciDAC Center for Reacting Flow Science [3] found the CCA approach greatly increases their productivity and performance when incorporating cutting-edge discretization ideas into their code [4, 5]. Similarly, the quantum chemistry community has realized order-of-magnitude performance improvements [6], and rapidly benchmarked alternative numerical implementations [7]. Tools to support the automated generation and building of CCA wrappers for legacy codes have been incorporated in the open-source Eclipse software-productivity environment.

CCA Technologies. Emphasizing the importance of community-defined, "common" software interfaces to increasing the sharing and reuse of scientific codes is a hallmark of CBSE. The CCA is a catalyst for such efforts in several application domains.

Along with the core CCA component model defined by its specification, reference implementations of the technologies and tools associated with the CCA environment exist. These include the Scientific Interface Definition Language (SIDL) [8], the Babel compiler [9], and Ccaffeine [10]. SIDL provides an implementation-language neutral mechanism for specifying component Application Programming Interfaces (APIs). Babel, the SIDL compiler, makes CCA components interoperable across languages *and* CCA frameworks. Ccaffeine is a reference framework that supports parallel application composition from CCA components for execution in massively parallel environments. A number of alternative CCA framework implementations also exist that target distributed computing and hybrid approaches [11–13].

CCA technology is being evaluated or used in a number of communities. Numerous studies demonstrate small overheads of the CCA environment that are easily amortized in typical scientific applications. Eleven application areas are currently experimenting with the technology, including biomedical engineering, climate modeling, combustion simulation, computational chemistry, and fusion energy. Companies such as Cray, Fluent, IBM, MSC.Software, and Tech-X have expressed interest in or are using CCA.

CCA technologies have also impacted a number of standards efforts. SIDL is being used for interface standardization in areas such as meshing software, linear and nonlinear solvers, and computational chemistry. The design of the CCA has influenced the Business Process Execution Language (BPEL), a web services work-flow specification language used by Microsoft, IBM, BEA, SAP, and other major corporations. The Chasm array-interoperability API used in Babel was accepted by the J3 standard committee to be part of the next Fortran standard following Fortran 2003.

Reusable Scientific Components. In addition to simple component examples and hands-on exercises available in the CCA tutorial materials, a growing set of components has been made available on the web since August 2003. Many of these components are used in the scientific applications mentioned elsewhere in this paper. The CCA component toolkit is based on widely used software packages, including: ARMCI (one-sided messaging), CUMULVS (visualization and parallel data redistribution), CVODE (integrators), DRA (parallel I/O), Epetra (sparse linear solvers), Global Arrays (parallel programming), GrACE (structured adaptive meshes), netCDF and parallel netCDF (input/output), TAO (optimization), TAU (performance measurement), and TOPS (linear and nonlinear solvers).

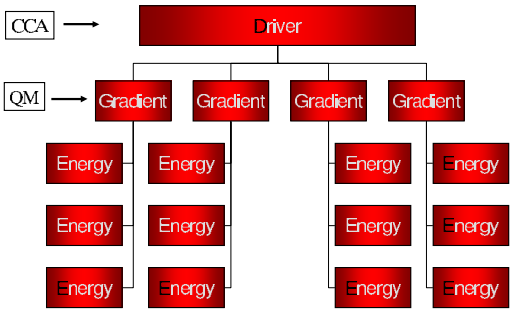
4. Example: CCA Use in Quantum Chemistry

Central to many public, industrial and scientific endeavors, chemistry is a ubiquitous science in which simulation plays an indispensable role. Developers of two major quantum chemistry (QC) packages, MPQC [14] and NWChem [15], are using the Common Component Architecture to transform the way QC software is developed and used.

In a field where lack of software interoperability frequently frustrates collaboration, MPQC and NWChem developers teamed with experts in optimization to study the applicability of state-of-the-art optimization algorithms to the determination of molecular structures. Interchangeable CCA-compliant components were created from MPQC and NWChem (the former written in C++, the latter in Fortran 77) for the evaluation of molecular energies, gradients, and Hessians. These were coupled with the TAO [16] optimization component and supporting interchangeable linear algebra components derived from Global Arrays (GA) [17] or PETSc [18]. This novel software enabled a study that demonstrated the benefits of new limited memory variable metric (LMVM) optimization approaches to QC [7], with reductions of as much as 42% in execution time.

