



Getting Started With Trilinos

Alexander Heinlein¹ Matthias Mayr²

September 12, 2022

¹Delft University of Technology

²Universität der Bundeswehr München

II. Introduction to Trilinos

Disclaimer

The following slides will give a brief overview over the software package TRILINOS. It is far from complete, but on the final slides, some *references to additional introductory material and tutorials will be given.*

What is Trilinos?

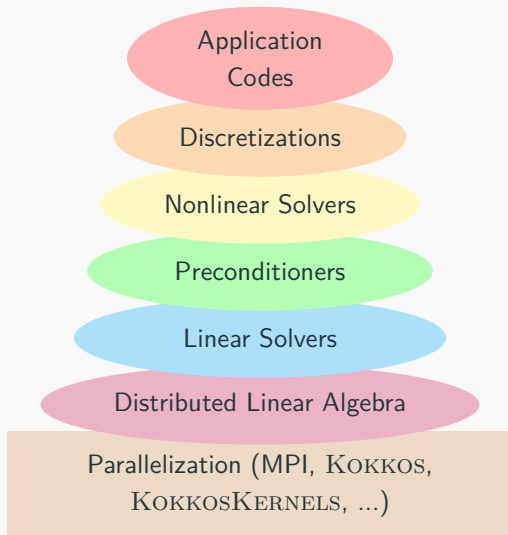


An Open-Source Library of Software for Scientific Computing

Mission statement¹: *“The TRILINOS Project is an effort to facilitate the design, development, integration, and ongoing support of mathematical software libraries and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems on new and emerging high-performance computing (HPC) architectures”.*



Layers of a Trilinos-based application



Why using Trilinos?

Wide range of functionality (organized in 5 *product areas*)

Data services	Vectors, matrices, graphs and similar data containers, and related operations
Linear and eigen-problem solvers	For large, distributed systems of equations
Nonlinear solvers and analysis tools	Includes basic nonlinear approaches, continuation methods and similar
Discretizations	Tools for the discretization of integral and differential equations
Framework	Tools for building, testing, and integrating Trilinos capabilities

Performance portability for various parallel programming paradigms

TRILINOS is targeted for all major parallel architectures, including

- distributed-memory using the Message Passing Interface (MPI),
- multicore using a variety of common approaches,
- accelerators using common and emerging approaches, and
- vectorization.

Performance portability is achieved through the KOKKOS programming model².

*“... as long as a given algorithm and problem size contain enough latent parallelism, **the same Trilinos source code** can be compiled and execution on **any reasonable combination of distributed, multicore, accelerator and vectorizing computing devices**.”* — Trilinos Website

Overview of Trilinos packages

TRILINOS is a collection of more than 50 software packages:

- Each TRILINOS package is a *self-contained, independent piece of software* with its own set of requirements, its own development team³ and group of users.
- However, there are often certain *dependencies between different TRILINOS packages*. Some TRILINOS packages also *depend on third party libraries (TPLs)*.
- Generally, a *certain degree of interoperability* of the different TRILINOS packages is provided.

Contents of `trilinos/packages`:

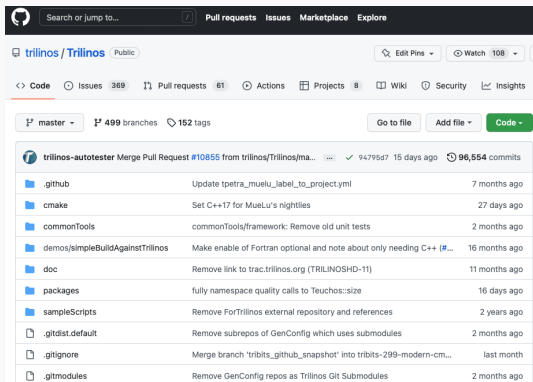
adelus	epetra	isorropia	nox	rol	stratimikos	triutils
amosos	epetraext	kokkos	pamgen	rtop	teko	xpetra
amosos2	fei	kokkos-kernels	panzer	rythmos	tempus	zoltan
anasazi	framework	komplex	percept	sacado	teuchos	zoltan2
aztecoo	galeri	minitensor	phalanx	seacas	thyra	
belos	ifpack	ml	pike	shards	tpetra	
common	ifpack2	moertel	piro	shylu	TriKota	
compadre	intrepid	muelu	pliris	stk	trilinoscouplings	
domi	intrepid2	new_package	PyTrilinos	stokhos	Trilinos_DLLEXPOTMacro.h.in	

