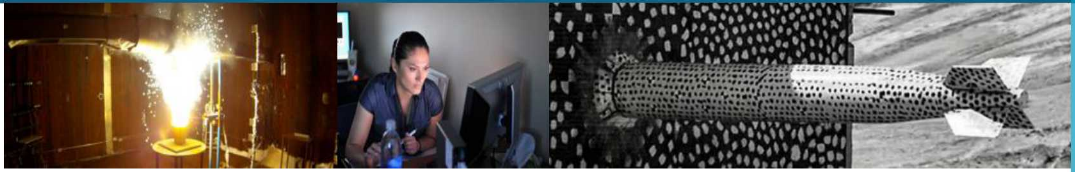
SAND2019-10367PE

Kokkos and SNAP work in support of EXAALT and LAMMPS



Stan Moore

2019 CoPA All-Hands Meeting

Santa Fe, NM

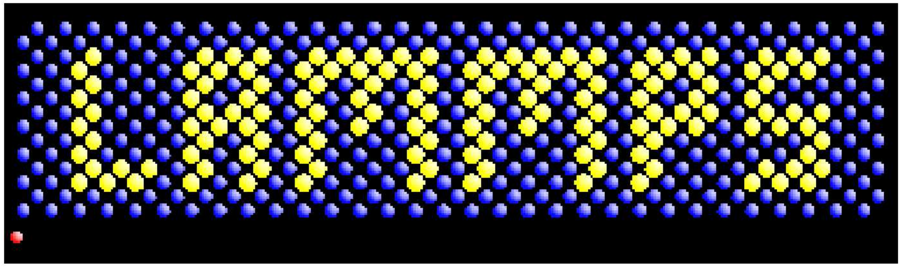
Recent Performance Work

2x improvement of **small Lennard-Jones systems** (~1000 atoms) on a single V100 GPU

2x improvement of **PPPM long-range electrostatics** on a V100 GPU (to be released soon)

5x improvement of **SNAP potential** on V100 GPUs (regular CPU version is also over 2x faster on CPUs than before)

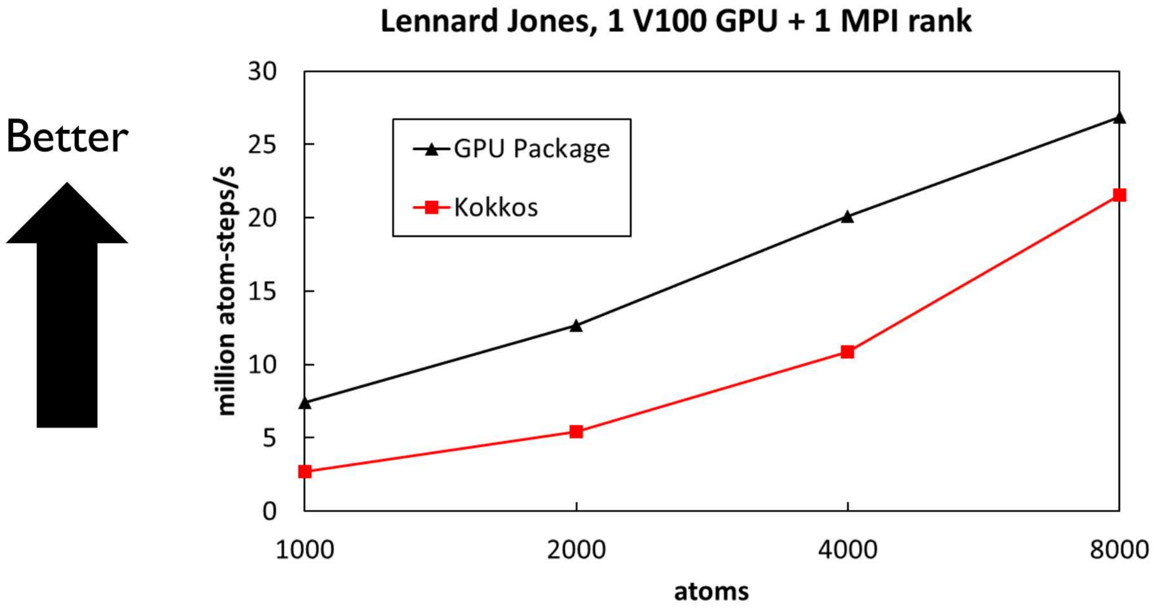
Improved **OpenMP threading performance** by adding Kokkos ScatterView data duplication option (helped several pair styles, from Lennard-Jones to ReaxFF)



