

Coupling GAMESS via Standardized Interfaces

Fang Peng, Meng-Shiou Wu, Masha Sosonkina, Ricky A. Kendall, Michael W. Schmidt, Mark S. Gordon

Abstract—GAMESS, a software package for electronic structure calculations, enjoys great popularity among high-performance application scientists. It contains a variety of parallel methods and provides a sophisticated distributed data interface. This paper presents experiences in designing interoperability features for parallel GAMESS. In particular, the newly developed Common Component Architecture (CCA) components for GAMESS are described. They adhere to the existing "general" CCA chemistry interfaces, which enable dynamic coupling of GAMESS with other quantum chemistry packages, such as NWChem. To justify the versatility of the design, the Tuning and Analysis Utility (TAU) components have been coupled with GAMESS-CCA, so that the performance of GAMESS may be analyzed for a wide range of system parameters. While both TAU and NWChem have been integrated with GAMESS under the same component architecture, the integration procedures took different paths. The paper explains these differences, proposes possible integration solutions, and emphasizes general lessons learned.

Index Terms— Common Component Architecture, GAMESS, NWChem, TAU, DDI

I. INTRODUCTION

THE General Atomic and Molecular Electronic Structure System (GAMESS) is an *ab initio* quantum chemistry program, which has been under development for more than twenty years [1]. GAMESS is able to solve a wide range of quantum chemistry computations including Hartree-Fock (HF) wavefunctions (RHF, ROHF, UHF), GVB, and MCSCF using

Manuscript received April 10, 2006. This work was performed under auspices of the U. S. Department of Energy under contract W-7405-Eng-82 at Ames Laboratory operated by the Iowa State University of Science and Technology. Funding was provided by the Mathematical, Information and Computational Science division of the Office of Advanced Scientific Computing Research. This research used resources of the Scalable Computing Laboratory at Ames Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

Fang Peng is with the Scalable Computing Laboratory, Ames Laboratory, US DOE. (phone: 515-441-294-9469; fax: 515-294-4491; e-mail:fangp@scl.ameslab.gov)

Meng-Shiou Wu is with the Scalable Computing Laboratory, Ames Laboratory, US DOE. (e-mail: mswu@scl.ameslab.gov)

Masha Sosonkina is with the Scalable Computing Laboratory, Ames Laboratory, US DOE. (e-mail: masha@scl.ameslab.gov)

Ricky A. Kendall is with National Center for Computational Sciences, Oak Ridge National Laboratory, US DOE. (e-mail: kendallra@ornl.gov)

Michael W. Schmidt is with the Scalable Computing Laboratory, Ames Laboratory, US DOE. (e-mail: mike@si.fi.ameslab.gov)

Mark S. Gordon is with the Scalable Computing Laboratory, Ames Laboratory, US DOE. (e-mail: mark@si.fi.ameslab.gov)

the self-consistent field method [1]. It is installed on many high performance computing systems, including those at most DOE, DOD, and NSF supercomputer centers, many academic institutions, and widely in the private sector. It is also part of the standard benchmark suites employed, for example, by NERSC, by the High Performance Computer Modernization Program, and by several computer companies (e.g., IBM). The number of GAMESS users is estimated to be on the order of 100,000.

Most of the source code of GAMESS is designed with FORTRAN 77 since it was the most popular programming language for scientific computing at the time the project started. While portability can be achieved through this design (every modern cluster has a FORTRAN 77 compiler), incorporating an external module or interacting with other scientific packages can be very difficult since scientific packages developed in recent years seldom use FORTRAN 77 exclusively.

During the last few years much research effort has been aimed at developing architecture to provide interoperability for high performance scientific software, and the Common Component Architecture (CCA) [2] is constructed for this purpose. CCA provides a framework for components from different packages to be dynamically loaded to solve a computational problem, without knowing which programming language was used to design a component [3]. This provides us the opportunity to allow GAMESS and other scientific packages to interoperate seamlessly with minimum modification to the GAMESS source code. Most CCA frameworks use Babel [4], the language interoperability tool, for solving the interoperability of components that are implemented in different programming languages such as Fortran, C, C++, Python, and Java. Without such a component model, data exchange between two scientific packages can only be accomplished through a large amount of file recoding. Although there exist many other frameworks that support component-based applications, such as CORBA [5], COM [6], and JavaBeans [7], they are not designed for parallel computing and are hardly used to create components for high performance scientific programs. The Common Component Architecture was designed for the component-based application parallel High Performance Computing (HPC).

In the Common Component Architecture, the *components* are basic units of software that are composed together to provide a run-time component environment [2]. Instances of components are created and managed within a *framework*, which provides the basic services for components to operate and communicate with each other [2]. *Ports* are the fully

abstract interfaces, through which components interact with each other and with the encapsulating framework [2]. A component must declare its *Provides* port to provide its own functions or services for other components to use, and also registers its *Uses* ports to connect references to *Provides* ports that are provided by other components or by the containing framework [2]. The communications between different components or between components and frameworks are enabled by connecting matched *Provides-Uses* port pairs through the framework.

