SAND2014-16794PE

An Update on Kokkos, Our C++ Library for Manycore Performance Portability

Photos placed in horizontal position with even amount of white space between photos and header

Photos placed in horizontal position with even amount of white space between photos and header

Computational Science Seminar Series

August 19, 2014

SAND2014-*****PE (Unlimited Release)

Sandia National Laboratories

Exceptional service in the national interest

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Increasingly Complex Heterogeneous Future

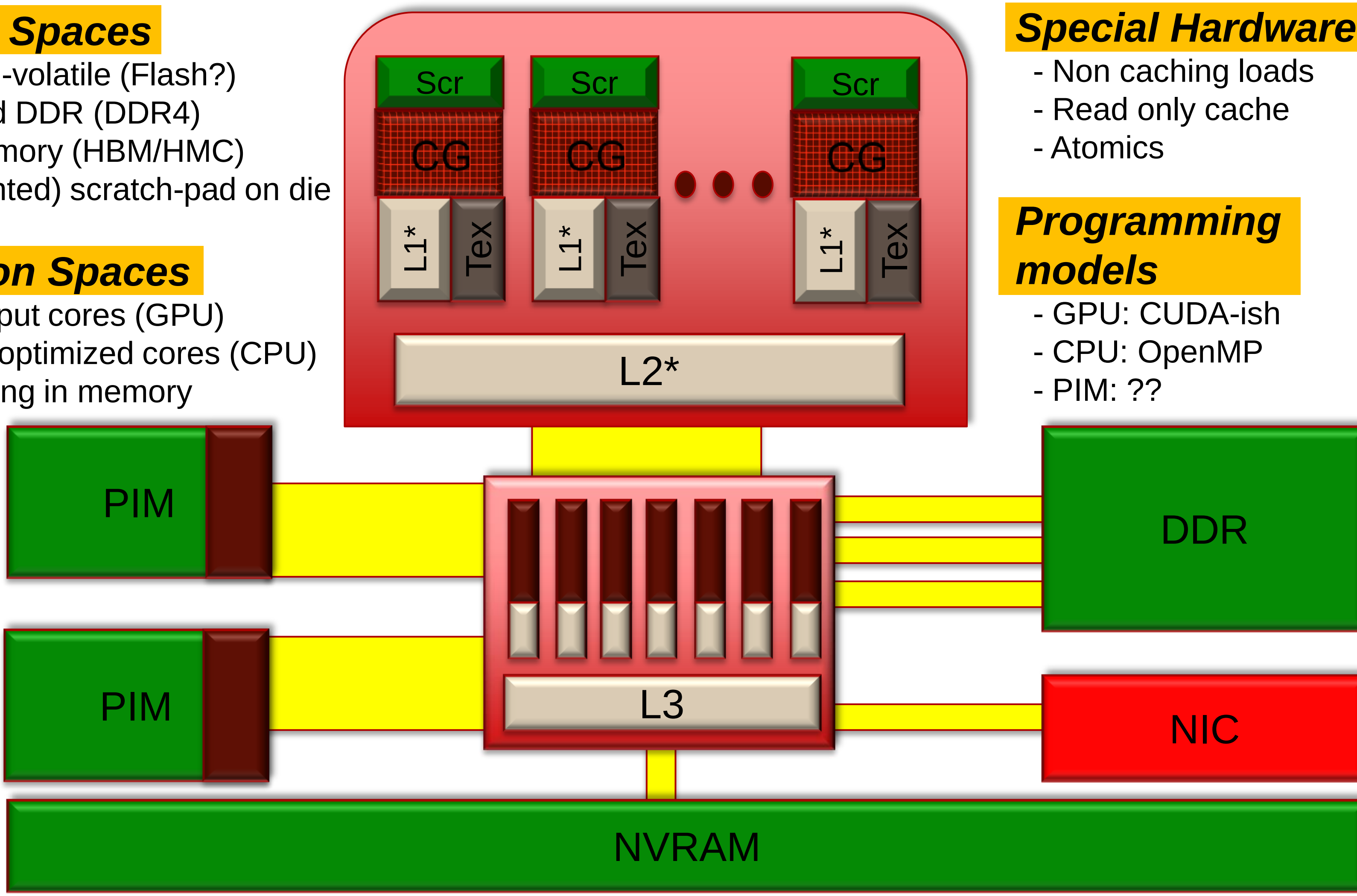
¿ Performance Portable and Future Proof Codes?

Memory Spaces
- Bulk non-volatile (Flash?)
- Standard DDR (DDR4)
- Fast memory (HBM/HMC)
- (Segmented) scratch-pad on die

Execution Spaces
- Throughput cores (GPU)
- Latency optimized cores (CPU)
- Processing in memory

Special Hardware
- Non caching loads
- Read only cache
- Atomics

Programming models
- GPU: CUDA-ish
- CPU: OpenMP
- PIM: ??



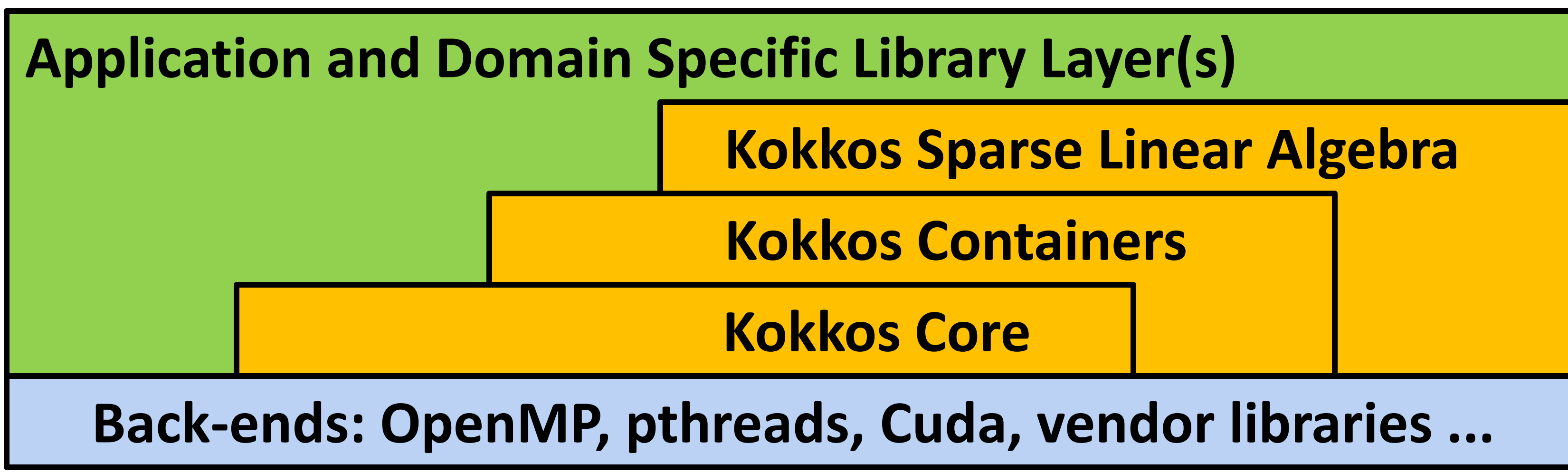
Vision for Managing Heterogeneous Future

