SAND REPORT

SAND2003-2952
Unlimited Release
August 2003, last updated September 2007

Trilinos Users Guide[a]

Michael A. Heroux and James M. Willenbring

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.

[a]For Trilinos Release 8.0

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

SAND2003-2952
Unlimited Release
August 2003, last updated September 2007

Trilinos Users Guide[†]

Michael A. Heroux and James M. Willenbring
Scalable Algorithms 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. A new software capability is introduced into Trilinos as a *package*. 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.

The Trilinos Users Guide is a resource for new and existing Trilinos users. Topics covered include how to configure and build Trilinos, what is required to integrate an existing package into Trilinos and examples of how those requirements can be met, as well as what tools and services are available to Trilinos packages. Also discussed are some common practices that are followed by many Trilinos package developers. Finally, a snapshot of current Trilinos packages and their interoperability status is provided, along with a list of supported computer platforms.

[†]For Trilinos Release 8.0

Acknowledgments

The authors would like to acknowledge the support of the ASCI and LDRD programs that funded development of Trilinos and recognize all of our fellow Trilinos contributors: Chris Baker, Teri Barth, Ross Bartlett, Paul Boggs, Erik Boman, Todd Coffey, Jason Cross, David Day, Clark Dohrmann, David Gay, Michael Gee, Bob Heaphy, Ulrich Hetmaniuk, Robert Hoekstra, Russell Hooper, Vicki Howle, Jonathan Hu, Tammy Kolda, Kris Kampshoff, Sarah Knepper, Joe Kotulski, Richard Lehoucq, Kevin Long, Joe Outzen, Roger Pawlowski, Mike Phenow, Eric Phipps, Marzio Sala, Andrew Rothfuss, Andrew Salinger, Paul Sery, Paul Sexton, Chris Siefert, Ken Stanley, Heidi Thornquist, Ray Tuminaro and Alan Williams.

Contents

Figures

Tables

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.
Didasko	The tutorial of Trilinos. Offers a quick introduction to several Trilinos packages. It contains a printable PDF document and a variety of well-commented browsable examples that illustrate how to use Trilinos. Didasko contains examples for the following Trilinos packages: Teuchos, Epetra, EpetraExt, TriUtils, Galeri, AztecOO, IFPACK,ML, NOX, Anasazi, TPetra.
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.
Amesos	The Direct Sparse Solver Package in Trilinos. The goal of Amesos is to make AX=B as easy as it sounds, at least for direct methods. Amesos provides clean and consistent interfaces to several popular third party libraries.
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 [8]. 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.
Claps	Claps is a collection of domain decomposition preconditioners and solvers.
Epetra	See description under ”Petra”.

EpetraExt A set of extensions to Epetra. To allow Epetra to remain focused on it's primary functionality as a Linear Algebra object support, EpetraExt was created to maintain additional support for such capabilities as transformations (permutions, sub-block views, etc.), coloring support, partitioning (Zoltan), and I/O.

Galeri Contains a suite of utilities and classes to generate a variety of (distributed) linear systems. Galeri's functionalities are very close to that of the MATLAB's gallery() function.

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.

Isorropia A repartitioning/rebalancing package intended to assist with redistributing objects such as matrices and matrix-graphs in a parallel execution setting, to allow for more efficient computations. Isorropia is primarily an interface to the Zoltan library, but can be built and used with minimal capability without Zoltan.

Jpetra See description under "Petra".

Kokkos A collection of the handful of sparse and dense kernels that determine much of the performance for preconditioned Krylov methods. In particular, it contains function class for sparse matrix vector multiplication and triangular solves, and also for dense kernels that are not part of the standard BLAS. Kokkos is not intended as a user package, but to be incorporated into other packages that need high performance kernels.

Komplex Complex linear equation solver using equivalent real formulations [1], 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.

Moertel Supplies capabilities for nonconforming mesh tying and contact formulations in 2 and 3 dimensions using Mortar methods.

MOOCHO MOOCHO (Multifunctional Object-Oriented arCHitecture for Optimization) is designed to solve large-scale, equality and inequality nonlinearly constrained, non-convex optimization problems (i.e. nonlinear programs) using reduced-space successive quadratic programming (SQP) methods.

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.

Pliris An object-oriented interface to a LU solver for dense matrices on parallel platforms. These matrices are double precision real matrices distributed on a parallel machine.

PyTrilinos A set of python wrappers for selected Trilinos packages. Allows a python programmer to dynamically import Trilinos packages into a python script or the python command-line interpreter, allowing the creation and modification of Trilinos objects and the execution of Trilinos algorithms, without the need to constantly recompile.

RTOp RTOp (reduction/transformation operators) provides the basic mechanism for implementing vector operations in a flexible and efficient manner.

Rythmos A transient integrator for ordinary differential equations and differential-algebraic equations with support for explicit, implicit, one-step and multi-step algorithms. Aimed at supporting operator-split algorithms, multi-physics applications, block linear algebra, and adjoint integration.

