Automatic Differentiation of C++ Codes on Emerging Manycore Architectures with Sacado

Eric Phipps (etphipp@sandia.gov), Roger Pawlowski, and Christian Trott, Sandia National Laboratories
Albuquerque New Mexico USA

23rd Euro AD Workshop

Aug. 11-13, 2020
Sacado: AD Tools for C++ Apps

- Package in Trilinos
  - [https://github.com/trilinos](https://github.com/trilinos)
  - Open source license

- Forward mode AD
  - Based on Fad<> library of Di Césaré, Aubert and Pironneau
  - Tries to eliminates OO overhead via expression templates
  - DFad<double>: Derivative array determined at run-time
  - SFad<double,N>: Derivative array length = N
  - SLFad<double,N>: Derivative array length at most N

- Reverse mode AD
  - David Gay’s Rad library

- AD applied through template-based generic programming
  - Template on scalar type
  - Instantiate on AD data types

- Manually exploit simulation structure/sparsity
  - AD applied at “element” level
  - Template “physics”
  - Manually incorporate derivatives into global linear algebra objects

Iso-velocity adjoint surface for fluid flow in a 3D steady MHD generator in Drekar computed via Sacado (Courtesy of T. Wildey)
Kokkos
performance portability for C++ applications

Applications & Libraries

LAMMPS
Albany
Drekar

EMPIRE
SPARC
SIERRA

Trilinos
etc...

Multi-Core
Many-Core
APU
CPU+GPU

template <typename ViewTypeA, typename ViewTypeB, typename ViewTypeC>
void run_mat_vec(const ViewTypeA& A, const ViewTypeB& b, const ViewTypeC& c) {
    typedef typename ViewTypeC::value_type scalar_type;  // The scalar type
    typedef typename ViewTypeC::execution_space execution_space;  // Where we are running

    const int m = A.extent(0);
    const int n = A.extent(1);
    Kokkos::parallel_for(
        Kokkos::RangePolicy<execution_space>( 0,m ), // Iterate over [0,m)
        KOKKOS_LAMBDA (const int i) {
            scalar_type t = 0.0;
            for (int j=0; j<n; ++j)
                t += A(i,j)*b(j);
            c(i) = t;
        }
    );
}

// Use default execution space (OpenMP, Cuda, ...) and memory layout for that space
Kokkos::View<double**> A("A",m,n);  // Create rank-2 array with m rows and n columns
Kokkos::View<double*> b("b",n);  // Create rank-1 array with n rows
Kokkos::View<double*> c("c",m);  // Create rank-1 array with m rows

// ...
run_mat_vec(A,b,c);
# Layout Polymorphism for Performant Memory Accesses

- **CPU**
  - Each thread accesses contiguous range of entries
  - Ensures neighboring values are in cache

- **GPU**
  - Each thread accesses strided range of entries
  - Ensures coalesced accesses (consecutive threads access consecutive entries)

---

## Architecture Description

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Description</th>
<th>Execution Space</th>
<th>Measured Bandwidth (GB/s)</th>
<th>Expected Throughput (GFLOP/s)</th>
<th>Measured Throughput (GFLOP/s)</th>
<th>Wrong Layout (GFLOP/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skylake (1 socket)</td>
<td>Intel Xeon Gold 6154, 36 threads</td>
<td>OpenMP</td>
<td>64.4</td>
<td>16.1</td>
<td>18.0</td>
<td>15.3</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA V100</td>
<td>Cuda</td>
<td>833</td>
<td>208</td>
<td>213</td>
<td>26.3</td>
</tr>
</tbody>
</table>
Sacado and Kokkos?

- What happens when we use Sacado AD on manycore architectures with Kokkos?

