Exceptional service in the national interest

FROM EPETRA TO TPETRA

Migrating Your Code with Trilinos

Christopher Siefert, Carl Pearson and Timothy Fuller

EuroTUG '24

SAND2024-07785C

WHY THE TPETRA TRANSITION?

WHY THE TPETRA TRANSITION?

- Epetra does not…
  - Work on GPU platforms (or work well with OpenMP).
  - Handle over 2.1B unknowns.
  - Allow for non-double Scalar types (e.g. float, complex, AD types).
  - Make optimal use of vendor TPLs.
- Tpetra does all of these (and more).
- Epetra stack to be removed from Trilinos in FY25.

TPETRA STACK IS TESTED NIGHTLY ON A LOT OF PLATFORMS

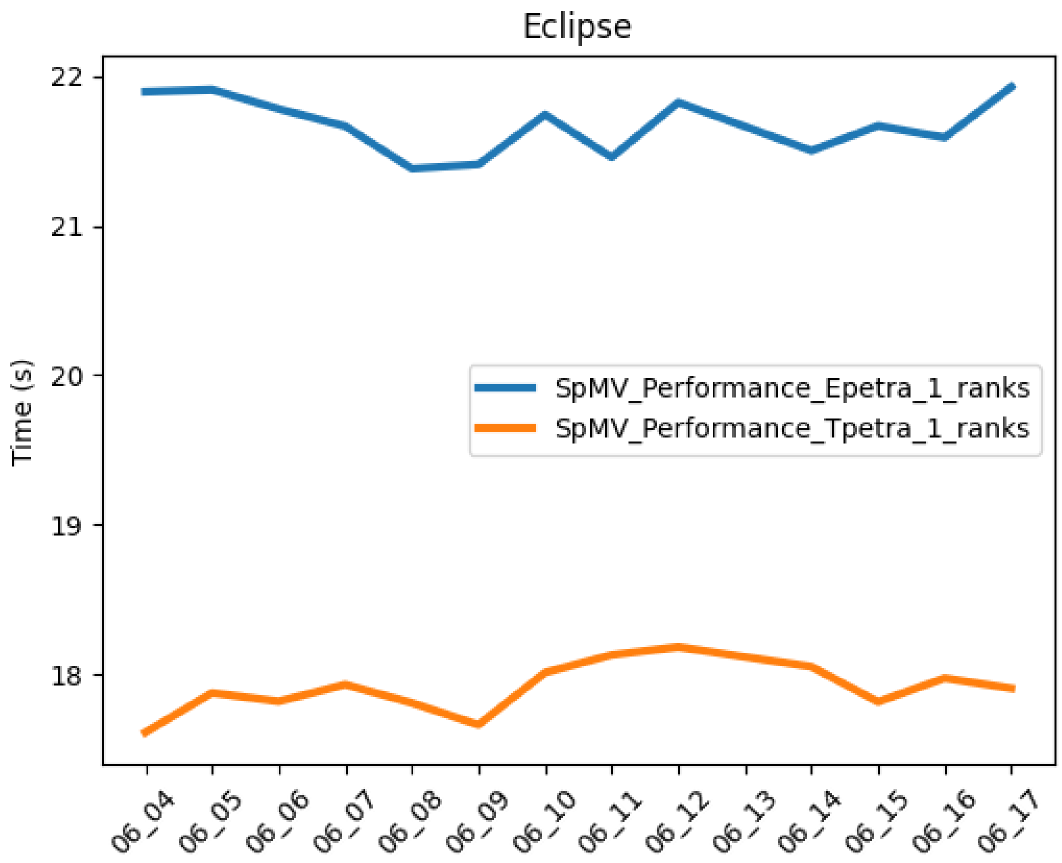
CPU
Amber (Sapphire Rapids/DDR)
Eclipse (Broadwell)
Rocinante (Sapphire Rapids/HBM)
Stria (Arm Cavium Thunder-X2)

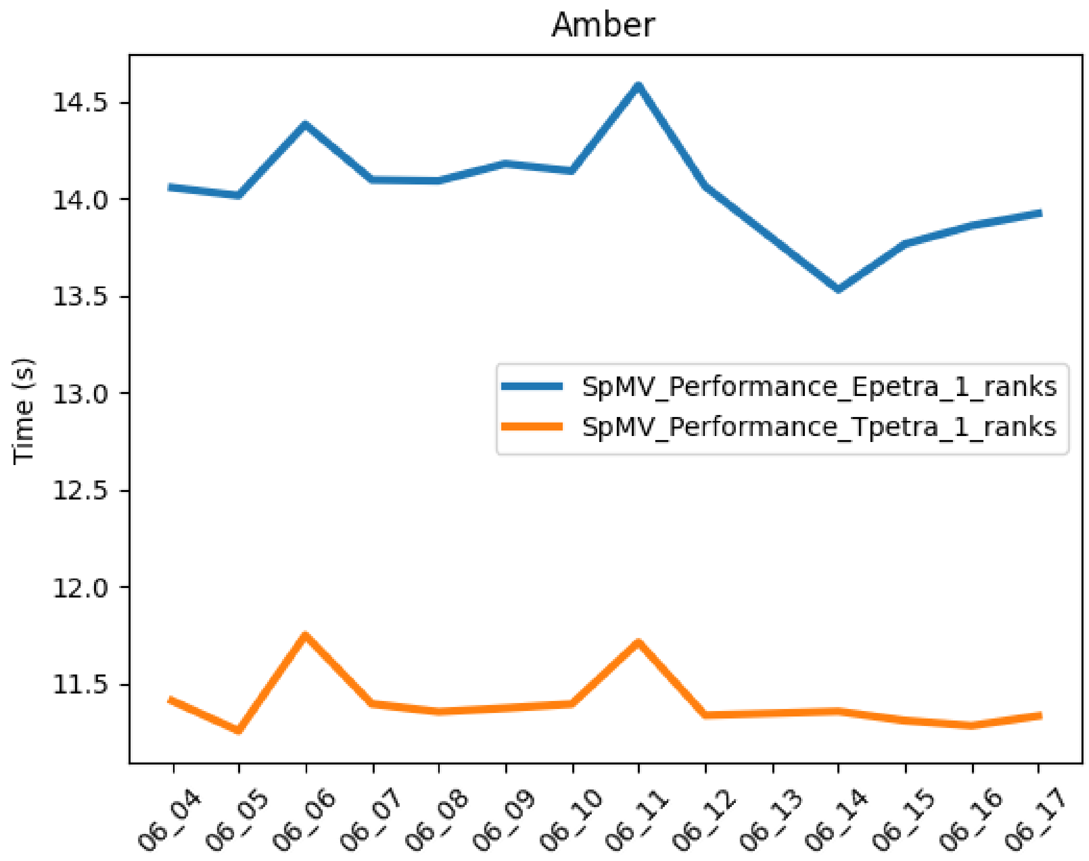
NVIDIA GPU
Vortex (V100)
Perlmutter (A100)
Hops (H100)

AMD GPU
Frontier (MI250)
Tioga (MI250)
RZVernal (MI300)

Intel GPU
Sunspot (Intel Max) coming soon

EVEN ON CPUS, TPETRA IS FASTER





Tpetra is roughly 20% faster for SpMV!

