Sustainability and Performance through Kokkos: A Case Study with LAMMPS

Christian Trott, Si Hammond, Stan Moore, Tzu-Ray Shan
crtrott@sandia.gov

Center for Computing Research
Sandia National Laboratories, NM
Kokkos: *Performance, Portability and Productivity*

Multi-Core | Many-Core | APU | CPU + GPU
---|---|---|---
LAMMPS | Trilinos | Sierra | Albany

**Code:** github.com/kokkos/kokkos

**Tutorial:** github.com/kokkos/kokkos-tutorials

**GTC2016:** multiple talks + tutorial
Kokkos: *Performance, Portability and Productivity*

- A programming model implemented as a C++ library
- Abstractions for Parallel Execution and Data Management
  - Execution Pattern: What kind of operation (for-each, reduction, scan, task)
  - Execution Policy: How to execute (Range Policy, Team Policy, DAG)
  - Execution Space: Where to execute (GPU, Host Threads, PIM)
  - Memory Layout: How to map indices to storage (Column/Row Major)
  - Memory Traits: How to access the data (Random, Stream, Atomic)
  - Memory Space: Where does the data live (High Bandwidth, DDR, NV)
- Supports multiple backends: OpenMP, Pthreads, Cuda, Qthreads, Kalmar (experimental)
- Profiling Hooks are always compiled in
  - Stand alone tools + interfaces to Vtune/Nsight etc. available
Going Production

- Kokkos released on github in March 2015
  - Develop / Master branch system => merge requires application passing
  - Testing Nightly: 11 Compilers, total of 90 backend configurations, warnings as errors
  - Extensive Tutorials and Documentation > 300 slides/pages
    - [www.github.com/kokkos/kokkos](http://www.github.com/kokkos/kokkos)
    - [www.github.com/kokkos/kokkos-tutorials](http://www.github.com/kokkos/kokkos-tutorials)

- Trilinos NGP stack uses Kokkos as only backend
  - Tpetra, Belos, MueLu etc.
  - Working on threading all kernels, and support GPUs

- Sandia Sierra Mechanics and ATDM codes going to use Kokkos
  - Decided to go with Kokkos instead of OpenMP (only other realistic choice)
  - SM: FY 2016: prototyping threaded algorithms, explore code patterns
  - ATDM: primary development on GPUs now: “If GPUs work, everything else will too”
LAMMPS a general purpose MD code

- C++, MPI based open source code:
  - lammps.sandia.gov and github.com/lammps/lammps
- Modular design for easy extensibility by expert users
- Wide variety of supported particle physics:
  - Bio simulations, semi conductors, metals, granular materials
  - E.g. blood transport, strain simulations, grain flow, glass forming, self assembly of nano materials, neutron star matter
- Large flexibility in system constraints
  - Regions, walls, geometric shapes, external forces, particle injection, ...
- Scalable: simulations with up to 6 Million MPI ranks demonstrated
LAMMPS a general purpose MD code

- C++, MPI based open source code:
  - lammmps.sandia.gov and github.com/lammmps/lammps

- Modular design for easy extensibility by expert users

- Wide variety of supported particle physics:
  - Bio simulations, semi conductors, metals, granular materials
  - E.g. blood transport, strain simulations, grain flow, glass forming, self assembly of nano materials, neutron star matter

- Large flexibility in system constraints
  - Regions, walls, geometric shapes, external forces, particle injection, ...

- Scalable: simulations with up to 6 Million MPI ranks demonstrated

**Estimate: 500 Performance Critical Kernels**
LAMMPS – Getting on NGP

- Next generation platform support through packages
- GPU
  - GPU support for NVIDIA Cuda and OpenCL since 2011
  - Offloads force calculations (non-bonded, long range coulomb)
- USER-CUDA
  - GPU support for NVIDIA Cuda
  - Aims at minimizing data transfer => run everything on GPU
  - Reverse offload for long range coulomb and bonded interaction
- OMP
  - OpenMP 3 support for multi threading
  - Aimed at low thread count (2-8)
- INTEL
  - Intel Offload pragmas for Xeon Phi
  - Offloads force calculations (non-bonded, long range coulomb)
LAMMPS – Getting on NGP

- Next generation platform support through packages
- GPU
  - GPU support for NVIDIA Cuda and OpenCL since 2011
  - Offloads force calculations (non-bonded, long range coulomb)

Packages replicate existing physics modules:

