An Overview of the Trilinos Project

MICHAEL A. HEROUX
ROSCOE A. BARTLETT
VICKI E. HOWLE
ROBERT J. HOEKSTRA
JONATHAN J. HU
TAMARA G. KOLDA
RICHARD B. LEHOUCQ
KEVIN R. LONG
ROGER P. PAWLOWSKI
ERIC T. PHIPPS
ANDREW G. SALINGER
HEIDI K. THORNQUIST
RAY S. TUMINARO
JAMES M. WILLENBRING
ALAN WILLIAMS
Sandia National Laboratories
and
KENDALL S. STANLEY
Oberlin College

The Trilinos Project is an effort to facilitate the design, development, integration and ongoing support of mathematical software libraries within an object-oriented framework for the solution of large-scale, complex multi-physics engineering and scientific problems. Trilinos addresses two fundamental issues of developing software for these problems: (i) Providing a streamlined process and set of tools for development of new algorithmic implementations and (ii) promoting interoperability of independently developed software.

Trilinos uses a two-level software structure designed around collections of *packages*. A Trilinos package is an integral unit usually developed by a small team of experts in a particular algorithms area such as algebraic preconditioners, nonlinear solvers, etc. Packages exist underneath the Trilinos top level, which provides a common look-and-feel, including configuration, documentation, licensing, and bug-tracking.

Here we present the overall Trilinos design, describing our use of abstract interfaces and default concrete implementations. We discuss the services that Trilinos provides to a prospective package and how these services are used by various packages. We also illustrate how packages can be combined to rapidly develop new algorithms. Finally, we discuss how Trilinos facilitates high-quality software engineering practices that are increasingly required from simulation software.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; G.4 [**Mathematics of Computing**]: Mathematical Software; D.2.13 [**Software Engineering**]: Reusable Software

General Terms: Algorithms, Design, Performance, Reliability

Additional Key Words and Phrases: Software framework, Interfaces, Software Quality Engineering

1. INTRODUCTION

Research efforts in advanced solution algorithms and parallel solver libraries have historically had a large impact on engineering and scientific computing. Algorithmic advances increase the range of tractable problems and reduce the cost of solving existing problems. Well-designed solver libraries provide a mechanism for leveraging solver development across a broad set of applications and minimize the cost of solver integration. Emphasis is required in both new algorithms and new software in order to maximum the impact of our efforts.

General-purpose linear and eigensolvers have been successfully used across a broad set of applications and computer systems. EISPACK [Smith et al. 1976], LINPACK [Dongarra et al. 1979] and LAPACK [Anderson et al. 1995] are just a few of the many packages that have made a tremendous impact, providing robust portable solvers to a broad set of applications. More recently packages such as PETSc [Balay et al. 1998b; 1998a; 1997], Scalapack [Blackford et al. 1997] and Aztec [Tuminaro et al. 1999] have provided a large benefit to applications by giving users access to parallel distributed memory solvers that are easy-to-use and robust.

Sandia has historically had efforts to develop scalable solver algorithms and software. Often this development has been done within the context of a specific application code, providing a good robust solver that specifically meets the needs of that application. Even Aztec, one of the most important general-purpose solvers developed at Sandia, was developed specifically for MPSalsa [Salinger et al. 1996; Shadid et al. 1995] and only later extracted for use with other applications. Unfortunately, even though application-focused solvers tend to be very robust and can often be made into very effective general-purpose solvers, the opportunity to re-use the basic set of tools developed for one solver in the development of another solver becomes very difficult.

The Trilinos Project grew out of this group of established numerical algorithms efforts at Sandia, motivated by a recognition that a modest degree of coordination among these efforts could have a large positive impact on the quality and usability of the software we produce and therefore enhance the research, development and integration of new solver algorithms into applications. With the advent of Trilinos, the degree of effort required to develop new parallel solvers has been substantially reduced, because our common infrastructure provides an good starting point. Furthermore, many applications are standardizing on the Trilinos matrix and vector classes. As a result, these applications have access to all Trilinos solver packages without interface modifications.

The Trilinos project encompasses a variety of efforts that are to some extent self-contained but at the same time inter-related. The Trilinos design allows in-

dividual packages to grow and mature autonomously to the extent the algorithms and package developers dictate. This document provides an overview of the project, focusing on the project philosophy and description, and providing the reader with a summary of the project in its current state. Integration of a package into Trilinos, and what Trilinos can provide to a package, have multiple possibilities that will be discussed in Section 2. Section 3 discusses two special collections of Trilinos packages: Petra and TSF. The general definition of a Trilinos package is presented in Section 5 An overview of current software research and development is given in Section 6. Section 7 presents the Meros package in greater detail because it illustrates how multiple Trilinos packages can be combined to quickly provide production implementations of state-of-the-art algorithms. Finally, Section 8 discusses the role of Trilinos to improve software quality and reduce the cost of software quality assurance processes, an increasingly important aspect of computer modeling and simulation for science and engineering.

2. TRILINOS DESIGN PHILOSOPHY

Each Trilinos package is a self-contained, independent piece of software with its own set of requirements, its own development team and group of users. Because of this, Trilinos itself is designed to respect the autonomy of packages. Trilinos offers a variety of ways for a particular package to interact with other Trilinos packages. It also offers a set of tools that can assist package developers with builds across multiple platforms, generating documentation and regression testing across a set of target platforms. At the same time, what a package *must* do to be called a Trilinos package is minimal, and varies with each package. The current collection of Trilinos packages is shown in Figure 1.

2.1 Services Provided by Trilinos

Trilinos provides a variety of services to a developer wanting to integrate a package into Trilinos. In particular, the following are provided:

—**Configuration management:** Autoconf [Free Software Foundation a], Automake [Free Software Foundation b] and Libtool [Free Software Foundation f] provide a robust, full-featured set of tools for building software across a broad set of platforms (see also the "Goat Book" [Vaughan et al. 2000]). Although these tools are not official standards, they are widely used. All existing Trilinos packages use Autoconf and Automake. Libtool support will be added in future releases.

New package developers who are not currently using autotools, but would like to, can get a jump start by using a Trilinos package called "new_package" (see below).

Trilinos provides a set of M4 [Free Software Foundation d] macros that can be used by any other package that wants to use Autoconf and Automake for configuring and building libraries. These macros perform common configuration tasks such as locating a valid LAPACK [Anderson et al. 1995] library, or checking for a user- defined MPI C compiler. These macros minimize the amount of redundant effort in using Autotools, and make it easier to apply a general change to the configure process for all packages.