Sacado A set of automatic differentiation tools for C++ applications. Provides templated classes for forward, reverse and Taylor mode automatic differentiation.

Stratimikos Contains a unified set of Thyra-based wrappers to linear solver and preconditioner capabilities in Trilinos. Can also be used for unified testing of linear solvers and preconditioners.

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.

Thyra A set of interfaces and supporting code that defines basic interoperability mechanisms between different types of numerical software. The foundation of all of these interfaces are the mathematical concepts of vectors, vector spaces, and linear operators. All other interfaces and support software is built on the basic operator/vector interfaces.

Tpetra See description under ”Petra”.

TriUtils A package of utilities for other Trilinos packages.

WebTrilinos A scientific portal; a web-based environment to use several Trilinos packages through the web.

1 Introduction

The Trilinos Project is an effort to facilitate the design, development, integration and ongoing support of mathematical software libraries. Trilinos also provides a set of core utility libraries that provide common vector, graph and matrix capabilities, as well as a common abstract interface for applications to access any appropriate Trilinos package.

The overall objective of Trilinos is to promote rapid development and deployment of high-quality, state-of-the-art mathematical software in an environment that supports interoperability of packages while preserving package independence.

The Trilinos Users Guide is meant to assist new and existing Trilinos users. Topics covered include how to configure, build, and install Trilinos, as well as how to run

tests to insure proper installation. In addition, issue reporting and how to sign up for and use Trilinos mail lists are discussed. Finally, directions for obtaining Trilinos itself and documentation for individual Trilinos packages are provided.

For a higher-level view of the Trilinos project, please see An Overview of Trilinos [4]. The Trilinos Tutorial is useful resource for new and existing Trilinos users [6]. A document is also available specifically for Trilinos developers; please see the Trilinos Developers Guide [5]. The Developers Guide contains both tutorial and general reference material.

An overview of current Trilinos package capabilities can be found on the Trilinos website at `http://trilinos.sandia.gov/capabilities.html` . A summary of the packages included in every Trilinos release is also on the website at `http://trilinos.sandia.gov/releases.html` .

1.1 Typographical Conventions

Our typographical conventions are found in Table 1.

Notation	Example	Description
`Verbatim text`	`../configure --enable-mpi`	Commands, directory and file name examples, and other text associated with text displayed in a computer terminal window.
`CAPITALIZED_TEXT`	`CXXFLAGS`	Environment variables used to configure how tools such as compilers behave.
`<text in angle brackets>`	`../configure <user parameters>`	Optional parameters.

Table 1. Typographical Conventions for This Document.

2 Getting Started

This chapter covers some of the basics that a user will need to know when beginning to use Trilinos. Specifically, we address how to obtain, configure, build, install and test Trilinos. We will also discuss how to link to Trilinos libraries. Although this user guide is current at the time of printing, the Trilinos Project is under active development. For the most up-to-date information, please visit the online Trilinos Home Page. The most accurate and complete information will be kept at this site.

Tip: Check out the Trilinos Home Page at `http://trilinos.sandia.gov`. At the website a user can:

- Find up-to-date information about Trilinos and its packages.
- Download Trilinos.
- Sign up for mail lists.
- Find documentation.
- Submit a bug report.

2.1 Obtaining a Copy of Trilinos

Trilinos can be obtained in two different ways. Users outside of Sandia most commonly download Trilinos in the form of a tarball from the Trilinos website at `http://trilinos.sandia.gov/download/` . On most systems, the following commands can be used to expand the tarball:

Command: `gunzip trilinos-8.0.1.tar.gz`

Command: `tar xf trilinos-8.0.1.tar`

For versions of Trilinos other than 8.0.1, the file names in the above commands will need to be adjusted.

Users who download the tarball from the website may skip to Section 2.2.

Users may be able to obtain a copy of Trilinos via the Trilinos CVS repository. Access to the Trilinos repository is granted only to co-developers, Sandians, and in select special cases.

To access the repository, an account on software.sandia.gov is required. In ad-

dition, the user account must be a in the “cvs” group on software.sandia.gov. To request an account, send a note to `trilinos-help@software.sandia.gov` . The following two environment variables must be set to access the repository:

Command: `CVSROOT :ext:your_user_name@software.sandia.gov:/space/CVS`

Command: `CVS_RSH ssh`

Replace “your_user_name” with your user name on software.sandia.gov.

To checkout a working copy of the development branch of Trilinos in the current directory, type

Command: `cvs checkout Trilinos`

To checkout a working copy of a release branch of Trilinos in the current directory, type

Command: `cvs checkout -r name_of_release_branch Trilinos`

Replace “name_of_release_branch” with the name of the release branch. The name of the current release branch can be obtained by sending a note to `trilinos-help@software.sandia.gov` .

