Albany: Unstructured Finite Element Codebase

April 24th, 2019

PRESENTED BY
Jerry Watkins and Irina Tezaur

1st Kokkos User Group Meeting
Albuquerque, New Mexico
Albany – finite element codebase in C++

Albany is built primarily for Rapid Application Development (RAD) from Trilinos Agile Components.

Component Examples (*package name*)
- Discretization Tools (*Intrepid2*)
- Nonlinear solver (*NOX*)
- Preconditioners (*Ifpack2*)
- Linear solver (*Belos*)
- Field DAG (*Phalanx*)
- Automatic Differentiation (*Sacado*)
- Distributed Memory Linear Algebra (*Tpetra*)
- Shared memory parallelism (*Kokkos*)
- Many more...

ProSPect – project under SciDAC

ProSPect = Probabilistic Sea Level Projections from Ice Sheet and Earth System Models
5 year SciDAC4 project (2017-2022).

Role: to develop and support a robust and scalable land ice solver based on the First-Order (FO) Stokes equations → Albany Land Ice

Requirements for Albany Land Ice (formerly FELIX):

- **First-order Stokes model**
- **Unstructured** meshes
- **Scalable, fast** and **robust**
- **Verified** and **validated**
- **Portable** to new architecture machines
- **Advanced analysis** capabilities: deterministic inversion, model calibration, uncertainty quantification, sensitivity analysis

As part of **DOE E3SM Earth System Model**, solver will provide actionable predictions of 21st century sea-level change (including uncertainty bounds).

https://doe-prospect.github.io/
Albany Land Ice performance is split between the **linear solve** (50%) and **FEA** (50%)

Aeras (explicit method) ~99% FEA

- **Piro** manages the nonlinear solve
- **Tpetra** manages **distributed** memory linear algebra (**MPI+X**)
- **Phalanx** manages **shared** memory computations (**X**)
  - **Gather** fills element local solution
  - **Interpolate** solution/gradient to quad. Points
  - **Evaluate** residual/Jacobian
  - **Scatter** fills global residual/Jacobian

- First step towards performance portability is the **FEA**

https://github.com/SNLComputation/Albany
Phalanx – directed acyclic graph (DAG)-based assembly

**Advantages:**
- Increased flexibility, extensibility, usability
- Arbitrary data type support
- Potential for task parallelism

**Disadvantage:**
- Performance loss through fragmentation

**Extension:**
- Performance gain through memoization

---

**DAG Example**

- Scatter
- Residual
- Interpolate Parameter
- Gather Parameter
- Interpolate Solution
- Basis Functions
- Gather Solution
- Gather Coordinates

**DAG Example (memoization)**

- Scatter
- Residual
- Interpolate Solution
- Stored Field
- Gather Solution

---

**Bar Chart**

- Single CPU socket or GPU
- Improvements:
  - \( a \): Base
  - \( b \): Memoization

- Haswell 16MPI: 2.0x
- Haswell 16(MPI+2OMP) 68(MPI+4OMP): 2.1x
- P100 1(MPI+GPU): 3.9x
A Phalanx node (evaluator) is constructed as a C++ class

- Each evaluator is templated on an evaluation type (e.g. residual, Jacobian)
- The evaluation type is used to determine the data type (e.g. double, Sacado data types)
- Kokkos RangePolicy is used to parallelize over cells over an ExeSpace (e.g. Serial, OpenMP, CUDA)
- Inline functors are used as kernels
- MDField data layouts
  - Serial/OpenMP – LayoutRight (row-major)
  - CUDA – LayoutLeft (col-major)

```cpp
template<typename EvalT, typename Traits>
void StokesFOResid<EvalT, Traits>::
evaluateFields(typename Traits::EvalData workset) {
  Kokkos::parallel_for(
    Kokkos::RangePolicy<ExeSpace>(0, workset.numCells),
    *this);
}

template<typename EvalT, typename Traits>
KOKKOS_INLINE_FUNCTION
void StokesFOResid<EvalT, Traits>::
operator() (const int& cell) const{
  for (int node=0; node<numNodes; ++node) {
    Residual(cell, node, 0) = 0.;
  }
  for (int node=0; node < numNodes; ++node) {
    for (int qp=0; qp < numQPs; ++qp) {
      Residual(cell, node, 0) +=
        Ugrad(cell, qp, 0, 0)*wGradBF(cell, node, qp, 0) +
        Ugrad(cell, qp, 0, 1)*wGradBF(cell, node, qp, 1) +
        force(cell, qp, 0)*wBF(cell, node, qp);
    }
  }
}
```
Sacado – Automatic Differentiation (AD)

