SAND2014-3216C

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

Sandia National Laboratories

Exceptional service in the national interest

Kokkos: Enabling Performance Portability Across Manycore Architectures

H. Carter Edwards, Christian Trott, Daniel Sunderland

Sandia National Laboratories

PADAL Workshop

April 28-29, 2014 | Lugano, Switzerland

SAND2014-####C (Unlimited Release)

U.S. DEPARTMENT OF ENERGY

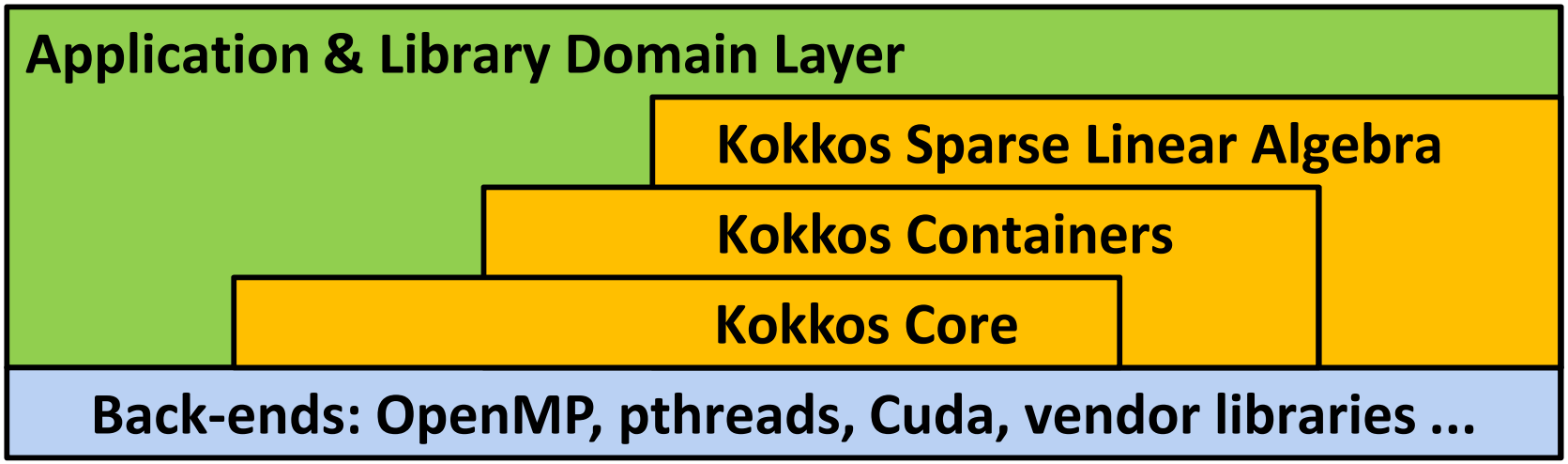
NNSA National Nuclear Security Administration

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

Kokkos: A Layered Collection of Libraries

Sandia National Laboratories

- Standard C++, Not a language extension
  - In *spirit* of TBB, Thrust & CUSP, C++AMP,...
  - *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)
  - Prefer C++2011 for its concise lambda syntax
    - As soon as vendors catch 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 ?

Performance Portability Answer

Sandia National Laboratories

- **Thread parallel computation (for, reduce, scan)**
  - Dispatched to an execution space (CPU, GPU, Xeon Phi)
  - Operates on data in memory spaces (CPU, GPU, CPU-pinned, GPU-UVM, …)
  - ➢ **Should use device-specific memory access pattern; how to portably?**
- **Multidimensional Arrays, *with a twist***
  - Layout mapping: 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 threads in one or more execution spaces
  - Polymorphic multidimensional array layout
  - ➢ **Control dispatch ○ layout → control memory access pattern**
  - Utilization of special hardware; e.g., GPU texture cache

Multidimensional Array
Allocation, Access, and Layout

Sandia National Laboratories

- **Allocate and access multidimensional arrays**

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

  - Dimension [N][M][3][8] ; two runtime, two compile-time
  - **a(i,j,k,l)** : access data via multi-index with device-specific map
  - Index map inserted at compile-time (C++ template meta programming)
- **Identical C++ 'View' objects used in host and device code**
- **Assertions that 'a(i,j,k,l)' access is correct**
  - Compile-time:
    - Execution space can access memory space (instead of runtime segfault)
    - Array rank == multi-index rank
  - Runtime (debug mode)
    - Array bounds checking
    - Uses Cuda 'assert' mechanism on GPU

