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.

Sandia has developed scalable solver algorithms and software for many years. Often this development has been done within the context of a specific application code, providing a good robust solver that meets the particular needs of that application. Even Aztec [Tuminaro et al. 1999], 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 across 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 for new development. Furthermore, many applications are standardizing on the Trilinos matrix and vector classes. As a result, these applications have access to all Trilinos capabilities without interface modifications.

Trilinos has a two-level design where the fundamental building block is a *package*. Although package is a common term, we define it rigorously within Trilinos. Specifically, each package is a numerical library (or collection of libraries) that is:

—Focused on important, state-of-the-art algorithms in its problem regime. For example: algebraic preconditioners, nonlinear solvers, scalable data models, etc.

—Developed by a small team of domain experts.

—Self-contained, with minimal dependencies on other packages.

—Configurable, buildable, tested and documented on its own.

These packages can be distributed within Trilinos or separately. The Trilinos frame-

work provides a common look-and-feel that includes configuration, documentation, licensing, and bug tracking. There are also guidelines and tools for adding new packages to Trilinos.

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 individual 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. Section 2 discusses work that is related to Trilinos. Integration of a package into Trilinos, and what Trilinos can provide to a package, will be discussed in Section 3. Section 4 discusses the current collection of Trilinos packages. Section 5 presents some code fragments that illustrate the interoperability of Trilinos packages. This section also discusses 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 6 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. RELATED WORK

General-purpose solver libraries have been used successfully 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 libraries that have made a tremendous impact, providing robust portable solvers to a broad set of applications. More recently, libraries such as PETSc [Balay et al. 1998b; 1998a; 1997], ScaLAPACK [Blackford et al. 1997], PLAPACK [Alpatov et al. 1997] and Aztec [Tuminaro et al. 1999] have enabled development of parallel applications by giving users access to parallel distributed memory solvers that are easy-to-use and robust.

The purpose of the Trilinos project is to foster development of new solver libraries by minimizing the costs of new development, while still leveraging the investment in established libraries such as those just mentioned. These two goals are accomplished by active research and development of new libraries and by use of the Trilinos package architecture. Although Trilinos is unique in design, a number of other projects have some similarities. In particular, PETSc, the Common Component Architecture (CCA) [Forum 2004], the Matrix Template Library (MTL) [A. Lumsdaine 2004] and POOMA [Oldham 2002] share attributes with Trilinos.

2.1 Trilinos and PETSc

Trilinos is similar to PETSc in that both provide libraries for constructing and using sparse and dense distributed matrices and vectors. Also, both projects provide solver libraries for linear, nonlinear, time-dependent and eigenvalue problems. Trilinos differs from PETSc in that Trilinos is written primarily in C++, has an explicit modular architecture and each package is interoperable with other packages but not interdependent. Most Trilinos packages are data-neutral in design and in fact could easily use PETSc libraries to provide a variety of capabilities via straight-forward implementation of documented abstract interfaces, without modifying PETSc or

Trilinos source code.

2.2 Trilinos and the Common Component Architecture

The Trilinos package architecture, which is the primary difference between Trilinos and PETSc, is the primary similarity between Trilinos and the CCA. Like the CCA, the Trilinos package architecture supports interoperability of independent pieces of software. However, the CCA uses an advanced runtime environment to manage the coupling of components, whereas Trilinos uses configure-enabled conditional compilation and the polymorphism that is inherent in the C++ language. Both approaches are essential and very compatible. In fact, the modularity and independence of Trilinos packages make them easy to wrap as CCA components.

2.3 Trilinos and MTL

The Trilinos packages Epetra and Tpetra are similar to MTL. All three are written in C++ and provide numerical linear algebra objects that can be used to implement numerical algorithms. MTL makes more aggressive use of template facilities in C++ and allows more elegant and flexible implementations. However, portability of MTL is a major issue since a number of C++ compilers do not support some of the features used by MTL. Another difference is that MTL provides its own functionality that is similar to the BLAS, whereas Epetra and Tpetra rely on BLAS libraries for performance. A final major difference is that MTL executes on serial machines only. Epetra and Tpetra support distributed memory objects and provide parallel data repartitioning capabilities [Boman et al. 2004].

2.4 Trilinos and POOMA

Trilinos and POOMA are similar in that they both provide serial and parallel distributed memory linear algebra objects, along with automatic support of interprocessor communication. POOMA differs from Trilinos in its focus on arrays and overloaded operator syntax, which make grid-based calculations and explicit methods easy to implement. In fact, much of POOMA terminology uses grid concepts. POOMA also focuses only on basic linear algebra computations. No implicit solvers are provided, although grid-based implicit solvers can be easily built using POOMA objects. Trilinos packages such as Epetra and Tpetra can be used for grid-based computations, but their primary focus is on irregular, unstructured computations. Other Trilinos packages access linear algebra services via abstract vector and operator interfaces. Therefore, POOMA object could be used via these interfaces.

