SAND2013-8430J

??

Albany: A Component-Based Partial Differential Equation Code Built on Trilinos

ANDREW G. SALINGER, ROSCOE A. BARTLETT, QUISHI CHEN, XUJIAO GAO, GLEN A. HANSEN, IRINA KALASHNIKOVA, ALEJANDRO MOTA, RICHARD P. MULLER, ERIK NIELSEN, JAKOB T. OSTIEN, ROGER P. PAWLOWSKI, ERIC T. PHIPPS, WAICHING SUN, Sandia National Laboratories

The code development strategy, software design, and results from two application projects are presented for the Albany code: an implicit, unstructured grid, finite element code for the solution and analysis of partial differential equations. The driving strategy behind the development of Albany is the notion that it is increasingly possible, and advantageous, to build an application code from reusable software libraries connected by well-designed abstract interfaces. The main advantages and disadvantages of this strategy are presented. This approach is increasingly possible because of the tremendous breadth of capabilities now available in software libraries. These notably include the libraries delivered through the Trilinos suite of computational science tools which are the building blocks of Albany. The major features of the design of Albany, specifically the use of abstract layers and heavy use of independent libraries, are presented. Finally, two distinct case studies are shown that validate this approach by using the Albany code base to rapidly develop application codes born with a large set of solution and analysis capabilities.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: Algorithm design and analysis; G.1.8 [**Partial Differential Equations**]: Finite element methods; D.1.5 [**Programming Techniques**]: Object-oriented Programming

General Terms: Algorithms, Design, Documentation

Additional Key Words and Phrases: Partial differential equations, finite element analysis, template-based generic programming.

ACM Reference Format:
Andrew G. Salinger, Roscoe A. Bartlett, Quishi Chen, Xujiao Gao, Glen A. Hansen, Irina Kalashnikova, Alejandro Mota, Richard P. Muller, Erik Nielsen, Jakob T. Ostien, Roger P. Pawlowski, Eric T. Phipps, WaiChing Sun. 2013. Albany: A Component-Based Partial Differential Equation Code Built on Trilinos. *ACM Trans.*

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Corresponding author: Andrew G. Salinger, Sandia National Laboratories, Numerical Analysis and Applications Department, PO Box 5800 MS-1318, Albuquerque, New Mexico, 87185, USA. agsalin@sandia.gov.

Roscoe A. Bartlett's current affiliation is Oak Ridge National Laboratories.

Quishi Chen's current affiliation is Clemson University, Department of Civil Engineering.

WaiChing Sun's current affiliation is Columbia University, Department of Civil Engineering and Engineering Mechanics.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2013 ACM 0098-3500/2013/10-ART?? $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000

Math. Softw. ??, ??, Article ?? (October 2013), 27 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

In this paper we present the Albany code, a parallel, unstructured-grid, implicit, finite element code for the solution and analysis of partial differential equations (PDEs). Albany has been developed to drive and demonstrate a component-based approach to scientific application development. This approach, which we will discuss in detail in this paper, is to make very broad use of computational science libraries, abstract interfaces, and software engineering tools. The idea is that by making extensive use of external libraries, the application code can remain focused on the PDE development and have access to numerous algorithms, each written by domain experts, where the cost of verification and maturation are amortized over many projects. The hypothesis is that codes can be written more rapidly, be born with more sophisticated algorithms, and have a smaller code base to support. There are drawbacks to depending so extensively on external libraries, but these can in large part be mitigated by good software design and processes.

The breadth of useful computational science libraries have grown beyond the widely-accepted LAPACK, Blas, and MPI, and even well beyond general linear solver libraries. The Trilinos [Heroux et al. 2005] and Dakota [Adams et al. 2009] toolkits house many dozens of computational science libraries that, in aggregate, perform the vast majority of computational tasks needed for the setup, solution, and analysis of systems of PDEs. The initial goals of the Albany project were to push how much of an application code, outside of the implementation of the application-specific PDE terms and response functions, could be satisfied with general-purpose libraries and to identify capability gaps that could be satisfied with libraries but are currently redundantly developed in each application code. So, where possible, Albany makes use of the computational science libraries within Trilinos and Dakota. Much of the early code development work in Albany therefore focused on developing, maturing, and satisfying interfaces between libraries that had been independently developed. In the beginning of the development process, the interoperability between libraries and maturity of the abstract interfaces in Trilinos varied significantly. In many cases existing Trilinos capabilities were mature and fully usable out of the box, in some cases there was an initial implementation present that needed maturation, and in some cases the Albany code was the incubator for the capability that was later migrated to Trilinos for general use.

At this point, the code has reached a high level of maturity, where there is a well-defined modular code design and where dozens of independent libraries all contribute to the application code. As a result, applications developed within Albany are born with mesh I/O, load balancing, finite element discretizations, distributed-memory linear algebra objects, preconditioners, nonlinear solvers, optimization algorithms, and uncertainty quantification (UQ) capabilities, all through library calls and satisfying abstract interfaces. The code base to support in Albany is therefore relatively small. As a measure of success of this approach, approximately 80% of the lines of code in Albany involve definition and implementation of the equations and responses (*i.e.,* various quantities of interest).

Certainly all of the elements listed in the previous paragraph are satisfied by other finite element libraries and frameworks. A differentiation for Albany is that it was designed to maximize external dependencies and to evaluate how well this process works in practice. A number of finite element applications and toolkits have been developed that leverage this component integration model. An exhaustive list is beyond the scope of this paper. A natural first point of abstraction for implicit and semi-implicit methods is to separate the finite element assembly process from the linear and nonlinear solu-

tion algorithms. The assembly tools (*e.g.,* basis function library and meshing library) are usually developed internally by the application code and solvers are leveraged from external libraries such as PETSc [Balay et al. 2013] and Trilinos [Heroux et al. 2005]. Examples of codes that fall into this category are the Differential Equations Analysis Library (`deal.II`) [Bangerth et al. 2007], libMesh [Kirk et al. 2006], Life V [Prud'homme 2007], Sierra [Stewart and Edwards 2003], and Uintah [de St. Germain et al. 2000]. Some projects additionally section the assembly process into separately releasable components or rely on external components. Projects in this class include Albany, the FEniCS project [Logg et al. 2012], the MOOSE project [Gaston et al. 2009], and the Sundance rapid development system [Long et al. 2010].

In this paper, we discuss our experiences in developing a scientific application code with a very aggressive component-based approach. In section 2 we provide details on how we define a component-based approach, what the scope of the current effort is, and what we see as the advantages and disadvantages of this approach.

In Section 3 we present the design of the Albany code. In particular, we show schematically where we have inserted abstract interfaces between major domains of the code to maintain modularity. We go into detail into the separate code domains and detail what capabilities are accessible from behind those abstract layers. Some of these layers live in Albany and some in Trilinos itself. There is certainly not a unique or optimal design for how to modularize a PDE code with abstract interfaces, but this design has held up well in the transition from a computational science research project to the current use of Albany as a platform for developing new application codes and analysis capabilities.

In Section 4 we present two computational science application projects that have been developed in Albany. The first is a computational mechanics research and development platform, that enables research in solution methods, discretizations, full coupling of mechanics to scalar equations, material models, and failure and fracture modeling. The second is a quantum device simulation and design capability, where nonlinear Poisson and coupled Schrodinger-Poisson systems are solved for designing quantum dots, the building blocks of quantum computers. The success of these projects in rapidly developing new application code with immediate access to a host of solution and analysis capabilities provides evidence to the strength of the component-based approach to computational science application development as embodied in the Albany code.

2. COMPONENT-BASED APPLICATION CODE DEVELOPMENT STRATEGY

The Albany code was written to drive and demonstrate the component-based strategy for application code development. This approach is to build application codes primarily from modular pieces, such as independently developed software libraries. The crux of this strategy involves the accumulation of components across four classes of software: libraries, interfaces, software quality tools, and demonstration applications, which form the foundation for the new code. The benefits of this approach are numerous, and are discussed in detail below. However, it is evident that having a significant collection of advanced algorithmic capabilities as a foundation for the development of new applications provides a large advantage over starting from scratch or retrofitting a monolithic code that was designed for a different class of problems. Just as compilers, BLAS, Lapack, and MPI have long been standard external dependencies (it is also common to depend on external linear solvers and meshing tools), we extrapolate this trend to include dozens of other required algorithmic capabilities that can be generalized into reusable libraries.

2.1. Computational Science Libraries

Figure 1 enumerates individual computational capabilities that can be deployed as independent libraries, and made available as building blocks for new application codes. The capabilities are grouped in a logical manner. This serves to organize the presentation of this list; but as will be discussed later, also shows where opportunities exist for the definition of abstraction layers around clusters of related libraries. Many of the listed components shown in the figure represent a set of capabilities, in that there are multiple independent libraries that provide competing or complimentary capabilities in the listed capability area (e.g. preconditioners).

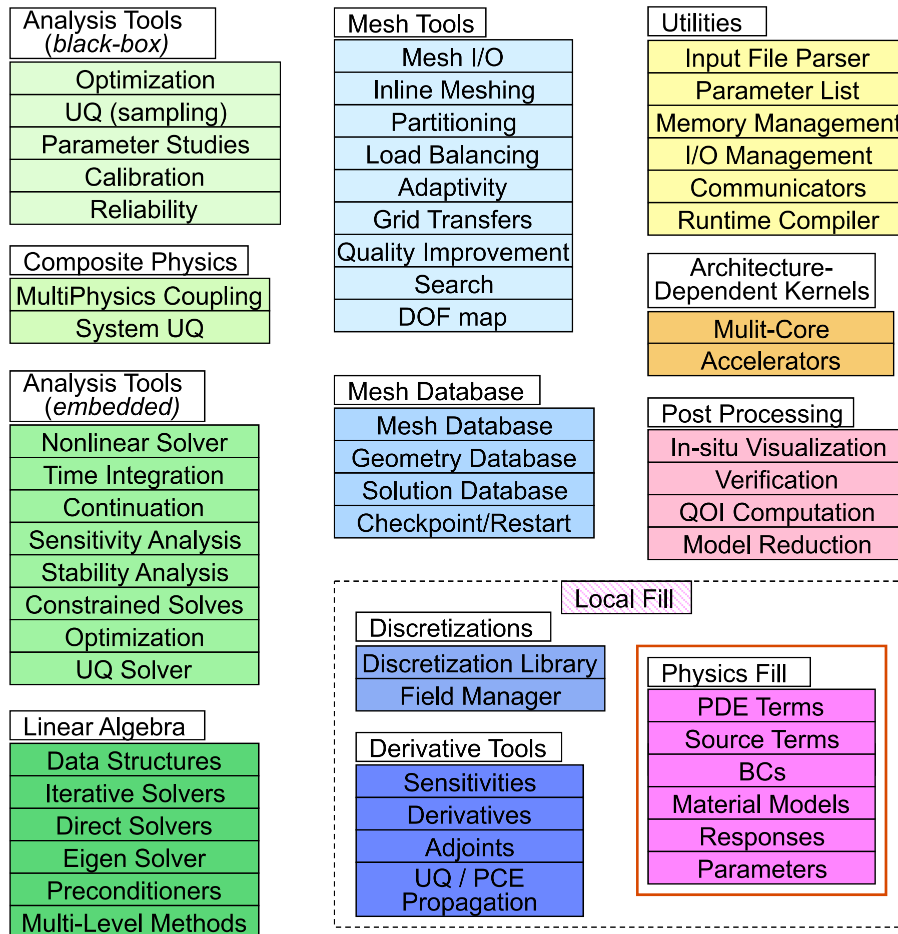

Fig. 1. Enumeration of various computational science capabilities that can be delivered through independent software libraries. Given that the above components exist in a mature state, the time to write new PDE codes employing these capabilities is dramatically reduced, with development time being concentrated in the `Physics Fill` box outlined in red.