- Kokkos::View< Sacado::Fad::SFad<double,p>**>:  
  - Derivative components always stored consecutively  
  - CPU: Good cache, vector performance  
  - GPU: Large stride causes bad coalescing
Sacado/Kokkos Integration

- Want good AD performance with no modifications to Kokkos kernels
- Achieved by specializing Kokkos::View data structure for Sacado scalar types
  - Rank-\( r \) Kokkos::View internally stored as a rank-\( (r+1) \) array of double
  - Kokkos layout applied to internal rank-\( (r+1) \) array
AD Performance Portability

Kokkos::View<Sacado::Fad::SFad<double,p>>:: A("A",m,n,p); // Create rank-2 array with m rows and n columns
Kokkos::View<Sacado::Fad::SFad<double,p>> b("b",n,p); // Create rank-1 array with n rows
Kokkos::View<Sacado::Fad::SFad<double,p>> c("c",m,p); // Create rank-1 array with m rows

// ...

run_mat_vec(A,b,c);

SFad, Derivative dimension p=8

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Expected Throughput (GFLOP/s)</th>
<th>Measured Throughput (GFLOP/s)</th>
<th>No View Specialization (GFLOP/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skylake</td>
<td>30.4</td>
<td>34.1</td>
<td>34.0</td>
</tr>
<tr>
<td>GPU</td>
<td>393</td>
<td>395</td>
<td>317</td>
</tr>
</tbody>
</table>

AD Throughput on NVIDIA V100 GPU

Throughput (GFLOP/s)

Derivative Dimension

View Spec.
No View Spec.
Hierarchical Parallelism

- Layout approach was explored to minimize code user-code changes for Sacado
  - Differentiate code without changing parallel scheduling

- Derivative propagation provides good opportunities for exposing more parallelism
  - Parallelism across derivative array
  - Code may not expose enough parallelism natively (e.g., small workset)

- Motivation is PDE assembly using worksets
  - Many codes group mesh cells into batches called worksets
  - Threaded parallelism over cells in each workset: want large worksets for GPUs with very high concurrency
  - Memory required proportional to size of workset: want small worksets because of limited high-bandwidth memory on GPUs

- Solution: apply fine-grained (warp-level) parallelism across derivative dimension on GPUs
  - Implementation uses Cuda code hidden behind Sacado’s overloaded operators
Advection Kernel Example

\[ r = \int_e \left( \vec{f}(x) \cdot \nabla \varphi(x) + s(x)\varphi(x) \right) \, dx \]

Kokkos::View<ScalarT, *, Layout, ExecSpace> wgb;
Kokkos::View<ScalarT, **, Layout, ExecSpace> flux;
Kokkos::View<ScalarT, **, Layout, ExecSpace> wbs;
Kokkos::View<ScalarT, **, Layout, ExecSpace> src;
Kokkos::View<ScalarT, **, Layout, ExecSpace> residual;

typedef Kokkos::RangePolicy<ExecSpace> Policy;
Kokkos::parallel_for(
    Policy( 0, num_cell ),
    KOKKOS_LAMBDA( const int cell )
    {

      for (int basis=0; basis<num_basis; basis++)
        {
          ScalarT value(0), value2(0);
          for (int qp=0; qp<num_points; ++qp)
            {
              for (int dim=0; dim<num_dim; ++dim)
                value += flux(cell,qp,dim)*wgb(cell,basis,qp,dim);
              value2 += src(cell,qp)*wbs(cell,basis,qp);
            }
      residual(cell,basis) = value+value2;
    });
Advection Kernel Example

\[ r = \int_e \left( \vec{f}(x) \cdot \nabla \varphi(x) + s(x) \varphi(x) \right) \, dx \]

const int VectorSize = 32;
typedef Kokkos::LayoutContiguous<Layout,VectorSize> ContLayout;
Kokkos::View<ScalarT****, ContLayout, ExecSpace> wgb;
Kokkos::View<ScalarT***, ContLayout, ExecSpace> flux;
Kokkos::View<ScalarT***, ContLayout, ExecSpace> wbs;
Kokkos::View<ScalarT***, ContLayout, ExecSpace> src;
Kokkos::View<ScalarT***, ContLayout, ExecSpace> residual;

typedef typename ThreadLocalScalarType<decltype(src)>::type local_scalar_type;
typedef Kokkos::TeamPolicy<ExecSpace> Policy;
Kokkos::parallel_for(
    Policy( num_cell, Kokkos::AUTO, VectorSize ),
    KOKKOS_LAMBDA( const typename Policy::member_type& team )
    {
        const int cell = team.league_index();
        const int ti = team.team_index();
        const int ts = team.team_size();
        for (int basis=ti; basis<num_basis; basis+=ts) {
            local_scalar_type value(0), value2(0);
            for (int qp=0; qp<num_points; ++qp) {
                for (int dim=0; dim<num_dim; ++dim)
                    value += flux(cell,qp,dim)*wgb(cell,basis,qp,dim);
                value2 += src(cell,qp)*wbs(cell,basis,qp);
            }
            residual(cell,basis) = value+value2;
        }
    });
Applications