CCA supports SPMD (Single Program Multiple Data), MPMD (Multiple Program Multiple Data) and distributed programming models. In this paper, we only discuss CCA in the SPMD programming model since the other two programming models are not used by the applications in our research [8]. When a CCA framework, such as Ccaffeine [2], is running in a parallel environment, each process has its own instance of a CCA framework, and an identical set of component instances and connections are loaded into each framework [8]. The set of similar component instances that are distributed across parallel processes can communicate with each other by using any available communication system, (i.e. MPI [9], PVM [10], Global Arrays [11], or shared memory), while each framework instance that contains the identical set of component instances and connections manages the interactions among component instances within its own process [8]. Different sets of component instances are allowed to use different communication systems simultaneously under the same framework [8]; this is useful for the integration of legacy codes under CCA frameworks since legacy software usually has its own communication mechanisms.

In this research, we implemented a GAMESS CCA interface in two different parallel models: GAMESS/DDI and GAMESS/DDI/MPI models. GAMESS uses the Data Distributed Interface (DDI) [12] as its parallel communication mechanism, which mainly relies on TCP/IP sockets for communication. Integrating the GAMESS/DDI system with CCA and constructing a new parallel model for GAMESS under the component architecture is our first research contribution. Besides DDI, the Message Passing Interface (MPI) can also be used for GAMESS communications and a different mechanism has been developed for integrating GAMESS with MPI. In this mechanism DDI depends on MPI, instead of TCP/IP sockets, as the communication method. Since MPI is a widely used message passing interface, the GAMESS CCA components in this model are easily compatible with other components within CCA frameworks. Our other contribution is to develop a GAMESS/DDI/MPI model for GAMESS under the component architecture. To test the compatibility of the GAMESS CCA components, we integrated GAMESS with other two packages, TAU [13], a package for measuring performance, and NWChem [14], another large quantum chemistry package. The paper is organized as follows. Section II explains our design choices for creating the GAMESS CCA components and introduces the Chemistry Component Toolkit, the testbed for the GAMESS CCA components. In Section III,

initial experiments of coupling the GAMESS component are presented.

II. GAMESS CCA COMPONENTS

A. The Structure of GAMESS Computations

There are three fundamental computations for quantum chemistry calculations: energy, gradient, and Hessian [13]. To run a calculation in GAMESS, e.g. a Hessian calculation, a set of input options are needed, such as the type of wave function, the point group symmetry of the molecule, nuclear coordinates, and the atomic basis sets. After GAMESS is initialized, it reads input options, decides the run type (computation), goes to the driver program for the specified types of calculations, and finally outputs the results.

GAMESS can be used on a wide range of parallel platforms. To achieve high performance as well as exploit the advances in HPC hardware and software, the communication mechanism of GAMESS has been constantly improved and the message passing library has been moved from the original TCGMSG [16] to the current Distributed Data Interface (DDI) [12]. DDI is a lightweight communication library that is based on TCP/IP for portability. This design makes it possible for GAMESS to be a self-sustained software suite, not relying on other communication packages. Thus GAMESS may run on any cluster regardless of the presence of an MPI implementation. DDI provides a large distributed array to all nodes by combining memory in individual compute nodes [4]. The distributed array is mainly used in computations that need large data structures, which are very common in many chemistry computations.

In the DDI communication model, two processes are normally assigned to a CPU, with one process performing the computational tasks, while the other exists solely to store and serve requests for the data associated with the distributed array [12]. There are some cases, in which a data server is not required, such as when using DDI over one-sided message

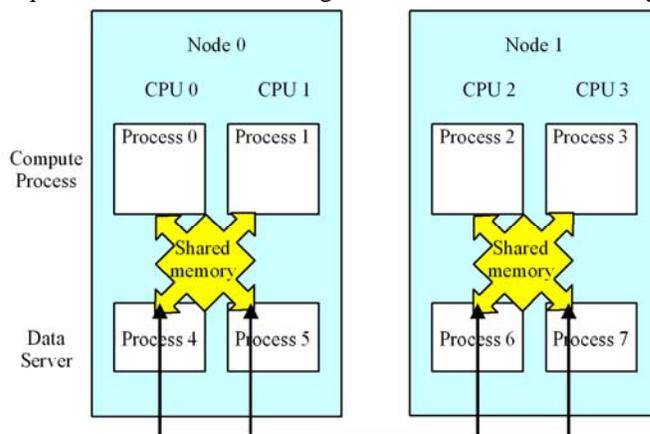


Fig. 1. When DDI is used on an SMP cluster, all DDI processes within a node can access the distributed array in the node. The communications between data servers among different nodes depend on the communication mechanism configured with DDI (i.e., TCP/IP sockets, or MPI) [12].

libraries¹. In this paper, we only consider the cases when data servers are needed. On an SMP machine or cluster (Figure 1), all the DDI processes (both compute and data server processes) within a node have direct access to all distributed array segments in the shared memory of that node. Thus, each compute process and data server can use system shared memory operations, such as copy or paste, locally to access the portion of a distributed array in its local shared memory without using any parallel communication mechanisms. Depending on the platform, communications between compute processes and data servers among different nodes occur either via TCP/IP sockets connections or MPI [12]. When DDI uses TCP/IP sockets for communication, the DDI kickoff program is used for starting the required number of processes on every requested machine in the cluster that will run the job. If MPI is used as the communication mechanism, then `mpirun` (or `mpiexec`) is used to start GAMESS processes.

