Computational Quality of Service for Scientific Components

Boyana Norris[1], Jaideep Ray[2], Rob Armstrong[2], Lois C. McInnes[1], David E. Bernholdt[3], Wael R. Elwasif[3], Allen D. Malony[4], and Sameer Shende[4]

[1] Argonne National Laboratory, Argonne, IL 60439 USA
{mcinnes,norris}@mcs.anl.gov ‡
[2] Sandia National Laboratories, Livermore, CA 94551 USA
{rob,jairay}@sandia.gov §
[3] Oak Ridge National Laboratory, Oak Ridge, TN 37831 USA
{bernholdtde,elwasifwr}@ornl.gov ¶
[4] University of Oregon, Eugene, OR 97403 USA
{malony,sameer}@cs.uoregon.edu ‖

Abstract. Scientific computing on massively parallel computers presents unique challenges to component-based software engineering (CBSE). While CBSE is at least as enabling for scientific computing as it is for other arenas, the requirements are different. We briefly discuss how these requirements shape the Common Component Architecture, and we describe some recent research on quality-of-service issues to address the computational performance and accuracy of scientific simulations.

1 Introduction

Massively parallel scientific computing, like its counterparts in the commercial sector, must contend with perpetually increasing software complexity. In scientific domains, software complexity arises from the desire to simulate intrinsically complicated natural phenomena with increasing fidelity. Current high-performance parallel simulations of this nature include climate modeling, nanotechnology, magnetohydrodynamics, quantum chromodynamics, computational biology, astronomy, and chemistry, and more recently, multiscale and multiphysics hybrids of two or more of these.

‡ Research at Argonne National Laboratory was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract W-31-109-ENG-38.

§ Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. Dept. of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

¶ Oak Ridge National Laboratory is managed by UT-Battelle, LLC for the U.S. Dept. of Energy under contract DE-AC-05-00OR22725.

‖ Research at the University of Oregon is sponsored by contracts (DEFG03-01ER25501 and DE-FG02-03ER25561) from the MICS program of the U.S. Dept. of Energy, Office of Science.

The motivation for component-based software engineering (CBSE) in scientific simulations is largely the same as that for other pursuits: components are a uniform way of compartmentalizing complexity in building blocks for applications. In this paper we present a brief overview of the requirements of CBSE for high-performance scientific computing, and we present the Common Component Architecture (CCA) approach, on which the computational quality-of-service (CQoS) work is based.

Component-based environments offer a degree of flexibility over traditional monolithic scientific applications that opens new possibilities for improving performance, numerical accuracy, and other characteristics. Not only can applications be assembled from components selected to provide the best performance, but they can also be changed dynamically during execution to optimize desirable characteristics. The quality-of-service (QoS) aspects of scientific component software that we consider in this paper differ in important ways from more common component-based sequential and distributed applications. Although performance is a shared general concern, high sequential and parallel efficiency and scalable performance is a more significant requirement in scientific component design and deployment. The factors that affect performance are closely tied to the component's parallel implementation, its management of memory, the algorithms executed, and other operational characteristics. In contrast, performance quality of service in nonscientific component software focuses more on system-related performance effects, such as CPU or network loads. The composition of scientific components also affects their individual performance behavior, suggesting the need for QoS metrics that measure cross-component performance.

Scientific component software is also concerned with *functional* qualities, such as the level of accuracy achieved for a particular algorithm. When components can operate under various functional modes while employing the same external interface and can switch between modes during execution, different service requirements can arise. Moreover, the interaction of the functional qualities with the performance qualities of scientific components makes dynamic service mechanisms distinctly important. For example, the selection of an algorithm for a given problem must take into account a possible tradeoff between speed and reliability. When these component-specific QoS concerns are considered globally in the context of the composite component application, opportunities to enhance the computation arise.

We refer to this concept of the automatic selection and configuration of components to suit a particular computational purpose as *computational quality of service* (CQoS). CQoS is a natural extension of the capabilities of the component environment. The name refers to the importance of the computational aspects—both functional and nonfunctional—of scientific components in how they are developed and used. CQoS embodies the familiar concept of quality of service in networking and the ability to specify and manage characteristics of the application in a way that adapts to the changing (computational) environment. We discuss techniques to support CQoS capabilities from the viewpoint of enhancing the computational service being offered.

