SAND2023-02568C

Exceptional service in the national interest

Sandia National Laboratories

GPU Programming in LAMMPS via Kokkos

Stan Moore

Sandia National Laboratories

IPAM New Mathematics for the Exascale: Applications to Materials Science

U.S. DEPARTMENT OF ENERGY
NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

About Me

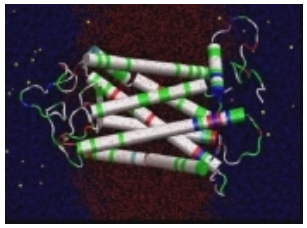
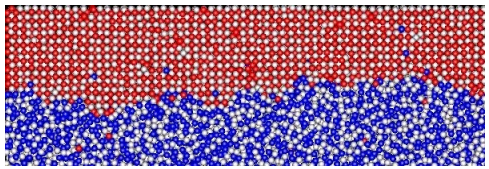
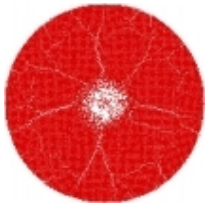
Sandia National Laboratories

- Stan Moore
  - One of the LAMMPS code developers at Sandia National Laboratories in Albuquerque, New Mexico
  - Been at Sandia for 10 years
  - Main developer of the KOKKOS package in LAMMPS (runs on GPUs and multi-core CPUs)
  - Expertise in long-range electrostatics methods
  - PhD in Chemical Engineering, dissertation on molecular dynamics method development for predicting chemical potential

LAMMPS

Sandia National Laboratories

- Large-scale Atomic/Molecular Massively Parallel Simulator
- https://lammps.org
  - Open source, C++ molecular dynamics code
  - Bio, materials, mesoscale

  - Particle simulator at varying length and time scales
    - Electrons → atomistic → coarse-grained → continuum
  - Spatial-decomposition of simulation domain for parallelism
  - Energy minimization, dynamics, non-equilibrium MD
  - GPU and OpenMP enhanced
  - Can be coupled to other scales: QM, kMC, FE, CFD, …

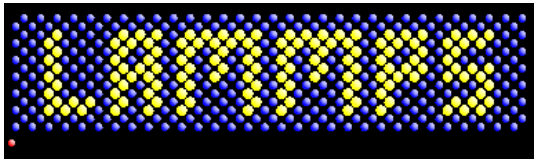
Kokkos Performance Portability Library

Sandia National Laboratories

- Kokkos is an abstraction layer between programmer and next-generation platforms
- Allows the same C++ code to run on multiple hardware (Intel CPU, NVIDIA GPU, Intel GPU, AMD GPU, etc.)
- Kokkos consists of two main parts:
  1. Parallel dispatch—threaded kernels are launched and mapped onto backend languages such as CUDA or OpenMP
  2. Kokkos views—polymorphic memory layouts that can be optimized for a specific hardware (LayoutLeft, LayoutRight, etc.)
- Used on top of existing MPI parallelization (MPI + X)
- Used by many codes, open-source, can be downloaded at https://github.com/kokkos/kokkos

C. R. Trott, et al. "Kokkos 3: Programming Model Extensions for the Exascale Era". IEEE Transactions on Parallel and Distributed Systems 33. 4(2022): 805-817.