B. Chemistry Component Toolkit

Most quantum chemistry packages perform fundamental chemistry calculations. Although existing chemistry packages may have a lot of overlapping functionalities, some of them may be more efficient in certain calculations while others may provide special functionality. The CCA provides an environment for different quantum chemistry packages to communicate with each other, and opens the possibility to utilize the best of each package. The Chemistry Component Toolkit (`cca-chem`) [17] already integrates several quantum chemistry packages, optimization solver packages, and parallel data management packages to perform geometry optimizations. The interface for chemistry components is a mixture of components and non-component classes that are instantiated and shared by components [18]. In Kenny *et al.*'s paper [18], the usability issues of `cca-chem` are also discussed.

The generic interfaces in the chemistry components for the quantum chemistry calculations include *Model*, *ModelFactory*, *Molecule*, and *MoleculeFactory*, where the general implementation of *Molecule* and *ModelFactory* interface are available for all the component-based applications. The *Model* interface declares the primary functions in quantum chemistry computations, such as evaluation of molecule energies, gradient and Cartesian Hessians. The *ModelFactory* interface declares methods to provide model options and initializes the model class. Similarly, the *Molecule* interface declares functions for gathering information of a molecule, such as Cartesian coordinates and atomic number. The *MoleculeFactory* interface declares functions to instantiate molecule classes [18].

Figure 2 shows an application example of the chemistry components under the CCA framework. The molecule factory component, model factory component and a driver component are instantiated under a single CCA framework. The model factory can get the reference of the molecule class through the *Provides* port of the molecule factory and invoke the method of

the molecule class. Similarly, the driver component can get the reference of the model class that instantiated and initialized by the model factory, and then invokes the methods of the model class, such as `get_energy`, `get_gradient`, and `get_hessian`. The driver component will also output calculation results from the model factory.

The quantum chemistry packages MPQC [19] and NWChem have already built their component-based applications by integrating to the Chemistry Component Toolkits. To provide interoperability between GAMESS and NWChem or MPQC, we decided to develop the analogous GAMESS CCA components based on the same generic chemistry interfaces/implementations of the chemistry components.

The integration of GAMESS into the chemistry optimization architecture consists mostly of the implementation of *Model* and *ModelFactory* interfaces and the integration of GAMESS and DDI with the CCA framework. The GAMESS CCA components are developed in C++, thus a Fortran 77/C wrapper for GAMESS is required for passing parameters and returning results between the component and GAMESS program. Since CCA is a light-weight framework, we can expect minimum performance impact on GAMESS. The implementation of the chemistry interfaces is quite straightforward, since it mostly follows the way of NWChem and MPQC CCA components. The GAMESS CCA components differ from the existing chemistry components in the requirement of the point symmetry group input from users. GAMESS depends on user input for determining the point symmetry group for efficient calculations, while CCA chemistry components assume the quantum chemistry package itself can detect the symmetry group.

C. GAMESS/DDI Model under CCA Framework

Simply implementing *Model* and *ModelFactory* interfaces for the GAMESS CCA components is not enough for

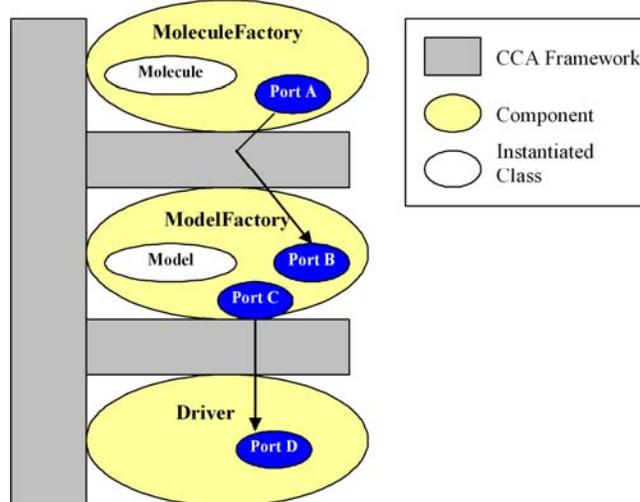


Fig. 2. Port A is a *Provides* port that is implemented by the molecule factory, through which the reference of the molecule class is passed to other components. Port C is a *Provides* port that implemented by the model factory, through which the reference of the model class is passed. Port B and port D are *Uses* ports that are registered by the model factory component and the driver component for using the service provided by other components.

¹ DDI relies on LAPI or SHMEM libraries rather than TCP/IP on some high-end parallel systems

GAMESS to run under the CCA framework, since GAMESS relies on DDI to start the computation, either sequential or parallel. Figure 3 shows the sequence of how the DDI kickoff program starts GAMESS or other programs.

First, the DDI kickoff program needs the program name and the host list as command-line arguments; the host list is a list of host machine name and the number of processors in each node. The master DDI kickoff process analyzes the host list to catch the information on how many compute processes and data servers reside on each host machine. Second, a copy of the DDI kickoff program, along with information about host machines is spawned on each remote host in binomial order. As soon as a copy of the DDI kickoff program is launched on a host node, it creates the requested number of compute and data server processes on that host machine. Finally, a copy of the GAMESS program, with the host machine list, socket ports, host machine and process identities as the command-line arguments, starts on each computer and data server process. The TCP/IP socket connections between a DDI kickoff process and a compute or data server process on the same host machine is created after the program starts the DDI initialization procedures. The DDI kickoff process on each host machine will wait for each compute and data server process to check in by listening to TCP/IP socket connections. As soon as all compute and data server processes are checked in, the communication is established for all compute and data server processes.

