SAND REPORT

SAND2003-2927
Unlimited Release
Printed August 2003

An Overview of Trilinos

Michael Heroux, Roscoe Bartlett, Vicki Howle
Robert Hoekstra, Jonathan Hu, Tamara Kolda,
Richard Lehoucq, Kevin Long, Roger Pawlowski,
Eric Phipps, Andrew Salinger, Heidi Thornquist,
Ray Tuminaro, James Willenbring and Alan Williams

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1110

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/ordering.htm

SAND2003-2927
Unlimited Release
Printed August 2003

An Overview of Trilinos

Michael Heroux, Jonathan Hu, Richard Lehoucq,
Heidi Thornquist, Ray Tuminaro, and James Willenbring
Computational Mathematics and Algorithms Department

Roscoe Bartlett
Optimization and Uncertainty Estimation Department

Vicki Howle, Tamara Kolda, and Kevin Long
Computational Sciences and Mathematics Research Department

Robert Hoekstra, Roger Pawlowski, Eric Phipps, and Andrew Salinger
Computational Sciences Department

Alan Williams
High Performance Computing and Networking Department

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1110

Abstract

The Trilinos Project is an effort to facilitate the design, development, integration and ongoing support of mathematical software libraries. In particular, our goal is to develop parallel solver algorithms and libraries within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific applications. Our emphasis is on developing

robust, scalable algorithms in a software framework, using abstract interfaces for flexible interoperability of components while providing a full-featured set of concrete classes that implement all abstract interfaces.

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.

Trilinos packages are primarily written in C++, but provide some C and Fortran user interface support. We provide an open architecture that allows easy integration with other solver packages and we deliver our software to the outside community via the Gnu Lesser General Public License (LGPL) [20]. This report provides an overview of Trilinos, discussing the objectives, history, current development and future plans of the project.

Acknowledgments

The authors would like to acknowledge the support of the ASCI and LDRD programs that funded development of Trilinos.

Intentionally Left Blank

Contents

Figures

Nomenclature

Trilinos The name of the project. Also a Greek term which, loosely translated means “a string of pearls,” meant to evoke an image that each Trilinos package is a pearl in its own right, but is even more valuable when combined with other packages.

Package A self-contained collection of software in Trilinos focused on one primary class of numerical methods. Also a fundamental, integral unit in the Trilinos framework.

new_package A sample Trilinos package containing all of the infrastructure to install a new package into the Trilinos framework. Contains the basic directory structure, a collection of sample configuration and build files and a sample “Hello World” package. Also a website.

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

AztecOO Linear solver package based on preconditioned Krylov methods. A follow-on to the Aztec solver package [47]. Supports all Aztec interfaces and functionality, but also provides significant new functionality.

Belos A Greek term meaning “arrow.” Belos is the next generation of iterative solvers. Belos solvers are written using “generic” programming techniques. In other words, Belos is written using TSF abstract interfaces and therefore has no explicit dependence on any concrete linear algebra library. Instead, Belos solvers can be used with any concrete linear algebra library that implements the TSF abstract interfaces.

Ifpack Object-oriented algebraic preconditioner, compatible with Epetra and AztecOO. Supports construction and use of parallel distributed memory preconditioners such as overlapping Schwarz domain decomposition, Jacobi scaling and local Gauss-Seidel relaxations.

Komplex Complex linear equation solver using equivalent real formulations [11], built on top of Epetra and AztecOO.

LOCA Library of continuation algorithms. A package of scalable stability analysis algorithms (available as part of the NOX nonlinear solver package). 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.

Meros Segregated preconditioning package. Provides scalable block preconditioning for problems that couple simultaneous solution variables such as Navier-Stokes problems.

ML Algebraic multi-level preconditioner package. Provides scalable preconditioning capabilities for a variety of problem classes.

NOX A collection of nonlinear solvers, designed to be easily integrated into an application and used with many different linear solvers.

Petra A Greek term meaning “foundation.” Trilinos has three Petra libraries: Epetra, Tpetra and Jpetra that provide basic classes for constructing and manipulating matrix, graph and vector objects. Epetra is the current production version that is split into two packages, one core and one extensions.

- Epetra Current C++ production implementation of the Petra Object Model. The “E” in Epetra stands for “essential” implying that this version provides the most important capabilities that are commonly needed by our target application base. Epetra supports real, double-precision floating point data only (no single-precision or complex). Epetra avoids explicit use of some of the more advanced features of C++, including templates and the Standard Template Library, that can be impediments to portability.
- Tpetra The future C++ version of Petra, using templates and other more advanced features of C++. Tpetra supports arbitrary scalar and ordinal types, and makes extensive use of advanced C++ features.
- Jpetra A Java implementation of Petra, supporting real, double-precision data. Written in pure Java, it is designed to be byte-code portable and can be executed across multiple compute nodes.

