Using the GA and TAO Toolkits for Solving Large-Scale Optimization Problems on Parallel Computers

S. BENSON (benson@mcs.anl.gov)
Mathematics and Computer Science Division, Argonne National Laboratory
M. KRISHNAN (manoj@pnl.gov)
Computational Sciences and Mathematics, Pacific Northwest National Laboratory
L. MCINNES (mcinnes@mcs.anl.gov)
Mathematics and Computer Science Division, Argonne National Laboratory
J. NIEPLOCHA (Jarek.Nieplocha@pnl.gov)
Computational Sciences and Mathematics, Pacific Northwest National Laboratory
J. SARICH (sarich@mcs.anl.gov)
Northwestern University

7/24/2006

Challenges in the scalable solution of large-scale optimization problems include the development of innovative algorithms and efficient tools for parallel data manipulation. This paper discusses two complementary toolkits from the collection of Advanced CompuTational Software (ACTS), namely, Global Arrays (GA) for parallel data management and the Toolkit for Advanced Optimization (TAO), which have been integrated to support large-scale scientific applications of unconstrained and bound constrained minimization problems. Most likely to benefit are minimization problems arising in classical molecular dynamics, free energy simulations, and other applications where the coupling among variables requires dense data structures. TAO uses abstractions for vectors and matrices so that its optimization algorithms can easily interface to distributed data management and linear algebra capabilities implemented in the GA library. The GA/TAO interfaces are available both in the traditional library mode and as components compliant with the Common Component Architecture (CCA). We highlight the design of each toolkit, describe the interfaces between them, and demonstrate their use.

Categories and Subject Descriptors:
General Terms: Numerical Optimization, Distributed Data Structures, One-Sided Communication, Load Balancing, Molecular Dynamics, Lennard-Jones Potential.

---

1. INTRODUCTION

Modern scientific applications are becoming increasingly complex in terms of their mathematical models, computational algorithms, and parallel computing techniques; these projects often require the contributions of software developed by different groups. Some of the challenges in large-scale, interdisciplinary problems can be addressed by the adoption of reusable software toolkits and libraries, which can help application scientists incorporate the expertise of domain specialists. For example, a recent project in molecular geometry optimization [Kenny et al. 2004] used MPQC [Janssen 2003] and NWChem [Bernholdt et al. 1995] to evaluate energy functions, Global Arrays (GA) [Nieplocha et al. 1994; 1996] to manage distributed arrays of data, the Toolkit for Advanced Optimization (TAO) [Benson et al. 2005, Benson et al. 2001] for the optimization algorithms, and MPI [MPI-Forum 1994] for interprocess communication. Managing the interactions among multiple software packages, however, often imposes other types of challenges to application scientists. Technical issues such as overlapping

namespaces and the use of different programming languages can hinder the adoption of independently developed software libraries. In addition, diverse software design philosophies can complicate the development of interfaces among various packages and may require potentially expensive restructuring and redistribution of data. A desire to achieve efficient and scalable performance that is portable across different high-end architectures further magnifies these difficulties.

The Advanced CompuTational Software (ACTS) initiative was established to overcome some of the major impediments in developing of scientific applications. One of this initiative's goals was to make software toolkits used within the laboratories of the U.S. Department of Energy (DOE) more applicable and interoperable [DOE-ACTS 1998, Drummond and Marques 2003]. Attaining this goal involves creating consistent interfaces, generalizing data management strategies for complex programming needs, resolving runtime issues such as thread safety, addressing language interoperability, and providing long term support for evolving research software. As a result of the effort funded under this initiative, the two ACTS toolkits described in this paper, Global Arrays and TAO, provide state-of-the art capabilities in the areas of distributed data management and large-scale optimization, respectively. In this paper, we show how GA and TAO have been designed and integrated to assist computational scientists working in projects that involve the advanced management of dense distributed arrays, dynamic load balancing, and large-scale optimization. These projects typically involve many variables, each of which is coupled to the others in a nonlinear fashion. The combined use of GA and TAO as a result of this integration effort has proven valuable for large-scale quantum chemistry simulations, where benchmarking of molecular geometry problems that use NWChem and MPQC, along with GA for parallel data management and TAO for numerical optimization, has demonstrated reduced run times up to 43% compared to the stand-alone chemistry packages [Kenny et al. 2004]. Such capabilities have potentially broad applicability, as the discovery of molecular geometries corresponding to stable and reactive chemical species is a fundamental step in computational processes, including combustion modeling, catalyst design, and the simulation of biological processes.

A second example of problems targeted by the combined use of GA and TAO is a classic application kernel known as the Lennard-Jones problem [Averick et al. 1991]. The determination of the minimum energy configuration of a cluster of particles or molecules is known as the molecular conformation problem. This problem is central to the study of cluster statistics. For additional background on this problem, see, for example, [Hoare 1979]. More formally, given the positions $p_1, p_2, ..., p_n$ of n particles (atoms or molecules) in $\Re^d$, the function V on the set of particles p is defined as

$$V(p) = \sum_{j=2}^{n} \sum_{i=1}^{j-1} v(\| p_j - p_i \|_2),$$

where $v:\Re \rightarrow \Re$ is the potential function on the distance between a pair of particles. The Lennard-Jones potential function is defined by

$$v(r) = r^{-12} - 2r^{-6},$$

where r is the distance between the particles. The molecular conformation problem is to determine the positions of the n particles that minimize the energy potential function, V, which is invariant with respect to permutations, translations, and rotations of the n molecules. Translating the cluster so that the center of gravity is at the origin can eliminate invariance with respect to translations. While relatively simple with a concise problem statement, this example serves as an excellent test for optimization software [Averick et al. 1991] because it can be made arbitrarily large, and its key features are

similar to our target applications (see, e.g., [Kenny et al. 2004]). Small instances of this problem involving fewer than a thousand particles have been solved many times, but larger instances mandate efficient capabilities to manage dense, distributed data and effective optimization solvers to compute molecular configurations with as few energy evaluations as possible.

Various software packages are available for solving optimization problems [Moré and Wright 1993]. Most of these packages are restricted to single processors, but excellent efforts in parallel optimization software include OPT++ [Meza 1994], APPSPACK [Hough et al. 2001, Kolda 2005], and Dakota [Dakota 2005]. The latter two packages, in particular, achieve parallelism through simultaneous evaluations of the objective function. This approach is very efficient when an optimization solver needs to evaluate the objective function at multiple points before determining the next step. In contrast, TAO includes a set of numerical optimization solvers whose parallelism is achieved in the function evaluations and the linear algebra operations. This approach is appropriate when a large number of variables that can be distributed over multiple processors. To compute a minimum, TAO solvers require users to provide first derivatives in the form of gradient vectors (and sometimes second derivatives in the form of Hessian matrices). These derivative evaluations should also work in parallel.

Because in the Lennard-Jones problem each particle p_j exerts a force on each of the other particles to create a dense coupling among variables, distributed data structures for dense arrays are important for the efficient parallel computation of the derivatives of the energy function. The Global Arrays toolkit [Nieplocha et al. 1994; 1996] presents to the application developer a distributed data structure as a single object and allows operations on distributed-memory architectures as though the data resided in shared memory. In addition, the availability of one-sided (put/get) communication and atomic operations is valuable for implementing effective dynamic load balancing, as shown in Section 6 in the context of the Lennard-Jones problem. These features help application scientists to work at higher levels of abstraction and increase code reuse. This strategy reduces the amount of new code that must be written and enables scientists to program in terms of physically meaningful concepts rather than manipulating low-level distributed data structures and handling explicit interprocessor communication. Thus, this approach can help to make scientists more productive and permit more time to be spent improving performance-critical algorithms and application kernels [Bernholdt et al. 2004].

