KokkosArray
A C++ Library for Manycore Performance-Portability

H. Carter Edwards, Christian Trott, Daniel Sunderland
Sandia National Laboratories

2012 Trilinos User Group

SAND2012-9215P
Unlimited Release

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy's National Nuclear Security Administration
under contract DE-AC04-94AL85000.

Project Charter

- R&D within ASC CSSE
  - CSSE: Computational Systems and Software Environment
  - "Heterogeneous Computing" project
    - PM: Rob Hoekstra (1426), PI: Carter Edwards (1444)
    - Effective use of heterogeneous architectures
    - Emphasis on heterogeneity at the node-level
  - Heterogeneous parallelism (MPI + threading + vectorization)
- Deliverables
  - ➢ Research performance-portable programming models
  - ➢ Develop proxy-applications to demonstrate and evaluate programming model

KokkosArray Library

- **KokkosArray IS:**
  - An implementation of the programming model
  - Consolidation of proxy-applications' common functionality
  - "Low level" enabling data structures and algorithms
  - Extremely attentive to:
    1. Portability & performance (as per project charter)
    2. Usability: ease of use, error detection, extensibility, maintainability, ...
- **KokkosArray IS NOT:**
  - A linear algebra library
  - A discretization library
  - A mesh library
  - ➢ Intent: Build such libraries on top of KokkosArray

The Problem / Challenge
Future of HPC: Manycore Accelerators

- **Multicore CPU**
  - **Increasing core counts with decreasing global memory / core**
  - **Cores share caches and memory controllers**
  - **Non-uniform memory access (NUMA), performance issues**
  - **Increasing vector unit lengths**
  - ➢ **Memory access patterns critical for *best* performance**
- **Manycore GPU (e.g., NVIDIA Kepler, AMD Fusion)**
  - **Physically separate memory with data-transfer overhead**
  - **Work-dispatch interaction between host and device**
  - **Memory controller optimized for thread-gang (warp) based access**
  - ➢ **Memory access patterns critical for *acceptable* performance**
- **Is all about Memory Access Patterns**

The Problem / Challenge Future of HPC: Manycore Accelerators

- Shared Memory Threading within MPI is *required*
  - Cannot run MPI-everywhere on GPU
  - Cannot afford MPI process memory for every core
  - Cannot scale MPI collectives to millions of CPU cores
    - Unless you have heroic hardware: Blue Gene Q
- Memory Access Patterns are Critical
  - Correctness – no race conditions among threads
  - Performance – proper blocking or striding
- Access Pattern Requirements are Device-dependent
  - CPU-core : blocking for cache and cache-lines
  - GPU : striding for coalesced access
  - "array of structures" vs. "structure of arrays"

Programming Model Concept

- **Manycore Device**
  - **Has a separate memory space (physically or logically)**
  - **Dispatch work to cores/threads of the device**
  - **Work : computations + data residing on the device**
  - **Currently supported devices CPU+pthreads, CUDA**
- **Classic Multidimensional Arrays, *with a twist***
  - **Map multi-index (i,j,k,...) ↔ memory location *on the device***
    - **Should be efficient for both memory used and time to compute**
  - **Map is derived from a Layout**
  - ➢ **Choose Layout for device-specific access pattern requirements**
    - **Layout must change when porting among devices**
  - **Layout changes are transparent to the user code;**
  - ➢ **IF the user code honors the simple array API: a(i,j,k,...)**

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy's National Nuclear Security Administration
under contract DE-AC04-94AL85000.

Programming Model Implementation

- **Standard C++ Library, not a Language extension**
  - **In *spirit* of Intel's TBB, NVIDIA's Thrust & CUSP, MS C++AMP, ...**
  - **Not a language extension like OpenMP, OpenACC, OpenCL, CUDA**

- **Template Meta-Programming**
  - **For device-specializations and array layout polymorphism**
  - **C++1998 standard (would really be nice to have C++2011)**

- **Extremely Attentive to:**
  1. **Portability – the project charter R&D constraint**
  2. **Performance – the project charter R&D objective**
  3. **Usability – the SQE objective**

