Intrepid2
a Performance-Portable Package for Compatible High-Order Finite Element Discretizations

SAND2018-2652C

K. Kim, M. Perego,
N. Ellingwood, K. Peterson, N. Roberts

CCR
Center for Computing Research

Sandia National Laboratories, NM, USA

SIAM Parallel Processing, Tokyo, March 8, 2018

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-

ECP
EXASCALE COMPUTING PROJECT

U.S. DEPARTMENT OF ENERGY

Sandia National Laboratories

Brief history of Intrepid2 package

Intrepid (*INteroperable Tools for Rapid dEveloPment of compatIble Discretizations)*

Trilinos package for **advanced** discretizations of Partial Differential Equations (PDEs)
Developers: **P. Bochev, H. C. Edwards, R.C. Curby, K. Peterson, D. Ridzal**

Provides compatible high-order discretizations for finite elements,
Supports hybrid discretizations: FEM, FV and FD

Intrepid2

Refactoring of Intrepid for performance portability, using Kokkos.
Early developers (stage 1): **I. Demeshko, A. Delora**

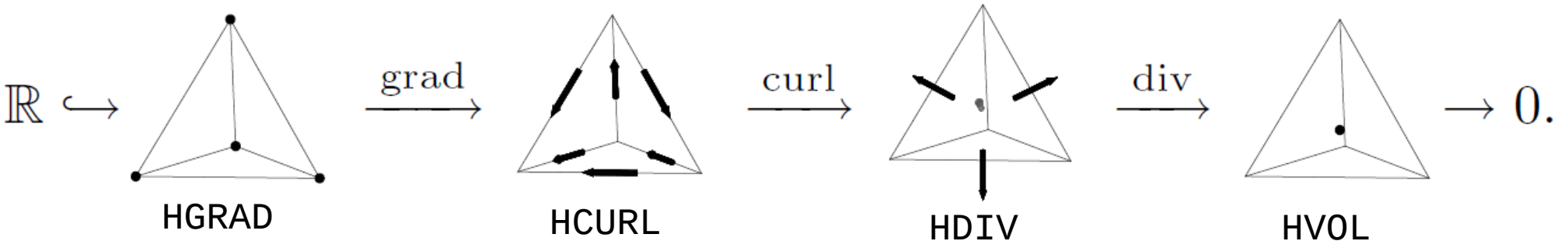
Current developers (stage 2)**: K. Kim, M. Perego, N. Ellingwood, K. Peterson, N. Roberts**

Stage 1: fully compatibility with Intrepid interface, adoption of Kokkos

Stage 2: no backward compatibility but same mathematical/logical interface.
Use of Kokkos dynamic rank views.

Tools for Local Assembly of Compatible finite elements

De Rham complex



$\Phi^*_G : H(\text{grad}, \widehat{\kappa}) \mapsto H(\text{grad}, \kappa)$ $\quad \Phi^*_G(\widehat{u}) = \widehat{u} \circ F_\kappa^{-1}$

$\Phi^*_C : H(\text{curl}, \widehat{\kappa}) \mapsto H(\text{curl}, \kappa)$ $\quad \Phi^*_C(\widehat{\mathbf{u}}) = ((DF_\kappa)^{-\mathsf{T}} \cdot \widehat{\mathbf{u}}) \circ F_\kappa^{-1}$

$\Phi^*_D : H(\text{div}, \widehat{\kappa}) \mapsto H(\text{div}, \kappa)$ $\quad \Phi^*_D(\widehat{\mathbf{u}}) = (J_\kappa^{-1} DF_\kappa \cdot \widehat{\mathbf{u}}) \circ F_\kappa^{-1}$

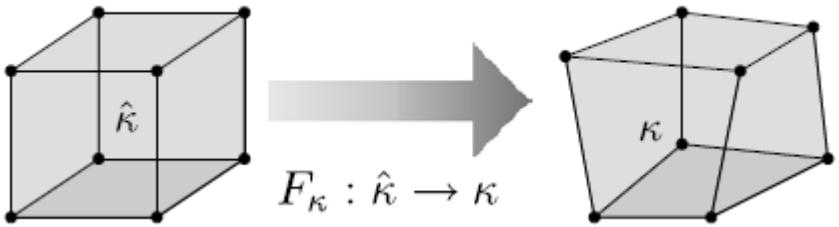
$\Phi^*_V : H(\text{vol}, \widehat{\kappa}) \mapsto H(\text{vol}, \kappa)$ $\quad \Phi^*_V(\widehat{u}) = (J_\kappa^{-1} \widehat{u}) \circ F_\kappa^{-1}$

