Photos placed in horizontal position
with even amount of white space
between photos and header

Tpetra and Data Services

Karen Devine, October 24, 2018

Tpetra team

Geoff Danielson
Karen Devine
Tim Fuller
Mark Hoemmen
Jonathan Hu
Kyungjoo Kim
Chris Luchini
Will McLendon
Chris Siefert
Christian Trott

U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. SAND2018-12216PE

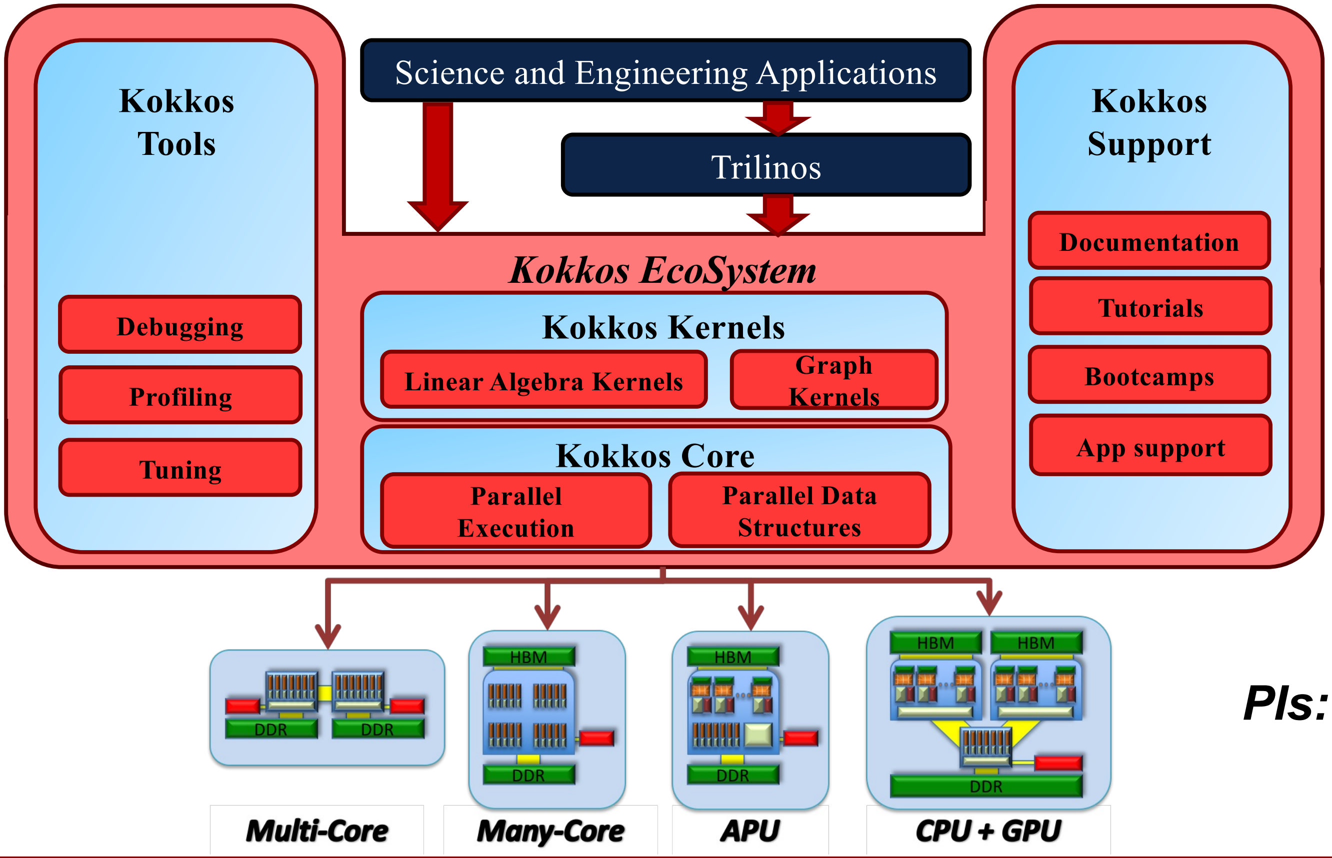
Tpetra FY18 efforts

- New Tpetra team
  - Define high-performance path through Tpetra
  - Reduce complexity of Tpetra code base
  - Increase robustness and usability of Tpetra
- Finite Element Assembly (see talk by Tim Fuller, Chris Siefert)
  - Interfaces, performance optimizations, FEMultiVector, deprecation of DynamicProfile
- Deprecations (see talk by Mark Hoemmen)
  - Template parameters (Global/local ordinal, Node); defaultPlatform
- Documentation
  - User's guide, examples for finite element assembly
- Benchmark performance tests
- https://github.com/trilinos/Trilinos/wiki/Tpetra-Information-Page