The coarse-grained interchangeability of functionality between QC packages demonstrated in this study represents another important benefit of the component approach. By agreeing on common interfaces for smaller units of functionality within the codes, it becomes possible for one code to take

advantage of functionality present only in the other code to produce new capabilities not present in either code separately. The current work focuses on enhancing the capabilities in the evaluation of molecular properties. Common interfaces, agreed to by larger groups, can form the basis for turning many standalone packages into a community code base, allowing faster and easier software development, and ultimately, more and better scientific progress. This is beginning to happen in the QC community, as the developers of another package, GAMESS [19], have recently also joined the molecular properties effort.



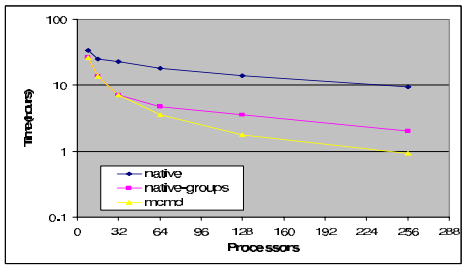


Figure 1. Scalability of Numerical Hessian.

NWChem developers are also using the CCA to increase the parallel scalability of certain types of calculations. For many QC methods, Hessians (second derivatives of the energy) must be calculated by numerical differentiation of gradients, which, in turn, may be obtained from either analytic or numerical differentiation of the energy. Using CCA and GA, NWChem developers were able to implement a computational scheme that provides three different levels of parallelism (Figure 1): the top level driver for the Hessian calculation subdivides the available processors to perform multiple gradient calculations simultaneously. By dynamically creating and managing the processor groups to carry out the calculation, available computational resources can be used much more effectively. The CCA and GA-based multi-level implementation (*MCMD*) yields a factor of 10 speedup (Figure 1) [6] in NWChem. The Hessian is perhaps the simplest example of this type of approach. The multilevel scheme opens the door to a rich set of algorithms that require a dynamic architecture to enable advancement in large-scale chemical simulations.

5. Future Research

The CCA goal is to accelerate the rate and scope of scientific discovery by managing the complexity inherent in leadership-class scientific computing applications. Future research directions are prioritized by sources of complexity that constrain and prolong software solutions. Included is complexity derived from hardware, software and the consequent need to build community software. Because next generation simulations will incorporate federations of codes representing multiple kinds and scales of physics simulations, a wide range of parallel coupling services need to be incorporated to automate discovery/attachment, parallel data sharing, and interpolation schemes in space and time, with flux conservation constraints, units conversion, etc. Advanced programming models not only make language interoperability easier, they can impart semantic information that could be leveraged to automatically enforce proper code usage [20]. Generally referred to as Computational Quality of Service (CQoS), information contained in frameworks and programming models regarding an application's structure, algorithmic characteristics, and performance history can be exploited to optimize execution dynamically. Prototypes exist for hybrid algorithms [21] and automated performance tuning [22].

Acknowledgments

The CCA has been under development since 1998 by the CCA Forum and represents the contributions of many people, all of whom we gratefully acknowledge. We further acknowledge our collaborators outside

the CCA Forum and the early adopters of the CCA for the important contributions they have made to making it a practical and usable environment.

This work has been supported in part by the U. S. Department of Energy's Scientific Discovery through Advanced Computing (SciDAC) initiative, through the Center for Component Technology for Terascale Simulation Software, of which Argonne, Lawrence Livermore, Los Alamos, Oak Ridge, Pacific Northwest, and Sandia National Laboratories, Indiana University, and the University of Utah are members. Oak Ridge National Laboratory is managed by UT-Battelle, LLC for the US Dept. of Energy under contract DE-AC-05-00OR22725.

References

[1] D. E. Bernholdt, B. A. Allan, R. Armstrong, F. Bertrand, K. Chiu, T. L. Dahlgren, K. Damevski, W. R. Elwasif, T. G. W. Epperly, M. Govindaraju, D. S. Katz, J. A. Kohl, M. Krishnan, G. Kumfert, J. W. Larson, S. Lefantzi, M. J. Lewis, A. D. Malony, L. C. McInnes, J. Nieplocha, B. Norris, S. G. Parker, J. Ray, S. Shende, Ti. L. Windus, and S. Zhou. A component architecture for high-performance scientific computing. *Intl. J. High-Perf. Computing Appl.*, 2005. ACTS Collection special issue, in press.

[2] Common Component Architecture Forum homepage. `http://www.cca-forum.org/`, 2005.

