SAND2015-8901C

Maintainability and Performance for LAMMPS

Christian Trott (crtrott@sandia.gov), Tzu-Ray Shan, Stan Moore, Aidan Thompson and Steve Plimpton
Center for Computing Research
Sandia National Laboratories

Over the last few years, molecular dynamics (MD) simulation codes have been at the forefront of supporting new hardware architectures. For example, most of the major older codes, including AMBER, Gromacs, NAMD, and LAMMPS, have provided significant support for GPUs for several years, and newer codes like HOOMD have been developed specifically for GPUs. We believe the older codes all approached this task initially by creating special variants of their most important MD kernels for the GPU. Some codes support running completely on GPUs, with no regular data transfers to the CPU other than for MPI communication. Others offload only certain key computations to the GPU. Many support hybrid execution where, for example, non-bonded interactions are calculated on the GPU at the same time bonded interactions and long-range Coulombics are calculated on the CPU. In general, performance improvements over many-core CPU-only clusters have been quite good.

For LAMMPS, this approach of creating variants of key kernels tuned for each architecture (CPU/MPI-only, OpenMP, GPU, Phi, etc), has become increasingly difficult to sustain over time as the code has grown and new architectures proliferate. As a general materials simulation code with models at the atomic to meso to continuum scales, LAMMPS currently has non-bonded kernels for ~120 different functional forms. There are likewise ~40 different functional forms for bonded interactions (bonds, angles, dihedrals, impropers) and several variants of long-range Coulombic models. The code also has 100+ options for add-on calculations which affect the dynamics such as thermostats, barostats, different time integration schemes, interactions with boundaries and other objects, external forces, etc., as well as several dozen optional diagnostics, which involve loops over atoms or more complex calculations.

Overall, we estimate LAMMPS thus contains around 500 unique kernels, which if not optimized could become bottlenecks in a particular simulation on specific hardware. The challenge this creates is reflected in the current GPU capabilities of LAMMPS, which only support a small fraction of all these LAMMPS features. The problem is compounded because LAMMPS input scripts allow users to combine these capabilities in flexible and unpredictable ways. As a consequence, just porting the most important kernels to a given architecture may result in poor performance for the majority of simulations users actually run.

To address this issue a significant portion of new code development for LAMMPS is now moving in a different direction by using Kokkos. Kokkos is a programming model recently developed at Sandia National Laboratories to enable performance

portability across multiple hardware architectures for a single source-code base [cite Kokkos, cite LAMMPS Kokkos paper]. Kokkos is openly available at github.com/kokkos/kokkos. We note that Kokkos is not LAMMPS-specific. Rather it is a general tool now in use by many groups for a variety of scientific applications. Within Sandia, it is being integrated into several production-level engineering codes as well as the Trilinos linear and non-linear solver library [cite] as the path forward to hopefully insulate them from the changing HPC hardware landscape.

In brief summary, Kokkos is based on 6 abstraction concepts:

> (1) Users express parallel computations with parallel patterns; e.g., for-each, reduce, scan, and directed acyclic graphs (DAGs) of tasks.
>
> (2) Parallel computations occur within execution spaces of a heterogeneous architecture; e.g., latency-optimized CPU cores and throughput optimized GPU cores.
>
> (3) Parallel computations are scheduled according to execution policies; e.g., statically scheduled range [0..N) and dynamically scheduled thread teams.
>
> (4) Data are allocated within memory spaces of a heterogeneous architecture; e.g., in CPU main memory and GPU memory.
>
> (5) Data are allocated through multidimensional arrays with polymorphic layout that specifies how an array's multi-index domain space is mapped to an allocation within a memory space.
>
> (6) Arrays may be annotated with access intent traits such as "random access" or "atomic access." Kokkos can use these traits to map array entry access to architecture-specific mechanisms such as GPU texture cache or atomic instructions.

The key idea is that writing code that uses these abstraction concepts allows one to create performance-portable applications. The details are beyond the scope of this short paper, but extensive documentation to help get started with Kokkos is available, including tutorials of different lengths at github.com/kokkos/kokkos-tutorials, a programming guide, a variety of miniApps, and code examples.

In LAMMPS, Kokkos variants of various kernels are for now available as a KOKKOS "package". It currently includes ~30 non-bonded interaction types, 7 bonded interaction types and a handful of the above-noted add-ons and diagnostics. The infrastructure within LAMMPS to provide Kokkos-support is mostly implemented, so that moving forward is now more a matter of breadth of coverage across LAMMPS kernels rather than changes to the LAMMPS core. The same source code in all of these Kokkos-enabled kernels can be run either on CPUs (MPI-only, OpenMP), GPUs, or Intel Phis, with MPI providing inter-node parallelism. The KOKKOS package also supports various execution modes including complete offload,

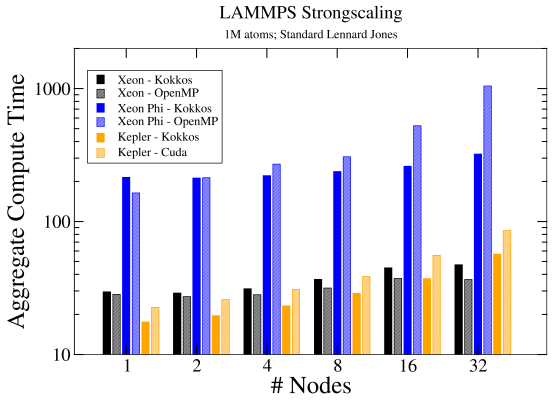
partial offload and true hybrid execution, which LAMMPS leverages for different node architectures. It also allows users to choose the optimal mode for a specific simulation.