- "MPI + X" Programming Model, separate concerns
  - Inter-node: MPI and domain specific libraries layered on MPI
  - Intra-node: Kokkos and domain specific libraries layered on Kokkos
- Intra-node parallelism, heterogeneity & diversity concerns
  - Execution spaces' (CPU, GPU, PIM, ...) diverse performance requirements
  - Memory spaces' diverse capabilities and performance characteristics
  - Vendors' diverse programming models for optimal utilization of hardware
- Desire standardized performance portable programming model
  - Via vendors' (slow) negotiations: OpenMP, OpenACC, OpenCL, C++17
  - Vendors' (biased) solutions: C++AMP, Thrust, CilkPlus, TBB, ArrayFire, ...
  - Researchers' solutions: HPX, StarPU, Bolt, Charm++, ...
- Necessary condition: address execution & memory space diversity
  - Execution { CPU, Xeon Phi, NVIDIA GPU }, Memory { GDDR, DDR, NVRAM }
  - SNL Computing Research Center's Kokkos (C++ library) solution
  - Engagement with ISO C++ Standard committee to influence C++17

Kokkos: A Layered Collection of Libraries

Sandia National Laboratories

- **Standard C++, Not a language extension**
  - In *spirit* of TBB, Thrust & CUSP, C++AMP, LLNL's RAJA, ...
  - *Not* a language extension like OpenMP, OpenACC, OpenCL, CUDA, ...
- **Uses C++ template meta-programming**
  - Rely on C++1998 standard (supported everywhere except IBM's xlC)
  - Moving to C++2011 for concise & convenient lambda syntax
    - Vendors slowly catching up to C++2011 language compliance



Performance Portability Challenge:

Device-Specific Memory Access Patterns are Required

- **CPUs (and Xeon Phi)**
  - Core-data affinity: consistent NUMA access (first touch)
  - Hyperthreads' cooperative use of L1 cache
  - Array alignment for cache-lines and vector units
- **GPUs**
  - Thread-data affinity: coalesced access with cache-line alignment
  - Temporal locality and special hardware (texture cache)
- **¿ "Array of Structures" vs. "Structure of Arrays" ?**
  - ➢ This has been the ***wrong*** question

Right question: Abstractions for Performance Portability ?

Kokkos Performance Portability Answer

- **Thread parallel computation**
  - **Dispatched to an execution space**
  - **Operates on data in memory space(s)**
  - ➢ **How to portably use device-specific memory access pattern?**
- **Multidimensional Arrays, *with a twist***
  - **<u>Layout</u> mapping: array multi-index (i,j,k,...) ↔ memory location**
  - ➢ **Choose layout to satisfy device-specific memory access pattern**
  - **Layout changes are invisible to the user code;**
  - ➢ **IF the user code uses Kokkos' simple array API: a(i,j,k,...)**
- **Manage device specifics under simple portable API**
  - **Dispatch computation to one <u>or more</u> execution spaces**
  - **Polymorphic multidimensional array layout**
  - **Utilization of special hardware; e.g., GPU texture cache**

Performance Evaluations

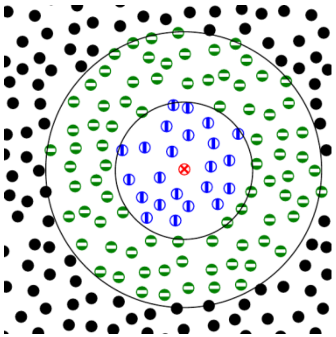
Evaluate Performance Impact of Array Layout

Sandia National Laboratories

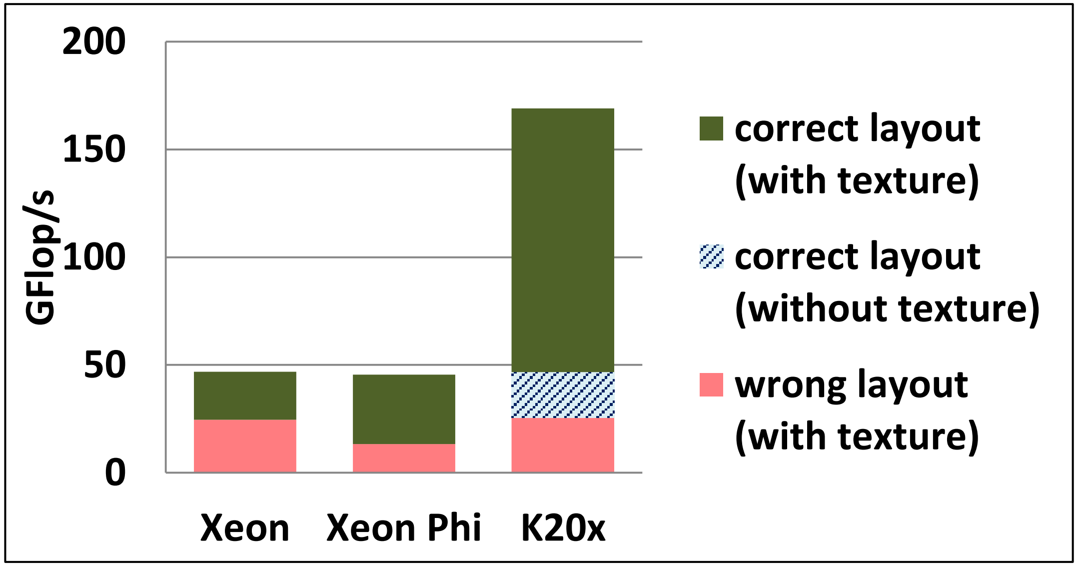
- **Molecular dynamics computational kernel in miniMD**
- **Simple Lennard Jones force model:**
- **Atom neighbor list to avoid N^2 computations**

$$F_i = \sum_{j,\, r_{ij} < r_{cut}} 6\varepsilon \left[\left(\frac{\varsigma}{r_{ij}} \right)^7 - 2 \left(\frac{\varsigma}{r_{ij}} \right)^{13} \right]$$

```
pos_i = pos(i);
for( jj = 0; jj < num_neighbors(i); jj++) {
  j = neighbors(i,jj);
  r_ij = pos_i - pos(j); //random read 3 floats
  if (|r_ij| < r_cut) f_i += 6*e*((s/r_ij)^7 - 2*(s/r_ij)^13)
}
f(i) = f_i;
```