The granularity of the definition of an independent library is a software design decision, where the extremes (having all capabilities as one monolithic framework or

having every C++ class an independent library) are obviously sub-optimal. For this description, which has a direct correlation to the development of independent packages in the Trilinos suite, a library is typically constructed by one to three domain experts. This level of effort is small enough that the lead developers can know and understand the entire code base of each package. In this definition, an existing library may be separated into a set of smaller independent pieces if the common usage involves only a subset of the capabilities contained in the original library. For instance, a library that contains both a GMRES linear solver and an ILU preconditioner would be split into two separate libraries since we would want to enable the use of the ILU preconditioner with any linear solver.

2.2. Software Quality Tools

Central to productivity of code teams are the use of software quality tools and processes. The benefits of these tools increase significantly as project teams grow in number of developers and become geographically distributed. In Figure 2, we present a list of software quality tools which enhance productivity of a code project such as Albany. As with the libraries presented in the previous section, it is not necessary to use all the capabilities to see benefits. Having a full set of these tools available for a new project saves the project from needing to independently select appropriate tools and integrate them into the development process; the new project is born with suitable tools and processes in place.

Software Quality Tools

Backups
Version Control
Build System
Regression Tests
Configuration Mgmt
Automated Tests
Porting
Verification Tests
Mailing Lists
Web Pages
Bug Tracking
Performance Testing
Code Coverage
Unit Testing
Release Process

Fig. 2. In addition to computational science libraries, the rapid development of new application codes is also strongly dependent on the availability and use of an effective set of software quality tools and processes. These support developer productivity, software quality, and the longevity of a project.

For computational science organizations, there is great benefit in sharing the same sets of tools and processes across many software projects. With a decrease in each project-specific learning curve, staff have more agility to make an impact on multiple projects. For this reason, the Albany project has largely adopted the set of tools used by the Trilinos project.

2.3. Advantages and Disadvantages of Component-Based Code Design

With our experience in Albany and other application codes that use Trilinos libraries, we have noted significant advantages and some disadvantages in using the component-based approach to code design. These span both technical and social issues, and are influenced by the organizational culture, funding, and several other factors. A more extensive discussion of component-based design is presented in a technical report [Salinger 2012].

Advantages of a component-based approach to application development include:

(1) The costs of writing, verifying, maturing, extending, and maintaining a library is amortized over multiple projects.
(2) Shared support of an algorithmic capability over several projects allows the pooling of a critical mass of funding for a subject matter expert to develop extensive expertise and software capabilities in a targeted area.
(3) Using the algorithmic library developed above typically gives the application code full access to the talents of the subject-matter expert.
(4) The use of general-purpose libraries developed externally to an application code forces the code to take on a more modular design. This makes it more flexible and extensible in the long run.
(5) The use of libraries decreases the code base that must be maintained by the application team. The finer granularity of this approach creates natural divisions between code appropriate for open source release and code that must be protected (e.g. for intellectual property or export control reasons), decreasing the amount of code that requires protections.
(6) The use of abstract interfaces around groups of related capabilities facilitates implementation and investigation of alternative algorithms. Using an example from Trilinos, several direct and iterative solvers share the same interface and can be selected in Albany at run time.
(7) The effort to create abstract interfaces that support multiple concrete implementations improves the extensibility and flexibility of the code. Creating an abstract layer between the mesh database and the mesh data structures used in the PDE assembly enables us to flexibly use multiple mesh databases with minimal impact on the code.

In contrast with a monolithic application code that contains all required algorithms as part of the application, disadvantages of a component-based approach are:

(1) The use of numerous Third-Party Libraries (TPLs) can complicate the build process. It can be particularly difficult to keep track of what versions of libraries are compatible with each other.
We mitigate this issue by focusing on the use of libraries from Trilinos, which synchronizes the release of its numerous (> 50) libraries. Albany also links to a host of parallel unstructured mesh and adaptation libraries contained within the Rensselaer Polytechnic Institute (RPI) Scientific Computation Research Center (SCOREC) toolset [Seol et al. 2012], these libraries are likewise synchronized to the underlying Trilinos build by sharing the TriBITS [Bartlett et al. 2012] build system.
(2) When debugging the application, developers on the application code team may have difficulty tracking down issues in unfamiliar components and may not get access to or help from the component developer.
(3) The development of abstract interfaces that compartmentalize the code are difficult to write and require a different skill set than those always present on an application development team.
(4) General purpose libraries with an improper interface design can lead to applications that do not perform optimally (*e.g.,* performing unnecessary data copies) and have unnecessarily high memory requirements.
(5) The dependence on external components can significantly impact the deployment of an application to novel/disruptive technologies. For example, the porting of a code from traditional CPU cores to general purpose graphics processing units (GPGPUs) requires that many components be rewritten to support that architecture. Even if some components support the architecture, it may not be possible to

run the application on the new hardware until all, or a large subset of components provide that support.

The application development projects described in this paper in Section 4, and related observations, give us anecdotal experience that the component-based approach has net benefits. In particular, we cite the ability to rapidly add new algorithms and capabilities by making use of pre-existing libraries. The other strength of this approach is that it demonstrates an accelerated development process that builds on experience; the more that libraries are developed and matured, the more rapidly the next application using those libraries can be constructed and verified. This can yield significant strategic return on investment across a computational science organization even when it may be of less immediate value within an individual code project.

3. ALBANY COMPONENT-BASED CODE DESIGN

The Albany code was developed to refine, demonstrate, and evaluate the component-based code design strategy. We seek to answer the question of whether a fully-functional PDE application code can be written primarily from computational science libraries, and what gaps remain. Along the way, proxy applications of heat transfer and incompressible flows were supplanted by independently-funded application projects as the drivers (see Section 4). In this section we present the software design of the Albany code, detailing where we have placed abstract interfaces to gain access to general-purpose libraries and to maintain the flexibility and extensibility of a modular design.

Albany is designed to compute approximate solutions to coupled PDE problems represented abstractly as

$$\mathcal{L}(\dot{u}(x,t), u(x,t)) = 0, \quad x \in \Omega, \quad t \in [0,T], \quad \dot{u}, u \in H, \tag{1}$$

where $\Omega \subset \mathbb{R}^d$ ($d = 1,2,3$) and $[0,T]$ are the spatial and temporal domains, $\mathcal{L}$ is a (possibly nonlinear) differential operator, H is a Hilbert space of functions upon which $\mathcal{L}$ is defined, u is the (unknown) PDE solution, and $\dot{u}$ its corresponding time-derivative. Equation 1 is then discretized in space via the (generally unstructured grid) finite element method resulting in the finite-dimensional differential-algebraic system

$$\mathbf{f}(\dot{\mathbf{u}}(t), \mathbf{u}(t), \mathbf{p}) = 0, \tag{2}$$

where $\mathbf{u} \in \mathbb{R}^n$ is the unknown solution vector, $\dot{\mathbf{u}} \in \mathbb{R}^n$ is its time derivative, $\mathbf{p} \in \mathbb{R}^m$ is a set of model parameters, and $\mathbf{f} : \mathbb{R}^{2n+m} \to \mathbb{R}^n$ is the DAE residual function. In Albany, we have focused on fully-implicit solution algorithms which require evaluating and solving linear systems involving the Jacobian matrix

$$\alpha \frac{\partial \mathbf{f}}{\partial \dot{\mathbf{u}}} + \beta \frac{\partial \mathbf{f}}{\partial \mathbf{u}}, \tag{3}$$

and thus accurate and efficient evaluation of these derivatives is critical.

In addition to computing the approximate solution $\mathbf{u}(t)$ one is also often interested in evaluating functionals of the solution

$$\mathbf{s}(t) = \mathbf{g}(\mathbf{u}(t), \mathbf{p}), \tag{4}$$

which we call responses. Values of response functions at discrete time points are often targets of sensitivity and uncertainty analysis, as well as objective functions in optimization, design, and calibration problems. Many of these methods entail evaluation of derivatives of the responses s with respect to the model parameters $\mathbf{p}$, and often the performance of these methods is greatly improved when these derivatives are evaluated accurately. For steady-state problems, the response gradient can computed via

the formula

$$\frac{ds}{d\mathbf{p}} = \frac{\partial \mathbf{g}}{\partial \mathbf{u}}(\mathbf{u}^*, \mathbf{p})\frac{d\mathbf{u}^*}{d\mathbf{p}} + \frac{\partial \mathbf{g}}{\partial \mathbf{p}}(\mathbf{u}^*, \mathbf{p}) \\ = -\frac{\partial \mathbf{g}}{\partial \mathbf{u}}(\mathbf{u}^*, \mathbf{p})\left(\left(\frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{u}^*, \mathbf{p})\right)^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{p}}(\mathbf{u}^*, \mathbf{p})\right) + \frac{\partial \mathbf{g}}{\partial \mathbf{p}}(\mathbf{u}^*, \mathbf{p}), \tag{5}$$

where $\mathbf{u}^*$ satisfies $\mathbf{f}(\mathbf{u}^*, \mathbf{p}) = 0$. The necessity to quickly and accurately evaluate derivatives such as $\partial\mathbf{f}/\partial\mathbf{u}$ and $\partial\mathbf{f}/\partial\mathbf{p}$ (as well as other quantities such as polynomial chaos coefficients) needed by analysis algorithms, as well as to support an easily extensible interface for supplying these quantities to higher-level analysis algorithms, has dictated many of the code design decisions described below.

3.1. Overall Albany Code Design

At a high level, the code is separated into five main algorithmic domains separated by abstract interfaces, as shown in Figure 3. These domains will each be discussed in detail in the following sections.