Teuchos A collection of classes and service software that is useful to almost all Trilinos packages. Includes reference-counted pointers, parameter lists, templated interfaces to BLAS, LAPACK and traits for templates.

TSF A collection of abstract interfaces that supports application access to a variety of Trilinos capabilities, supports interoperability betweeen Trilinos packages and provides future extensibility. TSF is composed of several packages. The primary user packages are:

- TSFCore TSFCore provides a basic collection of abstract interfaces to vectors, linear operators, solvers, etc. These interfaces provide a common interface for applications to access one or more packages that implement the abstract interface. These interfaces can also be used by other packages in Trilinos to accomplish the same purpose.
- TSFExtended TSFExtended builds on top of TSFCore, providing implicit aggregation capabilities and overloaded operators.

1 Background

A core requirement of many engineering and scientific applications is the need to solve linear and non-linear systems of equations, eigensystems and other related problems. Thus it is no surprise that any part of the application that solves these problems is called a "solver." The exact definition of what specifically constitutes a solver depends on many factors. However, a good working definition of a solver is the following: *Any piece of software that finds unknown values for some set of discrete governing equations in an application.* Another characteristic of solvers is that we can often implement them in such a way that they are "general-purpose", so that the details of how the discrete problem was formed are not specifically needed for the solver to work (although information about problem characteristics can often be vital to robust solutions.)

General-purpose linear and eigensolvers have been successfully used across a broad set of applications and computer systems. EISPACK [40], LINPACK [14] and LAPACK [2] 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 [5, 4, 3] and Aztec [47] 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 [37, 39] 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 excellent 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 components without any unnecessary interface modifications.

This document provides an overview of the Trilinos project, focusing on the project philosophy and description, and providing the reader with a summary of the project in its current state.

2 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.

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.

Integration of a package into Trilinos, and what Trilinos can provide to a package, have multiple possibilities that will be discussed in Section 3. Section 4 discusses two special collections of Trilinos packages: Petra and TSF. The general definition of a Trilinos package is presented in Section 6 An overview of current software research and development is given in Section 7. Finally, this document contains an appendix, which gives a brief tutorial on object-oriented concepts for readers who are unfamiliar with the area.

3 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.