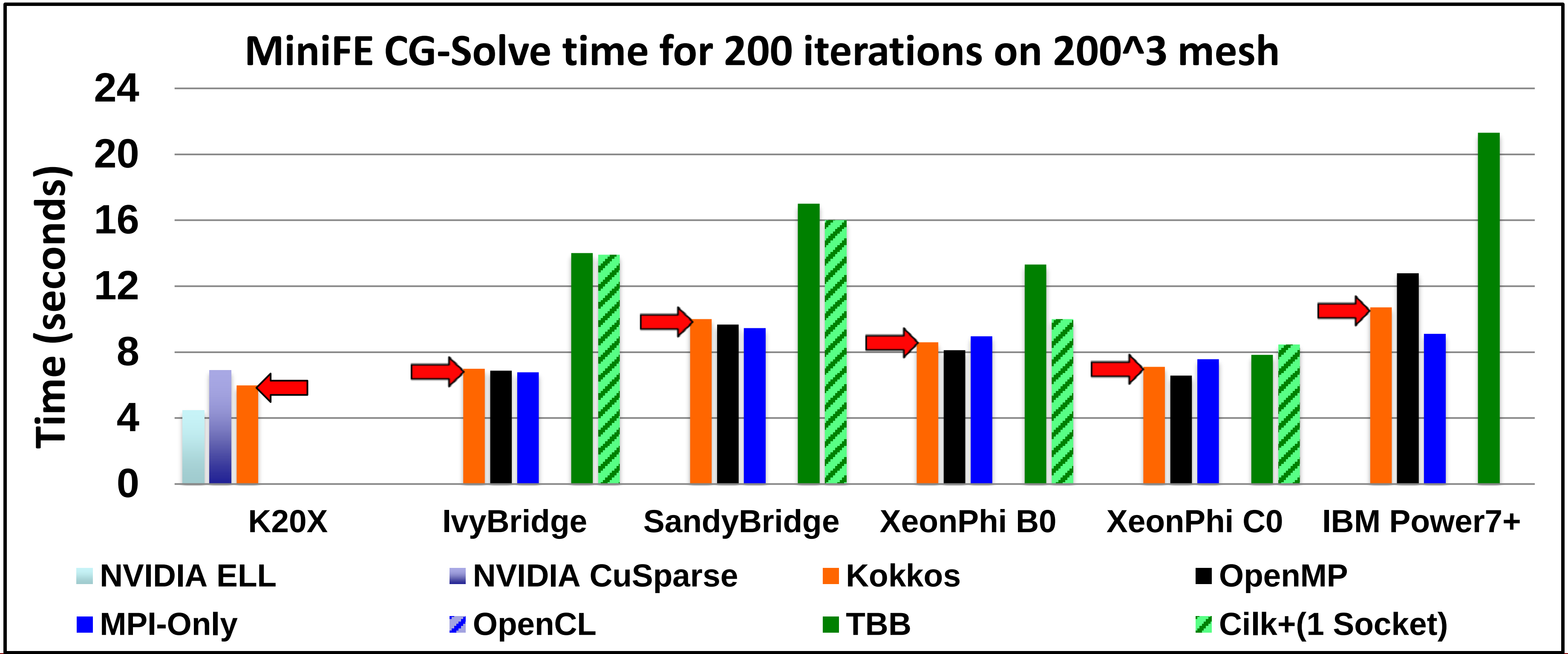
- **Test Problem**
  - 864k atoms, ~77 neighbors
  - 2D neighbor array
  - Different layouts CPU vs GPU
  - Random read 'pos' through GPU texture cache
- **Large performance loss with wrong array layout**



Evaluate Performance Overhead of Abstraction

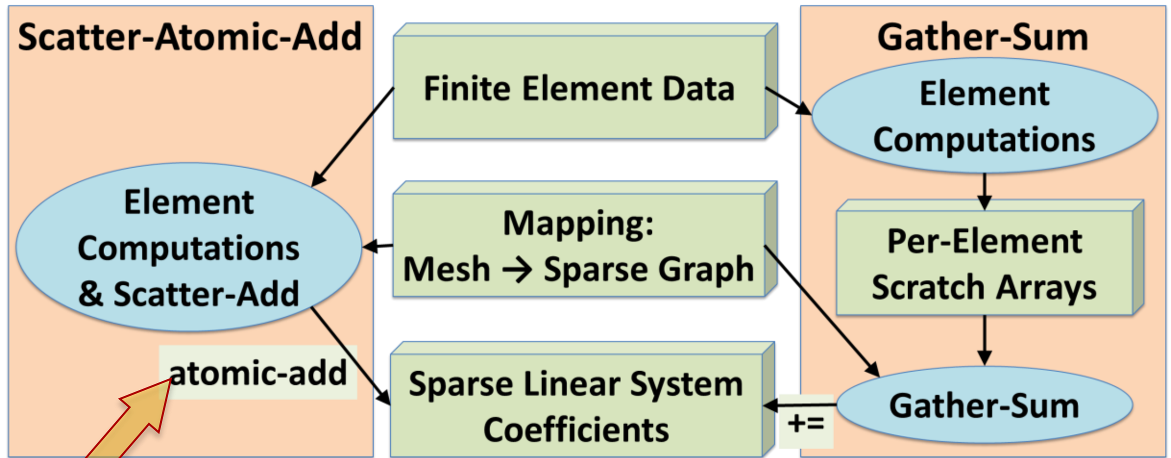
Kokkos competitive with native programming mechanisms

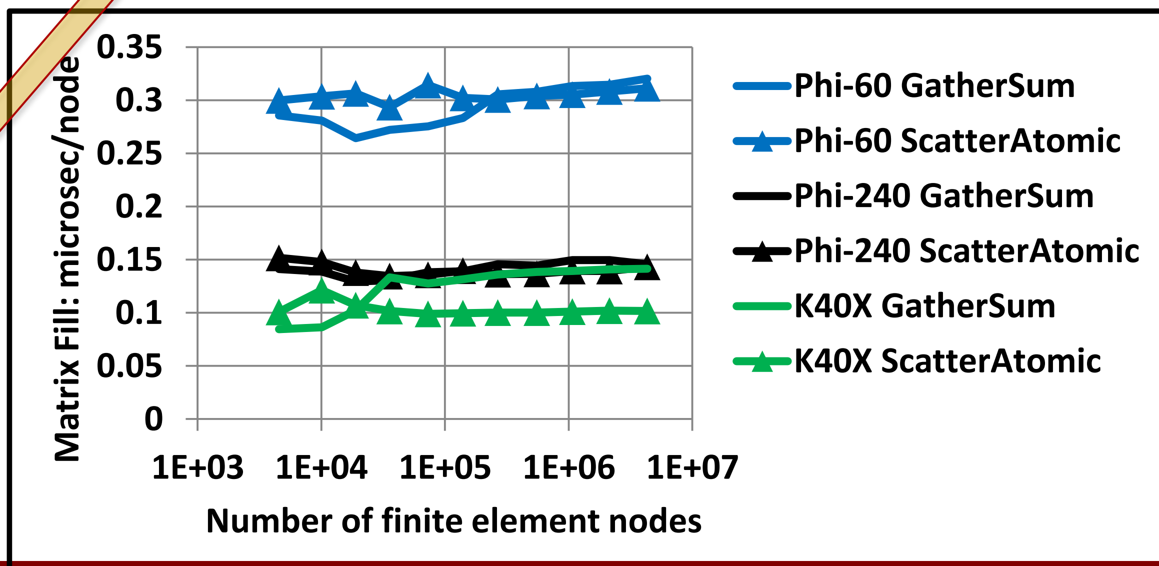
- MiniFE: finite element linear system iterative solver mini-app
- Compare to versions specialized for programming models
- Running on hardware testbeds



Thread-Scalable Fill of Sparse Linear System

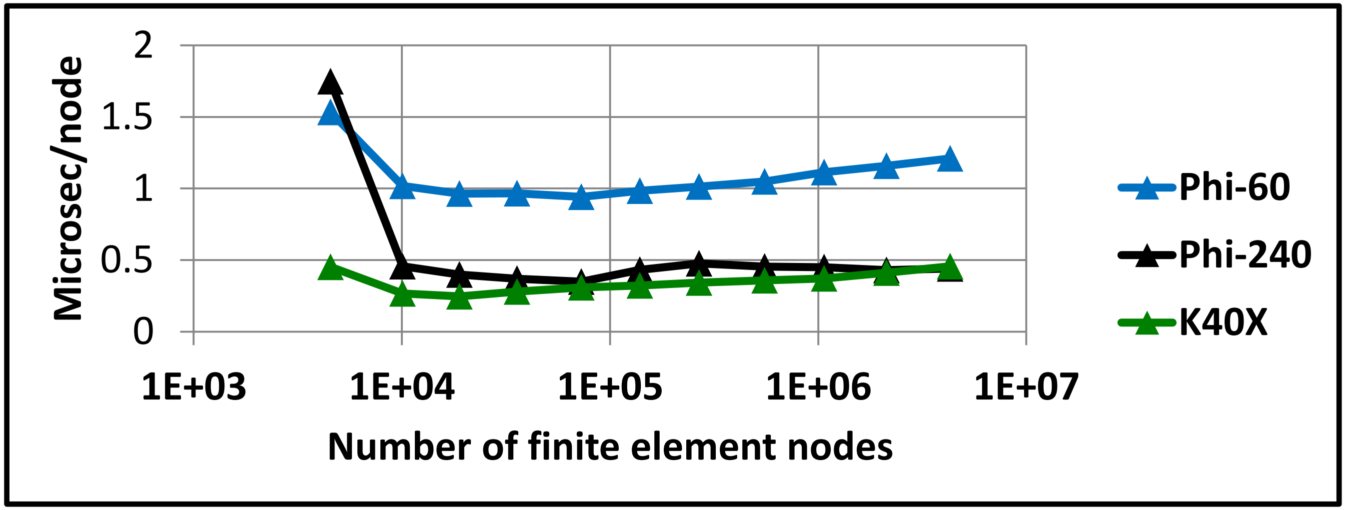
- **MiniFENL: Newton iteration of FEM:** $x_{n+1} = x_n - J^{-1}(x_n) r(x_n)$
- **Thread-scalable pattern: Scatter-Atomic-Add or Gather-Sum ?**
- **Scatter-Atomic-Add**
  - \+ Simpler
  - \+ Less memory
  - – Slower HW atomic
- **Gather-Sum**
  - \+ Bit-wise reproducibility
- **Performance win?**
  - Scatter-atomic-add
  - ~equal Xeon PHI
  - 40% faster Kepler GPU
- ✓ **Pattern chosen**
  - Feedback to HW vendors: performant atomics