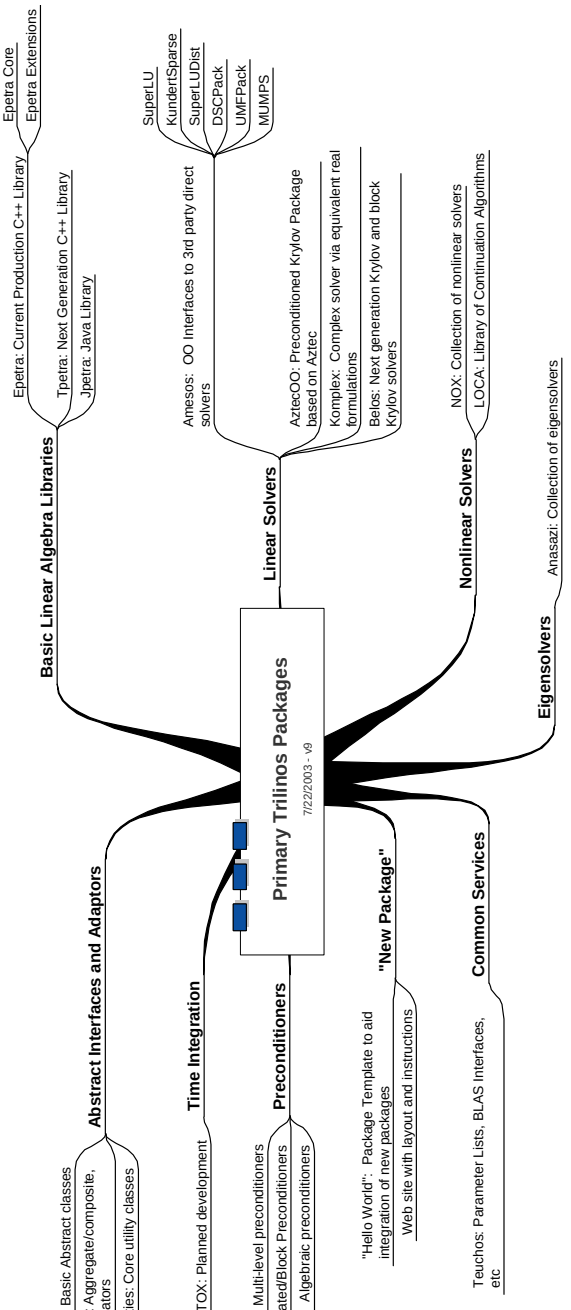

Fig. 1. Current collection of Trilinos Packages

—**Regression testing:** Trilinos provides a variety of regression testing capabilities. Although the test suite is always improving, good coverage testing is available for the major Trilinos packages. Integrating new tests into Trilinos is accomplished by creating specially named directories in the CVS repository and creating scripts that run package tests. These scripts can be executed manually and are also run as part of the automated regression test harness (see next item).

—**Automatic Testing:** Trilinos Packages that configure and build using Autotools can easily utilize the the Trilinos test harness. On a nightly basis, the test harness builds the most recent versions of Trilinos libraries and runs any tests that have been integrated into the testharness.

—**Portable interface to BLAS and LAPACK:** The Basic Linear Algebra Subprograms (BLAS) [Lawson et al. 1979; Dongarra et al. 1988; Dongarra et al. 1990] and LAPACK [Anderson et al. 1995] provide a large repository of robust, high-performance mathematical software for serial and shared memory parallel dense linear algebra computations. However, the BLAS and LAPACK interfaces are Fortran specifications, and the mechanism for calling Fortran interfaces from C and C++ varies across computing platforms. Epetra (and Teuchos) provide a set of simple, portable interfaces to the BLAS and LAPACK that provide uniform access to the BLAS and LAPACK across a broad set of platforms. These interfaces are accessible to other packages.

—**Source code repository and other software process tools:** Trilinos source code is maintained in a CVS [Free Software Foundation c] repository that is accessible via a secure connection from anywhere on the internet. It is also browsable via a web-based interface package called Bonsai [The Mozilla Organization a]. Features and bug reports are tracked using Bugzilla [The Mozilla Organization b], and email lists [Free Software Foundation e] are maintained for Trilinos as a whole and for each package. Support for new packages can easily be added. All tools are accessible from the main Trilinos website [Heroux].

—**Quick-start package infrastructure:** Via the new_package package in Trilinos, a new or existing software project can quickly adopt a variety of useful software processes and tools. new_package provides a starting point for:

  —Project organization: Illustrates one way of organizing files for a mathematical software package.

  —Autotools: As mentioned above, provides simple working example using autotools, and a set of M4 macros.

  —Automatically generated reference documentation: Shows how to mark up source code and use Doxygen [van Heesch] to produce accurate, extensive source code documentation.

  —Regression testing: Simple regression testing example is part of new_package.

  —Website: The Trilinos home page [Heroux] contains a new_package website that includes instruction on how to copy and modify the new_package web source for use with a new Trilinos package.

Note: It is worth mentioning that the Trilinos new_package package can be useful independent of Trilinos itself. Like all Trilinos packages, new_package is self-contained, and can be configured and built independently from the rest

of Trilinos. Similarly, the new_package website is self-contained and essentially independent from the rest of the Trilinos website.

3. PETRA AND TSF: TWO SPECIAL PACKAGE COLLECTIONS

In order to understand what Trilinos provides beyond infrastructure and the contributions of each Trilinos package, we briefly discuss two special collections of Trilinos packages: Petra and TSF. These two packages collections are complimentary, with TSF packages providing common abstract application programmer interfaces (APIs) for other Trilinos packages and Petra providing common concrete implementations of basic classes used by most Trilinos packages. Within the Petra collection of packages, Epetra is the most mature, portable and widely used package. Within the TSF collection, TSFCore provides a lean set of interfaces and TSFExtended provides a fuller feature set. TSFExtended builds on top of TSFCore, i. e. , TSFExtended classes inherit from TSFCore classes.

3.1 Epetra

Matrices, vectors and graphs are basic objects used in most solver algorithms. Most Trilinos packages interact with these kinds of objects via abstract interfaces that allow a package to define what services and behaviors are expected from the objects, without enforcing a specific implementation. However, in order to use these packages, some concrete implementation must be selected. Epetra (and in the future other packages described in Section 6.1) is a collection of concrete classes that supports the construction and use of vectors, sparse graphs, and dense and sparse matrices. It provides serial, parallel and distributed memory capabilities. It uses the BLAS and LAPACK where possible, and as a result has good performance characteristics.

3.2 TSFCore and TSFExtended

Many different algorithms are available to solve a given numerical problem. For example, there are many algorithms for solving a system of linear equations, and many solver packages are available to solve linear systems. Which package is appropriate is a function of many details about the problem being solved and the platform on which application is being run. However, even though there are many different solvers, conceptually, from an abstract view, these solvers are providing a similar capability, and it is advantageous to utilize this abstract view. TSF is a collection of abstract classes that provides an application programmer interface (API) to perform the most common solver operations. It can provide a single interface to many different solvers. Furthermore, TSFExtended has powerful compositional mechanisms that support the light-weight construction of composite objects from a set of existing objects (see Section 7). As a result, TSF users gain easy access to many solvers and can bring multiple solvers to bear on a single problem.

4. COMMON TOOLS PACKAGE: TEUCHOS

As the number of Trilinos packages grows, we have developed the need for a collection of tools that can be leverages across all packages. The Teuchos package is a relatively recent addition to Trilinos to facilitate collection of the common tools.

In order to retain the autonomy of other Trilinos packages, no package is required to adopt Teuchos classes. However, a design goal of Teuchos is robustness and portability such that dependency on Teuchos is not a practical liability.

Teuchos provides classes and interfaces for:

(1) Templated access to BLAS and LAPACK interfaces. Teuchos provides a set of interfaces that have a single templated parameter for the scalar field. In cases where the template is of type single, double, complex single or complex double, the user will be linked to standard BLAS and LAPACK functions. For other data types, we provide generic loops sets for a limited set of key kernels. For example, if the user specifies a dense matrix-matrix multiply operation, the standard GEMM BLAS kernel will be called for the four primary scalar types. For other data types, Teuchos provides a triple nested loop set that implements the same functionality in terms of the "+" and "*" operators and uses scalar traits to define zero and one. If the data type that user passed in supports "operator+" and "operator*" and has a well-defined concept of zero, identity and magnitude, this type of loop set with compile and execute correctly. We have used this mechanism to compute basic matrix and vector calculations in arbitrary precision arithmetic using ARPREC [Bailey et al. 2002]. This mechanism can also be used to support interval arithmetic, geometric transformation calculations, integers and calculations with many more data types. Clever use of this mechanism can be used in multiple ways to analyze and improve the robustness of numerical algorithms.

