F: 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.

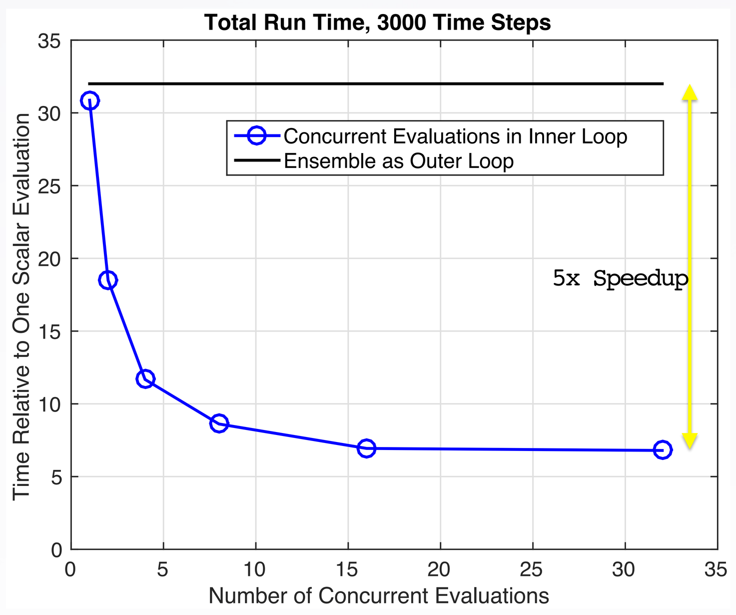
- Source code takes on different behaviors based on a values of a template parameter.
  - *E.g.*, a single Sort() function can be written for any data type that supports "<" and "=" operators.
- Templates allows for a greatly reduced source code base to maintain.
- Substitution occurs at compile time, 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.

Embedded Ensembles

Embedding 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 `[v]`
  - 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



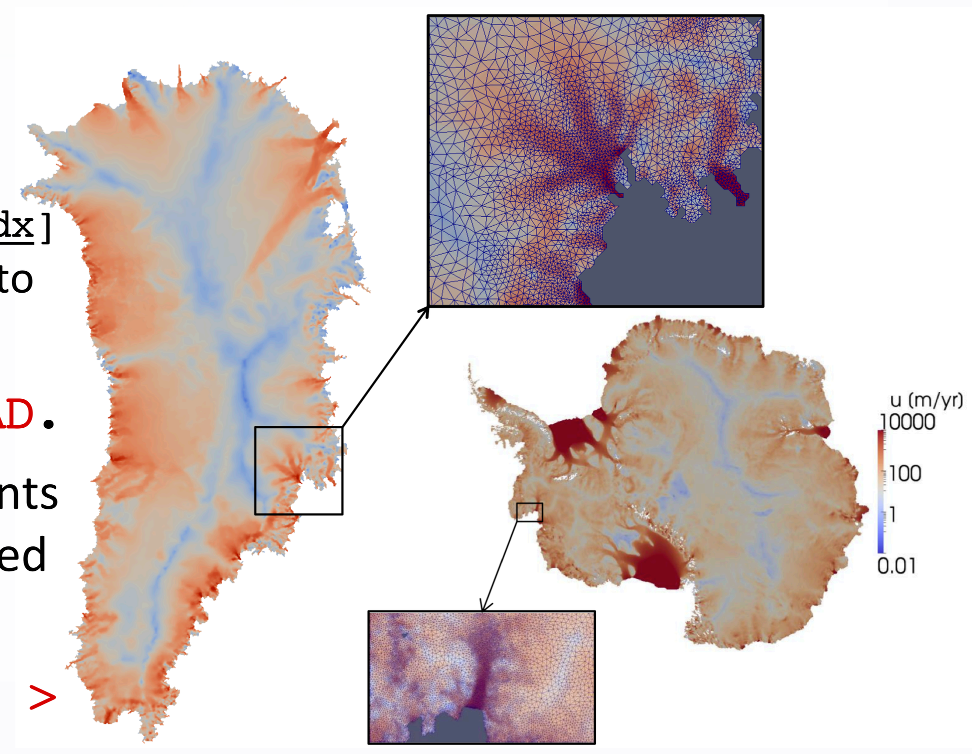
Relative timings for performing an ensemble of 32 calculations, comparing outer-loop and embedded inner-loop approaches. The array size for the vector of concurrent calculations is varied for the embedded case from 1 to 32. The application is the integration of the shallow water equations on a sphere in the Albany/Aeras code, where the height of a mountain is varied as an uncertain parameter.

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 `[val|dx]`
  - Operations (*, +, exp, sqrt, 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 tge PISCEES SciDAC.

