Integrating Large-scale Distributed and Parallel HPC (DPHPC) Applications Using a Component-Based Architecture

Nanbor Wang  
Tech-X Corporation  
5621 Arapahoe Ave. Suite A  
Boulder, CO 80303  
nanbor@txcorp.com  

Paul Hamill  
Tech-X Corporation  
5621 Arapahoe Ave. Suite A  
Boulder, CO 80303  
paulh@txcorp.com  

Fang Liu*  
Computer Science Department  
Indiana University  
Bloomington, IN 47405  
fangliu@cs.indiana.edu

ABSTRACT
Component-based software engineering (CBSE) is now a widely accepted paradigm for developing large-scale commercial software. The Common Component Architecture (CCA) and its associated Babel tool suite are designed to enable CBSE for High Performance Computing (HPC) scientific applications. Many scientific applications have adopted the CBSE paradigm and demonstrated its effectiveness using CCA.

Integrating multiple distributed HPC applications into a large-scale system allows the overall system to take advantage of scattered resources and to provide alternative strategies in enhancing system’s overall parallelism and performance. CCA is well-suited to facilitate such applications as it supports both parallel and distributed interaction models. For example, applications can utilize remote visualization and coordination of large-scale physics simulations.

In this paper, we demonstrate how VORPAL, a particle-in-cell (PIC) plasma simulation code, takes advantage of a Distributed and Parallel HPC (DPHPC) approach by constructing various data analysis strategies to perform simulation using distributed computing resources. We provide both qualitative and quantitative comparisons of the capabilities (such as reusability, composability, etc.) and speedup we achieved using a DPHPC approach with a component-based framework.

Categories and Subject Descriptors
D.2.13 [Software Engineering]: Reusable Software, Reuse Models

General Terms
Design, Performance

Keywords
Component-Based Software Development (CBSE), Common Component Architecture (CCA), High-Performance Computing (HPC), Framework, Parallel and Distributed Computing

1. INTRODUCTION
Component-based software engineering (CBSE) [11, 20] is now a widely accepted software paradigm for developing large and complex enterprise applications. CBSE promotes software reuse and simplifies building and maintaining these large applications by separating different aspects of software development, e.g. component interaction models, component implementations, and component composition, into individual entities [23]. There exist many popular enterprise component standards and implementations for different application domains.

The Common Component Architecture (CCA) [4] is a grassroots effort to address the specific needs of HPC communities such as support for scientific programming languages and parallel computing infrastructure, that are not adequately served by the commodity off-the-shelf software frameworks and tools [7]. In particular, the CCA takes an approach that’s neutral to any specific parallel programming model such as shared-memory or message-passing parallel programming models. The de facto standard Babel/Caffeine-based CCA implementation provides the CBSE environment that has benefited many scientific parallel high-performance applications.

Traditionally, most CCA implementations focused on bringing CBSE to the HPC application communities and did not incorporate native distributed component support until recently. Other CCA implementations, such as XCAT [1] focus on providing CCA mappings in the context of distributed environment and do not currently integrate fully with other CCA environments for parallel HPC frameworks. A Distributed and Parallel High-Performance Computing (DPHPC) environment that connects multiple parallel components via distributed objects communication enables a new programming paradigm that allows easy developing of large-scale high-performance applications from multiple parallel components that have been verified and validated. This can be achieved without major rework of parallel code in existing components, which will be complicated and non-adaptable. Furthermore, DPHPC allows applications to take advantage of distributed computing resources and can increase the overall parallelism.

Our past work in the area has been focusing on enabling the DPHPC environment in CCA [22]. In this paper, we present our ongoing work on developing example applications demonstrating the DPHPC approach. The remainder of the paper is organized as following: Section 2 first reviews the benefit for integrating distributed components in large-scale, distributed and parallel HPC applications and our past work on supporting such an environment; Section 3 then illustrates the design of our ongoing example development; Section 4 reviews related work; and finally, Section 5 summarizes our findings and sketches the future direction for our work.

*Additional authors are listed in Section 7 of this paper per ACM sig-alternate style guidelines.
2. DISTRIBUTED HPC COMPONENTS

High-performance parallel scientific codes often require years of development and validation. With the advance in science and computation, more and more physics of different scales are incorporated together to model complicated interactions as a whole. Examples include many large-scale scientific computing projects, such as the Fusion Simulation Project (FSP), Computational Chemistry and Climate Modeling. These projects incorporate computations that model different physics phenomena in vastly different scales.

Integrating multiple existing high-performance and parallel computational codes together using the traditional parallel programming approaches can be a very challenging as each of these parallel codes can employ different parallel programming models. Composing large-scale applications from multiple parallel codes using distributed object communication provides an approach that allows parallel codes to be integrated together with increased parallel performance but without drastic design. It also allows applications to benefit from connecting and utilizing vast amount of distributed and/or parallel computing resources.