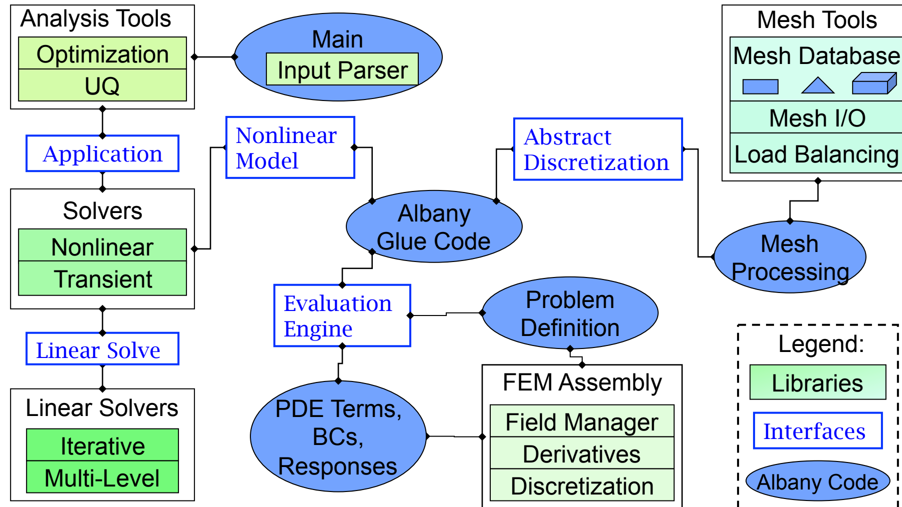

Fig. 3. The Albany code is built largely from software libraries (colored boxes with black font) and abstract interfaces (clear boxes with blue font), and employs software quality tools (not shown). The bulk of the capabilities come from Trilinos libraries encapsulated with abstract interfaces. The bulk of the coding effort for a new application involves writing PDE terms, boundary conditions, and responses.

The key part of the Albany code is depicted as 'Glue Code' in this figure, and is the part of the code base that is not separated as a library and is not physics specific. It depends on a discretization abstraction, which serves as a general interface to a mesh database and mesh services. (As described below in Section 3.2, this interface deals with linear algebra objects and standard vectors, and is agnostic to the specific mesh database.) It also uses a problem class abstraction to construct the set of PDEs, boundary conditions, and response calculations. As described in Section 3.3, the assembly of these physics pieces comes down to the evaluation of a directed graph of computations of field data. The Glue Code then uses these pieces to satisfy the nonlinear model abstraction, *e.g.,* computing a residual vector or Jacobian matrix.

With this nonlinear model interface satisfied, the full range of Trilinos solvers are available. This includes the embedded nonlinear analysis solvers such as nonlinear and transient solves described in Section 3.4. These solvers in turn call the linear solvers (see Section 3.5), which are the most feature rich and mature set of general purpose libraries. Albany was designed to demonstrate how to design a code born with analysis capabilities significantly beyond repeated forward simulation, so the nonlinear solver layer is not the top layer. A separate Analysis layer described in Section 3.6 wraps the solver layer, and performs parameter studies, optimization, and UQ, primarily using algorithms from the Dakota toolkit [Adams et al. 2009].

As presented in Figure 2, there are many software tools and processes that can improve the productivity of a project. Albany has mainly adopted the toolset from Trilinos to minimize the learning curve that Trilinos developers need to begin contributing to Albany. These include *git* for version control, *CMake* for configuration management, build, and porting, the associated *CTest* for regression testing, and Doxygen for automatic documentation based on the class design and comments. We have adopted the mailing lists and webpage design from Trilinos as well. We currently have scripts run under a *cron* job that perform continuous integration with Trilinos and Dakota that do a fresh build and regression testing nightly on multiple machines.

3.2. Global Discretization Abstraction and Libraries

A critical component of any finite element code is the mesh framework, which defines the geometry, element topologies, connectivities, and boundary information, as well as the field information that lives on the mesh. As with many modern codes, in Albany we are starting to support spatial adaptation, where the mesh may change by refining in certain areas, and perhaps coarsening in others, driven by evolving features and error indicators computed during the solution. A further complication involves the need to rebalance the workload between processors as the mesh is modified. Albany accesses the mesh database, adaptation and load balancing capabilities, together with functions used to transfer the solution information between mesh representations, using an abstract Global Discretization interface.

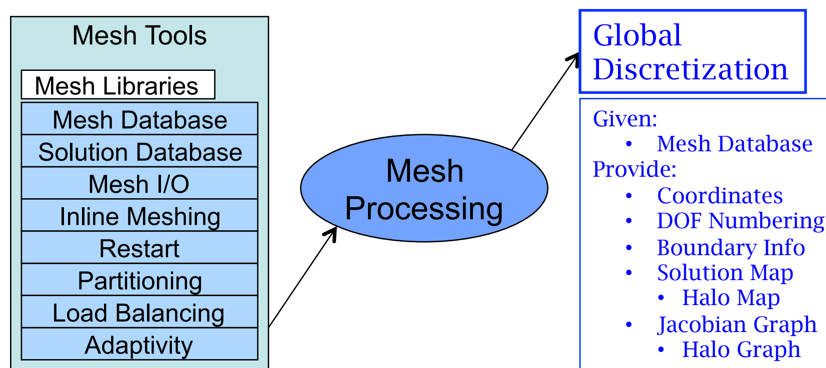

Fig. 4. The finite element mesh and related quantities are exposed to the Albany code through the abstract Global Discretization interface. Depending on the internal details of the mesh library in use, a specialization of the Global Discretization class will construct quantities in the layout needed by the rest of the code, such as coordinates, solution vectors, sparse-matrix graphs, and degree-of-freedom (DOF) numbering/connectivity information.

The global discretization abstraction, presented schematically in Figure 4, gives the finite element assembly process access to the all of the bookkeeping (data distribution) information required by the linear algebra objects. In all cases, mesh information is contained in an in-memory mesh database that is accessed through a specialization of the abstract Global Discretization interface class. These specializations, unique to each mesh library Albany supports, provides a set of common services. These include reading and writing mesh data files present on the file system through I/O routines, providing element topology and vertex coordinate information, and optionally mesh adaptation, load balancing, and solution transfer capabilities. Of note is that each mesh library is different internally and provides services in a unique way. The specialization of the abstract global discretization class may interpret or "fill in" missing or incompatible data representations when required.

We note that the placement of the abstract Global Discretization interface is above the location where a general interface to mesh databases would lie in a domain design. (The design of an abstract interface to mesh databases (e.g. iTaps [Diachin et al. 2007]) has proven tricky, with the competing and at times contradictory demands of codes that use explicit or implicit algorithms, static and adaptive meshes, and C++ vs. C or FORTRAN.) Our interface has methods for the quantities needed directly in the finite element assembly, such as the Jacobian graph and coordinate information, in the data structures desired by the assembly. The offset between the mesh database and the Global Discretization interface is denoted as the Mesh Processing layer in Figure 4. Functions in this layer satisfy the interface using calls and data structures specific to the underlying mesh database. We found this to be a tractable solution for our needs but would only scale to a modest number of mesh databases with distinct interfaces.

Albany currently supports two independent discretization interfaces; (1) the *Sierra ToolKit* (STK) package [Edwards et al. 2010] in Trilinos, and (2) the *Parallel Unstructured Mesh Interface* (PUMI) [Seol et al. 2012] being developed by the Scientific Computation Research Center (SCOREC) at Rensselaer Polytechnic Institute. Furthermore, the STK mesh database can be loaded in several ways: reading of a mesh file in the Exodus format (typically generated witht he CUBIT meshing program), inline meshing with the Pamgen package in Trilinos, and simple rectangular meshes directly created in the code base.

In a typical simulation the interaction with the mesh library begins by Albany instantiating an object of the desired specialized class (which activates constructors in the appropriate places in the underlying mesh library), based on the type of input mesh and geometry files specified by the user. At construction, the mesh library reads the mesh information in serial or parallel depending on the simulation, and performs the degree of processing required to service requests from Albany for discretization data. As the simulation initializes, Albany Glue Code invokes virtual member functions in the abstract discretization object to access coordinate data, connectivity, and to read (when restarting) and write solution data to the specialized class (and underlying library).

For adaptive simulations, there are two additional capability hierarchies that manage both the mesh adaptation process and the criteria used to determine the degree of adaptation needed, each Albany time, load, or displacement step. These interfaces are likewise abstract and specialized to suit the requirements of the mesh adaptation library specified for the simulation.

Other information that is processed on the mesh and accessed through the abstract discretization interface includes the multidimensional array that holds the list of elements on this processor, each with the array of local nodes, and pointers to the solution vector, coordinate vector, and any other field data stored at the nodes. By processing the element connectivity information, as well as some local discretization information

(how many unknowns are on a mesh node), the sparse graph of the Jacobian matrix can be processed. For dealing with overlap (*a.k.a.,* halo or ghosted) information, several objects have both "owned" and "overlap" versions.

3.3. Problem Abstraction and Finite Element Assembly

Given a finite element mesh as supplied by the abstract discretization components, the purpose of the problem abstraction and finite element assembly components is to evaluate the discrete finite element residual, Eq. (2), for the PDE problem at hand, as well as derived quantities such as Jacobian matrices and parameter derivatives needed for simulation and analysis. Our approach for facilitating these calculations that is scalable in not only problem size but also in problem complexity and the number of supported analysis approaches is fully described elsewhere [Pawlowski et al. 2012a; Pawlowski et al. 2012b]. Here we briefly summarize the salient features of this approach and its use within the Albany context.

Multiphysics simulation introduces a number of difficulties that must be addressed by the software framework including managing a multitude of physics models, adapting the simulation to different problem regimes, and ensuring consistency of the coupled PDE residual evaluation with respect to the full system degrees-of-freedom. To manage this complexity, Albany employs the graph-based evaluation approach [Notz et al. 2011; Pawlowski et al. 2012a; Pawlowski et al. 2012b] as provided by the Trilinos Phalanx package [Pawlowski 2011]. Here, the residual evaluation for a given PDE problem is decomposed into a set of terms (at a level of granularity chosen by the developer), each of which is encoded into a Phalanx `evaluator`. Each `evaluator` encodes the variables it depends upon (e.g., temperature evaluated quadrature points for a given set of basis functions), the variables it evaluates (*e.g.,* a material property at those same quadrature points), and the code to actually compute the term. Phalanx then assembles all of the `evaluators` for a given problem into a directed acyclic graph representing the full PDE residual evaluation for a given set of mesh cells stored in a data structure called the `field manager`. The roots of the graph are evaluator(s) that extract degree-of-freedom values from the global solution vector and the leaves are evaluator(s) that assemble residual values into the global residual vector. The full finite element assembly then consists of a loop over mesh cells with the body of the loop handled by the Phalanx evaluation (typically multiple cells are processed by each evaluator, called a work set, to improve performance by amortizing function call overhead over many mesh cells). This approach improves code reuse by allowing common evaluators to be used by many problems, improves efficiency by ensuring each term is only evaluated as necessary, ensures correctness by requiring all evaluator dependencies are met, and allows a wide variety of multi physics problems to be easily constructed. While not required, most terms within Albany employ the Intrepid package [Bochev et al. 2012] for local cell discretization services such as finite element basis functions and quadrature formulas. This graph-based evaluation approach is used by several frameworks for handling multiphysics complexity including the Aria application code in SIERRA [Stewart and Edwards 2003], the Drekar code [Smith et al. 2011], the MOOSE framework [Gaston et al. 2009], and the Unitah framework [de St. Germain et al. 2000].