Thread-Scalable Sparse Matrix Construction

- **MiniFENL: Construct sparse matrix graph from FEM connectivity**
- **Thread scalable algorithm for constructing a data structure**
  1. Parallel-for : fill **Kokkos lock-free unordered map** with FEM node-node pairs
  2. Parallel-scan : sparse matrix rows' column counts into row offsets
  3. Parallel-for : query unordered map to fill sparse matrix column-index array
  4. Parallel-for : sort rows' column-index subarray

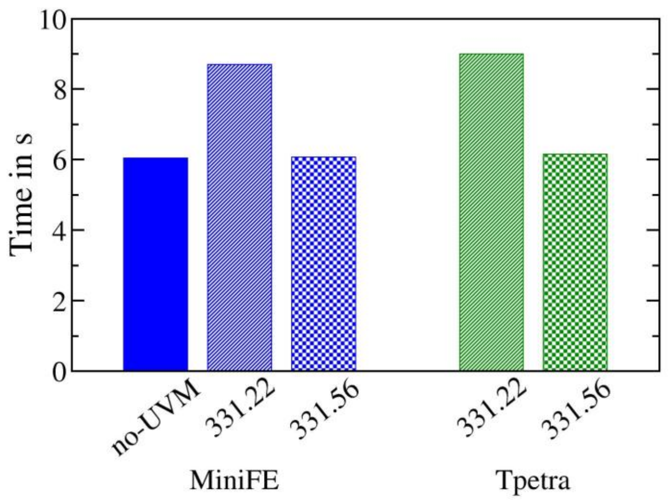


- **Pattern and tools generally applicable to construction and dynamic modification of data structures**

Tpetra: Domain Specific Library Layer for Sparse Linear Algebra Solvers

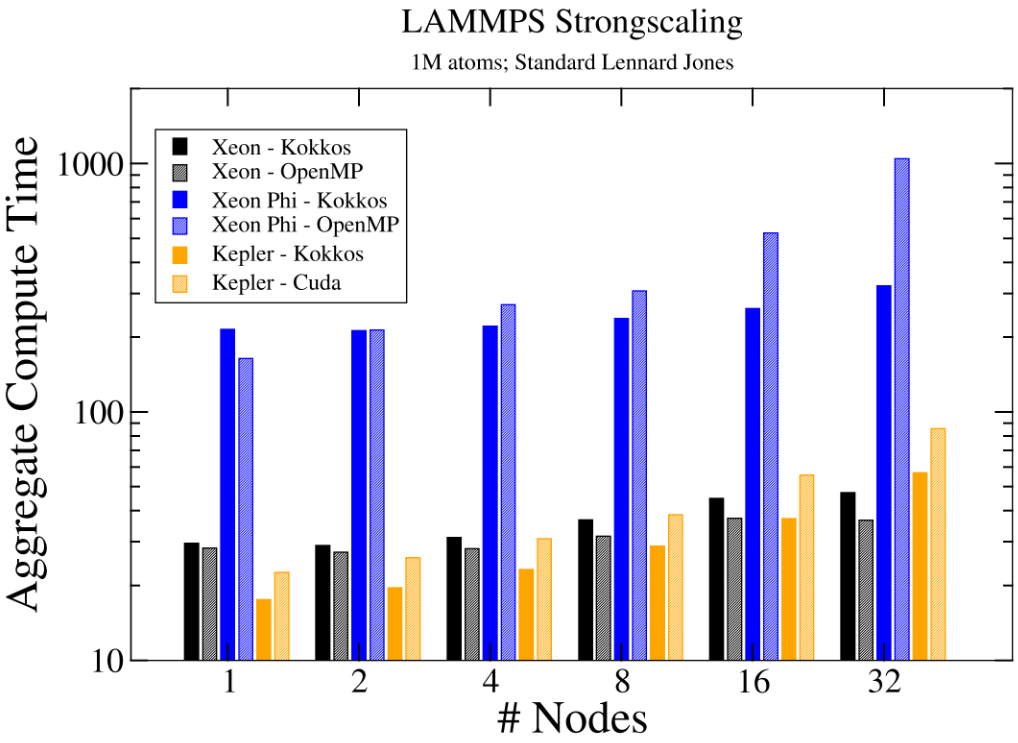
Sandia National Laboratories

- **Funded by ASC/Algorithms and ASCR/EASI**
- **Tpetra: Sandia's templated C++ library for sparse linear algebra**
  - **Templated on "scalar" type: float, double, automatic derivatives, UQ, ...**
  - **Incremental refactoring from pure-MPI to MPI+Kokkos**
- **CUDA UVM (unified virtual memory) codesign success**
  - **Sandia's early access to CUDA 6.0 via Sandia/NVIDIA collaboration**
  - **Allows CPU to directly access GPU memory, details hidden by Kokkos API**
  - **Enables incremental refactoring and testing**
- **Early access to UVM a win-win**
  - **Expedited refactoring + early evaluation**
  - **Identified performance issue in driver**
  - **NVIDIA fixed before their release**



LAMMPS (molecular dynamics application) Porting to Kokkos has begun

- **Funded by LAMMPS' projects**
- **Enable thread scalability throughout code**
  - Replace redundant hardware-specialized manycore parallel packages
- **Current release has optional use of Kokkos**
  - Data and device management
  - Some simple simulations can now run entirely on device
- **Performs as well or better than original hardware-specialized packages**



Recent and In-Progress Enhancements to Abstractions and API:
Spaces, Policies, Defaults, and C++11