{ HGRAD_transform_VALUE()

{ HGRAD_transform_GRAD(),
HCURL_transform_VALUE()

{ HCURL_transform_CURL(),
HDIV_transform_VALUE()

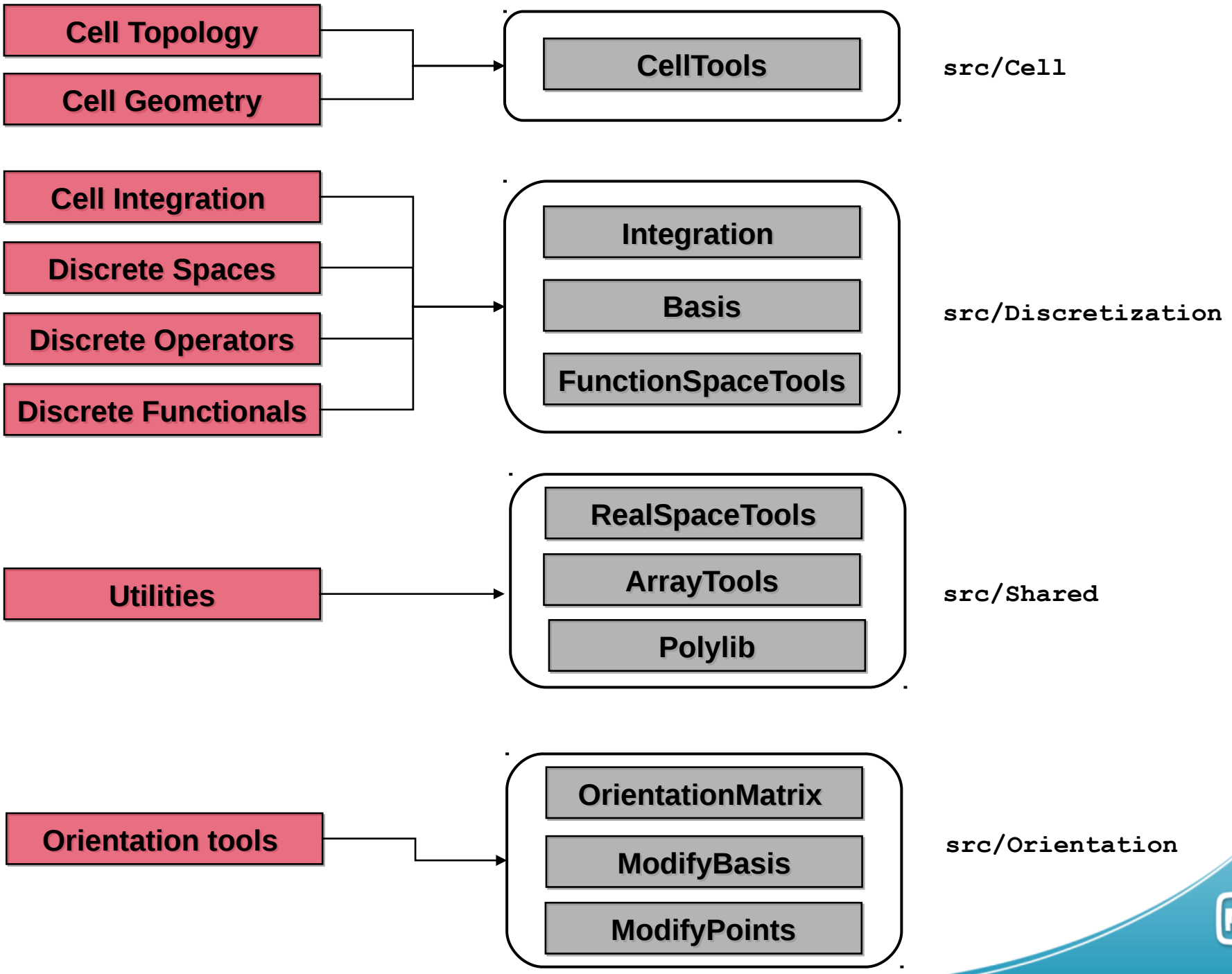
{ HDIV_transform_DIV(),
HVOL_transform_VALUE().