	MPI (EPETRA-based)	MPI+X (TPETRA-based)
Linear algebra	Epetra & EpetraExt	Tpetra
Direct sparse solvers	Amesos	Amesos2
Iterative solvers	AztecOO	Belos
Preconditioners: <ul style="list-style-type: none"> • One-level (incomplete) factorization • Multigrid • Domain decomposition 	Ifpack ML	Ifpack2 MueLu ShyLU
Eigenproblem solvers		Anasazi
Nonlinear solvers	NOX & LOCA	
Partitioning	Isorropia & Zoltan	Zoltan2
Example problems	Galeri	
Performance portability		Kokkos & KokkosKernels
Interoperability	Stratimikos & Thyra	
Tools	Teuchos	
⋮	⋮	⋮
⋮	⋮	⋮

- Packages, that do not depend on EPETRA or TPETRA work in both software stacks, e.g. GALERI, NOX & LOCA, TEUCHOS
- More details on <https://trilinos.github.io>.

Source code repository

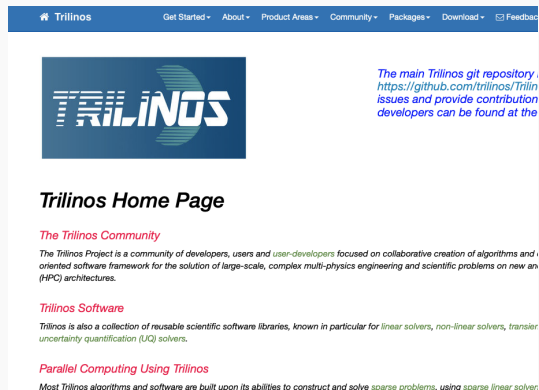
- GitHub:
<https://github.com/trilinos/Trilinos>
- Default branch: master
- Development branch: develop



The screenshot shows the GitHub interface for the Trilinos repository. At the top, there's a search bar and navigation links for Pull requests, Issues, Marketplace, and Explore. Below this, the repository name 'trilinos / Trilinos' is displayed with a 'Public' badge. A navigation bar shows 'Code' as the active tab, with counts for Issues (369), Pull requests (61), Actions, Projects (8), Wiki, Security, and Insights. Below the navigation bar, there's a section for 'master' branch with 499 branches and 152 tags. A 'Go to file' button and an 'Add file' button are visible. The main content area shows a list of files and directories with their commit history. The files listed are: .github (Update tpetra_muelu_label_to_project.yml, 7 months ago), cmake (Set C++17 for MueLu's nightlies, 27 days ago), commonTools (commonTools/framework: Remove old unit tests, 2 months ago), demos/simpleBuildAgainstTrilinos (Make enable of Fortran optional and note about only needing C++ (#..., 16 months ago), doc (Remove link to trac.trilinos.org (TRILINOSH-11), 11 months ago), packages (fully namespace quality calls to Teuchos::size, 16 days ago), sampleScripts (Remove ForTrilinos external repository and references, 2 years ago), .gitdist.default (Remove subrepos of GenConfig which uses submodules, 2 months ago), .gitignore (Merge branch 'tribits_github_snapshot' into tribits-299-modern-cm..., last month), and .gitmodules (Remove GenConfig repos as Trilinos Git Submodules, 2 months ago).