Sacado data types are used for derivative components (ND = number of components)

- **DFad** (most flexible) – ND is set at run-time
- **SLFad** (flexible/efficient) – maximum ND set at compile-time
- **SFad** (most efficient) – ND set at compile-time

**ND Size Example:** Tetrahedral elements (4 nodes), 2 equations, ND = 4*2 = 8

*Fad Type Comparison for StokesFO<Jacobian> (Serial, OpenMP (12 threads), CUDA)*
WIP: Hierarchical Parallelism

Hierarchical parallelism is used to expose more parallelism when strong scaling

- Kokkos TeamPolicy, TeamThreadRange is used to parallelize over cells and nodes
- Kokkos scratch space is used to store node/quadrature values in shared memory
- ~2x speedup for small problem sizes on GPU (need padding for large problem sizes)
- Slowdown for all problem sizes on CPU (need different layout)
Performance Study – Greenland Ice Sheet (GIS)

- Unstructured **tetrahedral** element meshes
- **Wall-clock time** averaged over 100 global assembly evaluations (residual + Jacobian)
- Performance analysis focuses on **finite element assembly**
- **Notation** for performance results:

  \[ r(MPI + jX), \ X \in \{\text{OMP, GPU}\} \]

  \[ r = \# \text{MPI ranks} \]

  \[ j = \# \text{OpenMP threads or GPUs/rank} \]

  \[ X = \text{architecture for shared memory parallelism} \]
Performance Study – Architectures

Architectures:

• Cori (NERSC): 2,388 Haswell nodes [2 Haswell (32 cores)]
  9,688 KNL nodes [1 Xeon Phi KNL (68 cores)] (Cray Aries)

• Blake (SNL): 40 nodes [2 Skylake (48 cores)] (Intel OmniPath Gen-1)

• Mayer (SNL): 43 nodes [2 ARM64 Cavium ThunderX2 (56 cores)] (Mx EDR IB)

• Ride (SNL): 12 nodes [2 POWER8 (16 cores) + P100 (4 GPUs)] (Mx C-X4 IB)

• Waterman (SNL): 10 nodes [2 POWER9 (40 cores) + V100 (4 GPUs)] (Mx EDR IB)

Compilers: gcc/icpc (xlC, armclang++ WIP)

Models:

• 3 models: MPI-only, MPI+OpenMP, MPI+CUDA

• MPI+OpenMP: MPI ranks are mapped to cores, OpenMP threads are mapped to hardware-threads

• MPI+GPU: MPI ranks assigned a single core per GPU
  • CUDA UVM used for host to device communication
Performance Results – Node Utilization

**Node:** Single dual-socket CPU or quad-GPU

**Speedup achieved across most execution spaces**

- Kokkos Serial vs. OpenMP or CUDA (Doesn’t include refactoring improvements)
- **12.6x** speedup on POWER8+P100, **2.0x** speedup on POWER9+V100
- Very little improvement on Skylake

Tpetra Export poor on V100 (WIP w/ Tpetra and CUDA9 GPUDirect issue on POWER systems)

**Blue** (SMAssembly): shared memory local/global assembly (assembly/computation)

**Yellow** (DMAssembly): distributed memory global assembly handled by Tpetra (mostly communication)
Performance Results – Strong Scalability

Legend: HSW, SKX=Haswell, Skylake CPU; KNL=Xeon Phi; TX2=ThunderX2; P100,V100=GPU

Reasonable scaling across all devices **without** machine-specific optimization in Albany

- Poor GPU scaling (Export WIP w/ Tpetra and CUDA9 GPUDirect issue)
- Best case: Skylake at 32 devices (768 cores)
Single CPU/GPU shared memory profile

