Research Initiatives for Plug-and-Play Scientific Computing

Lois Curfman McInnes,[1] Tamara Dahlgren,[2] Jarek Nieplocha,[3] David Bernholdt,[4] Ben Allan,[5] Rob Armstrong,[5] Daniel Chavarria,[3] Wael Elwasif,[4] Ian Gorton,[3] Joe Kenny,[5] Manoj Krishan,[3] Allen Malony,[6] Boyana Norris,[1] Jaideep Ray,[7] and Sameer Shende[6]

[1] Mathematics and Computer Science, Argonne National Laboratory, Argonne, IL
[2] Lawrence Livermore National Laboratory, Livermore, CA
[3] Computational Sciences and Mathematics, Pacific Northwest Nat. Laboratory, Richland, WA
[4] Computer Science and Mathematics, Oak Ridge National Laboratory, Oak Ridge, TN
[5] Scalable Computing R & D, Sandia National Laboratories, Livermore, CA
[6] Computer and Information Science, University of Oregon, Eugene, OR
[7] Advanced Software R & D, Sandia National Laboratories, Livermore, CA

E-mail: `mcinnes@mcs.anl.gov`

Abstract. This paper introduces three component technology initiatives within the SciDAC Center for Technology for Advanced Scientific Component Software (TASCS) that address ever-increasing productivity challenges in creating, managing, and applying simulation software to scientific discovery. By leveraging the Common Component Architecture (CCA), a new component standard for high-performance scientific computing, these initiatives tackle difficulties at different but related levels in the development of component-based scientific software: (1) deploying applications on massively parallel and heterogeneous architectures, (2) investigating new approaches to the runtime enforcement of behavioral semantics, and (3) developing tools to facilitate dynamic composition, substitution, and reconfiguration of component implementations and parameters, so that application scientists can explore tradeoffs among factors such as accuracy, reliability, and performance.

1. Introduction

This paper introduces three component technology initiatives that focus on reducing the software development challenges faced by today's computational scientists. Rapid advances and increasing diversity in high-performance hardware platforms continue to spur the growing complexity of scientific simulations. The resulting environment presents ever-increasing productivity challenges associated with creating, managing, and applying simulation software to scientific discovery. As key facets within the SciDAC Center for Technology for Advanced Scientific Component Software (TASCS) [1], these initiatives leverage the component standard for scientific computing under development by the Common Component Architecture (CCA) Forum [2]. Component technology (see, e.g., [3]), which is now widely used in mainstream computing but has only recently begun to make inroads in high-performance computing (HPC), extends the benefits of object-oriented design by providing coding methodologies and supporting infrastructure to improve software's extensibility, maintainability, and reliability. All three initiatives are based on the premise that, in addition to aiding software development, the component environment can facilitate the deployment of new computational capabilities to benefit the entire lifecycle of scientific simulation software.

The CCA Forum is addressing the aforementioned productivity challenges by developing tools for plug-and-play composition of applications in both high-performance parallel and distributed computing contexts. The core of this work is a component model and reference implementation [4] tailored to the needs of high-end scientific computing. Key features are capabilities for language-neutral specification of common component interfaces, interoperability for software written in programming languages important to scientific computing, and dynamic composability, all with minimal runtime overhead. The three component technology initiatives, which extend this preliminary work, are motivated by and will be validated through collaborations with a variety of computational science groups, including SciDAC application teams working in combustion [5], quantum chemistry [6], core-edge fusion modeling [7], accelerator modeling [8], subsurface modeling [9], and proteomics [10], as well as with mathematicians in the ITAPS [11] and TOPS [12] centers. Further information about the CCA approach and additional collaborating application teams is provided in [13].

The three TASCS initiatives introduced in this paper address challenges at different but related levels in the development of component-based scientific software. The first initiative, introduced in Section 2, focuses on helping users deploy component technology-based applications on massively parallel and heterogeneous architectures. The second initiative, introduced in Section 3, investigates new approaches to the runtime enforcement of behavioral semantics, with an emphasis on techniques for reducing their performance impact during deployment. The third initiative, introduced in Section 4, focuses on developing tools to help application scientists dynamically compose, substitute, and reconfigure component implementations and parameters, taking into account trade-offs among *computational quality of service* factors such as performance, accuracy, mathematical consistency, and reliability.