One of the design goals of Albany was to provide native support for a wide variety of embedded nonlinear analysis approaches such as derivative-based optimization and polynomial chaos-based uncertainty quantification. A significant challenge with these approaches is they require calculation of a wide variety mathematical objects such as Jacobians, Hessian-vector products, parameter derivatives, and polynomial chaos expansions, all of which require augmentation of the assembly process. This is a significant burden on the simulation code developers, which means these approaches are

often not incorporated. This not only limits the impact of these methods but also limits potential research on analysis algorithms for complex multiphysics applications. To address these issues, Albany leverages the template-based generic programming (TBGP) approach [Pawlowski et al. 2012a; Pawlowski et al. 2012b] to provide a framework for easily incorporating existing and new embedded analysis approaches. This technique employs C++ templates and operator overloading to automatically transform code for evaluating the PDE residual into code for computing the quantities described above, and is an extension of operator overloading-based automatic differentiation (AD) ideas to the general case of computing other non-differential objects.

In the Albany setting, each evaluator is written as C++ template code, with a general `EvalT` template parameter. This parameter encodes the evaluation type, such as a residual, Jacobian, parameter derivative or polynomial chaos expansion. Each evaluation type defines a scalar type, which is the data type used within the evaluation itself (e.g., `double` for the residual evaluation or an AD type for the Jacobian and parameter derivative). Each evaluator is then instantiated on all of the supported evaluation types relying on the Sacado [Phipps and Pawlowski 2012; Phipps 2013a] and Stokhos [Phipps 2013b] libraries to provide overloaded operator implementations for all of the arithmetic operations required for each scalar type. This allows the vast majority of evaluators to be implemented in a manner agnostic to the scalar type and the corresponding mathematical object being computed. Furthermore, any evaluator can provide one or more template specializations for any evaluation type where custom evaluation is needed.

Albany leverages template specialization to implement the gather and scatter phases of the finite element assembly for each evaluation type (see Fig. 5). For example the residual specialization of the gather operation extracts solution values out of the global solution vector, and the scatter operation adds residual values into the global residual vector. Likewise, the Jacobian specialization of the gather phase both extracts the solution values and seeds the derivative components of the AD objects, while the scatter operation both extracts the dense element Jacobian matrix from the AD objects and sums their contributions into the global sparse Jacobian matrix. These gather/scatter evaluators are written once for each evaluation type, are the only place in the code where a significant amount of new code must be written each time a derived quantity is desired by the analysis algorithms. These are written independently of the equations being solved and are used for all problems. Thus, this approach effectively orthogonalizes the tasks of developing new multiphysics simulations from the tasks of incorporating new nonlinear analysis methodologies. A full description of the Template Based Generic Programming approach can be found in this pair of publications [Pawlowski et al. 2012a; Pawlowski et al. 2012b].

3.3.1. Boundary Conditions. Correctly and efficiently applying boundary conditions in multiphysics simulations over complex geometries/domains is also a significant source of software complexity. Currently, Albany supports simple Dirichlet conditions as well as a growing list of Neumann-type boundary condition types such as scalar flux conditions normal to boundaries, Robin conditions, and various traction and pressure boundary conditions. Dirichlet conditions are applied in the strong form directly to global linear algebra objects produced for each evaluation type after the volumetric finite element assembly by replacing the finite element residual equation for the corresponding nodes with a Dirichlet residual equation. For example, the Dirichlet condition $u(x) = a$ for $x \in \partial\Omega_D$ is implemented by replacing the finite element residual values corresponding to degrees-of-freedom associated with $\partial\Omega_D$ with $\mathbf{u} - a$. Thus enforcement of the boundary condition is left to the nonlinear solver. We have found this approach is effective for nonlinear analysis problems such as sensitivity analysis,

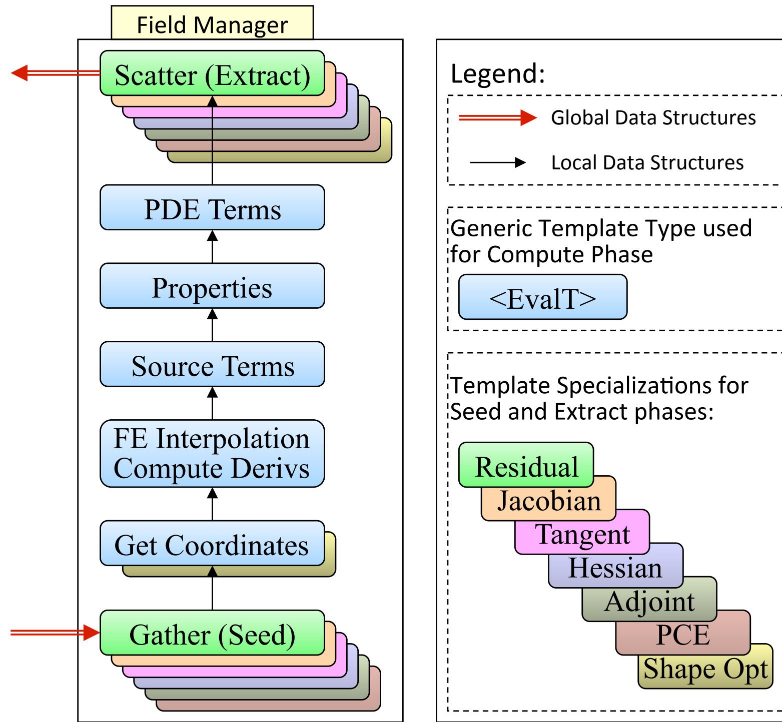

Fig. 5. The PDE assembly in Albany relies on the template-based generic programming (TBGP) approach and a graph-based assembly of individual `evaluators`. With the TBGP approach, the developer must just program the residual equations and identify design parameters. The TBGP infrastructure and automatic differentiation libraries in Trilinos will automatically compute the Jacobian matrix and direct sensitivities. The graph-based approach simplifies implementation of new models and allows for broad reuse between applications.

continuation/bifurcation analysis, optimization, and uncertainty quantification when the boundary condition must be a parameter in the problem as it allows for straightforward computation of derivatives with respect to the boundary condition, but with little additional cost in solver complexity.

The Neumann BC implementation depends on a separate finite element assembly that performs the FEM surface integrals over the designated boundaries $\partial\Omega_N$. The form of these conditions can vary significantly, some examples supported by Albany include:

(1) Flux conditions for scalar equations, such as the heat equation. For this case, one typically wishes to specify a heat flux through a surface $\partial\Omega_N$,

$$\frac{\partial T}{\partial \mathbf{n}}(x) = q(x), \tag{6}$$

for $x \in \partial\Omega_N$, where $\mathbf{n}$ is the unit normal to the boundary $\partial\Omega_N$ and $q(x)$ is the specified flux.

(2) Prescribed tractions on the boundary of mechanics problems,

$$\mathbf{t} = \sigma\mathbf{n} = \bar{\mathbf{t}}, \tag{7}$$

on $\partial\Omega_N$; σ is the Cauchy stress tensor, $\mathbf{t}$ is the traction vector on the boundary, and $\bar{\mathbf{t}}$ are the specified traction components. Pressure boundary conditions are a special case of traction $\bar{\mathbf{t}} = -p\mathbf{n}$, where p is the fluid pressure.

(3) A Robin condition is a mixed condition taking the form of a weighted combination of both Dirichlet and Neumann conditions,

$$au(x) + b\frac{\partial u(x)}{\partial \mathbf{n}} = h(x), \tag{8}$$

on $\partial\Omega_R$, where $h(x)$ is the boundary function or constraint being applied, and a and b are weights. These types of conditions are often called *impedance boundary conditions* in electromagnetic problems and *insulating boundary conditions* in convection-diffusion problems where one specifies that the convective and diffusive fluxes sum to zero $h(x) = 0$, $\forall x \in \partial\Omega_R$.

In the case of Neumann conditions, the field manager accesses surface and boundary element information from the abstract discretization interface, and Albany performs a finite element integration and assembly process over each boundary $\partial\Omega_N$ defined. Similar to the element integration process employed elsewhere in Albany, the Intrepid package is used to integrate the weak form of one of the above expressions over the portion of each element (the *element side*) that lies on the boundary. The contribution of the Neumann integral term for all evaluation type (residual, Jacobian, etc.) is computed using the same TBGP infrastructure as the volumetric terms.

3.3.2. Responses (Quantities of Interest). An implication of supporting embedded nonlinear analysis such as embedded optimization is post-processing of simulation solution values must now be handled by the simulation code, and furthermore, not only must the quantities of interest themselves be computed but also derived quantities such as response gradients. Thus Albany supports a growing list of response functions that employ the TBGP framework to simplify the evaluation of these quantities. All response functions implement a simple interface that abstracts evaluation of the response function and corresponding derivatives, and simple response functions such as the solution at a point implement this interface directly. Many response functions however can be written as an integral of a functional of the solution over some or all of the computational domain. These response functions employ the field manager described above and implement the functional as evaluators applied to the corresponding sequence of mesh cells. Generally this works just as the finite element assembly process described above, however with the additional wrinkle that response values/derivatives must be reduced across processors when run in parallel. To handle this, the response values for each evaluation type are reduced across processors before being extracted into their corresponding global linear algebra data structures using the template interface to MPI provided by the Teuchos [Thornquist et al. 2013] package in Trilinos.

3.4. Nonlinear Model Abstraction and Libraries

The 'Nonlinear Model' abstraction in Figure 3 is a Trilinos class called the `EpetraExt::ModelEvaluator`, which we will hereafter refer to as the ModelEvaluator. More complete documentation of this class and associated functionality is given in a technical report [Belcourt et al. 2011]. Albany satisfies this interface, making available all the embedded nonlinear analysis solution methods in Trilinos.

