Kokkos: Enabling Performance Portability of C++ Applications and Libraries across Manycore Architectures

NECDC / October 22, 2014

SAND2014-18527C (Unlimited Release)
Increasingly Complex Manycore Architectures

¿ 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: ??

PIM
PIM

L1* Scr
L1* Scr
L1* Scr

CG
CG
CG

L2*

Tex
Tex
Tex

L1* Tex
L1* Tex
L1* Tex

L3

NIC

DDR

PIM
PIM

NVRAM
Vision for Heterogeneous Parallelism

- “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, ...) have diverse performance requirements
  - Memory spaces have diverse capabilities and performance characteristics
  - Vendors have diverse programming models for optimal utilization of their hardware

- 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
  - 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

- Applications and Domain Libraries written in Standard C++
  - *Not* a language extension like OpenMP, OpenACC, OpenCL, CUDA, ...
  - Require C++1998 standard (supported everywhere except IBM’s xIC)
  - Prefer C++2011 for its concise lambda syntax (LLNL’s RAJA requires this)
    - As soon as vendors catch up to C++2011 language compliance

- Kokkos implemented with C++ template meta-programming
  - In *spirit* of TBB, Thrust & CUSP, C++AMP, LLNL’s RAJA, ...

---

Application and Domain Specific Library Layer

- Kokkos Sparse Linear Algebra
- Kokkos Containers
- Kokkos Core

Back-ends: Cuda, OpenMP, pthreads, vendor libraries ...
Performance Portability Challenge:

Best (good) performance requires computations to implement architecture-specific memory access patterns

- CPUs (and Xeon Phi)
  - Core-data affinity: consistent NUMA access (first touch)
  - Array alignment for cache-lines and vector units
  - Hyperthreads’ cooperative use of L1 cache

- GPUs
  - Thread-data affinity: coalesced access with cache-line alignment
  - Temporal locality and special hardware (texture cache)

- Array of Structures (AoS) vs. Structure of Arrays (SoA) dilemma
  - This has been the wrong concern

Ask the right question: Abstractions for Performance Portability?
Kokkos maps users’ parallel computations to threads
- Standard parallel programming model pattern; e.g., parallel-for
- Users implement C++ functions or lambdas for their parallel loop bodies
- Kokkos calls user’s code from architecture’s "hardware" threads

Kokkos’ multidimensional array data structure has a twist
- Layout mapping: multi-index \((i,j,k,...) \leftrightarrow \text{memory location}\)
- Kokkos chooses layout for architecture-specific memory access pattern
- Layout changes are invisible to user code
  - IF user code honors Kokkos’ simple array API: \(a(i,j,k,...)\)
  - “Polymorphic multidimensional array layout”

Bonus: utilize special hardware invisibly to users’ code
- GPU texture cache to speed up read-only random access patterns
- Atomic operations for thread safety
Kokkos-Core’s Multidimensional Array API

- **View<double**[3][8]** , Space > a("a",N,M);**
  - User allocates array data in “Space” with dimensions [N][M][3][8]
    - Each * indicates a runtime supplied dimension
  - Kokkos chooses layout appropriate for “Space”

- **a(i,j,k,l) : User’s access to array data**
  - Bounds checking of indices when debugging
  - “Space” accessibility enforced; e.g., GPU code cannot access CPU memory

- **View Semantics:** **View<double**[3][8]** ,Space> b = a ;**
  - A shallow copy: ‘a’ and ‘b’ are *pointers* to the same allocated array data
  - Reference counting: last View automatically deallocates data

- **deep_copy(destination_view , source_view );**
  - Copy array data across memory spaces
  - Never hide an expensive deep copy operation from the user
    - Only deep copy when a user calls the deep_copy function
Kokkos-Core’s Multidimensional Array API

- Advanced: `View<ArrayType,Layout,Space,Attributes>`
  - `ArrayType`: scalar type, # runtime dimensions, compile-time dimensions
  - `Layout`: user can override Kokkos’ choice for layout
  - `Attributes`: user’s access intentions

- Why manually specify `Layout`?
  - Force compatibility with legacy code while incrementally porting
  - Optimize performance with exotic layout
    - `View<double**,Tile<8,8>,Space> m("matrix",N,N);`
    - Tiling layout hidden from user code `m(i,j)`
  - A “plug in” extension point

- Access intention attributes
  - Turn off reference counting to wrap an legacy code’s array
  - Indicate random access to utilize GPU texture cache
    - `View< const double **, Cuda, RandomAccess>`
  - A “plug in” extension point
Kokkos-Core’s Parallel Execution API

- `parallel_for( N , [=]( int i ) { z(i) = x(i) + y(i); } );`  ← simplest API
  - Using C++11 lambda expression and default execution space
  - Call parallel loop body from hardware threads with i = [0..N)
  - Kokkos chooses which threads call with which value of ‘i’