EPETRA TO TPETRA WITHOUT KOKKOS

CASE STUDY: SNL'S ALEGRA CODE

- Goal
  - Move from Epetra/AztecOO/ML to Tpetra/Belos/MueLu.
  - Enable > 2.1 billion unknowns.
  - Not interested in GPU porting.

- Alegra's Strategy
  - Support *both* Epetra and Tpetra via runtime switch.
  - Compare results w/ nightly testing.
  - Work with Trilinos developers to develop missing capabilities.

EPETRA-TO-TPETRA: MAP

```
Epetra_Map * map = new Epetra_Map(-1,num_owned, &(indices.front()), 0, Comm);
```

```
using LO = Tpetra::Map<>::local_ordinal_type;
using GO = Tpetra::Map<>::global_ordinal_type;
using NT = Tpetra::Map<>::node_type;
const Tpetra::global_size_t invalid = Teuchos::OrdinalTraits<Tpetra::global_size_t>::invalid();
using map_type = Tpetra::Map<LO,GO,NT>;

auto map = Teuchos::rcp(new map_type(invalid,indices(),0,comm));
auto map_owned_plus_shared = Teuchos::rcp(new
        map_type(invalid,indices_owned_plus_shared(),0,comm));
```

EPETRA-TO-TPETRA: FECRSGRAPH

```
Epetra_FECrsGraph * graph = new Epetra_FECrsGraph(Copy,*map,&nnz.front());
for (auto item : items) {
    graph->InsertGlobalIndices(1,&grid, gcids.size(), &gcids.front());
}
graph->GlobalAssemble();  graph->OptimizeStorage();
```

```
using graph_type = Tpetra::FECrsGraph<LO,GO,NT>;
auto graph = Teuchos::rcp(new graph_type(map,map_owned_plus_shared,MAX_NNZ_ROW));
Tpetra::beginAssembly(*graph);
for (auto item : items) {
    graph->insertGlobalIndices(grid, gcids());
}
Tpetra::endAssembly(*graph);
```

EPETRA-TO-TPETRA: FEVECTOR

```
Epetra_FEVector * vec = new Epetra_FEVector(*map);
for (int i=0; i<vec->MyLength(); i++)
        (*vec)[0][i] = <whatever>;
```

```
using my_type = Tpetra::FEMultiVector<SC,LO,GO,NT>;
auto vec = Teuchos::rcp(new mv_type(map,graph->getImporter(),1));
size_t len = vec->getMap()->getLocalNumElements();
auto data = vec->get1dViewNonConst();

for (int i=0; i<len; i++)
  data[i] = <whatever>;
```

EPETRA-TO-TPETRA: FECRSMATRIX

```
Epetra_FECrsMatrix * matrix = new Epetra_FECrsMatrix(Copy,*graph);
for (auto item : items) {
    matrix->ExtractGlobalRowView(grid,num_entries,values,indices);
    for(int j=0; j<num_entries; j++) values[j] = <whatever>;
}
matrix->OptimizeStorage();
```

```
using matrix_type = Tpetra::FECrsMatrix<SC,LO,GO,NT>;
auto matrix = Teuchos::rcp(new matrix_type(graph));
Tpetra::beginAssembly(*matrix);
for (auto item : items) {
    matrix->getLocalRowView(grid, indices,values);
    for(int j=0; j<indices.size(); j++) values[j] = <whatever>;
}
Tpetra::endAssembly(*matrix);
```

LESSONS LEARNED

- For performance reasons, Tpetra requires you do more work than Epetra does
  - Max nnz per row is required for graphs (avoid in-loop dynamic memory allocation)
  - FECrsGraph: owned_plus_shared map avoid link-list structures.

- Use typedefs/using: You can wind up with loooooooong code lines otherwise.

- Alegra team worked with Trilinos devs to improve user interfaces and add needed features to Belos & MueLu.
  - Alegra was the one of the first users of the Tpetra::FE* tools, so they helped design better interfaces.
  - Some new features were “compatibility modes” in the new stack, others were algorithms which hadn’t yet been ported to Tpetra.

ADDING KOKKOS TO TPETRA

TPETRA’S FINITE ELEMENT ASSEMBLY EXAMPLE

- Background on Kokkos
  - Kokkos Lectures & Slides (https://kokkos.github.io/kokkos-core-wiki/videolectures.html)
  - Single code for all CPUs, GPUs, and whatever else Kokkos supports.

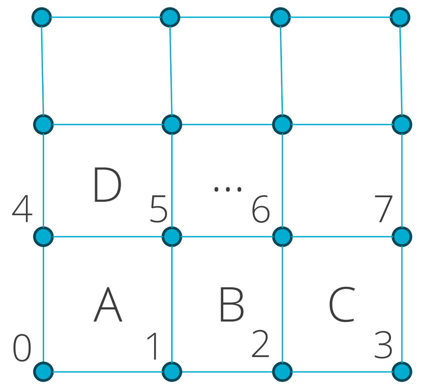
- Example: Trilinos/packages/tpetra/core/example/Finite-Element-Assembly

- Application provides (our example mocks these)
  - Map of elements to nodes in global indices.
  - Methods for computing element matrices.

- Type-1 assembly
  - Local elements contribute to off-rank FE matrix rows for off-rank nodes.
  - Does not require ghosting of element state.