To checkout a working copy of the development version of only one Trilinos package in the current directory, type

Command: `cvs checkout <package_name>`

Replace “package_name” with the name of the package. Please note that most packages have dependencies on other Trilinos packages.

If the machine that is downloading Trilinos from software.sandia.gov has gzip available, it is possible to speed up the checkout proces using the -z option. For example, to checkout all of Trilinos using the most aggressive compression, type

Command: `cvs -z 9 checkout Trilinos`

For those not familiar with CVS, a brief discussion covering some of the most

common CVS commands is available in Section 3.2. For a more complete listing of CVS commands, see the GNU CVS Home Page [2].

2.2 Recommended Build Directory Structure

Via Autoconf and Automake the Trilinos configuration facilities provide a great deal of flexibility for configuring and building the existing Trilinos packages. However, unless a user has prior experience with Autotools, we very strongly recommend the following process to build and maintain local builds of Trilinos.

To start, we defined two useful terms:

- Source tree - The directory structure where source files are found. A source tree is obtained by expanding a distribution tar ball, or by checking out a copy of the Trilinos repository.
- Build tree - The directory structure where object and library files, as well as executables are located.

Although it is possible to run `./configure` from the source tree (in the directory where the configure file is located), we recommend separate build trees. The greatest advantage to having a separate build tree is that multiple builds of the libraries can be maintained from the same source tree. For example, both serial and parallel libraries can be built. This approach also eliminates problems with configuring in a 'dirty' directory (one that has already been configured in).

Key Point: ... we recommend separate build trees ... multiple builds of the libraries can be maintained from the same source tree ... problems with configuring in a 'dirty' directory (are eliminated) ...

Setting up a build tree is straight-forward. Figure 1 illustrates the recommended layout. First, from the highest directory in the source tree (Trilinos for a repository copy, trilinos-8.0.1 for a distribution), make a new directory - for an MPI build on a Linux platform, a typical name is `LINUX_MPI` . Finally, from the new directory, type

Command: `../configure --with-mpi-compilers`

(Note that various configure options might be necessary, see Section 2.3 for details.) Finally, type

Command: `make everything`

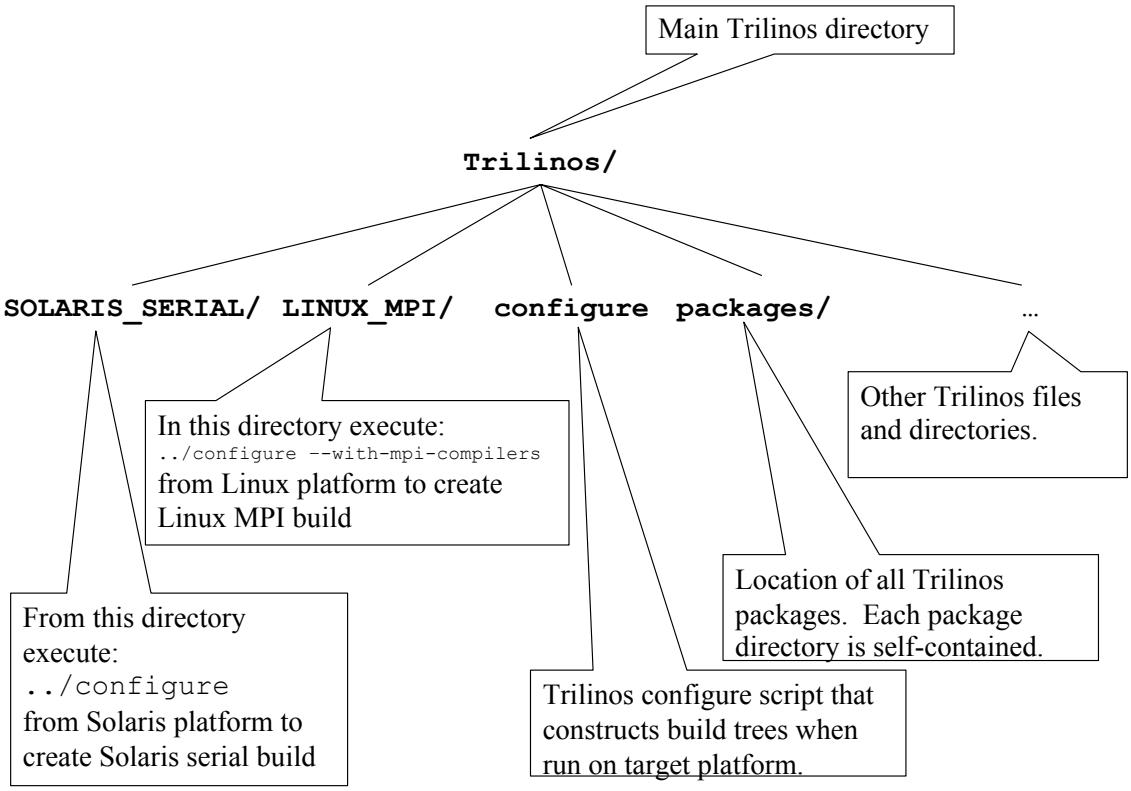

Figure 1. Recommended Layout for Trilinos Build Directories

In summary:

```
cd Trilinos
mkdir LINUX_MPI
cd LINUX_MPI
../configure --with-mpi-compilers
make everything
```

At this point, the MPI version of Trilinos on a Linux platform is built and completely contained in the `LINUX_MPI` directory. No files outside this directory have been modified. This procedure can be repeated for any number of build targets.