(2) Parameter lists: A parameter list is a collection of key-value pairs that can be used to communicate with a packages. A parameter can be used to tune how a package is used, or can provide information back to the user from a package. For example the pair ("Residual Tolerance", 1.0E-6) could be used to specify the tolerance that a package should use for convergence testing in an iterative process. Similarly, the pair ("Residual Norm", 9.3245E-7) can be passed back to the user as the actual computed residual norm.
Although a number of packages in Trilinos use their own implementation of parameter lists internally, all packages will be able to parse Teuchos lists. This allows users to utilize the same parameter list constructs across multiple Trilinos packages.

(3) Memory management tools: Classes for aiding in correct allocation and deletion of memory. In particular, a reference counting pointer class that allows multiple references to a single object, deleting the object after the last reference is removed. These tools are very helpful in reducing the possibility of memory leaks in a program.

(4) Traits: Traits mechanisms [Myers 1995] are effective techniques for providing detailed information about supported generic data types. Teuchos provides three types of traits: ScalarTraits, OrdinalTraits and PacketTraits. ScalarTraits defines a variety of properties for supported scalar types. A partial list of traits includes:
—zero (one): The appropriate value for zero (one) for the given scalar type.
—magnitudetype: The data type that would be used by a norm for the given scalar type. For example, the magnitude type for double and complex double

is double.

—random: Function that produces a single random value of the given scalar type.

—Optional machine parameters: Optionally, a scalar type can also have machine parameters defined. These parameter have a one-to-one match with the LAPACK LAMCH parameters. A partial list of these parameters includes machine epsilon, arithmetic base, underflow and overflow. These parameters are important for robust floating point calculations in many situations, but proper definitions may not be obvious or essential for non-standard scalar types.

OrdinalTraits provides information for data types such as int. Again zero and one are defined, as is a descriptive label. Other ordinal traits are not needed at this point. PacketTraits is used to define the "size" of a packet type. This allows generic use of data transfer algorithms such as distributed data communications via MPI.

(5) Operation Counts: This class provides mechanisms for tracking and reporting operation counts, and associating a counting object with one or more computational objects.

(6) Exception handler: Error reporting class for uniform exception handling.

(7) Timers: Uniform interface to wall-clock timers.

5. TRILINOS PACKAGE INTEROPERABILITY MECHANISMS

As mentioned above, what a package *must* do to be called a Trilinos package is minimal, and varies with each package. In this section we list the primary mechanisms for a package to become part of Trilinos. Note that each mechanism is an extension or augmentation of package capabilities, creating connections between packages. Thus, a package does not need to change its internal structure to become part of Trilinos.

Mechanism 1: Package Accepts User Data as Epetra Objects. All solver packages require some user data (usually in the form of vectors and matrices) or require the user to supply the action of an operator on a vector. Accepting this data in the form of Epetra objects is the first Trilinos interoperability mechanism. Any package that accepts user data this way immediately becomes accessible to an application that has built its data using Epetra. We expect every Trilinos package to implement this mechanism in some way. Since Epetra provides a variety of ways to extract data from an Epetra object, minimally we expect that a package can at least copy data from the user objects that were built using Epetra. More often, a well-designed package can typically encapsulate Epetra objects and ask for services from the Epetra objects without explicitly copying them. In the future, as Tpetra matures (and C++ compilers mature), we expect Tpetra to be a companion package to Epetra, fulfilling a similar role.

Mechanism 2: Package Callable via TSF Interfaces. TSF provides a set of abstract interfaces that can be used to interface to a variety of solver packages. TSF can accept pre-constructed solver objects, e.g., preconditioners, iterative solvers,

etc., by simple encapsulation or it can construct solver objects using one of a variety of factories. Once constructed, a solver object can be further modified by passing it a parameter list containing a list of key-value pairs that can control solver behavior when it is trying to solve a problem. For example, the parameter list could specify a residual tolerance for an iterative solver.

A package is callable via TSF if it implements one or more of the TSF abstract class interfaces, making it available to TSF users as one of a suite of possible solver options.

Mechanism 3: Package Can Use Epetra Internally. Another interoperability mechanism available to a package is that of using Epetra objects as the internal objects for storing vector, matrices, etc. that are seldom or never seen by the user. In many instances, this mechanism has no practical advantages. However, in some instances, there can be a saving in storage requirements. Furthermore, by using Epetra objects internally, a package can in turn use other Trilinos packages to manipulate its own internal objects.

Mechanism 4: Package accesses services via TSF interface. TSF provides an abstract solver interface with access to multiple concrete solvers. A package can access solver services via TSF and therefore be able to use any solver that implements the TSF interface. By using TSF to access external objects such vectors, linear operators and solvers, a package has access to any concrete implementation of the TSF interfaces. This is beneficial for access to a broad set of concrete classes, and also minimizes the need for additional abstract interfaces and the corresponding concrete implementations of these additional abstract interfaces.

Mechanism 5: Package Builds Under Trilinos `configure` *Scripts.* Trilinos uses Autoconf [Free Software Foundation a] and Automake [Free Software Foundation b] to build libraries and test suites. The Trilinos directory structure keeps each Trilinos package completely self-contained. As such, each package is free to use its own configuration and build process. At the same time, Trilinos has a top-level configure script that traverses the directory structure invoking package configure scripts, passing on any parameter definitions from the top level. Similarly, the make process is also recursive.

A package may easily be automatically built from the top-level Trilinos configuration and make process by copying and modifying the Autoconf and Automake scripts from another package. The benefit for doing this is that Autoconf and Automake improve the portability of a package across a broad set of platforms. Also, Automake provides a rich set of targets for building libraries, software distributions, test suites and installation processes. If a package adopts the Trilinos configuration and build process, it will be built automatically along with other Trilinos packages.

6. OVERVIEW OF CURRENT PACKAGE DEVELOPMENT

6.1 The Petra Object Model

The Petra class libraries provide a foundation for all Trilinos solver development. Petra provides object classes for constructing and using parallel, distributed memory matrices and vectors. Petra exists in multiple forms. Its most basic form is as an object model [Heroux et al. 2003]. As such, it is an abstract description

of a variety of vector, matrix and supporting classes, along with a description of how these classes interact. There are presently three implementations of the Petra Object Model: Epetra, Tpetra and Jpetra.

6.1.1 *Epetra: Essential Implementation of Petra Object Model.* Epetra [Heroux 2002] the current production version of Petra, is written for real-valued double-precision scalar field data only, and restricts itself to a stable core of the C++ language standard. As such, Epetra is very portable and stable, and is accessible to Fortran and C users. Epetra combines in a single package (i) support for generic parallel machine descriptions, (ii) extensive use of standard numerical libraries, (iii) use of object-oriented C++ programming and (iv) parallel data redistribution. The availability of Epetra has facilitated rapid development of numerous applications and solvers at Sandia because many of the complicated issues of working on a parallel distributed memory machine are handled by Epetra.

Application developers can use Epetra to construct and manipulate matrices and vectors, and then pass these objects to most Trilinos solver packages. Furthermore, solver developers can develop new algorithms using Epetra classes to handle the intricacies of parallel execution. Epetra also has extensive parallel data redistribution capabilities, including an interface to the Zoltan load-balancing library [Devine et al. 1999]. Epetra is split into two packages: a core package and a set of extensions.