In this paper, we first overview the background and requirements for CBSE in scientific computation. Next, we briefly describe the CCA. We then discuss the concept of computational quality of service as it applies to components for high-performance scientific applications, and we describe an initial implementation of a CQoS environment that is being integrated with the CCA technology. We conclude with prospects for future work.

2 CCA Overview

Apart from a social reticence to accept solutions not developed within their own scientific communities, researchers are particularly concerned about the performance implications of a relatively new approach such as CBSE. Timescales for message-passing operations on modern supercomputers are measured in microseconds, and memory latencies in nanoseconds. The conventional rule of thumb is that environments that incur a performance cost in excess of 10 percent will be rejected outright by computational scientists. In addition, scientists are concerned about the impact of applying new techniques to extensive bases of existing code, often measured in hundreds of thousands of lines developed over a decade or more by small groups of researchers; extensive rewriting of code is expensive and rarely justifiable scientifically.

While there have been a number of experiments with commodity component models in a high-performance scientific context [1, 2], so far they have not had noticeable acceptance in the scientific community. Unfortunately, various aspects of commercial component models tend to limit their direct applicability in high-performance scientific computing. Most have been designed primarily with distributed computing in mind, and many have higher overheads than desirable, even where multiple components within the same address space are supported. Support for parallel computing is also a crucial consideration. The effort required to adapt existing code to many commercial component models is often high, and some impose constraints with respect to languages and operating systems. For example, in high-end computational science, Java is still widely viewed as not providing sufficient performance, making an approach like Enterprise JavaBeans unattractive; and almost no supercomputers run Windows operating systems, limiting the applicability of COM.

The scientific high-performance computing (HPC) community has made some tentative steps toward componentlike models that are usually limited to a specific domain, for example Cactus [3], ESMF [4], and PALM/Prism [5]. While successful in their domains, these approaches do not support cross-disciplinary software reuse and interoperability.

In response, the Common Component Architecture Forum [6] was launched in 1998 as a grassroots initiative to bring the benefits of component-based software engineering to high-performance scientific computing. The CCA effort focuses first and foremost on developing a deeper understanding of the most effective use of CBSE in this area and is proceeding initially by developing an independent component model tailored to the needs of HPC.

Space constraints require that we limit our presentation of the CCA here; however, further details are available at [6], and a comprehensive overview will be published soon [7]. The specification of the Common Component Architecture defines the rights, responsibilities, and relationships among the various elements of the model. Briefly, *components* are units of encapsulation that can be composed to form applications; *ports* are the entry points to a component and represent interfaces through which components interact—*provides* ports are interfaces that a component implements, and *uses* ports are interfaces that a component uses; and the *framework* provides some standard *services*, including instantiation of components, as well as *uses* and *provides* port connections.

The CCA employs a minimalist design philosophy to simplify the task of incorporating existing HPC software into the CCA environment. This approach is critical for acceptance in scientific computing. CCA-compliant components are required to implement just one method as part of the `gov.cca.Component` class: the component's `setServices()` method is called by the framework when the component is instantiated, and it is the primary means by which the component registers with the framework the ports it expects to provide and use. *Uses ports* and *provides ports* may be registered at any time, and with the `BuilderService` framework service it is possible programmatically to instantiate/destroy components and make/break port connections. This approach allows application assemblies to be dynamic, under program control, thereby permitting the computational quality-of-service work described in Section 3. Furthermore, this approach ensures a minimal overhead (approximately the cost of a virtual function call) for component interactions [8].

