Getting Started With Trilinos

Alexander Heinlein[1] Matthias Mayr[2]

June 24 - 26, 2024

[1]Delft University of Technology

[2]Universität der Bundeswehr München

I. Welcome to EuroTUG 2024

Welcome to EuroTUG 2024

European Trilinos & Kokkos User Group (EuroTUG)

HSU/UniBwH, June 24-26, 2024

Photo: Medienzentrum HSU/UniBwH

What is EuroTUG?

- EuroTUG Meeting = European Trilinos & Kokkos User Group Meeting
- Meeting series for Europe-based users and developers of the Trilinos and Kokkos projects:
  - learn about recent developments in Trilinos & Kokkos
  - report on their use cases and experiences with Trilinos & Kokkos
  - form a European network of Trilinos & Kokkos users and developers

Acknowledgement:

The European Trilinos & Kokkos User Group 2024 is supported by the project hpc.bw, funded by dtec.bw — Digitalization and Technology Research Center of the Bundeswehr. dtec.bw is funded by the European Union – NextGenerationEU.

Organization

Schedule:

- June 24, 2024
  - Tutorial "Introduction to KOKKOS"
  - User & Developer Presentations
- June 25, 2024
  - Tutorial "TRILINOS for Beginners"
  - User & Developer Presentations
  - Dinner
- June 26, 2024
  - Keynote "HPC in Hamburg"
  - Tour of HSUper

Detailed schedule on the EuroTUG website:
https://eurotug.github.io

Organizers:

- Dr. Alexander Heinlein, TU Delft, FROSCH developer
- Dr. Matthias Mayr, University of the Bundeswehr Munich, MUELU developer
- Prof. Dr. Philipp Neumann, DESY & U Hamburg



External support

- Jakob Bludau, TUM
- Romin Tomasetti, U Liege

Practical information

Wifi

- eduraom
- FreeHSU

Breaks, Lunch, Dinner

- Coffee breaks
- Lunch will be in the mensa (free, vouchers to be redeemed at the cash desk)
- **Dinner on Tuesday:** conference dinner at 7.30 pm (food is free, drinks on a self-pay basis):

  Alster Lagune, Ballindamm 14b, 20095 Hamburg

Photos

- We will take a group photo at the beginning of the lunch break on Tuesday

II. Introduction to Trilinos

Disclaimer I

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*.

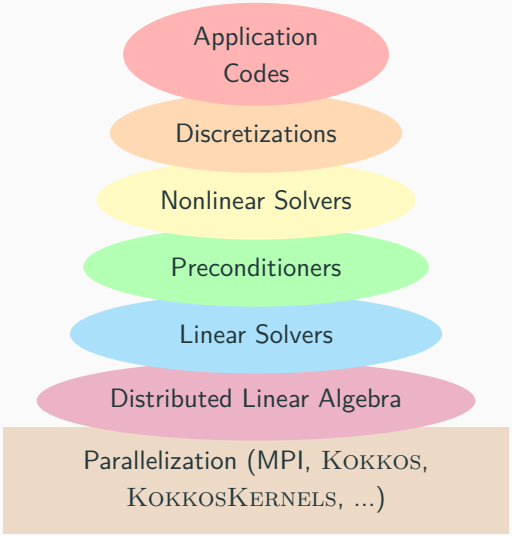
Disclaimer II

We will do the hands-on exercises *"on demand"*.

An Open-Source Library of Software for Scientific Computing

Mission statement[1]: *"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 model[2].

" . . . 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 team*[3] *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	PyTrilinos2	stokhos	Trilinos_DLLExportMacro.h.in
amesos	epetraext	kokkos	pamgen	rol	stratimikos	TrilinosInstallTests
amesos2	framework	kokkos-kernels	panzer	rtop	teko	triutils
anasazi	galeri	krino	percept	sacado	tempus	xpetra
aztecoo	ifpack	minitensor	phalanx	seacas	teuchos	zoltan
belos	ifpack2	ml	piro	shards	thyra	zoltan2
common	intrepid	muelu	pliris	shylu	tpetra	
compadre	intrepid2	new_package	PyTrilinos	stk	trilinoscouplings	

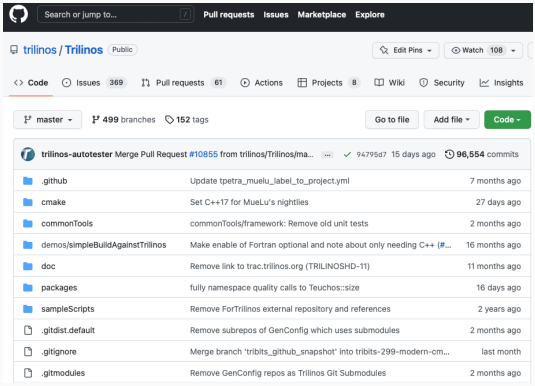
	MPI (Epetra-based)	MPI+X (Tpetra-based)
Linear algebra	**Epetra** & **EpetraExt**	**Tpetra**
Direct sparse solvers	**Amesos**	**Amesos2**
Iterative solvers	**AztecOO**	**Belos**
Preconditioners: • One-level (incomplete) factorization • Multigrid • Domain decomposition	 **Ifpack** **ML**	 **Ifpack2** **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
- **Important:** Epetra stack **to be removed**; **only Tpetra stack will remain**:
  - Talk by Chris Siefert yesterday evening on how to migrate from Epetra → Tpetra
  - Part VI of today: *Recent Developments & Future Trends in* Trilinos
- More details on https://trilinos.github.io.