The following use cases further exemplify the driving force of integrated distributed and parallel technologies into HPC applications.

- Parallel applications often adopt certain middleware technologies, such as MPI or OpenMP, to provide portable abstraction for communication among parallel processors. However, these parallel message exchange middleware frameworks provide a rather low level of abstraction and are often tightly coupled with applications. It is very hard to integrate these codes at the level of these parallel middleware frameworks. Furthermore, integrating all these code into one parallel code running on massive number of processes may not scale. Due to the inherent low-level interaction and the lack of hierarchy, it is also very hard to benchmark, adapt and load-balance the resulting applications. Loosely coupling these multi-physics/multi-scale codes using distributed interfaces provide a high-level of abstraction for inter-code interaction and could allow a greater degree of parallelism.

- Another approach commonly adopted by computational scientists is to handle the data generation, analysis, and visualization in batch mode. Using this approach, users need to buffer the generated data files locally in the storage available at the supercomputing facility. However, most users often have limited disk quota for storing generated data. As a result, users are forced to decide what data might be of interests and deserve further analysis and discard the remaining data before any analyses are done. To avoid wasting precious computational cycles, it is sometimes suitable to collect as much data as possible to explore new scientific discoveries by streaming all potentially interesting data out of the supercomputer to a remote data storage in real time [8]. Because multiple connections can effectively expand the TCP window-size, we can fully utilize the network of high-bandwidth and high-latency networks [3] commonly found among major research institutes and supercomputing centers.

- Parallel physics components are not necessarily optimized for the same parallel architecture. For example, one physics component might be internally tightly coupled and require a shared-memory machine to run well. Another physics components might be able to take advantage of multi-level parallelism and run best on a parallel vector machine. A third physics component might not benefit at all by running on anything but a standard Beowulf cluster. In cases like these, a loosely coupled integrated physics application composed using distributed components would allow optimal use of computational resources.

- Many peripheral resources, such as disk storage, at supercomputing centers are often limited to users. It is not uncommon for users to perform only the computationally intensive work at those machines and transport the resulting data back to local clusters or desktop machines for post-processing, data analysis, and visualization. Such resource restrictions can also stem from a lack of software tools on supercomputers commonly used by scientists. For example, data visualization is often not done remotely at a supercomputer due to a lack of hardware supports at supercomputing center and the inefficiency incurred by running remote visualization over the X protocol. Connecting numerical computation jobs with remote data analysis and visualization applications allows users to streamline the workflow of scientific explorations.

As we can see in Section 4, these issues have long been observed by computation and computer scientists alike. With the advance of both available computing resources, either geographically distributed or centralized leadership-class supercomputing facilities, and the knowledge and experience in massively parallel, multi-physics and multi-scale computation, the aforementioned issues become a more significant challenge facing modern computational scientists. Many ongoing research efforts in designing parallel applications using the Multi-Program Multi-Data (MPMD) model or the Multi-Level Parallelism (MLP) are trying to address various challenges. Integrating parallel codes using distributed technologies has shown to be an effective approach [16].

The DPHPC environment utilizes both the distributed and parallel capability in the CCA framework to present an alternative to support MCMD application composition. Distributed CCA is by no means a novel idea. There exists several CCA implementations, such as XCAT [1] and SciRUN [2], that provide distributed component capability. However, they do not currently interoperate with each other or other CCA frameworks that target high-performance parallel components.

We have previously explored various strategies in connecting distributed parallel applications [21]. Figure 1 illustrates how two component-based parallel applications can be integrated using DPHPC approach. Applications can utilize asynchronous messaging [9] to enhance the parallelism of the overall systems. The addition of Remote Method Invocation (RMI) support in Babel [14] opens up the door to easily compose existing parallel CCA components through connections of distributed Babel interfaces with little or no modification.

Thanks to Babel RMI’s open architecture approach, we have implemented the TAOIOP Babel RMI library that enable user to connect distributed CCA applications into a large-scale systems using industrial standard CORBA protocol. We have conducted performance benchmarking on TAOIOP implementations and optimized our implementation. We have shown that we can achieve interconnection using the standard distributed object computing protocol with minimal performance penalty.

3. DPHPC COMPONENT-BASED APPLICATION EXAMPLES

As a proof of concept of our component-based application framework, we are developing reference applications that demonstrate...
invoking distributed VORPAL components to asynchronously perform simulation tasks while collaborating with a service component. VORPAL is a high-performance plasma simulation code designed for maximum flexibility through use of advance C++ techniques. Through use of inheritance and meta-template programming, VORPAL incorporates multiple models for the plasma and electromagnetic fields with different dimensions. VORPAL also supports both MPI and parallel I/O. This enables numerous scenarios in which these reference applications can be of practical use in helping physicists to more optimally perform compute-intensive simulations, either on high-performance supercomputers or on more common commodity-class clusters.