Note: Although we recommend the above location for build trees, they can be set up anywhere.

Note: `make everything` builds all of the libraries, tests, and examples that were enabled at configure time. For more information about make targets, see Section 2.5

2.3 Configuring Trilinos

The most common issue encountered when configuring Trilinos is that it is nearly impossible to determine what caused configure to fail based on the standard output. If the output from configure is inadequate, look at the config.log file (in the buildtree) for the package that failed to configure properly.

Key Point: . . . to determine what caused configure to fail . . . look at the config.log file . . .

To determine which package failed to configure, look at the bottom of the output from the `configure` command. One of the last lines will say something like:

```
configure: error: /bin/sh '../../../packages/epetra/configure'
failed for packages/epetra
```

This particular error indicates to look in `packages/epetra/config.log` .

To configure from a remote build tree, simply run the configure script in source tree from the root of the build tree. In the example above, cd to the SOLARIS_SERIAL directory and type

Command: `../configure <configure options>`

A detailed list of configure options can be seen by typing

Command: `./configure --help=recursive`

from the top level of the source tree. This will display the help page for the Trilinos level as well as all Trilinos packages that use Autoconf and Automake. The output from this command is quite extensive. To view the help page for an individual package, cd to the home directory for the package in the source tree and type

Command: `./configure --help`

This command will also display the help page for Trilinos level options when used from the Trilinos home directory in the source tree.

Many of the Trilinos configure options are used to describe the details of the build. For instance, serial or mpi, all of the packages, or just a proper subset.

To configure for serial libraries, no action is necessary, but to configure for parallel libraries, a user must append appropriate arguments to the configure invocation line as described in "Trilinos Configuration Options", section 2.4.

Also, to build the default set of Trilinos libraries, no action is necessary, but to exclude a package that is built by default, AztecOO for example, append `--disable-aztecoo` to the configure invocation line. Similarly, to include a package that is not currently built by default, Komplex for example, append `--enable-komplex` to the configure invocation line. Most Trilinos packages are not built by default; virtually all users will have to enable additional packages.

Users are strongly encouraged to build only the packages that are necessary because configuring and building can take a long time. It is well worth the time to look at which packages are build by default enable and disable packages as necessary.

Key Point: ... build only the packages that are necessary because configuring and building can take a long time.

It is recommended that users always configure from the Trilinos level and use `--disable-<package>` as required, rather than trying to configure from a lower level. To see which packages are built by default and which ones aren't, simply cd to the Trilinos home directory and type

Command: `./configure --help`

NOTES:

1. **Enabling/Disabling package builds:** The configure process is set up to detect when a `--disable-<package>` option would break a package dependency. For example, Ifpack depends on Epetra, so if a user wants to build Ifpack, but types `--disable-epetra` , Epetra will be configured and built anyway.

2. **Installing libraries and header files:** To install libraries and header files in a particular location, use `--prefix=<dir>` on the configure line. If this option is used, libraries will be located in `<dir>/lib` and header files in `<dir>/include/<package>` .

3. **Providing additional information to Autotools:** Although Autotools will try to determine all configuration information, the user must provide anything that Autotools needs and cannot find. Also, if Autotools selects, for example, the wrong BLAS library by default, the user must indicate which BLAS library to use. Other issues such as standards non-compliance are also commonly dealt with using configure options. If all required libraries (often the BLAS and LAPACK) are located in standard places and no special compiler flags are required, try configuring without providing additional information.

4. **Sample configure invocation scripts:**

   Sample configure invocation scripts for a wide variety of platforms can be found in `Trilinos/sampleScripts` . These scripts are generally named using the following convention: `arch_comm_machine` . For example, `sgi64_mpi_atlantis` .

   Key Point: Sample configure invocation scripts for a wide variety of platforms can be found in `Trilinos/sampleScripts` .

   Note that these scripts are examples only and are primarily useful for the values of options such as `LDFLAGS` , `CPPFLAGS` , and `CXXFLAGS` . Do not expect to be able to find a script that can be used without modification; try to find a script for a similar machine to use as a guide. Users will also have to enable the correct set of packages based on their individual needs.

   The scripts in the repository are not always up to date. If a user submits a script for a machine that few Trilinos developers have an account on, that script may become obsolete if it is not updated by the user who submitted it.