Accelerator Packages in LAMMPS



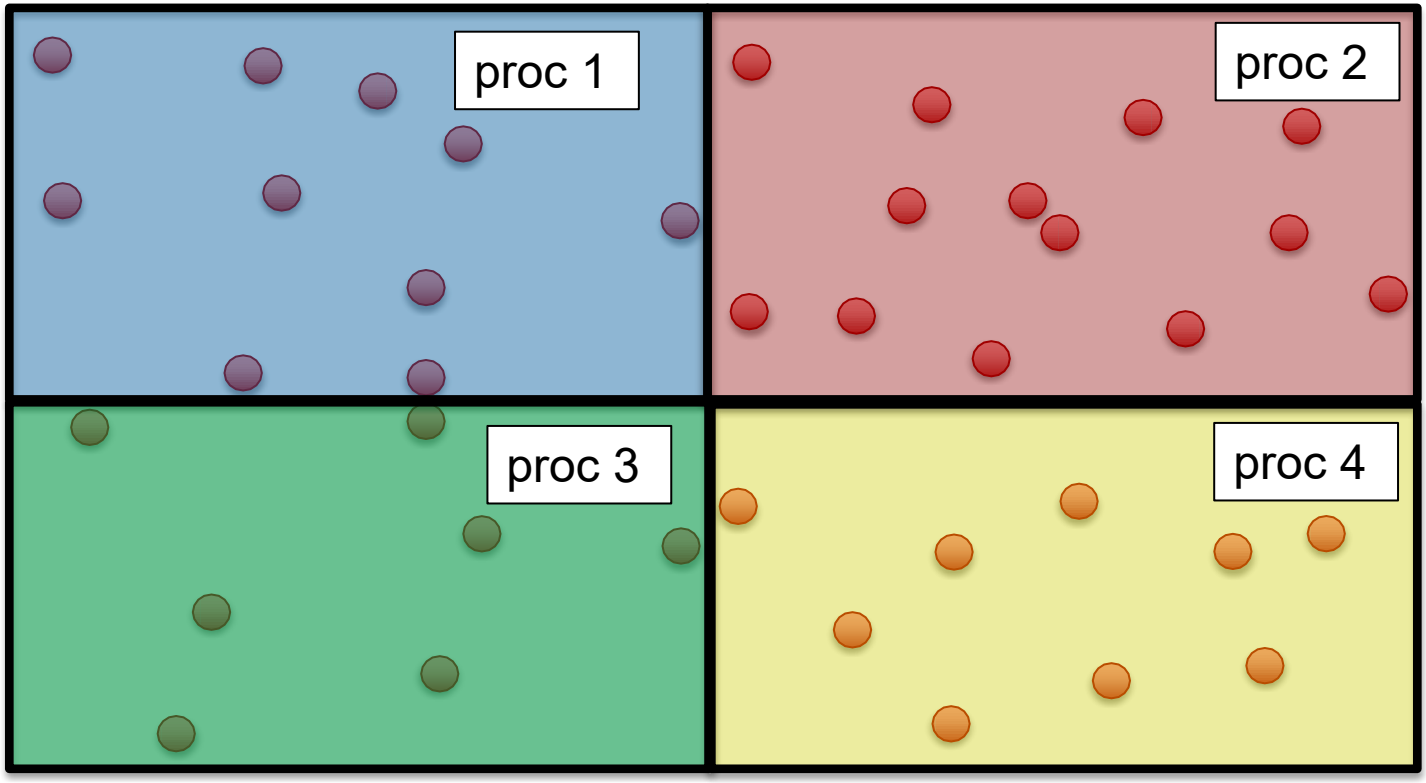
- **Vanilla C++ version**

Accelerator packages:

- **USER-OMP Package:** native OpenMP threading
- **USER-INTEL**: native OpenMP threading, enhanced SIMD vectorization, uses hardware intrinsics, fast on CPUs but very complex code
- **GPU Package**: native CUDA and OpenCL support, only runs a few kernels (e.g. *pair* force calculation) on GPU, needs multiple MPI ranks per GPU to parallelize CPU calculations
- **KOKKOS Package**: tries to run everything on device, supports CUDA (NVIDIA GPUs), HIP (AMD GPUs), SYCL (INTEL GPUs), and OpenMP (CPU) threading backends via Kokkos library