**Hard to maintain.**

**Prone to inconsistencies.**

**Much more code.**

- Offloads force calculations (non-bonded, long range coulomb)
GPU Execution Modes
GPU Execution Modes

Homogenous

IO -> Init -> Comm -> Pair -> KSPACE -> Constraints -> IO

Constraints -> IO

IO -> IO
GPU Execution Modes

Homogenous

Init → Comm → Pair → KSPACE → Constraints

Reverse Offload

Init → Comm → Pair → Constraints

KSPACE

IO → Init → Comm → Pair → Constraints → Constraints → Pair → Comm → Init → IO
As can be seen, the KOKKOS package consistently outperforms the other packages on the next generation architectures for this benchmark, while only sacrificing a small fraction of performance on classical CPUs.

Another demonstration of the performance achievable is the recent development of a Kokkos variant of the ReaxFF force field, which is significantly more complex (many body, reactive, charge equilibration) than the simpler Lennard Jones model. We did investigations both on a BlueGene Q system with 512 nodes, where performance was compared to MPI only runs and an alternative implementation using native OpenMP, as well as on a standard Cray XC30 system with dual Intel Ivy Bridge CPUs and a system with NVIDIA K80 GPUs. Again the performance achieved with the Kokkos variant is as well as or better than the alternative implementations.

In summary, we believe that the strategy we have outlined will allow us to more easily maintain a smaller code base while delivering consistent good performance across the diversity of current (and hopefully future) hardware architectures. Our choice to use Kokkos has primarily been driven by the large number of kernels in LAMMPS to optimize; other programming models such as OpenMP 4.0 have a similar goal of enabling hardware independent performance portability.

Codes with fewer key kernels to optimize may well be better off following the strategy of replicating kernels in architecture specific programming models. This allows for easier hardware specific tailoring of individual kernels to achieve maximal performance.


Homogeneous – Reax Manybody

ReaxFF - CPU/GPU
464k atoms

Performance (atomsteps/sec)

# of Nodes/Devices

- NVIDIA K80
- NVIDIA K20
- Intel Haswell (32c/64t)
Reverse Offload – Using Asynchronous DeepCopy

- `deep_copy(ExecutionSpace(), src, dst)`
  - Guaranteed synchronous with respect to ExecutionSpace
  - Reality: requires DMA engine, works between CudaHostPinnedSpace and CudaSpace

```cpp
// Launch short range force compute on GPU
parallel_for(RangePolicy<Cuda>(0,N), PairFunctor);
// Asynchronously update data needed by kspace calculation
deeap_copy(OpenMP(), x_host, x_device);
// Launch Kspace force compute on Host using OpenMP
parallel_for(RangePolicy<OpenMP>(0,N), KSpaceFunctor);
// Asynchronously copy Kspace part of force to GPU
deeap_copy(OpenMP(), f_kspace, f_host);
// Wait for short range force compute to finish
Cuda::fence();
// Merge the force contributions
parallel_for(RangePolicy<Cuda>(0,N), Merge(f_device,f_kspace));
```
Reverse Offload – Using Asynchronous DeepCopy

- LAMMPS/example/accelerate/in.phosphate
- Goal overlap Pair with Kspace
  - find cutoff to balance weight of pair and kspace (here: 14)
- Kspace not threaded:
  - use 4 MPI ranks/GPU
  - use MPS server to allow more effective sharing
- When Overlapping:
  - Comm contains pair time since it fences to wait for pair force
  - 96% of Kspace time reduction

![Wall Time Measure Graph](image)
KokkosP Profiling Interface

- Dynamic Runtime Linkable profiling tools
  - Not LD_PRELOAD based (hooray!)
  - Profiling hooks are always enabled (i.e. also in release builds)
    - Compile once, run anytime, profile anytime, no confusion or recompile!
  - Tool Chaining allowed (many results from one run)
  - Very low overhead if not enabled

- Simple C Interface for Tool Connectors
  - Users/Vendors can write their own profiling tools
  - VTune, NSight and LLNL-Caliper

- Parallel Dispatch can be named to improve context mapping
- Initial tools: simple kernel timing, memory profiling, thread affinity checker, vectorization connector (APEX-ECLDRD), vtune connector, nsight connector

- www.github.com/kokkos/kokkos-tools
Basic Profiling

- Provide names for parallel operations:
  - `parallel_for("MyUserProvidedString", N, KOKKOS_LAMBDA ... );`
  - By default: typename of functor/lambda is used