2.5 Trilinos and Other Solver Libraries

Trilinos provides significant new solver capabilities as self-contained Trilinos packages. It also provides an explicit, documented, modular architecture that facilitates interoperability. This allows Trilinos to easily use other solver libraries and to also provide, via any individual Trilinos package, solver capabilities to other libraries. External solver libraries are made Trilinos-compatible, not by integrating them into Trilinos, but by augmenting the capabilities of the external library. This difference in design philosophy is subtle but important, especially for scalable growth in package count. Using this approach, Trilinos will never become a large monolithic piece

of software, something that is very important as the volume of solver software continues to grow.

3. TRILINOS PACKAGE ARCHITECTURE AND SERVICES

In our experience mathematical libraries tend to be written by small teams of domain experts. For example, approximately 25 staff members (not including students) contribute to Trilinos development across approximately 25 different packages, but most individual Trilinos packages are developed by one to three staff members, and no single package has more than five developers. Some staff members contribute to more than one package, but very few contribute to more than three packages. Another observation is that mathematical libraries tend to be written by experienced numerical software developers who do not have much, if any, experience with formal software tools and processes. Both of these observations have motivated the Trilinos design and implementation. The Trilinos package architecture naturally supports small inter-related team development efforts. Trilinos services, provided on a package-by-package basis, directly address the second observation.

3.1 Modularity via Packages

Each Trilinos package is fully self-sufficient and self-contained, unless there are explicit dependencies designed into a package. Package source code and revision history is contained within a single directory structure. Mail lists, software faults, documentation, websites and interoperability are all organized via packages.

Package interoperability is accomplished via configure-time enabling of package-to-package coupling. For example, the algebraic preconditioner package called IFPACK has many parameters to control how preconditioners are constructed and used, e. .g ., level of fill in an incomplete factorization. Teuchos, described in Section 4.1, is a package of commonly used utility classes such as portable BLAS interfaces and timers. Teuchos provides a parameter list class that can be used to specify parameter values. If the argument `--enable-teuchos` is specified when IFPACK is configured, Teuchos ParameterList support will be compiled in IFPACK, otherwise this code will not be enabled and only IFPACK's internal parameter setting method is available. Configure-time enabling of package interoperability is commonly used for any optional package coupling that makes sense. In this way packages retain autonomy but can easily be combined with any other package that makes sense algorithmically.

3.2 Package Services Provided by Trilinos

Trilinos provides a variety of services to package developers. When a new package is introduced into Trilinos, each of these services is established, immediately providing the new package with necessary tools to address important software quality engineering practices (see Section 6 for details). All tools are accessible from the main Trilinos website [Heroux 2004]. Specifically the services we establish are:

—**Source code repository:** Trilinos source code is maintained in a CVS [Free Software Foundation 2004c] repository that is accessible via a secure connection from anywhere on the internet. For most new packages, use of a supported CVS

repository is the most important service Trilinos provides, since development is typically in the early stages and there are few if any users.

—**Mail lists:** Trilinos uses the Mailman [Free Software Foundation 2004e] list manager to provide communication support for each package. Package lists start with the package name followed by the list function, e. g. , AztecOO-Announce. The following mail lists are established for each package:
  —Package-Announce: All news related to this package is sent to this list, such as release announcements, etc.
  —Package-Developers: Package developer discussions occur on this list, including design and policy discussions. Key development decisions are posted here for archival purposes.
  —Package-Users: List for package users. General discussions about use of the package are conducted here, typically monitored by the package development team.
  —Package-Checkins: All log messages that are submitted with source code changes to the CVS repository are sent to this list. Anyone wishing to see the moment-to-moment activity in package development can subscribe to this list.
  —Package-Regression: All output from the regression test suite for the package is sent to this list.

—**Issue tracking:** Bugzilla [The Mozilla Organization 2004b] is a web-based issue tracking application that supports submission and tracking of software issues, including enhancements and faults. Each package has its own Bugzilla product.

—**Fault identification:** Bonsai [The Mozilla Organization 2004a] is a web-based application that supports a variety of CVS repository browsing capabilities, and links changes in the repository to Bugzilla issues. Bonsai is most useful as a way to quickly identify changes in source code that have cause a software fault.

—**Configuration management:** Autoconf [Free Software Foundation 2004a], Automake [Free Software Foundation 2004b] and Libtool [Free Software Foundation 2004f] 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.

  Trilinos provides a library of M4 [Free Software Foundation 2004d] 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, checking for a user-defined MPI [Snir et al. 1998] C compiler or determining inter-language linking rules. This library of macros minimizes the amount of redundant effort in using Autotools, and make it easier to apply a general change to the configure process for all packages.

—**Automated Regression testing:** Trilinos provides a variety of regression testing capabilities. Integrating new tests into Trilinos is accomplished by creating specially named directories in the CVS repository and creating scripts that run package tests. For example, an executable script committed to the repository in the directory Trilinos/packages/epetra/test/scripts/daily/serial can be executed

manually and will also run daily on any platform that has the Epetra serial test harness installed, as part of the automated regression test harness. On a nightly basis, the test harness builds the most recent versions of Trilinos libraries and runs any tests that are present in one of these special directories.