Multidimensional Array
Layout and Access Attributes

Sandia National Laboratories

- **Override device's default array layout**

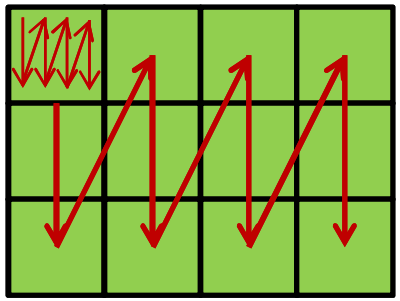
```
class View<double**[3][8], Layout , Device> a("a",N,M);
```

  - E.g., force row-major or column-major
  - ➢ Multi-index access is unchanged in user code
  - *Layout* is an extension point for blocking, tiling, etc.

- **Example: Tiled layout**

```
class View<double**, TileLeft<8,8> , Device> b("b",N,M);
```

  - ➢ Layout changes are transparent to user code
  - ➢ IF the user code honors the a(i,j,k,...) API



- **Data access attributes – user's intent**

```
class View<const double**[3][8], Device, RandomRead> x = a ;
```

  - Constant + RandomRead + GPU → read through GPU texture cache
  - Transparent to user code

Kokkos Core: Deep Copy Array Data

Sandia National Laboratories

NEVER have a hidden, expensive deep-copy

- Only deep-copy when explicitly instructed by user code
- Avoid expensive permutation of data due to different layouts
  - Mirror the layout in Host memory space

```
typedef class View<...,Device> 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

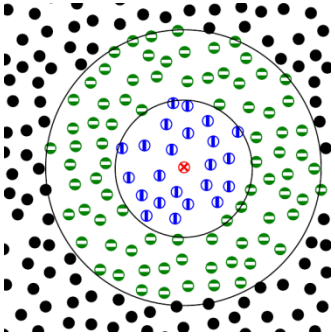
```
MyViewType::HostMirror a_h = create_mirror_view( a );
```

  - If Device uses host memory *or* if Host can access Device memory space (CUDA unified virtual memory)
  - Then 'a_h' is simply a view of 'a' and deep_copy is a no-op

Evaluate Performance Impact of Array Layout

- **Molecular dynamics computational kernel in miniMD**
- **Simple Lennard Jones force model:** $F_i = \sum_{j, r_{ij} < r_{cut}} 6\varepsilon \left[\left(\frac{\varsigma}{r_{ij}} \right)^{7} - 2 \left(\frac{\varsigma}{r_{ij}} \right)^{13} \right]$
- **Use atom neighbor list to avoid N^2 computations**