2. Emerging HPC Environments

Scientists who are developing petascale computational science capabilities continue to face major challenges in adapting or developing software to effectively use emerging hardware environments. Upcoming petascale computer systems will be characterized by large processor counts (systems with $\mathcal{O}(10^4 - 10^5)$ processors are already being deployed [14]) and increasing use of heterogeneous and specialized environments, in which FPGA, graphics processors, and other hardware will be harnessed to accelerate general scientific computing. An example of such an architecture is the IBM Roadrunner system to be delivered to Los Alamos National Laboratory. Roadrunner, which will deploy hybrid processing nodes based on the AMD Opteron and the IBM-Sony-Toshiba Cell processor, is designed to achieve sustained performance of 1 petaflop/s, or 1 quadrillion calculations per second. The Cell processor internally is composed of a general-purpose PowerPC processor and eight synergistic processing engines.

Heterogeneous computing environments will require adaptation of programs to work in the presence (or absence) of various specialty interfaces, which are currently unique to each vendor. The CCA is expected to deliver significant benefits to applications running on heterogeneous hardware by helping to manage and interface software components internally using hardware accelerators. We are developing the CCA event service as a mechanism to "glue together" components that run on heterogeneous hardware.

High processor counts will require applications to expose more parallelism. For situations in which it does not make scientific sense to scale the problem size or resolution, approaches in which different groups of processes carry out different parallel tasks simultaneously can be used to increase parallelism. Such approaches, however, are not easy to manage or code with current tools.

We will provide CCA users with more flexible and dynamic means to express application parallelism through the development of support for the management of process groups and Multiple Component Multiple Data (MCMD) applications. This work will develop standard CCA tools and interfaces, guided in particular by our experience with such applications in chemistry [15] and the threaded parallel "task graph" execution model provided by the Uintah Computational Framework [16].

The sheer number of parts in petascale systems will pose significant issues with respect to hardware (and software) faults, some of which may be most effectively handled by fault awareness at the application level. With respect to fault tolerance, our approach is to ensure that CCA-based applications can take full advantage of the capabilities being developed by other DOE project focusing on this area, including [17].

This initiative will provide CCA users with new tools that will simplify and accelerate the development of true petascale applications on diverse hardware platforms. Users will be able to flexibly and dynamically express higher levels of parallelism, transparently take advantage of specialized coprocessing resources, and support intelligent application-level responses to the hardware failures that are inevitable on systems of this scale.

3. Software Quality and Verification

To help make the vision of interchangeable components a reality for scientific software, the Software Quality and Verification initiative focuses on the composition- and execution-time verification of interface semantics. Component interfaces, expressed separately from the implementation, can be extended with semantic information to provide concise, human-readable, machine-processable specifications. Unlike traditional verification techniques based either on postexecution comparisons with prior or analytical results or on algorithm-based fault tolerance techniques, this approach enables error detection closer to the point of failure. The result is improved testing, debugging, and runtime monitoring of software quality, thereby providing scientific software developers with a powerful tool for catching errors early and ensuring correct software usage.

Preliminary work has already enhanced the basic syntactic descriptions of component interfaces, or application programming interfaces (APIs), through the addition of nonnegotiable behavioral contracts [18]. Current contract annotations support general constraints on the input and output of method calls and object properties. Research efforts thus far have concentrated on exploring the impact on performance, coverage, and failure detection of a variety of traditional and experimental enforcement heuristics [18, 19].

Future work will focus on expanding supported annotations for domain-specific behavioral and quality-of-service features, with an emphasis on facilitating automated behavioral adaptation across disciplines within computational science. Annotations of interest include constraints relating to algorithm characteristics, precision, result quality, and method invocation sequencing. Examples of properties include whether an algorithm is implicit or explicit, whether the storage order of arrays is exchanged through its interfaces, and whether a two-dimensional or three-dimensional space is represented. Stable characteristics such as these are expected to be expressible in an implementation language-neutral form, such as the Scientific Interface Definition Language (SIDL) [20], while dynamic characteristics will require representation in a more flexible format, such as that presented in Section 4.