Most parallel scientific simulations use a single-program/multiple-data (SPMD) paradigm, in which an identical program runs on every process/processor, using the Message Passing Interface (MPI) [9] or an equivalent message-passing mechanism over an interconnection fabric. This approach sometimes is relaxed to the multiple-program/multiple-data (MPMD) pattern, which includes multiple communicating instances of SPMD programs. Analogously, the CCA's model is that of many "same-process" component assemblies instantiated as a parallel cohort across all participating processes (see Figure 1). In direct contrast with a distributed object model of components (e.g., CORBA), component connections occur within a single process for maximum performance. Interprocess communication, usually MPI, is left to the components themselves without CCA interference. Both single-component/multiple-data and multiple-component/multiple data paradigms are supported, analogous to SPMD and MPMD programs.

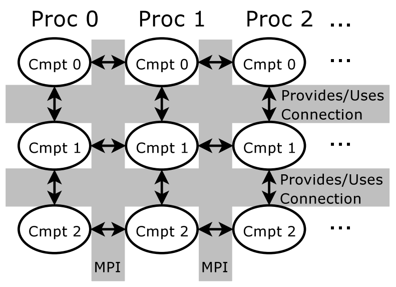

Fig. 1. Components are directly connected to peers in the same process (vertical) and communicate among their own cohorts between processes using MPI (horizontal).

3 Computational Quality of Service

Quality of service is often associated with ways of implementing application priority or bandwidth reservation in networking. Here computational quality of service (CQoS) refers to the automatic selection and configuration of components to suit a particular computational purpose. While CBSE helps to partition complexity in parallel simulations, it also presents its own problems. For example, if data is distributed across all participating processors (Fig. 1), each component must deal with the distributed data as it is presented; it is almost never efficient to redecompose the problem optimally for each component. If the components are thorough black boxes, then there would be no mechanism to optimize this decomposition over all components interacting with it. However, if metadata is provided either as part of the static information associated with the component repository, or as dynamic information computed in real time, a "resource-controller" component could configure its peer components by taking the global situation into consideration (see Fig. 2). This special-purpose component interprets mechanistic, performance, or dependency metadata, provided by its peer components, to make an optimal solution within the context of an entire application or a local container component. For more information on CCA containers, see [10].

This approach not only solves CBSE problems but presents new opportunities, primarily that of being able to dynamically replace poorly performing components. Component concepts help to manage complexity by providing standard building blocks; these concepts also enable a degree of automation at a high level. Here we will describe how CBSE in scientific computing provides opportunities to automate scientific simulations for better performance and accuracy.

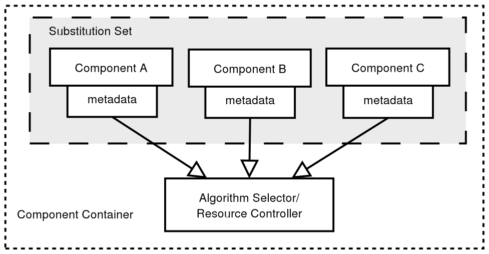

Fig. 2. CQoS component organization.

CQoS metadata may be used to compose or dynamically adapt an application. A detailed design of an infrastructure for managing CQoS-based component application execution was proposed in [11]. The CCA enables the key technology on which CQoS depends, including component behavior metadata and component proxies for performance modeling or dynamic substitution. By associating CQoS metadata with a component's *uses* and *provides* ports, one can effectively express that component's CQoS requirements and capabilities.

CQoS employs global information about a simulation's composition and its environment, so that sound choices for component implementations and parameters can be made. Building a comprehensive CQoS infrastructure, which spans the algorithms and parallel decomposed data common to scientific simulations, is an enormous task but, given the need to automate the cooperation of algorithmically disparate components, a necessary one. The research reported in the rest of this section is a first step toward this aim and thus first addresses problems that interest the scientific simulation community.

Performance Measurement and Monitoring. The factors that affect component performance are many and component dependent. To evaluate component CQoS, one must have a performance system capable of measuring and reporting metrics of interest. We have developed a performance monitoring capability for CCA that uses the TAU parallel performance system [12] to collect performance data for assessing performance metrics for a component, both to understand the performance space relative to the metrics and to observe the metrics during execution. After performance data have been accumulated, performance models for single components or entire applications can be constructed. An accurate performance model of the entire application can enable the automated optimization of the component assembly process.