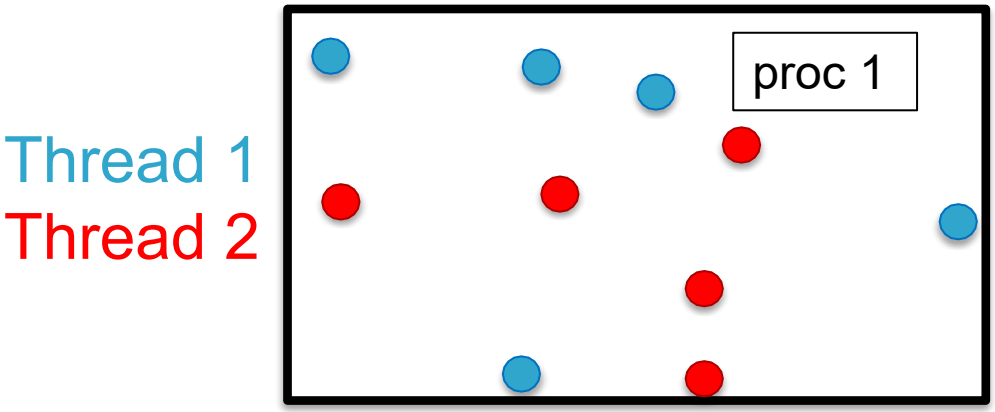
MPI Parallelization Approach

- Domain decomposition: each processor owns a portion of the simulation domain and atoms therein



Multithreading (e.g. OpenMP, CUDA)

- Used on top of MPI domain decomposition
- Each thread processes a subset particles in a processor's subdomain
- Threads run concurrently, **no guarantee of order**



Threading in LAMMPS

Sandia National Laboratories

- Typically thread over owned atoms (*nlocal*)
- Additionally can thread over neighbors to expose more parallelism, but can have overheads. Default for 16k or less atoms with simple pair-wise potentials, also used for expensive machine learning potentials with low atom counts/GPU
- SNAP threads over atoms, neighbors, and bispectrum
- Can collapse multiple loops
- Typically use 1 MPI rank per GPU
- If significant parts of the code are not running on the GPU, then using multiple MPI ranks per GPU can help (need to enable CUDA MPS on NVIDIA GPUs)