Small Lennard-Jones Systems

When comparing LAMMPS Kokkos package to CUDA version (GPU package) as well as other MD codes (e.g. OpenMM) noticed large performance gap for small LJ systems

Needs for small systems: expose more parallelism and fuse kernels to reduce launch latency



Optimizations for Small Systems

1. Added team threading over neighbors in addition to atoms when system size is 16K atoms or less (slower for large systems due to overheads)

2. When running on 1 MPI rank, fused 12 communication buffer pack and unpack kernels into 2 kernels

3. Reduced data transfer for small systems by reducing the "extra" amount allocated for atom data, but also growing exponentially by a constant factor

4. Collapsed two scalar views into a single view which got rid of a deep copy

5. Run periodic boundary kernel on host CPU if comm is already on the host, avoids GPU data movement and one kernel launch

6. Only copy force on ghost atoms from GPU to CPU if using newton option

And more…

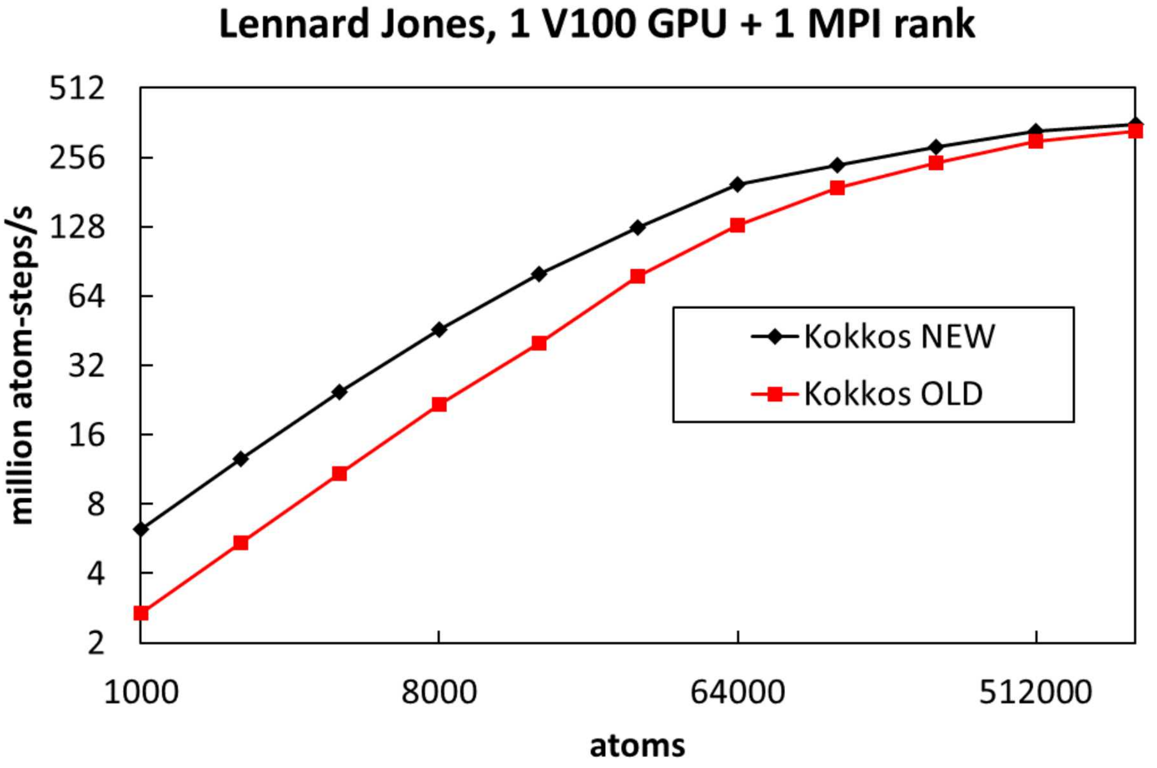
*Tried fusing Verlet integrator initial and final kernels for intermediate timesteps. Also fused force zeroing kernel with pair compute. Gives good speedup, but breaks modularity of LAMMPS. Is this tradeoff worth it?

Results

2.3x speedup for 1000 atoms

Can’t use fused comm with more than 1 MPI rank, use Cabana method instead?