Both TAO and GA offer functionality that can reduce the complexity of solving minimization problems, and the design of each toolkit determines how easily they can used together. With applications such as the Lennard-Jones cluster in mind, GA and TAO have been developed to enable interoperability with other packages through the Common Component Architecture (CCA) as well as using more traditional interfaces.

The remainder of this paper is organized as follows. Section 2 describes the parallel data management capabilities of the GA toolkit. Section 3 outlines the optimization solvers in TAO and highlights the abstractions used for vector and matrix operations, which enable TAO to use the data structures provided by GA. Section 4 describes the integration of these two toolkits as conventional libraries, while Section 5 discusses their use as components that support the CCA. Section 6 presents numerical results using GA and TAO to solve large instances of the Lennard-Jones problem. Section 7 summarizes our experiences in interfacing between these two toolkits.

2. GLOBAL ARRAYS

Global Arrays can be used as a toolkit for managing dense distributed arrays in applications based on message passing (e.g., MPI) or as a complete parallel programming environment based on a shared-memory paradigm. Although a class of operations for sparse data management was recently added [Nieplocha et al. 2001], the core capabilities of GA focus on managing dense distributed arrays. The original version of the GA package offered explicit support for two-dimensional arrays that was based on one-sided communication in the Aggregate Remote Memory Copy Interface [Nieplocha and Carpenter 1999, Nieplocha and Ju 2000]. One-sided communication allows a process to access remote data owned by another process without explicit cooperation by that process. (Depending on the platform, one-sided access can be implemented using shared memory, threads, network hardware, Remote Direct Memory Access (RDMA), or other vendor-specific mechanisms.) The current version of GA supports *n*-dimensional array capabilities involving integer, floating point, double precision, and character data types. The maximum number of dimensions is a compile time option; the default is seven, the maximum number of dimensions supported by Fortran.

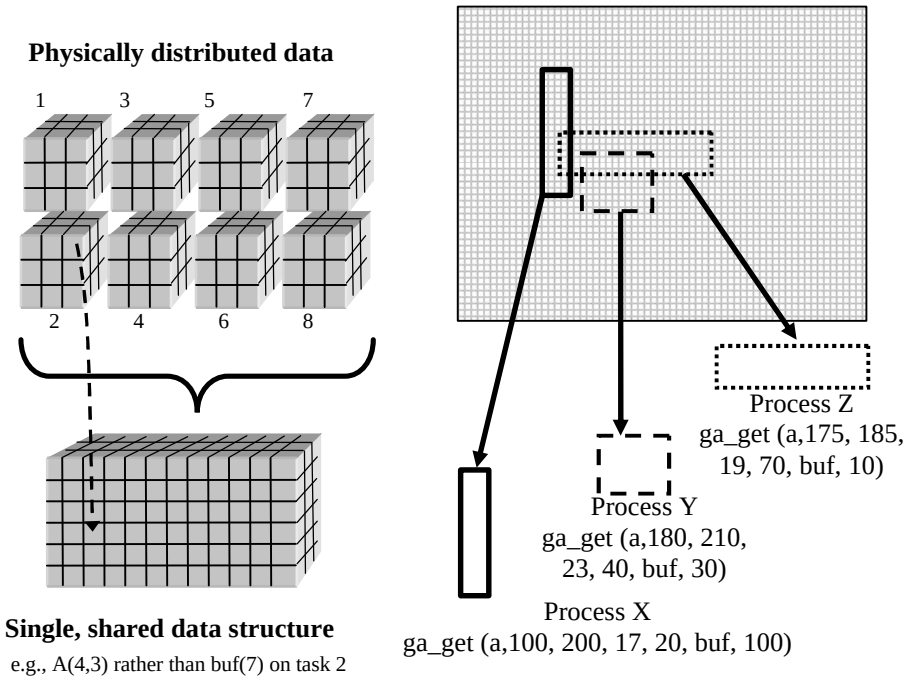

Figure 1: *Left*: GA manages a distributed array as a single shared data object. As shown, any process can access the distributed data using global indexing (e.g., using global index A(4,3)). *Right*: Any part of the array can be accessed noncollectively as if it is located in shared memory (e.g., Process X gets a block of the global array with global indices starting at (100,17) and blocksize=100x3).

Figure 1 illustrates the concept of shared-memory access to distributed arrays, that is, global arrays. The Global Arrays toolkit strives to combine the better features of the shared and distributed-memory programming models [Nieplocha et al. 2002, Nieplocha et al. 1994] by implementing a shared-memory programming approach in which the user has the ability to manage data locality. Explicit calls to GA functions transfer data between a global address space (a distributed array) and local storage. In this respect, the GA model is similar to distributed shared-memory (DSM) models that provide an explicit acquire/release protocol (see, e.g., [Zhou et al. 1996]). However, unlike DSM systems [Cox et al. 1997] that rely on virtual memory to facilitate access to remote shared data, GA is implemented as a library with specific user interfaces for accessing shared data

based on multidimensional array APIs. In addition, the GA model acknowledges that remote data is slower to access than is local data and allows applications optionally to specify and manage data locality when allocating GA arrays or accessing data. These capabilities are supported through explicit library calls. Although they can be ignored by the programmer in most parts of the application code, they can be quite useful for optimizing performance and scalability in communication intensive parts of the applications [Zhang et al. 2005]. By using communication operations in ARMCI that were optimized to maximize transfer rates of sections of multidimensional arrays, and moving only the data requested by the user rather than full memory pages or cache lines like in DSM, GA also avoids issues such as false sharing or redundant data transfers present in some DSM solutions. See, for example, [Bershad et al. 1993, Cox et al. 1997, Freeh and Andrews 1996]. GA provides a relaxed consistency for the shared data equivalent to the location consistency model [Gao and Sarkar 2000]. The toolkit can be used in C, C++, Fortran 77, Fortran 90, and Python programs.

The GA toolkit provides extensive support for controlling array distribution and accessing locality information, which is important for some applications [Zhang et al. 2005]. Applications can create global arrays by allowing the toolkit to determine the array distribution, specifying a decomposition only for one array dimension and allowing the toolkit to determine the others, specifying the distribution block size for all dimensions, or specifying an irregular distribution as a Cartesian product of irregular distributions for each axis. The distribution and locality information available through toolkit operations specify the array section held by a process, indicate which process owns a particular array element, and return a list of processes and the blocks of data owned by each process corresponding to a given section of an array.

The GA toolkit supports both task and data parallelism. *Task parallelism* uses one-sided copy operations that transfer data between global memory (distributed/shared array) and local memory. In addition, each process can access data held in a section of a global array that is logically assigned to that process. The one-sided communications used by Global Arrays eliminate the need for the programmer to manage responses by remote processors. Only the process issuing the data request is involved, thereby considerably reducing algorithmic complexity compared to the programming effort required to move data around in a two-sided communication model. Copying data from a local buffer to a distributed array requires a single call. Based on the data distribution, the GA toolkit decomposes the call into point-to-point data transfers to each of the different processors.