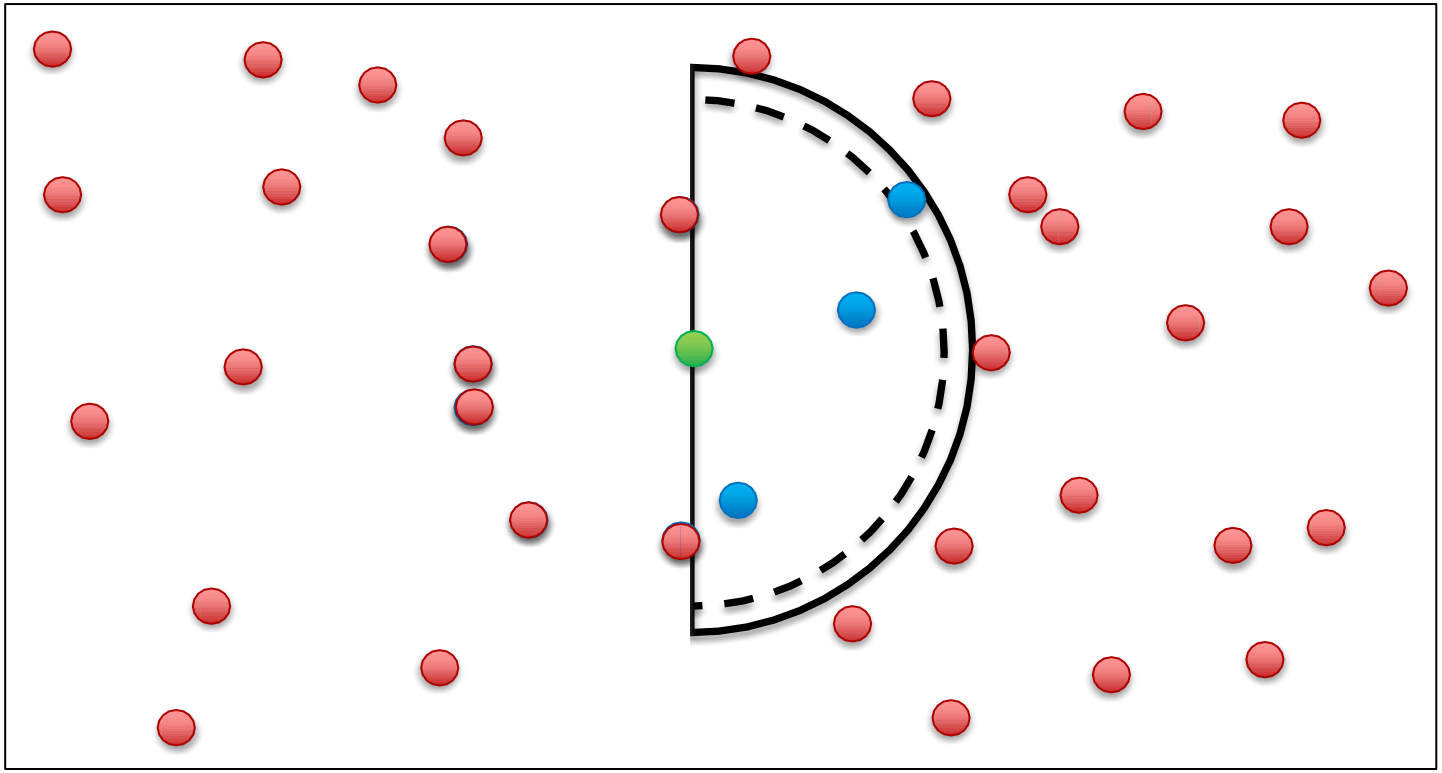
Thread Safety

Sandia National Laboratories

- Threads can execute in any order in a parallel loop, cannot assume same ordering as in serial
- Two threads writing to the same memory location at the same time causes a “data race”, e.g. half list when one thread is updating force on atom *i*, another thread is updating force on neighbor *j*, and *i* = *j*
- On GPUs, protect with *atomic* operations. Kokkos supports individual atomic operations and atomic views
- On CPUs, atomics are typically slow and thread count is low. Duplicating memory for each thread and then summing at the end can be faster (but has memory overhead)
- Kokkos provides both options conveniently via *Kokkos::ScatterView* (uses atomics for CUDA and data duplication for OpenMP)