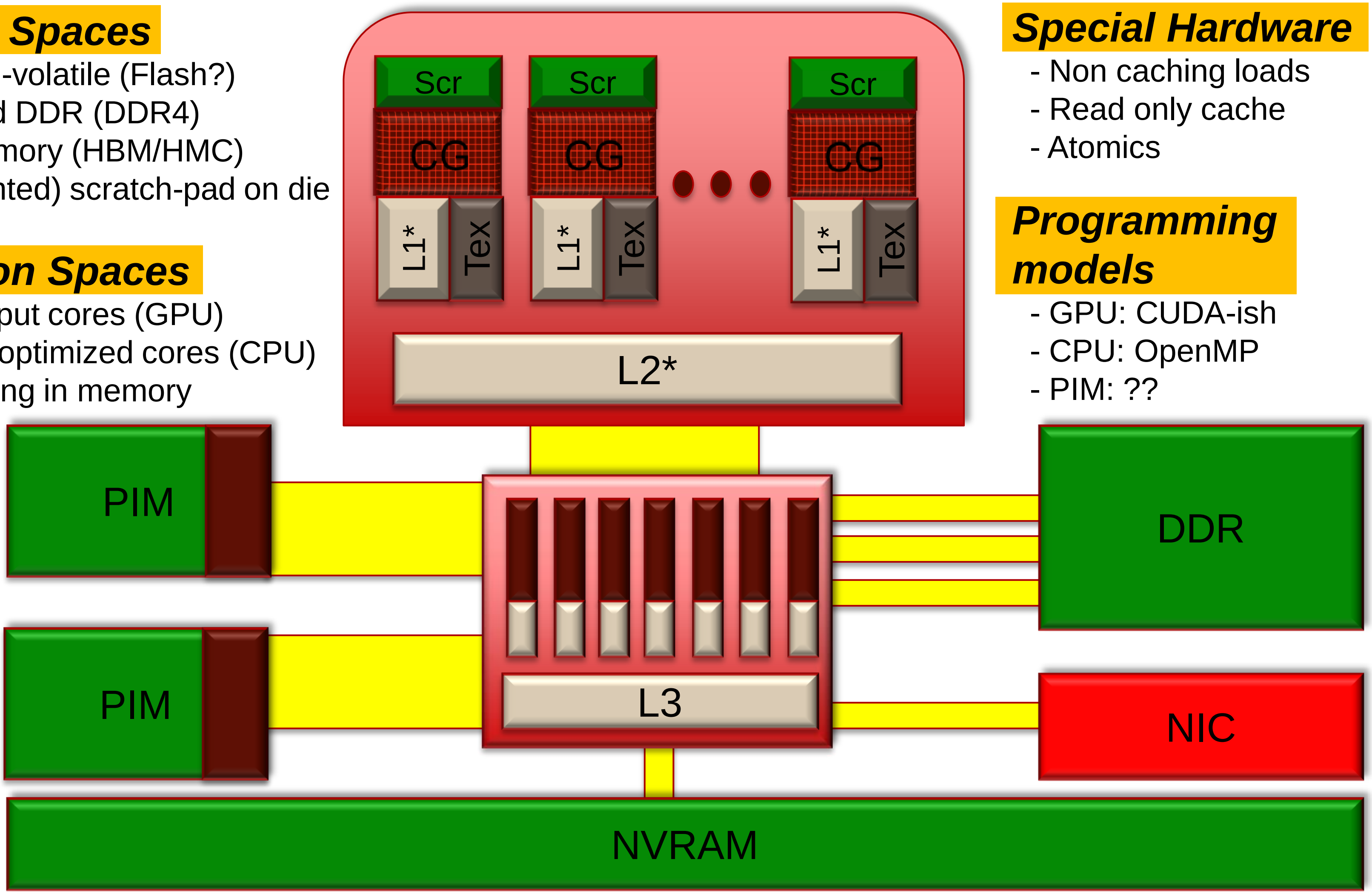
Complex Heterogeneous Architectures, Abstractions to prepare us for this future...

Memory Spaces
- Bulk non-volatile (Flash?)
- Standard DDR (DDR4)
- Fast memory (HBM/HMC)
- (Segmented) scratch-pad on die

Execution Spaces
- Throughput cores (GPU)
- Latency optimized cores (CPU)
- Processing in memory

Special Hardware
- Non caching loads
- Read only cache
- Atomics

Programming models
- GPU: CUDA-ish
- CPU: OpenMP
- PIM: ??



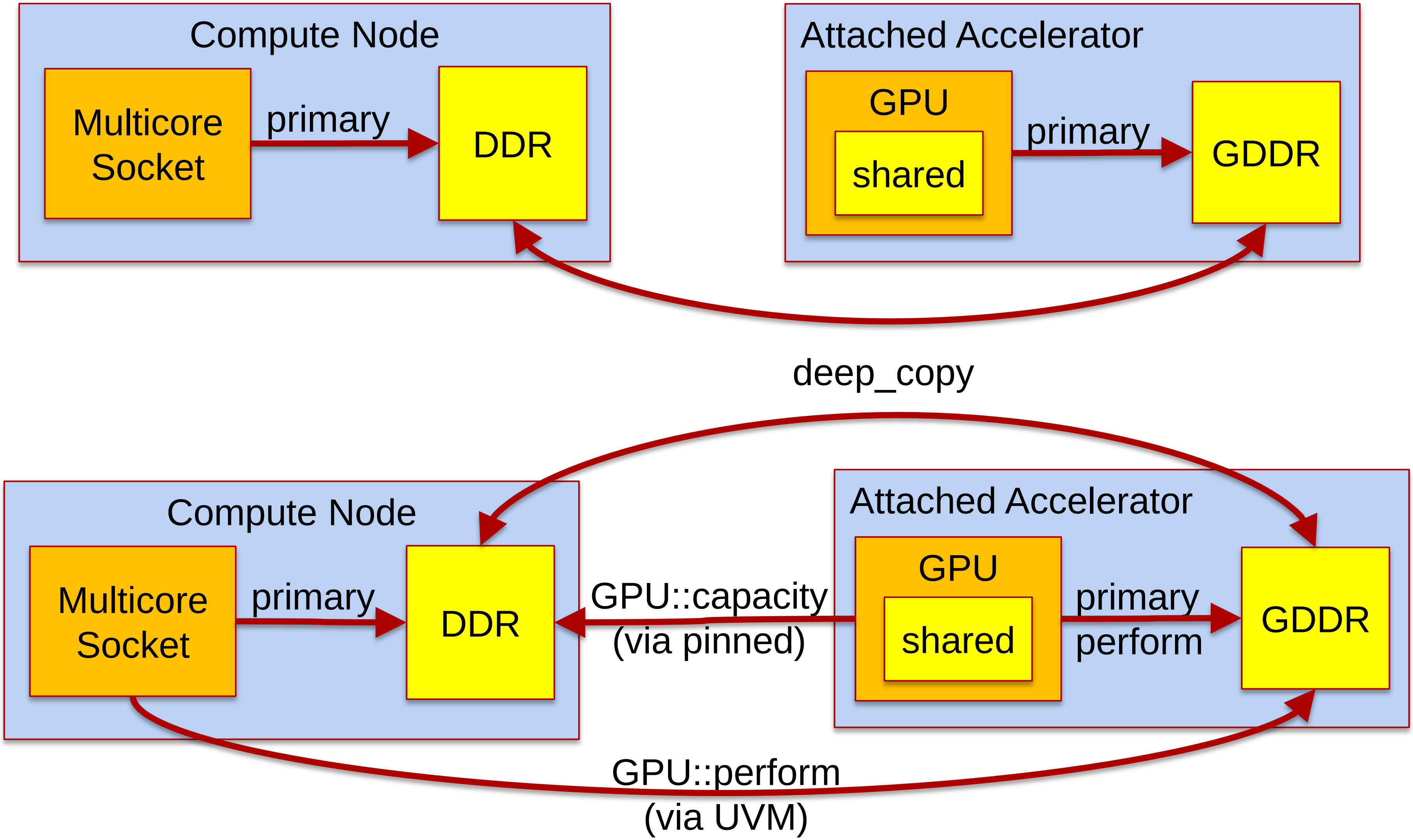
Execution Space(s)