   Users who create scripts for other machines are encouraged to check them into the repository for the benefit of other users. Users who do not have access to the repository can send scripts to trilinos-help@software.sandia.gov.

The following is an example configure invocation script for an SGI machine:

```
../configure --enable-mpi --with-mpi-libs=-lmpi \
--with-cflags=-64 --with-fflags=-64 \
--with-cxxflags="-64 -LANG:std  -LANG:ansi-for-init-scope=ON \
-ptused -DMPI_NO_CPPBIND" \
LDFLAGS=" -64 -L/usr/lib64/mips4/r10000 -L/usr/lib64/mips4 \
-L/usr/lib64 " \
--enable-epetraext --enable-new_package \
--disable-komplex --enable-thyra
```

2.4 Trilinos Configuration Options

The following options apply to all Trilinos packages unless an option doesn't make sense for a particular package (for example, a package that does not include any Fortran code will not be sensitive to `F77=g77`), or otherwise noted. For options specific to an individual package, cd to the home directory of the package and type

Command: `./configure --help`

Basic Options

- `--with-gnumake`

  Gnu's make has special functions we can use to eliminate redundant paths in the build and link lines. **Use this option if you use GNU make to build Trilinos.** This requires that perl is in your path or that you have specified the perl executable with `--with-perl=<perl executable>` . For example `--with-perl=/usr/bin/perl` .

- `--with-blas=<lib>`

  Specify which BLAS library to use. For example `--with-blas=/usr/path/lib/libblas.a` .

- `--with-lapack=<lib>`

  Specify which LAPACK library to use. For example `--with-lapack="-L/usr/path/lib/ -llapack"` .

- `--enable-examples`

  Build examples for all Trilinos packages. By default, this option is enabled. Note that after building the libraries, `make examples` can be used to compile all of the examples that were enabled at configure time.

- `--enable-tests`

  Build tests for all Trilinos packages. By default, this option is enabled. Note that after building the libraries, `make tests` can be used to compile all of the tests that were enabled at configure time.

- `--enable-debug`

  (NOX only.) This turns on compiler debugger flags. It has not been fully tested. As an alternate, specify CXXFLAGS on the configure line.

- `--enable-opt`

  (NOX only.) This turns on compiler optimization flags. It has not been fully tested. As an alternate, specify CXXFLAGS on the configure line.

- `--with-cppflags`

  Specify additional preprocessor flags (e.g., "-Dflag -Idir")

- `--with-cxxflags`

  Specify additional C++ flags

- `--with-ldflags`

  Specify additional linker flags (e.g., "-Ldir")

- `--with-ar`

  Specify a special archiver command, the default is "ar cru".

Influential Environmental Variables

- `CC`

  C compiler command.

- `CFLAGS`

  C compiler flags.

- `CXX`

  C++ compiler command.

- `CXXFLAGS`

  C++ compiler flags.

- `LDFLAGS`

  Specify linker flags.

- `CPPFLAGS`

  C/C++ preprocessor flags.

- `CXXCPP`

  C++ preprocessor.

- `F77`

  Fortran 77 compiler command.

- `FFLAGS`

  Fortran 77 compiler flags.

MPI-Related Options

- `--enable-mpi`

  Enables MPI mode. Defines HAVE_MPI in the (Package)_Config.h file. Will test for the ability to preprocess the MPI header file and may test ability to link with MPI. This option is rarely necessary as many of the below options also turn MPI on.

- `--with-mpi-compilers`

  Sets CXX = mpicxx (or mpiCC if mpicxx not available), CC = mpicc and F77 = mpif77. Automatically enables MPI mode. To use compilers other than these, specify MPI locations with the below options. If none of these options are necessary, use `--enable-mpi` to enable MPI mode. In this case, CXX, CC, and F77 have to be set if the correct compilers are not chosen by default.

- `--with-mpi=MPIROOT`

  Specify the MPI root directory. Automatically enables MPI mode. If this option is set, `--with-mpi-incdir` and `--with-mpi-libdir` should not be used. `--with-mpi` is a shortcut for setting
  `--with-mpi-libdir=MPIROOT/lib` and
  `--with-mpi-incdir=MPIROOT/include` .

- `--with-mpi-libdir=DIR`

  Specify the MPI libraries location. Defaults to MPIROOT/lib if `--with-mpi` is specified. If multiple directories must be specified, try
  `--with-ldflags="-L<dir1> -L<dir2>"` instead.

- `--with-mpi-libs="LIBS"`

  Specify the MPI libraries. Defaults to `"-lmpi"` if either `--with-mpi` or `--with-mpi-libdir` is specified.

- `--with-mpi-incdir=DIR`

  Specify the MPI include files location. Defaults to `MPIROOT/include` if `--with-mpi` is specified. If multiple directories must be specified, try `--with-cppflags="-I<dir1> -I<dir2>"` instead.

Developer-Related Options

- `--enable-maintainer-mode`

  Enable make rules and dependencies not useful (and sometimes confusing) to the casual installer.

