Software-Facilitated Performance Improvements
Performance Portability, Embedded Ensembles, and Implicit Methods
Andy Salinger, Eric Phipps, Irina Demeshko, Jeff Fike [SNL]
and the Albany, Trilinos, and Kokkos development teams

Introduction: C++ Templates
C++ Templates are a powerful tool in the computational scientist’s toolbox. Their use can greatly simplify the development and maintenance of advanced features.
- Source code takes on different behaviors based on a values of a template parameter.
  - E.g., a single Sort() function can be written for any data type that supports “<” and “==” operators.
- Templates allows for a greatly reduced source code base to maintain.
- Substitution occurs at compile time, so there is no run-time overhead.
Templates provide an additional interface to your code that can be exploited in very creative and impactful ways. Here, we show 3 examples of how the use of C++ Templates in the Albany code greatly enhances capabilities, with limited code development and maintenance costs.

Example of Templated Code
```cpp
typedef Kokkos::OpenMP ExecutionSpace;
//typedef Kokkos::cCUDA ExecutionSpace;
//typedef Kokkos::Serial ExecutionSpace;
template<typename ScalarT>
vecGrad<scalarT>::vecGrad()
Kokkos::View<scalarT**, ExecutionSpace> vecGrad(numCells, numQP, numVec, numDim);
}

**************
template<typename ScalarT>
void vecGrad<scalarT>::evaluateFields()
{
Kokkos::parallel_for<ExecutionSpace> (numCells, *this);
}

**************
template<typename ScalarT>
KOKKOS_INLINE_FUNCTION
void vecGrad<scalarT>::operator() (const int cell) const
{
for (int qp = 0; qp < numQP; qp++) {
  for (int dim = 0; dim < numDim; dim++) {
    for (int nd = 0; nd < numNode; nd++) {
      vecGrad(cell, qp, dim, i) += val(cell, nd, dim) * basisGrad(nd, qp, i);
    }
  }
}
```

Embedded Ensembles
Embedded ensembles move them to an inner loop instead of an outer loop, executing them concurrently.
- The Sacado library in Trilinos has an Ensemble type
  - The Ensemble type combines an array of values (\( \{ y \} \))
  - Operations (*, +, exp, sqrt, cos) are implemented as operations on an array of data
- ScalarT is instantiated with Sacado::Ensemble
- Large performance gains can be realized by:
  - Amortizing costs for mesh-dependent calculations
  - Easy compiler vectorization of kernels over ensembles
  - Amortizing latency over larger MPI messages
  - Contiguous memory access for arrays of data

Performance Portability using Kokkos
The Kokkos programming model enables performance portability of kernels.
- Kokkos uses the ExecutionSpace parameter to tailor code for a device:
  1. Memory layout for the MultiDimVector.
     (Accessor syntax \( v(i,j,k) \) is unchanged.)
  2. Parallel kernel launch directives under the Kokkos::Parallel_for() call.
- New architectures handled by new ExecutionSpace parameters being implemented in Kokkos.
- Leverage of ongoing DOE investments
- Kokkos implementation requires separating thread-safe kernel into separate function that is launched with integer loop index (e.g., \( cell \)).

Automatic Differentiation for Implicit Solves
Automatic Differentiation (AD) is implemented in the Albany/FELIX ice sheet velocity solver under MPAS-LI.
- The Sacado library from Trilinos has a FAD datatype
  - FAD type combines a value and derivative array (\( v(i,j,k) \))
  - Operations (*, +, exp, sort, cos) are overloaded in FAD to propagate its derivative via the chain rule.
- ScalarT in code is instantiated with Sacado::FAD.
- An analytic Jacobian matrix, sensitivities, and gradients with AD have enabled robust solves and adjoint-based inversion in Albany/FELIX under the PISEES Scidac.
- Data types nest cleanly: Ensemble<FAD<double> >