- Execution Space *Instance*
  - Hardware execution resources (e.g., cores, hyperthreads)
  - Expect functions to execute concurrently on those resources
  - Degree of potential concurrency (cores, hyperthreads) determined at runtime
  - Number of execution space instances determined at runtime
- Execution Space *Type* (CPU, Xeon Phi, CUDA)
  - Functions compiled to execute on a type of execution space
  - These types determined at configure/compile time
- Host Space
  - The main process and its functions execute in the Host Space
  - One type, one instance, and is serial (potential concurrency == 1)
- Execution Space *Default* : one instance of one type
  - Configure/build with one type – it is the default
  - Initialize with one instance – it is the default

Memory Spaces

- Memory Space *Types* (GDDR, DDR, NVRAM, Scratchpad)
  - The *type* of memory is defined with respect to an execution space type
  - Primary: (default) space with allocable memory (e.g., can malloc/free)
    - Performant : best performing space (e.g., GDDR)
    - Capacity : largest capacity space (e.g., DDR)
    - Contemporary system: Primary == Performant == Capacity
  - Scratch : non-allocable *and* maximum performance
  - Persistent : usage can persist between process executions (e.g., NVRAM)
- Memory Space *Instance*
  - Has relationship with execution space instances (more later)
  - Directly addressable by functions in that execution space
  - Contiguous range of addresses
- Memory Space *Default*
  - Default execution spaces' primary memory space

Examples of Execution and Memory Spaces



Execution / Memory Space Relationships

- (Execution Space , Memory Space , Memory Access Traits)
  - Accessibility : functions can/cannot access memory space
    - E.g., Host functions can never access GPU scratch memory
    - E.g., GPU functions can access Host capacity memory only if it is pinned
    - E.g., Host functions can access GPU performant memory only if it is UVM
  - Readable / Writeable
    - E.g., GPU performant memory using texture cache is read-only
  - Bandwidth : potential rate at which concurrent instructions can read or write
  - Capacity for views to (allocable) data
- Memory Access Traits (extension point) potential examples:
  - read-only, write-only, volatile/atomic, random, streaming, ...
  - Converting between "views" with same space and different traits
  - ➢ Default is simple readable/writeable – no special traits
- Future opportunity
  - Execution space access to remote memory space (similar to MPI 1-sided)

Views and Defaults (API update in-progress)

- typedef View< ArrayType , Layout , Space , Traits > view_type ;
  - Omit Traits : no special compile-time defined access traits
  - Omit Space : default execution space's default memory space
  - Omit Layout : allocable memory space's default layout
  - default everything: View< ArrayType >
- ArrayType, by example: View< double**[3][8] >
  - Four dimensional array of 'double' : [N][M][3][8]
  - N and M are runtime defined dimensions
- view_type a(optional_traits , N0 , N1 , ...);
  - optional_traits : a collection of optional runtime defined traits
  - label trait : string used in error and warning messages, default is none
  - initialize trait, default is parallel in-place construction of each member
  - reference counting trait, default is reference count

Allocation Semantics (API update in-progress)

- View<double**[3][8], Space> a(N,M);
  - Allocate ‘double[N][M][3][8]’ memory in ‘Space’
  - Layout will vary with ’Space’ or ‘Layout’ template argument
  - Dimensions may be padded for alignment
  - a(i,j,k,l) : access data via multi-index
  - Optional array bounds checking for debugging
- View semantics (hidden reference counting)
  - View<double**[3][8],Space> b = a ; // SHALLOW copy
  - Both ‘b’ and ‘a’ reference the same allocated memory
  - Memory deallocated when last referencing view is destroyed
- ‘Const-ness’ of views and viewed arrays
  - View<<u>const</u> double **[3][8],Space> c = a ; // OK, view to const array
  - const View<double**[3][8],Space> d = c ; // ERROR, non-const view of const

Deep Copy and "Mirror" Semantics

- **deep_copy(destination_view , source_view);**
  - **Copy allocated array of 'source_view' to allocated array of 'destination_view'**
  - **Kokkos policy: never hide an expensive deep copy operation**
  - **Only deep copy when explicitly instructed by the user**
- **Avoid expensive permutation of data due to different layouts**
  - **Mirror the layout in Host memory space**

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

- **Avoid unnecessary deep-copy**

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

  - **If Space is Host memory *or* if Host can access Space (e.g., CUDA UVM)**
  - **Then 'a_h' is simply a view of 'a' and deep_copy is a no-op**

Subview : View of a sub-array

SrcViewType src_view(...);

DstViewType dst_view = subview<DstViewType>(src_view, ...*args*)

- ***...args*** : list of indices or ranges of indices

- **Challenging capability due to polymorphic array Layout**
  - View's are strongly typed: View<ArrayType,Layout,Traits>
  - Compatibility constraint among DstViewType, SrcViewType, ...*args*
    - number of dimensions (rank of array)
    - runtime / compile-time dimensions
    - destination layout can accommodate when stride != dimension
    - 'const-ness' and other memory access traits
  - Performance of deep_copy between subviews
- **Using C++11 'auto' type would help address this challenge**
  - auto dst_view = subview(src_view , ...*args*);
  - Let implementation choose a compatible view type
  - Caution: user will not have a priori knowledge of this type

Execution Policy (API update in progress)

- <u>How</u> Potentially Concurrent Functions are Executed
  - Where : in what execution space (type and instance)
  - Parallel Work: current capabilities [0..N) or (#teams, #thread/team)
  - Scheduling : currently static scheduling of data parallel work
  - Map work function calls onto resources of the execution space
    - E.g., contiguous spans of [0..N) to a CPU thread for contiguous access pattern
    - E.g., strided subsets of [0..N) to GPU threads for coalesced access pattern

- Compose Pattern & Policy; e.g., parallel_for(policy , functor);
  - Call functor in parallel according to policy
  - Functor can be a C++11 lambda

```
parallel_for( N , [=]( int i ) { /* lambda-function body */ } );
```

  - Call functor 'N' times in parallel with i = 0 .. N-1
  - Default: N → RangePolicy<DefaultExecutionSpace>(0,N)

Execution Policies, Patterns, and Defaults

- **Patterns: parallel_for, parallel_reduce, parallel_scan**

- **parallel_*pattern*(policy , functor);**
  - **Call functor::operator()(work , ...other_args...)**
  - **Call on policy's execution space according to policy's scheduling**
  - **functor argument and API requirements defined by pattern and policy**

- **parallel_reduce functor API requirements and defaults**
  - **functor::init(value_type & update) const ; // new(& update) value_type();**
  - **functor::join(volatile value_type & update ,**
    volatile const value_type & in) const ; // update += in ;
  - **functor::final(value_type & update) const ; //**

- **parallel_scan functor has similar requirements and defaults**

Defaults enable C++11 Lambda for Functors

- **Dot product becomes simple with C++11 lambda with defaults**

```
double dot( View<double*> x , View<double*> y ) {
  double d = 0 ;
  parallel_reduce( x.dimension_0() , [=](int i, double & v) { v += x(i) * y(i); } , d );
  return d ;
}
```

- **Parallel reduce and scan defaults**
  - Reduction type: deduced from lambda's argument list
  - Initialize: default constructor
  - Join: operator +=

- **Expect Cuda / nvcc version 7 to support C++11 lambda**

- **Anecdote: our experienced developers prefer functors**

Execution Policy – an extension point

- **Policy calls functor's work function in parallel**
  - PolicyType<ExecSpace>::member_type // data parallel work item
    void Func::operator()(PolicyType<...>::member_type) const ;
- **Range policy (existing)**
  - parallel_for(RangePolicy<ExecSpace>(0,N) , functor);
    void Func::operator()(integer_type i) const ;
- **Thread team policy (existing)**
  - parallel_for(TeamPolicy<ExecSpace>(#teams,thread/team) , functor);
    void Func::operator()(TeamPolicy<ExecSpace>::member_type team) const ;
- **Extension point for new policies**
  - Multi-indices [0..M)x[0..N)
  - Dynamic scheduling / work stealing

Execution Policy for Functor with multiple 'operator()(...)'