Trilinos resources

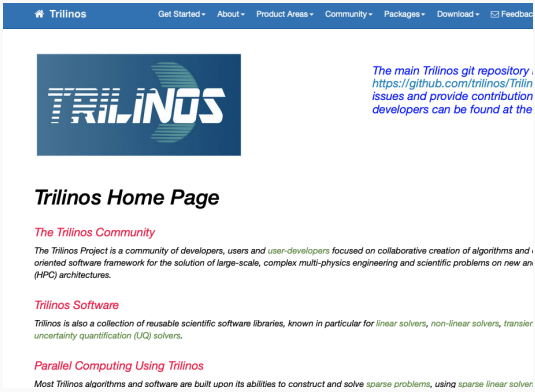
Source code repository

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



Website

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



III. How to install Trilinos?

1. TriBITS: Tribal Build, Integrate, and Test System

2. TriBITS for building Trilinos

Different installation mechanisms

- **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**.
- **Spack**[4]
  - 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

Dependencies

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.

What is TriBITS?

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 System[12]

- 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
  - ...
- CMake:
  - 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

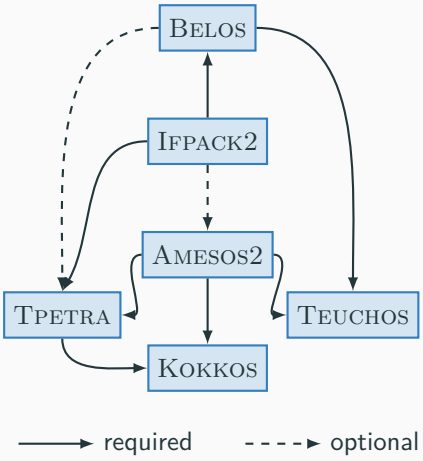
Why TriBITS?

- 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

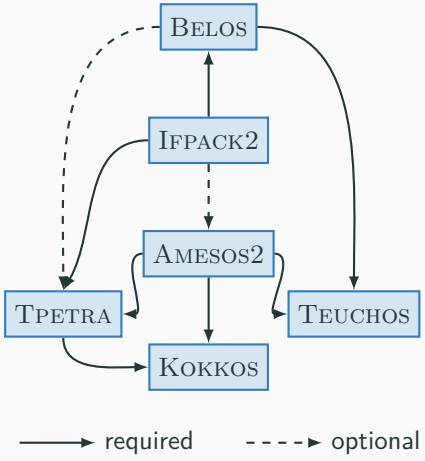
Example from Trilinos:



Activation of Trilinos packages:

```
cmake \
  -D ... \
  -D Trilinos_ENABLE_ALL_PACKAGES:BOOL=OFF \
  -D ... \
  -D Trilinos_ENABLE_Amesos2:BOOL=ON \
  -D Trilinos_ENABLE_Belos:BOOL=ON \
    -D Belos_ENABLE_Tpetra:BOOL=ON \
  -D Trilinos_ENABLE_Ifpack2:BOOL=ON \
    -D Ifpack2_ENABLE_Amesos2:BOOL=ON \
  -D Trilinos_ENABLE_MueLu:BOOL=OFF \
  -D Trilinos_ENABLE_Teuchos:BOOL=ON \
  -D Trilinos_ENABLE_Tpetra:BOOL=ON \
  -D ... \
  -D TPL_ENABLE_MPI:BOOL=ON \
  -D TPL_ENABLE_ParMETIS:BOOL=ON \
  -D ... \
  {$TRILINOS_SOURCE}
```

Example from Trilinos:



Scope of this tutorial

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 ⇒ 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 readibitliy
- Ongoing refactorings in TRIBITS: distinction between package and TPL might vanish in the future

```
#!/bin/bash

TRILINOS_SOURCE=path/to/src/directory
TRILINOS_BUILD=path/to/build/directory
TRILINOS_INSTALL=path/to/install/directory

cmake \
  -D CMAKE_CXX_COMPILER_FLAGS:STRING="..." \
  -D CMAKE_INSTALL_PREFIX:PATH="$TRILINOS_INSTALL" \
  -D ... \
  \
  -D Trilinos_ENABLE_ALL_PACKAGES:BOOL=OFF \
  -D ... \
  \
  -D Trilinos_ENABLE_Amesos2:BOOL=ON \
  -D Trilinos_ENABLE_Belos:BOOL=ON \
    -D Belos_ENABLE_Tpetra:BOOL=ON \
  -D Trilinos_ENABLE_Ifpack2:BOOL=ON \
    -D Ifpack2_ENABLE_Amesos2:BOOL=ON \
  -D Trilinos_ENABLE_MueLu:BOOL=OFF \
  -D Trilinos_ENABLE_Teuchos:BOOL=ON \
  -D Trilinos_ENABLE_Tpetra:BOOL=ON \
  -D ... \
  \
  -D TPL_ENABLE_MPI:BOOL=ON \
  -D TPL_ENABLE_ParMETIS:BOOL=ON \
  -D ... \
  ${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`

Using Trilinos in application codes — Overview

Prerequesites:

- CMake version > 3.23
- 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 3.23.0:

```
cmake_minimum_required(VERSION 3.23.0)
```

- 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();
```

Hands-on exercises

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:
  - Possibility for collaboration, discussion and joint problem solving
  - Some "tutors" will circle the room to answer questions and assist if necessary
  - Raise your 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.)

IV. Using Trilinos in application codes - Part I

3 TPETRA Package

4 `Tpetra::Map`

5 `Tpetra::Vector`

6 `Tpetra::MultiVector`

7 `Tpetra::CrsMatrix`

8 `Tpetra::CrsMatrix` – Matrix assembly

9 Matrix-vector multiplication

10 `Tpetra::Import` & `Tpetra::Export`

Scope and goals

Scope

Focus on an **introduction to the Tpetra linear algebra package with respect to distributed-memory (MPI) parallelization**.

Out of the scope

An introduction to all TRILINOS packages including **shared-memory (X) parallelization using `Kokkos`**.

Teuchos

Before working with Trilinos, please also take a look at the TEUCHOS package! It provides **many useful tools** and is used all over the TRILINOS code.