The *data parallel* computing model, as employed in Section 4 of the *TAO-GA-Application* Object, relies on a set of collective functions that operate on either entire arrays or sections of global arrays. This set of functions includes BLAS-like operations (for example, copy, addition, transpose, dot product, and matrix-vector multiplication). These are collective data-parallel operations that are called by all processes in the parallel job. For example, data between different arrays can be moved by using a single function call. Provided two patches of data have the same number of elements, that function call can move a patch of data, identified by a set of lower and upper indices in the global index space, from one global array to another global array. The array distributions do not have to be identical, and the implementation can perform the necessary data reorganization (i.e., “MxN” problem [CCA-Forum 2005]).

Some data parallel operations, such as addition, are defined and supported for all array dimensionality. Other operations, such as matrix-vector multiplication, are limited to one- and two-dimensional arrays (matrix-vector multiplication is also offered on two-dimensional subsections of higher-dimensional arrays). GA extends its capabilities in the area of linear algebra by offering interfaces to third-party libraries such as ScaLAPACK [Blackford et al. 1997] for linear equation solvers and PeIGS [PeIGS 2005] for standard

and generalized real symmetric eigensolvers. Additional atomic operations,[1] executed serially, allow applications to synchronize processes and verify computations. As discussed in the context of the Lennard-Jones example in Section 6, the atomic fetch-and-increment operation proved valuable for implementing dynamic load-balancing schemes and updating a shared counter (stored in a global array) representing the number of outstanding tasks. This capability, in combination with the shared-memory access supported by GA, simplifies the development of applications requiring dynamic load balancing and exhibiting irregular communication patterns. Other implementations relying on the traditional message-passing model require explicit coordination between the sender and receiver of data.

3. THE TOOLKIT FOR ADVANCED OPTIMIZATION

The Toolkit for Advanced Optimization [Benson et al. 2005, Benson et al. 2001] focuses on the solution of large-scale optimization problems of the form

$$\text{minimize } \{F(x)\text{: subject to } l \le x \le u\},$$

where the bounds l, $u \in R^n$ and $F{:}\Re^n \rightarrow \Re$ is the objective function of the variables $x \in \Re^n$. The vectors l and u define lower and upper bounds on the variables. When lower or upper bounds on the problem do not exist, the bounds may be set to negative or positive infinity, respectively. Problems of particular interest can be characterized by a large number of variables and an objective function that is differentiable, expensive to evaluate, and decomposable over multiple processors.

Optimization problems of this form pervade many areas of science and engineering. The finite-element formulation of many elliptic partial differential equations, for instance, can be expressed as a minimization problem. The obstacle problem, which minimizes the area of a surface over a closed domain subject to boundary conditions and obstacle constraints [Rodrigues 1987], is a common example of a bound-constrained optimization problem. Considered in more detail in this paper is another example that arises in molecular dynamics, where a stable molecular configuration minimizes the potential energy function associated with it. As introduced in Section 1, the Lennard-Jones potential function $v(\cdot)$ is a nonconvex function that is bounded below. Since $v(r)$ is not defined at r=0, the distance between two particles must always be positive. Indeed, $v(r)$ approaches $+\infty$ as r approaches 0, while $v(r)$ converges to zero as r approaches $+\infty$. The global minimum of v occurs at r=1. Because each particle p_j exerts a force on each of the other particles, the second partial derivative $d^2V(p)/\, dp_i\, dp_j$ is nonzero for all i and j. The solution of such problems demands robust and flexible solution strategies, including efficient algorithms and implementations that allow scientists to exploit problem-specific structure to improve solver performance.

The collection of solvers in TAO includes Newton-based methods, which use first- and second-derivative information from the objective function and are well known for their fast local convergence properties. Global convergence of these methods is enforced by using one of two techniques. First, a line search can ensure that each iteration of the algorithm provides a better solution than the previous iteration. The GPCG solver [Moré and Toraldo 1991], in particular, uses a projected line search that can add or remove multiple variables from the set of variables fixed to the lower or upper bound. This technique makes GPCG especially suitable for large-scale problems. Second, a trust

[1] Atomic operations in GA are guaranteed to be executed serially, in an uninterrupted manner with respect to atomic operations targeting the same memory and issued by the same or other processors. The library does not guarantee that atomic operations called by different processors will be completed in the exactly the same order as they were issued.

region can be designed to represent regions near the current iteration where the quadratic model is sufficiently accurate. The TRON solver [Lin and Moré 1999] implemented in TAO, for example, uses a trust region in conjunction with approximate solutions to the linearized Newton systems provided by iterative solvers.

Other solvers in TAO do not require second-derivative information. Nonlinear conjugate gradient solvers use the first-order derivatives of the objective function to compute a step direction. The use of these methods originated in solving linear systems involving a symmetric, positive definite matrix. Several variants, including Fletcher-Reeves, Polak-Ribiérre, and Hestenes-Stiefel [Nocedal and Wright 1999], extend linear conjugate gradient techniques to unconstrained minimization and are available in TAO. Limited-memory variable-metric (also known as quasi-Newton) solvers use first-order derivatives at several points to approximate second-order information. Each iteration of these solvers uses relatively little time and memory compared with full Newton methods. These solvers have proven successful for unconstrained problems and have been adapted to bound-constrained optimization [Benson and Moré 2001].

Although multiple implementations of these various optimization techniques exist, TAO has carefully addressed issues of portability, versatility, and scalability within parallel environments. Even on single-processor architectures, TAO does not make rigid assumptions about the way mathematical objects such as vectors and matrices are represented by the computer. Therefore, users can employ a natural representation of data for a particular application instead of one imposed by the optimization library. These benefits are magnified by the nature of multiprocessor architectures, as robust and efficient implementations of mathematical functions involves the added considerations of parallel data structures and communication among processors.

The optimization methods in TAO employ abstractions for vector and matrix objects. These abstractions include about fifty well-defined operations, such as duplication and destruction of these objects, vector sums, inner products, norms, and matrix-vector products. The subroutines that implement these operations are the only routines whose implementation is particular to a specific representation of the data. Because these subroutines perform most of the floating-point operations and data manipulation associated with the optimizations algorithm, efficient implementations are of paramount importance. These subroutines constitute the interface between TAO and other packages that support vector and matrix operations. The design of TAO allows users to implement these interfaces, but the amount of effort required to write these subroutines from scratch may discourage all but the most ambitious of users. Therefore, it is important for the developers of TAO to write and support interfaces to other popular toolkits.

The use of object-oriented techniques in TAO allows scientists and engineers to employ the toolkit in the context of data structures and data management packages appropriate for their applications. This design decision is strongly motivated by the challenges inherent in the use of large-scale distributed memory architectures and the reality of working with large scientific codes. For example, the initial versions of TAO supported the parallel sparse vector and matrix objects within PETSc [Balay et al. 2005, Balay et al. 1997]. This interface is appropriate for discretizations of many continuous optimization problems. The current effort has extended the availability of TAO to optimization problems using GA and components compliant with the Common Component Architecture. More explanation of the CCA is given in Section 5.

4. INTEGRATING AN APPLICATION WITH GLOBAL ARRAYS AND TAO

While TAO and GA were designed so that users can employ the packages together, the joint work among both project teams has facilitated the simultaneous use of the two

packages by implementing an interface that combines their complementary capabilities. Through the component model discussed in the next section, interfaces between GA and TAO are available in several different programming languages and styles. For users more comfortable with traditional software libraries, we also provide a procedural interface in the C programming language. The choice of a procedural over an object-oriented C++ interface was made to be consistent with the original GA and TAO interfaces, which are familiar and preferable to the packages' current users, who are a primary target for employing these new capabilities. This section provides an overview of this interface between GA and TAO, which is described in full detail in the documentation that accompanies this freely available software[Benson et al. 2005].