Courtesy: P. Bochev

Sandia National Laboratories

Package structure



Choice of multidimensional array

Intrepid2 work with multi-dimensional arrays: e.g.,

```
double basisVals[F][P];
double basisGradVals[F][P][D];
double jacMat[C][P][D][D];
```

```
// C - # of cells
// N - # of nodes per cell
// D – spacial dimension (2 or 3)
// F - # of field DOFs
// P - # of integration points
```

Operations often done on batches of cells

It is natural to declare these array as `Kokkos::view`

However, in some cases the rank of the arrays might not be known at compile time.
Solution: **`Kokkos::DynRankView`**

BASIS	INPUT	VALUE	GRAD	CURL	DIV
HGRAD	(P, D)	(F, P)	(F, P, D)		
HCURL	(P,D)	(F, P, D)		(F,P,D)	
HDIV	(P,D)	(F, P, D)			(F, P)

Rank of differential operators evaluated at quadrature points

Properties of Kokkos::DynRankView

```
// 1. Construction of a dynamic rank view
//     - rank is deduced from the number of dimensions arguments

Kokkos::DynRanView<value_type, exec_space> a(“a”, 3, 9);


// 2. Same subview syntax
auto part = Kokkos::subview(a, 0, Kokkos::ALL());  // part = a(1, :);


// 3 Inter-operability with Kokkos static view
Kokkos::View<value_type**, exec_space> b(“b”, 3, 9);

b = a; // okay
a = b; // error, a static view cannot be converted to DynRankView

Kokkos::deep_copy(b, a);  //okay
Kokkos::deep_copy(a, b);  //okay
```

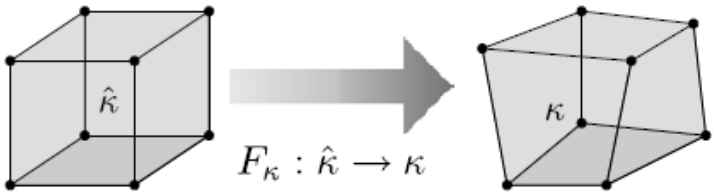
Advantages of using Kokkos::DynRankView

This functor computes the following three contractions
outputFields(cl, lf, rf) = leftFields(cl, lf, qp) * rightFields(cl, rf, qp);
outputFields(cl, lf, rf) = leftFields(cl, lf, qp, i) * rightFields(cl, rf, qp, i);
outputFields(cl, lf, rf) = leftFields(cl, lf, qp, i, j) * rightFields(cl, rf, qp, i, j);

Using a view, one would need to replicate the code for the scalar, vector and matrix cases.

```
KOKKOS_INLINE_FUNCTION
     void operator()(ordinal_type iter) const {
       ordinal_type cl, lf,  rf;
       unrolIndex(cl, lf, rf, iter, outputFields.dimesion(0), outputFields.dimesion(1), outputFields.dimesion(2));

       // member variables outputFields,  leftFields, rightFields

       auto result = Kokkos::subview( outputFields,  cl, lf, rf );

       // extra ranl ignored when computing subviews (only w/ DynRankView)
       const auto left  = Kokkos::subview( leftFields,  cl, lf, Kokkos::ALL(), Kokkos::ALL(), Kokkos::ALL() );
       const auto right = Kokkos::subview( rightFields, cl, rf, Kokkos::ALL(), Kokkos::ALL(), Kokkos::ALL() );

       constordinal_type npts = left.dimension(0);
       const ordinal_type iend = left.dimension(1);
       const ordinal_type jend = left.dimension(2);

       value_type tmp(0);
       for (ordinal_type qp = 0; qp < npts; ++qp)
         for (ordinal_type i = 0; i < iend; ++i)
           for (ordinal_type j = 0; j < jend; ++j)
             tmp += left(qp, i, j)*right(qp, i, j);
         result() = result() + tmp;
}
```