- **Memory management** (e.g., `Teuchos::RCP` **smart pointers** or `Teuchos::Array` **arrays with additional functionality**)
  (very helpful to replace many standard C++ data types and containers)
- **Parameter lists**
  (very helpful for handling parameters for functions, classes, or whole programs)
- **Communication** (e.g., `Teuchos::Comm`)
  (See https://docs.trilinos.org/dev/packages/teuchos/doc/html/classTeuchos_1_1Comm.html)
- **Numerics** (e.g., `BLAS` and `LAPACK` wrappers)
- **Output support**, **exception handling**, **unit testing support**, and much more ...

→ TEUCHOS `Doxygen` documentation:
https://docs.trilinos.org/dev/packages/teuchos/doc/html/

Tpetra Package

Important classes:

`Tpetra::Map`	**Parallel distributions**: Contains information used to distribute vectors, matrices, and other objects
`Tpetra::Vector` & `Tpetra::MultiVector`	**Distributed sparse vectors**: Provides vector services such as scaling, norms, and dot products.
`Tpetra::Operator`	**Base class for linear operators**: Abstract interface for operators (e.g., matrices and preconditioners).
`Tpetra::RowMatrix`	**Distributed sparse matrices**: An abstract interface for row-distributed sparse matrices; derived from `Tpetra::Operator`.
`Tpetra::CrsMatrix`	**Distributed sparse matrices**: Specific implementation of `Tpetra::RowMatrix`, utilizing compressed row storage (CRS) format
`Tpetra::Import` & `Tpetra::Export`	**Import/Export classes**: Allow efficient transfer of objects built using one mapping to a new object with a new mapping.

→ Tpetra Doxygen documentation:
https://docs.trilinos.org/dev/packages/tpetra/doc/html/

- The parallel linear algebra objects from TPETRA are typically **distributed based on the rows**.
- **Example:** Consider the case of a vector $V \in \mathbb{R}^5$ and a sparse matrix $A \in \mathbb{R}^{5\times 5}$

$$V = \begin{bmatrix} v \\ w \\ x \\ y \\ z \end{bmatrix} \qquad A = \begin{bmatrix} a & b & & & \\ c & d & e & & \\ & f & g & h & \\ & & i & j & k \\ & & & l & m \end{bmatrix}$$

distributed among two parallel processes:

$$V = \begin{bmatrix} v \\ w \\ x \\ y \\ z \end{bmatrix} \qquad A = \begin{bmatrix} a & b & & & \\ c & d & e & & \\ & f & g & h & \\ & & i & j & k \\ & & & l & m \end{bmatrix}$$

proc 0

proc 1

- This can be implemented by storing the *local portions of the vector and the matrix*:

$$V_0 = \begin{bmatrix} v \\ x \\ z \end{bmatrix} \qquad A_0 = \begin{bmatrix} a & b & & & & \\ & & f & g & h & \\ & & & & l & m \end{bmatrix} \qquad \text{proc 0}$$

$$V_1 = \begin{bmatrix} w \\ y \end{bmatrix} \qquad A_1 = \begin{bmatrix} c & d & e & & & \\ & & & i & j & k \end{bmatrix} \qquad \text{proc 1}$$

Problem: If only the partitioned data is available on the processes, the global vector V and matrix A cannot be restored. In particular, it is not clear where the local rows are located in the global matrix.

- Therefore, we additionally store the **global row indices corresponding to the local rows**, here denoted as M_0 and M_1 (local-to-global map):

$$V_0 = \begin{bmatrix} v \\ x \\ z \end{bmatrix} \qquad A_0 = \begin{bmatrix} a & b & & & & \\ & & f & g & h & \\ & & & & l & m \end{bmatrix} \qquad M_0 = \begin{bmatrix} 0 \\ 2 \\ 4 \end{bmatrix} \qquad \text{proc 0}$$

$$V_1 = \begin{bmatrix} w \\ y \end{bmatrix} \qquad A_1 = \begin{bmatrix} c & d & e & & & \\ & & & i & j & k \end{bmatrix} \qquad M_1 = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \qquad \text{proc 1}$$

- Using the local-to-global map, the global objects are fully specified.

Process 0:

$$V_0 = \begin{bmatrix} v \\ x \\ z \end{bmatrix} \quad A_0 = \begin{bmatrix} a & b & & & \\ & f & g & h & \\ & & & l & m \end{bmatrix} \quad M_0 = \begin{bmatrix} 0 \\ 2 \\ 4 \end{bmatrix} \quad \text{proc 0}$$

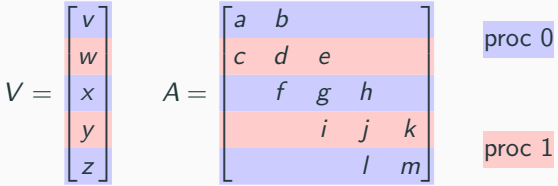
$$\to V_0 = \begin{bmatrix} v \\ \\ x \\ \\ z \end{bmatrix} \quad A_0 = \begin{bmatrix} a & b & & & \\ & & & & \\ & f & g & h & \\ & & & & \\ & & & l & m \end{bmatrix}$$

Process 1:

$$V_1 = \begin{bmatrix} w \\ y \end{bmatrix} \quad A_1 = \begin{bmatrix} c & d & e & & \\ & & & i & j & k \end{bmatrix} \quad M_1 = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad \text{proc 1}$$

$$\to V_1 = \begin{bmatrix} \\ w \\ \\ y \\ \\ \end{bmatrix} \quad A_1 = \begin{bmatrix} \\ c & d & e & & \\ \\ & & & i & j & k \\ \\ \end{bmatrix}$$

