Automatic Differentiation on Emerging Manycore Architectures with Sacado and Kokkos

Eric Phipps (etphipp@sandia.gov) and H. Carter Edwards
Sandia National Laboratories

2017 Albany Users Meeting

Jan 17-18, 2017

SAND2017-0572 PE
Sacado Automatic Differentiation Tools in Albany

• Albany makes heavy use of AD as an enabling technology
  – Jacobians, Hessians, parameter derivatives, ...

• Forward mode AD tools:
  – Derivative objects consist of a value and derivative array
  – DFad<double>: Derivative array length chosen at run-time
  – SFad<double,N>: Derivative array length = N
  – SLFad<double,N>: Derivative array length at most N

• AD applied in Albany at element level through template-based
generic programming
  – Template physics evaluators on scalar type
  – Instantiate evaluators on AD data types
  – Specialized gather-scatter evaluators initialize each AD object and
assemble global derivative objects (e.g., Jacobian)

• Use of AD in evaluators requires Sacado to work with Kokkos
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.dimension_0();
const int n = A.dimension_1();
Kokkos::parallel_for(
    Kokkos::RangePolicy<execution_space>( 0,m ), // Iterate over [0,m)
    KOKKOS_LAMBDA (const int i) { // "[=]" (capture by value)
        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);
Kokkos Layout Polymorphism for Performant Memory Accesses

- **CPU/MIC**
  - 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)
## Performance Portability

<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>Haswell (1 socket)</td>
<td>Intel Xeon E5-2698 v3, 32 threads</td>
<td>OpenMP</td>
<td>47.4</td>
<td>11.9</td>
<td>13.0</td>
<td>6.8</td>
</tr>
<tr>
<td>MIC</td>
<td>Intel Xeon Phi 7120P, 240 threads</td>
<td>OpenMP</td>
<td>147</td>
<td>36.8</td>
<td>35.9</td>
<td>3.4</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA K80</td>
<td>Cuda</td>
<td>150</td>
<td>37.5</td>
<td>42.0</td>
<td>7.4</td>
</tr>
</tbody>
</table>

- \(m = 1e6, n=100\)

- Expected Throughput = Measured Bandwidth \(\times\) 2 FLOPS / 8 Bytes

- Manually specify incorrect layout for “Wrong Layout”, e.g.,

  ```cpp
  Kokkos::View<double**, Kokkos::LayoutRight, Kokkos::Cuda> A("A",m,n);
  Kokkos::View<double*, Kokkos::LayoutRight, Kokkos::Cuda> b("b",n);
  Kokkos::View<double*, Kokkos::LayoutRight, Kokkos::Cuda> c("c",m);
  ```

* Bandwidth measured through STREAM (Triad) benchmark
What happens when we use Sacado in Kokkos parallel kernels?

• Kokkos::View< Sacado::Fad::SFad<double,p>**>:
  – Derivative components always stored consecutively
  – CPU: Good cache, vector performance
  – GPU: Large stride causes bad coalescing
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 doubles
  – Kokkos layout applied to internal rank-(r+1) array
Some Implications

- **View<SFad<double,p>** does not store array of SFad objects
  - Internally stores array of doubles

- 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/reference of return value

- View constructor needs AD derivative dimension
  - Needed to properly allocate internal array
  - `View<SFad<double,p>**>(m,n,p+1)`

- 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

- Solutions for compile-time polymorphic code:
  - Use `View::reference_type` when requesting a reference to a View element
  - Use `ViewFactory` for creating temporary View’s in evaluators
### AD Performance Portability

**Architecture** | **Measured Bandwidth (GB/s)** | **Expected Throughput (GFLOP/s)** | **Measured Throughput (GFLOP/s)** | **No View Specialization (GFLOP/s)**
---|---|---|---|---
Haswell | 47.4 | 22.4 | 24.3 | 23.1
MIC | 147 | 69.4 | 69.4 | 43.2
GPU | 150 | 70.8 | 81.2 | 35.1

- \(m = 1e6, n=100, p = 8\) (derivative dimension)
- Expected Throughput \(\sim\) Measured Bandwidth \(\times (4p+2)\) FLOPS / \(8(p+1)\) Bytes
- \texttt{SFad\langle double, p\rangle} AD data type

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

// ...
run_mat_vec(A,b,c);
```
Throughput Varying Derivative Dimension

NVIDIA K80 GPU
(m=10^5, n=100)

Throughput (GFLOP/s)

Derivative Dimension

View Spec.
No View Spec.
Hierarchical Parallelism

- Layout approach was explored to minimize code user-code changes for Sacado

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

- Exploring modifications to Sacado, Kokkos to map GPU thread parallelism across derivative calculations
  - Outcome of ATDM FY16 L2 Milestone
  - With Christian Trott, Eric Cyr, Matt Bettencourt, Roger Pawlowski

- Two approaches are being explored:
  - Reference vector index (threadIdx.x) in Sacado overloaded operators
  - Create strided AD objects in View::operator() with derivative size reduce by thread warp dimension (blockDim.x)
Advection Kernel Example (from Panzer)

\[ \int_c^e c \left( \vec{f}(x) \cdot \nabla \varphi_i(x) + s(x) \varphi_i(x) \right) \, dx \]

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;
ScalarT coeff;

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) = coeff*(value+value2);  
      }  
   });
# Kernel with Hierarchical DFad