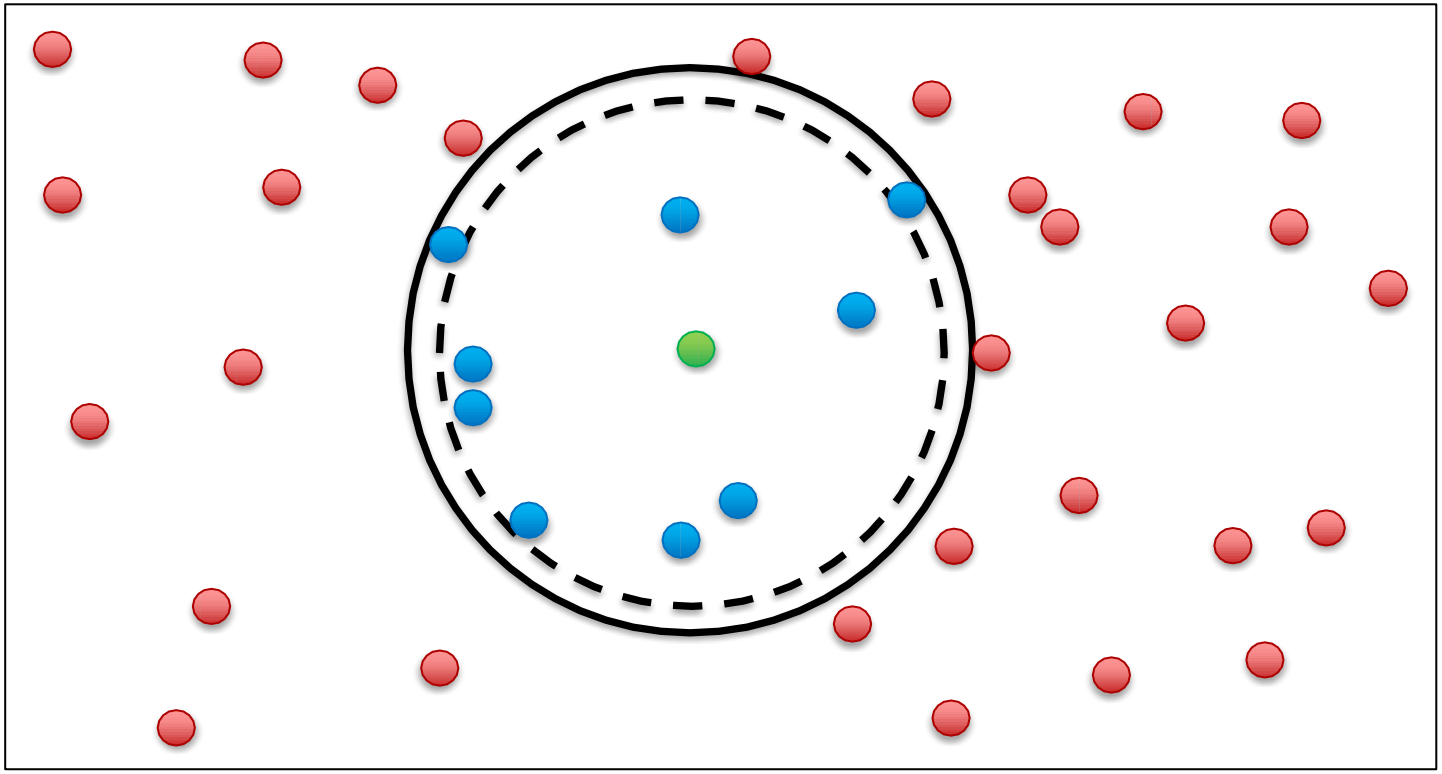
Half Neighbor List

- With newton flag on, each pair is stored only once, requires atomic operations for thread-safety



Full Neighbor List

- Each pair stored twice which doubles computation but reduces communication and doesn't require atomic operations for thread safety



Newton Option

Sandia National Laboratories

- Newton flag to *off* means that if two interacting atoms are on different processors, **both processors compute their interaction** and the resulting force information is not communicated
- Setting the newton flag to *on* saves computation but increases communication
- Performance depends on problem size, force cutoff lengths, a machine's compute/communication ratio, and how many processors are being used
- **Newton off typically better for GPUs**

KOKKOS Package Options

Sandia National Laboratories

- For CPUs, half neighbor list and newton on typically fastest
- For GPUs and simple pairwise potentials (e.g. Lennard Jones), full neighbor list and newton off typically fastest
- For manybody potentials (e.g. Tersoff), more work to duplicate, half neigh list is typically fastest on GPUs (fast hardware FP64 thread atomics on modern GPUs are a game changer)
- These options can be controlled via the *package* command, see https://docs.lammps.org/package.html
- Package commands can also be invoked on the command line:
  `-pk kokkos newton on neigh half`

Execution Spaces

- With GPUs, *Host* execution space = CPU backed (serial or OpenMP), *Device* = GPU
- All Kokkos pair styles are templated on *DeviceType*
- Compiler creates two different versions of the code, one for CPU backend and one for GPU backend
- User can choose at runtime which version to use via the LAMMPS *suffix* command

Memory Spaces

Sandia National Laboratories

- GPUs typically have high bandwidth memory that is not accessible from CPU: pointers to CPU DRAM cannot be accessed on GPU; pointers to GPU HBM cannot be access on CPU
- Performance penalty when transferring data between GPU and CPU: **try to keep memory on GPU as much as possible**
- If a LAMMPS style is not ported to Kokkos it will run on CPU in serial and require data transfer every time it is invoked: consider porting to Kokkos to improve performance
- In LAMMPS we use *Kokkos::DualView* sync and modify on Device and Host to transfer data
- Do not use *LMPDeviceType* directly, use *DeviceType* template parameter instead, since could be running on Host