Since TCP/IP is the major communication mechanism used by DDI for the communications between compute processes and data servers, we first need to construct the GAMESS CCA components under the GAMESS/DDI model. As long as the GAMESS CCA components work under the GAMESS/DDI

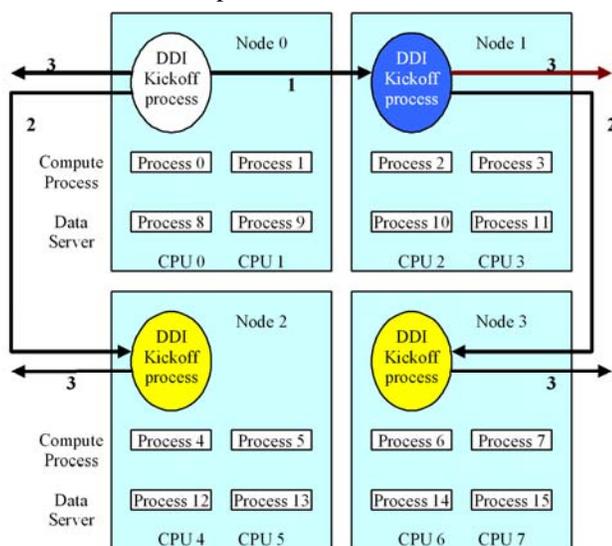


Fig. 3. The numbers along with the arrows show the sequence of how the DDI kickoff program starts the remote DDI kickoff processes. First, the DDI kickoff program starts the master DDI kickoff process (the white one) in Node 0. Then, it starts a copy of remote DDI kickoff process (the blue one) in Node 1. Both DDI kickoff processes in Node 0 and Node 1 will send commands to start the remote DDI kickoff processes (the yellow ones) on Node 2 and Node 3. Next, all the DDI kickoff processes will start the remote DDI kickoff processes in other Nodes if needed. The same procedure will continue until all the required nodes have a copy of DDI kickoff program running. Finally, each copy of the DDI kickoff program will create one compute process and one data server process on each CPU and GAMESS (or other programs) will be running in each compute/data server process.

model, it should also work when other communication libraries are used instead of DDI. Eventually, we expect to develop a model that applies to the SPMP or MPMD model of the CCA framework and also maintains the performance of GAMESS.

The DDI kickoff program is used to start the requested number of compute processes and data servers in each node. An instance of CCA framework will be started on each compute process/data server. Each instance of the CCA framework will then initialize components and build connections between components and between components and the framework according to user inputs. All the components and connections contained in a framework are identical on each process. The GAMESS CCA components contained in the framework of each process will start a DDI initialization procedure for that process. However, in this case, only the GAMESS CCA components have the communication ability because only GAMESS uses DDI as the communication mechanism. The CCA framework or other components under the same framework cannot communicate with each other within processes, since under the GAMESS/DDI model, DDI uses TCP/IP sockets as communication tools while other communication mechanisms used by the CCA framework or other components, such as MPI, have not been initialized.

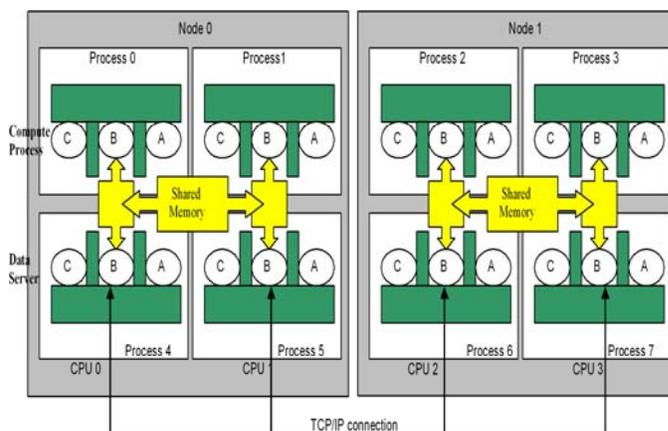


Fig. 4. Under this model, one compute process/data server pair is created for each CPU. The CCA framework (green part) is running on each compute process and data server. A is the driver component, which gets the model object from B (the GAMESS component) through *Provides/Uses* ports. C is the MoleculeFactory component, which provides the molecule object to the GAMESS CCA component. The yellow area is the portion of distributed arrays that are stored in the local shared memory of a node, where the compute processes and data server processes can directly access. The communication of processes among different nodes is through the TCP/IP sockets connections.

Figure 4 shows a simple structure of the GAMESS/DDI model under the CCA framework. The DDI kickoff process on each node first starts one compute process and one data server for each CPU of that node, and then each compute process and data server starts an instance of the CCA framework. The framework and the component instances and connections that are contained in the CCA framework are identical for all processes. As long as the DDI initialization procedure succeeds and the communication layer of DDI is established, the GAMESS CCA components within the same node can directly access the distributed arrays that are stored in the local shared memory of that node, and the GAMESS component in data servers among different nodes can communicate with each other by using TCP/IP.