Tpetra FY19 Efforts

- New capability to allow run-time decisions about which execution and memory spaces to use during operations
  - Related to Node deprecation
- Application support: performance portability of matrix assembly for GPU and KNL
  - Performance evaluation/remediation in applications; FECrsMatrix
- Removal of deprecated features
  - Targeting Q3 removal; work with applications
- Improved communication performance of Tpetra
  - Import/Export/Distributor

Kokkos and KokkosKernels Ecosystem



Coordinated integration testing, releases, snapshots into Trilinos

PIs: Christian Trott
Siva Rajamanickam

STK Highlights

- STK: Mesh database developed in SIERRA, snapshotted into Trilinos
- Working on improved GPU support in NGP-Mesh (built on Kokkos) this year
  - Integrating into SIERRA-TF, SIERRA-SM, Nalu-Wind
- Trying to be responsive to Trilinos issues as well
  - Recent GPU warnings issues are taking longer than expected to resolve
- Using STK Balance (built on Zoltan2) in SIERRA-TF and SIERRA-SM for dynamic load balancing
  - If others are interested in using this capability, STK team would be happy to partner
- PI: Kendall Pierson

Have you noticed? (Probably not)

- Package deprecations to remove obsolete, unsupported, untested code
- MeshingGenie package – removed
  - Vorocrust (Mohamed Ebeida)
- Stk_classic subpackage – removed
  - STK
- ThreadPool package – PR #3725
  - Kokkos
- Suggestions for others?

Trilinos-wide topics for Developer Day discussion

- UVM / CUDA_LAUNCH_BLOCKING usage throughout Trilinos
  - Currently required in several packages, including Tpetra
  - If not needed in Tpetra, would other packages still need it?
  - Cost/benefit/requirements?
- Deprecation and release schedule
- Developer documentation (doxygen? wiki?)
- Separation of Tpetra and Epetra stacks
- Testing requirements on test-bed platforms (regarding MPI and compiler versions)
- Use of boost in Trilinos – how necessary is it? how remove warnings?

Exceptional service in the national interest

Sandia National Laboratories

Photos placed in horizontal position with even amount of white space between photos and header

Tpetra Finite Element Assembly

Tim Fuller and Chris Siefert, October 24, 2018

Tpetra team

Geoff Danielson
Karen Devine
Tim Fuller
Mark Hoemmen
Jonathan Hu
Kyungjoo Kim
Chris Luchini
Will McLendon
Chris Siefert
Christian Trott

U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. SAND2018-12216PE

Tpetra Finite Element Assembly

- Motivation:
  - Provide matrix/vector construction that supports variety of application scenarios
  - Provide high-performance path on new architectures
  - Reduce maintenance costs within Tpetra code base
- Deprecation of Dynamic Profile
- User interfaces:
  - Type 1: Simple global insertions
  - Type 2: Two-map insertions
  - Type 3: Applications with ghosted elements
- Other optimizations
- Other deprecations

Deprecation of Dynamic Profile

- CrsGraph/CrsMatrix default for construction: DynamicProfile
  - User did not have to provide correct information on number of nonzeros per row before inserting nonzeros