As illustrated by Figure 2, the interface among GA, TAO, and the application-specific code is contained in a *TAO-GA-Application object*, which is an input argument to the user-defined routines for evaluating the objective function and its derivatives. This object wraps the GA implementations of vectors and matrices into objects whose methods are defined in the abstract classes in TAO, thereby encapsulating the interface between the two packages without exposing users to details. There are about fifty of these vector and matrix methods, including sums, products, norms, and several operations specific to optimization solvers in TAO, such as elementwise addition of two arrays and shifting the diagonal of a two-dimensional array.

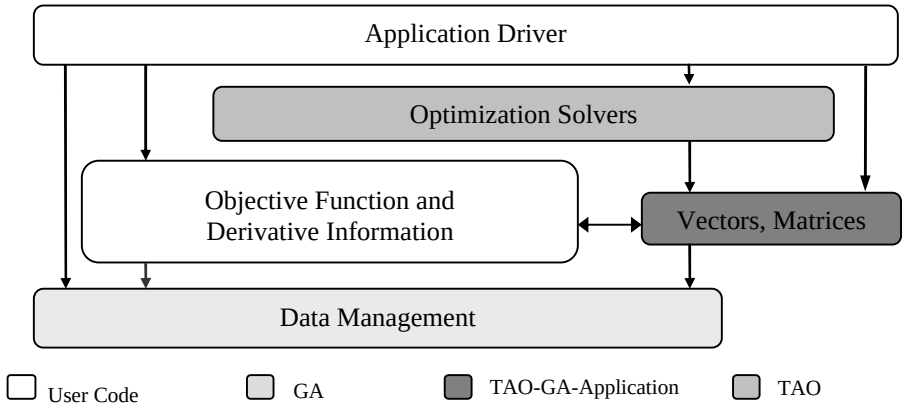

Figure 2: Applications involving TAO and GA require the coupling of an objective function and its derivatives, an optimization solver, and linear algebra capabilities.

The routine that evaluates the objective function to be minimized and its gradient must be written by the application developer. The name of the routine is arbitrary, but a set of arguments must match the following declaration

```
FormFunctionGradient(TAO_GA_APPLICATION taogaapp, GAVec X,
        double *f, GAVec G, void *ptr);
```

The first argument is the TAO-GA-Application object, which encapsulates details about TAO and GA. The second argument is a GA vector with the current approximate solution. The newly computed objective value and gradient vector should be returned by using the third and fourth arguments, respectively. The coupling of objective value and gradient evaluations improves performance in some problems. The final argument is a pointer to a user-defined structure or object that this user-defined routine may use to perform the function and gradient computations. Neither TAO nor GA will access this data, so the application developer is free to define this structure in a manner appropriate

for the application. Although it can point to a specific type of object, it will be passed through the solver as a `(void*)` type in order to maintain generality in the software.

The code fragment in Figure 3 shows how we pass the optimization problem to the solver. In particular, line 6 calls a subroutine that creates a TAO-GA-Application object. The subroutine in line 8 creates a GA vector that we pass to the application object in line 9. In lines 11--12, we pass a function pointer to the subroutine `FormFunctionGradient` to the application object. We also cast the address of a structure containing application-specific information, `AppCtx,` to a `(void*)` type and pass it to the solver in line 12 with the function pointer. The address of this structure will be passed back to the application as the fifth argument of `FormFunctionGradient`. We pass lower and upper bounds on the variables, in the form of GA Vector objects, to the application object in another routine. Line 14 calls a routine that creates a TAO solver that implements the limited-memory variable-metric ("tao_lmvm") method. Line 15 passes the TAO-GA-Application object to the TAO solver, which computes a minimizer of the objective function. Additional routines may be necessary to generate the data, set various options, and save the solution, but this snippet of code indicates how the application can interact with the software libraries.

```
1   GAVec X;                      /* solution vector */
2   TAO_SOLVER tao;               /* TAO optimization solver */
3   TAO_GA_APPLICATION taogaapp;  /* TAO application using GA */
4   AppCtx user;                  /* user-defined data structure */
5
6   TaoGAApplicationCreate (MPI_COMM_WORLD, &taogaapp);
7
8   X = NGA_Create (C_DBL,1,&n,"GA_X", NULL);
9   TaoGAAppSetInitialSolutionVec(taogaapp, X);
10
11  TaoGAAppSetObjectiveAndGradientRoutine (taogaapp,
12                        FormFunctionGradient, (void *)&user);
13
14  TaoCreate(MPI_COMM_WORLD, "tao_lmvm", &tao);
15  TaoSolveGAApplication(taogaapp,tao);
```

Figure 3: Code fragment in C illustrating the use of TAO and GA.

When Newton methods are employed, the application must also provide a routine that evaluates the Hessian matrix. GA has implemented a dense matrix class and supports the necessary operations on these structures, such as matrix-vector products and linear solves. Like the routine that evaluates the objective and gradient, the arguments for this routine are the TAO-GA-Application object, a GA vector containing the variables, a GA matrix for the Hessian, and a pointer to application-specific data.

As Figure 3 shows, the interface between the application and optimization toolkit is minimal. It consists of creating an object, giving it an objective function and initial variable vector, and passing the object to a TAO solver to compute a minimum. Most details underlying parallel data structures are embedded within the objects, insulating the user from many of the complications of distributed-memory computing. The most

computationally intensive aspect of solving the application is evaluating the objective and gradient, and the details of these operations presumably lie within the application developer's realm of expertise.

5. COMMON COMPONENT ARCHITECTURE

An alternative to linking to external libraries through a procedural interface, as described for GA and TAO in Section 4, is the use of the Common Component Architecture (CCA) [Armstrong et al. 1999, CCA-Forum 2005]. The CCA is a component model specifically designed to address the needs of high-performance scientific computing by the CCA Forum, which originated as a grass-roots undertaking by some of the participants of the DOE-2000 ACTS initiative [DOE-ACTS 1998]. Currently a subset of CCA Forum members continue development of component-based scientific software and tools[DOE-SciDAC 2005]. From an application scientist's perspective, components allow software developers to describe the calling interfaces of libraries and applications in a manner that hides low-level details, such as implementation language, compiler, parallelism, or location on a network. Components encapsulate the knowledge, experience, and work of other scientists, and they provide building blocks that speed application development. To provide these benefits to applications, we have developed CCA component interfaces for both the GA and TAO toolkits. We expect that the advantages offered by the CCA interfaces will be of particular interest to computational science research projects that require the coordinated use of a variety of independently developed software capabilities, such as multimodel chemistry and physics implementations, tools for numerical computing, parallel data management and input/output, and visualization, which naturally may evolve over the lifetime of the projects. In other words, the component approach to GA and TAO facilitates their interoperability both with each other and with a variety of additional external software packages.

While the details of the CCA specification are beyond the scope of this paper, we highlight the key points that are most pertinent for the combined parallel use of GA and TAO components in the single-program multiple-data (SPMD) mode; see [Bernholdt et al. 2005, CCA-Spec 2005, McInnes et al. 2005] for further details. The CCA approach consists of three main elements: *components*, *ports*, and *frameworks*. Briefly, *components* are basic units of software functionality that can be composed together at runtime to form applications; *ports* are the abstract interfaces through which components interact, and *frameworks* manage components as they are assembled into applications and executed. One of the fundamental assumptions in the CCA is that components may be written in different programming languages. In order to facilitate language interoperability, the Scientific Interface Definition Language (SIDL) has been adopted to describe component interfaces [Cleary et al. 1999, Kohn et al. 2001].