The major difficulty we encountered in designing this model is passing command-line arguments from the CCA framework to the GAMESS CCA components. The GAMESS component has to start the DDI initialization procedure, instead of the CCA framework, and the command-line arguments must be passed from the DDI kickoff program to the CCA framework on each process. Without the command-line arguments, DDI initialization cannot connect with the corresponding DDI kickoff program in that host machine, and the communication layers cannot be established correctly. To solve the problem, the Stovepipe Library provided by the CCA framework is used to convey the argument list from the CCA framework to the GAMESS CCA components.

D. GAMESS/DDI/MPI Model under CCA Framework

DDI also supports a mixed MPI/TCP model in which processes are started with the MPI startup program instead of the DDI kickoff program. In this model the compute process/data server mechanism is also used, such that for each CPU, there are one compute process and one data server process. Also, processes in the same node have direct access to the portion of distributed arrays in the local shared memory of that node. This is different from the previous model in two ways. First, both MPI and TCP/IP are used for communication between processes among different nodes. MPI is used to pass the actual data, such as a part of distributed arrays, when a process tries to access the portion of the distributed arrays that is not in its local shared memory. The TCP/IP is used for some smaller messages, such as a system call for waking up a sleeping process. The mixed message passing method is used, since most MPI implementations require a process to continuously check for the incoming calls. Thus, using pure MPI will make a data server compete for CPU resources with compute processes. In the TCP/IP implementation, while waiting for a request, each data server process is put to sleep, thus essentially yielding full CPU access to the compute process [12]. Therefore, the mixed MPI/ TCP model for DDI should out-perform using pure MPI.

The second mechanism for the GAMESS CCA components is based on the mixed MPI/TCP model of DDI. This model allows parallelization of the CCA framework, since the CCA framework also uses MPI as one method of passing messages between processes. The CCA framework will start the MPI

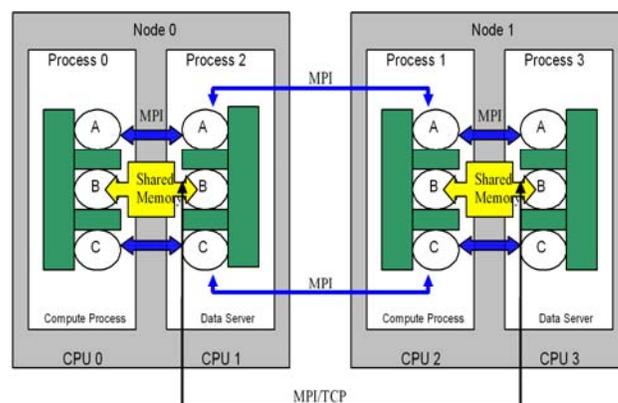


Fig. 5. Under this model, half of the processes will be assigned as compute processes and the other half will be assigned as data server processes by the DDI initialization procedure. The CCA framework (the green area) is running on each process, where A is the driver component, which gets the model object from the GAMESS component through the CCA *Provides/Uses* port. C is the MoleculeFactory component, which provides the molecule object to B, GAMESS CCA component. The yellow area is the portion of distributed arrays in the local shared memory of a node, to which the compute processes and data server processes have direct access. The communication of all the processes for A and C is enabled by MPI. The communication of the processes in the different nodes for GAMESS CCA component is enabled by either MPI or TCP/IP.

initialization procedure, and the DDI communication level will be initialized by the GAMESS CCA components. To enable communication in GAMESS, DDI requires an even number of processes in each host machine; such that the processes can be divided equally into compute processes and data server processes. This will not affect the communication of the CCA framework and other components under the same framework, since DDI just gathers information from MPI Common World group without modifying anything in the configuration of MPI program.

The general structure of how the GAMESS CCA components and other components run under this model is shown in Figure 5. When the GAMESS CCA Components run under the DDI/MPI model, the MPI program is used for starting up all the processes. The CCA framework runs on each process, which contains a driver component, a molecule factory component and a model factory component. While components in a single framework communicate with each other by *Provides/Uses* ports, the communication mechanism of similar components among different processes depends on the implementation of each component. The GAMESS CCA components in the compute processes and data server processes of the same node use local System V operations for accessing the data in the local shared memory of that node. The communication between all the molecule factory components uses MPI, and the same is true for the communication of driver components among the different processes. The communication of GAMESS CCA components in the different nodes is enabled by either MPI or TCP/IP.

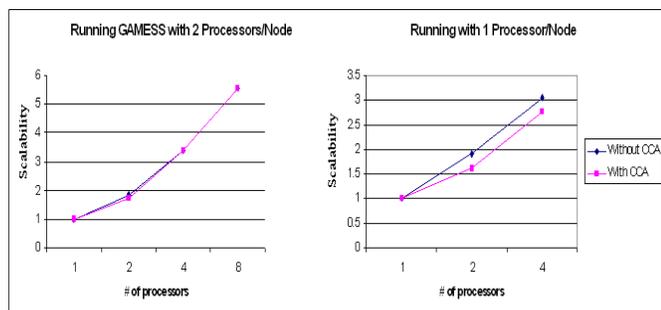


Fig. 6. The Hessien calculation of the molecule “Glycine” run on both the original GAMESS program and the GAMESS CCA component, which we labeled as “without CCA” and “with CCA”, respectively.