- No worksetting

FE ASSEMBLY IN BRIEF



app. discretization

varies per element, fixed in this example

$$\begin{bmatrix} 2 & -1 & & -1 \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ -1 & & -1 & 2 \end{bmatrix}$$

app. element matrix

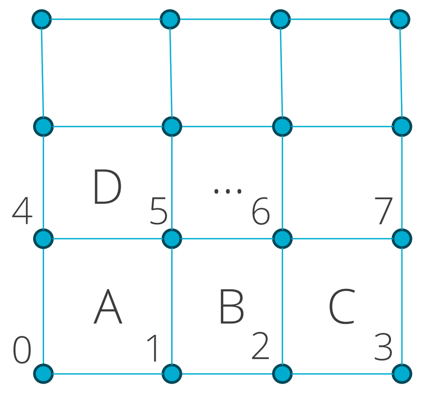
A → {0, 1, 5, 4}
B → {1, 2, 6, 5}
C → {2, 3, 7, 6}
…

app. element-to-node map

$$\begin{array}{c|cccccccc c} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \cdots \\ \hline 0 & 2 & -1 & & & -1 & & & & \\ 1 & -1 & 2 & & & & -1 & & & \\ 2 & & & & & & & & & \\ 3 & & & & & & & & & \\ 4 & -1 & & & & 2 & -1 & & & \\ 5 & & -1 & & & -1 & 2 & & & \\ 6 & & & & & & & & & \\ 7 & & & & & & & & & \\ \vdots & & & & & & & & & \end{array}$$

Global FE matrix

ELEMENT B'S CONTRIBUTION



app. discretization

$$\begin{bmatrix} 2 & -1 & & -1 \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ -1 & & -1 & 2 \end{bmatrix}$$

app. element matrix

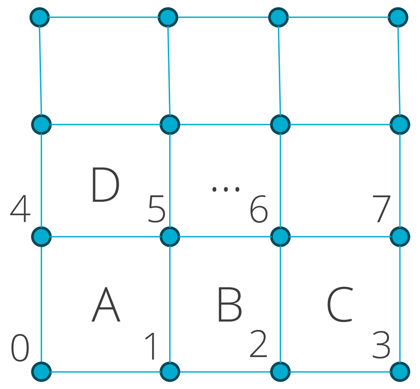
A ⟶ {0, 1, 5, 4}
B ⟶ {1, 2, 6, 5}
C ⟶ {2, 3, 7, 6}
...

app. element-to-node map

$$\begin{array}{c|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 0 & 2 & -1 & & & -1 & & & \\ 1 & -1 & 4 & -1 & & & -2 & & \\ 2 & & -1 & 2 & & & & -1 & \\ 3 & & & & & & & & \\ 4 & -1 & & & & 2 & -1 & & \\ 5 & & -2 & & & -1 & 4 & -1 & \\ 6 & & & -1 & & & -1 & 2 & \\ 7 & & & & & & & & \end{array}$$

Global FE matrix

ELEMENT C'S CONTRIBUTION



app. discretization

$$\begin{bmatrix} 2 & -1 & & -1 \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ -1 & & -1 & 2 \end{bmatrix}$$

app. element matrix

A $\longrightarrow$ {0, 1, 5, 4}
B $\longrightarrow$ {1, 2, 6, 5}
C $\longrightarrow$ {2, 3, 7, 6}
...

app. element-to-node map

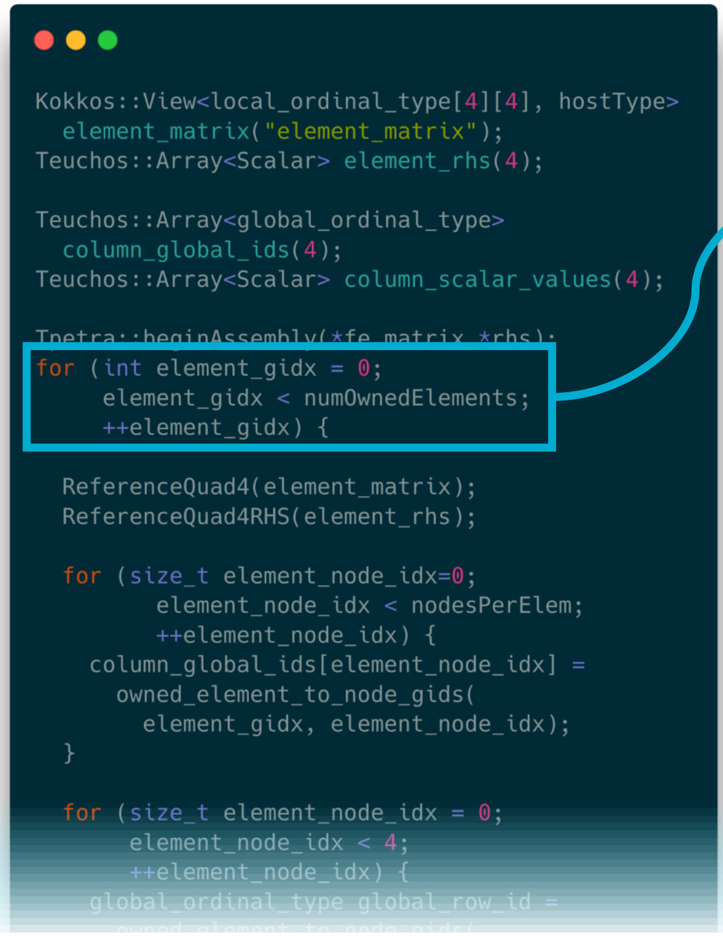
$$\begin{array}{c|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 0 & 2 & -1 & & & -1 & & & \\ 1 & -1 & 4 & -1 & & & -2 & & \\ 2 & & -1 & 4 & -1 & & & -2 & \\ 3 & & & -1 & 2 & & & & -1 \\ 4 & -1 & & & & 2 & -1 & & \\ 5 & & -2 & & & -1 & 4 & -1 & \\ 6 & & & -2 & & & -1 & 4 & -1 \\ 7 & & & & -1 & & & -1 & 2 \end{array}$$

Global FE matrix

FIVE CHANGES TO WATCH OUT FOR

• Bare for-loops	• Kokkos::parallel_for • Allow device execution • Supports CPU execution too
• Host allocations	• Kokkos::View
• Functions	• KOKKOS_FUNCTION annotation
• Tpetra::[…]::getHostView() • Convenient use of global indices	• Tpetra::[]::getDeviceView() • Easier to use local indices “on device”
• Normal addition	• Atomic addition