Work on interface semantics addresses the need to more explicitly define behavior and quality-of-services constraints in a concise, human-readable, machine-processable form. Integrating the specification of constraints on stable characteristics into SIDL and dynamic characteristics into the control infrastructure discussed in Section 4 will enable runtime adaptation according to a variety of behavioral and service quality criteria.

4. Computational Quality of Service

As computational science progresses toward ever more realistic multiphysics and multiscale applications, the complexity is becoming such that no single research group can effectively select or tune all of the components in a given application, and no single tool, solver, or solution strategy can seamlessly span the entire spectrum *efficiently*. Common component interfaces enable easy access to suites of independently developed algorithms and implementations. The challenge then becomes how, during runtime, to make the best choices for reliability, accuracy, and performance.

We are addressing this challenge by developing tools for *computational quality of service* (CQoS) [21] or the automatic selection and configuration of components to suit a particular computational purpose. As further explained in [22], CQoS embodies the familiar concept of quality of service in networking as well as the ability to specify and manage characteristics of the application in a way that adapts to the changing (computational) environment. Specific scientific applications that motivate this research are parallel mesh partitioning in combustion simulations within the CFRFS project [5], resource management in quantum chemistry [6], and efficient solution of linear systems arising in accelerator and fusion simulations (collaborations with the FACETS [7] and COMPASS [8] projects).

We expect that the logic involved in characterizing these problems and choosing appropriate solution strategies will be vastly different. Nevertheless, we believe that the *software infrastructure* to analyze and characterize each problem and its potential solutions, as well as the *software infrastructure* to implement a decision (once made by domain-specific logic), is similar and may be generalized.

We are thus developing CQoS infrastructure to help analyze, select, and parameterize components for these motivating applications. The two main facets of our CQoS tools are (1) *measurement and analysis infrastructure*, which combines performance information and models from historical and runtime databases along with interactive analysis, including statistical analysis and machine learning technology using the TAU performance system [23]; and (2) *control infrastructure*, which encompasses decision-making components that evaluate progress based on domain-specific heuristics and metrics, along with services for dynamic component replacement. These two groups of CQoS tools, which may be employed for initially composing, for configuring, and for runtime control of an application, are largely decoupled and interact primarily through a substitution assertion database.

This initiative exploits work on interface semantics discussed in Section 3 and is a collaboration with the PERI [24] project on performance optimization. Further details about motivation, related work, and plans for future CQoS research are discussed in [22].

5. Conclusion

This paper introduced three TASCS technology initiatives that leverage the CCA component environment to address new challenges being faced by SciDAC applications teams. A particularly interesting new development is that these projects are transitioning to employ Bocca [25], a comprehensive suite of CCA build tools that is under development in support of these initiatives as well as the needs of general SciDAC applications teams. Bocca provides project management and a comprehensive build environment for creating and managing applications composed of components written in all of the common HPC workstation languages: C, C++, Fortran, Fortran77, Python, and Java. Its design embraces the philosophy pioneered by Ruby Rails for web applications: start with something that works, and evolve it to the user's purpose. Bocca automates the tasks related to the component glue code, freeing the user to focus on the scientific aspects of the application.

Acknowledgments

This work was funded by the U. S. Department of Energy Office of Science through the SciDAC program. Argonne National Laboratory operates under contract DE-AC02-06CH11357. Lawrence Livermore National Laboratory is managed by the University of California under contract W-7405-Eng-48. Pacific Northwest National Laboratory is operated by Battelle under contract DE-AC06-76RLO 1830. Oak Ridge National Laboratory is managed by UT-Battelle, LLC under contract DE-AC-05-00OR22725. Research at the University of Oregon is sponsored under contracts DEFG03-01ER25501 and DE-FG02-03ER25561. Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, under contract DE-AC04-94-AL85000.

