

Enhancing Performance and Productivity Using the CCA

B. Norris, L. McInnes (ANL); W. Elwasif, T. Wilde (ORNL);
J. Nieplocha (PNNL); B. Allan, R. Armstrong*, J. Ray (SNL);
Collaborators: A. Malony, S. Shende, N. Trebon (Univ. Oregon)

Summary

A primary goal of component-based software design is to enhance developer productivity and application performance. As various scientific application groups adopt the Common Component Architecture (CCA) paradigm, the Center for Component Technology for Terascale Simulation Software (CCTSS) strives to ensure that such benefits are realized in the scientific arena. We summarize the various mechanisms by which these enhancements are being actualized by CCA users.

Taming Software Complexity

One of the simplifications afforded by any kind of modularization is the creation of logically consistent software “modules” or *components*, which individually are not overwhelmingly complex. Such componentization enhances innovation in several ways: (1) because individual components are more easily comprehended, the development of new and efficient algorithms is encouraged; (2) larger and more complex component assemblies become feasible because they can still be grasped by developers; and (3) individual components can be assembled in novel ways. To quantify the software structure of a component-based toolkit for large-scale combustion simulations, researchers in the Computational Facility for Reacting Flow Science (CFRFS) SciDAC project performed a statistical analysis of their code. The bulk of the components were found to contain less than 1,000 lines. Most components had 1 or 2 interfaces and used interfaces from 1 or 2 other components. Interfaces typically had 8 or fewer functions. This analysis indicates a design with sparse interfaces, simple components, and sparse component connectivity, which are characteristics of

codes built hierarchically through the integration of component sub-systems. Such a locality-preserving design allows new researchers to contribute quickly, by simply restricting their knowledge of the software to a particular subassembly.

Proxy-Based Code Analysis

Component-based environments offer a unique opportunity to provide analysis capabilities at component granularity, which are non-intrusive and language independent. By inserting automatically-generated proxies into the connections between components, a wide range of performance, debugging, or other information can be collected without the need to modify any user software. A proxy-based solution has been applied by CFRFS, CCTSS, and SciDAC Performance Evaluation Research Center (PERC) researchers to the performance measurement and modeling of a combustion code. Proxies sit between components to control when performance data is logged and utilize another component based on the Tuning and Analysis Utilities (TAU) library to actually collect the performance data. Instrumenting a CCA application can be as simple as adding the performance measurement subassembly and

* 925-294-2470, rob@sandia.gov

inserting proxies at the point of interest. The same mechanism can be exploited by a *PortMonitor* proxy to log component invocation information, such as arguments. Logged data can be manipulated, analyzed, and even replayed to the component for debugging and testing purposes.

Computational Quality of Service (CQoS)

One of the most significant benefits of components is the use of common interfaces for multiple algorithms and implementations that are functionally equivalent, but may differ in accuracy, stability, or performance. A key challenge in component-based scientific applications is selecting the set of components that best satisfy a set of application-specific quality requirements. These can be traditional metrics, such as overall time to solution, or more algorithm-specific requirements, such as numerical gradient accuracy. As part of our collaborative work with the SciDAC PERC project, we have developed an infrastructure (Figure 1) based on runtime monitoring (using TAU) and logging of relevant metrics from a simulation in a low-overhead runtime database. This information can then be exploited to choose and configure linear solvers for the numerical solution of nonlinear partial differential equations used to describe complex physical processes. We also developed a larger but slower database to archive historical runtime information to conduct offline analyses, refine the parameterization of solvers, and develop automatic, adaptive solver strategies.

Productivity

The CCA is developing mechanisms that automate and encapsulate many of the routine and mechanical aspects of component code development and deployment. We are augmenting the Eclipse integrated development environment platform with tools for developing SIDL-based CCA ports and components. This high-level graphical interface also provides guidance to users who are not accustomed to the component style of

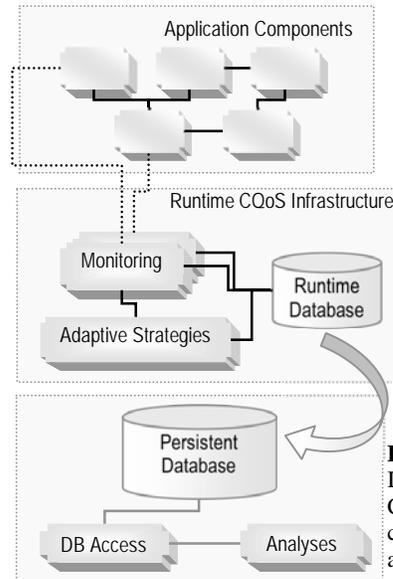


Figure 1. Infrastructure for CQoS in scientific component applications.

programming. Other areas under development include automated build support for components as well as interface generation to facilitate use of legacy code in the CCA environment.

High-Level Abstractions for Managing Multilevel Parallelism

High-level abstractions can improve programmer productivity and provide opportunities for performance optimizations in the underlying implementations. CCTTSS researchers have been developing high-level abstractions for parallel programming based on Global Arrays' shared/distributed array ideas in the context of the CCA. One of the important research goals is to exploit multiple levels of parallelism available in modern scientific applications without substantial modifications of the existing applications. These ideas have been applied in computational chemistry, with a factor of ten performance improvement on one problem.

For further information on this subject contact:

Rob Armstrong (CCTTSS PI)
Sandia National Laboratories
Livermore, CA 94551
Email: rob@sandia.gov
Phone: 925-294-2470
<http://www.cca-forum.org>