HOST LOOP -> KOKKOS::PARALLEL_FOR



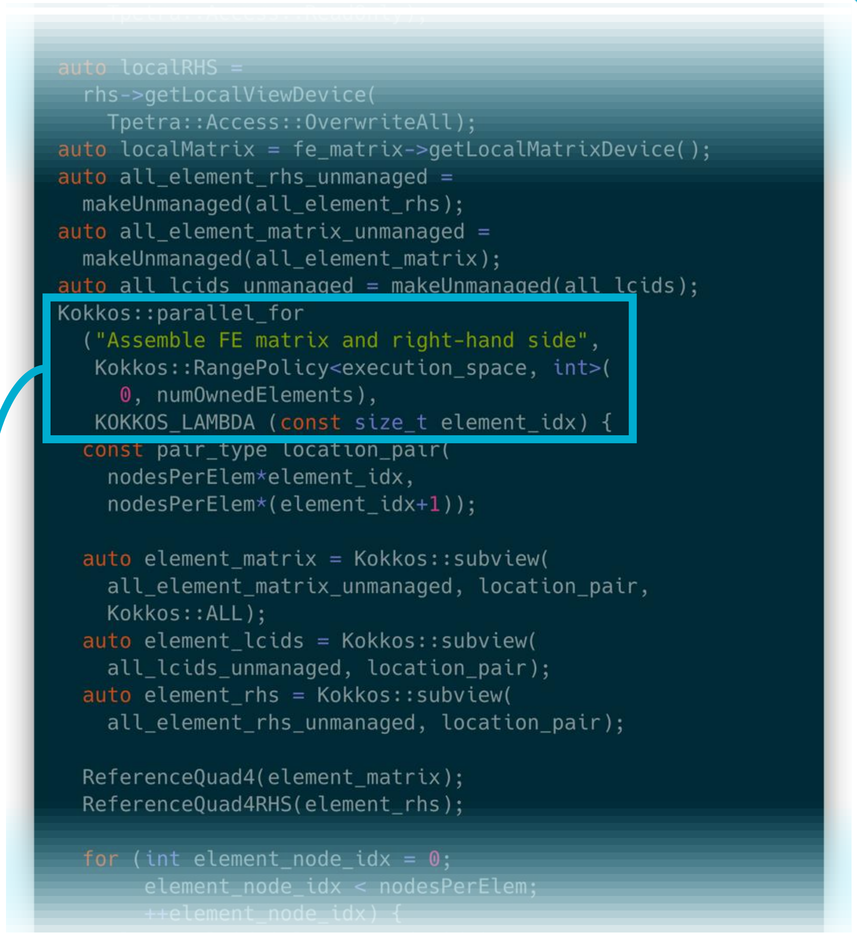
```
Kokkos::View<local_ordinal_type[4][4], hostType>
  element_matrix("element_matrix");
Teuchos::Array<Scalar> element_rhs(4);

Teuchos::Array<global_ordinal_type>
  column_global_ids(4);
Teuchos::Array<Scalar> column_scalar_values(4);

Tpetra::beginAssembly(*fe_matrix,*rhs);
for (int element_gidx = 0;
     element_gidx < numOwnedElements;
     ++element_gidx) {

  ReferenceQuad4(element_matrix);
  ReferenceQuad4RHS(element_rhs);

  for (size_t element_node_idx=0;
         element_node_idx < nodesPerElem;
         ++element_node_idx) {
    column_global_ids[element_node_idx] =
      owned_element_to_node_gids(
        element_gidx, element_node_idx);
  }

  for (size_t element_node_idx = 0;
       element_node_idx < 4;
       ++element_node_idx) {
    global_ordinal_type global_row_id =
      owned_element_to_node_gids(
```

Tpetra

Tpetra: a single active thread, loops over each local element

Kokkos: operate on local elements in parallel

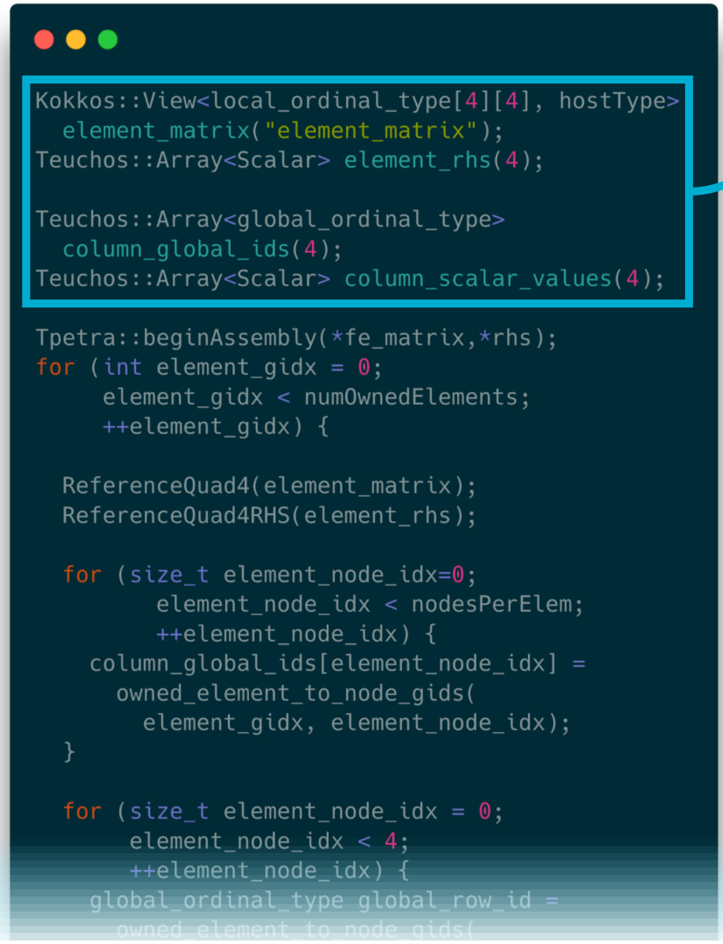
Also works on host, single-threaded (`Kokkos::Serial`) or multi-threaded (`::OpenMP`, `::Threads`)



```
    Tpetra::Access::ReadOnly);

auto localRHS =
  rhs->getLocalViewDevice(
    Tpetra::Access::OverwriteAll);
auto localMatrix = fe_matrix->getLocalMatrixDevice();
auto all_element_rhs_unmanaged =
  makeUnmanaged(all_element_rhs);
auto all_element_matrix_unmanaged =
  makeUnmanaged(all_element_matrix);
auto all_lcids_unmanaged = makeUnmanaged(all_lcids);
Kokkos::parallel_for
  ("Assemble FE matrix and right-hand side",
   Kokkos::RangePolicy<execution_space, int>(
     0, numOwnedElements),
   KOKKOS_LAMBDA (const size_t element_idx) {
  const pair_type location_pair(
    nodesPerElem*element_idx,
    nodesPerElem*(element_idx+1));

  auto element_matrix = Kokkos::subview(
    all_element_matrix_unmanaged, location_pair,
    Kokkos::ALL);
  auto element_lcids = Kokkos::subview(
    all_lcids_unmanaged, location_pair);
  auto element_rhs = Kokkos::subview(
    all_element_rhs_unmanaged, location_pair);

  ReferenceQuad4(element_matrix);
  ReferenceQuad4RHS(element_rhs);

  for (int element_node_idx = 0;
         element_node_idx < nodesPerElem;
         ++element_node_idx) {
```