Parallel Kernel Abstractions

Sandia National Laboratories

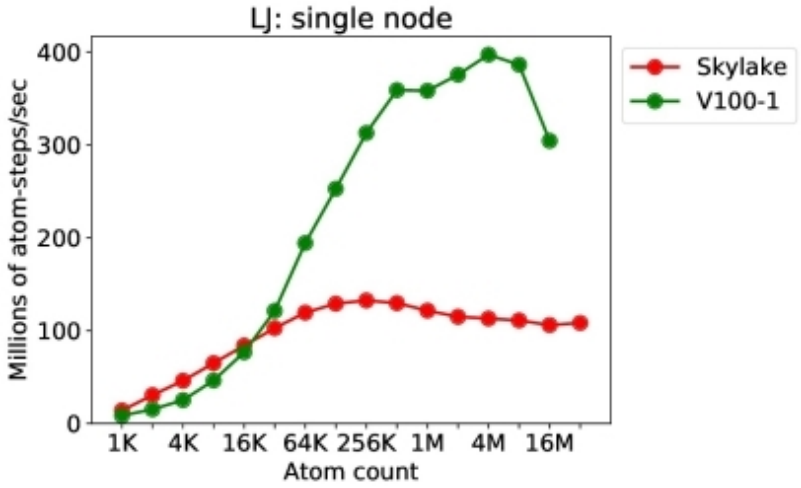
- Kokkos supports functors, tagged kernels where the whole class is the functor, and C++ lambdas (anonymous functors)
- Functors are the most general but take the most programming effort (have to copy all the needed data into the functor)
- Typically use tagged kernels in LAMMPS for convenience
- Can use C++ lambdas for simple kernels, but they have several limitations
  - Can't capture *this* pointer either explicitly or implicitly on GPUs. Requires creating a local copy of every class variable that gets captured instead
  - Cannot call class device functions within the lambda either
  - Better with C++17