Website

- Link: <https://trilinos.github.io>
- Provides general information
- Details on all packages
- Links to Doxygen source code documentation



The screenshot shows the Trilinos website. At the top, there's a navigation bar with links for Get Started, About, Product Areas, Community, Packages, Download, and Feedback. Below this, there's a large blue banner with the Trilinos logo. To the right of the logo, there's a text box that says: 'The main Trilinos git repository, <https://github.com/trilinos/Trilinos> issues and provide contribution developers can be found at the'. Below the banner, there's a section titled 'Trilinos Home Page'. Under this, there's a section titled 'The Trilinos Community' with a link to <https://github.com/trilinos/Trilinos>. Below this, there's a section titled 'Trilinos Software' with a description: 'Trilinos is also a collection of reusable scientific software libraries, known in particular for linear solvers, non-linear solvers, transfer uncertainty quantification (UQ) solvers.' Below this, there's a section titled 'Parallel Computing Using Trilinos' with a description: 'Most Trilinos algorithms and software are built upon its abilities to construct and solve sparse problems, using sparse linear solver.'

III. How to install Trilinos?

1 TRIBITS: Tribal Build, Integrate, and Test System

2 TRIBITS for building TRILINOS

- Package manager of your operating system
 - TRILINOS is **available through most package managers** for Linux operating systems.
 - However, when installing TRILINOS via package manager, you **do not have full control over its configuration**.
- Spack⁴
 - Similar to a package manager, but with from-source-build-and-installation
 - **Easy to get started** with, automatically **takes care of dependencies**
 - Allows to maintain multiple versions of TRILINOS on the same machine
 - **Tedious to prescribe your desired configuration**
- Manual installation from source files
 - In order to have **full control over the configuration** of TRILINOS, it may be compiled and installed from the source files.
 - Especially recommended if you plan to modify TRILINOS source code / develop in TRILINOS

The dependencies result from the choice of TRILINOS packages.

Examples:

MPI	—	Message Passing Interface ⁵
BLAS	—	Basic Linear Algebra Subprograms ⁶
LAPACK	—	Linear Algebra PACKage ⁷
BOOST	—	Peer-reviewed portable C++ libraries ⁸
METIS & PARMETIS	—	Graph Partitioning ⁹
HDF5	—	Hierarchical Data Format ¹⁰
MUMPS	—	MULTifrontal Massively Parallel sparse direct Solver ¹¹
⋮	⋮	

Some observations and requirements:

- TRILINOS is a large software project with many internal and external dependencies.
- These dependencies need to be managed properly, in particular, by a suitable build system.
- TRILINOS' package architecture allows but also requires software modularity.
- User needs to specify list of enabled/disabled packages.
- Automated checks for satisfaction of dependencies and modularity are necessary.

Build system

TRILINOS uses **TriBITS** for configuration, build, installation and test management.

⇒ We will now briefly look into TRIBITS and learn how to use it to configure, build, and install TRILINOS with a user-chosen set of packages.

Requirements for large software projects

- Multiple software repositories and distributed development teams
- Multiple compiled programming languages (C, C++, Fortran) and mixed-language programs
- Multiple development and deployment platforms (Linux, MacOS, Super-Computers, etc.)
- Stringent software quality requirements

TriBITS = Tribal Build, Integrate, and Test System¹²

- Stand-alone build system for complex software projects
- Built on top of CMAKE
- TRIBITS provides a custom CMAKE build & test framework

Why CMake?

- Open-source tools maintained and used by a large community and supported by a professional software development company (Kitware^a).
- CMAKE:
 - Simplified build system, easier maintenance
 - Improved mechanism for extending capabilities (CMAKE language)
 - Support for all major C, C++, and Fortran compilers.
 - Automatic full dependency tracking (headers, src, mod, obj, libs, exec)
 - Shared libraries on all platforms and compilers
 - ...
- CTEST:
 - Parallel execution and scheduling of tests and test time-outs
 - Memory testing (Valgrind)
 - Line coverage testing (GCC LCOV)
 - Better integration between the test system and the build system



<https://cmake.org>

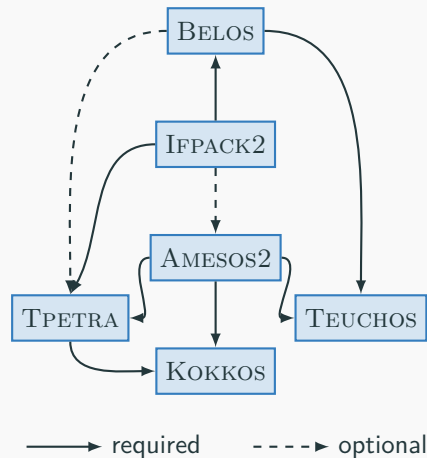
^a<https://www.kitware.com>

- Framework for large, distributed multi-repository CMAKE projects
- Reduce boiler-plate CMAKE code and enforce consistency across large distributed projects
- Subproject dependencies and namespacing architecture: packages
- Automatic package dependency handling (for build & testing)
- Additional functionality missing in raw CMAKE
- Changes in default CMAKE behavior when necessary