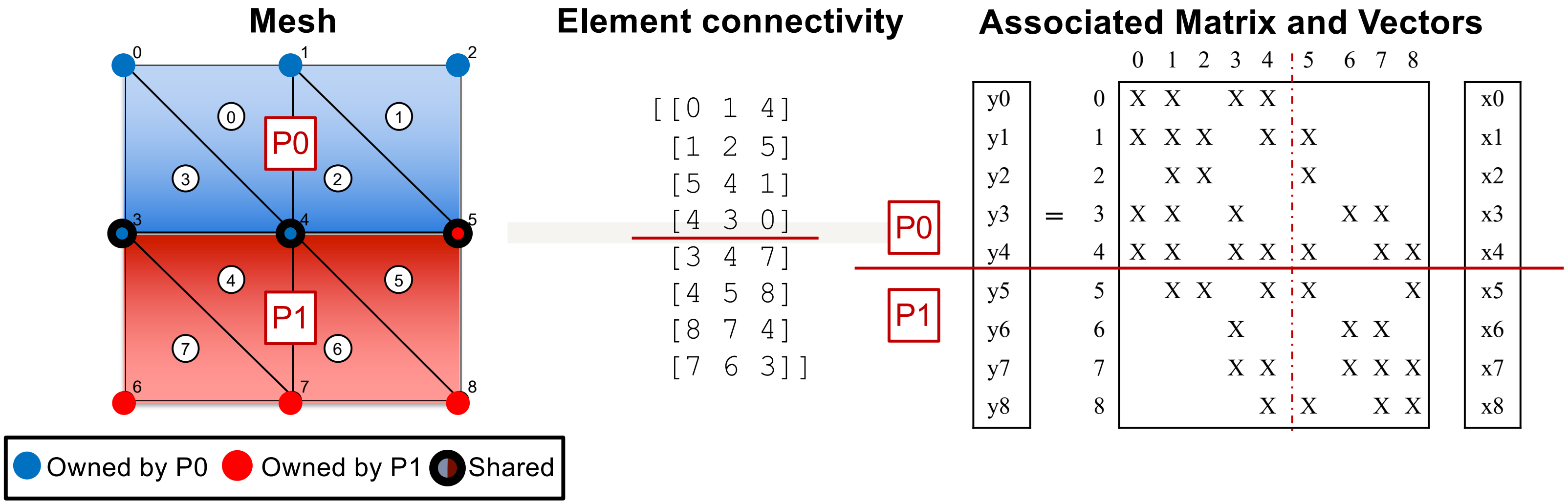
```
CrsMatrix (const Teuchos::RCP<const map_type>& rowMap,
           size_t maxNumEntriesPerRow,
           ProfileType pftype = DynamicProfile,
           const Teuchos::RCP<Teuchos::ParameterList>& params = Teuchos::null);
```

- DynamicProfile requires dynamic memory allocation in Tpetra
  - Higher execution time
  - Current interface is infeasible on GPUs
- DynamicProfile option is being removed now; Trilinos tests being updated.

Use StaticProfile

- StaticProfile is already an option in Tpetra: Applications can use it NOW
  - User provides either *accurate* max number of nonzeros per row OR actual number of nonzeros in each row
  - Tpetra allocates memory once before insertions
  - Tpetra will throw error if counts are insufficient; applications can then increase counts and try again
- Application conversion:
  - Many applications already have needed nonzeros counts; give them to Graph/Matrix constructor with StaticProfile flag
  - Otherwise, applications need to compute nonzero counts and provide them to Graph/Matrix constructor with StaticProfile flag
- Schedule: all Tpetra code changes in place FY19 Q1; DynamicProfile removed Q3.
- Examples in Tpetra repository; will post to wiki

Assumptions for this discussion



- Element-based partition of mesh (i.e., elements owned uniquely by processes)
- DOFs associated with mesh nodes
- Mesh nodes have unique owner but may be copied (“shared”) on many processes
- 1D partition of matrix by rows

Type 1 assembly: Global insertions

- Processor *p* can insert nonzeros into any processor's rows
  - insertGlobalValues(…)
- Tpetra uses map to build directory and send nonzeros to owning processors
  - globalAssemble(…) in fillComplete(…)

- Advantages:
  - Simplest use-case for applications
- Disadvantages:
  - Most expensive path for Tpetra – much off-processor discovery needed