The purpose of the ModelEvaluator is to facilitate the development and usability of sophisticated, general purpose solution and analysis algorithms such as those listed as 'Solvers' in Figure 6. For instance, a general purpose nonlinear solver needs an interface to ask the application code to compute a residual vector f as a function of a

solution vector u in order to solve the nonlinear algebraic system,

$$f(u) = 0. \tag{9}$$

To fully perform a Newton solution process, the solver needs to query the application for other quantities, such as a Jacobian matrix or an approximation to the Jacobian for use in generating a preconditioner. By using a standard interface, sophisticated solution algorithms can be written that are agnostic to the physics, model, and data structures needed to support the above matrix and vector abstractions. This satisfies the component-based application code design philosophy of making different parts of the development effort (in this example, PDE description and implementation of nonlinear solution algorithms) essentially orthogonal to each other.

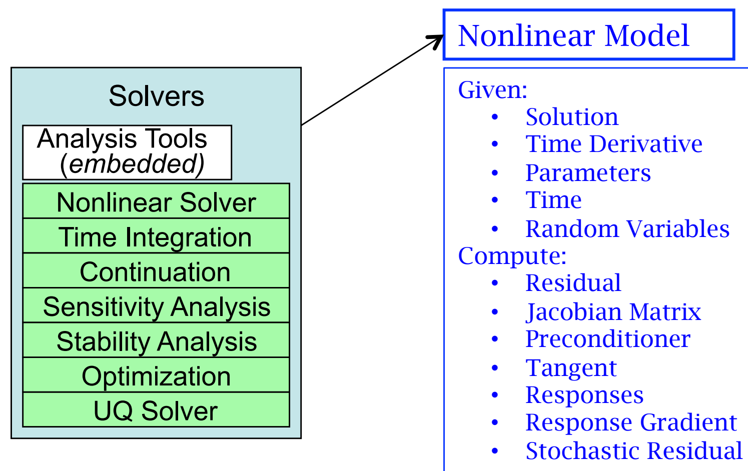

Fig. 6. Access to the embedded solvers in Trilinos requires that Albany satisfy the Nonlinear Model abstraction. In its simplest usage, this abstraction is used to compute the nonlinear residual $f(u)$. The interface is general to accommodate the computation of Jacobian operators, user-defined preconditioners, and up to and including stochastic Galerkin expansions.

Beyond this nonlinear solver example, the ModelEvaluator provides a flexible extensible interface to the application code for the analysis algorithms. As time dependent, continuation, sensitivity analysis, stability analysis, optimization, and uncertainty quantification capabilities are desired, the interface requirements grow to involve dependence on not just the solution vector u but also the time derivative $\dot{u}$, a set of parameters p, and the time t. Outputs of the interface includes sensitivities $\frac{df}{dp}$ as well as responses (*a.k.a.,* quantities of interest) and response gradients with respect to u and p.

This interface definition needs to keep pace with the leading edge of algorithmic research and development. The design has been extended to support the ability to take polynomial expansions of stochastic random variables as inputs to return polynomial representations of the output quantities including the residual vector, Jacobian matrix, and responses.

In Albany we have implemented a comprehensive set of these quantities to make use of the capabilities of the 'Solvers' in Figure 6. This single interface is all that is needed by the Trilinos Piro package. At run time, Piro will then select the desired

solver method in the NOX, LOCA, Rythmos, or Stokhos package, and subsequent performs any requested sensitivity analysis.

3.5. Linear Solver Abstraction and Libraries

The next algorithmic area isolated by an abstract interface is the linear solver. The use of libraries and common interfaces for linear solves is more common and mature than the other areas of a finite element code. In Albany, there is no need to interface directly to the linear solvers, as such solves occur as inner iterations of the nonlinear, transient, continuation, optimization, and UQ solvers that were presented in the previous section.

For linear solves, there are a wide assortment of direct and iterative algorithms, and the iterative methods can make use of a variety of algebraic, multi-level, and block preconditioners. Furthermore, these algorithms can be called in a diversity of ways with different algorithms being used on various blocks of the matrix, on various levels of the multi-level method, and isolated to sub-domains of various sizes.

Much of this flexibility is configurable at run time through the use of the Trilinos Stratimikos package, the linear solver strategies interface. The Stratimikos package “wraps” the numerous linear algebra objects, solvers, and preconditioners found in Trilinos using a common abstraction (Thyra), and the approach supports the use of a factory pattern to create the desired linear solver object. The object is fully configurable at run time using parameters given in the Albany input file. In Trilinos, this involves the Ifpack, ML, Amesos, and Teko preconditioning packages and the AztecOO, Belos, and Amesos solver packages.

In Albany, the majority of the regression test problems employ a GMRES iterative solver that in turn uses either ILU or multi-level preconditioning. There are examples of how to use block methods and matrix-free solution approaches may be selected, also at run time from the input file.

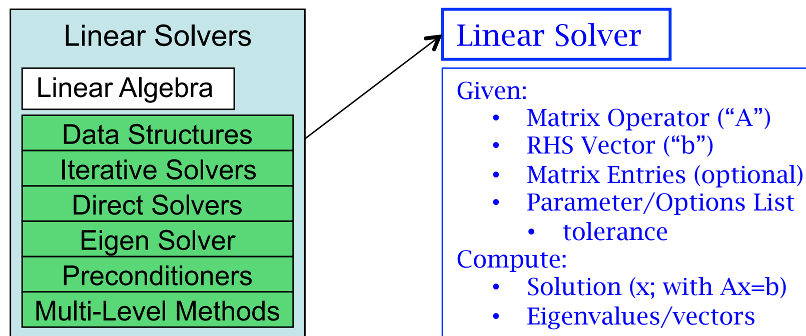

Fig. 7. The linear solver abstraction provides full access to all the linear solvers and preconditioners in Trilinos. A factory class supports run-time solution configuration through input file options.

3.6. Analysis Tools Abstraction and Libraries

Present at the top level of the software stack are the analysis tools. These tools may be used to perform a single forward solve, sensitivity analysis, parameter studies, optimization, and uncertainty quantification (UQ) runs. The analysis tools have a common

abstract behavior in that they involve repeated calls to the solvers in Section 3.4 to determine how the solution changes as a function of the parameter.

The common abstraction layer that the analysis tools conform to (and that is implemented by the 'Solvers'), takes parameter values as input and returns responses and (optionally) response gradients with respect to the parameters as output. The analysis abstraction interface does not contain references to solution vectors or PDE residuals, as analysis at this level operates along the manifold of the equations being solved.

The analysis abstraction layer is contained within the Trilinos Piro (Parameters In Responses Out) package. However, the majority of the specific analysis functionality actually resides within the Dakota framework. This is a mature, widely-used, and actively developed software framework that provides a common interface to a number of analysis, optimization, and UQ algorithms. Dakota optimization capabilities include gradient-based local algorithms, pattern searches, and genetic algorithms. Available UQ algorithms range from Latin hypercube stochastic sampling to stochastic collocation methods for polynomial chaos approaches. Dakota can be run as a separate executable that repeatedly launches a given application code, using scripts to modify parameters in input files. In Albany, Dakota is used in library mode through an abstract interface. A small Trilinos package called TriKota provides adapters between the Dakota and Trilinos analysis abstraction classes.

In Albany, a software stack is available to provide analytic gradients to the analysis tools. The parameters in the PDEs are exposed so that automatic differentiation can be employed to compute sensitivities of the residual with respect to the parameters. Likewise, the response functions use automatic differentiation to compute gradients with respect to the solution vector and parameters. The 'Solvers' then use this information, along with the system Jacobian to compute the gradient of responses with respect to responses along the manifold of the PDEs being solved, *analytically*. Currently, Hessian information is not computed, although much of the infrastructure exists to do so.

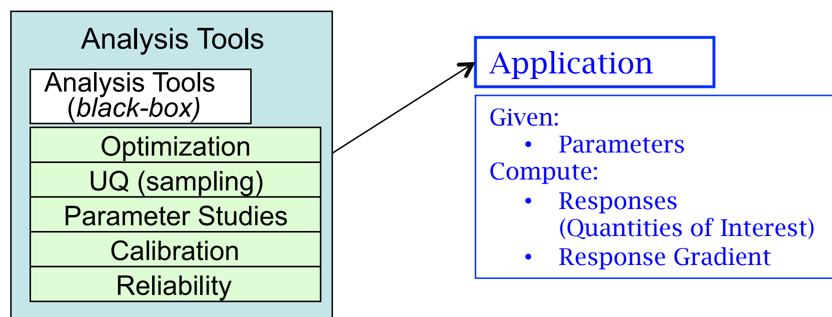

Fig. 8. At the top of the Albany computational hierarchy is the Analysis layer, where unconstrained optimization and UQ may be performed around the embedded nonlinear analysis solver layer. The interface accepts design parameters and returns responses (*a.k.a.,* quantities of interest or objective functions) and response gradients. The embedded solvers are wrapped to satisfy this interface and may be driven by the 'Analysis Tools.'

4. ALBANY APPLICATIONS

Albany's general discretization interface together with the use of a templated physics residual abstraction makes it quite suitable to host a wide variety of applications. Fur-

thermore, it is straightforward to rapidly implement new applications, which is best demonstrated by the number of different examples contained within the regression test suite and the number of analysis applications using Albany on a regular basis. The regression test suite contains simple to moderately complex problems representing a broad spectrum of phenomena, including:

— Computational mechanics: elasticity, J2 plasticity, thermomechanics, unsaturated poroelasticity, thermo-poro-mechanics, diffusion-deformation, reactor fuel cladding hydride reorientation, gradient damage, rate independent hardening minus recovery (RIHMR)
— Fluid mechanics: compressible Navier-Stokes, ice sheet flows, prototype nuclear reactor model, vortex shedding, Rayleigh-Bernard
— Diffusion, miscellaneous: Heat equation, Poisson, Schrodinger, Cahn Hilliard / Elasticity, Poisson-Nernst-Planck

In addition, the type of solution and analysis performed on these applications covers a broad spectrum:

— steady, transient, continuation / load stepping, embedded Stochastic-Galerkin, sensitivity analysis, stability analysis, uncertainty propagation

There are several analysis projects and simulation activities that have adopted Albany. Albany is the code base for an new Ice Sheet project based on a nonlinear Stokes equation. It is being used to extend and mature mesh quality enhancement techniques based on the Laplace Beltrami equations [Hansen et al. 2005] for ultimate use in arbitrary Lagrangian Eulerian (ALE) analysis codes and to model the behavior of hydrides of Zircaloy in used nuclear reactor fuel during transport and handling operations [Chen et al. 2013].

Each physics set can be turned on or off during the configuration step of the build process. All applications run from the same executable, where the physics set is selected at the top of the input file. There is however a separate executable for invoking Stochastic-Galerkin solves then for deterministic solves.

In the remainder of this section, we highlight the two most mature applications hosted in Albany. The purpose of these anecdotes is to provide evidence towards the themes of this paper. Specifically, that new applications can be rapidly written using established libraries, software tools, and interfaces, and be born with embedded analysis algorithms, robust solvers, and scalable linear solves.

