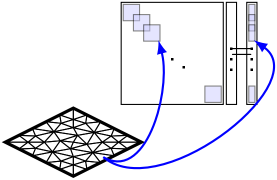
Using Intrepid2 for high-order FE assembly in a multiphysics code

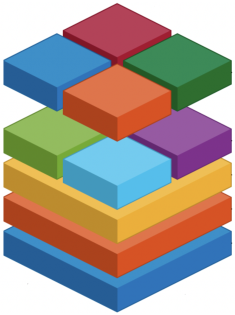
Arnst Maarten   Tomasetti Romin

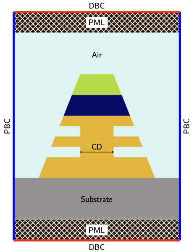
University of Liège, Belgium

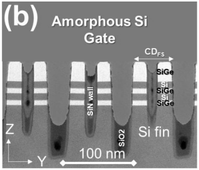
October 24th, 2024

Context









Development of *High-performance finite-Element Library for Multiphysics applications* (HELM).

Outline

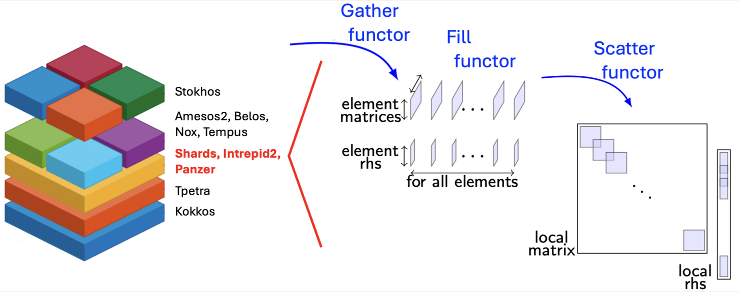
1. `HELM`
2. `Intrepid2`
3. Use of `Intrepid2` for high-order FE assembly in `HELM`
4. Thoughts about possible future developments
5. Conclusion

HELM

Starting points (2020–. . .)

- Trilinos/packages/tpetra/core/example/Finite-Element-Assembly/
- Trilinos/packages/trilinoscouplings/examples/fenl/

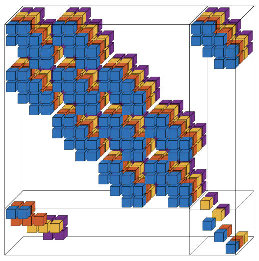
Our approach in HELM



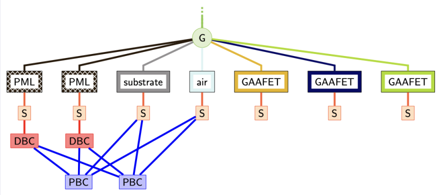
- Design FE code in terms of functors that expose parallelism.
- FE assembly using gather-fill-scatter pattern.

HELM

HELM as a computational laboratory



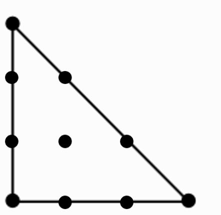
[Ph.D. thesis of K. Liegeois]

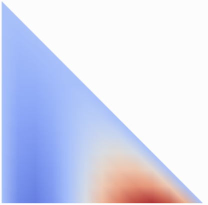


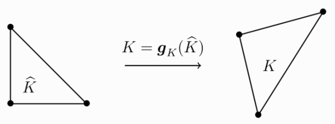
[Ph.D. thesis of R. Tomasetti]

- Asynchronous execution model using `Kokkos::Experimental::Graph`.
- Device-resident strategy (device version of `panzer::DOFManager`, ...).
- UQ scalar types (`Sacado::MP::Vector`).
- ...

Intrepid2







$$f_i^K = \int_K f(\boldsymbol{x})\varphi_i(\boldsymbol{x})d\boldsymbol{x}$$
$$\approx \sum_q f\left(\boldsymbol{g}_K(\hat{\boldsymbol{x}}_q)\right)\hat{\varphi}_i(\hat{\boldsymbol{x}}_q)\det \boldsymbol{J}_K(\hat{\boldsymbol{x}}_q)w_q$$

Shards

- Element-level numbering of topological entities (volume, face, edge, vertex) and their upward and downward adjacencies.