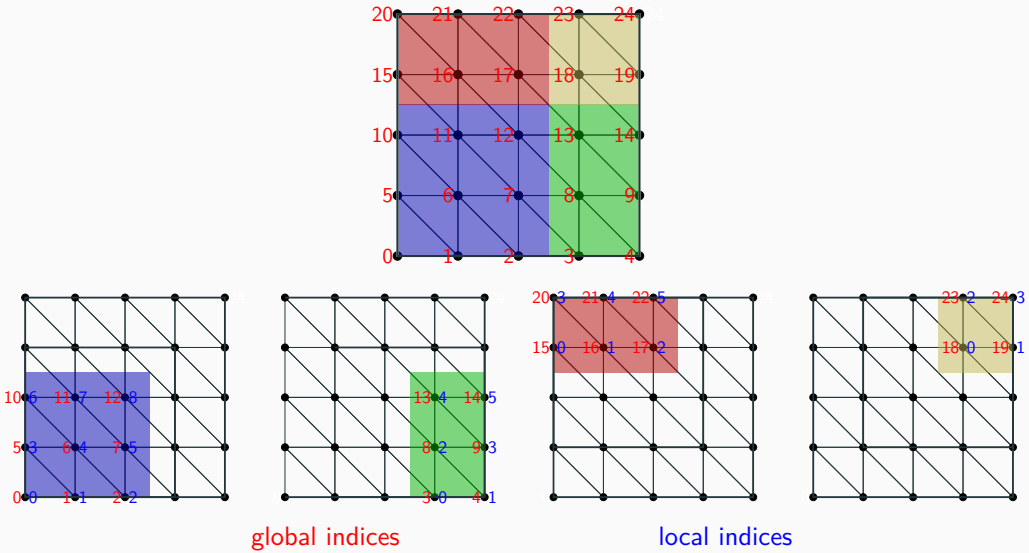
- In summary, in addition to the **local portions of the global Tpetra objects**, **local-to-global mappings** are necessary to describe parallel distributed global objects:



- The local-to-global mappings are stored in `Tpetra::Map` objects.

See https://docs.trilinos.org/dev/packages/tpetra/doc/html/classTpetra_1_1Map.html for more details.

Tpetra::Map – Exemplary Map/Distribution for a Mesh



Tpetra::Vector

As previously shown, a **parallel distributed vector** (`Tpetra::Vector`) essentially corresponds to

- arrays containing the **local portions of the vectors** (entries) and
- a `Tpetra::Map` storing the **local-to-global mapping**.

$$V = \begin{bmatrix} v \\ w \\ x \\ y \\ z \end{bmatrix} \quad \begin{matrix} \text{proc 0} \\ \\ \\ \text{proc 1} \\ \\ \end{matrix}$$

$$V_0 = \begin{bmatrix} v \\ x \\ z \end{bmatrix} \quad M_0 = \begin{bmatrix} 0 \\ 2 \\ 4 \end{bmatrix} \quad \text{proc 0}$$

$$V_1 = \begin{bmatrix} w \\ y \end{bmatrix} \quad M_1 = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad \text{proc 1}$$

Constructor:

```
Vector ( const Teuchos::RCP< const map_type >& map, /* optional */ )
```

map: `Tpetra::Map` object specifying the parallel distribution of the `Tpetra::Vector`. The map also defines the length (local and global) of the vector.

- The `Tpetra::MultiVector` allows for the construction of **multiple vectors with the same parallel distribution**:

$$V = \begin{bmatrix} v_{11} & \cdots & v_{1m} \\ v_{21} & \cdots & v_{2m} \\ \vdots & \ddots & \vdots \\ v_{(n-1)1} & \cdots & v_{(n-1)m} \\ v_{n1} & \cdots & v_{nm} \end{bmatrix} \in \mathbb{R}^{n \times m} \quad \text{with } n >> m$$

- A typical use case would be a **linear equation system with multiple right hand sides**:

$$AX = B$$

with $A \in \mathbb{R}^{n \times n}$, $X \in \mathbb{R}^{n \times m}$, and $B \in \mathbb{R}^{n \times m}$. Here, A would typically be a sparse matrix and X and B multivectors.

- It can also be used to implement **skinny dense matrices**.

→ Constructing a `Tpetra::MultiVector` requires the number of vectors to be specified.

As previously shown, a **parallel distributed sparse matrix** (`Tpetra::CrsMatrix`) essentially corresponds to

- the **local portions** of the sparse matrix and
- a `Tpetra::Map` storing the **local-to-global mapping** corresponding to the rows.

$$A = \begin{bmatrix} a & b & & & & \\ c & d & e & & & \\ & f & g & h & & \\ & & & i & j & k \\ & & & & l & m \end{bmatrix} \quad \text{proc 0} \quad \text{proc 1}$$

$$A_0 = \begin{bmatrix} a & b & & & & \\ & f & g & h & & \\ & & & & l & m \end{bmatrix} \qquad M_0 = \begin{bmatrix} 0 \\ 2 \\ 4 \end{bmatrix} \quad \text{proc 0}$$

$$A_1 = \begin{bmatrix} c & d & e & & & \\ & & & i & j & k \end{bmatrix} \qquad M_1 = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad \text{proc 1}$$

In the `Tpetra::CrsMatrix`, the local portions of the sparse matrix are stored in *compressed row storage (CRS) format*.