4.1. Laboratory for Computational Mechanics

The Laboratory for Computational Mechanics (LCM) project adopted the Albany code base as a research platform to study issues in fracture and failure of solid materials and multi-physics coupling. These mechanics issues were effectively and efficiently introduced due in large part to object-oriented design and abstract interfaces. At present, these capabilities include quasi-static solution of the balance of linear momentum with various constitutive models in the small strain regime, as well as a total-Lagrange, finite deformation formulation. Since many problems of interest involve multiple physical phenomena, various coupled physics systems have been implemented in a monolithic fashion. Taking advantage of the template-based generic programming paradigm, and relying on graph based assembly, as well as automatic differentiation, analytic sensitivities are assembled for optimally convergent Newton iterations, regardless of how many physical governing equations are involved in the system residual.

Abstractions in the code base permit virtual isolation for the application specific physics developer. Implementation of a physical quantity, such as the strain tensor, re-

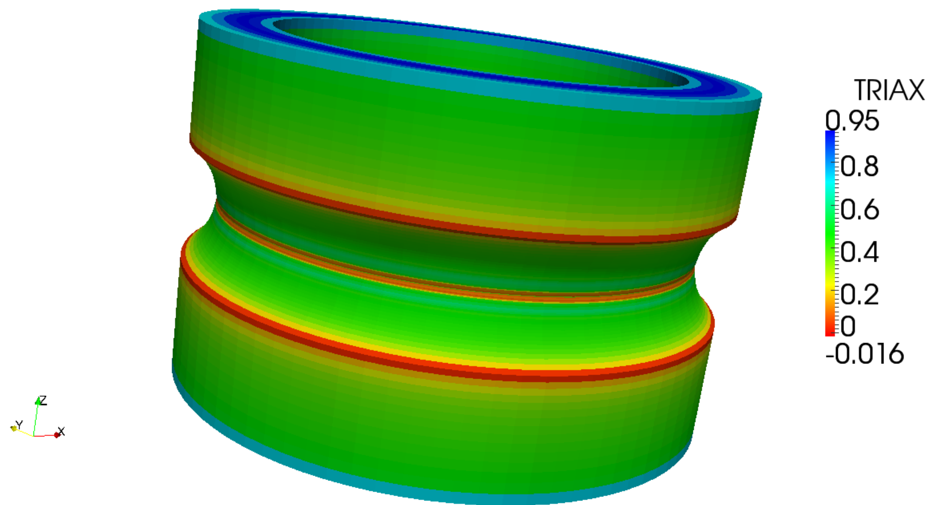

Fig. 9. Contours of stress triaxiality for a notched tube in a state of combined tension and torsional loading computed with a Gurson type model.

quires virtually no knowledge of the underlying infrastructure or data structures. As a result, domain specific expertise in writing constitutive models can be leveraged in an efficient way. To that end a number of constitutive models are available in Albany that span simple elastic behavior at small strain, through three-invariant models for geomaterials, and including finite deformation, temperature dependent metal plasticity models.

Specific implementation of constitutive models is greatly aided by the use of automatic differentiation, available from the Trilinos Sacado package. Constitutive response oftens requires the solution of a set of nonlinear equations that govern the evolution of the internal state variables local to the integration point. The system of equations typically becomes more difficult to solve as the physical fidelity of the model increases. Efficient solution of the local set of equations is often achieved employing an implicit, backward Euler integration scheme, solved using a Newton-Raphson iterative scheme, and requiring formulation and construction of the local system Jacobian for optimal convergence. Implementation of the local system of equations using automatic differentiation types has two significant advantages. The first is that the computed local Jacobian provides analytic sensitivities for the Newton iteration, resulting in optimal local convergence. The second advantage is that model changes do not require the re-derivation and re-implemenation of the local Jacobian, saving substantial development time that can instead be spent on model verification and evaluation. An example calculation using a constitutive model that employs this strategy can be seen in Figure 9, where a Gurson type constitutive model with a set of 4 local independent variables is solved at each integration point.

The existence of the load stepping capability, available through the continuation algorithms contained in the Trilinos Library of Continuation Algorithms (LOCA) package, allows for the solution of boundary value problems with nonlinearities in both the material and geometric sense. In addition, the stepping parameter can be adaptively selected based on characteristics of the current solution. For example, this adaptive

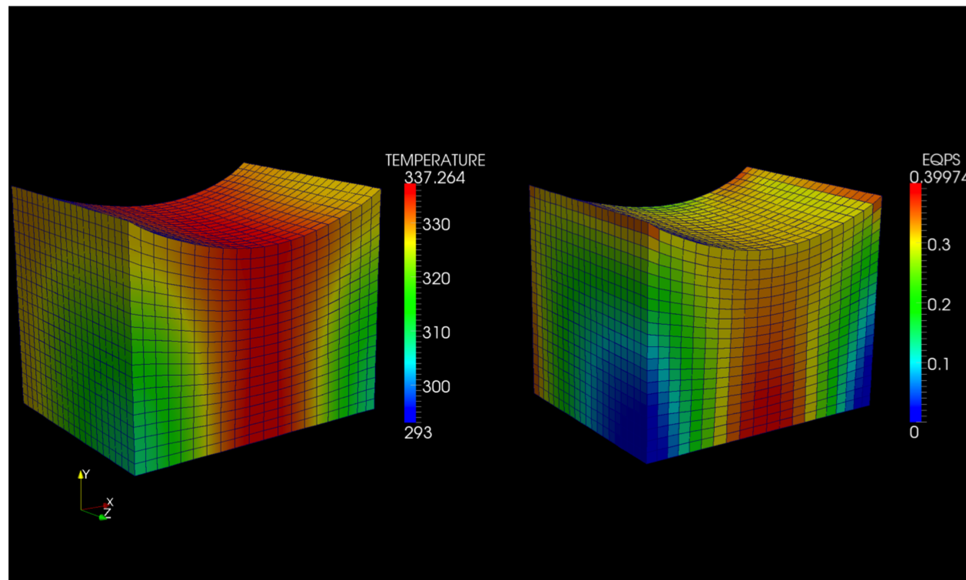

Fig. 10. Cubic domain fully clamped on x faces to eliminate contraction and given a prescribed displacement on top and insulated boundaries. Resultant temperature field stems solely from mechanical source terms.

step refinement is essential for the robust solution of problems experiencing a plastic localization, where convergence is difficult to achieve and smaller continuation steps are required. Mechanics development can leverage this adaptive stepping capability without the need for domain expertise in its formulation and implementation, providing great value for the mechanics researcher.

From the persepective of the LCM application team, a strength of Albany is the ease in which coupled systems of PDEs can be implemented. This team has formulated, implemented, and demonstrated several coupled physics problems including thermo-mechanics, hydrogen diffusion-mechanics, and poro-mechanics. Each of these physics sets was implemented in a fully coupled sense and solved in a monolithic fashion with analytic Jacobian sensitivities provided by the automatic differentiation of the system residual. In particular, the graph based assembly can explicitly show dependencies and can be a tremendous aid during model development and debugging. Example results from the thermo-mechanics problem can be seen in Figure 10. A demonstration of the poro-mechanics capabilities, outlined in [Sun et al. 2013], and applied to a geomechanical footing problem can be seen in Figure 11.

Another strength of the Albany system design becomes apparent when considering the scalability of solving the resulting linear systems. The ability to explore the use of massively parallel solvers and scalable multi-grid preconditioners, such as that provided by the Trilinos ML package, makes Albany a desirable open source research environment. The general interface to the ML preconditioner involves obtaining mesh coodinate information from the abstract discretization interface which supplies information about the rigid body modes (the null space characteristics) of the system. Currently, Albany supports computing the number of rigid body modes both with and without the presence of other coupled solution fields, and the scalability of the preconditioner has been established up to many millions of degrees of freedom.

In summary, the design of Albany has allowed for the rapid implementation of the fundamental computational mechanics infrastructure, paving the way for research ef-

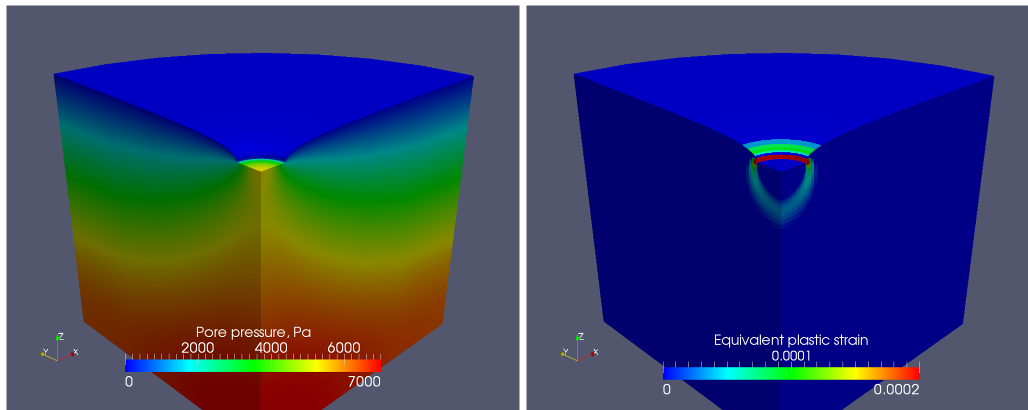

Fig. 11. Contours of pore pressure and equivalent plastic strain for a cylindrical footing.

forts into new methods and models. The open source nature of the code base serves as a foundation for academic collaboration. Successful research ideas are targeted for transition into Sandia's internal production analysis codes.

4.2. Quantum Computer Aided Design

The quantum computer aided design (QCAD) project uses Albany to develop a simulation and design capability for the electronic structure of laterally-gated quantum dots, to determine their utility as qubits in quantum computing devices. Such a task is a subset of semiconductor device simulation. In this case, we are targeting a regime not well covered by previous tools, specifically low-temperature operation close to absolute zero Kelvin, and few- or one-electron devices. Albany was chosen because it provided access to the many finite element, solver, and analysis libraries, and a programming model that enabled us to efficiently implement several physics sets that our application presents.

Quantum dots are regions in a semiconductor where the local electrostatics allows "puddles" of electrons to form, typically near a semiconductor-insulator interface. We often use a silicon metal-oxide-semiconductor (MOS) system, with an additional level of gates in the insulator to deplete the sheet into puddles that form quantum dots, as shown in Figure 12. The depletion gates themselves in experimental quantum dots can have many different and complex three-dimensional (3D) geometries. Figure 13 shows three examples of typical depletion gate patterns in a top view. The quantum effects we wish to use to form qubits are most pronounced with few numbers of electrons, and a major challenge is to design robust enough structures that allow to form few-electron dots. This often involves modifying the shapes of the gates and the spacings between different layers.