3.3 The new_package Package

In order to reduce the start up time for a new Trilinos package, whether it is the importing of existing software or development of new source, the new_package package provides a good starting point for accessing the services that Trilinos provides. new_package provides a starting point for:

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

—Autotools: 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 2004] to produce accurate, extensive source code documentation.

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

—Website: The Trilinos home page [Heroux 2004] 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 independent from the rest of the Trilinos website. Both new_package and its website have been successfully used by other projects that have nothing to do with Trilinos.

3.4 Package Maturation Process

Typically in the early development stages of a new package, many of the services mentioned above are not heavily used. In fact, the new package is often isolated from other packages and use of the CVS repository is of primary importance. As the package matures and the user base grows, package use of the other services also grows, as does the need for interoperability with other packages. Gradually, over the span of several years, a package matures to the point where it is fully using all services provided by Trilinos and is fully interoperable with other packages.

One strength of the Trilinos package architecture is its natural support of gradual package maturation. At any given point in time, each Trilinos package can be in any state of development. As a Trilinos release date approaches, we categorize packages for public, limited or no release. For each package that will be released, the package development team certifies the package and provides us with a repository tag for the tested version of the package. In this way, even the release process is distributed and scalable.

Package	Description	Release			
		3.1 (Sep 2003)		4.0 (Jun 2004)	
		General	Limited	General	Limited
Amesos	3rd Party Direct Solver Suite		X	X	X
Anasazi	Eigensolver package			X	X
AztecOO	Linear Iterative Methods	X	X	X	X
Belos	Block Linear Solvers				X
Epetra	Basic Linear Algebra	X	X	X	X
EpetraExt	Extensions to Epetra		X	X	X
Ifpack	Algebraic Preconditioners	X	X	X	X
Jpetra	Java Petra Implementation				X
Kokkos	Sparse Kernels			X	X
Komplex	Complex Linear Methods	X	X	X	X
LOCA	Bifurcation Analysis Tools	X	X	X	X
Meros	Segregated Preconditioners		X		X
ML	Multi-level Preconditioners	X	X	X	X
NewPackage	Working Package Prototype	X	X	X	X
NOX	Nonlinear solvers	X	X	X	X
Pliris	Dense direct Solvers				X
Teuchos	Common Utilities			X	X
TSFCore	Abstract Solver API			X	X
TSFExt	Extensions to TSFCore			X	X
Tpetra	Templated Petra				X
Totals		8	11	15	20

Table I. Trilinos Package Summary and Release Status

4. OVERVIEW OF CURRENT PACKAGE DEVELOPMENT

Trilinos package counts have grown rapidly in the four years of its existence. The "Tri" in Trilinos originally stood for the initial three packages it contained. Table I shows a brief description and status of packages that have been part of the past two releases. In this section we provide an overview of the primary packages in current release of Trilinos. The discussion is ordered so that packages that are most fundamental and broadly useful are presented first.

4.1 Common Tools Package: Teuchos

As the number of Trilinos packages grows, we have developed the need for a collection of tools that can be leveraged 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. Many packages have some interoperability with Teuchos but very few have an essential dependence on it. Parameter values for most high-level packages can be set using the Teuchos ParameterList class. For example, ML, IFPACK and AztecOO parameters can be set this way, as long as Teuchos support was enabled (via `--enable-teuchos`) at configure time.

Teuchos provides classes and interfaces for:

(1) Templated access to BLAS and LAPACK interfaces. Teuchos provides a set of

interfaces that have templated ordinal and scalar types. Typically the ordinal type is the common integer type. For the scalar type, 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. Support for two multiprecision scalar datatypes is already provided, specifically ARPREC [Bailey et al. 2002] and GMP [Granlund 2004].

(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.

(3) Memory management tools: Classes for aiding in correct allocation and deletion of memory. In particular, Teuchos provides 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.

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.

Although Teuchos has been available for less than a year, it has been adopted by many packages to provide functionality and uniform access by applications. Teuchos parameter lists and BLAS interfaces have been especially useful.

Object	Packet Definition
Vector	Single vector value.
Multivector	Row of vector values.
Compressed Index Storage Graph (CISGraph)	List of column/row indices for one graph row/column.
Compressed Index Storage Matrix (CISMatrix)	List of values and column/row indices for one matrix row/column.

Table II. Packet Definitions for Common Petra Object Types

4.2 The Petra Object Model and Packages

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.

The Petra class libraries provide a foundation for all Trilinos solver development. Petra provides 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. 2004]. 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. Before describing these implementations, we first discuss distribution concepts.

Although a detailed discussion of data and work distribution in the Petra Object Model is beyond the scope of this paper, we briefly present these topics. We also introduce *ElementSpace* objects, objects that describe the layout of distributed objects in Petra, and their relationship to distributed vectors, multivectors, graphs and matrices. A full description of the Petra Object Model can be found in [Boman et al. 2004].