A particular challenge for migrating an existing large application like LAMMPS to Kokkos is to handle changed data structures. For LAMMPS we designed a "dual-view" strategy whereby Kokkos is responsible for allocating the main data structures. LAMMPS first allocates one structure identical to the legacy data structure. Then, if beneficial for the accelerator hardware (GPU, Phi), a second structure is allocated with a different layout (e.g. row-order vs. column-order 2d arrays). This allows legacy code modules to remain functional by using the first layout, while Kokkos-optimized LAMMPS modules use the second layout without knowing details of the layout itself (abstraction #5 above). Invisibly to the LAMMPS developer, the Kokkos library handles the movement and rearrangement of data between the dual layouts..

We also strive to keep interoperability between the KOKKOS package, the base physics modules and the other hardware specific modules functional. This means a user does not have to wait until all desired functionality is available in the KOKKOS package, but can mix and match. Longer term this will allow us to have a baseline threaded performance portable code through the use of Kokkos, while hardware specific optimizations of particular important kernels can be done in the hardware-specific programming models for each architecture.

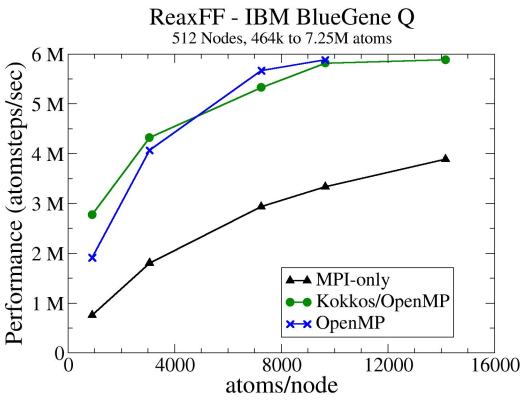
One significant challenge for the code migration effort is funding of baseline porting. Most grants used to develop code for LAMMPS are tied to providing new functionality. It is significantly harder to obtain funding for porting of existing features to a new programming model, even if this allows support for a new hardware architecture.

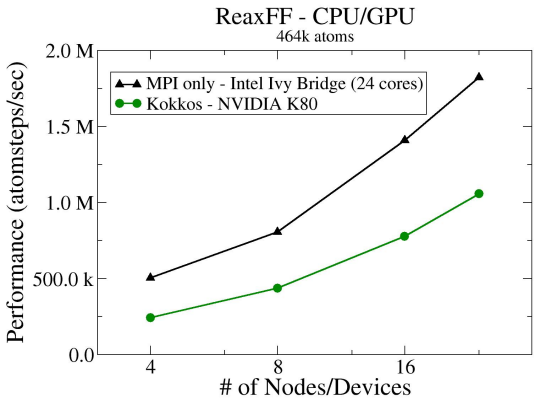
Performance results are so far very favorable, with the Kokkos variants of kernels being roughly on par with the hardware-specific variants. An example is shown below where a strong scaling run of a standard Lennard Jones system is shown for 1 to 32 nodes. The Xeon is a dual socket Sandy Bridge node with 16 cores per node, the Xeon Phi is a 57 core variant, where each Xeon Phi device is counted as a node (we run in native mode), and the Kepler system uses K20x GPUs where each GPU is counted as its own node. The aggregate compute time is equivalent to the total node-hours per run and is computed as wall clock time multiplied by the number of nodes. Therefore the ideal result would be constant height bars for all node counts, while increasing bar height indicates suboptimal strong scaling. We compared the KOKKOS package on all three architectures with the OpenMP and the USER-CUDA packages in LAMMPS.



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 was recently given by the development of a Kokkos variant of the Reax Force Field, which is significantly more complex than the standard 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 raw 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 on par or better than the alternative implementations.





We believe that the strategy we have taken will allow us to keep a maintainable code base while delivering consistent good performance across all relevant hardware architectures. But we expect to see a divergence of approaches for performance portability between different codes in the field based on the number of unique kernels. While applications with hundreds of kernels will likely have to choose an abstraction layer such as Kokkos or a portable programming model such as OpenMP 4.x to keep maintainability, others might be better off following the existing strategy of replicating kernels in various architecture-specific programming models which allows them to tailor kernels to a higher degree and achieve better performance.

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.