```
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;
```

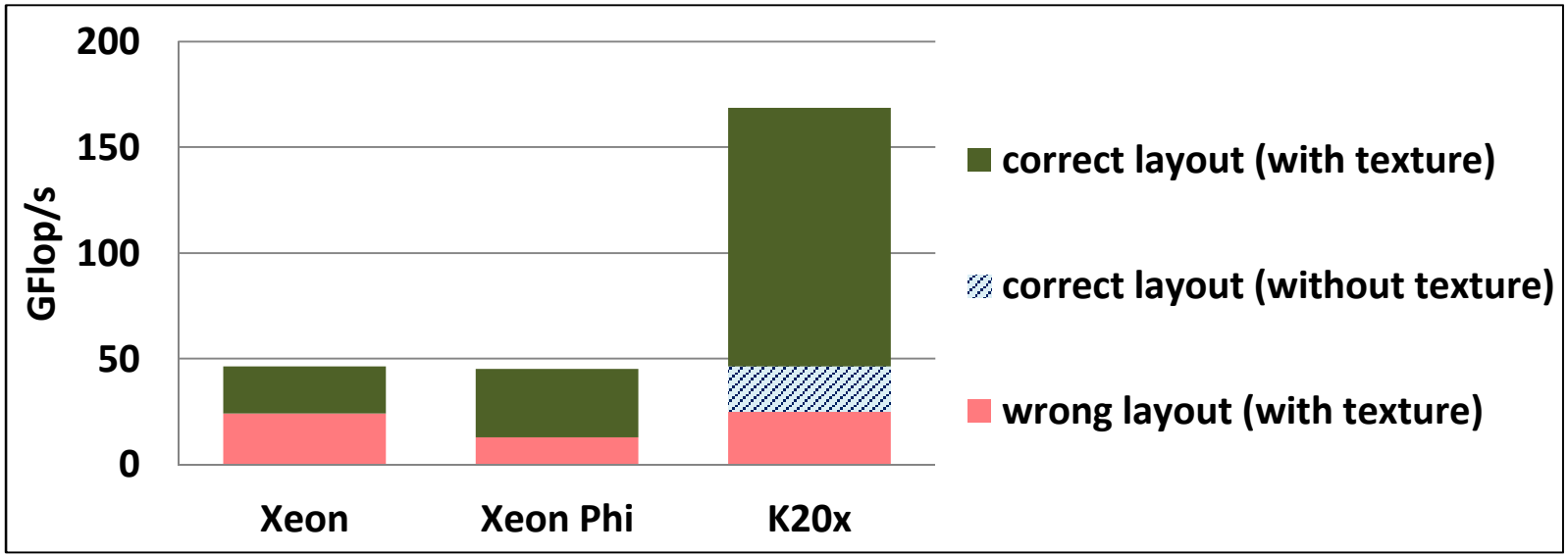


- **Moderately compute bound computational kernel**

Evaluate Performance Impact of Array Layout

Sandia National Laboratories

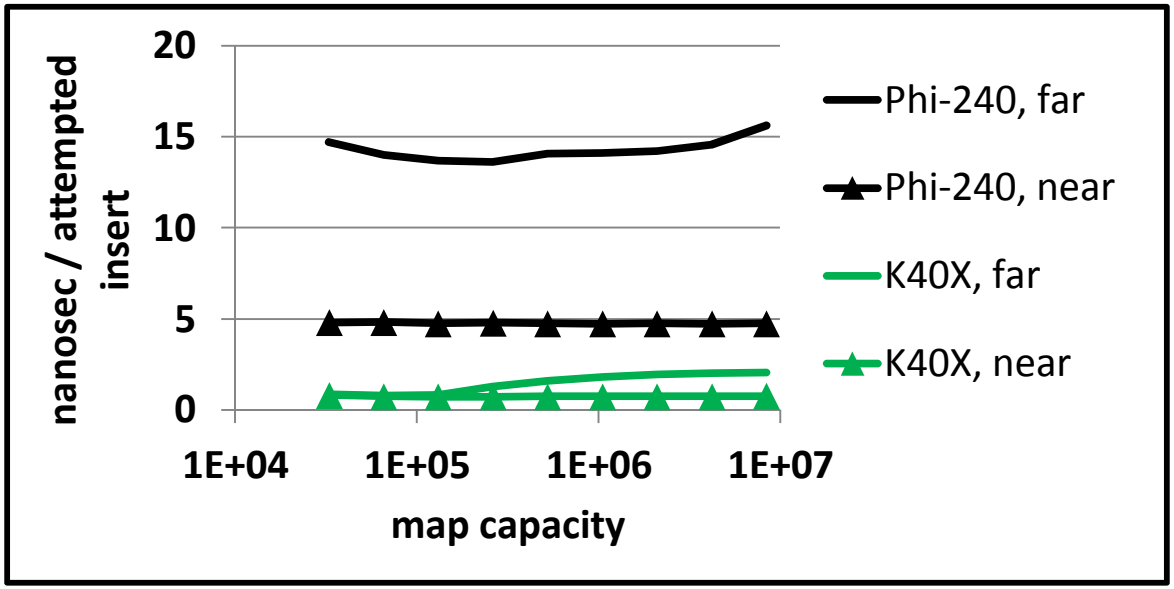
- Test Problem (#Atoms = 864k, ~77 neighbors/atom)
  - Neighbor list array with correct vs. wrong layout
    - CPU and GPU have different layouts
  - Random read of neighbor coordinate via GPU texture fetch



- Large loss in performance with (forced) wrong layout
  - Even when using GPU texture fetch
  - Kokkos, by default, selects the correct layout

Lock-Free Unordered Map

- **Essential building block for algorithms modifying dynamic data structures: graph construction, mesh adaptivity, ...**
- **State-of-practice: non-scalable lock-based implementations**
- **Performance evaluation stress tests**
  - **Parallel insert to 88% full with 16x redundant inserts (near/far threads)**
  - **NVidia Kepler K40X vs. Intel Xeon Phi COES2**

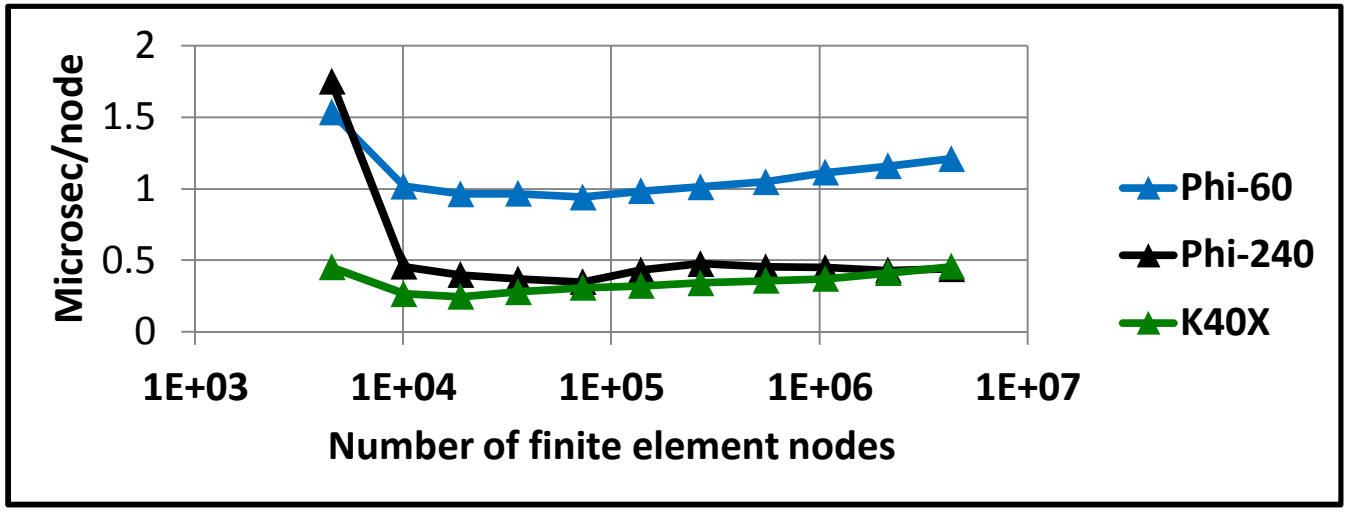


- **K40X dramatically better performance**
- **Xeon Phi implementation optimized using explicit non-caching prefetch**
- **Theory: due to cache coherency protocols and atomics' performance**

Thread Scalable Sparse Matrix Construction

Sandia National Laboratories

- **First time we could move graph construction to manycore**
- **Thread scalable algorithm with dynamic data structure**
  1. Parallel-for to fill unordered map with finite elements' node-node pairs
  2. Parallel-scan sparse matrix rows' column counts
  3. Parallel-for over unordered map to fill sparse matrix column-index array
  4. Parallel-for to sort rows' column-index subarray



- **Matrix graph construction 2x-3x longer than one Element+Fill**
  - Linearized hexahedron finite element for: $-k\,\Delta T + T^2 = 0$
  - 3D spatial Jacobian with 2x2x2 point numerical integration

Conclusion

Sandia National Laboratories

- Kokkos: layered collection of libraries
- Performance portability to CPU, GPU, Xeon Phi
- Parallel dispatch (for, reduce, scan)
- Multidimensional arrays with polymorphic layout
- Dispatch ○ polymorphic layout → memory access pattern
- AoS versus SoA solved with appropriate abstractions
- UnorderedMap with thread scalable insertion