A critical feature of Petra is the ability to easily define and efficiently execute data redistribution in parallel. Because of this, Petra's data model is built around this concept. Data redistribution requires the identification of data packets that should be moved as part of the redistribution. For a simple vector that is distributed across a parallel machine, the natural packet is a single vector value. For a collection of vectors with the same distribution, what we define to be a multivector, the natural packet is all values across a row of column vectors. For matrices, if we store nonzero entries row-by-row, then the index and nonzero values data are used to define a packet. We can similarly define a column-oriented matrix, or more generally a compressed index matrix, where the row or column orientation is part of the definition of the class attributes. A compressed index graph is similar to a matrix, except that it involves pattern information only. Table II lists the common linear algebra objects we use, and describes the packet definition for each object.

To facilitate redistribution and provide a generic analysis capability, we use elements as a representation of packets. Specifically, regardless of packet definition, we associate an element global ID (GID) with each packet of a distributed object. We do this by defining an *ElementSpace* object (called a Map object in Epetra).

Processor ID	Element GIDs
0	$\{0, 1, \ldots, 24\}$
1	$\{25, 26, \ldots, 49\}$
2	$\{50, 51, \ldots, 74\}$
3	$\{75, 76, \ldots, 98\}$

Table III. Standard Distribution of 99 Elements

ElementSpace objects are used to

(1) Define the layout of distributed object across a parallel machine, and
(2) Compute a plan to redistribute an object distributed via one ElementSpace to another ElementSpace distribution.

Example: Suppose we want to construct a vector with 99 entries so that it is approximately evenly distributed across four processors, and vector entries are stored in increasing order. Table III lists a natural distribution of element GIDs that would describe the layout of this type of vector.

In our object-oriented model, we construct a distributed object by first defining an *ElementSpace* object. Given an *ElementSpace* object, we can define any number of linear algebra objects with a compatible layout. For example, using an *ElementSpace* object with the element ID distribution in Table III, we can define any number of vectors having the first 25 vector entries on PE 0, the next 25 on PE 1, etc. We can also define a row matrix having rows 0 through 24 on PE 0, rows 25 through 49 on PE 1, etc. It is worth noting that, although the example in Table III is a simple linear distribution of GIDs, *ElementSpace* objects can contain GIDs in any order and multiplicity on any processor. This ability is important for data redistribution and replicating data across the parallel machine.

For distributed matrices and graphs four *ElementSpace* objects are needed:

—RowElementSpace: On each processor this *ElementSpace* lists the GIDs that will be "managed" by that processor, typically meaning that the processor owns part or all of the data associated with that row.
—ColumnElementSpace: Same as RowElementSpace, except that it deals with columns.
—DomainElementSpace: The distribution of GIDs associated with vectors and multivectors that are in the domain of the matrix. These GIDs must be uniquely associated with a processor.
—RangeElementSpace: Same as DomainElementSpace, except that it deals with the range space.

Given this general framework, the Petra Object Model supports any distribution of data across the parallel machine. Any matrix, graph, vector or multivector entry can be owned by any processor, or shared or replicated across multiple processors. Depending on the type of constructor used, data may be assigned by any processor, even if the processor is not the eventual owner of the data. A more efficient class of constructors requires that the processor which owns the data (as prescribed by the relevant *ElementSpace* objects) must define the data. Some constructors support data entry that is in between these two extremes. Finally, operations between

matrices and vectors can be performed on objects with different distributions as long as the DomainElementSpace and RangeElementSpace objects are compatible with the appropriate vectors.

We have only briefly touched on the data distribution and redistribution capabilities provided by the Petra Object Model. However, for brevity we now move on to a description of the three Petra implementations.

4.2.1 *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 including BLAS and LAPACK, (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. As mentioned above, Epetra supports only double-precision real arithmetic. Support for other types is found in Tpetra.

4.2.2 *Tpetra: Templated C++ Implementation of Petra Object Model.* In addition to Epetra, we are developing a templated version of Petra, called Tpetra, that implements the scalar and ordinal fields as templated types. Tpetra allows matrices and vectors to be composed of real or complex, and single or double precision scalar values. Building on Teuchos, Tpetra provides distributed memory parallel support for generic datatypes. 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.

4.2.3 *Jpetra: Java Implementation of Petra Object Model.* The primary design goals of Jpetra 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 2003] 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, LAPACK or MPI 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. Presently we are using CCJ [Nelisse et al. 2003] as the implementation of the Jpetra parallel machine interface for inter-node communications, and we are using JLAPACK [Doolin et al.

1998] to implement our interface for serial linear algebra kernels.

4.3 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. Furthermore, TSFExtended has powerful compositional mechanisms that support the light-weight construction of composite objects from a set of existing objects (see Section 5.2). 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 5.2 discusses this topic in detail.

Both TSFCore and TSFExtended are important because they allow interfacing to 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.

4.4 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.