Intrepid2

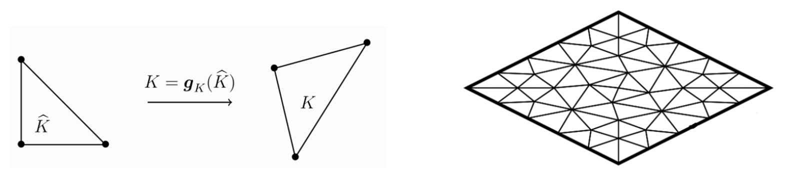
- Element-level geometry, geometric transformations, orientation, projection, quadrature rules, and basis functions.

Panzer::DOFManager

- Local and global numbering of degrees of freedom.

Intrepid2

Example of high-level API




```
//! Compute Jacobian matrix of geometrical mapping for all elems in a workset
template <typename DeviceType>
template <typename JacMatsViewType, typename NodeCoordsViewType, /* other params */>
static void Intrepid2::CellTools<DeviceType>::setJacobian(
    JacMatsViewType    stacked_jac_mats,
    NodeCoordsViewType stacked_node_coords,
    /* other params */
);
```

Intrepid2

Example of implementation by using functor

```
template <typename JacMatsViewType, typename NodeCoordsViewType, /* other params */>
struct F_setJacobian {
    JacMatsViewType    stacked_jac_mats;
    NodeCoordsViewType stacked_node_coords
    /* other members */

    KOKKOS_FUNCTION
    void operator()(const ordinal_t elm_id, const ordinal_t cub_point_id) const {
        /* fill jacobian matrix for elm_id and cub_point_id */
    }
};

template <typename DeviceType>
template <typename JacMatsViewType, typename NodeCoordsViewType, /* other params */>
static void Intrepid2::CellTools<DeviceType>::setJacobian(
    JacMatsViewType    stacked_jac_mats,
    NodeCoordsViewType stacked_node_coords,
    /* other params */
) {
    using execution_space = typename DeviceType::execution_space;
    Kokkos::parallel_for(
        Kokkos::MDRangePolicy<execution_space, /* other params */>(/* params */),
        F_setJacobian(stacked_jac_mats, stacked_node_coords, /* other params */)
    );
}
```

Intrepid2

Scalar-valued and vector-valued basis functions

```
//! Scalar-valued basis functions in H(grad).
using view_of_rank_two_t = Kokkos::View<double**, memory_space>;
view_of_rank_two_t evals_hgrad("evals_hgrad", num_basis_fnctns, num_cub_points);

//! Vector-valued basis functions in H(curl).
using view_of_rank_three_t = Kokkos::View<double***, memory_space>;
view_of_rank_three_t evals_hcurl("evals_hcurl", num_basis_fnctns, num_cub_points, dim);
```

Kokkos::DynRankView

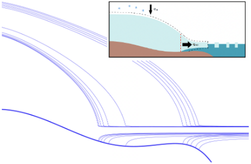
```
using view_of_dyn_rank_t = Kokkos::DynRankView<double, memory_space>;
view_of_dyn_rank_t evals_hgrad("evals_hgrad", num_basis_fnctns, num_cub_points);
view_of_dyn_rank_t evals_hcurl("evals_hcurl", num_basis_fnctns, num_cub_points, dim);
```

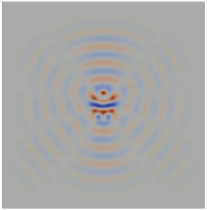
- The rank is not part of the type of `Kokkos::DynRankView`.
- *Largely* API compabible and inter-operable with `Kokkos::View`.

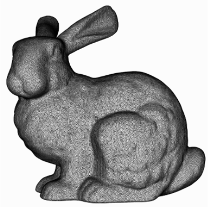
Use of Kokkos::DynRankView in Intrepid2

- Although `Intrepid2` templates classes and fnctns on view types, its implementation often relies on specificities of `Kokkos::DynRankView`.

Use of Intrepid2 for high-order FE assembly in HELM







- HELM can solve differential problems with
  - H(grad) basis functions: on Line2, Tria3, Quad4 and Tet4.
  - H(curl) basis functions: initial work on Tria3.