Minimal constructor:

```
CrsMatrix ( const Teuchos::RCP< const map_type > &rowMap,
            const size_t maxNumEntriesPerRow, /* optional */ )
```

`rowMap`	Parallel distribution of the rows
`maxNumEntriesPerRow`	Maximum number of nonzero entries per row

- In addition to the row map, which corresponds to the local-to-global mapping of the row indices, e.g.,

$$A = \begin{bmatrix} a & b & & & & \\ c & d & e & & & \\ & f & g & h & & \\ & & i & j & k & \\ & & & l & m & o \\ & & & & p & q \end{bmatrix}$$

$$M_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \text{proc 0}$$

$$M_1 = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \quad \text{proc 1}$$

$$M_1 = \begin{bmatrix} 4 \\ 5 \end{bmatrix} \quad \text{proc 2}$$

  there is also **local-to-global mapping for the column indices**, the *column map*.
- If the column map is not specified at the construction of the matrix, it can be generated automatically by the `Tpetra::CrsMatrix` object at a later point.

$$A = \begin{bmatrix} a & b & & & & \\ c & d & e & & & \\ & f & g & h & & \\ & & i & j & k & \\ & & & l & m & o \\ & & & & p & q \end{bmatrix}$$

$$M_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \text{proc 0}$$

$$M_1 = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \quad \text{proc 1}$$

$$M_1 = \begin{bmatrix} 4 \\ 5 \end{bmatrix} \quad \text{proc 2}$$

A compatible *column map* would corresponding to this *row map* would be:

$$A = \begin{bmatrix} a & b & & & & \\ c & d & e & & & \\ & f & g & h & & \\ & & i & j & k & \\ & & & l & m & o \\ & & & & p & q \end{bmatrix}$$

$$\tilde{M}_0 = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \quad \text{proc 0}$$

$$\tilde{M}_1 = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \quad \text{proc 1}$$

$$\tilde{M}_2 = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \quad \text{proc 2}$$

- Column maps are **generally not unique**, as in our example:

$$A = \begin{bmatrix} a & b & & & & \\ c & d & e & & & \\ & f & g & h & & \\ & & i & j & k & \\ & & & l & m & o \\ & & & & p & q \end{bmatrix}$$

$$\tilde{M}_0 = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} \quad \text{proc 0}$$

$$\tilde{M}_1 = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \quad \text{proc 1}$$

$$\tilde{M}_2 = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \quad \text{proc 2}$$

Not unique means that multiple processes share global indices.

- After construction of the matrix, in order to **insert values into the matrix**, the functions `insertLocalValues()` and `insertGlobalValues()` can be used.
- The entries to be inserted in a row are in specified in **sparse format**:

  `row` Index of the row.
  `cols` Indices of the columns where values should be inserted.
  `vals` Values to be inserted.

  (Multiple values inserted at the same location will be added up)

`insertLocalValues()` All indices have to be local. Furthermore,
- the *column map must be available*, and
- the row must be *owned by the calling MPI rank*.

`insertGlobalValues()` All indices have to be global.
- Rows which are *not owned by the calling MPI rank* are later communicated to the *owning MPI rank*.

- If no column map is specified at construction, only `insertGlobalValues()` can be used. Then, the column map is later built by the `Tpetra::CrsMatrix`.

- When all values have been inserted into the matrix, the assembly is finalized by calling `fillComplete()`. Then:
  - Rows on non-owning MPI ranks are communicated to the owning MPI ranks.
  - The final CSR format of the matrix is computed. In particular, the indices are sorted and multiple values inserted at the same location are added up.
  - Global indices are transformed into local indices. Therefore, a new *column map* may be built.
- Only after calling `fillComplete()` the matrix can be further used, e.g., compute a matrix-vector product.
- In case the *row map* or *column map* (in particular, if it was automatically generated) is needed, it can be obtained using the member functions:

`getRowMap()`	Returns the *row map* of the `Tpetra::CrsMatrix`
`getColMap()`	Returns the *columns map* of the `Tpetra::CrsMatrix`

- After calling `fillComplete()`, no new values may be inserted. In order to insert new values, `resumeFill()` has to be called.
- In order to change values at existing locations in the sparsity pattern of the matrix, `replaceLocalValues()` and `replaceGlobalValues()` as well as `sumIntoLocalValues()` and `sumIntoGlobalValues()` may be used.

Matrix-vector multiplication

- As mentioned earlier, the class `Tpetra::CrsMatrix` is derived from **`Tpetra::Operator`**. Any `Tpetra::Operator` can be applied to a `Tpetra::Vector` or `Tpetra::MultiVector` resulting in another `Tpetra::Vector` or `Tpetra::MultiVector`, respectively.
- The parallel application of any `Tpetra::Operator` is characterized by two maps, the *domain map* and the *range map*.

  domain map The map of any vector the operator is applied to.

  range map The map of the resulting vector.

  (Both the domain map and the range map have to be unique!)
- In particular, for a `Tpetra::CrsMatrix`, the following **very general situation**, where the *row map*, *domain map*, and *range map* are all different, is allowed:

$$\begin{bmatrix} a & b & & & \\ c & d & e & & \\ & f & g & h & \\ & & i & j & k \\ & & & l & m \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

- Performing the **matrix-vector multiplication**

$$\begin{bmatrix} a & b & & & \\ c & d & e & & \\ & f & g & h & \\ & & i & j & k \\ & & & l & m \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

will obviously **require communication**.

- The corresponding **communication is performed automatically**. However, the *domain map* and *range map* must have already been specified before application to a vector.

→ The *domain map* and *range map* can be specified within the `fillComplete()` call.

- If they are not specified, they will automatically be chosen as the *row map* of the matrix:

$$\begin{bmatrix} a & b & & & \\ c & d & e & & \\ & f & g & h & \\ & & i & j & k \\ & & & l & m \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

Caution: In contrast to the *domain map* and *range map*, the *row map* does not have to be unique.