Tpetra + Kokkos

HOST ALLOCATIONS -> KOKKOS VIEW



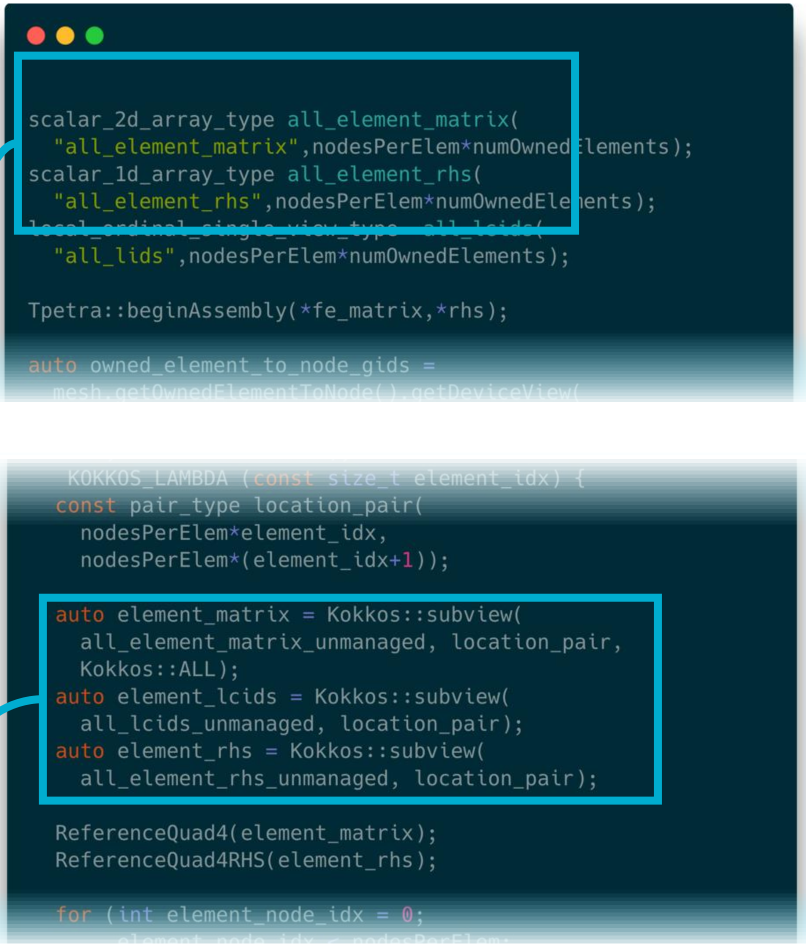
```
Kokkos::View<local_ordinal_type[4][4], hostType>
  element_matrix("element_matrix");
Teuchos::Array<Scalar> element_rhs(4);

Teuchos::Array<global_ordinal_type>
  column_global_ids(4);
Teuchos::Array<Scalar> column_scalar_values(4);

Tpetra::beginAssembly(*fe_matrix,*rhs);
for (int element_gidx = 0;
     element_gidx < numOwnedElements;
     ++element_gidx) {

  ReferenceQuad4(element_matrix);
  ReferenceQuad4RHS(element_rhs);

  for (size_t element_node_idx=0;
         element_node_idx < nodesPerElem;
         ++element_node_idx) {
    column_global_ids[element_node_idx] =
      owned_element_to_node_gids(
        element_gidx, element_node_idx);
  }

  for (size_t element_node_idx = 0;
        element_node_idx < 4;
        ++element_node_idx) {
    global_ordinal_type global_row_id =
      owned_element_to_node_gids(
```

Tpetra

Tpetra: allocate some scratch space on the stack. Reused for each iteration of element loop

Kokkos: allocate enough device memory for all active threads

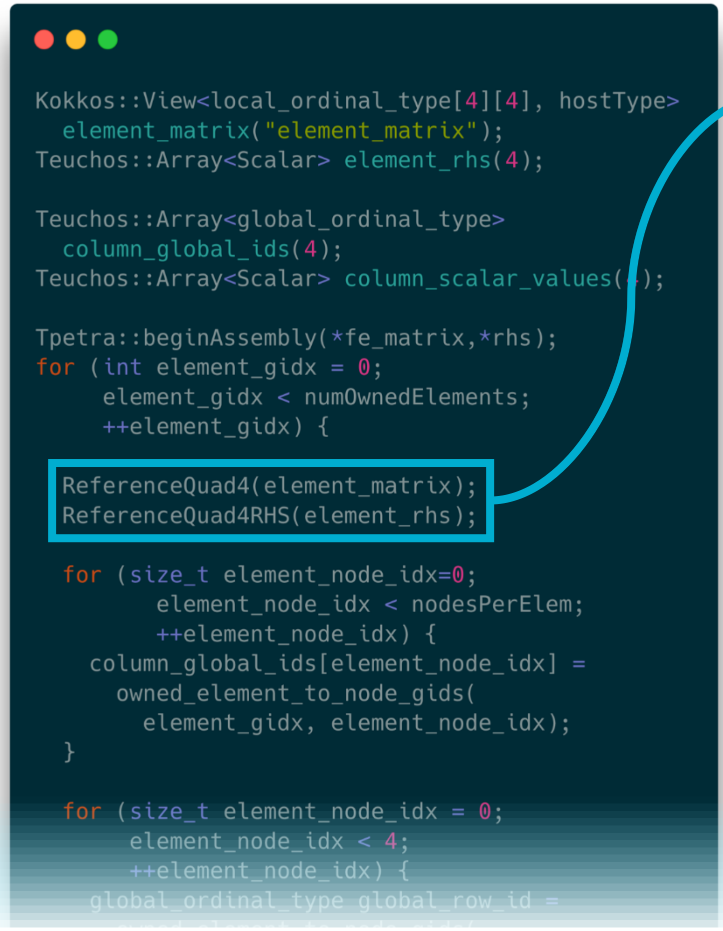
Kokkos: each thread gets its own piece of the preallocated scratch space



```
scalar_2d_array_type all_element_matrix(
  "all_element_matrix",nodesPerElem*numOwnedElements);
scalar_1d_array_type all_element_rhs(
  "all_element_rhs",nodesPerElem*numOwnedElements);
local_ordinal_single_view_type  all_lcids(
  "all_lids",nodesPerElem*numOwnedElements);

Tpetra::beginAssembly(*fe_matrix,*rhs);

auto owned_element_to_node_gids =
  mesh.getOwnedElementToNode().getDeviceView(

   KOKKOS_LAMBDA (const size_t element_idx) {
  const pair_type location_pair(
    nodesPerElem*element_idx,
    nodesPerElem*(element_idx+1));

  auto element_matrix = Kokkos::subview(
    all_element_matrix_unmanaged, location_pair,
    Kokkos::ALL);
  auto element_lcids = Kokkos::subview(
    all_lcids_unmanaged, location_pair);
  auto element_rhs = Kokkos::subview(
    all_element_rhs_unmanaged, location_pair);

  ReferenceQuad4(element_matrix);
  ReferenceQuad4RHS(element_rhs);

  for (int element_node_idx = 0;
         element_node_idx < nodesPerElem;
```