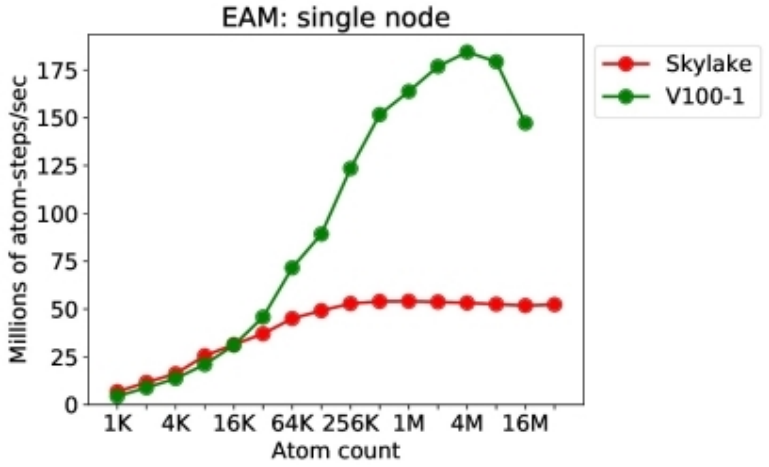
How to Optimize GPU Performance

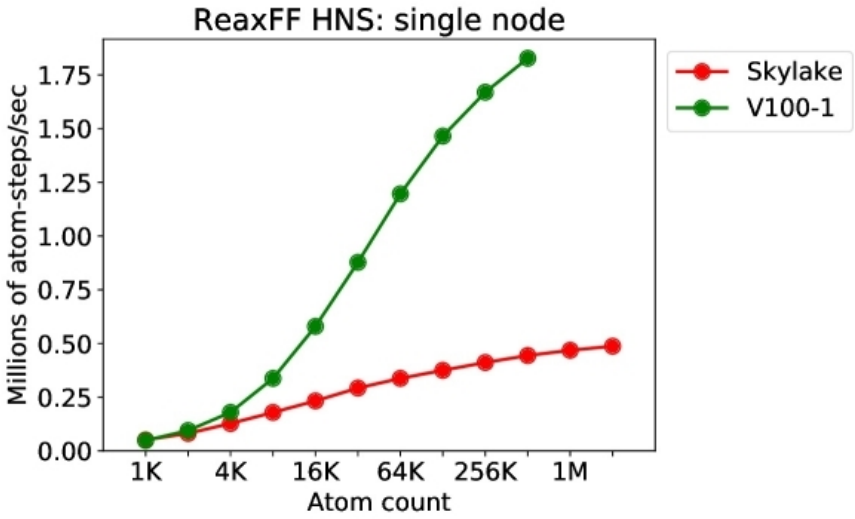
Sandia National Laboratories

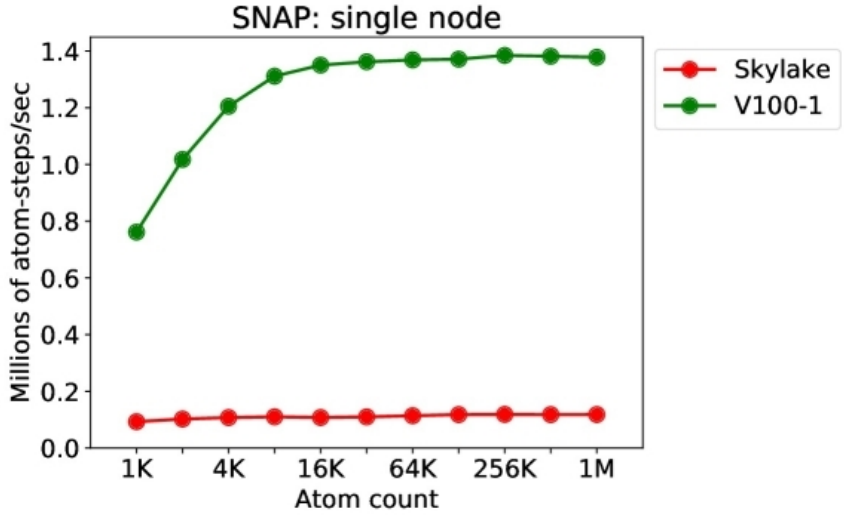
- Saturate GPU threads (increase number of atoms or expose more parallelism)
- Use caches efficiently (improve memory access patterns and data locality, be mindful of view *LayoutLeft* or *LayoutRight*)
- Keep atom data in GPU memory (avoid moving data as much as possible, port all kernels to Kokkos, use subview array instead of multiple scalar views)
- Avoid launch latency overhead for small systems (fuse kernels if possible)
- Avoid allocating memory every timestep (overallocate views and only grow if size is exceeded, don’t shrink)
- Watch out for Kokkos view initialization overheads (on by default but can turn off)

Performance of Different Potentials









Typical Debugging Workflow

Sandia National Laboratories

Much easier to debug on CPU than GPU!

1. Match Kokkos Serial backend (thermo output) with vanilla CPU version
   - Tools: Kokkos debug mode, gdb, valgrind, AddressSanitizer, printf
   - Compiling with "-O0" can help get an accurate backtrace
   - Typical issue: general bugs
2. Match Kokkos OpenMP backend running on 2 or more threads with vanilla CPU (or Kokkos Serial)
   1. Typical issue: data race or other thread safety issues
   2. Tools: Intel Inspector (many false positives), printf
3. Match Kokkos CUDA backend with Kokkos Serial backend:
   - Tools: cuda-gdb, cuda-memcheck, compile with UVM, printf
   - Compiling with Kokkos debug options (adds –lineinfo) or –G can help
   - Typical issues: missing sync/modify for data transfer (find with UVM), thread safety issues
   - Turn off fix langevin for determinism

Performance Profiling Tools

Sandia National Laboratories

1. Timing breakdown in LAMMPS log file*
2. Kokkos tools: my favorite tool is space-time-stack, shows both kernel times and memory use
3. nvprof (deprecated) for NVIDIA GPUs and rocprof for AMD GPUs
4. NVIDIA Nsight Compute and Systems tools (replacement for nvprof)
5. gprof, TotalView, etc. for CPU kernels

*Note: for KOKKOS package on NVIDIA GPUs, LAMMPS log file timing breakdown won't be accurate without *export CUDA_LAUNCH_BLOCKING=1*

Getting Help

Sandia National Laboratories

- Look at LAMMPS documentation, latest version here: https://docs.lammps.org/Manual.html
- Search the MatSci LAMMPS forum archives https://matsci.org/lammps, join and post new questions
- Submit a draft pull request on Github: https://github.com/lammps/lammps
- LAMMPS reference paper: gives an overview of the code including its parallel algorithms, design features, performance, and brief highlights of many of its materials modeling capabilities https://doi.org/10.1016/j.cpc.2021.108171

Code Examples

Sandia National Laboratories

- Coul/wolf, simple pairwise potential: https://github.com/lammps/lammps/blob/develop/src/KOKKOS/pair_coul_wolf_kokkos.cpp
- EAM, manybody potential: https://github.com/lammps/lammps/blob/develop/src/KOKKOS/pair_eam_kokkos.cpp
- PACE, machine learning potential: https://github.com/lammps/lammps/blob/develop/src/KOKKOS/pair_pace_kokkos.cpp

Thank You

- Questions?