Components may provide ports, meaning that they implement the functionality expressed in a port (called *provides ports*), or they may use ports, meaning that they make calls on a port provided by another component (called *uses ports*). This *uses/provides* paradigm allows components in the same process address space to be invoked directly, without intervention by the framework, and with data passed by reference if desired (also referred to as "direct connect" or "in-process" components). Figure 4 shows an example of components based on GA and TAO interfacing through ports.

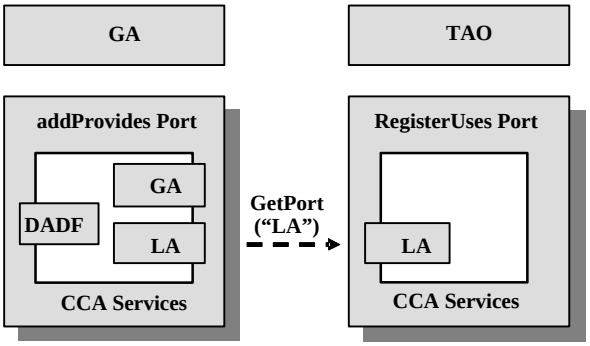

Figure 4: Example of two components interfacing through ports. The port LA is provided by the GA component and used by the TAO component.

For an SPMD application, such as the Lennard-Jones model introduced in Section 1, each parallel process would be loaded with the same set of components, with their ports connected in the same way. Interactions within a given parallel process occur through normal CCA port mechanisms. These would generally use the "direct connect'' approach to minimize the CCA-related overhead. Interactions within the parallel cohort of a given component are free to use the parallel programming model they prefer, and in this case we employ MPI.

We developed the Global Array and TAO components by adding thin wrappers around the libraries described in Sections 2 and 3 to manage port registration and framework interactions. While the details of this process are beyond the scope of this paper, we refer interested readers to [CCA-Forum 2005, McInnes et al. 2005, Sarich 2004] for examples and code. The TAO component provides the port "OptimizationSolver," and the Global Array component provides the ports "GlobalArrayPort," "DADFPort," and "LAPort." "GlobalArrayPort" has interfaces for creating and accessing distributed arrays. "DADFPort" offers interfaces for defining and querying array distribution templates and distributed array descriptors following the API proposed by the CCA Forum Distributed Data Working Group. "LAPort" provides TAO the core linear algebra support for manipulating vectors, matrices, and linear solvers.

Figure 5 illustrates the composition of the Lennard-Jones energy optimization using the graphical user interface of Ccaffeine, a CCA-compliant framework that supports the SPMD paradigm [Allan et al. 2002]. The *TaoSolver* component in this snapshot has been configured to use a limited-memory variable-metric method, which requires an application-specific component (implementing the Lennard-Jones model) to evaluate the model's function and gradient, as well as a linear algebra component (based on GA) to manipulate vectors, matrices, and linear solvers. The model uses a distributed array component (based on GA) for managing distributed data structures. The boxes in the wiring diagram show components for the distributed array, optimization solver, linear algebra, visualization, and application-specific driver and model. The connector lines in this diagram represent connections between *uses* and *provides* ports. For example, the *TaoSolver*'s optimization component's "OptimizationModel" *uses* port is connected to the *GA_LJMDModel*'s "OptimizationModel" *provides* port; hence, the optimization solver component can invoke the interface methods for function and gradient computation that the *GA_LJMDModel* component has implemented. The special "GoPort" (named *Optimize* in this application) is used to start the execution of the application.

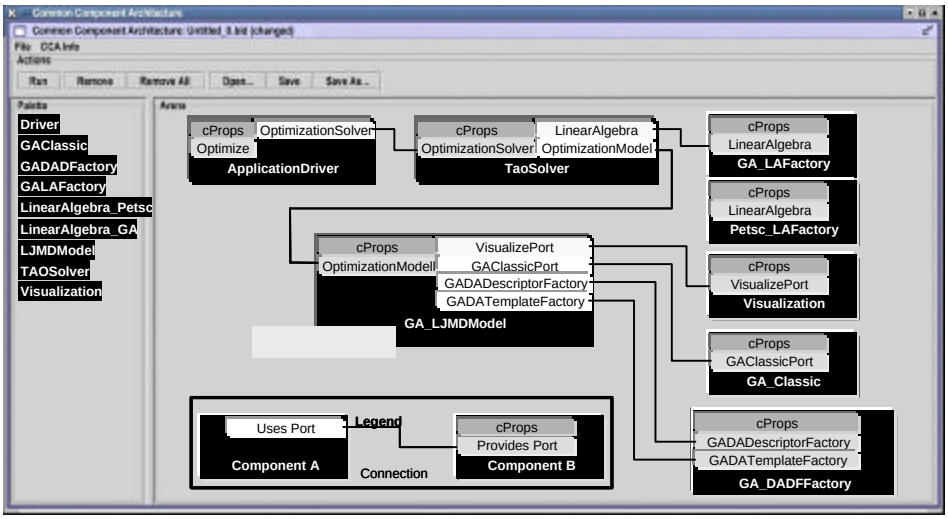

Figure 5: CCA component wiring diagram for Lennard-Jones energy optimization.

6. COMPUTATIONAL RESULTS

The combined use of GA and TAO resulting from this integration effort has already proven valuable for large-scale quantum chemistry simulations in molecular geometry optimization [Kenny et al. 2004]. These simulations, which involved MPQC and NWChem to evaluate energy functions, reduced run times up to 43% compared to the solvers implemented in stand-alone chemistry packages.

In this paper, we demonstrate computational results of the software for Lennard-Jones problems with the limited-memory variable-metric and nonlinear conjugate gradient methods. To measure component overhead and demonstrate efficient runtime performance, we consider both the traditional procedural interface to GA and TAO introduced in Section 4 and the CCA component approach highlighted in Section 5. We employ up to 170 processors for several problem sizes to demonstrate the combined capabilities and scalability of GA and TAO for large-scale optimization problems.

We store the particle coordinates in a global array distributed over the processors. Each processor computes a fixed set of interatomic forces [Plimpton and Heffelfinger 1992]. Symmetry of forces between two particles halves the amount of computation, and a centralized task list maintains another global array that stores the information about the order in which forces will be computed.

Load imbalance is a serious concern for force decomposition molecular dynamics algorithms [Plimpton 1995]. To address the potential load imbalance in our test problem, we use a simple and effective dynamic load-balancing technique called fixed-size chunking [Kruskal and Weiss 1985]. Initially, all the processes get a block from the task list. When a process finishes computing its block, it gets the next available block from the task list. Computation and communication overlap by issuing a nonblocking “get” call to the next available block in the task list, while computing a block [Tipparaju et al. 2003]. This implementation of the dynamic load-balancing technique takes advantage of the atomic and one-sided operations in the GA toolkit (see Figure 6). The GA one-sided operations eliminate explicit synchronization between a processor that executes a task and a processor that has the relevant data; see Figure 1. Atomic operations reduce the communication overhead in the traditional message-passing implementations of dynamic

```
Get task (i.e., block info) to be computed (atomic fetch-and-add)
Issue nonblocking get call for the first block
do (until last block/task)
   determine what the next block/task is
   issue nonblocking get call for the next block
   wait for previously issued get call
   compute Function-Gradient
            (overlapping communication with computation.
            i.e., receiving next block while computing previous
            block)
   accumulate function and gradient into respective Arrays
done
```

Figure 6: Function and gradient evaluation using GA for parallel data management.

load balancing based on the master-worker strategy. This strategy has associated scalability issues, because with the increased number of processors, management of the task list by a single master processor becomes a bottleneck. Hierarchical master-worker implementations (with multiple masters) [Matthey and Izaguirre 2001] address that part of the problem; however, the introduced synchronization between multiple masters degrades the performance. Moreover, the message-passing implementation of this strategy can be quite complex. On the other hand, the implementation of dynamic load balancing using GA atomics (*fetch-and-increment* operation) involves only a few lines of code, while the overall performance of the simulation is competitive with the MPI-1 version [Tipparaju et al. 2003].