Better



Performance comparison with GPU package

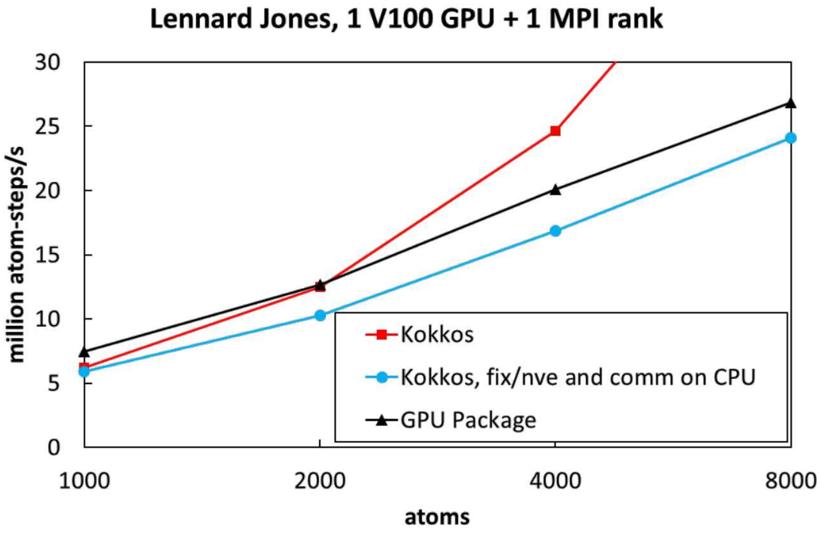
Kokkos uses special fused MPI comm kernel when running on a single GPU

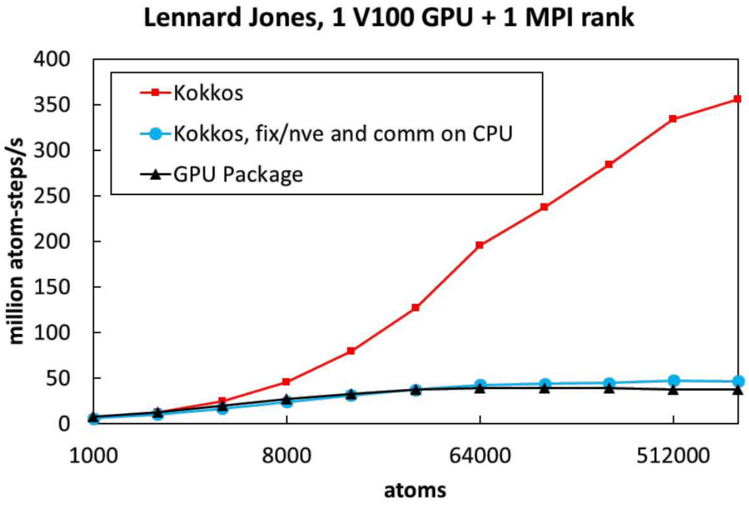
Integrator is running serially on CPU for GPU package