```cpp
#define SACADO_VIEW_CUDA_HIERARCHICAL_DFAD 1
#define SACADO_KOKKOS_USE_MEMORY_POOL 1

// LayoutContiguous triggers a new Sacado View specialization
// where derivative components are always kept contiguous
typedef Kokkos::LayoutContiguous<Layout> 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;
ScalarT coeff;

// Compute the size of memory required for a memory pool to manage
// DFad derivative array allocations
const bool is_cuda = std::is_same<ExecSpace, Kokkos::Cuda>::value;
const int warp_size = is_cuda ? 32 : 1;
const int concurrency = ExecSpace::concurrency();
const int s = Kokkos::dimension_scalar(src);
const int fad_dim = s > 0 ? s-1 : 1;
const size_t mem_pool_size =
    static_cast<size_t>(1.2*concurrency()*2*fad_dim*sizeof(double))/warp_size;
```
// Allocate memory pool and set global variables to make it
// accessible to Sacado overloaded operators
Sacado::createGlobalMemoryPool(ExecSpace(), mem_pool_size);

// Team-vector dimensions for hierarchical parallelism
typedef Kokkos::TeamPolicy<ExecSpace> Policy;
const int vector_size = is_cuda ? 32 : 1;
const int team_size = is_cuda ? 256 / vector_size : 1;

Kokkos::parallel_for(
    Policy( num_cell,team_size,vector_size ),
    KOKKOS_LAMBDA( const typename Policy::member_type& team )
    {
        const size_t cell = team.league_rank();
        const int team_index = team.team_rank();

        for (int basis=team_index; basis<num_basis; basis+=team_size) {
            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) = coeff*(value+value2);
        }
    });

Sacado::destroyGlobalMemoryPool(ExecSpace());
#define SACADO_VIEW_CUDA_HIERARCHICAL 1

// LayoutContiguous triggers a new Sacado View specialization
// where derivative components are always kept contiguous. Stride
// must equal the vector dimension below and is used to determine
// the thread-local derivative array length for SFad
const bool is_cuda = std::is_same<ExecSpace, Kokkos::Cuda>::value;
const int Stride = is_cuda ? 32 : 1;
typedef Kokkos::LayoutContiguous<Layout, Stride> 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;
ScalarT coeff;

// Typename of thread-local AD type incorporating the Stride above.
// For ScalarT == SFad, this will be SFad if and only if the derivative
// length is divisible by Stride, otherwise it will be SLFad
typedef typename Kokkos::ThreadLocalScalarType<decltype(src)>>::type local_scalar_type;
// Team-vector dimensions for hierarchical parallelism
typedef Kokkos::TeamPolicy<ExecSpace> Policy;
const int vector_size = Stride;
const int team_size = is_cuda ? 256 / vector_size : 1;

Kokkos::parallel_for(
    Policy( num_cell,team_size,vector_size ),
    KOKKOS_LAMBDA( const typename Policy::member_type& team )
    {
        const size_t cell = team.league_rank();
        const int team_index = team.team_rank();

        // Divide up the derivative components of coeff across threads
        const local_scalar_type c = Sacado::partition_scalar<Stride>(coeff);

        for (int basis=team_index; basis<num_basis; basis+=team_size) {
            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) = c*(value+value2);
        }
    });
Performance

NVIDIA K20x GPU
(p = 50)

<table>
<thead>
<tr>
<th></th>
<th>Time per Cell (sec)</th>
<th>Cells per Workset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat SFad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flat SLFad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flat DFad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hier. SFad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hier. SLFad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hier. DFad</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: p = 64 is used for Hierarchical SFad
Concluding Remarks

- Sacado specializes Kokkos data structures for AD performance portability
  - Allows Sacado to differentiate Kokkos kernels with little to no modifications
  - Provides generally good performance portability
  - ViewFactory and reference_type aid writing generic code

- Hierarchical parallelism can be used to increase performance
  - Map GPU warp across derivative dimension

- But requires more changes to Kokkos kernels
  - Use Team execution policy
  - Use new contiguous layout
  - Memory pool for DFad (requires bounding amount of memory allocation in kernel)
  - Thread-local scalar type for SFad/SLFad
  - Limitations in current implementations require vector_size == 32

- Unclear what approach codes such as Albany should take
  - Comments? Thoughts? Complaints?
Auxiliary Slides
Kokkos*: performance portability for C++ applications

κόκκος: “granule” or “grain”; like grains of sand on a beach