Current Capabilities

- **Multidimensional Arrays**
  - Declare dimensions and access data members
  - Allocate and deallocate in Device memory space
  - Deep-copy data between host and device memory space
  - Optionally choose or define your own Layout
- **Parallel-For and Parallel-Reduce**
  - Define thread-parallel work functors (function + data)
  - Dispatch work to device
  - Optionally wait for dispatched work to complete
  - Reduction is guaranteed deterministic, given same # of threads
- **Defer Task-Parallelism, Pipeline-Parallelism (for now)**

Multidimensional Array : API

- Multidimensional Array : Basic API

```
class View< double * * [3][8] , Device > a("a",N,M);
```

  - Dimensioned as [N][M][3][8] (two runtime, two compile-time)
  - Allocated in memory space of Device
  - a(i,j,k,l) : access data member via multi-index
    - Multi-index is mapped according to Device's default Layout
- Multidimensional Array : Advanced API

```
class View<double**[3][8], Layout , Device> a("a",N,M);
```

  - ➢ Multi-index access API is unchanged for user code
  - Override Device's default layout
    - E.g., force row-major or column-major
  - *Layout* is an extension point for blocking, tiling, etc.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy's National Nuclear Security Administration
under contract DE-AC04-94AL85000.

Multidimensional Array : API

- **View Memory Management : Basic API**

```
typedef class View<double**,Device> MyMatrixType ;
MyMatrixType a(“a”,N,M); // allocate array
MyMatrixType b = a ; // A new view to the same data
```

  – As per Trilinos standard practice, views are reference counted
    - Internal reference counting to avoid cluttering user-code

- **View Memory Management : Advanced API**

```
class View<const double**,Layout,Device,Unmanaged> c = a ;
```

  – A non-reference counted view
  – Faster to construct, assign, and destroy; *however*,
  ➢ User-code assumes responsibility to destroy ‘c’ before ‘a’
  – Can only allocate managed views

Multidimensional Array : API

- **Host / Device Deep Copy : Basic API**

```
typedef class View<...,Device> MyViewType ;
MyViewType a("a",...) ;
MyViewType::HostMirror a_host = create_mirror( a );
deep_copy( a , a_host ); deep_copy( a_host , a );
```

  - ➢ **NO hidden deep-copy, deep-copy only when told by user-code**
  - – **HostMirror: identical layout in Host space for fast memory-copy**

- **Host / Device Deep Copy: Advanced API**

```
MyViewType::HostMirror a_host = create_mirror_view( a );
```

  - – **If Device uses host memory then 'a_host' is simply a view of 'a'**
  - – **Deep-copy becomes a no-op**
  - – **Avoids deep-copy performance penalty if not needed**

Parallel_For API

- **Thread-Parallel Calls to a Functor on the Device**
  - Dispatch: parallel_for(NP , functor);
- **Functor : A function + its calling arguments**
  - Simple example:

```
template< class DeviceType > // allows for partial-specialization
struct AXPY {
  typedef DeviceType device_type ; // run on this device
  double a ;                        // parameter
  View<double*,device_type> x , y ; // arrays
  void operator()( int ip ) const { y(ip) += a * x(ip); } // function
};
```