Tpetra + Kokkos

HOST FUNCTIONS -> KOKKOS_FUNCTION



```
Kokkos::View<local_ordinal_type[4][4], hostType>
  element_matrix("element_matrix");
Teuchos::Array<Scalar> element_rhs(4);

Teuchos::Array<global_ordinal_type>
  column_global_ids(4);
Teuchos::Array<Scalar> column_scalar_values(4);

Tpetra::beginAssembly(*fe_matrix,*rhs);
for (int element_gidx = 0;
     element_gidx < numOwnedElements;
     ++element_gidx) {

  ReferenceQuad4(element_matrix);
  ReferenceQuad4RHS(element_rhs);

  for (size_t element_node_idx=0;
          element_node_idx < nodesPerElem;
          ++element_node_idx) {
    column_global_ids[element_node_idx] =
      owned_element_to_node_gids(
        element_gidx, element_node_idx);
  }

  for (size_t element_node_idx = 0;
       element_node_idx < 4;
       ++element_node_idx) {
    global_ordinal_type global_row_id =
```

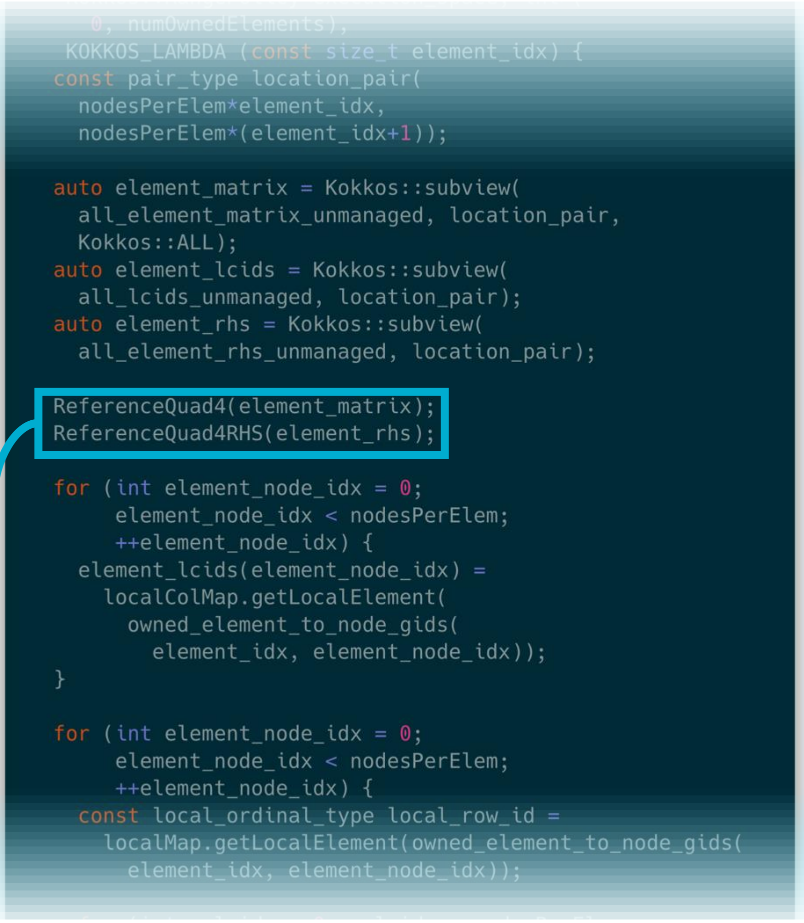
Tpetra

Tpetra: fill scratch space with matrix for this element

Kokkos: each thread fills scratch space in parallel

These functions must be allowed to execute on the device

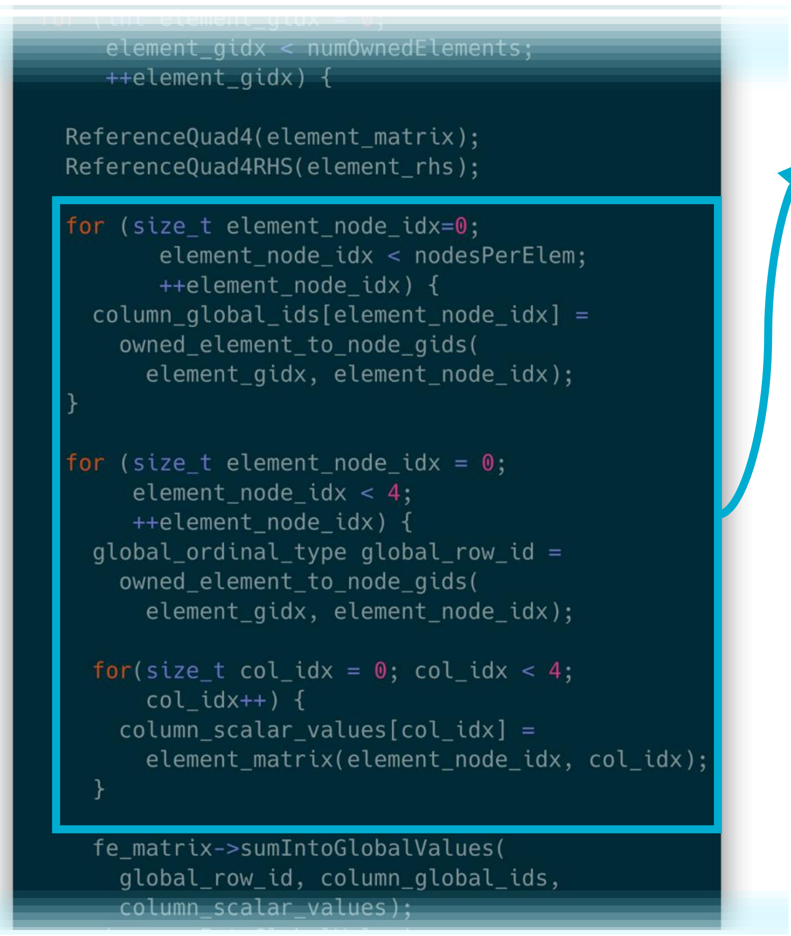
```
KOKKOS_FUNCTION
void Reference4Quad(...) {
  ...
};
```



```
    0, numOwnedElements),
  KOKKOS_LAMBDA (const size_t element_idx) {
  const pair_type location_pair(
    nodesPerElem*element_idx,
    nodesPerElem*(element_idx+1));

  auto element_matrix = Kokkos::subview(
    all_element_matrix_unmanaged, location_pair,
    Kokkos::ALL);
  auto element_lcids = Kokkos::subview(
    all_lcids_unmanaged, location_pair);
  auto element_rhs = Kokkos::subview(
    all_element_rhs_unmanaged, location_pair);

  ReferenceQuad4(element_matrix);
  ReferenceQuad4RHS(element_rhs);

  for (int element_node_idx = 0;
       element_node_idx < nodesPerElem;
       ++element_node_idx) {
    element_lcids(element_node_idx) =
      localColMap.getLocalElement(
        owned_element_to_node_gids(
          element_idx, element_node_idx));
  }

  for (int element_node_idx = 0;
       element_node_idx < nodesPerElem;
       ++element_node_idx) {
    const local_ordinal_type local_row_id =
      localMap.getLocalElement(owned_element_to_node_gids(
        element_idx, element_node_idx));
```