4.5 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 contains flexible and block variants of CG and GMRES. The flexible variants allow variable 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.

4.6 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]. Amesos also wholly contains a single serial solver called KLU [Davis and Stanley 2004]. KLU provides us with a default solver capability, even when no third-party solver is available.

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 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.

4.7 Komplex: Solver Suite for Complex-valued Linear Systems

Komplex solves complex-valued linear systems using equivalent real-valued formulations of twice the dimension. It 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].

4.8 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 algebraic 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.

4.9 ML: Multi-level Preconditioner Package

ML is 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].

(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.

More information is available in either the ML User's manual [Tong and Tuminaro 2000] and at the ML website [Tuminaro and Hu 2004].

4.10 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 5.2.

4.11 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. Although users can provide their own Vector and Group implementation, NOX provides three implementations of its own: LAPACK, Epetra and PETSc. Complete details are provided on the NOX website [Kolda and Pawlowski 2004].

4.12 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 differen-

tial equations, and to run on distributed memory parallel machines [Salinger et al. 2002].

4.13 Anasazi: Eigensolver package

Anasazi is an extensible and interoperable framework for large-scale eigenvalue algorithms written using TSF interfaces to abstract operator and vector objects. The first version of Anasazi includes 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. Details can be found at the Anasazi home page [Thornquist et al. 2004].

4.14 Future Packages

In addition to the package discussed above, we anticipate the inclusion of numerous new packages in the coming 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. The dense solver developed for, among other things, the Linpack benchmark will also become a Trilinos package called Pliris [Kotulski 2004]. 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.

5. TRILINOS PACKAGE INTEROPERABILITY

What a package *must* do to be Trilinos compatible is minimal, and varies with each package. In this section we discuss the primary mechanisms for Trilinos compatibility and then go on to illustrate with code fragments how some of these mechanisms work. Table IV lists the six primary interoperability mechanisms. 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 Trilinos compatible.

5.1 Using Epetra Objects with Trilinos Packages

In this section we provide several examples of how Epetra matrices and vectors can be initialized and used by multiple Trilinos packages. In this particular case we use the Epetra entry-by-entry construction facilities for simplicity, but it is worth noting that Epetra matrices can be efficiently constructed row-by-row or column-by-column, where entries can be single scalar values or variable sized block entries. There is also support for clique-based construction where, for example, finite element stiffness matrices may be passed in to fill the matrix. Finally, almost any combination of these fill techniques can be used on a given matrix.

Figure 1 shows a very simple test program that builds a tridiagonal matrix distributed across the parallel machine with 1000 rows per processor. The right-hand-side and solution vectors are chosen to have the same layout. We do not require

Mechanism	Comments
Package accepts user matrices and vectors as Epetra objects.	—Any package that accepts user data this way immediately becomes accessible to an application that has built its data using Epetra. —Minimally we expect that a package can copy data from user objects built using Epetra. Often a package can encapsulate Epetra objects without explicitly copying data.
Package parameters can be set and retrieved via Teuchos ParameterLists.	—The Teuchos ParameterList provides a uniform way to handle parameters across packages. —Package dependence on Teuchos is typically conditional, enabled by `--enable-teuchos`.
Package provides adaptors to TSF interfaces.	—Most packages can supply preconditioning or solver services in a generic sense. —If a package provides an implementation of one or more TSF abstract interfaces, it is usable by any other package written using TSF interfaces.
Package can use Epetra internally.	—Epetra (and in the future Tpetra) can be used for storing vector, matrices, etc. that are seldom or never seen by the user. —By using Epetra objects internally, a package can in turn use other Trilinos packages to manipulate its own internal objects.
Package accesses services via TSF interfaces.	—Example of generic programming. —Allows access to multiple other Trilinos packages.
Package build process is compatible with Trilinos `configure` script.	—The Trilinos `configure` script supports many options such as selective enabling of packages, specification of BLAS and LAPACK location, etc. —Packages can enhance interoperability by supporting these build options as appropriate.

Table IV. Package Interoperability Mechanisms

that matrices and vector have identical distributions. However, a discussion of the parallel data distribution features of Epetra is beyond the scope of this paper. The reader is referred to [Boman et al. 2004] for a full discussion of this topic. Next we construct a linear problem, which assures that the matrix and vectors are compatible. Finally we construct an AztecOO solver object to solve the system using Jacobi-scaled Conjugate Gradients.