One such scenario is distributed visualization. The popular scientific visualization package VisIt is often used to produce high-fidelity interactive visualizations. However, VisIt’s default usage requires pre-processing and assembling the data on a local filesystem for the visualization tools to read. Given the extremely large size of some simulation data, and the fact that processing often occurs across distributed parallel nodes, preparing the data is often a time-consuming serial task that occurs long after the simulation is completed. Using VisIt’s API for remote data streaming and our DPHPC component framework, we intend to explore a far more optimal model of visualization in which blocks of data processed on distributed nodes are streamed directly to VisIt as they are produced, without a need for intermediate data aggregation.

Data could potentially be subdivided by geometric region as well as by timestep. In this way, we hope to show that visualization can be efficiently integrated into the distributed parallel processing paradigm, permitting data to be viewed almost as soon as it is produced. If this effort is successful, it could lead into further functional optimization and efficiency improvement by enabling near-real-time simulation steering. If scientists are able to visualize data as the simulation is running, they may be able to alter and optimize their simulations much earlier during the run, saving both “wall clock” time and compute resources.

Another usage scenario we intend to explore is for general-purpose post processing of simulation data. In conventional approaches, results generated on each node are saved to files, which are then copied and aggregated either at the end of a time step, or at the end of the simulation run. This process introduces inefficiencies, as gathering results can depend on synchronization of simulation steps between nodes, as well as significant duplication of data on disk across compute units. However, by building on our distributed component architecture, we can instead use loosely coupled asynchronous messages to communicate tasks and data between worker and coordinator components. This way, data can be received as it is generated and sent to post-processing steps, rather than relying on synchronized simulation step completion and inefficient file handling operations. Through this approach, we hope to obviate the need to store intermediate results on disk on compute nodes, as well as relaxing requirements that all nodes process the same time step in unison.

We are building these reference applications by MPI-enabling simple server and client applications that build on our component framework to communicate using either Babel RMI’s built-in Simple Protocol or TAOIIOP messaging. MPI enables simple parallel execution and process-level coordination across an arbitrary number of nodes. Babel RMI messaging functionality provides loosely coupled, reliable, asynchronous communication of application-specific data and events. With this basic approach, rapidly invoking complex distributed processing application environments becomes a relatively simple operation.

The first of our preliminary examples demonstrates how to perform real-time “post-processing” of simulation data on-the-fly using a separate cluster. As shown in Figure 2, simulation was run in parallel on a remote cluster. A parallel data analysis application was running at a local cluster acting as server and calculate the cell density for the simulation job at remote cluster. As mentioned earlier, we utilize the “nonblocking” and “oneway” operation semantics to reduce the impact of distributed communication. The results show approximately 20% speedup by using a remote data analyzer compared to performing the data analysis on the remote cluster directly. We should be able to gain further speedup if data analysis requires even more CPU cycles.

A second example demonstrate how to monitor a parallel VORPAL simulation running on remote cluster by sending history data to a local server. Unlike the first data analysis, the monitoring server is only running on a single process (non-parallel.) For both examples, we have to defined the mapping between VORPAL and data/monitor server if the number of client and server processes can not be mapped one-to-one. This is due to the lack of group concept in our implementation (and Babel/CCA). We will explore a similar abstraction for groups in our future work.

4. RELATED WORK

There are several ongoing efforts in the CCA community to support distributed CCA applications. As we already mentioned, the XCAT project developed by the Indiana University and SUNY Bing-
hampton, concentrates on exploring a CCA infrastructure over a
distributed model. It currently does not interact with other CCA
implementations. However, researchers at SUNY Binghamton are
working on a high-performance Web Service-based Babel RMI li-
brary that can support both the flexible nature of Web Services and
the performance offered by most binary-based wire-protocol [19]
such as CORBA.

The SciRUN team at University of Utah is developing the meta-
component model in the SciRUN-2 project [24]. It will provide in-
teroperability between Caffeine/Babel components inside a SciRUN-
2 runtime. Like our distributed components, this interoperability
will enable scientists to take advantage of Babel RMI work and
connect numeric computation with remote SciRUN data process-
ing and analysis process. Furthermore, SciRUN already support
CORBA as one of its component models. Babel/Caffeine applica-
tion using distributed CCA systems can interoperate with SciRUN2
either via CORBA directly, or through a specific Babel RMI library.

The Portable Parallel CORBA Object (Paco++) [15] in the Paris
Project developed by la Institut de Recherche en Informatique et
Systèmes Aléatoires (IRISA), France, takes a different approach to
facilitate deploying parallel objects over a distributed network.
Their work focuses on deploying and representing a series of tightly-
coupled parallel CORBA objects as a single identity. Although
their work does not required any modification to the CORBA spe-
cification, OMG recently adopted the Data Parallel CORBA Spec-
ification [17]. This new specification is synergistic both to Paco++
and to our work.