Performance penalty for moving atom data between GPU and CPU

Double precision only

Better

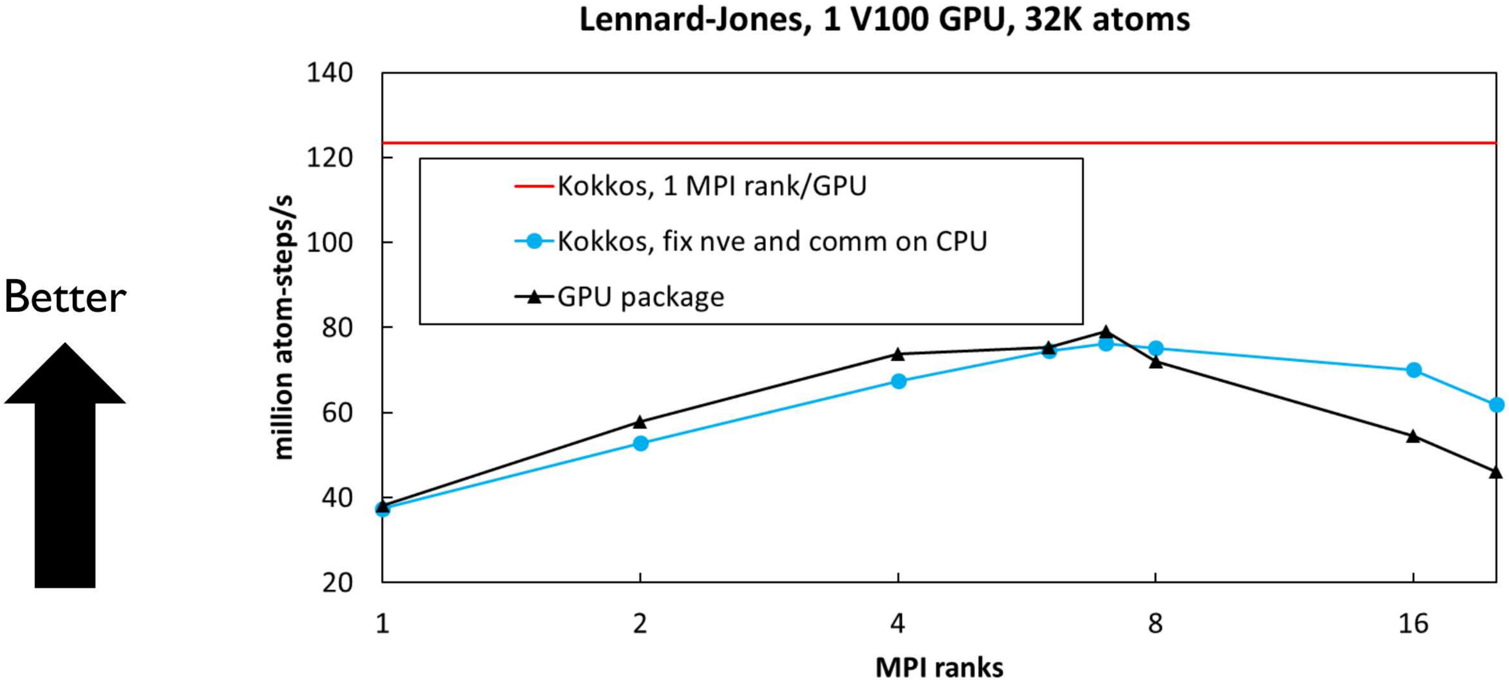




Multiple MPI ranks per GPU

Using multiple MPI ranks/GPU can help when parts of the code are not Kokkos-enabled

MUST use CUDA MPS with multiple MPI ranks per GPU to get good performance



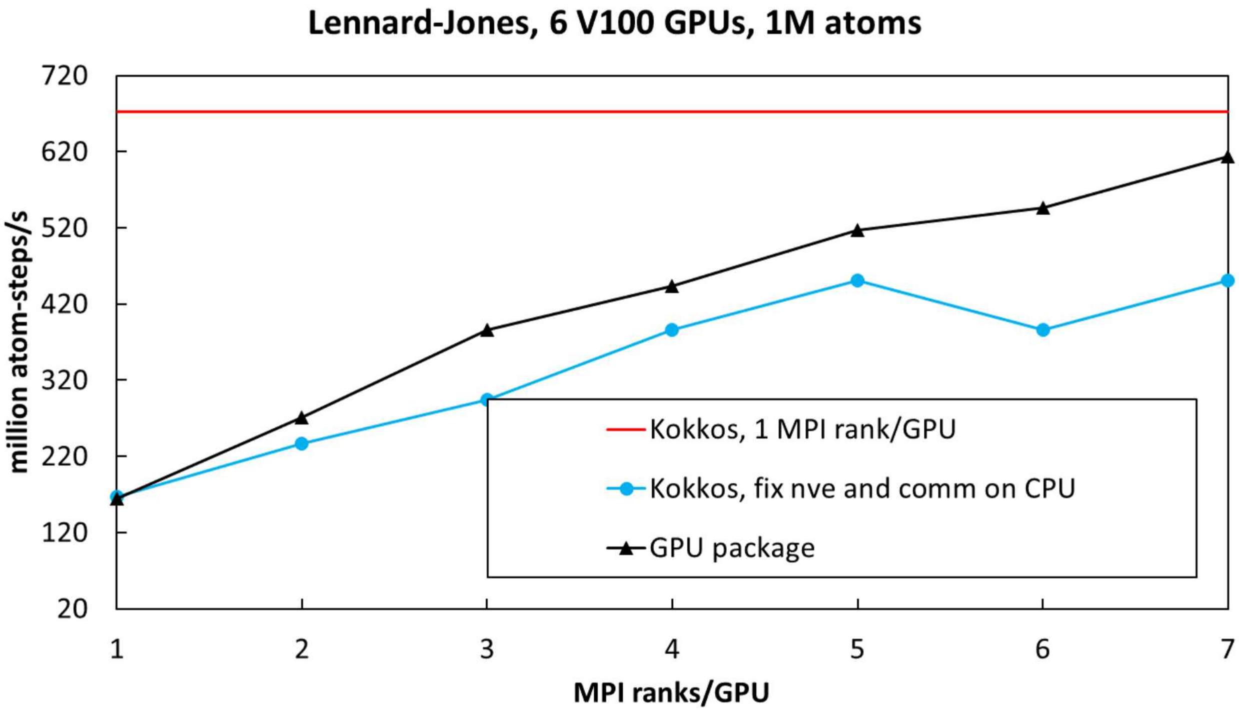
Performance comparison with GPU package

Full Summit node (6 V100 GPUs)

