Enabling Performance Portability across Manycore Architectures (Kokkos)

CIS ERB / June 2, 2014

SAND2014-4173P (Unlimited Release)
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
  - SNL Computing Research Center’s Kokkos (C++ library) solution
  - Engagement with ISO C++ Standard committee to influence C++17
Programmatics

- **ASC/CSSE (FY11 start):** Heterogeneous Computing project
  - Tight integration with co-design, mini-application, and testbed projects
  - Manycore (CPU, GPU, Xeon Phi, ...) performance portable “X” for MPI+X
    - Kokkos library is the “X” for fine grain data parallelism
  - 1.0-1.4 FTE split among ~2 staff + interns (FY14 @ 1.0 FTE)

- **LDRD (FY14 start):** Unified Task+Data Manycore Parallelism
  - For solver-preconditioners, finite elements, informatics, transport sweeps, ...
  - 0.9 FTE split among ~4 staff

- Internal/external interests, and resource challenge ahead
  - Trilinos, LAMMPS, SIERRA, other ASC codes (SNL, LANL, LLNL), AWE, ...
  - ISO C++ standards addressing fine grain parallelism (am a voting member)
  - Currently under-resourced for expected success
Kokkos: A Layered Collection of Libraries

- **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)
  - Prefer C++2011 for its concise lambda syntax (LLNL’s RAJA requires this)
    - 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 ?
Kokkos Performance Portability Answer

- Thread parallel computation
  - Dispatched to an execution space
  - Operates on data in memory spaces
  - Should use device-specific memory access pattern; how to portably?

- Multidimensional Arrays, *with a twist*
  - Layout mapping: multi-index \((i,j,k,...) \leftrightarrow \text{memory location}\)
  - 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 or more execution spaces
  - Polymorphic multidimensional array layout
  - Utilization of special hardware; e.g., GPU texture cache
Evaluate Performance Impact of Array Layout

- Molecular dynamics computational kernel in miniMD
- Simple Lennard Jones force model:
  \[ F_i = \sum_{j, r_{ij} < r_{cut}} 6 \varepsilon \left( \frac{\sigma}{r_{ij}} \right)^7 - 2 \left( \frac{\sigma}{r_{ij}} \right)^{13} \]
- 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 += 6*\varepsilon((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 models

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

![Graph showing MiniFE CG-Solve time for 200 iterations on 200^3 mesh]
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
  - \( \approx \) equal Xeon PHI
  - 40% faster Kepler GPU
- Pattern chosen
  - Feedback to HW vendors: performant atomics

\[ \text{Matrix Fill: microsec/node} \]

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

\[ \text{Number of finite element nodes} \]
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

- Funded by ASC/Algorithms (not funded through Kokkos)
- 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
  - Hidden in Kokkos, can neglect memory spaces and maintain correctness
  - 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

![Bar Chart](chart.png)
LAMMPS (molecular dynamics application) Porting to Kokkos has begun

- Funded by LAMMPS’ projects (not funded though Kokkos)
- Enable thread scalability throughout code
  - Replace redundant hardware-specialized manycore parallel packages
- Next release with 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
Takeaways

- Compose data parallel dispatch \(\bigcirc\) polymorphic array layout
  - Control data access pattern for performance portability
  - AoS versus SoA is solved

- Negligible performance overhead versus native implementation

- Lock-free unordered map
  - Enable scalable algorithms with dynamic data structures
  - First time for sparse matrix graph construction

- Transition of Legacy Codes (funded by those code projects)
  - Tpetra expedited with early access to CUDA UVM
  - LAMMPS can remove redundant, device-specific code