Tpetra::Import & Tpetra::Export

- It is possible to change the parallel distribution of Tpetra objects. For example, from

$$V = \begin{bmatrix} v \\ w \\ x \\ y \\ z \end{bmatrix} \qquad A = \begin{bmatrix} a & b & & & \\ c & d & e & & \\ & f & g & h & \\ & & i & j & k \\ & & & l & m \end{bmatrix}$$

$$M_0 = \begin{bmatrix} 0 \\ 2 \\ 4 \end{bmatrix} \quad \text{proc 0} \qquad M_1 = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad \text{proc 1}$$

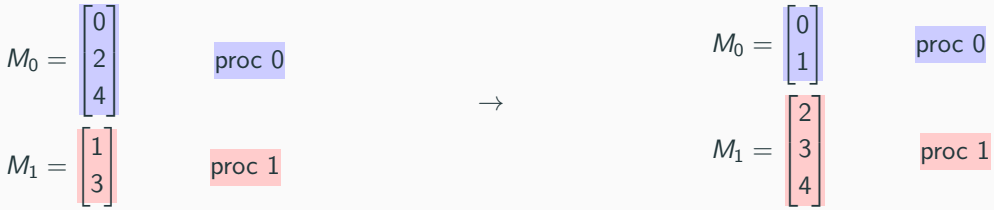
to

$$V = \begin{bmatrix} v \\ w \\ x \\ y \\ z \end{bmatrix} \qquad A = \begin{bmatrix} a & b & & & \\ c & d & e & & \\ & f & g & h & \\ & & i & j & k \\ & & & l & m \end{bmatrix}$$

$$M_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \text{proc 0} \qquad M_1 = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \quad \text{proc 1}$$

- The row maps of the distributions are different. Furthermore, data transfer between the processes is necessary. The data transfer is performed by a **Tpetra::Import** or **Tpetra::Export** object.

- `Tpetra::Import` and `Tpetra::Export` objects are constructed using the `Tpetra::Map` of the original distribution (`source` map) and the `Tpetra::Map` of the desired distribution (`target` map):



Constructors

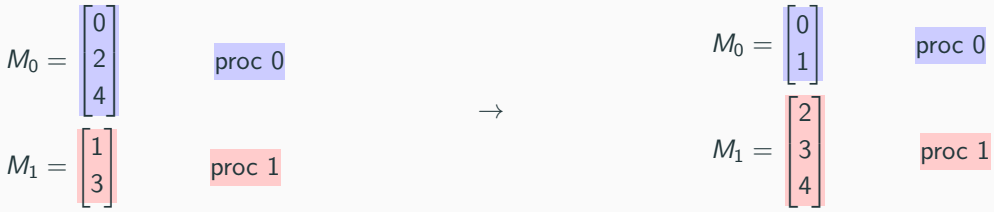
- `Tpetra::Import`

```
Import (const Teuchos::RCP< const map_type > &source,
        const Teuchos::RCP< const map_type > &target);
```

- `Tpetra::Export`

```
Export (const Teuchos::RCP< const map_type > &source,
        const Teuchos::RCP< const map_type > &target);
```

- Obviously, the redistribution



  involves:
  - Sending the global rows 2 and 4 from proc 0 to proc 1
  - Sending the global row 1 from proc 1 to proc 0
- Communication is then performed using the member function

```
Tpetra::DistObject<Packet,LocalOrdinal,GlobalOrdinal,Node>::doExport(
        const SrcDistObject<Packet,LocalOrdinal,GlobalOrdinal,Node> &source,
        const Export<LocalOrdinal,GlobalOrdinal,Node> &exporter,
        const CombineMode CM);
```

  for the parallel distributed `target` object (vector, graph, matrix). The `source` object is the corresponding parallel distributed map with the original distribution.
  (In the corresponding `doImport()` function, the `source` and `target` objects are swapped)

Hands-on exercises

Assemble a linear system:

- Complete the app ex_02_assemble to assemble a linear system (discretized Laplace operator) in Tpetra
- Material: exercises/ex_02_assemble

V. Using Trilinos in application codes - Part II

11 Laplacian Model Problem

12 Preconditioned Gradient Descent (PCG) Method

13 One-Level Schwarz Preconditioner

Scope of this tutorial

- Use linear solvers/preconditioners from TRILINOS to solve systems of linear equations.

Prerequesites:

- Application code with parallel distributed data based on TPETRA
- Linear system $Ax = b$ with matrix A and righ-hand side vector b already assembled

Linear solvers in Trilinos

- Linear solvers available for both Epetra and Tpetra stack.
- Concrete choice of packages depends on linear algebra stack.

Direct solvers

- Packages: Amesos, Amesos2[a]
- Solver implementation / interfaces:
  - KLU (implemented in Trilinos)
  - UMFPACK
  - SuperLU-DIST
  - Pardiso
  - MUMPS

  (Except for KLU, Trilinos has to be configured with the respective TPLs)

[a] Bavier, E. *et al.* **Amesos2 and Belos: Direct and Iterative Solvers for Large Sparse Linear Systems.** *Scientific Programming* **20,** 241–255. http://dx.doi.org/10.3233/SPR-2012-0352 (2012).

Iterative solvers

- Packages: AztecOO[a], Belos[b]
- Methods (also some block variants):
  - Conjugate Gradient (CG)
  - BiCGStab
  - GMRES / Flexible GMRES
  - MINRES
  - LSQR / TFQMR
  - ...

[a] Heroux, M. A. ***AztecOO User Guide.*** Tech. rep. SAND2004-3796 (Sandia National Laboratories, Albuquerque, NM (USA) 87185, 2007).

[b] Bavier, E. *et al.* **Amesos2 and Belos: Direct and Iterative Solvers for Large Sparse Linear Systems.** *Scientific Programming*

Preconditioners in Trilinos

- Preconditioners available for both EPETRA and TPETRA stack.
- Concrete choice of packages depends on linear algebra stack.

One-level methods

- Packages: IFPACK[a], IFPACK2[b]
- Solver implementations:
  - Incomplete LU
  - Relaxation methods (Jacobi, Gauss-Seidel, ...)
  - Polynomial (Chebyshev, ...)
  - ...

[a] Sala, M. G. & Heroux, M. A. ***Robust Algebraic Preconditioners using IFPACK 3.0.*** Tech. rep. SAND2005-0662 (Sandia

Multigrid methods

- Packages: ML[a], MUELU[b]
- Methods:
  - PA-AMG
  - SA-AMG
  - Emin
  - Structured AMG
  - ...