Automated Application Assembly. CCA scientific simulation codes are assemblies of components created at runtime. If multiple implementations of a component exist (i.e., they can be transparently replaced by each other), it becomes possible to construct an "optimal" CCA code by choosing the "best" implementation of each component, with added consideration for the overhead of any potentially necessary data transformations. This construction requires the specification of quality attributes with which to discriminate among component implementations. In this discussion, we will focus on execution time as the discriminant.

Performance data can be measured and recorded transparently via the proxy-based system described in [13]. Component interface invocations are recorded, resulting in a call graph for the application. The net result of a fully instrumented run is the creation of data files containing performance parameters and execution times for every invocation of an instrumented component as well as a call graph with nodes representing components, weighted by the component's execution time.

Performance models are created through regression analysis of the data collected by this infrastructure. The call-graph is also processed to expose the *cores*, or components that are significant from the perspective of execution time. This processing is done by traversing the call tree and pruning branches whose execution time is an order of magnitude less than the inclusive time of the nodes where they are rooted. Since component performance models can be constructed from performance data collected from unrelated runs or from unit tests, the models consequently scale, at worst, as the total number of component implementations. The final composite model for a component assembly reduces to a summation over the performance models of each of the components in the cores. At any point before or during the simulation, the performance models of each of the component implementations are evaluated for the problem's size to obtain the execution times of any component assembly prior to choosing the optimal set. Once an optimal set of components have been identified, the performance modeling and optimization component, named *Mastermind*, modifies the existing component assembly through the `BuilderService` interface introduced in Section 2.

Adaptive Polyalgorithmic Solvers. While application assembly is typically done once before a scientific simulation starts, often the same set of component implementations does not satisfy CQoS requirements throughout the application's entire execution. Many fundamental problems in scientific computing tend to have several competing solution methods, which differ in quality attributes, such as computational cost, reliability, and stability. For example, the solution of large-scale, nonlinear PDE-based simulations often depends on the performance of sparse linear solvers. Many different methods and implementations exist, and it is possible to view each method as reflecting a certain tradeoff among several metrics of performance and reliability. Even with a limited set of metrics, it is often neither possible nor practical to predict what the "best" algorithm choice for a given choice may be. We are in the initial stages of investigating dynamic, CQoS-enhanced *adaptive* multimethod linear solvers, which are used in the context of solving a nonlinear PDE via a pseudo-transient Newton-Krylov method. Depending on the problem, the linear systems solved in the course of the nonlinear solution can have different numerical properties; thus, a single linear solution method may not be appropriate for the entire simulation. As explained in detail in [14], the adaptive scheme uses a different linear solver during each of the three phases of the pseudo-transient Newton-Krylov algorithm, leading to increased robustness and potentially better overall performance.

4 Conclusion

CBSE provides a mechanism for managing software complexity and enabling hundreds of scientists to participate in the development of large-scale simulation software, something currently lacking in scientific computing. The CCA model of component-based development offers a standard approach to component and application construction that is specific to parallel scientific computing but also generally applicable to many domains within computational science. The CCA has already been proven successful in several scientific domains, including climate modeling [15], combustion [16], and computational chemistry [17].

The emergence of these component-based scientific codes has motivated the development of an abstract infrastructure for describing computational quality-of-service (CQoS) requirements and capabilities. CQoS requires an environment that contains services for monitoring and managing performance data, analyzing static and dynamic performance information, optimizing application assembly, and adaptively substituting components. A CQoS environment should be developed in a manner consistent with a CBSE methodology to maintain coherence with the engineering of scientific component applications. The work described here demonstrates the utility of such an environment and lays the groundwork for it. As parallel computing hardware becomes more mainstream, our hope is to see a corresponding increase in commodity simulation components that can be easily used to build parallel scientific applications.

Acknowledgments. This work has been supported in part by the U.S. Department of Energy's Scientific Discovery through Advanced Computing initiative, through the Center for Component Technology for Terascale Simulation Software. We thank Nick Trebon (University of Oregon) for his help in implementing the performance measurement infrastructure as well as the call graph pruning algorithms. We also thank Sanjukta Bhowmick (Penn State University) for her work in implementing the adaptive polyalgorithmic solvers and Gail Pieper (ANL) for copyediting a draft of this paper.