Assembly of element stiffness matrix

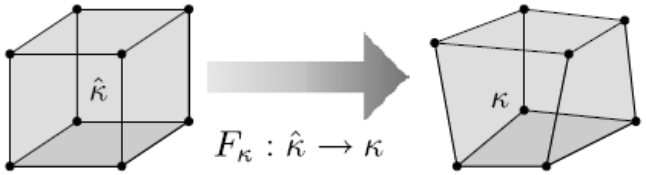


$$
\begin{aligned}
A_{ij} = \quad & \int_\kappa \nabla v_i \cdot \nabla v_j d\,\mathbf{x} \\
& \int_{\hat{\kappa}} \left((DF_\kappa)^{-T} \nabla \hat{\mathbf{v}}_i \right) \cdot \left((DF_\kappa)^{-T} \nabla \hat{\mathbf{v}}_j \right) J d\hat{\mathbf{x}} \\
& \sum_q \left((DF_\kappa)_q^{-T} \nabla \hat{\mathbf{v}}_{i,q} \right) \cdot \left((DF_\kappa)_q^{-T} \nabla \hat{\mathbf{v}}_{j,q} \right) J_q w_q
\end{aligned}
$$

Intrepid2 APIs

- Cubature::getCubature
- Basis:getValue
- CellTools: setJacobian, setJacobianDet
- FunctionSpaceTool::applyTransform, computeCellMeasure
- FunctionSpaceTools::integrate

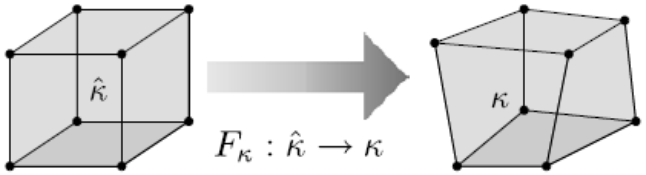
Assembly of element stiffness matrices



$$
\begin{aligned}
A_{ij} = & \int_\kappa \nabla v_i \cdot \nabla v_j d\mathbf{x} \\
& \int_{\hat{\kappa}} \left((DF_\kappa)^{-T} \nabla \hat{\mathbf{v}}_i\right) \cdot \left((DF_\kappa)^{-T} \nabla \hat{\mathbf{v}}_j\right) J d\hat{\mathbf{x}} \\
& \sum_q \left((DF_\kappa)_q^{-T} \nabla \hat{\mathbf{v}}_{i,q}\right) \cdot \left((DF_\kappa)_q^{-T} \nabla \hat{\mathbf{v}}_{j,q}\right) J_q w_q
\end{aligned}
$$

```
// Dimensions:
// C - # of cells
// N - # of nodes per cell
// D – spacial dimension (2 or 3)
// F - # of field DOFs
// P - # of integration points
Kokkos::DynRankView

// elements on reference coordinates
cubPoints(P, D),
cubWeights(P),
refGradVals(F,P, D),

//cell geometry
nodes(C, N, D),
jacMat(C, P, D, D),
jacInv(C, P, D, D),
jacDet(C, P),

//elements on physical coordinates
physGradVals(C, F, P, D),
elemStiff(C, F, F);
```

Assembly of element stiffness matrices



$$A_{ij} = \int_\kappa \nabla v_i \cdot \nabla v_j d\mathbf{x}$$

$$\int_{\hat{\kappa}} \left((DF_\kappa)^{-T} \nabla \hat{\mathbf{v}}_i\right) \cdot \left((DF_\kappa)^{-T} \nabla \hat{\mathbf{v}}_j\right) J d\hat{\mathbf{x}}$$