- **Allow a functor to have multiple parallel work functions**
  - typedef PolicyType< ExecSpace , TagType > policy ;
  - parallel_*pattern*(policy(...) , functor);

  void FunctorType::operator()(const TagType &, policy::member_type) const ;
  - Parallel work functions differentiated by 'TagType'
    - TagType used instead of class' method name

- **Motivations**
  - Algorithm (class) with multiple parallel passes using the same data
  - Operators can share member data and member functions
  - Common need in LAMMPS
    - allow LAMMPS to remove clunky "wrapper functor" pattern

In-Progress Task/Data Parallelism
Kokkos/Qthreads LDRD
Abstractions and API

Execution Policy for Task Parallelism

- **TaskManager< ExecSpace > execution policy**
  - Policy object shared by potentially concurrent tasks

    ```
    TaskManager<...> tm( exec_space , ... );
    Future<> fa = spawn( tm , task_functor_a ); // single-thread task
    Future<> fb = spawn( tm , task_functor_b );
    ```

  - Tasks may be data parallel

    ```
    Future<> fc = spawn_for( tm.range(0..N) , functor_c );
    Future<value_type> fd = spawn_reduce( tm.team(N,M) , functor_d );
    wait( tm ); // wait for all tasks to complete
    ```

  - Destruction of task manager object waits for concurrent tasks to complete
- **Task Managers**
  - Define a scope for a collection of potentially concurrent tasks
  - Have configuration options for task management and scheduling
  - Manage resources for scheduling queue

Execution Policy for Task Parallelism

- Tasks’ execution dependences
  - Start a task only after other specified tasks have completed

    ```
    Future<> array_of_dep[ M ] = { /* future for other specified tasks */ };
    ```
  - Single threaded task:

    ```
    Future<> fx = spawn( tm.depend(M,array_of_dep) , task_functor_x );
    ```
  - Data parallel task:

    ```
    spawn_for( tm.depend(M,array_of_dep).range(0..N) , task_functor_y );
    ```
  - Tasks and dependences define a directed acyclic graph (dag)
- Challenge: A GPU task cannot ‘wait’ on dependences
  - An executing GPU task cannot be suspended – waiting blocks a processor
  - A parent task may spawn child tasks but cannot complete until child tasks have completed
  - Solution: ‘respawn’ parent task with new dependences

    ```
    respawn( tm.depend(M,array_of_child), parent );
    return ; // immediately return to be run after children have completed
    ```

Multithreaded Graph Library (MTGL) / Kokkos

- **Discover gaps in Kokkos for supporting Graph Algorithms**
  - Strategy: Prototype a port of MTGL onto Kokkos
- **Successful port of data structures and data parallel algorithms**
  - Prototype MTGL/Kokkos is running on GPU, performance looks promising
  - Graph iteration algs on K40X 3-7x faster than 20threads on Ivybridge
- **Major gap: GPU memory too small**
  - Sufficient space for graph vertex data
  - Insufficient space for graph edge data
- **Address GPU memory size gap**
  - Option A: GPU directly access edge data via host-pinned memory
    - New Kokkos memory space, fits well with future NVLINK hardware
    - Motivated (in part) updating Kokkos abstractions
  - Option B: Stream edge data in/out of GPU buffers
    - Might perform better now, more complex, consumes GPU memory

Embedded UQ on Manycore

Stokhos/Kokkos LDRD

Equinox ASCR project

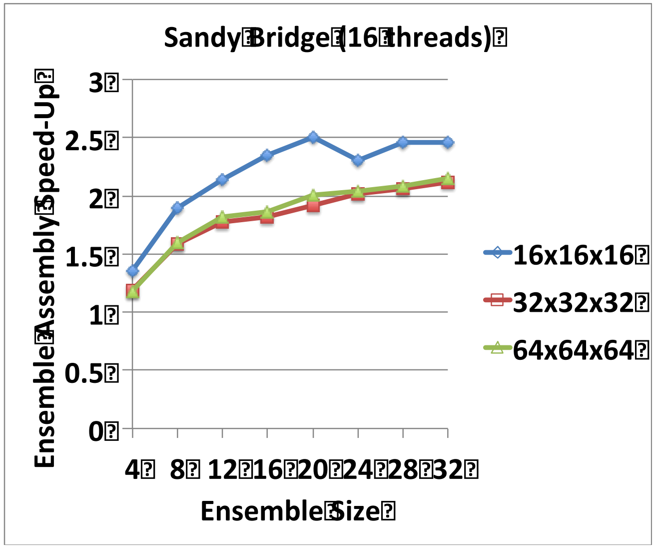
Premise: Embedding UQ Increases Computational Intensity

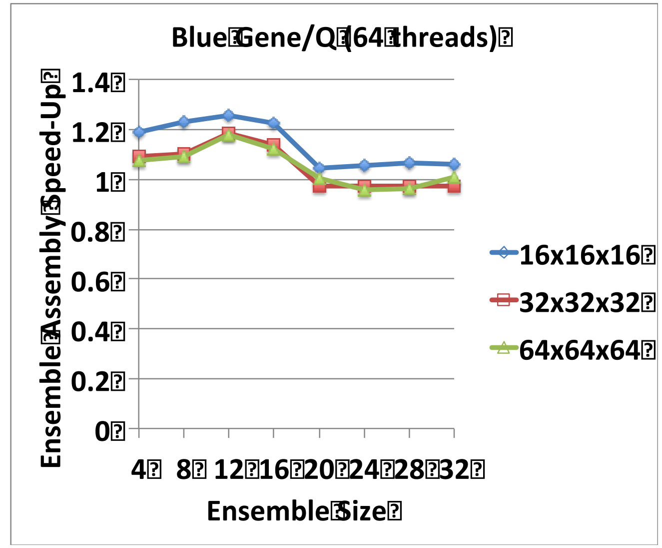
- **Computations' "Scalar" type becomes a vector quantity**
  - **Coefficients of a polynomial chaos expansion (PCE)**
  - **Sampling ensemble**
  - **Scalar math operations replaced by vector or tensor operations**
- **Data parallel vector and tensor operations performant on GPU**
  - **Vector units (i.e., GPU warps)**
  - **Indirection (e.g., sparse mat-vec) lookups yield vector instead of scalar**
- **Communicate vectors instead of scalars**
  - **Larger messages for halo exchanges vs. more halo exchanges**
  - **Fewer messages, reduced latency cost**
- **Challenge: Embedding UQ "scalar" type**