In Figure 2 we show two different preconditioners setup processes. Each is a legitimate substitution for the single line `paramlist.set("precond", AZ_Jacobi);`

```
// Header files omitted...
int main(int argc, char *argv[]) {

// Initialize MPI, MpiComm
MPI_Init(&argc,&argv);
Epetra_MpiComm Comm( MPI_COMM_WORLD );

// Put same number of equations on each PE
int NumMyElements = 1000 ;
Epetra_Map Map(-1, NumMyElements, 0, Comm);
int NumGlobalElements = Map.NumGlobalElements();

// Create Matrix: tridiag(-1,2,-1)
Epetra_CrsMatrix A(Copy, Map, 3);
double negOne = -1.0; double posTwo = 2.0;

for (int i=0; i<NumMyElements; i++) {
  int GlobalRow = A.GRID(i);
  int RowLess1 = GlobalRow - 1;
  int RowPlus1 = GlobalRow + 1;
  if (RowLess1!=-1)
    A.InsertGlobalValues(GlobalRow, 1,
                         &negOne, &RowLess1);
  if (RowPlus1!=NumGlobalElements)
    A.InsertGlobalValues(GlobalRow, 1,
                         &negOne, &RowPlus1);
  A.InsertGlobalValues(GlobalRow, 1, &posTwo,
                       &GlobalRow);
}
// Complete matrix preprocessing
A.FillComplete();

// ***** Create x and b vectors
Epetra_Vector x(Map);
Epetra_Vector b(Map);
b.Random(); // Fill RHS with random #s
```

```
// Create Linear Problem
Epetra_LinearProblem problem(&A, &x, &b);

// Create/define AztecOO instance
AztecOO solver(problem);

// Create Teuchos Parameter List
Teuchos::ParameterList paramlist;

// Tell AztecOO to solve using CG
paramlist.set("AZ_solver", AZ_cg);

// Select AztecOO Jacobi preconditioner
// In later examples this single line will
// be replaced by construction and setup of
// other preconditioners such as ML and IFPACK

paramlist.set("precond", AZ_Jacobi);

// Iterate up to 100 iterations or until
// tolerance of 1.0E-8 is met.
solver.Iterate(1000, 1.0E-8);

// Report results, finish
cout << "Solver performed "
     << solver.NumIters()
     << " iterations." << endl
     << "Norm of true residual = "
     << solver.TrueResidual()
     << endl;

MPI_Finalize() ;
return 0;
}
```

Fig. 1. Simple C++ program to construct and solve a tridiagonal system using Epetra and AztecOO

in Figure 1. The primary observation we make here is that, although the Epetra, AztecOO, ML and IFPACK packages are independently developed, they are fully interoperable with each other.

5.2 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

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ p \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ 0 \end{pmatrix} \tag{1}$$

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

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} = \begin{pmatrix} I & 0 \\ BF^{-1} & I \end{pmatrix} \begin{pmatrix} F & B^T \\ 0 & -S \end{pmatrix} \tag{2}$$

```
// This code fragment constructs an ML
// multi-level preconditioner based on
// smoothed aggregation.
// It can directly replace the single line:
// paramlist.set("precond", AZ_Jacobi);
// in the code in Figure 1.


// Create ML multilevel preconditioner
ML *ml_handle;

// Maximum number of levels
int N_levels = 10;

// output level
ML_Set_PrintLevel(3);
ML_Create(&ml_handle,N_levels);

// wrap Epetra matrix A into ML matrix
// (data is NOT copied)
EpetraMatrix2MLMatrix(ml_handle, 0, A);

// We are interested in smoothed aggregation
// so create a ML_Aggregate object
ML_Aggregate *agg_object;
ML_Aggregate_Create(&agg_object);

// specify max coarse size (ML will not
// coarsen further
// if the matrix at a given level is
// smaller than specified here)
ML_Aggregate_Set_MaxCoarseSize(agg_object,1);

// generate the hierarchy
N_levels =
  ML_Gen_MGHierarchy_UsingAggregation
  (ml_handle, 0, ML_INCREASING, agg_object);

// Set a symmetric Gauss-Seidel smoother for
// MG method (change if matrix  not symmetric)
ML_Gen_Smoother_SymGaussSeidel(ml_handle,
ML_ALL_LEVELS, ML_BOTH, 1, ML_DEFAULT);

// generate solver
ML_Gen_Solver(ml_handle,ML_MGV, 0, N_levels-1);

// wrap ML_Operator into Epetra_Operator
ML_Epetra::MultiLevelOperator
  MLop(ml_handle,Comm,*Map,*Map);

// Register this operator as preconditioner
solver.SetPrecOperator(&MLop);

// ... Now continue with execution of AztecOO
```

```
// This code fragment constructs an IFPACK
// preconditioner based on incomplete
// Cholesky with threshold tolerances
// It can directly replace the single line:
// paramlist.set("precond", AZ_Jacobi);
// in the code in Figure 1.


// This time we will count FLOPS
Epetra_Flops fact_counter;

// Create and start timer
Epetra_Time timer;

// Create Incomplete Cholesky preconditioner
// Drop threshold of 10E-6.
// Maximum entries per row is 100.
Ifpack_CrsIct ICT(A, 1.0e-6, 100);

// Associate flop counter with preconditioner
ICT.SetFlopCounter(fact_counter);

// Initialize preconditioner values
ICT.InitValues(A);

// Compute factor
ICT.Factor();

// Get elapsed time since timer was created
elapsed_time = timer.ElapsedTime();

// Get number of FLOPS, MFLOPS
total_flops = fact_counter.Flops();
MFLOPs = total_flops/elapsed_time/1000000.0;

cout << "Time to construct preconditioner = "
     << elapsed_time << endl
     << "MFLOPS for Factorization = "
     << MFLOPs << endl;

// Compute condition number estimate
// for preconditioner
double Condest;
ICT.Condest(transA, Condest);
cout << "Preconditioner condition estimate = "
     << Condest << endl;

// Register this operator as preconditioner
solver.SetPrecOperator(&ICT);

// ... Now continue with execution of AztecOO
```