As explained in Section 3, through a single interface TAO provides access to both nonlinear conjugate gradient (CG) and limited-memory variable-metric (LMVM, also known as L-BFGS) methods. Although Newton-type methods are also supported, large-scale problems with dense Hessian matrices easily exhaust the available computing resources. For example, a matrix with 64,000 rows and columns would require over 32 GB of RAM and $O(64{,}000^3)$ floating-point operations to factor. Therefore, CG and LMVM methods, which do not require the Hessian matrix, offer an appealing alternative for the Lennard-Jones application, which has global connectivity among its variables. The ability of variable-metric methods to create a coarse approximation to the Hessian [Nocedal and Wright 1999] has meant that they can often find solutions to minimization problems by using fewer function evaluations than do CG methods. Figure 7 demonstrates this

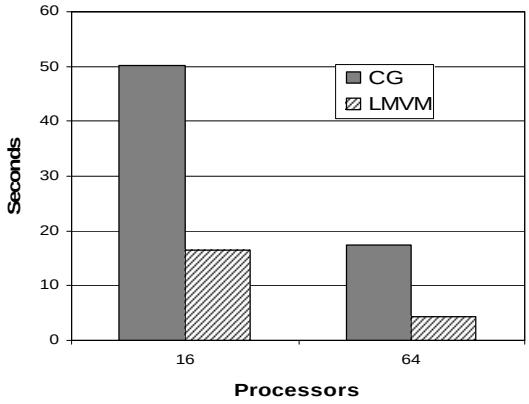

Figure 7: Comparison of the CG and LMVM algorithms for 2,048 particles.

situation for a sample Lennard Jones problem with 2,048 particles on 16 and 64 processors, where we used the Polak-Ribière version of CG as well as LMVM with the default value of five correction pairs.

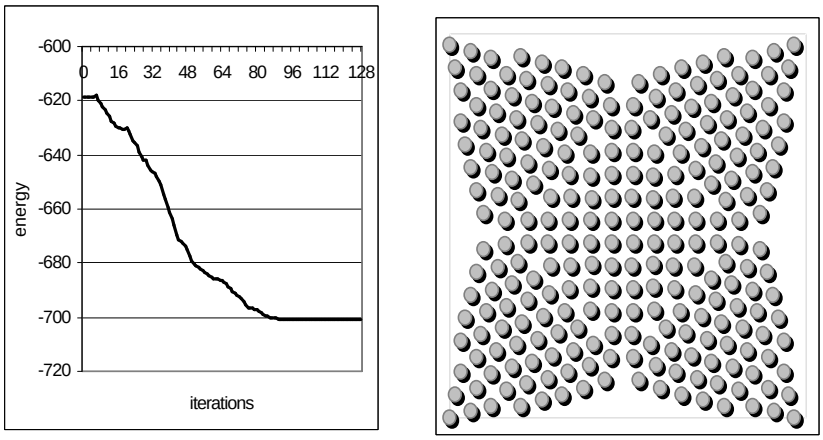

Figure 8: Energy optimization steps of the *limited-memory variable-metric method*. Energy after each iteration (left) and the final configuration for a 256-particle system in the computational domain (right) with periodic boundary conditions.

Molecular configurations with a relative residual (defined to be the norm of the gradient divided by the absolute value of the objective function) less than 0.001 were considered sufficiently optimal to terminate the solver. The experiments were performed on the 2.4 GHz Pentium-4 Linux cluster with Myrinet-2000 at the State University of New York at Buffalo. Figure 8 shows the energy after each iteration of the LMVM method (left) and the final configuration for a 256-particle system in the computational domain (right) with periodic boundary conditions.

Figure 9 shows the parallel speedup of the LMVM algorithm, which is defined as the ratio of the time required to solve a problem using a single processor and the time needed

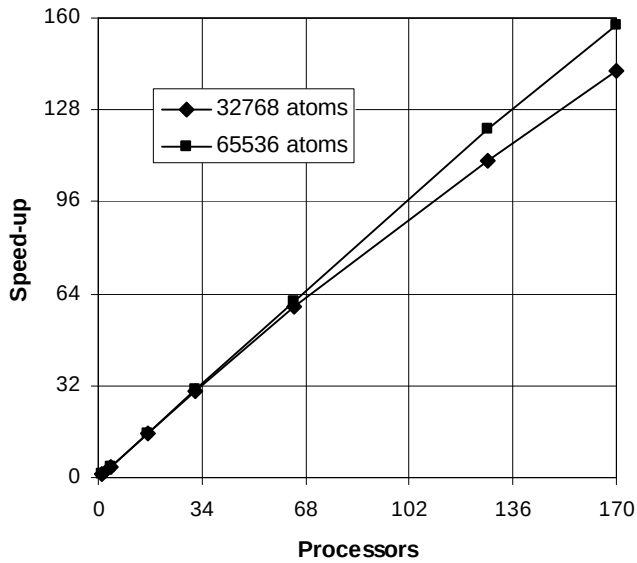

Figure 9: Speedup in the Lennard-Jones potential energy optimization for 32,768 and 65,536 particles.

to solve the problem using *n* processors. Sixty-four processors achieved speedups of 59.4 and 61.5 for the 32,768 and 65,536 particle examples, respectively. For these large problems, the cost of the parallel computations dominates the cost of passing messages among processors. When run on 128 processors, the LMVM solver employed 939 function/gradient evaluations for the 32,768 particle example and 1,084 function/gradient evaluations for the 65,536 particle example. The number of function evaluations using other processor counts was nearly identical, which indicates that the algorithms is numerically stable with respect to multiple processors. While in exact arithmetic the number of iterations should be the same, in our experience, small changes in the iteration numbers for the LMVM method are to be expected and are not a cause for concern. We also note that the nonlinear conjugate gradient algorithm scaled comparably.

Additional tests were performed to evaluate the overhead of the CCA component version with respect to the noncomponent (library-approach) C interface version. We measured the overall wall-clock times of the C and CCA versions and repeated each test five times. Denoting the average wall-clock times over five runs of the C and CCA versions as t_c and t_{cca} respectively, we measure the overhead of CCA as $(t_{cca} - t_c)/t_c$. As shown in Figure 10, the performance overhead due to the CCA approach is less than 0.2% in most cases, a negligible amount when compared to the total execution time. For example, when a minimization of 32,768 particles was run on 64 processors, each iteration of LMVM used about 1 second, and the 0.002 seconds overhead incurred by the CCA was about the same as the time for a global reduction.