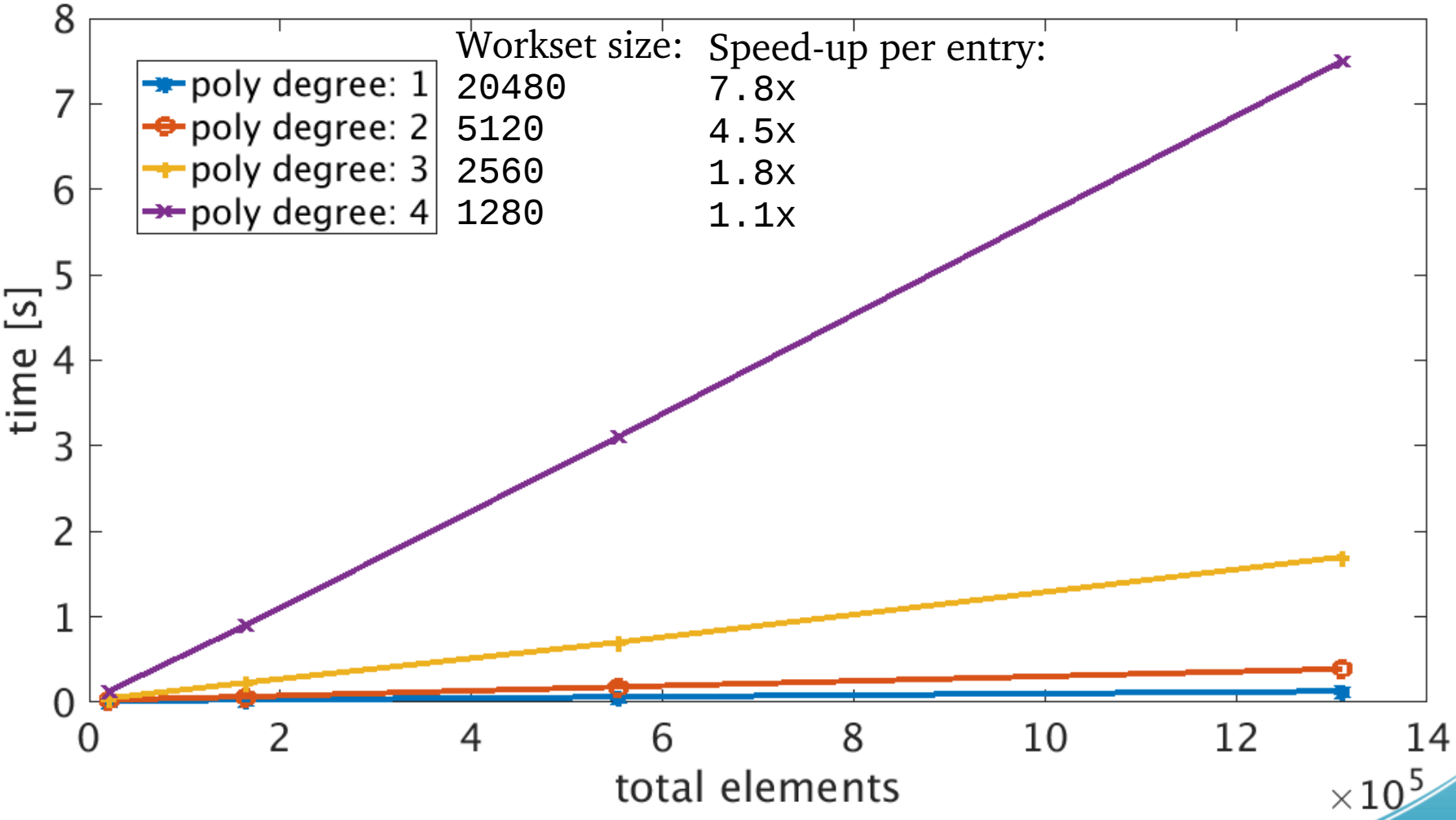
$$\sum_q \left((DF_\kappa)_q^{-T} \nabla \hat{\mathbf{v}}_{i,q}\right) \cdot \left((DF_\kappa)_q^{-T} \nabla \hat{\mathbf{v}}_{j,q}\right) J_q w_q$$

```
// nodal basis for Tets
Basis_HGRAD_TET_FEM<DeviceSpace,double,double> basis(degree);

// compute reference element
cubTet.getCubature(cubPoints, cubWeights);
basis.getValues(refGradVals, cubPoints, OPERATOR_GRAD);

// compute cell transformation matrices
CellTools::setJacobian(jacobian, cubPoints, nodes, basis);
CellTools::setJacobianInv(jacInv, jacobian);
CellTools::setJacobiaDet(jacDet, jacobian);

//Transform basis to physical coordinates
using FunctionalSpaceTool FCT;

FCT::computeCellMeasure(weightedMeasure,
                        jacDet, cubWeights);
FCT::HGRADtransforHGRAD(phyGradVals,jacInv, refGradVals);
Fct::multiplyMeasure(phyGradValsWeighted,
                     weigthedMeasure, phyGradVals)
FCT::integrate(elemStiff, phyGradVals, phyGradValsWeighted);
```