Fig. 2. Two interchangeable code fragments that construct preconditioners for program in Figure 1.

where $S = BF^{-1}B^T$ is the Shur complement. Applying the inverse of the third term in Equation 2 to the equation itself we get

$$\begin{pmatrix} F & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} F & B^T \\ 0 & -S \end{pmatrix}^{-1} = \begin{pmatrix} I & 0 \\ BF^{-1} & I \end{pmatrix} \tag{3}$$

If we could use the matrix

$$\begin{pmatrix} F & B^T \\ 0 & -S \end{pmatrix}^{-1} \tag{4}$$

as a right preconditioner for a Krylov method applied to our original problem (Equation 1) then our preconditioned operator would be the right-hand-side of Equation 2 and at most 2 iterations of GMRES would be needed for convergence. Since this is not practical, we instead observe that we can write

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

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 iterations and Epetra for interfacing to the application and for basic parallel linear algebra. Figure 3 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.

6. 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

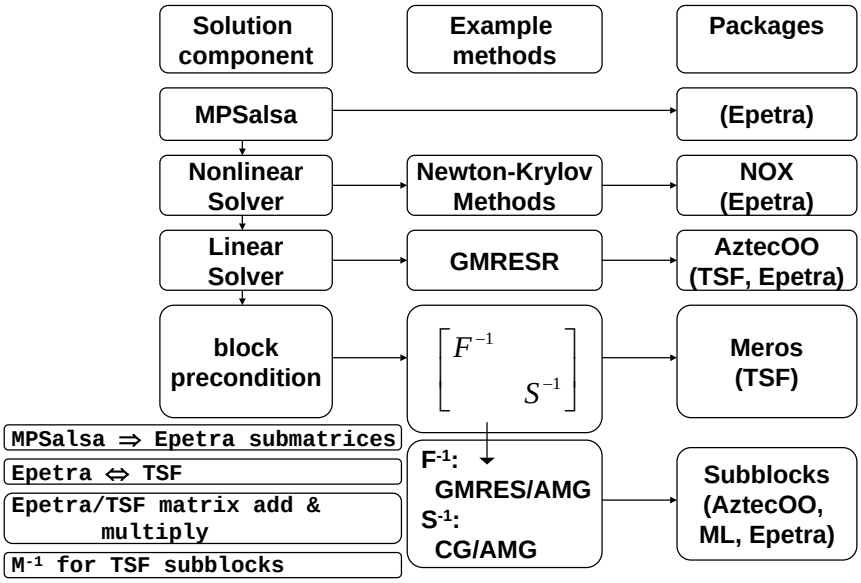

Fig. 3. Meros Interaction Diagram

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 and risk management.

One of the most important goals of Trilinos is to minimize the work of SQE for 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. Table V highlights how Trilinos aids package developers with some of the 47 practices listed in [Zepper et al. 2003]. Details of Trilinos vs. package responsibilities is presented in [Heroux et al. 2003]. In general, 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

Trilinos Service	SQE Practices Impact
Yearly Trilinos User Group Meeting (TUG) and Developer Forum: Trilinos Users and Developers gather once a year for tutorials, package feature updates, user/developer requirements discussion and developer training.	—All steps of the Requirements including gathering, derivation, documentation, feasibility, etc. —User training. —Developer training.
Monthly Trilinos leaders meetings: Trilinos leaders, including package development leaders, key managers, funding sources and other stakeholders participate in monthly phone meetings to discuss any timely issues related to the Trilinos Project.	—Requirements tracking. —Developer Training. —Design reviews. —Policy decisions across all development phases.
Trilinos and package mail lists: Trilinos lists for leaders, announcements, developers, users, checkins and similar lists at the package level support a variety of communication. All lists are archived, providing critical artifacts for assessments and audits.	—Developer, user and client communication. —Repository of requirements, design and testing artifacts. —Announcement and documenting of releases.
Trilinos and Trilinos3PL source repositories: All source code, development and user documentation is retained and tracked. In addition, reference versions of all external software, including BLAS, LAPACK, Umfpack, etc. are retained in Trilinos3PL.	—Source management. —Versioning. —Third-party software management.
Trilinos Bugzilla Database: Supports collection, tracking and management of requirements, enhancements and software faults.	—Requirements gathering and tracking. —Customer support.
Trilinos `configure` script and M4 macros: The Trilinos `configure` script and related macros support portable installation of Trilinos and its packages..	—Portability. —Software release.
Trilinos test harness: Trilinos provides a base testing plan and automated testing across multiple platforms, plus creation of testing artifacts. Test harness results are used to derive a variety of metrics for SQE.	—Pre-checkin and regression testing. —Software metrics.