The gate voltages dictate Dirichlet boundary conditions along the surfaces of the regions that form the gates. We have developed and validated three major solvers of increasing computational complexity. The Nonlinear Poisson solver determines the electrostatic potential profile that results from the gate voltages and other device parameters in a given device by treating electrons semi-classically, that is, as classical particles that obey quantum (Fermi-Dirac) statistics. The simplest formulation facilitates rapid simulations of many designs, which enables fast semi-classical understanding of device behavior and hence rapid feedback on device designs. The Schrodinger-Poisson (S-P) solver is a multi-physics model which couples the nonlinear Poisson solver and a Schrodinger solver in a self-consistent manner to capture quantum effects in our de-

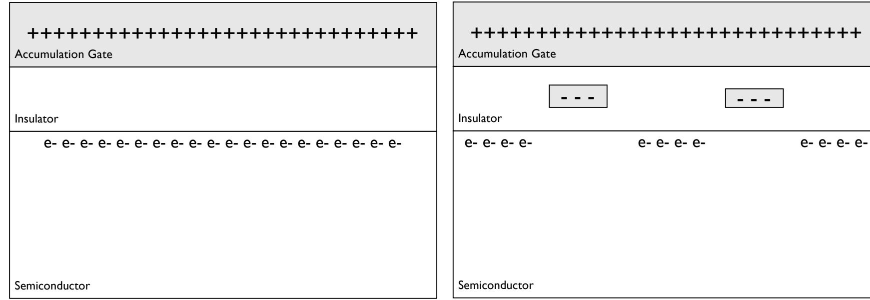

Fig. 12. Cross-section view of a simplified quantum dot device to illustrate the concept. We can form sheets ("e-") of electrons at a MOS interface using an accumulation gate with a positive ("+") voltage (left figure). By introducing additional depletion gates with negative ("-") voltage, we can deplete most of this sheet, leaving puddles that form quantum dots (right figure).

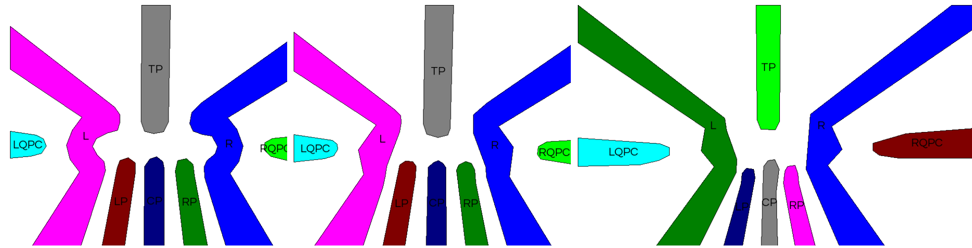

Fig. 13. Examples of typical depletion gate patterns in experimental quantum dot devices in a top view. Each color in the left, middle, and right figures indicates a metal or polysilicon gate that can be set to a different voltage to form a quantum dot.

vices. Finally, the Configuration Interaction solver takes single-particle solutions from the S-P solver and determines multi-electron solutions that include quantum interactions between electrons.

The Albany framework has made it straightforward and fast to implement these QCAD solvers. The general Poisson equation is written as

$$\nabla(\epsilon_s \nabla\phi) = \rho(\phi), \tag{10}$$

where ϕ is the electrostatic potential to be solved for and $\rho(\phi)$ can be a nonlinear function. The corresponding finite element weak form (leaving out the surface term for this presentation)

$$\int \epsilon_s \nabla\phi \cdot \nabla w d\Omega + \int \rho(\phi) w d\Omega = 0, \tag{11}$$

with w being the nodal basis function and the LHS being defined as residual. To solve the equation in the Albany framework, we created a concrete `QCAD::PoissonProblem` class derived from `Albany::AbstractProblem`, in which we constructed the residual by evaluating and putting together each term. The static permittivity ϵ_s and the source $\rho(\phi)$ are evaluated in separate QCAD-specific `evaluators`, while the integrations are done by general-purpose Albany `evaluators`. The automatic differentiation (AD) capability, parallelization, and nonlinear and linear solvers were available without any development effort for the QCAD projects physics sets. Through parallelism, robustness, and automation enabled by analysis algorithms, the throughput of quantum dot

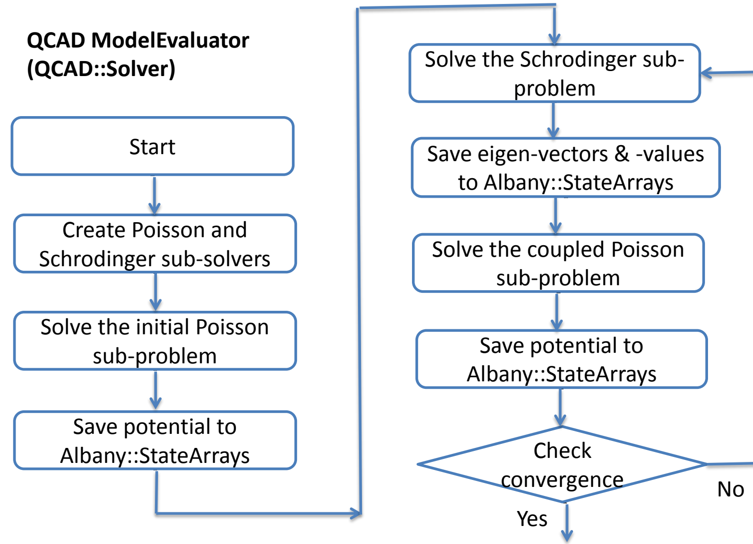

Fig. 14. Schematic diagram showing the Schrodinger-Poisson implementation in QCAD

simulations increased several orders of magnitude over the previous simulation process that was being employed.

The Schrodinger-Poisson (S-P) solver self-consistently couples the nonlinear Poisson solver, above, with a Schrodinger eigensolver. The latter solves a single-particle effective mass Schrodinger equation

$$-\frac{\hbar^2}{2}\nabla(\frac{1}{m^*}\nabla\psi) + V(\phi)\psi = E\psi. \tag{12}$$

The weak form of this equation was implemented similar to the implementation of the nonlinear Poisson solver. The Trilinos eigensolver Anasazi is used to approximate the leading modes of the discretized eigenproblem after undergoing a spectral transformation, using infrastructure originally developed for stability analysis [Lehoucq and Salinger 2001]. The self-consistent loop is done in an aggregate ModelEvaluator, which splits the S-P problem into Schrodinger and Poisson sub-problems and calls the corresponding solve to solve each, as illustrated in Figure 14. The iteration is continued until a pre-defined convergence criterion is satisfied.

A large part of the code development for the QCAD project is to compute application-specific response functions. We have coded several responses for our quantum devices, including average value and integral of a field in a given region. One particular response that has been crucial for our devices is finding the saddle path between two minima. The saddle path searching algorithm is fairly complicated and it was relatively easy to fit into the Albany response framework. The AD capability is critical for computation of gradients of responses.

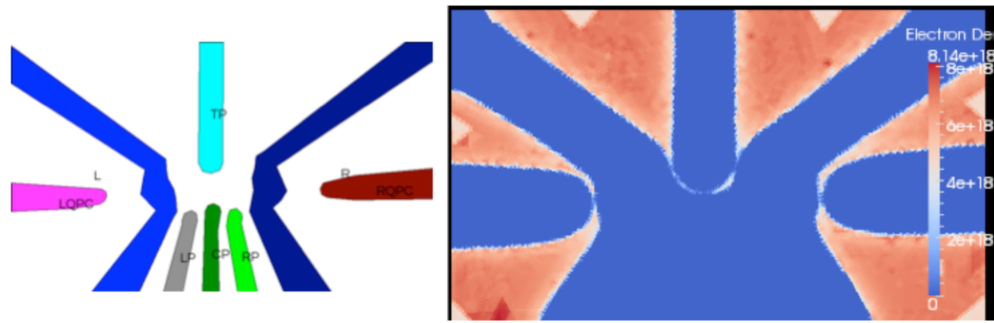

Fig. 15. Optimization of the Ottawa Flat 270 structure. The left figure shows a top view of the depletion gate configuration for the structure, and the right figure shows the resulting electron density after optimization was achieved, with a variety of constraints detailed in the text.

Another key element to the process of developing the QCAD code was the additional packages integrated in the Albany workflow. In particular, the Cubit mesh generator [Hanks et al. 2013] and the Dakota optimization & UQ package [Adams et al. 2009]. Albany supports a variety of finite element topologies such as quad and tri in 2D, hex and tet in 3D. The code is written for arbitrary nodal discretization order, though only linear and quadratic basis functions have been accessed. The code can import the meshed from the ExodusII [Sjaardema et al. 2013] format, which is generated by Cubit. This capability allows us to use Cubit to create highly non-uniform 3D tet meshes, since our structures often have complex 3D shapes as shown in Figure 13. The Dakota package available to Albany via the TriKota interface provides various optimization options that have been extremely useful in optimizing complicated targets for our devices.

An example of the type of optimization we performed is given in Figure 15. We wished to optimize a quantum dot to contain exactly two electrons, with tunable tunnel barriers in and out of the dot region, between the left and right electrons of the dot, and with the channels on the sides also having tunable tunnel barriers. The voltages on all gates (shown from a top view on the lefts side of Figure 15) are allowed to vary as design parameters, with the left/right symmetry in the gate voltages imposed as a constraint. The right side of Figure 15 shows the resulting electron density after Dakota found the optimal voltages that satisfied all the targets. This was performed by repeatedly calling the nonlinear Poisson solver for the response and analytically-computed gradients. The red region is the “sheet” of electrons, and the blue regions have few electrons and somewhat follow the shapes of the depletion gates. The quantum dot itself is the narrow curved region underneath the gate labeled TP in the left.

In summary, the numerous capabilities that Albany provides enable us to rapidly develop application-focused QCAD solvers. The resulting design tool has many more functionalities than we had proposed at the beginning of the project. As a result, QCAD simulations have become an integral part of the *experimental* effort in silicon qubit design.

5. CONCLUSIONS

In this paper, we have articulated a strategy for the construction of computational science applications that promotes the use of reusable software libraries, abstract interfaces, and modern software engineering tools. We believe the success of the component-based approach should be expected, for many of the reasons articulated, and by extrapolating on the broadly successful use of linear algebra libraries. It remains common,

however, for application codes to make limited use of external libraries. Some of the reasons include the learning curve of using (and debugging) someone else's code, difficulties in maintaining compatible versions and in porting, and the challenges with interfacing a collection of different libraries.

Many of these issues have been overcome in the Trilinos suite. The libraries built in Trilinos share a common build system and release schedule. Where possible, independent capabilities that should work together, like a nonlinear solver inside of an implicit time integrator, provide a general interface. Also, many capabilities that are typically used in a similar way, such as linear solvers and embedded nonlinear analysis tools, can be called with the same interface and selected at run time through a factory pattern.

