



Exceptional service in the national interest

# USING KOKKOS IN TPETRA FINITE ELEMENT ASSEMBLY

[Carl Pearson](#), Christopher Siefert

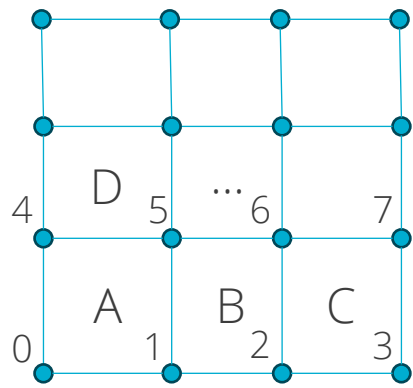
2:30 PM Wednesday November 1<sup>st</sup>, 2023,

Trilinos User Group Meeting

# TPETRA'S FINITE ELEMENT ASSEMBLY EXAMPLE

- Previously: [Link to TUG Kokkos Training Material]
  - Single code for all CPUs, GPUs, and whatever else Kokkos supports
- [Trilinos/packages/tpetra/core/example/Finite-Element-Assembly](https://trilinos.org/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
- No worksetting

# FE ASSEMBLY IN BRIEF



app. discretization

|    |    |    |    |
|----|----|----|----|
| 2  | -1 |    | -1 |
| -1 | 2  | -1 |    |
|    | -1 | 2  | -1 |
| -1 |    | -1 | 2  |

app. element matrix

varies per element,  
fixed in this example

A → {0, 1, 5, 4}

B → {1, 2, 6, 5}

C → {2, 3, 7, 6}

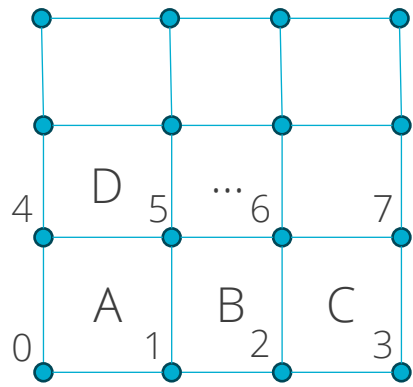
...

app. element-to-  
node map

|   | 0  | 1  | 2 | 3 | 4  | 5  | 6 | 7 | ... |
|---|----|----|---|---|----|----|---|---|-----|
| 0 | 2  | -1 |   |   | -1 |    |   |   |     |
| 1 | -1 | 2  |   |   |    | -1 |   |   |     |
| 2 |    |    |   |   |    |    |   |   |     |
| 3 |    |    |   |   |    |    |   |   |     |
| 4 | -1 |    |   |   | 2  | -1 |   |   |     |
| 5 |    | -1 |   |   | -1 | 2  |   |   |     |
| 6 |    |    |   |   |    |    |   |   |     |
| 7 |    |    |   |   |    |    |   |   |     |
| ⋮ |    |    |   |   |    |    |   |   |     |

Global FE matrix

# ELEMENT B'S CONTRIBUTION



*app. discretization*

|    |    |    |    |
|----|----|----|----|
| 2  | -1 |    | -1 |
| -1 | 2  | -1 |    |
|    | -1 | 2  | -1 |
| -1 |    | -1 | 2  |

*app. element matrix*

A → {0, 1, 5, 4}

B → {1, 2, 6, 5}

C → {2, 3, 7, 6}

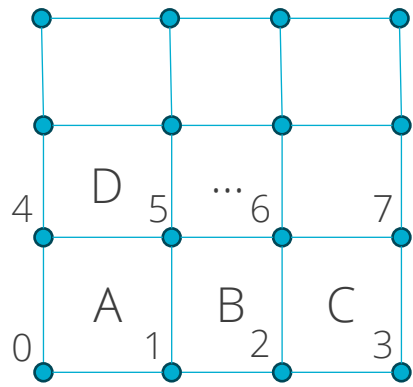
...

*app. element-to-node map*

|   | 0  | 1  | 2  | 3 | 4  | 5  | 6  | 7 |
|---|----|----|----|---|----|----|----|---|
| 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 |    |    |    |   |    |    |    |   |

*Global FE matrix*

# ELEMENT C'S CONTRIBUTION



*app. discretization*

|    |    |    |    |
|----|----|----|----|
| 2  | -1 |    | -1 |
| -1 | 2  | -1 |    |
|    | -1 | 2  | -1 |
| -1 |    | -1 | 2  |

*app. element matrix*

A → {0, 1, 5, 4}

B → {1, 2, 6, 5}

C → {2, 3, 7, 6}

...

*app. element-to-node map*

|   | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  |
|---|----|----|----|----|----|----|----|----|
| 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  |

*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()
  - Have 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(
                element_gidx, element_node_idx);
    }
}
```

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_lcid_unmanaged = makeUnmanaged(all_lcid);
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_lcid = Kokkos::subview(
        all_lcid_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(
        element_gidx, element_node_idx);
  }
}
```

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_id_array_type all_element_rhs(
  "all_element_rhs",nodesPerElem*numOwnedElements);
local_ordinal_single_view_type all_lcids(
  "all_lcids",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

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(...)** {  
 ...  
**};**

```
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(+);

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

```
Kokkos::RangePolicy<ExecutionSpace, int> policy(
    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.getOwnedElement(element_gidx).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

```

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);
    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

# CONCLUSION

- 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::[]::get*Device()` 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