Structural units of a TriBITS project

- **TRIBITS project:**
 - Complete CMAKE “project”
 - Overall project settings
- **TRIBITS repository:**
 - Collection of packages & TPLs
 - Unit of distribution and integration
- **TRIBITS package:**
 - Collection of related software & tests
 - Lists dependencies on packages & TPLs
 - Unit of testing, namespacing, and documentation
- **TRIBITS subpackage:**
 - Partitioning of package software & tests
- **TRIBITS Third Party Libraries (TPLs):**
 - Specification of external dependencies (libs)
 - Required or optional dependency
 - Single definition across all packages

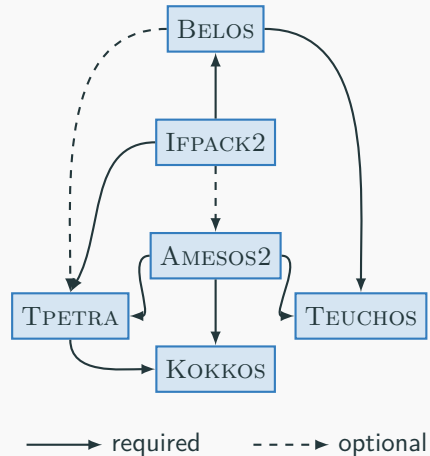
Example from Trilinos:



Activation of Trilinos packages:

```
1 cmake \  
2   -D ... \  
3   -D Trilinos_ENABLE_ALL_PACKAGES:BOOL=OFF \  
4   -D ... \  
5   -D Trilinos_ENABLE_Amesos2:BOOL=ON \  
6   -D Trilinos_ENABLE_Belos:BOOL=ON \  
7     -D Belos_ENABLE_Tpetra:BOOL=ON \  
8   -D Trilinos_ENABLE_Ipack2:BOOL=ON \  
9     -D Ifpack2_ENABLE_Amesos2:BOOL=ON \  
10  -D Trilinos_ENABLE_MueLu:BOOL=OFF \  
11  -D Trilinos_ENABLE_Teuchos:BOOL=ON \  
12  -D Trilinos_ENABLE_Tpetra:BOOL=ON \  
13  -D ... \  
14  -D TPL_ENABLE_MPI:BOOL=ON \  
15  -D TPL_ENABLE_ParMETIS:BOOL=ON \  
16  -D ... \  
17  {$TRILINOS_SOURCE}
```

Example from Trilinos:



Software development using TriBITS:

- Beyond the scope of this tutorial
- Please consult the TRIBITS online resources:
 - <https://tribits.org>
 - <https://github.com/TriBITSPub/TriBITS>

Building Trilinos using TriBITS:

- Packages: how is TRILINOS structured?
- Configure script: how to invoke CMAKE?
- Build and install TRILINOS



Invoking CMake via a configure script

How to invoke CMake?

- `$ cmake -D <option_1> -D <option_2> -D <...> {path/to/source}`

Why use a configure script?

- Number of options in `cmake` command grow very quickly \Rightarrow script reduces burden to type everything into the command line
- Script helps to
 - reproduce a configuration / re-configure
 - debug a configuration
 - share a configuration with colleagues and collaborators

Recommendation

Always invoke `CMAKE` through a configure script.

An exemplary configure script:

```
1  #!/bin/bash
2
3  SOURCE_DIR=path/to/src/directory
4  BUILD_DIR=path/to/build/directory
5  INSTALL_DIR=path/to/install/directory
6
7  cmake \
8    -D CMAKE_INSTALL_PREFIX:PATH="$INSTALL_DIR" \
9    -D CMAKE_CXX_COMPILER_FLAGS:STRING="..." \
10   -D ... \
11   -D ... \
12   {$SOURCE_DIR}
```

Remarks:

- Recommendation: out-of-source build (i.e. `SOURCE_DIR` \neq `BUILD_DIR` to keep source directory clean from build artifacts)
- `BUILD_DIR` and `INSTALL_DIR` can be the same (Depends on the project. Some projects require them to be different.)