Tpetra + Kokkos

GLOBAL INDICES -> LOCAL INDICES



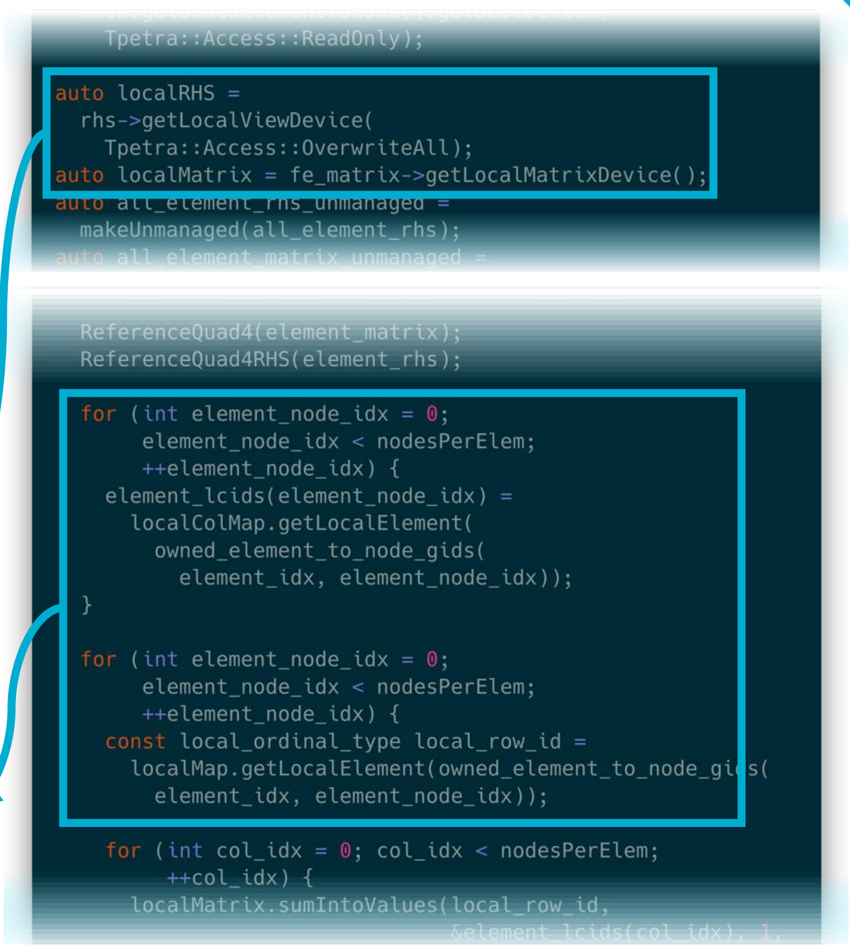
```
for (int element_gidx = 0;
     element_gidx < numOwnedElements;
     ++element_gidx) {

  ReferenceQuad4(element_matrix);
  ReferenceQuad4RHS(element_rhs);

  for (size_t element_node_idx=0;
       element_node_idx < nodesPerElem;
       ++element_node_idx) {
    column_global_ids[element_node_idx] =
      owned_element_to_node_gids(
        element_gidx, element_node_idx);
  }

  for (size_t element_node_idx = 0;
      element_node_idx < 4;
      ++element_node_idx) {
    global_ordinal_type global_row_id =
      owned_element_to_node_gids(
        element_gidx, element_node_idx);

    for(size_t col_idx = 0; col_idx < 4;
        col_idx++) {
      column_scalar_values[col_idx] =
        element_matrix(element_node_idx, col_idx);
    }

    fe_matrix->sumIntoGlobalValues(
      global_row_id, column_global_ids,
      column_scalar_values);
```

Tpetra

Tpetra: interact with FE matrix and RHS through global indices. Simpler interface, made possible by dynamic memory allocation

Kokkos: need device views of the local FE matrix and RHS to operate on.

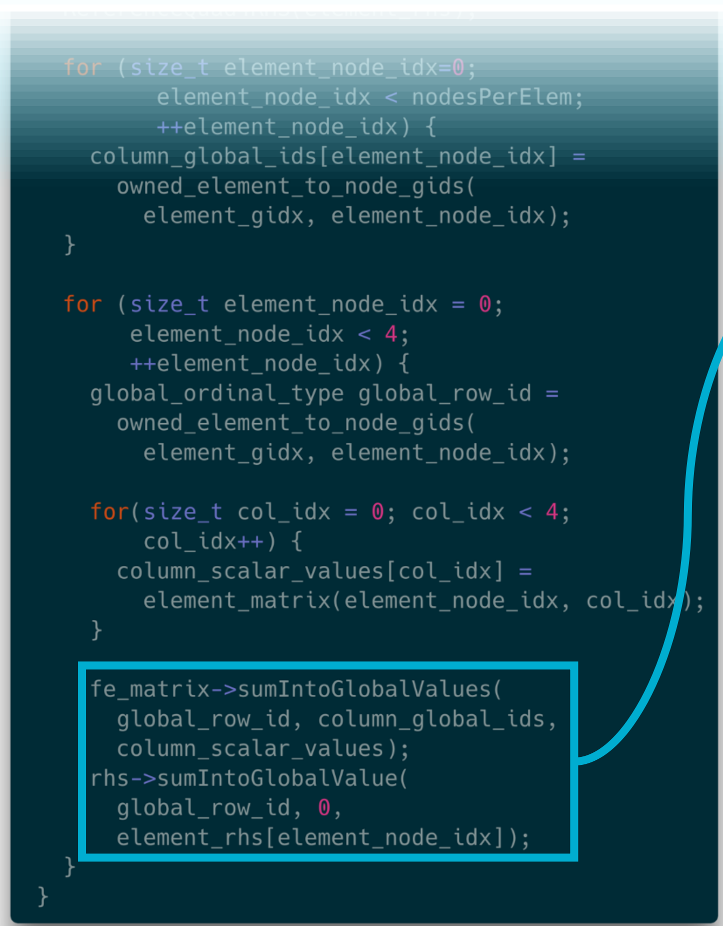
Kokkos: local FE matrix and RHS only understands local indices. Use `Tpetra::Maps` to translate between local and global



```
  mesh.getOwnedElementToNode().getDeviceView(
    Tpetra::Access::ReadOnly);

auto localRHS =
  rhs->getLocalViewDevice(
    Tpetra::Access::OverwriteAll);
auto localMatrix = fe_matrix->getLocalMatrixDevice();
auto all_element_rhs_unmanaged =
  makeUnmanaged(all_element_rhs);
auto all_element_matrix_unmanaged =

  ReferenceQuad4(element_matrix);
  ReferenceQuad4RHS(element_rhs);

  for (int element_node_idx = 0;
       element_node_idx < nodesPerElem;
       ++element_node_idx) {
    element_lcids(element_node_idx) =
      localColMap.getLocalElement(
        owned_element_to_node_gids(
          element_idx, element_node_idx));
  }

  for (int element_node_idx = 0;
       element_node_idx < nodesPerElem;
       ++element_node_idx) {
    const local_ordinal_type local_row_id =
      localMap.getLocalElement(owned_element_to_node_gids(
        element_idx, element_node_idx));

    for (int col_idx = 0; col_idx < nodesPerElem;
         ++col_idx) {
      localMatrix.sumIntoValues(local_row_id,
                                &element_lcids(col_idx), 1,
```