6.1.2 *Tpetra: Templated C++ Implementation of Petra Object Model.* In addition to Epetra, we have started development of a templated version of Petra, called Tpetra, that implements the scalar and ordinal fields as templated types. When fully developed, Tpetra will allow matrices and vectors to be composed of real or complex, and single or double precision scalar values. Furthermore, in principle, any abstract data type (ADT) can be used as the scalar field type as long as the ADT supports basic mathematical operations such as addition and multiplication and inversion. Specifically, we could compute using an interval scalar field, matrices, integers, etc., without any additional code development in Tpetra. Tpetra can also use any size integer for indexing. Typically the ordinal field would be an integral data type such as int or long int. However, any ADT that supports an indexing capability can be used, including integers in other bases, or cyclic indexing. Additionally, Tpetra also uses the C++ language standard more fully. In particular, it utilizes the Standard Template Library (STL) [Stroustrup 2000], to provide good algorithmic efficiency with minimal code development.

We are developing Tpetra as a peer library to Epetra. By using partial specialization of templates, we are basing Tpetra on established libraries such as the BLAS [Lawson et al. 1979; Dongarra et al. 1988; Dongarra et al. 1990] and LAPACK [Anderson et al. 1995] and therefore acquire the performance and robustness of these libraries. Like Epetra, Tpetra is written for generic parallel distributed memory computers whose nodes are potentially shared memory multiprocessors.

6.1.3 *Jpetra: Java Implementation of Petra Object Model.* In addition to Tpetra, we are developing a Java implementation of Petra. The primary design goals of this project are to produce a library that is a high performance, pure Java implementation of Petra. By restricting ourselves to Java and avoiding the use of the Java Native Interface (JNI) [Sun Microsystems] to link to other libraries, we

get the byte-code portability that Java promises. The fundamental implication of these goals is that we cannot rely on BLAS [Lawson et al. 1979; Dongarra et al. 1988; Dongarra et al. 1990], LAPACK [Anderson et al. 1995] or MPI [Snir et al. 1998] since they are not written in Java, and we do not use the JNI. As such, we must track the development of pure Java equivalents of these libraries. Several efforts, including Ninja [Moreira et al. 2001] and MPJ [Carpenter et al. 2000], provide equivalent functionality to the BLAS, LAPACK and MPI, but are completely written in Java.

We will fully implement Jpetra as a peer library to Epetra. By making extensive use of Java interfaces, we can create loose dependencies on emerging BLAS, LAPACK and MPI replacements as they become mature and stable. Recently, several research efforts [Moreira et al. 2001; Pozo and Miller] have shown that there is no fundamental performance bottleneck using Java. Instead, Java compilers and user practices have been the issue. As a result, Java holds much promise as a high performance computing language. Java also has native graphical user interfaces (GUI) support. A significant part of Jpetra will be the development of GUI tools for visualization and manipulation of Jpetra objects.

6.2 TSF: The Trilinos Abstract Class Packages

Many different algorithms are available to solve any given numerical problem. For example, there are many algorithms for solving a system of linear equations, and many solver packages are available to solve linear systems. Which package is appropriate is a function of many details about the problem being solved and the platform on which application is being run. However, even though there are many different solvers, conceptually, from an abstract view, these solvers are providing a similar capability, and it is advantageous to utilize this abstract view. TSF is a collection of abstract classes that provides an application programmer interface (API) to perform the most common solver operations. It can provide a single interface to many different solvers and has powerful compositional mechanisms that support the light-weight construction of composite objects from a set of existing objects. As a result, TSF users gain easy access to many solvers and can bring multiple solvers to bear on a single problem.

TSF is split into several packages. The most important user-oriented classes are TSFCore and TSFExtended:

(1) **TSFCore:** As its name implies, TSFCore contains a small set of core classes that are considered essential to almost any abstract linear algebra interface. The primary user classes in TSFCore are Vector, MultiVector, LinearOp and VectorSpace. TSFCore is discussed in detail in [Bartlett et al. 2003].
(2) **TSFExtended:** TSFExtended builds on top of TSFCore and includes overloaded, block and composite operators, all of which support powerful abstraction capabilities. The Meros package relies on TSFExtended to implicitly construct sophisticated Schur compliment preconditioners in terms of existing component operators with little overhead in time or memory. Section 7 discusses this topic in detail.

Both TSFCore and TSFExtended are important because they allow interfacing and sophisticated use of numerical linear algebra objects without requiring the user

or application to commit to any particular concrete linear algebra library. This approach allows us to leverage the investment in sophisticated abstract numerical algorithms across many concrete linear algebra libraries and gives application developers a single API that provides access to many solver packages.

TSF provides abstract interfaces for vector, matrix, operator and solver objects. In addition, it has powerful aggregation mechanisms that allow existing TSF objects to be combined in a variety of ways to create new TSF objects. TSF can be useful in many situations. For example:

(1) Generic Krylov method implementation: If a preconditioned Krylov solver is implemented using TSF vectors and operators, then any concrete package that implements the TSF vector and operator interfaces can be used with the Krylov solver. Both the Belos iterative linear solver package an Anasazi eigensolver package are based on TSF interfaces and are therefore independent of the details of how vectors and linear operators are implemented.
(2) Generic solver driver: If an application accesses solver services via the TSF solver interfaces, then any solver that implements the TSF solver interface is accessible to that application.
(3) Aggregate objects to implicitly construct aggregate operators: TSF provides mechanisms to implicitly construct a matrix of operators, the sum or composition of two operators, the inverse of an operator, etc. Similar aggregation mechanisms are available for vectors, matrices and solvers.

6.3 AztecOO: Concrete Preconditioned Iterative Solvers

AztecOO is an object-oriented follow-on to Aztec [Tuminaro et al. 1999]. As such, it has all of the same capabilities as Aztec, but provides a more elegant interface and numerous functionality extensions. AztecOO specifically solves a linear system $AX = B$ where A is a linear operator, X is a multivector containing one or more initial guesses on entry and the corresponding solutions on exit, and B contains the corresponding right-hand-sides.

AztecOO accepts user matrices and vectors as Epetra objects. The operator A and any preconditioner, say $M \approx A^{-1}$, need not be concrete Epetra objects. Instead, AztecOO expects A and M to be Epetra_Operator or Epetra_RowMatrix objects. Both Epetra_Operator and Epetra_RowMatrix are pure virtual classes. Therefore, any other matrix library can be used to supply A and M, as long as that library can implement the Epetra_Operator or Epetra_RowMatrix interfaces, something that is fairly straight-forward for most linear solver libraries.

AztecOO provides scalings, parallel domain decomposition preconditioners, and a very robust set of Krylov methods. It runs very efficiently on distributed memory parallel computers or on serial computers. Also, AztecOO implements the Epetra_Operator interface. Therefore, an AztecOO solver object can be used as a preconditioner for another AztecOO object.

6.4 Belos: Generic implementation of Krylov and Block Krylov Methods

Belos contains a collection of standard Krylov methods such as conjugate gradients (CG), GMRES and Bi-CGSTAB. It also contain flexible variants of CG and GMRES, and block versions CG and GMRES. The flexible variants allow vari-

able preconditioners to be used, such that the preconditioner at each iteration can change. Block variants allow the solution of multiple simultaneous right-hand-sides. Block methods can also be very effective for problems that have just a few small eigenvalues, even if the solution to only a single right-hand-side is needed.