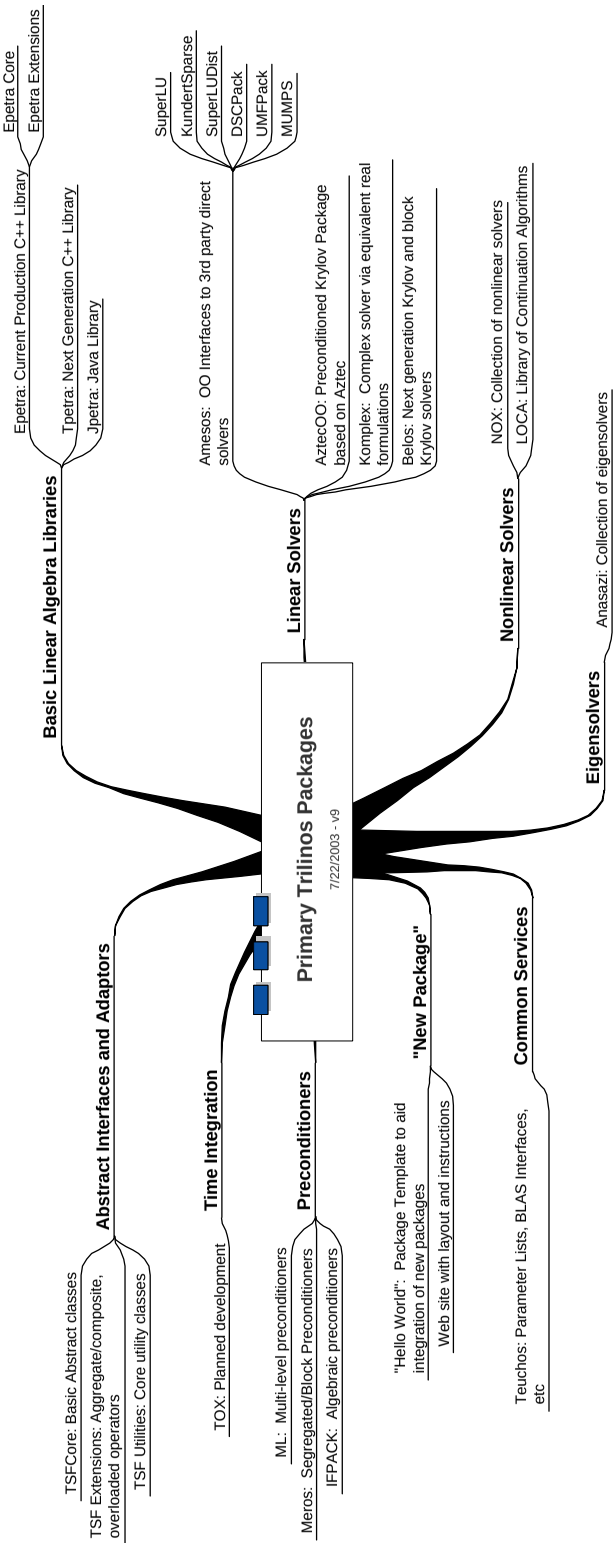

Figure 1. Current collection of Trilinos Packages

3.1 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 [17], Automake [18] and Libtool [22] provide a robust, full-featured set of tools for building software across a broad set of platforms (see also the "Goat Book" [52]). Although these tools are not official standards, they are commonly used in many packages. Nearly all existing Trilinos packages use Autoconf and Automake. Libtool support will be added in future releases.

  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 [21] 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 [2] 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.

- **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) [27, 16, 15] and LAPACK [2] 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 [19] 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 [44]. Features and bug reports are tracked using Bugzilla [45], and email lists 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 [25].

- **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 [49] to produce accurate, extensive source code documentation.
  - Regression testing: Simple regression testing example is part of new_package.
  - Website: The Trilinos home page [25] 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.

4 Petra and TSF: Two Special Package Collections

In order to understand what Trilinos provides beyond 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.

4.1 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 7.1) 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.

4.2 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. As a result, TSF users gain easy access to many solvers and can bring multiple solvers to bear on a single problem.

5 Common Tools Package: Teuchos

As the number of Trilinos packages grows, we have developed the need for a common 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 class for internal use. 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 provided generic loops sets for a limited set of key kernels (NOTE: Generic support for LAPACK functionality is very limited). 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 using arbitrary precision arithmetic. This capability can be used to support interval arithmetic, geometric transformation calculations, integers and calculations with many more data types.

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 [32] 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 provide 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 trait 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.

6 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. (See Appendix 8 for the definition of a factory.) 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 [17] and Automake [18] 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.

7 Overview of Current Package Development

7.1 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 [24]. 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.

Epetra: Essential Implementation of Petra Object Model

Epetra [**?**] 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 components. Furthermore, solver developers can develop many new algorithms relying solely on 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 [13]. Epetra is split into two packages: a core package and a set of extensions.

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) [42], 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 [27, 16, 15] and LAPACK [2] 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.

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) [43] 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 [27, 16, 15], LAPACK [2] or MPI [41] 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 [31] and MPJ [9], 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 [31, 33] 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.

7.2 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 [6].

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.

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.

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.

7.3 AztecOO: Concrete Preconditioned Iterative Solvers

AztecOO is an object-oriented follow-on to Aztec [47]. 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.

7.4 Belos: Generic Implementation of 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 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.

7.5 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 solver 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 [28], Kundert's Sparse solver (from Spice [34]),DSCPack [35], UMFPack [10] and MUMPS [1].

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 Appendix), 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 pur-

pose 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. One should note that this fine-grain control does not eliminate simple uses. If the "solve" method in the Amesos base solver class is called without any previous call to the numeric factorization, or to neither the symbolic or numeric factorization, the solver object will be aware of this and perform the necessary preliminary steps for the call to the solve method to succeed.

7.6 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. KomplexOO 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 [12].

7.7 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 large 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.

7.8 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 [51, 50]: 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 [7, 53].

   The smoother is a specialized distributed relaxation method [7]. 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 [7, 8]:

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 [46] and at the ML website [48].

7.9 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.

7.10 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 [26].

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. The Epetra interface is an interface to Epetra. The PETSc interface is an interface with the PETSc library.

All NOX solvers are in the NOX::Solver namespace. The solvers are accessed via the NOX::Solver::Manager. The recommended solver is the NOX LineSearch-Based 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.

7.11 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 [38].

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 [36]. This is done generically. This augmented system is then sent to the NOX library for solution. Instead of loading up 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 [26].

7.12 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.

7.13 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 [25].

8 Conclusions

The Trilinos project 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

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

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM Pub., second edition, 1995.

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

[4] S. Balay, W. Gropp, L. McInnes, and B. Smith. PETSc 2.0 users manual. Technical Report ANL-95/11 - Revision 2.0.22, Argonne National Laboratory, 1998.