Table V. Trilinos Framework support of package SQE practices

package level documentation and testing.

7. CONCLUSIONS

In this article we have presented an overview of Trilinos, a framework for the development and ongoing support of mathematical software libraries. By defining, documenting and prototyping its package architecture, Trilinos provides a ready-made infrastructure that substantially reduces the cost of mathematical software development. As a result, Trilinos has grown rapidly and is able to continue its growth in a scalable way. Furthermore, interoperability of packages supports a broad set of new solvers for coupled multi-physics applications that are a critical requirement for advanced high-fidelity simulations. Finally, the package-oriented delivery of services by Trilinos for source management, communication, issue tracking, configuration management and regression testing allow package developers to readily obtain a high level of SQE support at minimal cost.

REFERENCES

A. Lumsdaine, e. a. 2004. The matrix template library home page. http://www.osl.iu.edu/research/mtl.

Alpatov, P., Baker, G., Edwards, C., Gunnels, J., Morrow, G., Overfelt, J., van de Geijn, R., and Wu, Y.-J. J. 1997. Plapack: parallel linear algebra package design overview. In *Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM)*. ACM Press, San Jose, CA, 1–16.

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., Philadelphia, PA.

Bailey, D. H., Hida, Y., Li, X. S., and Thompson, B. 2002. ARPREC: An arbitrary precision computation package. Tech. Rep. LBNL-53651, Lawrence Berkeley National Laboratory. September.

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 lightweight object-oriented abstractions for the development of abstract numerical algorithms and interfacing to linear algebra libraries and applications. Tech. Rep. SAND2003-1378, Sandia National Laboratories. April.

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. 25,* 2.

Boman, E., Devine, K., Heaphy, R., Hendrickson, B., Heroux, M., and Preis, R. 2004. Ldrd report: Parallel repartitioning for optimal solver performance. Tech. Rep. SAND2004-0365, snl. February.

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.

Davis, T. and Stanley, K. 2004. Sparse lu factorization of circuit simulation matrices. http://www.cise.ufl.edu/ davis/techreports/KLU/pp04.pdf.

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

Doolin, D. M., Dongarra, J., and Seymour, K. 1998. Jlapack-compiling lapack fortran to java. http://icl.cs.utk.edu/projects/f2j/f2jreport/f2jreport.html.

Elman, H., Howle, V. E., Shadid, J. N., and Tuminaro, R. S. 2003. A parallel block multi-level preconditioner for the 3d incompressible navier-stokes equations. *Journal of Computational Physics 187,* 2, 504–523.

Forum, C. 2004. The common component architecture home page. http://www.cca-forum.org.

Free Software Foundation. 2004a. Autoconf Home Page. http://www.gnu.org/software/autoconf.

Free Software Foundation. 2004b. Automake Home Page. http://www.gnu.org/software/automake.

Free Software Foundation. 2004c. Gnu CVS Home Page. http://www.gnu.org/software/cvs.

Free Software Foundation. 2004d. Gnu m4 home page. http://www.gnu.org/software/m4.

Free Software Foundation. 2004e. Gnu mailman home page. http://www.gnu.org/software/mailman/mailman.html.

Free Software Foundation. 2004f. 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.

Granlund, T. 2004. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 4.1.3 ed. SWOX AB.

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

Heroux, M. A. 2004. Trilinos home page. http://software.sandia.gov/trilinos.

Heroux, M. A., Hoekstra, R. J., and Williams, A. B. 2004. 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. 2004. Nox home page. http://software.sandia.gov/nox.

Kotulski, J. D. 2004. Pliris home page. http://software.sandia.gov/Trilinos/packages/pliris.

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*.

Nelisse, A., Maassen, J., Kielmann, T., and Bal, H. E. 2003. Ccj: Object-based message passing and collective communication in java. *Concurrency & Computation: Practice & Experience 15,* 3–5, 341–369.

Oldham, J. D. 2002. *POOMA: A C++ Toolkit for High-Performance Parallel Scientific Computing*, 1.01 ed. CodeSourcery, LLC. http://www.codesourcery.com/public/pooma/manual.pdf.

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., 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. 128*, 261–279.

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. 2003. Java Native Interface. http://java.sun.com/products/jdk/1.2/docs/guide/jni.

The Mozilla Organization. 2004a. Mozilla Bonsai Home Page. http://www.mozilla.org/bonsai.html.

The Mozilla Organization. 2004b. Mozilla Bugzilla Home Page. http://www.mozilla.org/projects/bugzilla.

Thornquist, H., Lehoucq, R., and Hetmaniuk, U. 2004. Anasazi home page. http://software.sandia.gov/Trilinos/packages/anasazi.

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. 2004. Ml home page. http://www.cs.sandia.gov/ tuminaro/ML_Description.html.

van Heesch, D. 2004. 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: ???