Belos is considered a generic implementation because it relies on TSF interfaces for access to linear operator, preconditioner and vector objects. Therefore it is not explicitly tied to any concrete linear algebra library and can in principle be used with any library that implements the TSF interfaces. In particular, Epetra can be used since Trilinos provides an Epetra implementation of the TSF interfaces.

6.5 Amesos: Object-oriented Interface to Direct Solvers

The Amesos package is markedly different than most other Trilinos packages. It is designed to provide a common interface to a collection of third-party direct sparse solvers. There are a number of high-quality direct sparse solvers available to the general public, each of which (i) has a unique interface and (ii) can be especially suitable for specific uses. Because of this, we provide access to these solvers through a common interface. Specifically, we provide interfaces to all direct solvers supported by Amesos. These interfaces allow Epetra matrices and vectors to be used with each third-party solver. At this time, we provide support for SuperLU (serial), SuperLUDist [Li and Demmel 2003], Kundert's Sparse solver (from Spice [Quarles et al. 2003]),DSCPack [Raghavan 2003], UMFPack [Davis 2003] and MUMPS [Amestoy et al. 2003].

In addition to providing access to third-party solvers, Amesos provides an abstract base class that facilitates generic use of a third-party solver once a solver object is instantiated. This abstract interface is implemented by each Amesos direct solver class. For example, except for the construction phase (which can be accomplished generically using a "factory" as described in the Design Patterns book [Gamma et al. 1994]), an instance of a solver object, whether it be a SuperLU solver instance, DSCPack, etc., can be driven via the the Amesos base solver interface. This interface allows the user to request computation of a symbolic factorization, numeric factorization and a solve. How a specific third-party package is used to implement these can vary. The primary purpose of the Amesos base solver interface is to support efficient reuse of information. Specifically, if a sequence of factorizations uses the same nonzero structure but has different values, the Amesos base solver class can allow efficient reuse of the structure. Similarly, repeated right-hand-side solves can be done sequentially.

6.6 Komplex: Solver Suite for Complex-valued Linear Systems

Komplex solves complex-valued linear systems using equivalent real-valued formulations of twice the dimension. Given the following complex-valued linear system:

$$Cw = d, \tag{1}$$

where C is an m-by-n known complex matrix, d is a known right-hand side and w is unknown, we can write Equation (1) in its real and imaginary terms,

$$(A + iB)(x + iy) = b + ic. \tag{2}$$

Equating the real and imaginary parts of the expanded equation, respectively, gives rise to four possible 2-by-2 block formulations. We list one of these in Equation (3).

K1 Formulation.

$$\begin{pmatrix} A & -B \\ B & A \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ c \end{pmatrix}. \tag{3}$$

Although most preconditioning and iterative methods are generally well-defined for complex-valued systems, with real-valued systems being a special case, most widely-available solver packages focus exclusively on real-valued systems or treat complex-valued systems as an afterthought. Therefore, by transforming the complex-valued system into a real-valued system, we can immediately leverage all of the investment in real-valued solvers. Komplex constructs an equivalent real-valued formulation for a given complex-valued linear system and then calls AztecOO to solve the problem, returning the solution back to the user in a form compatible with the original complex-valued problem. Details of mathematical and practical issues of Komplex can be found in Day and Heroux [Day and Heroux 2001].

6.7 Ifpack: Parallel Algebraic Preconditioners

Ifpack provides local incomplete factorization preconditioners in a parallel domain decomposition framework. It accepts user data as Epetra_RowMatrix objects (including Epetra_CrsMatrix, Epetra_VbrMatrix and Epetra_MsrMatrix objects, since these classes implement the Epetra_RowMatrix interface) and can construct a variety of ILU preconditioners. Ifpack preconditioners implement the Epetra_Operator interface. Therefore, they can be used as preconditioners for AztecOO. The current released version of Ifpack provides a relaxed ILUK preconditioner and incomplete Cholesky with threshold dropping.

6.8 ML: Multi-level Preconditioner Package

ML is a multigrid, or more generally, a multi-level preconditioner package for solving linear systems from partial differential equation (PDE) discretizations. Although any linear system can be used with ML, problems that have an underlying PDE nature have the best chance of successful use of ML.

ML provides several approaches to constructing and solving the multi-level problem:

(1) Algebraic smoothed aggregation approach [Vanek et al. 1996; Vanek et al. 1998]: The matrix graph is colored to create aggregates (groups) of nodes. These aggregates define a preliminary projection operator. A final projection operator is created by applying a smoother to the preliminary operator.

(2) Algebraic multigrid for Maxwell's equations: This approach is intended for preconditioning linear systems of the form $Ax = b$, where $A = S + M$, S is a discrete form of the operator $\nabla \times \nabla \times E$, M is a mass matrix, and E is the electric field. Such systems arise from discretizations of the eddy current approximations to Maxwell's equations by either edge elements or Yee-type schemes [Bochev et al. 2003; Yee 1966].
The smoother is a specialized distributed relaxation method [Bochev et al. 2003]. This method explicitly smooths in range(S), smooths on a projected

residual equation in $\ker(S)$, and updates the approximate solution.
The prolongation operator is constructed so that $\ker(S)$ is properly represented on each level. In order for ML to build this prolongator, the user must provide two additional auxiliary operators: a discrete gradient operator, and a nodal finite element matrix. Both operators are easy to construct and are often already available in applications. Further details can be found in [Bochev et al. 2003; Bochev et al. 2003]:

(3) Adaptive Grid approach: The original grid is used as the coarse grid and the adaptive refinements determined the fine grid. Prolongation and restriction operators are determined using simple interpolation and weighted injection.

(4) Two-grid approach: A fine and (very) coarse grid are used. Graph and spatial coordinates are used, but there is no necessary correlation required between the two grids.

ML has two modes of operation. In the first mode, ML can be run as a stand-alone solver. ML provides its own smoothers and iterative methods. In the second mode of operation, ML can also be used as a preconditioner to iterative methods within Aztec or AztecOO.

ML is quite flexible with regard to matrix formats. ML accepts user matrix data in its own format. In this case, ML needs two matrix access functions, the first to return a matrix row and the second to perform a matrix-vector multiply. ML also accepts Epetra matrix objects. More information is available in either the ML User's manual [Tong and Tuminaro 2000] and at the ML website [Tuminaro and Hu].

6.9 Meros

Meros uses the compositional, aggregation and overloaded operator capabilities of TSF to provide segregated/block preconditioners for linear systems related to fully-coupled Navier-Stokes problems. This class of preconditioners exploits the special properties of these problems to segregate the equations and use multi-level preconditioners on the matrix sub-blocks. The overall performance and scalability of these preconditioners approaches that of multigrid for certain types of problems. Although the present target problems are related to computational fluid dynamics, Meros itself is purely algebraic. Because of this, other types of applications can potentially use Meros if a similar underlying physics structure is present. The details of Meros are discussed in Section 7.

6.10 NOX: Nonlinear Solver Package

NOX provides a suite of nonlinear solver methods that can be easily integrated into an application. Historically, many applications have called linear solvers as libraries, but have provided their own nonlinear solver software. NOX can be an improvement because it provides a much larger collection of nonlinear methods, and can be easily extended as new nonlinear methods are developed.

NOX currently contains basic solvers such as Newton's method as well as multiple globalizations including line search and trust region algorithms. Line search algorithms include full step, backtracking (interval halving), polynomial (quadratic and cubic) and More-Thuente. Directions for the backtracking algorithms include

steepest descent, Newton, quasi-Newton, and Broyden.