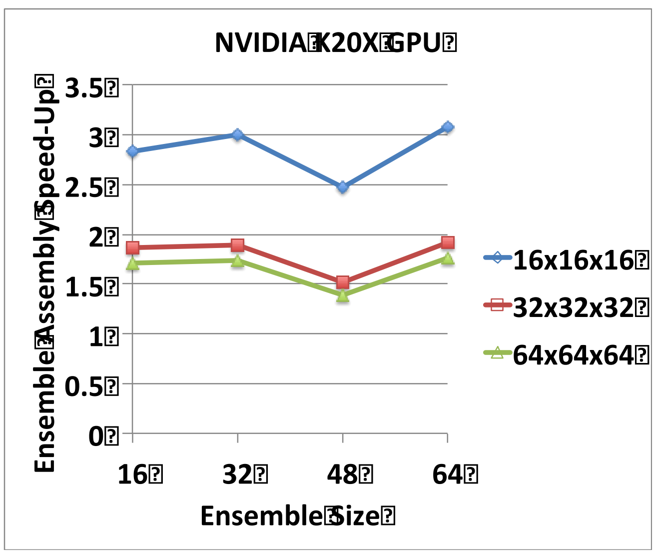
Challenge: Embedding UQ “Scalar” Type

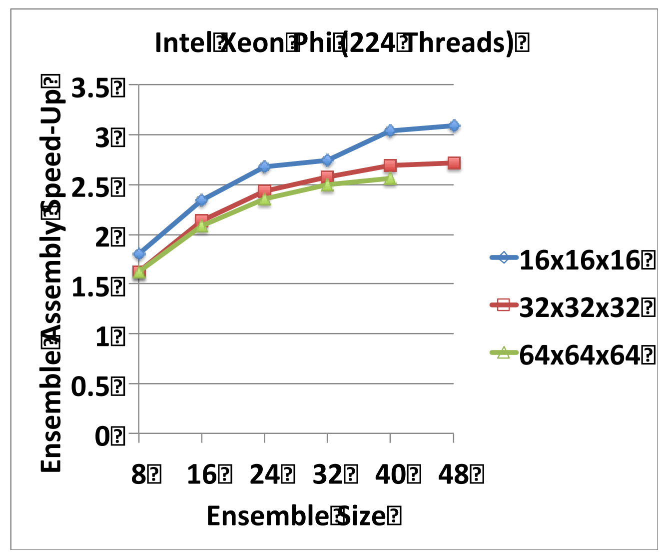
- **Allocating each individual “Scalar” type kills performance**
  - **Many small allocations & deallocations**
  - **Non-contiguous memory**
- **Leverage Kokkos View mechanism**
  - **Change “View< double * >” to “View< *UQScalar*<double> * >”**
  - ***UQScalar* vector length is an additional dimension of the array**
  - **Array layout map keeps *UQScalar*’s values contiguous**
- **Prototyped in FENL Mini-application**
  - **Trilinos/packages/trilinoscouplings/examples/fenl**
  - **Hybrid parallel : MPI + Kokkos**
  - **PDE Assembly to sparse linear system**
  - **Belos/MueLU/Tpetra to solve sparse linear system**

UQ Ensemble Assembly Speedups

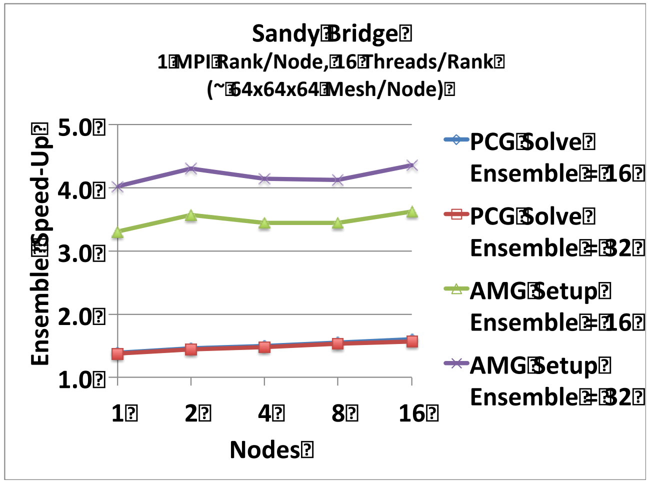


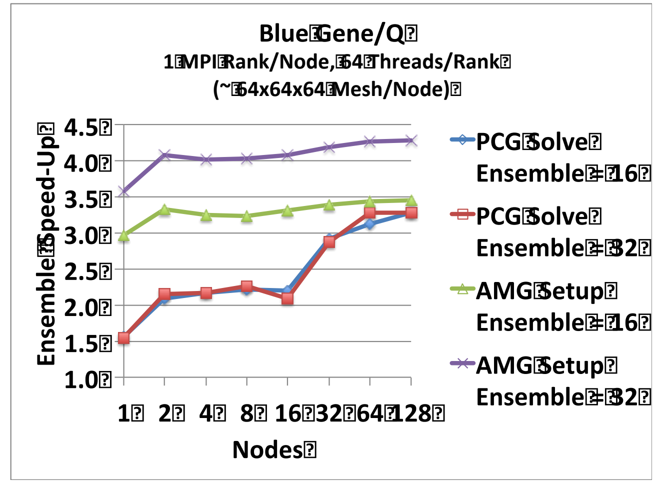


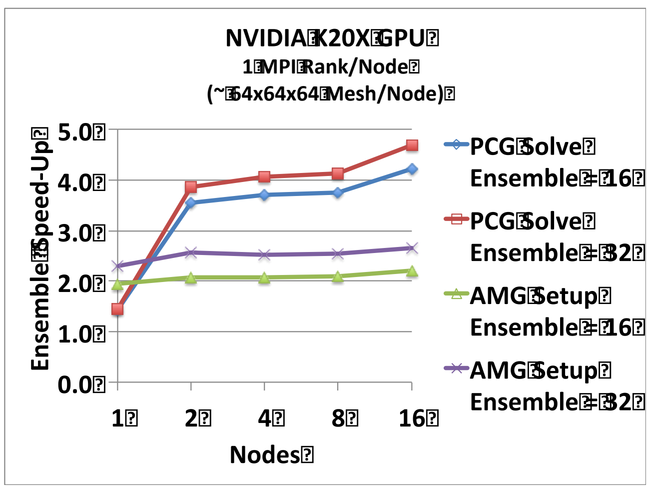


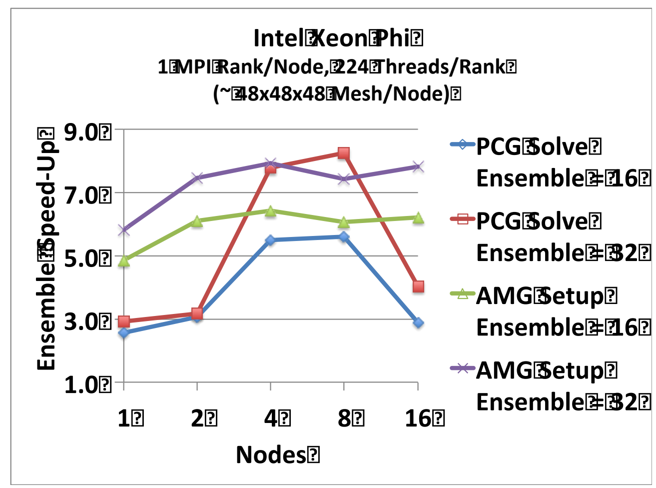


UQ Ensemble CG/MueLu Solver Speedups









Several ensemble AMG setup, solve kernels have not yet been optimized for GPU!

Vision for Migrating to MPI+X future

- Kokkos evolves from "pure research" to "production growth"
  - Recent usability review by "alpha" users for recommended improvements
  - Core abstractions and API stabilizes, as per today's presentation
- Tutorial Examples and Mini-Applications using Kokkos
  - How to use Kokkos via examples
  - How to design and implement thread-scalable algorithms via mini-apps
- SON Website: software.sandia.gov/drupal/kokkos
- Tpetra and LAMMPS are migrating
- Long Term Strategy: C++17 or C++21 instead of Kokkos
  - ISO C++ Committee working to incorporate threaded parallelism in standard
  - I am a voting member on this committee (several week-long mtgs/year)
  - Steer Kokkos and influence C++ standard → convergence

Recent Publication

Kokkos: Enabling manycore performance portability through polymorphic memory access patterns, Journal of Parallel and Distributed Computing, July 2014

http://dx.doi.org/10.1016/j.jpdc.2014.07.003