Practical tip

Sometimes when changing the CMAKE configuration, it can be necessary to clean the `BUILD_DIR` (in particular, the CMAKE files).

If the CMAKE configuration fails unexpectedly, try again after deleting the CMAKE files in the `BUILD_DIR`.

Writing your own configure scripts for Trilinos

Outline of a Trilinos configure script

1. Select your favorite shell environment
2. Define environment variables with necessary paths
3. The cmake command
 - 3.1 Compilation settings
 - 3.2 General TRILINOS settings
 - 3.3 Package configuration
 - 3.4 External dependencies / TPLs

Remarks:

- Structuring and indentation just a personal recommendation for better readability
- Ongoing refactorings in TRIBITS: distinction between package and TPL might vanish in the future

```
1  #!/bin/bash
2
3  TRILINOS_SOURCE=path/to/src/directory
4  TRILINOS_BUILD=path/to/build/directory
5  TRILINOS_INSTALL=path/to/install/directory
6
7  cmake \
8    -D CMAKE_CXX_COMPILER_FLAGS:String="..." \
9    -D CMAKE_INSTALL_PREFIX:PATH="$TRILINOS_INSTALL" \
10   -D ... \
11   \
12   -D Trilinos_ENABLE_ALL_PACKAGES:BOOL=OFF \
13   -D ... \
14   \
15   -D Trilinos_ENABLE_Amesos2:BOOL=ON \
16   -D Trilinos_ENABLE_Belos:BOOL=ON \
17     -D Belos_ENABLE_Tpetra:BOOL=ON \
18   -D Trilinos_ENABLE_Ipack2:BOOL=ON \
19     -D Ipack2_ENABLE_Amesos2:BOOL=ON \
20   -D Trilinos_ENABLE_MueLu:BOOL=OFF \
21   -D Trilinos_ENABLE_Teuchos:BOOL=ON \
22   -D Trilinos_ENABLE_Tpetra:BOOL=ON \
23   -D ... \
24   \
25   -D TPL_ENABLE_MPI:BOOL=ON \
26   -D TPL_ENABLE_ParMETIS:BOOL=ON \
27   -D ... \
28   {$TRILINOS_SOURCE}
```

Configure, build, and install Trilinos

1. Create desired directory structure (source, build, install directories)
2. Get the source code: `git clone git@github.com:Trilinos/Trilinos.git`
`<path/to/source/dir>`
3. Write a configure script
4. Run the configure script in the build directory
5. Build in parallel on `<numProc>` processes: `make -j <numProc>`
6. Install: `make install`

Prerequisites:

- CMAKE version > 3.17
- TRILINOS has been installed.

Tasks:

1. Make TRILINOS available to the build configuration of the application code
2. Include TRILINOS headers and instantiate TRILINOS objects

Goals:

- Assert required packages during configuration
- Maybe: use same compiler/linker settings for Trilinos build and build of the application
- Proper setup and tear-down of parallel environment (MPI, KOKKOS, ...)

Including Trilinos in CMakeLists.txt

- Set minimum CMake version to 3.17.1:

```
cmake_minimum_required(VERSION 3.17.1)
```

- Declare project, but don't specify language and compilers yet. Defer until having found TRILINOS to match compiler/linker settings to those of the TRILINOS installation.

```
project(name_of_your_project NONE)
```

- Get TRILINOS as one entity and assert required packages (e.g. TEUCHOS & TPETRA)

```
find_package(Trilinos REQUIRED COMPONENTS Teuchos Tpetra)
```