Type 1 assembly: Global insertions

- **Procedure:**

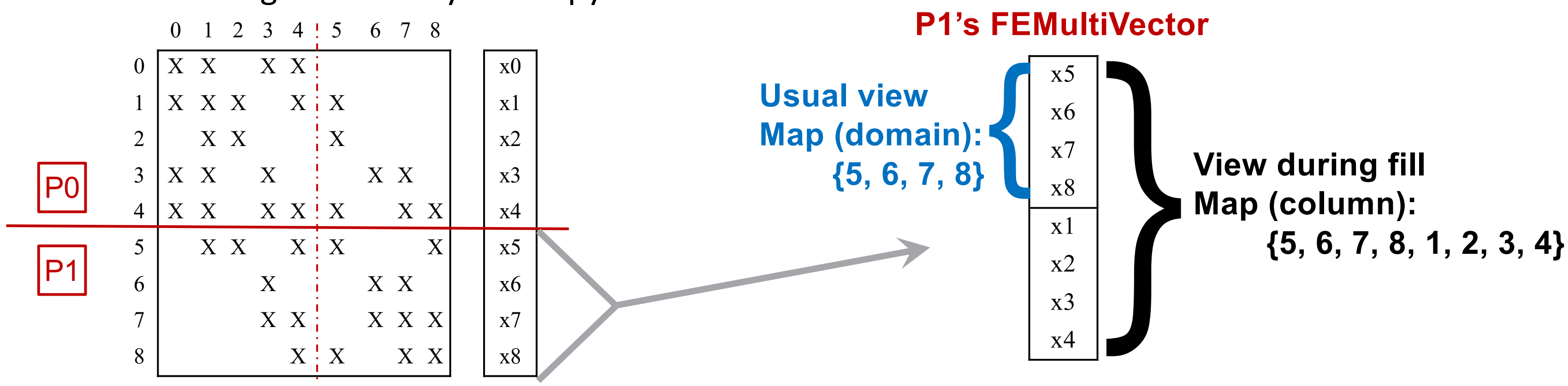
```
1. Construct map
   owned_map = {owned DOFs}
   domain_map = owned_map
   range_map = owned_map

2. Construct graph
   owned_graph = Graph(owned_map)
   for each element e
       for each DOF n of e
           owned_graph.insertGlobalIndices(n, {all DOFs of e})
   owned_graph.fillComplete(domain_map, range_map)

3. Construct matrices
   owned_mat = Matrix(owned_graph)
   for each element e
       for each DOF n of e
           owned_mat.sumIntoGlobalValues(n, {all DOFs of e}, vals)
   owned_mat.fillComplete(domain_map, range_map)
```

MultiVector Assembly:: FEMultiVector

- MultiVector's sumIntoGlobalValue works only for global IDs in MultiVector's map
- But right-hand-side assembly contributes to shared DOF's MultiVector values
- User could manage with multiple maps & multivectors and Export
- Easier alternative: Tpetra::FEMultiVector
- Implementation may or may not have two copies of vector data
  - If maps align, same storage used for both owned and owned+shared
  - Potential savings in memory and copy costs



FEMultiVector Assembly

- **Procedure**:

  1. Build and fillComplete owned_graph CrsGraph as before

  2. Construct FEMultiVector

  ```
  Tpetra::FEMultiVector femv(domain_map, owned_graph.getImporter())
  ```

  3. Fill FEMultiVector

  ```
  femv.beginFill  // enables access to shared DOFs
  for each element e
      for each DOF n of e
          femv.sumIntoGlobalValue(n, val)
  femv.endFill    // exports with ADD; disables access to shared DOFs
  ```

- Notes:
  - Can use with Type 2 assembly as well
  - Coming soon: FECrsGraph, FECrsMatrix

Type 2 assembly: Two maps/graphs/matrices

- I.e., Local Element loop
- FEMultiVector exploited two maps; can do same with matrices

- Build two graphs & two matrices
  - Owned graph and matrix for owned DOFs
  - Shared graph & matrix for shared DOFs
- Export shared graph & matrix to owned graph & matrix

- Advantages:
  - Less discovery needed in Tpetra, so more efficient
- Disadvantages:
  - More complex for user (goal is to alleviate complexity with FECrsGraph, FECrsMatrix)

Type 2 Assembly: Two maps/graphs/matrices

1. Construct two maps and exporter between them

```
owned_map = {Owned DOFs};
shared_map = {Shared DOFs}
exporter = Export(shared_map, owned_map)
```

2. Construct two graphs

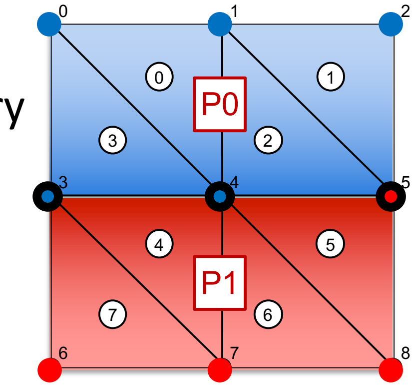
```
owned_graph = Graph(owned_map);
shared_graph = Graph(shared_map)
for each element e
    for each DOF n of e
        if (n is owned) owned_graph.insertGlobalIndices(n, {all DOFs of e})
        else shared_graph.insertGlobalIndices(n, {all DOFs of e})
shared_graph.fillComplete(domain_map, range_map)
owned_graph.doExport(shared_graph, exporter, INSERT)
owned_graph.fillComplete(domain_map, range_map)
```