2.5 Building and Installing Trilinos

If the configure stage completed successfully, just type

Command: `make`

to compile the libraries. To compile the tests and examples, type

Command: `make tests`

and

Command: `make examples`

For more about how to run the installation test suite, see Section 2.8. Finally, if `--prefix` was specified, type

Command: `make install`

to install libraries and headers.

Note that any code that links to Trilinos libraries must define `HAVE_CONFIG_H`.

Key Point: ... any code that links to Trilinos libraries must define `HAVE_CONFIG_H`.

Below is a listing of the make targets that will be most useful to users:

- make - Builds just the libraries
- make install - Installs just the libraries
- make examples - Makes just the examples (assumes libraries are built)
- make install-examples - Install the examples (assumes libraries are installed)
- make tests - Makes just the tests (assumes libraries are built)
- make everything - Builds libraries, tests, and examples
- make install-everything - Installs libraries and examples

2.6 Configure, Make and Install: Common Problems

Provided below is a list of common problems associated with configuring, building and installing Trilinos. A more complete FAQ list may be found online at

`http://trilinos.sandia.gov/faq.html` .

Key Point: A more complete FAQ list may be found online...

- Any code that links to Trilinos libraries must define `HAVE_CONFIG_H` .
- It is difficult to build Trilinos with different compiler versions. For example, if using g++ 4.1.1, it is best to use gcc 4.1.1 and gfortran 4.1.1. On some systems, multiple sets of compilers are available. If, for instance, there is one GCC 4 compiler set and one GCC 3 compiler set, it is possible that configure will pull the g++ and gcc compilers from GCC 4 and the g77 compiler from GCC 3.

  This will usually result in a gxx_personality error in the `packages/teuchos/config.log` file. Another possible clue in the `config.log` file is paths for linking to libraries that refer to multiple compiler sets. For example, `-L/usr/lib/gcc/x86_64-redhat-linux/4.1.1` and `-L/usr/lib/gcc/x86_64-redhat-linux/3.3.2` .

  The easiest way to avoid these problems is to explictly specify which compilers you want to use in the list of configure arguments. For example, `CXX=/path/g++` , `CC=/path/gcc` , and `F77=/path/gfortran` .

- Trilinos should be built with the `--with-gnumake` option whenever GNU make is used to build Trilinos. On most systems, ”make” will point to GNU make. It is easy to check, simply type `man make` on any UNIX-like machine. When building a large number of packages without using the gnumake option, the following error is common:

  ```
  execv: Argument list too long
  ```

  If GNU make is not available and this error is encountered, it will be necessary to disable the test or example that was linking when the error occurred.

- Do not attempt to specify optimization flags using the `--with-cxxflags`, `--with-cflags`, or `--with-fflags` options. Use `CXXFLAGS`, `CFLAGS` and `FFLAGS` instead. Use `--with-cxxflags`, `--with-cflags`, and `--with-fflags` to specify flags that are to be used in addition to the default optimization flags.

- When creating a configure invocation script, be sure to use line continuation characters properly. The characters should be at the end of every line, except the last line, and should not be followed by any spaces.

- To verify that the entire configure invocation script has been parsed by Autoconf, open the `config.status` file in the top level of the build tree and grep for the string ”with options”. Here you will find all of the options that Autoconf pulled from the invoke configure script.

- Autoconf cannot detect most spelling mistakes in configure invocation scripts.

- When experiencing problems during the make phase, it is often useful to `make clean` before attempting to `make` again. Sometimes it even helps to blow away the entire build tree and start over.

- When building with LAM under RH9 Linux, configure complains that it cannot find mpi++.h. The message in the config.log file is:

  ```
  /usr/include/mpi.h:1064:19: mpi++.h: No such file or directory
  ```

  The following modified configure invocation works:

  Command: `../configure --enable-mpi CXX="mpiCC -DLAM_BUILDING"`

- The build process will fail on OSX if “DropZip” is used to unzip the Trilinos tarball. This utility truncates long file names.

2.7 Tips for Making the Configure, Make, and Install Processes More Efficient

Trilinos has grown to become a large piece of software. Not surprisingly, it can take a very long time to configure and build all of Trilinos. Below are some tips for speeding up the process:

- Only build the Trilinos libraries that are necessary.

  An easy way to do this is to use the –disable-default-packages option. This option allows users to easily specify exactly which packages should be built. Packages that enabled packages are dependent on will be turned on automatically, so don't shy away from disabling all packages that are not used directly. If a package configures and builds that was not enabled explicitly, keep in mind that a package that was enabled probably depends on that package.

- Cache configure results.

  Adding `--cache-file=config.cache` to the list of configure arguments will store the results of many of the tests that configure performs in a file called `config.cache` , located in the top level of the build tree.