[3] CFRFS homepage. `http://cfrfs.ca.sandia.gov/`, 2005.

[4] B. Allan and J. Ray. The scalability impact of a component-based software engineering framework on a growing SAMR toolkit. In *International Conference on Parallel Computational Fluid Dynamics (PCFD2005)*, 2005.

[5] S. Lefantzi, J. Ray, C. Kennedy, and H. Najm. A Component-based Toolkit for Reacting Flow with High Order Spatial Discretizations on Structured Adaptively Refined Meshes. *Progress in Computational Fluid Dynamics: An International Journal*, 2004. to appear.

[6] M. Krishnan, Y. Alexeev, T. Windus, and J. Nieplocha. Multilevel parallelism in computational chemistry using common component architecture. submitted to Supercomputing 2005.

[7] J. P. Kenny, S. J. Benson, Y. Alexeev, J. Sarich, C. L. Janssen, L. C. Mcinnes, M. Krishnan, J. Nieplocha, E. Jurrus, C. Fahlstrom, and T. L. Windus. Component-based integration of chemistry and optimization software. *J. of Computational Chemistry*, 24(14):1717–1725, 2004.

[8] T. Dahlgren, T. Epperly, G. Kumfert, and J. Leek. *Babel User's Guide*. LLNL, Livermore, CA, 0.10.4 edition, 2005.

[9] Babel homepage. `http://www.llnl.gov/CASC/components/babel.html`, 2005.

[10] B. A. Allan, R. C. Armstrong, A. P. Wolfe, J. Ray, D. E. Bernholdt, and J. A. Kohl. The CCA core specification in a distributed memory SPMD framework. *Concurrency and Computation: Practice and Experience*, 14(5):1–23, 2002.

[11] K. Zhang, K. Damevski, V. Venkatachalapathy, and S. Parker. SCIRun2: A CCA framework for high performance computing. In *HIPS '04*, April 2004.

[12] F. Bertrand and R. Bramley. DCA: A distributed CCA framework based on MPI. In *HIPS '04*, April 2004.

[13] S. Krishnan and D. Gannon. XCAT3: A framework for CCA components as ogsa services. In *HIPS '04*, pages 90–97, April 2004.

[14] C. L. Janssen, I. M. B. Nielsen, and M. E. Colvin. *Encyclopedia of Computational Chemistry*, chapter Parallel Processing for ab Initio Quantum Mechanical Methods. John Wiley & Sons, Chichester, UK, 1998.

[15] R. A. Kendall, E. Apra, D. E. Bernholdt, E. J. Bylaska, M. Dupuis, G. I. Fann, R. J. Harrison, J. L. Ju, J. A. Nichols, J. Nieplocha, T. P. Straatsma, T. L. Windus, and A. T. Wong. High performance computational chemistry: An overview of NWChem, a distributed parallel application. *Computational Physics Communication*, 128:260–270, 2000.

[16] S. Benson, L. C. McInnes, J. Moré, and J. Sarich. TAO users manual. Technical Report ANL/MCS-TM-242 - Revision 1.5, Argonne National Laboratory, 2003. `http://www.mcs.anl.gov/tao/`.

[17] J. Nieplocha, R. J. Harrison, and R. J. Littlefield. Global arrays: A non-uniform-memory-access programming model for high-performance computers. *J. Supercomputing*, 10(2):169, 1996.

[18] S. Balay, K. Buschelman, W. Gropp, D. Kaushik, M. Knepley, L. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2003.

[19] M. Schmidt, K. Baldridge, J. Boatz, S. Elbert, M. Gordon, J. Jensen, S. Koseki, N. Matsunaga, K. Nguyen, S. Su, T. Windus, M. Dupuis, and Jr. J. Montgomery. General atomic and molecular electronic structure system. *J. Comput. Chem.*, 14(11):1347–1363, 1993.

[20] T. Dahlgren and P. Devanbu. Improving scientific software component quality through assertions. In *SE-HPCS'05*.

[21] P. Hovland, K. Keahey, L. McInnes, B. Norris, L. Diachin, and P. Raghavan. A quality-of-service architecture for high-performance numerical components. In *Proceedings of the Workshop on QoS in Component-Based Software Engineering*, June 2003.

[22] B. Norris, J. Ray, R. Armstrong, L. McInnes, D. Bernholdt, W. Elwasif, A. Malony, and S. Shende. Computational quality of service for scientific components. In *CBSE7*, May 2004.