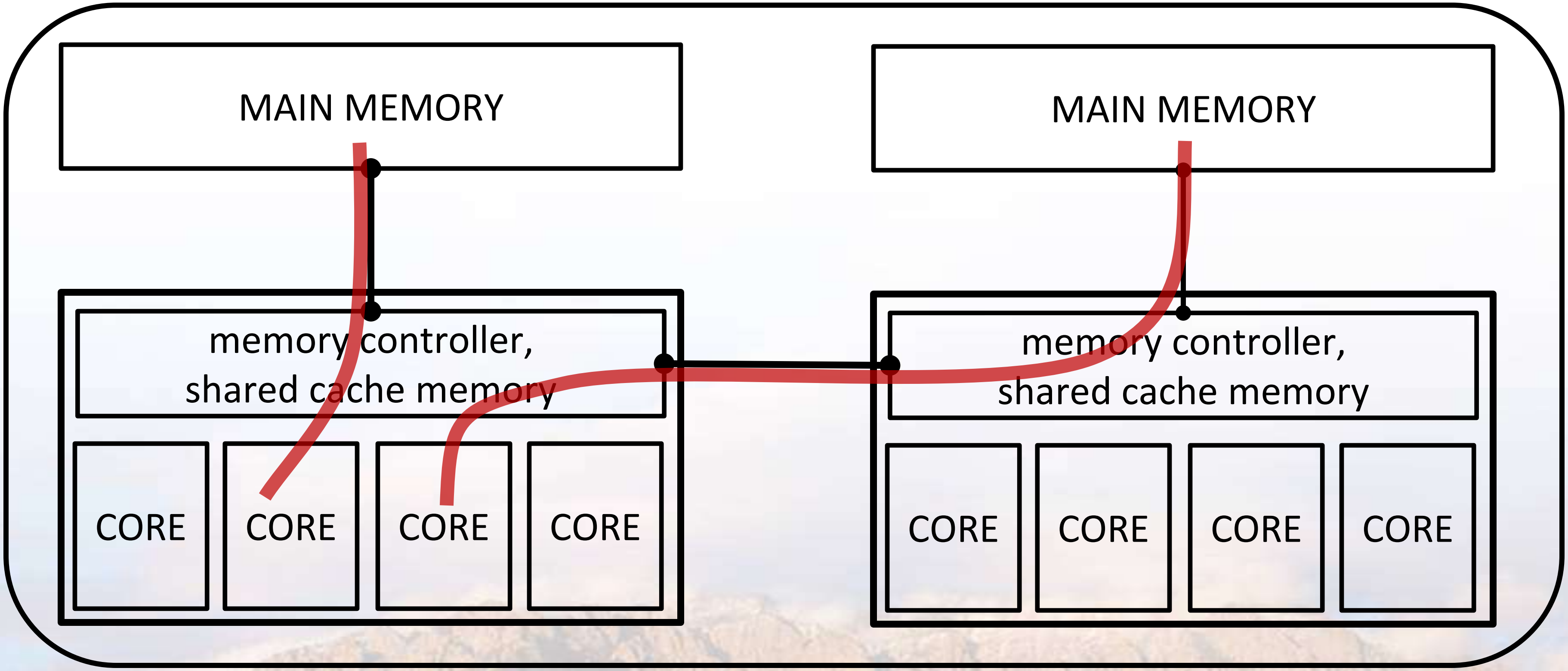
  - Call ‘operator()(ip) NP times where ip $\in$ [0,NP)
  - Array data access uses ‘ip’ to avoid race conditions

Functor Pattern

- **Dispatch NP units of *Work* to Manycore Device**
  - **Work = computation + data**
  - **Called ip ∈ [0,NP) times from (up to) NP different threads**
  - **Functor object is shared by all threads**
    - **Thus: void operator()(int ip) const ;**
- **Why Functor Pattern ?**
  - **Standard C++ and *Portable***
  - **Flexible: as many argument-members as you need**
- **Why Not : traditional Function + Argument List ?**
  - **Requires language / compiler extensions**
  - **E.g., CUDA, OpenCL, OpenACC, OpenMP, ...**
  - **Impedes device-specific specializations**

Parallel Work Affinity for NUMA Performance

- **KokkosArray manages Computation + Data Affinity**
  - **A CPU-core computes on y(ip); so y(ip) should be NUMA-local**
  - **A simplified model:**



Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Parallel_Reduce API
(parallel_for is so easy in comparison)

- **Similar to parallel_for, with *Reduction Argument***
  - Dispatch: result = parallel_reduce(NP , functor);
    - ➢Result is deterministic, given the same device and # threads
    - Result is a value, or View to a value, on the host or device
  - Called ip ∈ [0,NP) times: functor(ip , contribution);

```
struct DOT {
  typedef  DeviceType   device_type ;
  typedef double value_type ;  // type of the reduction argument
  View<double*,device_type> x , y ;
  void operator()( int ip , value_type & contrib ) const
   { contrib += y(ip) * x(ip); }
  // ... to be continued ...
};
```

Parallel_Reduce API
(what makes it harder)

- **Different than parallel_for : *Reduction Argument***
  - Called on up to NP different threads
    - Producing up to NP contributions toward the final result
  - Must reduce per-thread contributions
  - Must manage per-thread temporary data for contributions
  - Must yield deterministic result, for a given device and # threads
- **Flexibility and extensibility**
  - User defined value_type: scalar, simple ‘struct’, simple array
    - Not just a ‘double’
  - Place result on the host or device
  - Post-process result on the device

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000.