Fractal/ProActive is an example implementation of Grid Compo-
nent Model (GCM) [10] that’s currently under development within
the CoreGRID and GridCOMP European projects. Fractal/ProActive
provides a parallel and distributed component model that specifi-
cally targets grid applications. Compared to our approach, Frac-
tal/ProActive’s component model covers a much wider space from
different definitions and interactions, to component deployment
mechanisms. It also defines group interfaces such as Multicast and
Gathercast, for supporting operations on parallel objects [6].

Because of all the well-defined interfaces and mechanisms, it
enables the application of software design support to realize com-
ponentized scientific applications in a Grid environment [18] easi-
ly. JEM3D [5] demonstrates how a numerical solver for the 3D
Maxwell’s equations modeling the time domain propagation of elec-
 tromagnetic waves can be implemented using the Fractal/ProActive
model. Although Fractal/ProActive model provides a well-defined,
efficient and robust abstraction for parallelization via object-oriented
distributed interfaces, it often requires a rewrite of existing code
since few scientific codes are development in Java, which Fract-
tal/ProActive supports 1. Furthermore, despite the benefit of de-
composing high-performance numerical code using object-oriented
interfaces, most scientists are not familiar with the approach.
Instead, they often apply messaging passing and shared memory pro-
gramming models in their high-performance codes.

In comparison, although both utilizing distributed interfaces to
achieve better degree of parallelism, DPHPC targets at enhanc-
ing existing parallel codes to enable them to talk to each other
via distributed interfaces. Our work does not cover deployments
and group interfaces although both are important features that we
would like to cover in our future work.

Researchers have been trying to manage complex multi-scale
and/or multi-physics parallel applications for a while. Two on-
going fusion simulation projects, SWIM and FACETS, are both
trying to couple various simulation codes together. Multi-Program

---

1There is also a C implementation of ProActive available.

Multi-Data programming models are generally too complicated for
building using the low-level MPI middleware. However, many re-
searchers pointed out that using a multi-level parallelism (MLP)
can make managing the complexity of MPMD applications more
scalable.

For example, in the recent CCA Meeting (Jan. 2007), researchers
presented their work on this topic in a Multi-Component Multi-
Data (MCMD) workshop. MCMD is the solution proposed by the
component community to address the MPMD development chal-
egg. For example, researcher from the Pacific Northwest Na-
tional Laboratory presented their work on supporting (MLP) us-
ing both MPI and Global Array [13]. Moreover, researchers from
the Lawrence Livermore National Laboratory have presented their
work on applying Babel RMI to support a dynamic MCMD appli-
cation. They call their approach Cooperative Parallelism [16, 12].

5. DISCUSSION AND FUTURE WORK

The grass-root CCA efforts are tailored towards building a com-
mon architecture for building large scale, high-performance sci-
centific applications from well-tested software components. As
the CCA community begins to focus their effort in supporting MCMD
application development, our work on DPHPC environment presents
an alternative to support MCMD application composition. We have
implemented the distributed proxy components and the TAOIIOP
Babel RMI library for connecting distributed CCA applications into
a large-scale systems. Such a capability can both enhance the de-
gree of parallelism of the overall applications and allow better uti-
лизation of computing resources.

We are developing an example application that is composed of
multiple parallel applications running on multiple clusters for dis-
tributed data analysis and visualization. In this presentation, we
will demonstrate how we are extending VORPAL, a particle-in-cell
plasma simulation code, to offload post data processing and real-
time visualization using DPHPC approach.

6. ACKNOWLEDGMENTS

This research is funded by DOE Office of Advanced Scientific
Computing Research (grant #DE-FG02-04ER44099) and by Tech-
X Corporation. We would like to express our thanks to the CCA
community and the Babel team at the Lawrence Livermore Na-
tional Laboratory for their support and fruitful discussions.

7. ADDITIONAL AUTHORS

Additional authors: Stephen Tramer (Tech-X Corporation, email:
tramer@txcorp.com), Rooparani Pandaleeka (Tech-X Corpor-
rion, email: roopa@txcorp.com) and Randall Bramley (Indi-
ana University, email: bramley@cs.indiana.edu).

8. REFERENCES

L. McLnes, S. Parker, and B. Smolinski. Toward a
Common Component Architecture for High-Performance
High Performance Distributed Computing, Redondo Beach,
CA, 1999. IEEE.
Kasmi, and S. Lanteri. A Parallel Object-Oriented
Application for 3D Electromagnetism. In Proceedings of
IEEE International Parallel and Distributed Processing Symposium (IPDPS 2004), Santa Fe, NM, Mar. 2007.