References

1. Keahey, K., Beckman, P., Ahrens, J.: Ligature: Component architecture for high performance applications. The International Journal of High Performance Computing Applications **14** (2000) 347–356
2. Pérez, C., Priol, T., Ribes, A.: A parallel CORBA component model for numerical code coupling. International Journal of High Performance Computing Applications (IJHPCA) **17** (2003)

3. Allen, G., Benger, W., Goodale, T., Hege, H.C., Lanfermann, G., Merzky, A., Radke, T., Seidel, E., Shalf, J.: The Cactus code: A problem solving environment for the Grid. In: High Performance Distributed Computing (HPDC), IEEE Computer Society (2000) 253–260
4. Anonymous: Earth System Modeling Framework (ESMF). http://sdcd.gsfc.nasa.-gov/ESS/esmf_tasc/ (2004)
5. Guilyardi, E., Budich, R.G., Valcke, S.: PRISM and ENES: European Approaches to Earth System Modelling. In: Proceedings of Realizing TeraComputing - Tenth Workshop on the Use of High Performance Computing in Meteorology. (2002)
6. Common Component Architecture Forum: Common Component Architecture (CCA). http://www.cca-forum.org (2004)
7. et al., D.E.B.: A component architecture for high-performance scientific computing. Intl. J. High Perf. Comp. Appl. (submitted to ACTS Collection special issue)
8. Bernholdt, D.E., Elwasif, W.R., Kohl, J.A., Epperly, T.G.W.: A component architecture for high-performance computing. In: Proceedings of the Workshop on Performance Optimization via High-Level Languages (POHLL-02), New York, NY (2002)
9. Forum, M.P.I.: MPI: a message-passing interface standard. International Journal of Supercomputer Applications and High Performance Computing **8** (1994) 159–416
10. Bernholdt, D.E., Armstrong, R.C., Allan, B.A.: Managing complexity in modern high end scientific computing through component-based software engineering. In: Proc. of HPCA Workshop on Productivity and Performance in High-End Computing (P-PHEC 2004), Madrid, Spain, IEEE Computer Society (2004)
11. Hovland, P., Keahey, K., McInnes, L.C., Norris, B., Diachin, L.F., Raghavan, P.: A quality of service approach for high-performance numerical components. In: Proceedings of Workshop on QoS in Component-Based Software Engineering, Software Technologies Conference, Toulouse, France. (2003)
12. Shende, S., Malony, A.D., Rasmussen, C., Sottile, M.: A Performance Interface for Component-Based Applications. In: Proceedings of International Workshop on Performance Modeling, Evaluation and Optimization, International Parallel and Distributed Processing Symposium. (2003)
13. Ray, J., Trebon, N., Shende, S., Armstrong, R.C., Malony, A.: Performance measurement and modeling of component applications in a high performance computing environment: A case study. Technical Report SAND2003-8631, Sandia National Laboratories (2003) Accepted, 18th International Parallel and Distributed Computing Symposium, 2004, Santa Fe, NM.
14. McInnes, L., Norris, B., Bhowmick, S., Raghavan, P.: Adaptive sparse linear solvers for implicit CFD using Newton-Krylov algorithms. In: Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics. (2003)
15. Larson, J., Norris, B., Ong, E., Bernholdt, D., Drake, J., Elwasif, W., Ham, M., Rasmussen, C., Kumfert, G., Katz, D., Zhou, S., DeLuca, C., Collins, N.: Components, the common component architecture, and the climate/weather/ocean community. submitted to AMS04 (2003)
16. Lefantzi, S., Ray, J.: A component-based scientific toolkit for reacting flows. In: Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics, Boston, Mass., Elsevier Science (2003)
17. Benson, S., Krishnan, M., McInnes, L., Nieplocha, J., Sarich, J.: Using the GA and TAO toolkits for solving large-scale optimization problems on parallel computers. Technical Report ANL/MCS-P1084-0903, Argonne National Laboratory (2003) submitted to ACM Transactions on Mathematical Software.