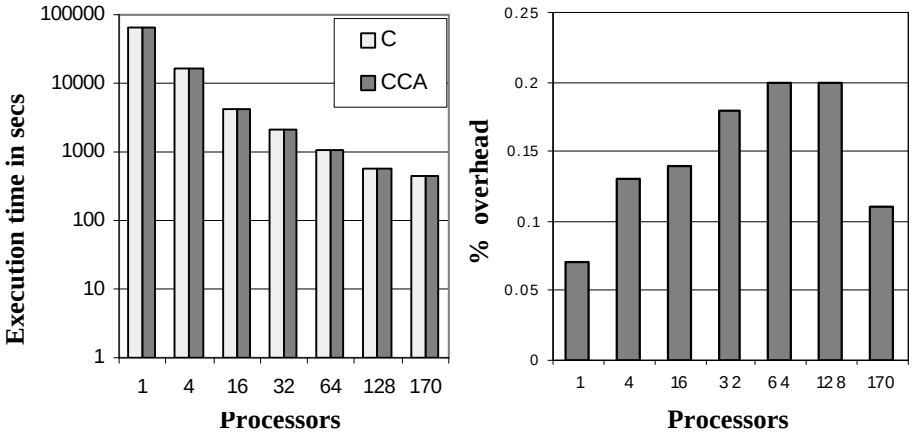

Figure 10: Performance (left) and percentage overhead (right) of the C version (noncomponent) vs. CCA version (component) of Lennard-Jones molecular dynamics using GA/TAO for 32,768 particles.

7. CONCLUSIONS

This paper highlights key features of two complementary toolkits within the ACTS collection, namely, the Global Arrays library and the Toolkit for Advanced Optimization. In combination they provide state-of-the art capabilities for distributed data management and optimization for large-scale computational science problems. The GA toolkit provides efficient vector and matrix classes, support for dynamic load balancing using one-sided and atomic operations, and an easy-to-use interface for efficiently managing dense distributed arrays. TAO provides first- and second-order optimization methods that are portable, robust, and efficient, with a common interface for each class of problems (such as unconstrained and bound-constrained), so that applications can switch from classic methods to recently developed techniques without restructuring their code.

Performance results of molecular configuration computations with a Lennard-Jones model demonstrate the combined effectiveness of the two toolkits using both conventional libraries and components compliant with the Common Component Architecture (CCA). Advantages for applications developers include (1) efficient and scalable performance achieved by leveraging highly tuned implementations of modern algorithms written by experts in the field and (2) reduced overall application development time by focusing users' code development on application-specific phases, where their primary interest and expertise reside. The latter feature is important even on single processor machines, and the complexities of multiprocessor systems magnify its advantages.

We also discuss some general principles in high-performance software integration that not only are important to this work with the GA and TAO libraries but are also broadly applicable to large-scale computational science projects. Such software interoperability, which promotes the reuse of well-tested and tuned software throughout the scientific computing community, is one of the key goals of the ACTS project [DOE-ACTS 1998]. Indeed, the fact that fundamental changes to the original toolkits were not needed and the CCA specification to enable them to function well together on a practical level is a testament to the careful thought and design that have gone into their development.

These features have already attracted the attention of computational scientists solving large-scale problems. In particular, efforts involving quantum chemistry packages MPCQ [Janssen 2003] at Sandia National Laboratories and NWChem [Bernholdt et al. 1995] at

Pacific Northwest National Laboratory have incorporated the CCA-compliant components based on GA and TAO into their applications [Kenny et al. 2004]. Both GA and TAO are freely available to the public and portable to a wide variety of modern parallel architectures.

ACKNOWLEDGMENTS

The work at ANL was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing, Office of Science, SciDAC Program, U.S. Department of Energy, under Contract W-31-109-Eng-38. The work at PNNL was supported by the DoE-2000 ACTS toolkit project and DoE SciDAC Center for Component Technology for Terascale Simulation Software. We thank our colleagues within the Common Component Architecture [CCA-Forum 2005] for stimulating discussions of issues in component and interface design. We also thank the anonymous referees, whose careful reading and constructive suggestions improved this article significantly.

REFERENCES

Allan, B.A., Armstrong, R.C., Wolfe, A.P., Ray, J., Bernholdt, D.E., and Kohl, J.A. 2002. The CCA core specification in a distributed memory SPMD framework. *Concurrency Computataions 14*, 1-23.

Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., and Smolinski, B. 1999. Toward a common component architecture for high-performance scientific computing. In *Eighth International Symposium on High Performance Distributed Computing*, 115 - 124.

Averick, B.M., Carter, R.G., and Moré, J.J. 1991. The MINPACK-2 test problem collection. ANL/MCS-TM-150. Argonne National Laboratory.

Balay, S., Buschelman, K., Eijkhout, V., Gropp, W., Kaushik D., Knepley, M., McInnes, L., Smith, B., and Zhang, H. 2005. PETSc users manual, ANL-95/11, Revision 2.3.0, Mathematics and Computer Science Division, Argonne National Laboratory. http://www.mcs.anl.gov/petsc.

Balay, S., Gropp, W.D., McInnes, L.C., and Smith, B.F. 1997. Efficient management of parallelism in object oriented numerical software libraries. In Arge, E., Bruaset, A.M. and Langtangen, H.P. eds. *Modern Software Tools in Scientific Computing*, Birkhauser Press, 163-202.

Benson, S., McInnes, L., Moré, J.J., and Sarich, J. 2005. TAO users manual. ANL/MCS-TM-242 - Revision 1.7. Mathematics and Computer Science Division, Argonne National Laboratory.

Benson, S.J., McInnes, L.C., and Moré, J.J. 2001. A case study in the performance and scalability of optimization algorithms. *ACM Transactions on Mathematical Software 27*, 3 (September), 361-376.

Benson, S.J., and Moré, J.J. 2001. A limited-memory variable-metric algorithm for bound-constrained minimization. ANL/MCS-P909-0901. Argonne National laboratory.

Bernholdt, D.E., Allan, B.A., Armstrong, R., Bertrand, F., Chiu, K., Dahlgren, T.L., Damevski, K., Elwasif, W.R., Epperly, T.G.W., Govindaraju, M., Katz, D.S., Diachin, L.F., Kohl, J.A., Krishnan, M., Kumfert, G., Lefantzi, S., Lewis, M.J., Malony, A.D., McInnes, L.C., Nieplocha, J., Norris, B., Parker, S.G., Ray, J., Shende, S., Windus, T.L., and Zhou.S. 2005. A component architecture for high-performance scientific computing. *Intl. J. High-Perf. Computing Appl. Submitted to ACTS Collection special issue, in press.*

Bernholdt, D.E., Apra, E., Fruchtl, H.A., Guest, M.F., Harrison, R.J., Kendall, R.A., Kutteh, R.A., Long, X., Nicholas, J.B., Nichols, J.A., Taylor, H.L., Wong, A.T., Fann, G.I., Littlefield, R.J., and Nieplocha, J. 1995. Parallel computational chemistry made easier: The development of NWChem. *Int. J. Quantum Chem. Symposium 29*, 475-483.

Bernholdt, D.E., Nieplocha, J., and Sadayappan, P. 2004. Raising the level of programming abstraction in scalable programming models. In *HPCA Workshop on Productivity and Performance in High-End Computing (P-PHEC 2004)*, Madrid, Spain.

Bershad, B.N., Zekauskas, M.J., and Sawdon, W.A. 1993. Midway distributed shared memory system. In *38th Annual IEEE Computer Society International Computer Conference - COMPCON SPRING '93, Feb 22-26 1993*, Publ by IEEE, Piscataway, NJ, USA, San Francisco, CA, USA, 528-537.

Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R.C. 1997. ScaLAPACK: A linear algebra library for message-passing computers. In Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, MN.