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

[6] Roscoe A. Bartlett, Michael A. Heroux, and Kevin R. Long. 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. Technical report, Sandia National Laboratories, 2003. To appear.

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

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

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

[10] Tim Davis. UMFPACK home page. http://www.cise.ufl.edu/research/sparse/umfpack, 2003.

[11] David Day and Michael A. Heroux. Solving complex-valued linear systems via equivalent real formulations. *SIAM J. Sci. Comput.*, 23(2):480–498, 2001.

[12] David Day and Michael A. Heroux. Solving complex-valued linear systems via equivalent real formulations. *SIAM J. Sci. Comput.*, 23(2):480–498, 2001.

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

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

[15] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, March 1990.

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

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

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

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

[20] Free Software Foundation. Gnu license home page. http://www.gnu.org/licenses/licenses.html.

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

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

[23] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns, Elements of Reusable Object Oriented Software*. Addison-Wesley, 1994.

[24] M. A. Heroux, R. J. Hoekstra, and A. B. Williams. An object model for parallel numerical linear algebra computations. Technical report, Sandia National Laboratories, 2003. In preparation.

[25] Michael A. Heroux. Trilinos home page. http://software.sandia.gov/trilinos.

[26] Tamara G. Kolda and Roger P. Pawlowski. Nox home page. http://software.sandia.gov/nox.

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

[28] Xiaoye Li and James Demmel. SuperLU home page. http://crd.lbl.gov/ xiaoye/SuperLU/, 2003.

[29] Scott Meyers. *Effective C++*. Addison-Wesley, 1998.

[30] Scott Meyers. *Effective STL*. Addison-Wesley, 2001.

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

[32] Nathan C Myers. Traits: a new and useful template technique. *C++ Report*, June 1995.

[33] R. Pozo and B. Miller. SciMark 2.0. http://math.nist.gov/scimark2/.

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

[35] Padma Raghavan. DSCPACK home page. http://www.cse.psu.edu/ raghavan/Dscpack, 2003.

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

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

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

[39] John N. Shadid, Harry K. Moffat, Scott A. Hutchinson, Gary L. Hennigan, Karen D. Devine, and Andrew G. Salinger. MPSalsa: A finite element computer program for reacting flow problems part 1 - theoretical development. Technical Report SAND95–2752, Sandia National Laboratories, 1995.

[40] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystem Routines – EISPACK Guide*, volume 6 of *Lecture Notes in Computer Science*. Springer–Verlag, New York, second edition, 1976.

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

[42] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 2000.

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

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

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

[46] C. Tong and R. Tuminaro. ML2.0 Smoothed Aggregation User's Guide. Technical Report SAND2001-8028, Sandia National Laboratories, Albq, NM, 2000.

[47] Ray S. Tuminaro, Michael A. Heroux, Scott. A. Hutchinson, and J. N. Shadid. *Official Aztec User's Guide, Version 2.1*. Sandia National Laboratories, Albuquerque, NM 87185, 1999.

[48] Ray S. Tuminaro and Jonathan Hu. Ml home page. http://www.cs.sandia.gov/ tuminaro/ML_Description.html.

[49] Dimitri van Heesch. Doxygen home page. http://www.doxygen.org.

[50] P. Vanek, M. Brezina, and J. Mandel. Convergence of Algebraic Multigrid Based on Smoothed Aggregation. Technical Report 126, UCD/CCM, Denver, CO, 1998.

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

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

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

A Brief Overview of Some Object-Oriented Concepts

Much of the discussion in this document assumes some familiarity with object-oriented concepts and terminology. We realize that some readers may not be very familiar with these topics. Therefore, we provide this appendix to cover some of the basic topics, as we understand them and use them.

Object-oriented Programming

We use the term object-oriented programming (OOP) to refer to a philosophy of software engineering where procedures (called methods or functions) and data that are logically related are kept together in a single logical unit called a class. Although it is not always clear which data and methods belong in a given class, we can generally agree on basic associations. As an example, one obvious class for a solver framework is a Vector. For our purposes, we consider a vector object to have finite dimension and a basis. Therefore, it contains data that can be indexed. Some obvious vector operations are norms, dot products and vector updates. Vectors can also be multiplied by a linear operator, or more specifically by a matrix. However, we commonly put this kind of method in the matrix class because matrices tend to be more complicated objects and writing the method in the matrix class is easier.