3. Construct two matrices

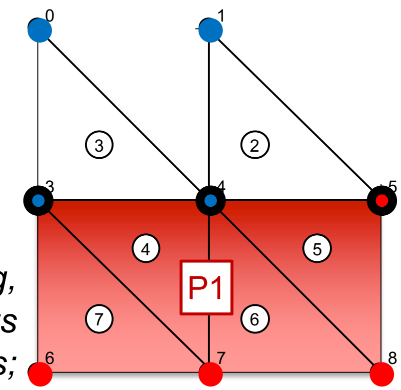
```
owned_mat = Matrix(owned_graph);
shared_mat = Matrix(shared_graph)
<fill owned_mat and shared_mat with values as above>
owned_mat.doExport(shared_mat, exporter, ADD)
owned_mat.fillComplete(domain_map, range_map)
```

Type 3 Assembly: Total element loop

- For applications with a layer of element copies around part boundary
  - Ghost elements, aura, total-element-loop
- All computations for owned DOFs can be done locally
  - No contributions to shared DOFs
- Application needs only owned map
- Application can use local indexing



- Advantage:
  - All insertions are local; no export needed
- Disadvantage:
  - Not all applications have ghost elements



With one layer of ghosting, P1 stores four owned elements and two ghost elements; fills values for DOFs {5, 6, 7, 8}

Type 3 Assembly: Total element loop

- **Procedure**:

```
1. Construct map
   owned_map = {owned DOFs}

2. Construct graph
   owned_graph = Graph(owned_map)
   for each owned and ghost element e
       for each owned DOF n of e
           owned_graph.insertLocalIndices(n, {all DOFs of e})
   owned_graph.fillComplete(domain_map, range_map)

3. Construct matrices
   owned_mat = Matrix(owned_graph)
   for each owned and ghost element e
       for each owned DOF n of e
           owned_mat.sumIntoLocalValues(n, {all DOFs of e}, vals)
   owned_mat.fillComplete(domain_map, range_map)
```

Optimizations: Local indexing and Kokkos::StaticCrsGraph

- Especially easy for Type 3 assembly

- Using local indices to fill graph/matrix/vector
  - avoids conversion from global to local through lookup table
  - E.g., `insertLocalIndices()`, `insertLocalValues()`

- Can provide three Kokkos::Views for CRS arrays (rowOffsets, columnIndices, values) and row/column maps to CrsMatrix constructor
  - See Tpetra tutorial Trilinos/packages/tpetra/core/example/Lesson07-Kokkos-Fill

- Similarly, can provide Kokkos::StaticCrsGraph

Optimizations: exploit mesh info

- Many applications already know owning processor of shared mesh nodes
- Allow Tpetra to use this information when available

```
1. Construct map
   row_map = {owned DOFs}
   Import importer (row_map, {owned+shared DOFs},
                             {processors owning the DOFs})
   column_map = importer.getTargetMap()

2. Construct graph
   owned_graph = Graph(row_map, column_map)
   …
```

- Nalu simulation of a Vestas wind turbine on 12,288 Haswell cores on NERSC's Cori
  - reduced matrix initialization costs from 109 seconds to 84.5 seconds (22.5% reduction)

Optimizations: column map layout

- If specifying column map, Aztec-style layout of column map may be best
  - Owned entries first
  - Followed by shared entries, grouped by owning processor
- Allows fewer copies during MPI communication
  - Communicate directly to/from memory
  - No need to gather into buffers
- May reduce communication time in MatVec
- `makeOptimizedColMapAndImport` creates new column map from row map and old column map

Optimizations: Contiguous DOF numbering

- Tpetra allows arbitrary numbering of owned rows
  - Directory used to lookup off-processor IDs
- Default Trilinos ordering is most efficient
  - Rows 0 to N0 on rank zero
  - Rows N0+1 to N1 on rank one
  - Rows N1+1 to N2 on rank two
  - Etc.
- Enables fast, simple lookup of IDs during Import/Export construction
  - Directory is trivial

Resources

- Tpetra tutorials
  - Trilinos/packages/tpetra/core/examples/tutorial
- Finite-element assembly examples (Types 1, 2, 3)
  - Trilinos/packages/tpetra/core/examples/Finite-Element-Assembly
- Tpetra wiki
  - https://github.com/trilinos/Trilinos/wiki/Tpetra-Information-Page
- FY18 Tpetra team

Geoff Danielson	Mark Hoemmen	Chris Luchini	Christian Trott
Karen Devine	Jonathan Hu	Will McLendon	
Tim Fuller	Kyungjoo Kim	Chris Siefert	