- Residual/Jacobian **Evaluation** most expensive
- **Gather/Scatter** becoming increasingly important...
- **Other**: some auxiliary routines are still expensive on the GPU (~10%)
Single GPU – Full profile

- Large portion is Kokkos routines
- Non-Kokkos is mostly boundary conditions (WIP)
**Single GPU – Kokkos and non-Kokkos**

- **Gather/Scatter**: Minimize by combining w/ Tpetra routines?
- **Interpolation**: Utilize Intrepid2/KokkosKernels (batch gemv, small “A” matrix)? Need Sacado?
- **Evaluation**: Nonlinear function within a gemm (Two types: double/Sacado)
Summary

- Progress towards **performance portability** across a variety of HPC architectures using a **single code base** by utilizing **Trilinos/Kokkos**
  - Multicore and manycore processors (Haswell, Skylake, KNL, TX2)
  - NVIDIA GPUs (P100, V100)

- We will be able to utilize next generation HPC architectures for **probabilistic sea-level predictions** using Albany Land Ice
  - Targets: Cori (Haswell, KNL), Summit (POWER9+V100), Aurora

- **Performance** can be improved on all architectures

- The open-source Albany multiphysics finite element code is available here:
  - [https://github.com/SNLComputation/Albany](https://github.com/SNLComputation/Albany)
Future Work

- Improve performance portability of **boundary conditions**
- More detailed **profiling**: vtune, nvprof, other tools?
- Code **optimizations** for finite element assembly:
  - More work on **hierarchical parallelism** (Intrepid2, KokkosKernels)
  - More **vectorization** on CPUs (docs on simd?)
  - Better **node utilization** (multiple CUDA instances on GPUs worth it?)
  - **Explicit data management** to minimize memory transfers (still worth it?)

- **Performance portability** for **solvers** is an ongoing research topic within **Trilinos**
  - Test next generation preconditioners (Multithreaded Gauss-Seidel, FastILU)
Support for this work was provided by Scientific Discovery through Advanced Computing (SciDAC) projects funded by the U.S. Department of Energy, Office of Science (OS), Advanced Scientific Computing Research (ASCR) and Biological and Environmental Research (BER).

Computing resources provided by the National Energy Research Scientific Computing Center (NERSC) and Oak Ridge Leadership Computing Facility (OLCF).
Motivation

• Earth-system models (ESMs) need more computational power to achieve higher resolutions.

• High performance computing (HPC) architectures are becoming increasingly more heterogeneous in a move towards exascale.

• Climate simulation software must adapt to continuously changing HPC architectures with different models for shared memory parallelism.
Performance Portability – a response to heterogeneity

**Generic Definition:** For an application, a reasonable level of performance is achieved across a wide variety of computing architectures with the same source code.

Let’s be more specific:

- **Performance** quantified by application execution time while strong/weak scaling.
- **Portability** includes conventional CPU, Intel KNL, NVIDIA GPU.

**Approach:** MPI+X Programming Model

- MPI: distributed memory parallelism – Tpetra
- X: shared memory parallelism – Kokkos
  - Examples: OpenMP, CUDA
- Minimize data movement (efficient programming)
- Increase arithmetic intensity (improve compute to memory transfer ratio)
- Saturate memory bandwidth (expose more parallelism)
Kokkos – Performance Portability

- **Kokkos** is a C++ library that provides **performance portability** across multiple **shared memory** computing architectures
  - Examples: Multicore CPU, NVIDIA GPU, Intel KNL and much more...

- Abstract **data layouts** and **hardware features** for optimal performance on **current** and **future** architectures

- Allows researchers to focus on **application development** instead of **architecture specific programming**

With Kokkos, you write an algorithm once for multiple hardware architectures. Template parameters are used to obtain hardware specific features.

https://github.com/kokkos/kokkos/
Performance Results – Weak Scalability

Legend: HSW, SKX=Haswell, Skylake CPU; KNL=Xeon Phi; TX2=ThunderX2; P100,V100=GPU

Reasonable scaling across all devices without machine-specific optimization in Albany

- Poor GPU scaling (Export WIP within Tpetra)
- Best case: Skylake at 10 devices (280 cores)