- Make sure to use same compilers and flags as TRILINOS

```
set(CMAKE_CXX_COMPILER ${Trilinos_CXX_COMPILER} )  
set(CMAKE_C_COMPILER ${Trilinos_C_COMPILER} )  
set(CMAKE_Fortran_COMPILER ${Trilinos_Fortran_COMPILER} )  
  
set(CMAKE_CXX_FLAGS "${Trilinos_CXX_COMPILER_FLAGS} ${CMAKE_CXX_FLAGS}")  
set(CMAKE_C_FLAGS "${Trilinos_C_COMPILER_FLAGS} ${CMAKE_C_FLAGS}")  
set(CMAKE_Fortran_FLAGS "${Trilinos_Fortran_COMPILER_FLAGS} ${CMAKE_Fortran_FLAGS}")
```

- Now, enable the compilers that we have gotten from TRILINOS

```
enable_language(C)
enable_language(CXX)
if (CMAKE_Fortran_COMPILER)
    enable_language(Fortran)
endif()
```

- Build the application `your_app` and link to TRILINOS

```
add_executable(your_app ${CMAKE_CURRENT_SOURCE_DIR}/main.cpp)
target_include_directories(your_app PRIVATE
    ${CMAKE_CURRENT_SOURCE_DIR} ${Trilinos_INCLUDE_DIRS} ${Trilinos_TPL_INCLUDE_DIRS})
target_link_libraries(your_app ${Trilinos_LIBRARIES} ${Trilinos_TPL_LIBRARIES})
```

Including Trilinos in your source code

- Since TRILINOS has been installed on your machine, include headers via

```
#include <Name_of_Trilinos_header.hpp>
```

- **Recommendation:** Setup parallel environment through `Tpetra::ScopeGuard` which hides details of MPI & kokkos initialization (and finalization) internally.

```
int main (int argc, char *argv[])
{
    Tpetra::ScopeGuard tpetraScope (&argc, &argv);
    {
        // Put all your code inside this scope to never let Tpetra objects persist after
        // either MPI_Finalize or Kokkos::finalize has been called. This is because the
        // objects' destructors may need to call MPI or Kokkos functions.
        // In particular, never create Tpetra objects at main scope.
    }
}
```

- Get the communicator object:

```
Teuchos::RCP<const Teuchos::Comm<int>> comm = Tpetra::getDefaultComm();
```

How to work on these exercises?

- Hands-on exercises in the docker container (repository available at <https://github.com/EuroTUG/trilinos-docker>)
- Code snippets to be completed (guided by instructions in a README file)
- Work in small groups:
 - ZOOM Break-out rooms with 3-5 participants per room
 - Possibility for collaboration, discussion and joint problem solving (e.g. screen sharing)
 - Some “tutors” will hop through all rooms to answer questions and assist if necessary
 - Raise the virtual hand if you have questions
- No pressure to finalize the exercise. Solutions are part of the repository for later study.

Configure Trilinos:

- Write a configure script for TRILINOS with the following packages enabled:
 - BELOS, GALERI, IFPACK2, TPETRA
 - You might need further packages to satisfy all required dependencies.
- Configure and build TRILINOS with this configuration.
- Material: `exercises/ex_01_configure`

Use Trilinos:

- Complete the CMakeLists.txt to include TRILINOS into the build of an exemplary application
- Complete the app's source code to setup MPI through `Tpetra::ScopeGuard`
- Get the communicator and print some of its information to the terminal
- Material: `exercises/ex_01_cmake`

Hint

Both exercises are independent of each other. You do not have to wait for the build in `ex_01_configure` to complete, since the second exercise uses a pre-installed TRILINOS installation. Just start a second instance of the docker container to get started on `ex_01_cmake`, while the first exercise is still building. (Or skip the build process at all.)

- TRILINOS **GitHub repository**: <https://github.com/Trilinos>
- TRILINOS **website**: <https://trilinos.github.io/index.html>
 - **Documentation**: <https://trilinos.github.io/documentation.html>
 - Each package has its own **Doxygen documentation**: For instance, Tpetra:
<https://docs.trilinos.org/dev/packages/tpetra/doc/html/index.html>
 - **Getting started**: https://trilinos.github.io/getting_started.html
- TRILINOS **hands-on tutorials**:
https://github.com/Trilinos_tutorial/wiki/TrilinosHandsOnTutorial
- KOKKOS resources on GitHub: <https://github.com/kokkos>