Some of the strengths of OOP are a strong emphasis on the interaction of objects with each other, that is, on interfaces between classes. By focusing on interfaces we get a variety of benefits. First, a well-designed interface prescribes *what* should be done by a piece of software, not *how* it should be done. This fact, combined with the fact that a class owns its data, allows great flexibility in how methods are implemented. Even more importantly, once software is in use, OOP techniques give us flexibility to change the implementation of a class without changing the interface. Since a user only works with the interface (methods) of a class, we can change the implementation of a class without requiring any major change in the user code.

We have used this flexibility within the Trilinos Project. In particular, earlier versions of Petra classes were based on code from Aztec, which allowed us to get working versions of Petra very quickly. Over time we replace the Aztec code with implementations that offered more flexibility and features. However, the overall design of many of the Petra classes has remained fundamentally the same.

Some Key OOP Terms

Throughout this paper we have used a number of terms repeatedly, and sometimes interchangeably. In this section, we define these terms. Note that these terms (and many more) are discussed in great detail in books by Stroustrup [42], Gamma et. al. [23], Meyers [29, 30] and many others.

Virtual Function, Pure Virtual Function

Virtual functions (also called virtual methods) are functions defined on a base class that can be redefined in any derived class. When a derived class redefines a method from its base class, it is said to override that method. A pure virtual function is a virtual function that is declared but not implemented in the base class. Pure virtual functions *must* be overridden by derived classes, while “non-pure” virtual functions need not be overridden.

Abstract Class, Pure Virtual Class, Interface, Virtual Class

These four terms are used to describe classes that are incomplete, and can not be constructed directly. The first term is used to describe any class that has one or more pure virtual methods. The second two terms describe classes that have *no* executable code. These classes contain method prototypes only and cannot be constructed explicitly. The term pure virtual class tends to be associated with C++ programming while interface is formally defined in Java. We tend to use these two terms interchangeably. A virtual class, like an abstract class, is one which has some pure virtual methods (prototypes without code), but has some methods that have a default implementation (sometimes these implementations are written in terms of other virtual methods). These classes cannot be constructed explicitly either. All four of these class types must be inherited by a concrete class that implements the virtual methods, therefore implementing the interface.

Concrete Class, Implementation

In order for an abstract class to be used, some other class must provide an implementation of the undeveloped methods of the abstract class. This implementation class, often called a concrete class, provides an implementation of the abstract class interface. Generally the term concrete class can be used to describe any class that can be constructed, i.e., any class which contains no pure virtual methods.

Base Class, Derived Class, Specialization

A concrete class that implements an abstract class is said to be a derived class while the abstract class is called a base class. Unrelated to abstract and concrete classes, we also mention another form of derived class called specialization. One class is a specialization of another (base) class if it is a subset (or special case) of the base class. For example, given an existing matrix class, a vector class can be derived by constructing a matrix object with one column. In other cases a derived class *extends* the base class, providing methods from the base class as well as methods not in the base class.

Base Class, Polymorphism, Factory

An abstract class, and in fact any class containing virtual methods, can be implemented by multiple concrete classes. In this situation each concrete class can be used interchangeably to behave as an instance of the base class. This interchangeability of the concrete classes that implement a common base is referred to as polymorphism. For convenience, and to hide the details of concrete class construction, we often develop a function or class called a factory that can construct one of a number of concrete classes that have a common base class. Once an instance of the concrete class is constructed, the object is returned as an object of the base class type. In this way, the calling code (the scope in which the object is to be used) need not know what the concrete type of the object actually is.

Multiple Inheritance

Multiple inheritance describes the case where a single concrete class inherits more than one base class. This feature of the C++ language is utilized by several classes in the Epetra package. For example, Epetra_CrsMatrix is a concrete compressed row sparse matrix class that implements the abstract interface Epetra_RowMatrix as well as Epetra_DistObject, the interface specification for import and export operations in distributed-memory parallel environments. An instance of a class that implements multiple base classes may be passed as an argument where any of those base classes is expected.

Templates, Traits

C++ classes (and stand-alone functions) may be written in terms of one or more generic type parameters. Such classes are called templates. An example could be a matrix class that may be instantiated with any type of coefficient data – double-precision floating-point numbers, integers, etc. In a templated class, the implementation code doesn't know the type of the template parameter. In many cases

this is a severe limitation, for instance if a templated vector class is to call through to BLAS functions it is necessary to distinguish between calling 'dnrm2', 'snrm2' or 'dznrm2'. Another example is the need to associate different MPI data-types with template parameters. This limitation can be addressed using a template technique called traits [32]. Traits are essentially a way of associating a set of types and methods with the specific type used to instantiate the template. This is accomplished by using a secondary template which has a specialization for each possible type that is to be supported. This secondary template is only used internally, and is not exposed to the end user.