IV. Using Trilinos in application codes - Part I

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.

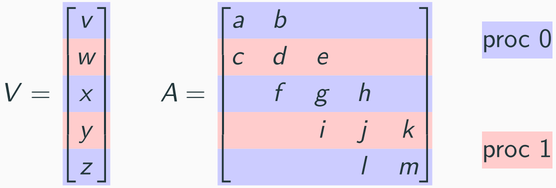
→ TPETRA Doxygen documentation:
https://docs.trilinos.org/dev/packages/tpetra/doc/html/

`Tpetra::Map`

- 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:



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

- 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:

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

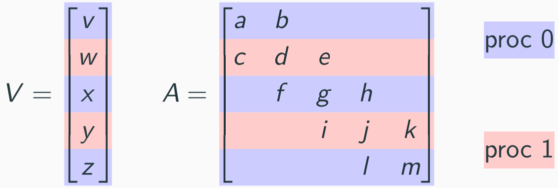
$$\rightarrow V_0 = \begin{bmatrix} v \\ \\ x \\ \\ z \end{bmatrix} \qquad A_0 = \begin{bmatrix} a & b & & & \\ & & & & \\ & f & g & h & \\ & & & & \\ & & & l & m \end{bmatrix}$$

Process 1:

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

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

- 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:

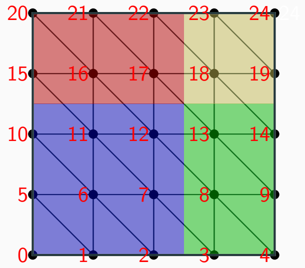


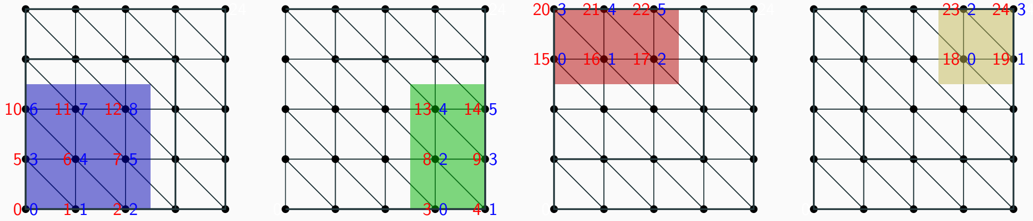
$$V = \begin{bmatrix} v \\ w \\ x \\ y \\ z \end{bmatrix} \qquad A = \begin{bmatrix} a & b & & & \\ c & d & e & & \\ & f & g & h & \\ & & i & j & k \\ & & & l & m \end{bmatrix}$$

- 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





global indices local indices

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**.

$$V = \begin{bmatrix} v \\ w \\ x \\ y \\ z \end{bmatrix} \qquad \begin{array}{l} \text{proc } 0 \\ \\ \\ \text{proc } 1 \end{array}$$

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

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

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.

Tpetra::MultiVector

- 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.

Tpetra::CrsMatrix

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.

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

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

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

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

`Tpetra::CrsMatrix` – Column Map

- 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Not unique means that multiple processes share global indices.

`Tpetra::CrsMatrix` – Matrix assembly

- 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

$$V = \begin{bmatrix} v \\ w \\ x \\ y \\ z \end{bmatrix} \qquad A = \begin{bmatrix} a & b & & & \\ c & d & e & & \\ & f & g & h & \\ & & i & j & k \\ & & & l & m \end{bmatrix} \qquad M_0 = \begin{bmatrix} 0 \\ 2 \\ 4 \end{bmatrix} \quad \text{proc 0} \qquad M_1 = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad \text{proc 1}$$

to

$$V = \begin{bmatrix} v \\ w \\ x \\ y \\ z \end{bmatrix} \qquad A = \begin{bmatrix} a & b & & & \\ c & d & e & & \\ & f & g & h & \\ & & i & j & k \\ & & & l & m \end{bmatrix} \qquad M_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \text{proc 0} \qquad M_1 = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \quad \text{proc 1}$$

- 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):

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

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

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

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

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

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