- `parallel_reduce( RangePolicy<Space>((N) , loop_body_functor );`;
  - Range execution policy specifies what execution space
  - functor is a user supplied C++ class object with
    - a member function implementing the parallel loop body
    - member variables used by the parallel loop body function

- API: `ParallelPattern( ExecutionPolicy , LoopBody )`
  - `ParallelPattern`: `parallel_for, parallel_reduce, parallel_scan`
  - `ExecutionPolicy`: execution space and parallel iteration space
  - `LoopBody`: user’s C++ class object or C++11 lambda expression

- Execution policy is a “plug in” extension point
  - E.g., thread league-team policy, asynchronous task policy
Evaluate Performance Impact of Array Layout

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

```c
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 += 6e*((s/r_ij)^7 - 2*(s/r_ij)^13)
}
f(i) = f_i;
```

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

![Performance Graph]
Evaluate Performance Overhead of Abstraction

Kokkos competitive with native programming models

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

### MiniFE CG-Solve time for 200 iterations on 200^3 mesh

<table>
<thead>
<tr>
<th>Hardware</th>
<th>NVIDIA ELL</th>
<th>NVIDIA CuSparse</th>
<th>Kokkos</th>
<th>TBB</th>
<th>OpenCL</th>
<th>OpenMP</th>
<th>Cilk+(1 Socket)</th>
</tr>
</thead>
<tbody>
<tr>
<td>K20X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IvyBridge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SandyBridge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XeonPhi B0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XeonPhi C0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM Power7+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MiniFENL Proxy Application

- Solve nonlinear finite element problem via Newton iteration
  - Focus on construction and fill of sparse linear system
  - Thread safe, thread scalable, and performant algorithms
  - Evaluate thread-parallel capabilities and programming models

- Construct sparse linear system graph and coefficient arrays
  - Map finite element mesh connectivity to degree of freedom graph
  - Thread-scalable algorithm for graph construction

- Compute nonlinear residual and Jacobian
  - Thread-parallel finite element residual and Jacobian
  - Atomic-add to fill element coefficients into linear system
    - Atomic-add for thread safety, performance?

- Solve linear system for Newton iteration
Thread-Scalable Fill of Sparse Linear System

- **MiniFENL**: Newton iteration of FEM: \( x_{n+1} = x_n - J^{-1}(x_n)r(x_n) \)
- **Fill sparse matrix via 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

---

**Graphs**

![Graph showing matrix fill time for different systems](image)

- Phi-60 GatherSum
- Phi-60 ScatterAtomic
- Phi-240 GatherSum
- Phi-240 ScatterAtomic
- K40X GatherSum
- K40X ScatterAtomic

---

**Legend**

- Matrix Fill: microsec/node
- Number of finite element nodes

---

MiniFENL: Newton iteration of FEM: \( x_{n+1} = x_n - J^{-1}(x_n)r(x_n) \)
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
Porting in Progress: Trilinos

- **Trilinos**: SNL’s suite of equation solver libraries (and others)
  - Currently MPI-only parallel
  - Incremental refactoring to MPI+Kokkos parallel

- **Tpetra**: Trilinos’ core parallel sparse linear algebra library
  - Vectors, multi-vectors, sparse matrices, parallel data distribution maps
  - Fundamental operations: axpy, dot, matrix-vector multiply, ...
  - Templated on “scalar” type: float, double, automatic differentiation (AD), embedded uncertainty quantification (UQ), ...

- **Port Tpetra to MPI+Kokkos, other libraries follow**
  - On schedule to complete in Spring 2015
  - Use of NVIDIA’s unified virtual memory (UVM) expedited porting effort

- **Embedded UQ already Kokkos-enabled through LDRD**
  - Greater computational intensity leads to significant speed-ups compared to non-embedded UQ sampling algorithms
Porting in Progress: LAMMPS

- LAMMPS: molecular dynamics application
  - Fully MPI-only parallel with some (prototype) thread-parallel user packages
    - Architecture specific with redundantly implemented physics
  - Incrementally refactoring to MPI+Kokkos parallel
    - Collapse redundantly implemented physics into “core” code base
  - MPI+Kokkos performing as well or better than thread-parallel user packages
Takeaways: MPI + Kokkos for hybrid parallel

- Performance portability across disparate manycore architectures
  - Compose mappings to control data access patterns
    - parallel loop body → hardware threads
    - multidimensional array layout → space-allocated data
  - AoS versus SoA dilemma is solved

- Negligible performance overhead versus native implementation

- R&D now addressing more challenging algorithms
  - “Plug in” extension points to facilitate R&D
  - Construction and fill of sparse linear system
  - Not discussed: LDRD for hybrid task-data parallelism and graph analytics

- Transition of legacy codes in progress: Trilinos, LAMMPS

- Kokkos to be released to public via GitHub in FY15/Q2