E. Performance Evaluation

To test the overhead of the CCA framework in GAMESS calculations, we compared the run time of the same set of GAMESS calculations with the original GAMESS program and by using the GAMESS CCA components. We performed all the tests on the following architecture: a cluster with 8 nodes, including four dual CPU Intel XEON™ nodes and four single-CPU nodes. All the nodes are running Debian Sarge w/Linux 2.6.6 kernels.

First, all the jobs are running in one processor. Table 1 shows the run time for four different GAMESS calculations. The results show that for three of the calculations using the GAMESS CCA components incurs less than 10 percent overhead.

Then, we run the Hessien calculation of the molecule “Glycine” in parallel for comparing the scalability of the original GAMESS program and the GAMESS CCA components. Figure 6 shows that the scalability of the GAMESS CCA components is about 10-15% less than the scalability of the original GAMESS program, which is still compatible.

TABLE I
WALLCLOCK TIME OF GAMESS CALCULATIONS WITH AND WITHOUT CCA

Molecule	Wallclock Time in Seconds (percentage of extra time)		
	GAMESS	GAMESS CCA Components	
		GAMESS/DDI	GAMESS/DDI/MPI
Glycine	61	61 (0%)	62 (1.6%)
Nicotine	1931	2308 (19.5%)	2300 (19.1%)
Firefly Luciferin	5905	6158 (4.3%)	5785 (-2%)
Ergosterol	29088	30592 (5.2%)	31856 (9.5%)

III. COUPLING GAMESS THROUGH CCA

With the GAMESS CCA interface constructed, the interoperation of GAMESS with other software packages can be done under the CCA framework. As an example, we chose to integrate GAMESS with a performance tool package, and to provide an interaction mechanism for GAMESS and NWChem.

A. Performance Sub-system for GAMESS

Within the scope of GAMESS, performance bottlenecks can occur in many places such as cache utilization, I/O or communication. Performance evaluation and monitoring tools for each of these potential bottlenecks may take years to develop, so starting from scratch is not a feasible solution. A useful approach is to use existing performance tools such as TAU (Tuning and Analysis Utilities) or PAPI [20], and incorporate them into GAMESS. These performance tools usually provide APIs for application developers to develop performance evaluation functions according to application needs.

Incorporating performance tools into GAMESS usually requires inserting performance function calls into the GAMESS source code, which is an intrusive approach. With GAMESS components, we prefer a performance tool that provides an interface compatible with the CCA standard, such that the access to performance tool APIs can be through component ports instead of direct calls to the API. In particular, the TAU performance system meets our requirements.

1) TAU Performance System

TAU is based on a general computation model [13], which is a superset of the one used by GAMESS. It provides technology for performance instrumentation, measurement, and analysis for complex parallel systems. Performance information can be captured at the node/context/thread level by using TAU. Besides performance instrumentation capability on both the component level [21] and the source code level, TAU also provides an interface to access the hardware counters through PAPI or PCL [21].

For CCA applications, TAU provides a performance component to measure the performance of CCA component software through the common *MeasurementPort* interface. Besides the performance component, TAU also provides *MasterMind* and *Optimizer* components for performance data collection for performance modeling of components and constructs optimal component assemblies, and Proxy Generators build proxies for both the *MeasurementPort* and the *Monitorport* in performance component [22]. To successfully install the TAU performance component and use all the provided functionality, both TAU and PDT (Program Database Toolkit) [23] must first be installed TAU performance components then can be set up.

2) Integration of GAMESS and TAU

For measuring the performance of the GAMESS CCA components, the *PerformanceMeasurement* component can be used. With TAU's CCA performance component installed and environmental variables set up properly, performance evaluation methods can be invoked in a component by connecting to the *PerformanceMeasurement* component through the *Provides/Uses MeasurementPort* under the CCA framework.

Performance evaluation on the component level is only a coarse grain evaluation, since the interactions of functions inside a component cannot be identified. For example, if we set the profiling interval of memory usage to one second in a Hessien evaluation, at the end of the computation we can plot the memory usage with time. This memory profiling only tells

us the memory usage of the whole Hessian computation; the memory usage of the energy and gradient upon which the Hessian is based, are within the plot of Hessian memory usage and cannot be isolated, unless we also develop components or methods inside the component for the energy and gradient. In other words, the detailed performance information available is determined by the granularity of GAMESS components. Through analysis and experiments of GAMESS, we concluded that source-code level instrumentation is unavoidable for developing a performance sub-system of GAMESS.

Even with the capability of the TAU performance tool, designing a useful performance sub-system for GAMESS requires careful analysis instead of simply inserting performance evaluation functions. For the present purpose we show the results of tracking memory usage for the same GAMESS computation, with CCA and without CCA. The computation is to calculate the Hessian of glycine. We measured the amount of memory used in each process. Without CCA, the maximum memory usage is 7.5 MBytes; with CCA, the maximum memory usage is 8.5 MBytes. This simple performance evaluation is to verify that usage of CCA will not hinder GAMESS computation, as GAMESS or the other chemistry packages usually requires a large amount of memory for computations. The performance sub-system of GAMESS is currently under development.