- Turbulent CFD
- Magnetohydrodynamics

Discretizations & Algorithms

- Compatible Discretizations
- Algebraic Multigrid (>100k cores)

Uncertainty Quantification

- IMEX
- PDE Constrained Optimization

\[ A = \begin{bmatrix} I & I \\ BF^{-1} & B^T \\ S \end{bmatrix} \]
\[ S = C - BF^{-1}B^T \]

Block Preconditioning

\[ \rho \quad \rho u \quad \varepsilon \quad \mathbf{E} \quad \mathbf{B} \]
Hierarchical Parallelism in Panzer

- Diffusion problem with mixed finite element discretization:

\[
\nabla^2 \phi = f \quad \text{on } \Omega \\
\phi = \phi_T \quad \text{on } \Gamma = \partial \Omega \\
\int_{\Omega} (\nabla \cdot g - f) (\nabla \cdot w) d\Omega = 0 \quad \forall w \in \mathcal{H}_r(\nabla)
\]

<table>
<thead>
<tr>
<th>Description</th>
<th>Operator</th>
<th>Panzer C++ Class Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Evaluate g at Quadrature Points</td>
<td>( g = \sum_i g_i w_i )</td>
<td>DOF</td>
</tr>
<tr>
<td>2. Evaluate ( \nabla \phi ) at Quadrature Points</td>
<td>( \nabla \phi = \sum_i \phi_i \nabla q_i )</td>
<td>DOFGradient</td>
</tr>
<tr>
<td>3. Evaluate ( \nabla \cdot g ) at Quadrature Points</td>
<td>( \nabla \cdot g = \sum_i g_i \nabla \cdot w_i )</td>
<td>DOFDiv</td>
</tr>
<tr>
<td>4. Integrate Eq. 6 with ( h = \nabla \phi - g )</td>
<td>( \int_{\Omega} (h) \cdot (\nabla q) d\Omega )</td>
<td>Integrate_GradBasisDotVector</td>
</tr>
<tr>
<td>5. Integrate Eq. 5 with ( s = \nabla \cdot g - f )</td>
<td>( \int_{\Omega} (s) (\nabla \cdot w) d\Omega )</td>
<td>Integrate_DivBasisTimesScalar</td>
</tr>
</tbody>
</table>

![Graph showing speedup over flat parallelism vs. workset size](image)
Concluding Remarks

- Highly parallel architectures like GPUs are here
  - AD tools and techniques need to work in these environments

- Sacado solves this problem through integration with Kokkos
  - Leverage layout polymorphism to enable AD of Kokkos kernels without modification
  - Incorporate GPU vector/warp-level parallelism for improved performance

- Code and performance tests are available within Sacado (and Panzer)
  - [https://github.com/trilinos](https://github.com/trilinos)

- Sacado+Kokkos part of several large code projects at Sandia
  - Albany ([https://github.com/SNLComputation/Albany](https://github.com/SNLComputation/Albany))
  - Panzer/Drekar
  - ATDM

- Future work:
  - Higher derivatives (Kokkos specializations for nested Sacado AD types)
  - Reverse mode with Kokkos?
Some Negative Implications

- View access operator returns AD handle object (pointing into rank-(r+1) array)
  - `View<SFad<double,p>**>::operator(i,j)` returns
    `ViewFad<double>(ptr+offset(i,j), stride, p)` temporary
  - Not the same as `SFad<double,p>&`
  - Can't take address of return value

- View constructor needs AD derivative dimension (needed to properly allocate internal array)
  - `View<SFad<double,p>**>(m,n,p)`

- Introduces challenges for
  - Templating on scalar type: `View<T**>` operates differently depending on type of T
  - Porting codes to use Kokkos: Can't get pointer of type `T*` to pass to legacy code

- Unclear how to efficiently extend this to nested Fad objects for higher derivatives