Tpetra + Kokkos

ATOMIC ADDITION

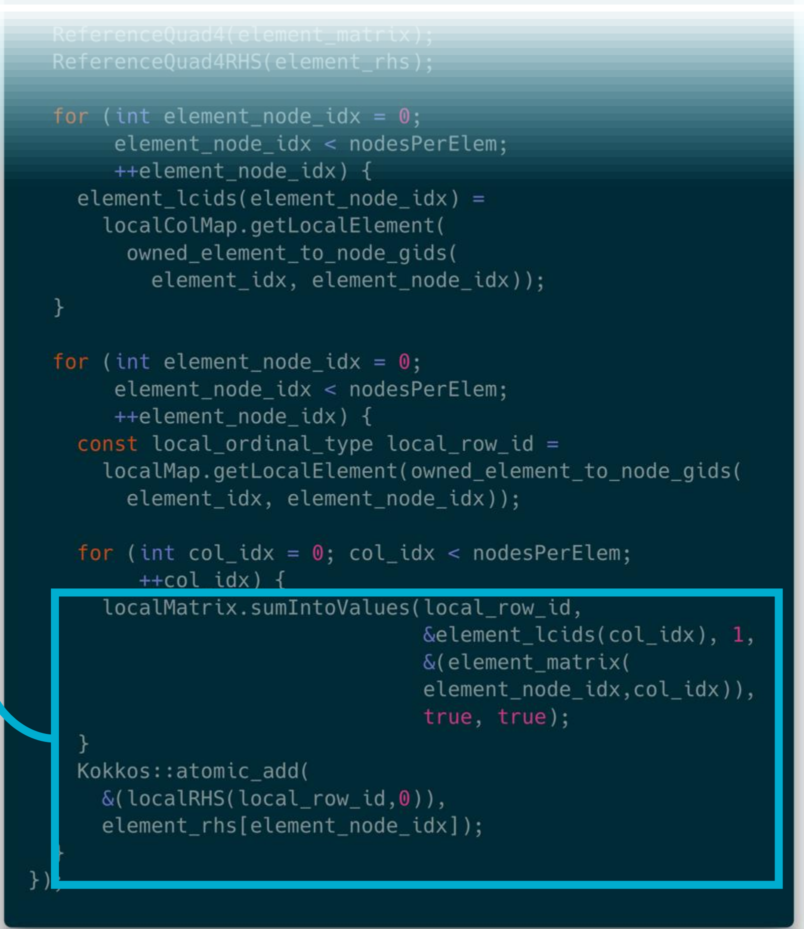


```
  for (size_t element_node_idx=0;
          element_node_idx < nodesPerElem;
          ++element_node_idx) {
    column_global_ids[element_node_idx] =
      owned_element_to_node_gids(
        element_gidx, element_node_idx);
  }

  for (size_t element_node_idx = 0;
        element_node_idx < 4;
        ++element_node_idx) {
    global_ordinal_type global_row_id =
      owned_element_to_node_gids(
        element_gidx, element_node_idx);

    for(size_t col_idx = 0; col_idx < 4;
        col_idx++) {
      column_scalar_values[col_idx] =
        element_matrix(element_node_idx, col_idx);
    }

    fe_matrix->sumIntoGlobalValues(
      global_row_id, column_global_ids,
      column_scalar_values);
    rhs->sumIntoGlobalValue(
      global_row_id, 0,
      element_rhs[element_node_idx]);
  }
}
```

Tpetra

Tpetra: contribute element values to FE matrix and RHS

Kokkos: atomic adds, since each thread (element) may contribute to the same node at the same time



```
  ReferenceQuad4(element_matrix);
  ReferenceQuad4RHS(element_rhs);

  for (int element_node_idx = 0;
       element_node_idx < nodesPerElem;
       ++element_node_idx) {
    element_lcids(element_node_idx) =
      localColMap.getLocalElement(
        owned_element_to_node_gids(
          element_idx, element_node_idx));
  }

  for (int element_node_idx = 0;
       element_node_idx < nodesPerElem;
       ++element_node_idx) {
    const local_ordinal_type local_row_id =
      localMap.getLocalElement(owned_element_to_node_gids(
        element_idx, element_node_idx));

    for (int col_idx = 0; col_idx < nodesPerElem;
         ++col_idx) {
      localMatrix.sumIntoValues(local_row_id,
                                 &element_lcids(col_idx), 1,
                                 &(element_matrix(
                                 element_node_idx,col_idx)),
                                 true, true);
    }
    Kokkos::atomic_add(
      &(localRHS(local_row_id,0)),
      element_rhs[element_node_idx]);
  }
});
```

Tpetra + Kokkos

WRAP UP: ADDING KOKKOS TO TPETRA

- Create parallel execution using `Kokkos::parallel_for`
    - Also supports host CPU execution
- `Kokkos::View` for data accessed in parallel regions
    - Convert `std::vector`, `Teuchos::Array`, `malloc`, `new`, …
- Functions called in that region must be `KOKKOS_FUNCTION`
    - e.g. producing the element matrix
    - …and any data it requires must be in a `Kokkos::View` (material properties, node coordinates, etc.)
- Use `Tpetra::[]::getDeviceView()` to get `Kokkos::View` of Tpetra data
    - As a consequence, have to operate with local rather than global indices
- Parallel regions may require atomics for their contributions

CONCLUSIONS

CONCLUSIONS

- Moving to Tpetra has a lot going for it…
  - GPU portability.
  - > 2.1B unknowns.
  - Better support for complex, AD types.
  - Tested on many architectures.
  - Faster SpMVs on CPU platforms.

- It will require more work (especially with graph assembly), but you gain in performance.

- Can be used with or without explicit use of Kokkos.
  - But if you want GPU support, you'll need to get familiar with Kokkos.