We have built the Albany finite element code attempting to follow, and test the efficacy of, the component-based strategy, and making use of the broad set of computational capabilities in Trilinos. In Section 3 we provided an overview of the software design and abstractions important in the development of Albany, an extensible generic unstructured-grid, implicit, finite element application code. The design is modularized with abstract interfaces, where we have shown that we can independently swap out physics sets, mesh databases, linear solvers, nonlinear solvers, and analysis tools. Dozens of independently-developed Trilinos libraries contribute to the code capabilities. The bulk of the code base involves the application-specific description of the PDE equations and response functions.

The evidence presented on the success of this approach and our implementation comes from two applications that have been built in the Albany code base, and were presented in Section 4. The feedback from these development efforts is that it is straightforward to rapidly develop sophisticated PDE codes with excellent parallelism, advanced discretizations, high performance linear solvers and preconditioners, a wide range of nonlinear and transient solvers, and sophisticated analysis algorithms, using the proposed methodology. The LCM code has been able to very naturally explore fully-coupled solution algorithms for mechanics coupled with additional scalar equations. By writing tensor operations in an independent library that is templated to allow for automatic differentiation data types, one may quickly investigate new models. In less than 2 years of effort, the QCAD project was able to improve their throughput by several orders of magnitude, leading to a new workflow where tentative design is thoroughly investigated by a running a suite of optimization runs on a high-fidelity model, instead of manually launching a handful of forward simulations. Using this capability, the project has been successful in injecting computational analysis into the design cycle used by experimentalists.

ACKNOWLEDGMENTS

The Albany code builds upon numerous computational science capabilities, and we would like to acknowledge the contributions of all the authors of these libraries and tools. There are several who directly impacted the component-based code design strategy and the Albany code base, including Mike Heroux, Jim Willenbring, Brent Perschbacher, Pavel Bochev, Denis Ridzal, Carter Edwards, Greg Sjaardema, Eric Cyr, Julien Cortial, Brian Adams, and Mike Eldred. In addition, this effort has had significant management support, including that of David Womble, Scott Collis, Rob Hoekstra, Ken Alvin, John Aidun, and Eliot Fang.

This work was funded by the US Department of Energy through the NNSA Advanced Scientific Computing (ASC) and Office of Science Advanced Scientific Computing Research (ASCR) programs, and the Sandia Laboratory Directed Research and Development (LDRD) program.

REFERENCES

B.M. Adams, W.J. Bohnhoff, K.R. Dalbey, J.P. Eddy, M.S. Eldred, D.M. Gay, K. Haskell, P.D. Hough, and L.P. Swiler. 2009. *DAKOTA, A Multilevel Parallel Object-Oriented Framework for Design Optimization,*

Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 5.0 User's Manual. Technical Report SAND2010-2183. Sandia National Laboratories. Updated December 2010 (Version 5.1) Updated November 2011 (Version 5.2) Updated February 2013 (Version 5.3).

S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith, and H. Zhang. 2013. *PETSc Users Manual*. Technical Report ANL-95/11 - Revision 3.4. Argonne National Laboratory.

W. Bangerth, R. Hartmann, and G. Kanschat. 2007. deal.II – A general-purpose object-oriented finite element library. *ACM Trans. Math. Softw.* 33, 4, Article 24 (Aug. 2007). DOI:http://dx.doi.org/10.1145/1268776.1268779

R. A. Bartlett, M. A. Heroux, and J. M. Willenbring. 2012. *TriBITS Lifecycle Model*. SAND Report SAND2012-0561. Sandia National Laboratories.

N. Belcourt, R. P. Pawlowski, R. A. Bartlett, R. W. Hooper, and R. C. Schmidt. 2011. *A Theory Manual for Multi-physics Code Coupling in LIME*. Technical Report SAND2011-2195. Sandia National Laboratories.

P. Bochev, H.C. Edwards, R. Kirby, K. Peterson, and D. Ridzal. 2012. Solving PDEs with Intrepid. *Scientific Programming* 20, 2 (2012), 151–180.

Q. Chen, J. T. Ostien, and G. Hansen. 2013. Development of a Used Fuel Cladding Damage Model Incorporating Circumferential and Radial Hydride Responses. *J. Nucl. Mater.* (2013). Submitted.

J. D. de St. Germain, J. McCorquodale, S.G. Parker, and C.R. Johnson. 2000. Uintah: A Massively Parallel Problem Solving Environment. In *Ninth IEEE International Symposium on High Performance and Distributed Computing*. IEEE, 33–41. http://software.sci.utah.edu/uintah.html

L Diachin, A Bauer, B Fix, J Kraftcheck, K Jansen, X Luo, M Miller, C Ollivier-Gooch, M S Shephard, T Tautges, and H Trease. 2007. Interoperable mesh and geometry tools for advanced petascale simulations. *Journal of Physics: Conference Series* 78, 1 (2007), 012015. http://stacks.iop.org/1742-6596/78/i=1/a=012015

H. C. Edwards, A. B. Williams, G. D. Sjaardema, D. G. Baur, and W. K. Cochran. 2010. *SIERRA Toolkit Computational Mesh Conceptual Model*. Technical Report SAND2010-1192. Sandia National Laboratories.

D. Gaston, C. Newman, G. Hansen, and D. Lebrun-Grandie. 2009. MOOSE: A parallel computational framework for coupled systems of nonlinear equations. *Nuclear Engineering and Design* 239, 10 (2009), 1768–1778.

B. Hanks and others. 2013. http://cubit.sandia.gov/. (2013).

G. Hansen, A. Zardecki, D. Greening, and R. Bos. 2005. A Finite Element Method for Three-Dimensional Unstructured Grid Smoothing. *J. Comput. Phys.* 202, 1 (2005), 281–297.

M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. B. Williams, and K. S. Stanley. 2005. An Overview of the Trilinos Project. *ACM Trans. Math. Softw.* 31, 3 (2005). http://trilinos.sandia.gov/.

B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. 2006. `libMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering with Computers* 22, 3–4 (2006), 237–254. http://dx.doi.org/10.1007/s00366-006-0049-3.

R.B. Lehoucq and A.G. Salinger. 2001. Large-Scale Eigenvalue Calculations for Stability Analysis of Steady Flows on Massively Parallel Computers. *International Journal of Numerical Methods in Fluids* 36 (2001), 309–327.

A. Logg, K-A Mardal, G. N. Wells, and others. 2012. *Automated Solution of Differential Equations by the Finite Element Method*. Springer. DOI:http://dx.doi.org/10.1007/978-3-642-23099-8

K. R. Long, R. C. Kirby, and B. G. van Bloemen Waanders. 2010. Unified Embedded Parallel Finite Element Computations via Software-Based Fréchet Differentiation. *SIAM J. Scientific Computing* (2010), 3323–3351.

P. K. Notz, R. P. Pawlowski, and J. C. Sutherland. 2011. Graph-based software design for managing complexity and enabling concurrency in multiphysics PDE software. *ACM Trans. Math. Softw.* (2011). Submitted.

R.P. Pawlowski. 2011. http://trilinos.sandia.gov/packages/phalanx/. (2011).

R P Pawlowski, E.T. Phipps, and A G Salinger. 2012a. Automating embedded analysis capabilities and managing software complexity in multiphysics simulation, Part I: Template-based generic programming. *Scientific Programming* 20 (2012), 197–219.

R P Pawlowski, E T Phipps, A G Salinger, S J Owen, C M Siefert, and M L Staten. 2012b. Automating embedded analysis capabilities and managing software complexity in multiphysics simulation part II: application to partial differential equations. *Scientific Programming* 20 (May 2012), 327–345.

E.T. Phipps and R.P. Pawlowski. 2012. Efficient Expression Templates for Operator Overloading-based Automatic Differentiation. In *Recent Advances in Algorithmic Differentiation*, S. Forth, P. Hovland, E.Phipps, J. Utke, and A. Walther (Eds.). Springer.

E. T. Phipps. 2013a. http://trilinos.sandia.gov/packages/sacado/. (2013).

E. T. Phipps. 2013b. http://trilinos.sandia.gov/packages/stokhos/. (2013).

C. Prud'homme. 2007. Life: Overview of a Unified C++ Implementation of the Finite and Spectral Element Methods in 1D, 2D and 3D. In *Applied Parallel Computing. State of the Art in Scientific Computing (Lecture Notes in Computer Science)*, Vol. 4699. Springer, 712–721.

A. G. Salinger. 2012. *Component-based Scientific Application Development*. Technical Report SAND2012-9339. Sandia National Laboratories.

S. Seol, C.W. Smith, D.A. Ibanez, and M.S. Shephard. 2012. A Parallel Unstructured Mesh Infrastructure. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*. 1124–1132. DOI: http://dx.doi.org/10.1109/SC.Companion.2012.135

G. Sjaardema and others. 2013. http://sourceforge.net/projects/exodusii/. (2013).

T.M. Smith, J. N. Shadid, R.P. Pawlowski, E.C. Cyr, and P. D. Weber. 2011. Reactor Core Subassembly Simulations Using a Stabilized Finite Element Method. In *The 14th International Topical Meeting on Nuclear Reactor Thermalhydraulics, NURETH-14*. Toronto, Ontario, Canada.

J. R. Stewart and H. C. Edwards. 2003. The SIERRA Framework for Developing Advanced Parallel Mechanics Applications. In *Large-Scale PDE-Constrained Optimization*, Lorenz T. Biegler, Matthias Heinkenschloss, Omar Ghattas, and Bart van Bloemen Waanders (Eds.). Lecture Notes in Computational Science and Engineering, Vol. 30. Springer Berlin Heidelberg, 301–315. DOI: http://dx.doi.org/10.1007/978-3-642-55508-4_18

W. Sun, J T Ostien, and A G Salinger. 2013. A stabilized assumed deformation gradient finite element formulation for strongly coupled poromechanical simulations at finite strain. *International Journal for Numerical and Analytical Methods in Geomechanics* (2013).

H. Thornquist and others. 2013. http://trilinos.sandia.gov/packages/teuchos/. (2013).

Received October 2013; revised ??; accepted June ??

Online Appendix to: Albany: A Component-Based Partial Differential Equation Code Built on Trilinos

ANDREW G. SALINGER, ROSCOE A. BARTLETT, QUISHI CHEN, XUJIAO GAO, GLEN A. HANSEN, IRINA KALASHNIKOVA, ALEJANDRO MOTA, RICHARD P. MULLER, ERIK NIELSEN, JAKOB T. OSTIEN, ROGER P. PAWLOWSKI, ERIC T. PHIPPS, WAICHING SUN, Sandia National Laboratories

© 2013 ACM 0098-3500/2013/10-ART?? $15.00
DOI:http://dx.doi.org/10.1145/0000000.0000000