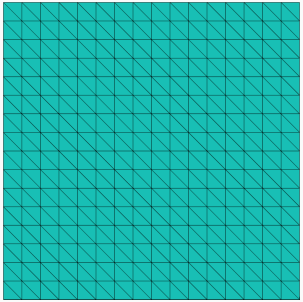
[a] Gee, M. W. *et al.* ***ML 5.0 Smoothed Aggregation User's Guide.*** Tech. rep. SAND2006-2649 (Sandia National Laboratories, Albuquerque, NM (USA) 87185, 2006).

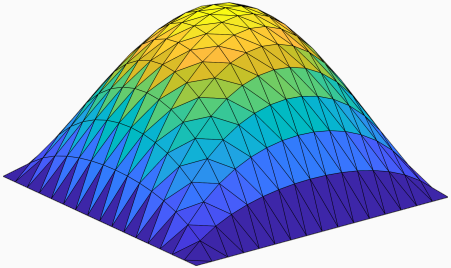
[b] Berger-Vergiat, L. *et al.* ***MueLu User's Guide.***

Multilevel domain decomposition methods

- Packages: SHYLU
- Methods:
  - Overlapping Schwarz, GDSW (FROSCH[a])
  - ...

[a] Heinlein, A. *et al.* ***FROSch: A Fast And Robust Overlapping Schwarz Domain Decomposition Preconditioner Based on Xpetra in Trilinos.*** in *Domain Decomposition Methods in Science and Engineering XXV* (eds Haynes, R. *et al.*) (Springer International Publishing, Cham, 2020), 176–184.



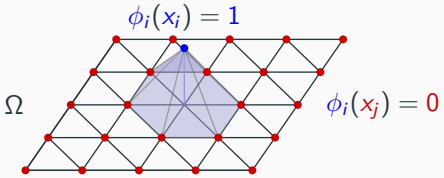


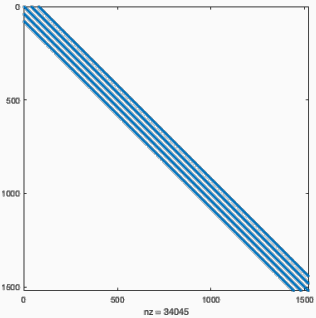
Let us consider the simple **diffusion model problem** ($\alpha(x) = 1$):

$$-\Delta u = f \quad \text{in } \Omega = [0,1]^2,$$
$$u = 0 \quad \text{on } \partial\Omega.$$

Discretization using finite elements yields the linear equation system

$$Ku = f.$$





- Due to the local support of the finite element basis functions, the resulting system is **sparse**.
- However, due to the **superlinear complexity and memory cost**, the use of direct solvers becomes infeasible for fine meshes, that is, for the **resulting large sparse equation systems**.

$\rightarrow$ We will employ iterative solvers:
For our elliptic model problem, the system matrix is symmetric positive definite. Hence, we can use the **conjugate gradient (CG) method**.

Theorem 1

Let $A \in \mathbb{R}^{n \times n}$ be symmetric positive definite. Then the **CG method** *converges and the following error estimate holds:*

$$\left\| e^{(k)} \right\|_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \left\| e^{(0)} \right\|_A,$$

where $\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$.

Do we need a preconditioner?

Theorem 2 (Condition number of the stiffness matrix)

There exists a constant $c > 0$, independent of h, such that

$$\kappa(K) \leq c \frac{h^d}{\left(\min_{T \in \tau_h} h_T\right)^{d+2}}.$$

$\Rightarrow$ **Convergence of the PCG method will deteriorate** when refining the mesh.

The **preconditioned conjugate gradient (PCG)** methods solves instead the preconditioned system

$$M^{-1}Ax = M^{-1}b \qquad \text{or more precisely} \qquad M^{-1/2}AM^{-1/2}x = M^{-1/2}b,$$

with the preconditioner $M^{-1} \approx A^{-1}$. This system is equivalent to the original system

$$Ax = b.$$

but **easier to solve**.

Theorem 3

Let $A \in \mathbb{R}^{n\times n}$ be symmetric positive definite. Then the **PCG method** *converges and the following error estimate holds:*

$$\left\| e^{(k)} \right\|_A \leq 2 \left(\frac{\sqrt{\kappa\left(M^{-1}A\right)} - 1}{\sqrt{\kappa\left(M^{-1}A\right)} + 1} \right)^k \left\| e^{(0)} \right\|_A,$$

where $\kappa\left(M^{-1}A\right) = \frac{\lambda_{\max}\left(M^{-1/2}AM^{-1/2}\right)}{\lambda_{\min}\left(M^{-1/2}AM^{-1/2}\right)}$.

Preconditioned Conjugate Gradient (PCG) Method

Algorithm 1: Preconditioned conjugate gradient method

Result: Approximate solution of the linear equation system $Ax = b$

Given: Initial guess $x^{(0)} \in \mathbb{R}^n$ and tolerance $\varepsilon > 0$

$r^{(0)} := b - Ax^{(0)}$

$p^{(0)} := y^{(0)} := M^{-1} r^{(0)}$

while $\left\| r^{(k)} \right\| \geq \varepsilon \left\| r^{(0)} \right\|$ **do**

$$\alpha_k := \frac{\left(p^{(k)}, r^{(k)}\right)}{\left(Ap^{(k)}, p^{(k)}\right)}$$

$$x^{(k+1)} := x^{(k)} + \alpha_k y^{(k)}$$

$$r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$$

$$y^{(k+1)} := M^{-1} r^{(k+1)}$$

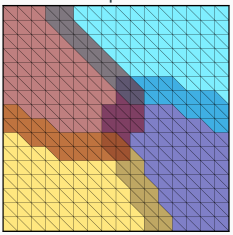
$$\beta_k := \frac{\left(y^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$$

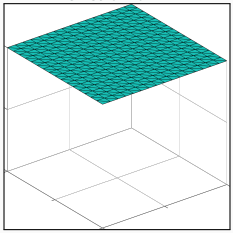
$$p^{(k+1)} := r^{(k+1)} - \beta_k p^{(k)}$$

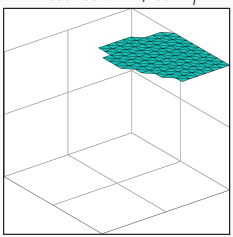
end

Let us use a **one-level Schwarz preconditioner**, which can be **constructed algebraically from the system matrix** $A \rightarrow$ IFPACK (for EPETRA), IFPACK2 (for TPETRA).