Parallel_Reduce API
Inter-Thread Reduction

- **Initialize and Join Per-Thread Contributions**

```
struct DOT {
  // ... continued ...
  typedef double value_type ;
  static void init( value_type & contrib ) { contrib = 0 ; }
  static void join( volatile value_type & contrib ,
                    const volatile value_type & input )
    { contrib = contrib + input ; }
};
```

  - Initialize thread's contrib via Functor::init
  - Join threads' contrib via commutative Functor::join
  - 'volatile' to insure correct inter-thread memory access
    - Prevents compiler from optimizing away join operation

Parallel_Reduce API : Advanced

- **Reduction Argument : A ‘struct’**

```
struct Centroid {
  typedef  DeviceType  device_type ;
  struct value_type { double x[3], mass ; }; // struct value_type
  View<double*[3],device_type> point ;
  View<double*,   device_type> mass ;
  void operator()( int ip , value_type & contrib ) const
  {
    contrib.x[0..2] += point(ip,0..2) * mass(ip); // pseudo code
    contrib.mass  += mass(ip);
  }
  static void init( value_type & contrib ) {...}
  static void join( volatile value_type & contrib ,
                    const volatile value_type & input ) {...}
};
```

Parallel_Reduce API : Advanced

- **Reduction Argument : Runtime-sized Array**

```
struct MultiVectorDOT {
  typedef DeviceType device_type ;
  typedef double value_type[ ] ;  // runtime array type
  const unsigned value_count ;  // runtime array count

  void operator()( int ip , double contrib[ ] ) const ;
  static void init( double contrib[ ] , unsigned count );
  static void join( volatile double contrib[ ] ,
                    const volatile double input[ ] ,
                    unsigned count );
};
```

  – Result is an array, or View to an array on the host or device

Parallel_Reduce API : Advanced

- “Finalizing” the Reduction Argument
  - A final, serial computation performed on the device
  - Example: norm2 requires a serial ‘sqrt’ of the dot product result
    - Store result on device; avoid device-host-device round-trip
    - parallel_reduce(NP , dot , norm2_finalize)

```
struct Norm2Finalize {
  typedef DeviceType device_type ;
  typedef double value_type ;
  View<double,device_type> view ;
  // called by one thread with the reduction result:
  void operator()( const value_type & result ) const
    { *view = sqrt( result ); }
};
```

Finite-Element Proxy-Applications
see kokkos/array/usecases

- **Explicit Dynamics : computationally intensive**
  - Element stress and internal force contributions to nodes
  - Node gather-assemble forces, apply boundary condition, compute acceleration, integrate motion
  - MPI + KokkosArray hybrid parallel
- **Nonlinear Thermal Conduction : memory intensive**
  - Newton iteration to solve nonlinear equation
  - Element computation of residual and Jacobian
  - Gather-assemble sparse linear system; CG iterative solver
  - Update nonlinear solution
  - MPI + KokkosArray hybrid parallel
- **Same finite element kernel source code on all devices**
  - Template instantiation inserts device specific array-maps

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy's National Nuclear Security Administration
under contract DE-AC04-94AL85000.

Plans

- Ports to OpenMP, Intel Phi (MIC), and AMD Fusion
- Tiled Array Layouts
- Embedded Data Types : View< Type **[3][8], device >
  - Type can be a UQ expansion, automatic differentiation, ...
- Multi-Functor Dispatch
- "Alpha" Use, Evaluation, and Improvement-Steering by
  - Tpetra, Mark Hoemann
  - UQ-on-GPU LDRD, Eric Phipps
  - LAMMPS ? Exploring via miniMD, up next: Christian Trott
  - Sierra Toolkit ? up last: Daniel Sunderland
  - Your library / application???
- Transition from "Experimental" to "Primary Stable" FY13