- HELM uses key functionalities provided by Intrepid2:
  - cubature points and weights.
  - evaluation of basis functions.
  - mapping from reference to physical domain.

Use of Intrepid2 for high-order FE assembly in HELM

```
const auto functor_jac = Intrepid2::FunctorCellTools::F_setJacobian(
    stacked_jac_mats, stacked_node_coords, /* other params */
);
const auto functor_det = Intrepid2::FunctorRealSpaceTools::F_det</* params */>(
    stacked_jac_dets, stacked_jac_mats
);
/* other functor instances */

Kokkos::parallel_for(
    Kokkos::MDRangePolicy<execution_space, /* other params */>(space, /* params */),
    KOKKOS_LAMBDA(const ordinal_t elm_id, const ordinal_t cub_point_id) {
        functor_jac(elm_id, cub_point_id);
        functor_det(elm_id, cub_point_id);
        /* other calls */
    }
);
```

- In HELM, we use Intrepid2's functor implementation: we compose our own functors and parallel regions from Intrepid2's functors.
- One of the advantages is that we can pass an execution space instance. In future work, we plan to use a graph execution model.

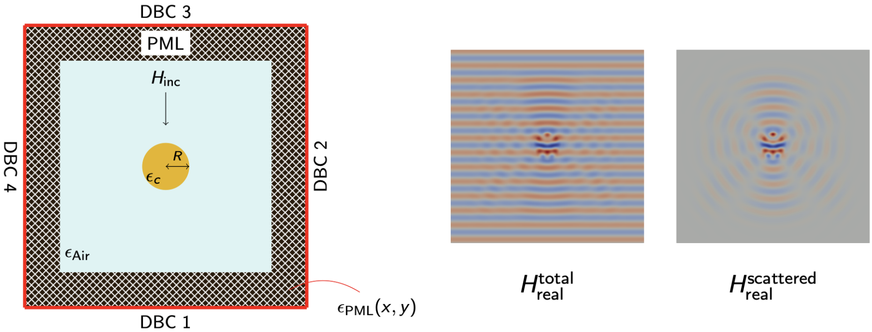
Use of Intrepid2 for high-order FE assembly in HELM

```
using view_of_rank_two_t = Kokkos::View<double**, memory_space>;
view_of_rank_two_t evals_hgrad("evals_hgrad", num_basis_fnctns, num_cub_points);

using view_of_dyn_rank_t = Kokkos::DynRankView<double, memory_space>;

intrepid2_basis_hgrad->getValues(
    exec,
    view_of_dyn_rank_t(evals_hgrad), // shallow copy
    view_of_dyn_rank_t(cub_points),  // shallow copy
    Intrepid2::OPERATOR_VALUE
);
```

- In `HELM`, we use `Kokkos::View` to store data. In any given application, we know which function spaces(s) are involved.
- We wrap a `Kokkos::View` into a `Kokkos::DynRankView` when passing it to `Intrepid2`, if needed.



Example application: electromagnetic wave scattering in 2D.

Thoughts about possible future developments

- Explicit template instantiation. Speed up compilation when including `Intrepid2` code in user code. Speed up `Intrepid2` test compilation.
- Passing execution space instances.
  - Initial work: https://github.com/trilinos/Trilinos/pull/12366.
- Interoperability with `Kokkos::View`.
  - Initial work: https://github.com/trilinos/Trilinos/pull/12842.
- Orientation and dof numbering in accordance with the *increasing vertex-index enumeration* convention.
- Use of `kokkos-kernels` and other optimized libraries for low-level operations, such as tensor contractions.

Conclusion

- We develop HELM (*High-performance finite-Element Library for Multiphysics applications*). One of its uses is as a computational laboratory to explore numerical methods and algorithms.

- In HELM, we use Shards, Intrepid2, and Panzer::DOFManager for high-order FE assembly.

- In HELM, we use Intrepid2's functor implementation. We compose our own functors and parallel regions from Intrepid2's functors.

- Suggestions for possible future developments include ETI; passing execution space instances; interoperability with Kokkos::View; and use of kokkos-kernels for low-level operations.