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 also thank our collaborators outside the CCA Forum, especially the scientific applications teams who help motivate and validate this work.

References

[1] Bernholdt D (PI) TASCS Center `http://www.scidac.gov/compsci/TASCS.html`
[2] Common Component Architecture (CCA) Forum `http://www.cca-forum.org/`
[3] Szyperski C 1999 *Component Software: Beyond Object-Oriented Programming* (New York: ACM Press)
[4] Bernholdt D et al 2006 *Int. J. High-Perf. Computing Appl., ACTS Collection special issue* **20** 163–202
[5] Najm H (PI) Computational Facility for Reacting Flow Science (CFRFS) `http://cfrfs.ca.sandia.gov`
[6] Gordon M (PI) Chemistry Framework using the CCA `http://www.scidac.gov/matchem/better.html`
[7] Cary J (PI) Framework Application for Core-Edge Transport Simulations `http://www.facetsproject.org`
[8] Spentzouris P (PI) Community Petascale Project for Accelerator Science and Simulation (COMPASS) DOE SciDAC project, 2007
[9] Sheibe T Computational hybrid integration of physical processes across scales (CHIPPS) `http:www.emsl.pnl.gov/capabs/hpmsf.shtml`
[10] High-Performance Mass Spectrometry Facility `http://www.emsl.pnl.gov/capabs/hpmsf.shtml`
[11] Diachin L (PI) Center for Interoperable Technologies for Advanced Petascale Simulations (ITAPS) `http://www.scidac.gov/math/ITAPS.html`
[12] Keyes D (PI) Towards Optimal Petascale Simulations (TOPS) Center `http://tops-scidac.org/`
[13] Kumfert G, Bernholdt D, Epperly T, Kohl J, McInnes L, Parker S and Ray J 2006 *Journal of Physics: Conference Series 46* 479–493
[14] Mauer H et al 2007 TOP500 Supercomputer Sites `http://www.top500.org`
[15] Krishnan M, Alexeev Y, Windus T L and Nieplocha J 2005 *Proceedings of SuperComputing* (ACM and IEEE)
[16] de St Germain J D, McCorquodale J, Parker S G and Johnson C R 2000 *Proc. 9th IEEE Int. Symp. on High Performance and Distributed Computing*
[17] Beckman P (PI) Coordinated Fault Tolerance for High-Performance Computing (CiFTS) `http://www.mcs.anl.gov/research/cifts`
[18] Dahlgren T L and Devanbu P T 2005 *Proc. 2nd Int. Workshop on Software Engineering for High Performance Computing System Applications* ed Johnson P M (St. Louis, MO) pp 73–77
[19] Dahlgren T L 2007 *Proc. 10th Int. Symp. on Component-Based Software Engineering (Boston, MA)* vol LNCS 4608 ed Schmidt H W et al (Berlin Heidelberg: Springer-Verlag) pp 157–172
[20] Dahlgren T, Epperly T, Kumfert G and Leek J 2006 *Babel User's Guide, v 1.0.0* CASC, Lawrence Livermore National Laboratory, UCRL-SM-205559 Livermore, CA
[21] Norris B, Ray J, Armstrong R, McInnes L C, Bernholdt D E, Elwasif W R, Malony A D and Shende S 2004 *Proc. Int. Symp. on Component-Based Software Engineering* (Edinburgh, Scotland)
[22] McInnes L C, Ray J, Armstrong R, Dahlgren T L, Malony A, Norris B, Shende S, Kenny J P and Steensland J 2006 Computational quality of service for scientific CCA applications: Composition, substitution, and reconfiguration Tech. Rep. ANL/MCS-P1326-0206 Argonne National Laboratory
[23] Shende S and Malony A 2006 *Int. J. High-Perf. Computing Appl., ACTS Collection special issue* **20** 287–331
[24] Lucas R (PI) Performance Engineering Research Institute (PERI) `http://www.peri-scidac.org`
[25] Elwasif W, Norris B, Allan B and Armstrong R 2007 Bocca: A development environment for HPC components submitted to Compframe 2007, Montreal, Canada