One-Level Schwarz Preconditioner







Based on an **overlapping domain decomposition**, we define an additive **one-level Schwarz preconditioner**

$$M_{\text{OS-1}}^{-1} = \sum_{i=1}^{N} R_i^T K_i^{-1} R_i,$$

where R_i and R_i^T are restriction and prolongation operators corresponding to Ω_i', and $K_i := R_i K R_i^T$. The K_i correspond to **local Dirichlet problems** on the overlapping subdomains.

Condition number bound:

$$\kappa\left(M_{\text{OS-1}}^{-1} K\right) \leq C\left(1 + \frac{1}{H\delta}\right)$$

where the constant C is **independent of the subdomain size H and the width of the overlap δ.**

Iterative solvers from the Belos package

- Use Belos::SolverFactory<SC, MV, OP> to create any BELOS solver
  - SC = Scalar type
  - MV = MultiVector type
  - OP = Operator type
- Initiate solver creation via the create() method
  - Select solver via its name passes as std::string
  - Pass solver parameters / configuration via a Teuchos::ParameterList

Example:

```
1 RCP<Teuchos::ParameterList> params = rcp (new ParameterList());
2 params->set("Maximum Iterations", 150);
3 params->set("Convergence Tolerance", 1.0e-6);
4
5 Belos::SolverFactory<SC, MV, OP> belosFactory;
6 RCP<Belos::SolverManager<SC, MV, OP>> solver = belosFactory.create ("GMRES", params);
```

- Pack matrix, left- and right-hand side into a Belos::LinearProblem<SC, MV, OP>
- If desired and available, include the ready-to-use preconditioner
- Pass the linear problem to the solver

```
1 RCP<Belos::LinearProblem<SC, MV, OP>> problem
2     = rcp(new Belos::LinearProblem<SC, MV, OP> (A, x, b));
3 problem->setProblem();
4
5 if (usePreconditioner)
6   problem->setRightPrec(preconditioner);
7
8 solver->setProblem(problem);
```

- Solve the linear system
- Return value indicates the convergence status

Example:

```
1 Belos::ReturnType solveResult = solver->solve();
```

Preconditioners from the Ifpack2 package

- Use Ifpack2::Factory::create<Tpetra::RowMatrix<SC,LO,GO,NO>> to create any IFPACK2 method
  - SC = Scalar type
  - LO = LocalOrdinal type
  - GO = GlobalOrdinal type
  - NO = KOKKOS node type
  - Select method via its name passes as std::string
  - Pass the matrix A

Example:

```
1 RCP<Ifpack2::Preconditioner<SC,LO,GO,NO>> prec
2     = Ifpack2::Factory::create<Tpetra::RowMatrix<SC,LO,GO,NO>> ("RELAXATION", A);
```

- Configure via a Teuchos::ParameterList
- Initialize and compute the preconditioner

Example:

```
1 Teuchos::ParameterList precParams;
2 precParams.set("relaxation: type", relaxationType);
3 precParams.set("relaxation: sweeps", numSweeps);
4 precParams.set("relaxation: damping factor", damping);
5 prec->setParameters(precParams);
6
7 prec->initialize();
8 prec->compute();
```

Disclaimer

Today's remarks on STRATIMIKOS are intended as an outlook for interested users. This package will not be covered in today's tutorial.

What is Stratimikos?

- unified set of Thyra-based wrappers to linear solver and preconditioner capabilities in TRILINOS
- enables solver customization through an xml-input deck

Exemplary input deck for Stratimikos:

```
<ParameterList>
  <Parameter name="Linear Solver Type" type="string" value="Belos"/>
  <ParameterList name="Linear Solver Types">
    <ParameterList name="Belos">
      <Parameter name="Solver Type" type="string" value="Block GMRES"/>
      <ParameterList name="Solver Types">
        <ParameterList name="Block GMRES">
          <Parameter name="Block Size" type="int" value="1"/>
          <Parameter name="Convergence Tolerance" type="double" value="1e-13"/>
          <Parameter name="Num Blocks" type="int" value="300"/>
          <Parameter name="Output Frequency" type="int" value="1"/>
          <Parameter name="Maximum Iterations" type="int" value="400"/>
        </ParameterList>
      </ParameterList>
    </ParameterList>
  </ParameterList>
  <Parameter name="Preconditioner Type" type="string" value="Ifpack"/>
  <ParameterList name="Preconditioner Types">
    <ParameterList name="Ifpack">
      <Parameter name="Prec Type" type="string" value="ILU"/>
      <Parameter name="Overlap" type="int" value="1"/>
      <ParameterList name="Ifpack Settings">
        <Parameter name="fact: level-of-fill" type="int" value="2"/>
      </ParameterList>
    </ParameterList>
  </ParameterList>
</ParameterList>
```

Hands-on exercises

Solve linear systems with a (preconditioned) Krylov solver:

- Complete the app ex_03_solve to solve various linear systems with
  - plain GMRES (without preconditioning)
  - preconditioned GMRES
- Material: exercises/ex_03_solve

References and detailed information on Trilinos

- 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 ressources on GitHub: https://github.com/kokkos