NOX does not depend on any particular linear algebra package, making it easy to install. In order to interface to NOX, the user needs to supply methods that derive from the NOX Vector and Group abstract classes. The Vector interface supports basic vector operations such as dot products and vector updates. The Group interface supports non-vector linear algebra functionality and contains methods to evaluate the function and, optionally, the Jacobian. Complete details are provided on the NOX website [Kolda and Pawlowski].

Although users can provide their own Vector and Group implementation, NOX provides three implementations of its own: LAPACK, Epetra and PETSc. The LAPACK interface is an interface to the BLAS/LAPACK library. It is not intended for large-scale computations, but to serve as an easy-to-understand example of how one might interface to NOX.

All NOX solvers are in the NOX::Solver namespace. The solvers are accessed via the NOX::Solver::Manager. The recommended solver is the NOX LineSearchBased solver, which is a basic nonlinear solver based on a line search. Each solver has a number of options that can be specified, as documented in each class or on the NOX Parameter Reference Page.

The search directions are in the NOX::Direction namespace and accessed via the NOX::Direction::Manager. The default search direction for a line-search based method is the Newton direction.

Several line searches are available, as defined in the NOX::LineSearch, and accessed via the NOX::LineSearch::Manager class.

Convergence or failure of a given solver method is determined by the status tests defined in the NOX::StatusTest namespace. Various status tests may be combined via the Combo object. Users are free to create additional status tests that derive from the Generic status test class.

6.11 LOCA: Library of Continuation Algorithms

LOCA is a package of scalable continuation and bifurcation analysis algorithms. It is designed as an extension to the NOX nonlinear solver package since the interfacing requirements are a superset of those needed for nonlinear solution. When integrated into an application code, LOCA enables the tracking of solution branches as a function of system parameters and the direct tracking of bifurcation points. It also provides an interface to the Anasazi Eigensolver for obtaining linear stability information. The algorithms are chosen to work with codes that use Newton's method to reach steady solutions and to have minimal additional interfacing requirements over the nonlinear solver. Furthermore, they are designed for scalability to large problems, such as those that arise from discretizations of partial differential equations, and to run on distributed memory parallel machines [Salinger et al. 2002].

LOCA provides robust parameter continuation algorithms with sophisticated step size controls for tracking steady solutions or bifurcations. There is also an artificial parameter homotopy algorithm. The approach in LOCA for locating and tracking bifurcations begins with augmenting the residual equations defining a steady state with additional equations that describe the bifurcation [Salinger et al. 2001]. This is done generically. This augmented system is then sent to the NOX library

for solution. Instead of assembling the Jacobian matrix for the entire augmented system (a task that involves second derivatives and dense matrix rows), bordering algorithms are used to decompose the linear solve into several solves with smaller matrices. Almost all of the algorithms just require multiple solves of the Jacobian matrix for the steady state problem to calculate the Newton updates for the augmented system. This greatly simplifies the implementation, since this is the linear system solve that an application code using Newton's method will have invested in. Only the Hopf tracking algorithm requires the solution of a larger matrix, which is the complex matrix involving the Jacobian matrix and an imaginary multiple of the mass matrix. For this solve the Komplex package is used. Online documentation is available through the NOX webpage [Kolda and Pawlowski].

6.12 Anasazi: Eigensolver package

Anasazi is an extensible and interoperable framework for large-scale eigenvalue algorithms. The goal of this framework is to provide a generic interface to a collection of algorithms for solving large-scale eigenvalue problems.

Anasazi is interoperable because both the matrix and vectors (defining the eigenspace) are considered to be opaque objects—only knowledge of the matrix and vectors via elementary operations is necessary. An implementation of Anasazi is accomplished via the use of interfaces. Current interfaces available include Epetra, so any libraries that understand Epetra matrices and vectors (such as AztecOO) may also be used in conjunction with Anasazi, and an abstract interface to the LOCA package.

One of the goals of Anasazi is to allow the user the flexibility to specify the data representation for the matrix and vectors and so leverage any existing software investment. The algorithms that will be initially available through Anasazi are block implicitly restarted Arnoldi and Lanczos methods and preconditioned eigensolvers. These include a locally optimal block preconditioned conjugate gradient iteration (LOBPCG) for symmetric positive definite generalized eigenvalue problems, and a restarted preconditioned eigensolver for nonsymmetric eigenvalue problems.

6.13 Future Packages

In addition to the package discussed above, we anticipate the inclusion of numerous new packages in the coming months and years. The Trilinos framework offers an attractive setting for algorithm developers who want a well-supported software environment and distribution mechanism, as well as the ability use their software with other packages. Presently we anticipate incorporating PyTrilinos, a Python interface to selected Trilinos functionality that allows use of the scripting language Python to drive Trilinos. We also expect that the dense solver developed for, among other things, the Linpack benchmark will also become a Trilinos package. A code for performing the nonlinear solution, continuation, and stability analysis of codes with fixed-point iterations (such as explicit integration codes), based on the Recursive Projection Method, is another solver package under development.

To see a complete list of new packages in the future, please look at the online version of this overview document, available from the Trilinos website [Heroux].

7. AN ILLUSTRATION OF TRILINOS INTEROPERABILITY

The Meros package in Trilinos is designed to provide scalable preconditioners for the incompressible Navier-Stokes equations and similarly structured problems [Elman et al. 2003]. It is based on and extends the work of Kay, Loghin and Wathen [Kay et al. 2002] and Silvester, Elman, Kay and Wathen [Silvester et al. 2001]. The discrete problem can be written in the form

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ p \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ 0 \end{pmatrix} \tag{4}$$

The first step in realizing the preconditioner is to formally define the block factorization:

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} = \begin{pmatrix} I & 0 \\ BF^{-1} & I \end{pmatrix} \begin{pmatrix} F & B^T \\ 0 & -S \end{pmatrix} \tag{5}$$

where $S = BF^{-1}B^T$ is the Shur complement. Applying the inverse of the third term in Equation 5 to the equation itself we get

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} F & B^T \\ 0 & -S \end{pmatrix}^{-1} = \begin{pmatrix} I & 0 \\ BF^{-1} & I \end{pmatrix} \tag{6}$$

If we could use the matrix

$$\begin{pmatrix} F & B^T \\ 0 & -S \end{pmatrix}^{-1} \tag{7}$$

as a right preconditioner for a Krylov method applied to our original problem (Equation 4) then our preconditioned operator would be the right-hand-side of Equation 5 and at most 2 iterations of GMRES would be needed for convergence. Since this is not practical, we instead observe that we can write

$$\begin{pmatrix} F & B^T \\ 0 & -S \end{pmatrix}^{-1} = \begin{pmatrix} F^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & -B^T \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -S^{-1} \end{pmatrix} \tag{8}$$

In this form it is clear that, to apply the above preconditioner, we need to in turn apply two nontrivial operators: S^{-1} to a vector in the discrete pressure space, and F^{-1} to a vector in the discrete velocity space. Since these tasks are too expensive, we instead use approximations to S^{-1} and F^{-1}.

A variety of approximations to S^{-1} and F^{-1} have been developed [Elman et al. 2003]. In general, the strength of this preconditioning approach is that well-established preconditioning methods can be applied on the subblock operators, in turn building up a preconditioner for fully-coupled problem. In particular, because the subblocks are simpler than the global problem, robust multi-level preconditioners can be defined that provide near-mesh independent convergence properties for the global problem.