Sandia National Laboratories

Assembly of element stiffness matrices

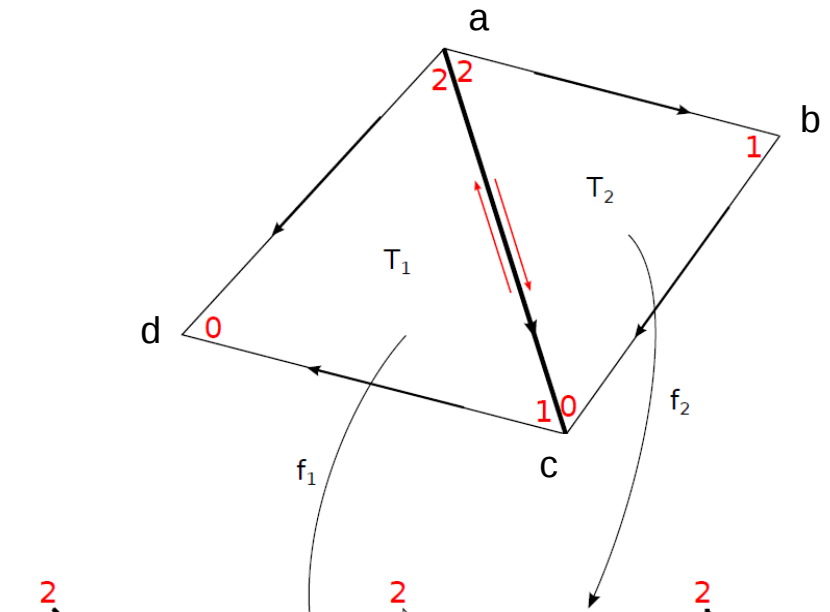
NVIDIA Tesla P100
Chip: 56 SMs, 32 FP64 CUDA Cores/SM
Peak performance: 4.7 TFLOPs in double precision, 732+ GB/s (HBM)



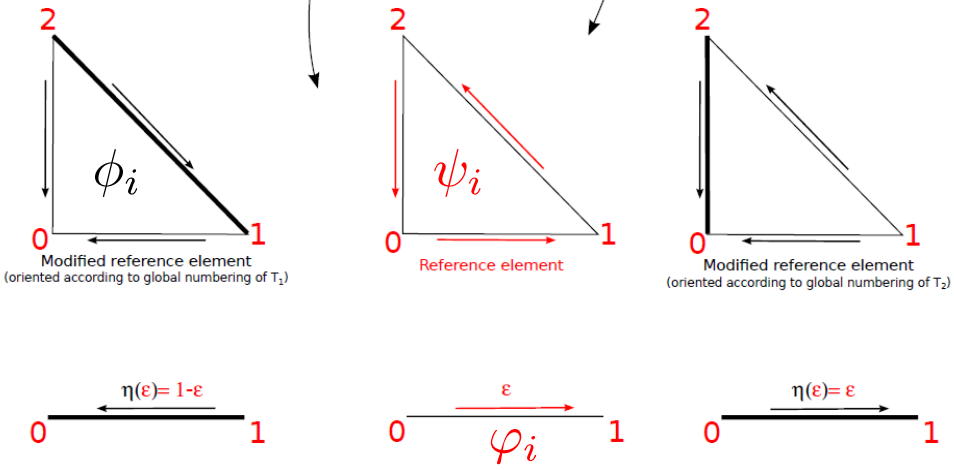
Courtesy: N. Roberts

Orientation Tools
(HCurl case, 2d)

In high-order finite elements
DOFs of adjacient elements may not be ordere
the same way on the shared face.
We need to enforce the matching of DOFs.