CCA-Forum. 2005. Common Component Architecture Forum. http://www.cca-forum.org.

CCA-Spec. 2005. http://www.cca-forum.org/docs/specification.html.

Cleary, A., Kohn, S., Smith, S., and Smolinski, B. 1999. Language interoperability mechanisms for high-performance scientific applications. In SIAM Workshop on O*bject Oriented Methods for Inter-operable Scientific and Engineering Computing*, 30-39.

Cox, A.L., Dwarkadas, S., Lu, H., and Zwaenepoel, W. 1997. Evaluating the performance of software distributed shared memory as a target for parallelizing compilers. In *1997 11th International Parallel Processing Symposium, IPPS 97, Apr 1-5 1997*, IEEE, Los Alamitos, CA, USA, Geneva, Switz, 474-482.

Dakota. 2005. Dakota Web Page. http://endo.sandia.gov/DAKOTA/software.html.

DOE-ACTS. 1998. The DOE ACTS Collection. http://acts.nersc.gov/.

DOE-SciDAC. 2005. http://www.osti.gov/scidac/.

Drummond, L.A., and Marques, O. 2003. The Advanced Computational Testing and Simulation Toolkit. LBNL-50414. Lawrence Berkeley National Laboratory. Technical Report.

Freeh, V.W., and Andrews, G.R. 1996. Dynamically controlling false sharing in distributed shared memory. In *1996 5th IEEE International Symposium on High Performance Distributed Computing, Aug 6-9 1996*, Syracuse, NY, USA, 403-411.

Gao, G., and Sarkar, V. 2000. Location consistency -- a new memory model and cache consistency protocol. *IEEE Transactions on Computers 49*, 8, 798-813.

Hoare, M.R. 1979. Structure and dynamics of simple microclusters. *Advances in Chemical Physics 40*, 49-135.

Hough, P., Kolda, T., and Torczon, V. 2001. Asynchronous parallel pattern search for nonlinear optimization. *SIAM J. Scientific Computing 23*, 1, 134-156.

Janssen, C. 2003. The Massively Parallel Quantum Chemistry Program. http://aros.ca.sandia.gov/~cljanss/mpqc/index.html.

Kenny, J.P., Benson, S.J., Alexeev, Y., Sarich, J., Janssen, C.L., McInnes, L.C., Krishnan, M., Nieplocha, J., Jurrus, E., Fahlstrom, C., and Windus, T.L. 2004. Component-based integration of chemistry and optimization software. Journal of Computational Chemistry 25, 14, 1717-1725.

Kohn, S., Kumfert, G., Painter, J., and Ribbens, C. 2001. Divorcing language dependencies from a scientific software library. In Tenth SIAM Conference on Parallel Processing.

Kolda, T. 2005. APPS web page. http://software.sandia.gov/appspack/.

Kruskal, C.P., and Weiss, A. 1985. Allocating independent subtasks on parallel processors. *IEEE Trans. Softw. Eng. 11*, 10, 1001-1016.

Lin, C.J., and Moré, J.J. 1999. Newton's method for large bound-constrained optimization problems. *SIAM Journal of Optimization 9*, 4, 1100-1127.

Matthey, T., and Izaguirre, J.A. 2001. ProtoMol: A molecular dynamics framework with incremental parallelization. In *Tenth SIAM Conf. on Parallel Processing for Scientific Computing (PP01)*, Society for Industrial and Applied Mathematics.

McInnes, L.C., Allan, B.A., Armstrong, R., Benson, S.J., Bernholdt, D.E., Dahlgren, T.L., Diachin, L.F., Krishnan, M., Kohl, J.A., Larson, J.W., Lefantzi, S., Nieplocha, J., Norris, B., Parker, S.G., Ray, J., and Zhou.S. 2005. Parallel PDE-based simulations using the common component architecture. ANL/MCS-P1179-0704. To appear in the book Numerical Solution of Partial Differential Equations on Parallel Computers, A. M. Bruaset, P. Bjorstad, and A. Tveito, editors, Springer-Verlag. Argonne National Laboratory.

Meza, J.C. 1994. OPT++: An object-oriented class library for nonlinear optimization. SAND94-8225. Sandia National Laboratories.

Moré, J.J., and Toraldo, G. 1991. On the solution of large quadratic programming problems with bound constraints. *SIAM Journal of Optimization 1*, 93-113.

Moré, J.J., and Wright, S.J. 1993. *Optimization Software Guide*. SIAM Publications, Philadelphia.

MPI-Forum. 1994. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications and High Performance Computing 8*, 3/4 (Fall-Winter), 159-416.

Nieplocha, J., and Carpenter, B. 1999. ARMCI: A portable remote memory copy library for distributed array libraries and compiler run-time systems. In RTSPP of IPPS/SDP'99.

Nieplocha, J., Harrison, R.J., Krishnan, M., Palmer, B., and Tipparaju, V. 2002. Combining shared and distributed memory models: Evolution and recent advancements of the Global Array Toolkit. In POHLL'2002 workshop of ICS-2002, New York.

Nieplocha, J., Harrison, R.J., and Littlefield, R.J. 1994. Global Arrays: A portable shared memory programming model for distributed memory computers. In Supercomputing, IEEE CS Press, 340-349.

Nieplocha, J., Harrison, R.J., and Littlefield, R.J. 1996. Global Arrays: A nonuniform memory access programming model for high-performance computers. Journal of Supercomputing 10, 2, 169-189.

Nieplocha, J., and Ju, J. 2000. ARMCI: A portable aggregate remote memory copy interface. http://www.emsl.pnl.gov:2080/docs/parsoft/armci/armci1-1.pdf.

Nieplocha, J., Trease, H., Ju, J., Rector, D., and Palmer, B. 2001. Building an application domain specific programming framework for computational fluid dynamics calculations on parallel computers. In Tenth SIAM Conference on Parallel Processing for Scientific Computing.

Nocedal, J., and Wright, S. 1999. Numerical Optimization. Springer-Verlag, New York.

PeIGS. 2005. PeIGS Home Page. http://www.emsl.pnl.gov/docs/nwchem/doc/peigs/docs/peigs3.html.

Plimpton, S. 1995. Fast parallel algorithms for short-range molecular-dynamics. Journal of Computational Physics 117, 1 (Mar 1), 1-19.

Plimpton, S., and Heffelfinger, G. 1992. Scalable parallel molecular dynamics on MIMD supercomputers. In *Scalable High Performance Computing Conference*, 246-251.

Rodrigues, J.F. 1987. *Obstacle Problems in Mathematical Physics*. Elsevier Publishing Company, Amsterdam.

Sarich, J. 2004. A programmer's guide for providing CCA component interfaces to the toolkit for advanced optimization. ANL/MCS-TM-279. Argonne National Laboratory.

Tipparaju, V., Krishnan, M., Nieplocha, J., Santhanaraman, G., and Panda, D. 2003. Exploiting non-blocking remote memory access communication in scientific benchmarks. In *High Performance Computing - HiPC*, 248-258.

Zhang, Y., Tipparaju, V., Nieplocha, J., and Hariri, S. 2005. Parallelization of the NAS conjugate gradient benchmark using the global arrays shared memory programming model. In *19th IEEE International Parallel and Distributed Processing Symposium.*

Zhou, Y., Iftode, L., and Li, K. 1996. Performance evaluation of two home-based lazy release consistency protocols for shared virtual memory systems. In *Operating Systems Design and Implementation Symposium.*