The Meros package utilizes many features of Trilinos in order to provide a scalable, parallel distributed memory implementation of the preconditioners described above. It takes advantage of the abstract interfaces in TSF, both to access other Trilinos packages and to implicitly construct approximations of S^{-1} and F^{-1}. In addition, it uses the ML package for implementing multi-level preconditioners, AztecOO for smoothers, Ifpack for algebraic preconditioners, NOX for nonlinear

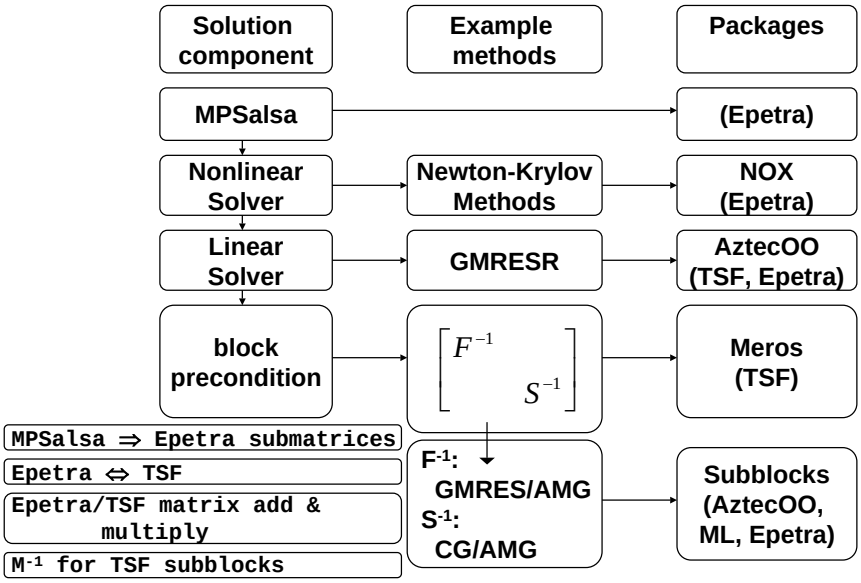

Fig. 2. Meros Interaction Diagram

iterations and Epetra for interfacing to the application and for basic parallel linear algebra. Figure 2 illustrates the collaboration and use of Trilinos packages by Meros in the context of MPSalsa [Salinger et al. 1996], a reacting flow modeling application. It is also worth noting that Meros was integrated into the Trilinos framework using the "new_package" package. Integration took less that one day.

8. SOFTWARE ENGINEERING ISSUES

As computer modeling and simulation play an increasingly important role in engineering and science, a critical issue is software quality. Multiple issues are important, but they can be summed up as follows: *If computer modeling and simulation is to be on the critical path of engineering and scientific processes, those who rely on our software must have confidence in our computational results.* It is worth noting that, although much of the work we do to improve software quality is technical in nature, ultimately it is our ability to instill trust in our clients that determines whether or not our software will be used in a production environment.

Much of Trilinos was developed under funding from the Advanced Scientific Computing Initiative (ASCI). A major focus of ASCI is Software Quality Engineering (SQE), which is the set of practices for ensuring that high-quality, relevant software is produced, and that software processes are well-defined, documented and followed. The present ASCI SQE practices for Sandia National Laboratories are defined in [Zepper et al. 2003]. This document describe 47 practices that must be adopted by each major software project receiving ASCI funding. These practices

cover areas such as software requirements, design, implementation and maintenance, project management, tracking and oversight, verification and validation, training, risk management, etc.

One of the most important goals of Trilinos is to minimize the work of SQE on the individual package development teams. Given that each package is typically written by five or fewer people, implementation of the ASCI SQE process by each package team would be almost impossible. Fortunately, the Trilinos infrastructure can address the majority of the ASCI SQE practice, fully or partially. In fact, of the 47 practices listed in [Zepper et al. 2003], 32 of them are the sole responsibility of the Trilinos framework, and the framework provides significant support for the remaining 15 [Heroux et al. 2003]. Only those practices that are truly unique to a package are primarily package responsibilities. This gives package developers the ability to focus on the core issues of algorithm design and implementation, and package level documentation and testing.

9. CONCLUSIONS

In this article we have presented an overview of Trilinos, a two-level framework for the development and ongoing support of mathematical software libraries. By clearly defining a package concept, Trilinos can provide a ready-made infrastructure that substantially reduces the overhead of mathematical software development and support. Trilinos provides a framework for integrating independent solver packages, making packages inter-operable and providing a common "look-and-feel" for Trilinos users. Furthermore, Trilinos provides a collection of useful services for independent solver developers, making integration of a package into Trilinos attractive to developers. The primary advantages that the Trilinos Project provides are:

(1) A common core of basic linear algebra classes: We can minimize redundant work and jumpstart a new parallel application by utilizing Petra class libraries to construct and manipulate matrix, graph and vector objects.

(2) Extensive use of abstract classes, primarily TSF, to define the interaction between Trilinos packages: By using abstract interfaces in Trilinos packages, we are not explicitly dependent on Petra classes for functionality. This allows us to use any concrete matrix and vector software with Trilinos packages, including PETSc, BLAS, and LAPACK.

(3) A collection of common software tools and processes: New packages can be integrated into Trilinos very easily. Furthermore, if a package does not have its own well-developed set of software engineering tools and processes, the Trilinos design makes it easy for a package to incorporate Autotools, bug and feature tracking, source code control and mail lists.

(4) A one-to-many API for applications: Application developers who adopt the TSF abstract interfaces gain access to many solvers via a single mechanism. Furthermore, additional third party solvers are easily added as necessary.

(5) Solver aggregation capabilities: Via the TSF aggregation capabilities, it is possible to combine many solvers and bring them to bear on a single problem.

REFERENCES

Amestoy, P. R., Duff, I. S., L'Excellent, J.-Y., and Koster, J. 2003. MUMPS home page. http://www.enseeiht.fr/lima/apo/MUMPS.

Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. 1995. *LAPACK Users' Guide*, Second ed. SIAM Pub.

Bailey, D. H., Hida, Y., Li, X. S., and Thompson, B. 2002. ARPREC: An arbitrary precision computation package. Tech. rep., Lawrence Berkeley National Laboratory.

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

Balay, S., Gropp, W., McInnes, L., and Smith, B. 1998a. PETSc 2.0 users manual. Tech. Rep. ANL-95/11 - Revision 2.0.22, Argonne National Laboratory.

Balay, S., Gropp, W., McInnes, L., and Smith, B. 1998b. PETSc home page. http://www.mcs.anl.gov/petsc.

Bartlett, R. A., Heroux, M. A., and Long, K. R. 2003. TSFCore 1.0: A package of light-weight object-oriented abstractions for the development of abstract numerical algorithms and interfacing to linear algebra libraries and applications. Tech. rep., Sandia National Laboratories. To appear.

Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Jemmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. 1997. *ScaLAPACK Users' Guide*. SIAM Pub.

Bochev, P. B., Garasi, C., Hu, J. J., Robinson, A. C., and Tuminaro, R. S. 2003. An improved algebraic multigrid method for solving Maxwell's equations. *SIAM J. Sci. Comput.*. To appear.

Bochev, P. B., Hu, J. J., Robinson, A. C., and Tuminaro, R. S. 2003. Towards robust 3D Z-pinch simulations: discretization and fast solvers for magnetic diffusion in heterogeneous conductors. *Electronic Transactions on Numerical Analysis 15*.

Carpenter, B., Getov, V., Judd, G., Skjellum, A., and Fox, G. 2000. MPJ: MPI-like message passing for Java. *Concurrency: Pract. Exper. 12,* 11 (September), 1019–1038.

Davis, T. 2003. UMFPACK home page. http://www.cise.ufl.edu/research/sparse/umfpack.