➢ Data types nest cleanly: `Ensemble<FAD<double> >`



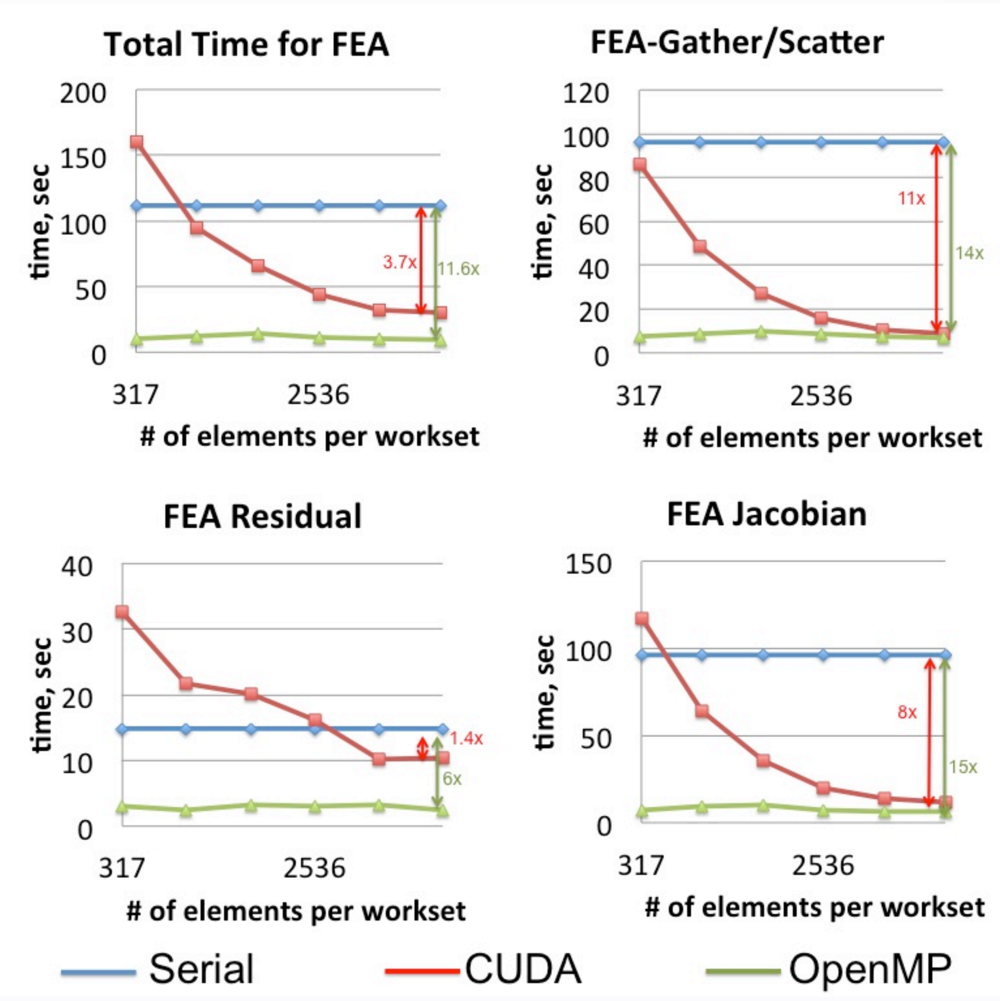
Example of Templated Code

```
typedef Kokkos::OpenMP ExecutionSpace;
//typedef Kokkos::CUDA ExecutionSpace;
//typedef Kokkos::Serial ExecutionSpace;
template<typename ScalarT>
vectorGrad<ScalarT>::vectorGrad()
{
Kokkos::View<ScalarT****, ExecutionSpace> vecGrad(numCells, numQP, numVec, numDim);
}
*********************************************
template<typename ScalarT>
void vectorGrad<ScalarT>::evaluateFields()
{
  Kokkos::parallel_for<ExecutionSpace> (numCells, *this);
}
*********************************************
template<typename ScalarT>
KOKKOS_INLINE_FUNCTION
void vectorGrad<ScalarT>:: operator() (const int cell) const
{
  for (int cell = 0; cell < numCells; cell++)
  for (int qp = 0; qp < numQP; qp++) {
    for (int dim = 0; dim < numVec; dim++) {
      for (int i = 0; i < numDim; i++) {
        for (int nd = 0; nd < numNode; nd++) {
          vecGrad(cell, qp, dim, i) += val(cell, nd, dim) * basisGrad(nd, qp, i);
} } } } }
```

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`).



Performance of the finite element assembly (FEA) in the Albany/FELIX application for ice sheet velocity solve. The same code base was used in all cases, but with different template parameters for the Kokkos execution space.

ACME Accelerated Climate Modeling for Energy

For additional information, contact:

Andy Salinger
Principal Member of Technical Staff

Sandia National Labs
(505) 845-3523
agsalin@sandia.gov

climatemodeling.science.energy.gov/acme

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

U.S. DEPARTMENT OF ENERGY