B. Integration of GAMESS and NWChem

1) NWChem and GAMESS

GAMESS and NWChem are two of the most popular chemistry software packages in the computational chemistry community. While there are overlapping functionalities in GAMESS and NWChem, such as calculations of Hessian, energy, and gradient, each has its strength in a certain area. For example, GAMESS has a rich set of properties while NWChem utilizes molecular symmetry better in some cases. Through CCA, it is possible to use a wave function calculated by NWChem as the starting wave function for a GAMESS computation.

The design philosophies behind GAMESS and NWChem are quite different. The approach GAMESS took is to be a self-sustained software package, without relying on any external packages. Thus the GAMESS development team designed DDI and uses it for GAMESS parallel computations. On the other hand, NWChem uses Global Arrays for communications, which in turn relies on ARMCI [24] and MPI.

2) The Integration Processes

NWChem relies on the Global Array (GA) toolkit as the underlying communication mechanism. GA and DDI are similar in the sense that they both provide an interface by which all processes in a parallel job can independently access and modify any data element in a distributed array, even when the array is physically distributed. They are also both compatible with the conventional MPI program. Therefore, we started with exploring whether we can instantiate the model factory implemented for both NWChem and GAMESS under a single CCA framework.

In principle, both model factories for NWChem and GAMESS should be able to coexist under the same CCA framework. However, certain limitations exist in the integration

of NWChem and GAMESS. The most significant one is the different requirements on the message-passing IDs/numbers on each host machine when integrating two model factories. In most cases, the DDI program requires an even number of processes on each host machine for dividing processes evenly as compute processes and data servers. For NWChem we observed that the ARMCI library requires consecutive message-passing IDs/numbers on the same host machine. Thus, both packages have their own restrictions on the configuration of MPI program. Another major restriction is the termination of model factories for both the NWChem and GAMESS under the same CCA framework. In the version of NWChem CCA interface we are currently using, the termination of the model factory of NWChem will also terminate the MPI program and even the CCA framework. The same is true for the current design of GAMESS CCA interface. This should not be a problem when only one model factory component is running under a single CCA framework. However, when multiple model factory components are instantiated and running under a single CCA framework, the finalization of one component will affect the correct termination of another.

IV. CONCLUSION

In this paper, we have presented our experience in designing interoperation mechanisms for GAMESS with scientific packages through the Common Component Architecture. To verify the design of the GAMESS CCA components, we integrated two packages, TAU and NWChem, with GAMESS through CCA. While the integration with TAU was straightforward, we encountered several difficulties in integrating with NWChem. The difficulties steamed mainly from the way the communication mechanisms of each component co-exist and interact with the CCA framework.

In general, when developing components for a large legacy code, we should consider not only its functionality and performance, but also its compatibility with other components. Their design of component parts, such as initialization and finalization, should not affect the global settings of a CCA framework, such as the configuration of MPI. In our experience, designing components for existing legacy packages is much more difficult than developing new components, as compatibility plays an important role in the integration process. The problem gets more complicated when two packages use the same message-passing library and each package has its own configuration to use the message-passing library. There are two approaches for designing such a component. A component can either use the traditional message-passing routines of the legacy codes, or adapt to the message-passing systems integrated in CCA or some other existing components [8]. The prior case is easier for a component provider. However, this case would allow each component to use different message-passing systems, such that the implementation of each component has to be very careful not to impose or create any restrictions for the other components under the same framework. For the latter case, it may require considerable re-coding in the legacy codes. For either case, much effort is required for programmers and unexpected problems can

emerge. A balanced approach should be developed for solving this issue, which may require a new standard or specification from the CCA groups in dealing with the compatibility issues when integrating different parallel computing packages.

While considerable efforts are still required to develop interfaces for numerous computations in GAMESS, many scientific codes may already benefit from the GAMESS CCA components. Since GAMESS is a very popular code, the success of GAMESS-CCA will encourage more software packages to adopt component paradigm gaining in flexibility computational capabilities.

ACKNOWLEDGMENT

We thank Manojkumar Krishnan and Theresa L. Windus from PNNL, and Joseph P. Kenny from Sandia National Laboratories for their helpful discussions on CCA and chemistry components, and Ryan M. Olson, Jonathan Bentz and Brett Bode from the Scalable Computing Laboratory of Ames Lab for the information on GAMESS and DDI.