- Will introduce barriers after each parallel operation

- Profile hooks for both GPU and CPU execution

- **Simple Timer:**
  - `export KOKKOS_PROFILE_LIBRARY=${KP_TOOLS}/kp_kernel_timer.so`
  - `${KP_TOOLS}/kp_reader machinename-PROCESSID.dat`
  - Collect: Time call-numbers time-per-call %of-kokkos-time %of-total-time

<table>
<thead>
<tr>
<th>Function</th>
<th>Time</th>
<th>Call-Num</th>
<th>Time-Per-Call</th>
<th>% of Total Time</th>
<th>% of Kokkos Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair::Compute</td>
<td>0.32084</td>
<td>101</td>
<td>0.00318</td>
<td>40.517</td>
<td>27.254</td>
</tr>
<tr>
<td>Neigh::Build</td>
<td>0.24717</td>
<td>17</td>
<td>0.01454</td>
<td>31.214</td>
<td>20.996</td>
</tr>
<tr>
<td>N6Kokkos4Impl20ViewDefaultConstructINS_6OpenMPEdLb1EEE</td>
<td>0.04194</td>
<td>113</td>
<td>0.00037</td>
<td>5.297</td>
<td>3.563</td>
</tr>
<tr>
<td>N6Kokkos4Impl20ViewDefaultConstructINS_4CudaEiLb1EEE</td>
<td>0.03112</td>
<td>223</td>
<td>0.00014</td>
<td>3.930</td>
<td>2.643</td>
</tr>
<tr>
<td>NVE::initial</td>
<td>0.02929</td>
<td>100</td>
<td>0.00029</td>
<td>3.699</td>
<td>2.488</td>
</tr>
<tr>
<td>32AtomVecAtomicKokkos_PackCommSelfIN6Kokkos4CudaELi1ELi0EE</td>
<td>0.02215</td>
<td>570</td>
<td>0.00004</td>
<td>2.797</td>
<td>1.881</td>
</tr>
<tr>
<td>NVE::final</td>
<td>0.02112</td>
<td>100</td>
<td>0.00021</td>
<td>2.667</td>
<td>1.794</td>
</tr>
</tbody>
</table>
Profiling Kokkos: Vtune Vanilla

- Template abstractions obscure the call stack
- Confusing identification of Parallel Regions
  - OpenMP parallel for is in a single file: Kokkos_OpenMP_Parallel.hpp
- Very long function names
Profiling Kokkos: Vtune Connector

- Use itt interface to add Domain and Frame markings
  - each kernel is its own domain, a frame is used for individual kernel invocations
- Vtune allows filtering, zoom in, etc. based on Domain and Frames
- Domain markings make Cuda Kernels visible
Profiling Kokkos: Nsight

- Nsight critical for performance optimization
  - Bandwidth analysis
  - Memory access patterns
  - Stall reasons

- Problem: again template based abstraction layers make awful function names, even worse than in vtune
Profiling Kokkos: Nsight Cuda 8

- Cuda 8 extends NVTX interface
  - Named Domains in addition to named Ranges
  - Using NVProf-Connector to pass user-provided names through
  - Shows Host Regions + GPU Regions
Early Experience with CUDA 8

- Critical bug fixes: relocatable device code required for Kokkos Tasking
- Significant improvements in compilation time, and binary size
- Only issue observed: different decisions about register usage (sometimes better, sometimes worse)

![Compile Time Diagram]

![Binary Size Diagram]

![Performance Diagram]
More Information

Code Projects:

- [www.github.com/kokkos/kokkos](www.github.com/kokkos/kokkos): Kokkos Core Repository
- [www.github.com/trilinos/Trilinos](www.github.com/trilinos/Trilinos): Trilinos Repository

Presentations:


At GTC:

- L6108 - Kokkos, Manycore Performance Portability Made Easy for C++ HPC Applications
- S6212 - Complex Application Proxy Implementation on the GPU Through Use of Kokkos and Legion
- S6292 - Gradually Porting an In-Use Sparse Matrix Library to Use CUDA (Wed 14:30 212A)
- S6145 - Kokkos Hierarchical Task-Data Parallelism for C++ HPC Applications (Thur 10:00 211A)
- S6257 - Kokkos Implementation of Albany: Towards Performance Portable Finite Element Code (Thur 10:30 211A)

Previous Talks at GTC 2014,2015
Exceptional service in the national interest