Day, D. and Heroux, M. A. 2001. Solving complex-valued linear systems via equivalent real formulations. *SIAM J. Sci. Comput. 23,* 2, 480–498.

Devine, K. D., Hendrickson, B. A., Boman, E. G., John, M. M. S., and Vaughan, C. 1999. Zoltan: A dynamic load-balancing library for parallel applications – user's guide. Tech. Rep. SAND99-1377, Sandia National Laboratories, Albuquerque, NM.

Dongarra, J., DuCroz, J., Hammarling, S., and Hanson, R. 1988. An extended set of Fortran basic linear algebra subprograms. *ACM Transactions on Mathematical Software 14*.

Dongarra, J. J., Bunch, J., Moler, C., and Stewart, G. 1979. *LINPACK Users' Guide*. SIAM Pub.

Dongarra, J. J., Du Croz, J., Hammarling, S., and Duff, I. 1990. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software 16,* 1 (March), 1–17.

Elman, H., Howle, V. E., Shadid, J. E., and Tuminaro, R. S. 2003. A parallel block multi-level preconditioner for the 3d incompressible navier-stokes equations.

Free Software Foundation. Autoconf Home Page. http://www.gnu.org/software/autoconf.

Free Software Foundation. Automake Home Page. http://www.gnu.org/software/automake.

Free Software Foundation. Gnu CVS Home Page. http://www.gnu.org/software/cvs.

Free Software Foundation. Gnu m4 home page. http://www.gnu.org/software/m4.

Free Software Foundation. Gnu mailman home page. http://www.gnu.org/software/mailman/mailman.html.

Free Software Foundation. Libtool Home Page. http://www.gnu.org/software/libtool.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1994. *Design Patterns, Elements of Reusable Object Oriented Software*. Addison-Wesley.

Heroux, M. A. Trilinos home page. http://software.sandia.gov/trilinos.

Heroux, M. A. 2002. *Epetra Reference Manual*, 2.0 ed. http://software.sandia.gov/trilinos/packages/epetra/doxygen/latex/EpetraReferenceManual.pdf.

Heroux, M. A., Hoekstra, R. J., and Williams, A. B. 2003. An object model for parallel numerical linear algebra computations. Tech. rep., Sandia National Laboratories. In preparation.

Heroux, M. A., Willenbring, J. M., and Heaphy, R. 2003. Trilinos Developers Guide Part II: ASCI Software Quality Engineering Practices Version 1.0. Tech. Rep. SAND2003-1899, Sandia National Laboratories.

Kay, D., Loghin, D., and Wathen, A. 2002. A preconditioner for the steady-state navier-stokes equations. *SIAM J. Sci. Comput.*.

Kolda, T. G. and Pawlowski, R. P. Nox home page. http://software.sandia.gov/nox.

Lawson, C., Hanson, R., Kincaid, D., and Krogh, F. 1979. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software 5*.

Li, X. and Demmel, J. 2003. SuperLU home page. http://crd.lbl.gov/ xiaoye/SuperLU/.

Moreira, J. E., Midkiff, S. P., Gupta, M., Artigas, P. V., Wu, P., and Almasi, G. 2001. The NINJA project. *Communications of the ACM 44,* 10 (October).

Myers, N. C. June 1995. Traits: a new and useful template technique. *C++ Report*.

Pozo, R. and Miller, B. SciMark 2.0. http://math.nist.gov/scimark2/.

Quarles, T., Pederson, D., Newton, R., Sangiovanni-Vincentelli, A., and Wayne, C. 2003. SPICE home page. http://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE.

Raghavan, P. 2003. DSCPACK home page. http://www.cse.psu.edu/ raghavan/Dscpack.

Salinger, A. G., Bou-Rabee, N. M., Pawlowski, R. P., Wilkes, E. D., Burroughs, E. A., Lehoucq, R. B., and Romero, L. A. 2001. LOCA: A library of continuation algorithms - Theroy and implementation manual. Tech. rep., Sandia National Laboratories, Albuquerque, New Mexico 87185. SAND 2002-0396.

Salinger, A. G., Devine, K. D., Hennigan, G. L., Moffat, H. K., Hutchinson, S. A., and Shadid, J. N. 1996. MPSalsa: A finite element computer program for reacting flow problems part 2 - user's guide. Tech. Rep. SAND96–2331, Sandia National Laboratories.

Salinger, A. G., Lehoucq, R. B., Pawlowski, R. P., and Shadid, J. N. 2002. Computational bifurcation and stability studies of the 8:1 thermal cavity problem. *Internat. J. Numer. Meth. Fluids 40,* 8, 1059–1073.

Shadid, J. N., Moffat, H. K., Hutchinson, S. A., Hennigan, G. L., Devine, K. D., and Salinger, A. G. 1995. MPSalsa: A finite element computer program for reacting flow problems part 1 - theoretical development. Tech. Rep. SAND95–2752, Sandia National Laboratories.

Silvester, D., Elman, H., Kay, D., and Wathen, A. 2001. Efficient preconditioning of the linearized navier-stokes equations for incompressible flow. *J. Comp. Appl. Math.*.

Smith, B. T., Boyle, J. M., Dongarra, J. J., Garbow, B. S., Ikebe, Y., Klema, V. C., and Moler, C. B. 1976. *Matrix Eigensystem Routines – EISPACK Guide*, Second ed. Lecture Notes in Computer Science, vol. 6. Springer–Verlag, New York.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J. 1998. *MPI-The Complete Reference, Volume 1, The MPI core*. The MIT Press.

Stroustrup, B. 2000. *The C++ Programming Language*. Addison-Wesley.

Sun Microsystems. Java Native Interface. http://java.sun.com/products/jdk/1.2/docs/guide/jni.

The Mozilla Organization. Mozilla Bonsai Home Page. http://www.mozilla.org/bonsai.html.

The Mozilla Organization. Mozilla Bugzilla Home Page. http://www.mozilla.org/projects/bugzilla.

Tong, C. and Tuminaro, R. 2000. ML2.0 Smoothed Aggregation User's Guide. Tech. Rep. SAND2001-8028, Sandia National Laboratories, Albq, NM.

Tuminaro, R. S., Heroux, M. A., Hutchinson, S. A., and Shadid, J. N. 1999. *Official Aztec User's Guide, Version 2.1*. Sandia National Laboratories, Albuquerque, NM 87185.

Tuminaro, R. S. and Hu, J. Ml home page. http://www.cs.sandia.gov/ tuminaro/ML_Description.html.

van Heesch, D. Doxygen home page. http://www.doxygen.org.

Vanek, P., Brezina, M., and Mandel, J. 1998. Convergence of Algebraic Multigrid Based on Smoothed Aggregation. Tech. Rep. 126, UCD/CCM, Denver, CO.

Vanek, P., Mandel, J., and Brezina, M. 1996. Algebraic Multigrid Based on Smoothed Aggregation for Second and Fourth Order Problems. *Computing 56*, 179–196.

Vaughan, G., Elliston, B., Tromey, T., and Taylor, I. 2000. *Gnu Autoconf, Automake, and Libtool*. New Riders.

Yee, K. 1966. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. *IEEE Trans. Antennas and Propagation 16*, 302–307.

Zepper, J., Aragon, K., Ellis, M., Byle, K., and Eaton, D. 2003. Sandia National Laboratories ASCI Applications Software Quality Engineering Practices, Version 2.0. Tech. rep., Sandia National Laboratories.

Received: ????; revised: ???; accepted: ???