Solution: locally map the DOFs of a face or ed
according to the global numbering and transf
the corresponding basis accordingly

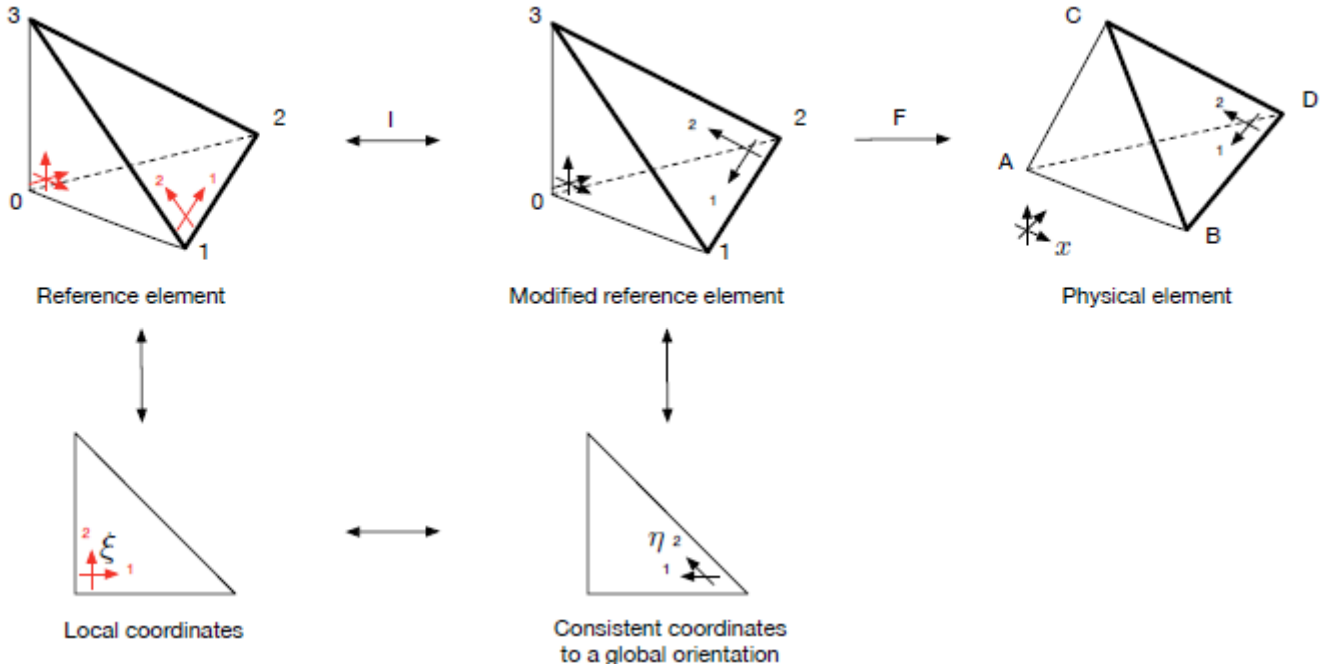


$$\phi_i^E = \sum_j A_{ij}^E \psi_j$$

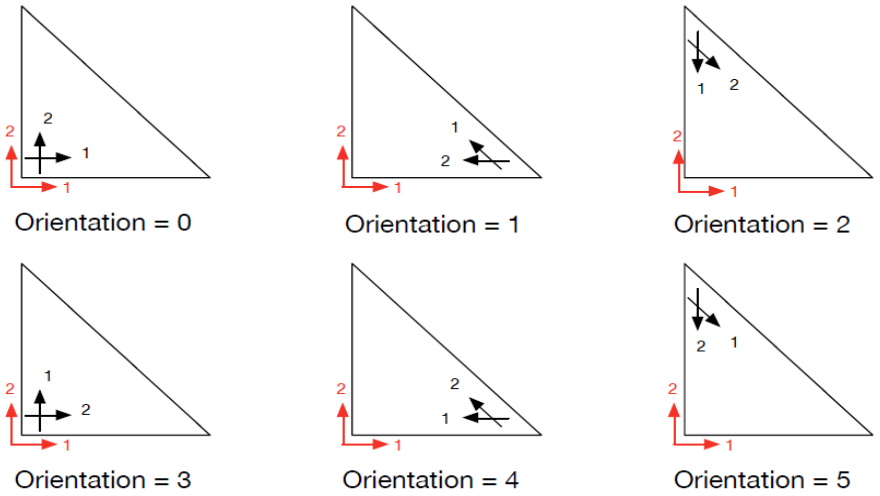
$$\sum_j A_{ij}^E \psi_j(E(\eta(\xi_k)) \cdot \mathbf{t}_E = \varphi_i(\xi_k) \cdot \hat{\mathbf{t}}$$