REFERENCES

- [1] M. W. Schmidt, K. K. Baldrige, J. A. Boatz, S. T. Elbert, M. S. Cordon, J.H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. J. Su, T. L. Windus, M. Dupuis, J. A. Montgomery, "General Atomic and Molecular Electronic Structure System", *J. Comput. Chem.* 14, 1347-1363 (1993)
- [2] CCA-Forum. Common Component Architecture Forum. <http://www.cca-forum.org>
- [3] 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, L. F. Diachin, J. A. Kohl, M. Krishnan, G. Kumpfert, S. Lefantzi, M. J. Lewis, A. D. Malony, L. C. McInnes, J. Nieplocha, B. Norris, S. G. Parker, J. Ray, S. Shende, T. L. Windus, and Zhou. S., "A Component Architecture for High-Performance Scientific Computing," *Intl. J. High-Perf. Computing Appl.*, 2004.
- [4] Babel, <http://www.llnl.gov/CASC/components/babel.html>
- [5] Steve Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments," *IEEE Communications Magazine*, vol. 14, no. 2, February 1997
- [6] COM, <http://www.microsoft.com/com/default.mspx>
- [7] JavaBean, <http://java.sun.com/products/javabeans/>
- [8] D. E. Bernholdt, Wael R. Elwasif, and James A. Kohl, "Communication Infrastructure in High-Performance Component-Based Scientific Computing", *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages: 260 – 270 (2002)
- [9] MPI, Message Passing Interface, <http://www.mpi-forum.org>
- [10] PVM, Parallel Virtual Machine, <http://www.csm.ornl.gov/pvm/>
- [11] J. Nieplocha, RJ Harrison, and RJ Littlefield, Global Arrays: A nonuniform memory access programming model for high-performance computers, *The Journal of Supercomputing*, 10:197-220, 1996
- [12] Ryan M. Olson, Michael W. Schmidt, Mark S. Gordon, Alistair P. Rendell, "Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model", *SC'03*, November 15-21, 2003, Phoenix, Arizona, USA
- [13] S. Shende and A. D. Malony, "The TAU Parallel Performance System," (submitted to) *International Journal of High Performance Computing Applications*, ACTS Collection Special Issue, 2005.
- [14] Aprà, E.; Windus, T.L.; Straatsma, T.P.; Bylaska, E.J.; de Jong, W.; Hirata, S.; Valiev, M.; Hackler, M.; Pollack, L.; Kowalski, K.; Harrison, R.; Dupuis, M.; Smith, D.M.A.; Nieplocha, J.; Tipparaju V.; Krishnan, M.; Auer, A.A.; Brown, E.; Cisneros, G.; Fann, G.; Fruchtl, H.; Garza, J.; Hirao, K.; Kendall, R.; Nichols, J.; Tsemekhman, K.; Wolinski, K.; Anchell, J.; Bernholdt, D.; Borowski, P.; Clark, T.; Clerc, D.; Dachsel, H.; Deegan, M.; Dyal, K.; Elwood, D.; Glendening, E.; Gutowski, M.; Hess, A.; Jaffe, J.; Johnson, B.; Ju, J.; Kobayashi, R.; Kutteh, R.; Lin, Z.; Littlefield, R.; Long, X.; Meng, B.; Nakajima, T.; Niu, S.; Rosing, M.; Sandrone, G.; Stave, M.; Taylor, H.; Thomas, G.; van Lenthe, J.; Wong, A.; Zhang, Z.; "NWChem, A Computational Chemistry Package for Parallel Computers, Version 4.7" (2005), Pacific Northwest National Laboratory, Richland, Washington 99352-0999, USA.
- [15] GAMESS, The General Atomic and Molecular Electronic Structure System (GAMESS) Homepage, <http://www.msg.ameslab.gov/GAMESS/GAMESS.html>.
- [16] TCGMSG, <http://www.emsl.pnl.gov/docs/parsoft/tcgmsg/tcgmsg.html>
- [17] The CCA Chemistry Component Toolkit, <http://www.cca-forum.org/~cca-chem/>
- [18] 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", *Journal of Computational Chemistry*, 24(14) 1717-1725 (2004).
- [19] MPQC, The Massively Parallel Quantum Chemistry Program, <http://www.mpqc.org>
- [20] Browne, S., Deane, C., Ho, G., Mucci, P. "PAPI: A Portable Interface to Hardware Performance Counters," *Proceedings of Department of Defense HPCMP Users Group Conference*, June 1999.
- [21] R. Berrendorf and B. Mohr. "PCL -- The Performance Counter Library: A Common Interface to Access Hardware Performance Counters on Microprocessors". Technical report, Research Centre Juelich GmbH, Juelich, Germany, September 2000.
- [22] TAU's CCA Tools, <http://www.cs.uoregon.edu/research/tau/cca>
- [23] PDT, Program Database Toolkit, <http://www.cs.uoregon.edu/research/pdt/pubs.php>
- [24] Jarek Nieplocha and Bryan Carpenter. "ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-time Systems." *Proc. 3rd Workshop on Runtime Systems for Parallel Programming (RTSPP) of International Parallel Processing Symposium IPPS/SDP'99*, San Juan, Puerto Rico, April 1999, in (1) J. Rolim eat al. (eds.) *Parallel and Distributed Processing*, Springer Verlag LNCS 1586, and (2) *IPPS/SDP'99 CDROM*, 1999.
- [25] HPC++ Working Group, "HPC++ White Papers," Technical Report TR 95633, Center for Research on Parallel Computation, 1995.
- [26] M. Krishnan, Y. Alexeev, T. L. Windus and J. Nieplocha, "Multilevel Parallelism in Computational Chemistry using Common Component Architecture", *Supercomputing 2005*, November 12-18, 2005, Seattle, Washington, USA
- [27] S. Shende, A. D. Malony, C. Rasmussen, M. Sottile, "A Performance Interface for Component-Based Applications," *Proc. International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems*, IPDPS'03, IEEE Computer Society, 278, 2003