  Using this option will allow the first package that is configured to store many test results that can be used by all subsequent packages. If it is later necessary to reconfigure with a different set of packages, all packages (including the first) can use these cached values. If configure options are changed that are associated with configure test results that have already been cached, or a relevant environment variable is changed, it is necessary to remove the `config.cache` file before reconfiguring. Failing to do so can cause the changes to be ignored.

  When configuring even a handful of packages, during the first configure a speedup of greater than two is attainable and a speedup of greater than three is possible for subsequent configures.

- Decrease build time on some machines by creating multiple jobs.

  If `-j` (jobs) is a valid option for `make` , specifing the `-j` option with a value of two times the number of processors that the machine has will typically result in a faster build process. For example, on a dual processor machine, try replacing `make` with

  Command: `make -j 4`

  during the build step.

On a single processor machine, the speedup is minimal; on a machine with multiple processors, the speedup can be quite significant. For example, speedups of 1.73 and 2.45 were observed on a dual processor machine and a four processor machine, respectively. Using two times the number of processors for the argument to the jobs option is only a suggestion based on observed performance; those who are interested in achieving optimal performance are encouraged to experiment with various values and to report their findings to `trilinos-help@software.sandia.gov` . The `-j` can also be passed without a corresponding value, in other words

Command: `make -j`

When used in this way, the number of jobs created is unlimited. For machines with a large number of processors this appears to work in some situations, but machines with fewer than four processors tend to get bogged down with overhead.

2.8 Testing the Build

There are two ways that users can run one or more sanity tests after building Trilinos.

The runtests utility, which is a component of the Trilinos test harness, can be accessed via the runtests-serial and runtests-mpi make targets. To run the test suite (the tool will only attempt to run tests that were built), simply do a make first and then type:

Command: `make runtests-serial`

To run the test suite for a single package, cd to `package/test` (replace ”package” with the name of a particular package) before running the above command.

The runtests-mpi target is very similar, but expects to you supply the variable `TRILINOS_MPI_GO` , either in the environment or as an argument to make. Here is an example of how to call the runtests-mpi make target:

Command: `make runtests-mpi TRILINOS_MPI_GO="'mpiexec -np '"`

Alternatively, the following set of arguments can be used:
`TRILINOS_MPI_MAX_PROC=2 TRILINOS_MPI_GO="' /bin/linux/mpirun -np '"` . Note that in either case, the `TRILINOS_MPI_GO` argument must be double quoted.

A simple alternative that provides a quick santity check is to execute one or more Trilinos tests manually. For example, if Epetra was one of the packages that was configured and built in the LINUX_MPI build tree, the Epetra tests were not disabled through the use of configure options, and lam mpi is installed, perform the following series of steps to run the CrsMatrix tests:

```
cd Trilinos/LINUX_MPI/packages/epetra/test/CrsMatrix
lamboot
mpirun -np 3 ./CrsMatrix_test.exe
echo $status
lamhalt
```

For a serial build, the process is shorter:

```
cd Trilinos/LINUX_SERIAL/packages/epetra/test/CrsMatrix
./CrsMatrix_test.exe
echo $status
```

In the serial case, the status returned will be zero if the tests passed, nonzero if the tests failed. The return codes for mpi executables is less reliable, so it is necessary to check the output of the test. Negative non zero error codes are errors, positive codes are warnings. Epetra has many other tests in subdirectories that are peers to the CrsMatrix directory listed above. There are also examples that are located in the example directory that is a peer to the test directory. Most other packages have a number of tests and or examples that can be run using the same basic process.

3 Ongoing Use and Support

In this section we discuss how to report issues (bugs) and how to sign up for mail lists.

3.1 Reporting Bugs

Feature and issue reports are tracked using Bugzilla [7]. Bugzilla can be found on the web at `http://software.sandia.gov/bugzilla` . A Bugzilla account is necessary for submitting bugs through Bugzilla. Generally only Trilinos developers and collaborators are able to get a Bugzilla account. Those interested can sign up

at the website. Users who do not have a Bugzilla account should send bug reports to `Trilinos-Bugs@software.sandia.gov` .

Regardless of how an issue is submitted, the bug report should be filled out with as much detail as possible. Specifically, be sure to include which version of Trilinos the bug was discovered in and which platform(s) and compiler(s) the bug is associated with. Also include any specific error messages and any additional information that can be provided.

Two commonly underutilized features of Bugzilla are the ability to assign a priority and a severity to a bug. Taking advantage of these features helps users more clearly express their concerns and helps developers organize tasks. Below are recommended usage guidelines for the priority and severity settings for Bugzilla bugs that are filed against Trilinos.

Severity: Severity should be determined by the person who files the bug (reporter) and should be based solely on how severe the reporter perceives the bug to be at the time the bug is filed. Considerations for selecting a proper severity include whether or not the bug is currently directly impeding progress, and if it has the potential to impede progress in the future. There is also an important distinction between enhancements and faults in existing functionality. Severity can be updated if conditions warrant a change. Possible severity settings include blocker, critical, major, normal, minor, trivial, and enhancement. The severity of a bug is not necessarily related to its priority.