Sandia National Laboratories

Orientation Tools
(Hcurl case, 3d)



Possible orientations for triangles



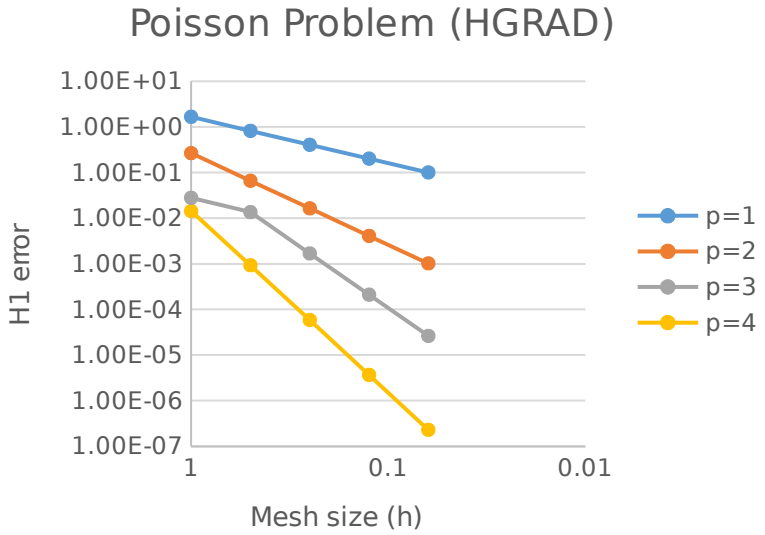
$$\sum_j A^T_{ij} \psi^T_j \left(T(\eta(\xi_k))\right) \cdot \mathbf{t}_{T,l} = \varphi_i(\xi_k) \cdot \hat{\mathbf{t}}_l$$

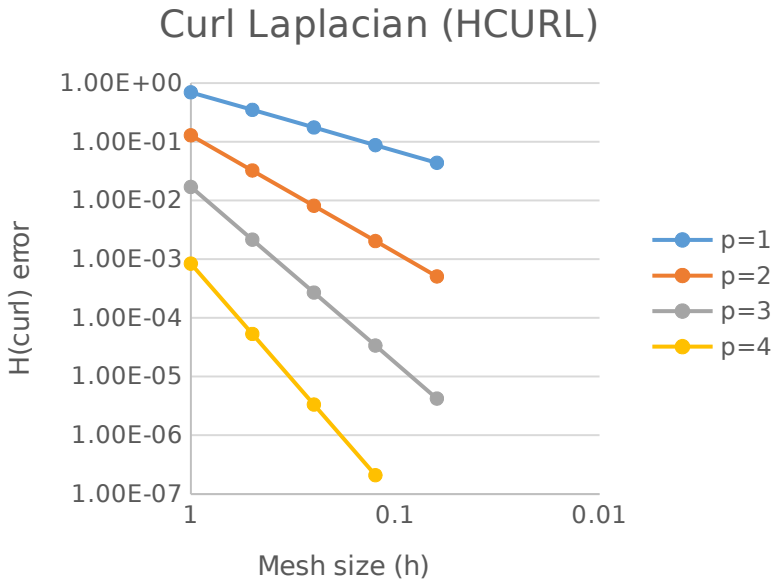
We store six matrices A,
one for each orientation.

Sandia National Laboratories

Orientation Tools
demonstration for HGRAD and HCURL problems

Problems implemented using Trilinos package Panzer.





Results show theoretical order of convergence.

Courtesy: K. Kim, E. Cyr

Sandia National Laboratories

Conclusion

- Portable implementation for advanced (finite element) discretizations
- Currently used by several applications including Albany (J. Watkins talk), Drekar(MFH), Camellia (DG) and in the Trilinos package Panzer.
- Actively developed and currently assessing performance on CPUs/GPUs:
  - recompute vs storing arrays, avoid subviews, hierarchical parallelism?
- Ongoing/future work:
  - optimization for PIC codes (evaluation at few points vs standard batch eval at quadrature points)
  - handling of hybrid meshes
  - projection tools for high-order finite elements

Thanks!