Priority: Priority should be determined by the owner of the bug. Priority can be chosen based on the relative importance of other existing bugs, as well as on how important the bug is to Trilinos and Sandia. When filling out a bug report, users can leave the bug at the default priority, or may try to guess the priority that the bug owner will select.

NOTE: In the context of Bugzilla, “bug” can refer not only to an error in existing code, but also to a desired enhancement. For example, a bug report should be submitted to Bugzilla to report a segmentation fault that occurs when using an existing Ifpack preconditioner, and a bug report should also be submitted to request a new Ifpack preconditioner. “Issue” and “bug” are used interchangeably in this discussion of Bugzilla.

3.2 Signing Up for Mail Lists

Email lists are maintained for Trilinos as a whole and for each package through Mailman [3]. This tool can be found on the web at

`http://software.sandia.gov/mailman/listinfo` . Those interested in signing up for one or more lists may do so at the website. A digest mode is available for those who wish to receive a daily summary of list activity. Non-Sandians are able to sign up for the "Users" and "Announce" lists. Sandians should keep this in mind when posting to these lists.

The example mailing lists mentioned below are to be used for issues relating to all of Trilinos. The names for the lists pertaining to individual packages follow the same naming scheme, simply replace "Trilinos" with the name of the package. For example the list for Trilinos users is called Trilinos-Users and the email address is `trilinos-users@software.sandia.gov` The list for Epetra users is called Epetra-Users and the associated email address is `epetra-users@software.sandia.gov`

Tip: While those who use any individual Trilinos package are also "Trilinos users", the lists are not set up to recognize this. In other words, those who subscribe to the Epetra-Users mailing list do not necessarily form a subset of those who subscribe to the Trilinos-Users mailing list. This is also true of all other list types. Keep this in mind when subscribing and posting to lists.

- Trilinos-Announce `trilinos-announce@software.sandia.gov`
  All Trilinos release announcements and other major news.
- Trilinos-Users `trilinos-users@software.sandia.gov`
  List for Trilinos Users. General discussions about the use of Trilinos.

References

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

[2] Free Software Foundation. Gnu CVS Home Page. http://www.gnu.org/software/cvs, 2004.

[3] Free Software Foundation. Gnu mailman home page. http://www.gnu.org/software/mailman/mailman.html, 2004.

[4] 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. An Overview of Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories, 2003.

[5] Michael A. Heroux, James M. Willenbring, and Robert Heaphy. Trilinos Developers Guide. Technical Report SAND2003-1898, Sandia National Laboratories, 2003.

[6] Marzio Sala, Michael A. Heroux, and David M. Day. Trilinos Tutorial. Technical Report SAND2004-2189, Sandia National Laboratories, 2004.

[7] The Mozilla Organization. Mozilla Bugzilla Home Page. http://www.mozilla.org/projects/bugzilla, 2004.

[8] 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.

Commonly Used CVS Commands

To access the Trilinos CVS repository, an account on software.sandia.gov is required. To request an account, send a note to `trilinos-help@software.sandia.gov`. The following two environment variables must be set to access the repository:

Command: `CVSROOT :ext:your_user_name@software.sandia.gov:/space/CVS`

Command: `CVS_RSH ssh`

(Replace "your_user_name" with your user name on software.sandia.gov.)

Below is a brief description of the CVS commands that are most commonly used by Trilinos users. For a more complete listing of CVS commands, see the GNU CVS Home Page [2].

- **Checking Out a Working Copy:** To checkout a working copy of the development branch of Trilinos in the current directory from the CVS repository, type

  Command: `cvs checkout Trilinos`

  To checkout a working copy of only one package of Trilinos in the current directory, type

  Command: `cvs checkout <package_name>`

  (Replace "package_name" with the name of the package.

  To checkout a different branch or a tagged version of Trilinos, type

  Command: `cvs checkout -r <name_of_branch_or_tag> Trilinos`

- **Updating a Working Copy:** To update after a version has been obtained use the `cvs update` command. First, cd to the directory that is to be updated (often the Trilinos root directory). Then type:

  Command: `cvs -q update -dP`

  The "-q" option means "be somewhat quiet". Try an update without the "-q" to see exactly what the option does.

The “-d” option means to get any new directories. For example, if a new package is added to the repository, but the “-d” option is not used, that new package will never appear in the working copy. However, the first time that the “-d” option is used, all of the new package directories will be downloaded, and from that time on, all CVS updates will update the directories that were downloaded. It is good practice to include this option for every CVS update.

Finally the “-P” option “prunes” empty directories. This helps to keep the directory structure from getting more cluttered than it needs to. For example, the old “petra” and “tsf” packages were removed from the repository, but the directory structures will remain if this option is not specified. If an empty directory is needed, simply issue one update command without the “-P” and the empty directories will be restored.