Using multiple MPI ranks/GPU closes most of the performance gap

Using pinned memory may help Kokkos with integrator and comm on host CPU

Better



FFTs for Long-range Electrostatics

3 options for FFTs on GPUs:

1. Run MPI-distributed on host CPUs: FFTs not GPU-accelerated, lots of data movement between CPUs and GPUs
2. Gather all the data to 1 or a few GPUs on a node and perform entire 3D FFT using cuFFT (not MPI-distributed): faster because FFTs are GPU-accelerated but problem size is limited to what can fit on a few GPUs
3. Run MPI-distributed but perform 1D FFTs (pencil decomposition) using cuFFT on GPUs, in theory can run huge FFTs on thousands of GPUs

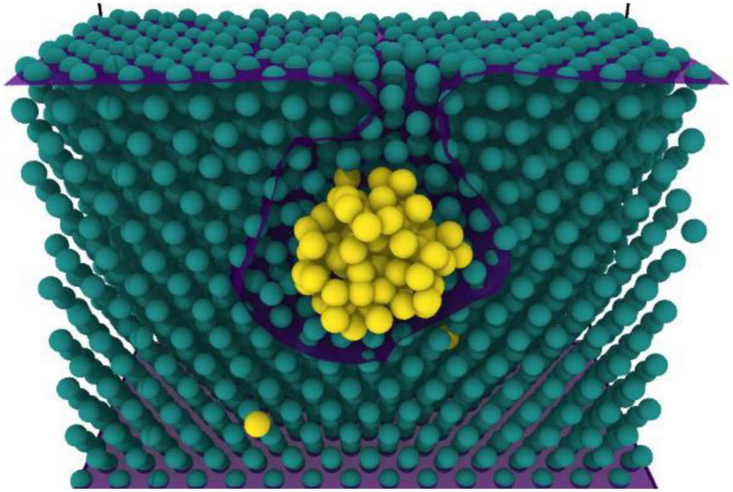
Option #1 currently in LAMMPS Kokkos package. Implemented option #3 in LAMMPS (will be released soon) and got a **~2x speedup** on the 32K atom Rhodopsin benchmark. Option #2 also being investigated.

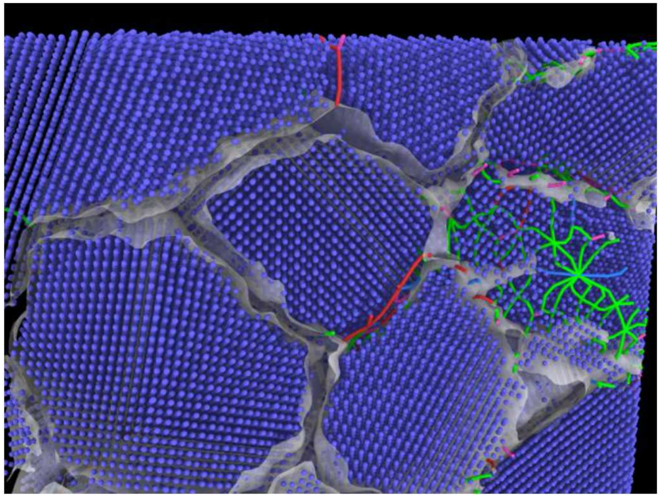
Connections between CoPA & EXAALT & LAMMPS

EXAALT ECP project seeks to extend accuracy, length, and time scales of material science simulations for fission/fusion reactors

Uses classical models like SNAP via LAMMPS, and quantum models (DFTB) via LATTE (also a CoPA emphasis)

EXAALT wants to run millions of small MD replicas (1K to 1M atoms) via ParSplice as fast as possible (not one large simulation with billions of atoms)



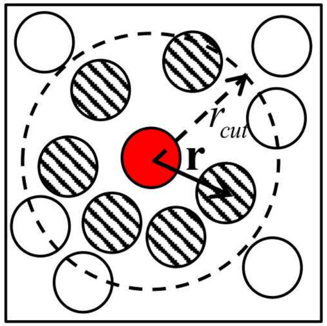


SNAP Potential

- Machine-learned MD potential that seeks for quantum-chemistry accuracy
- Neighbors of each atom are mapped onto unit sphere in 4D

$$(q_0, q, f) = \left(q_0^{max}\, r/r_{cut}, \cos^{-1}(z/r), \tan^{-1}(y/x)\right)$$

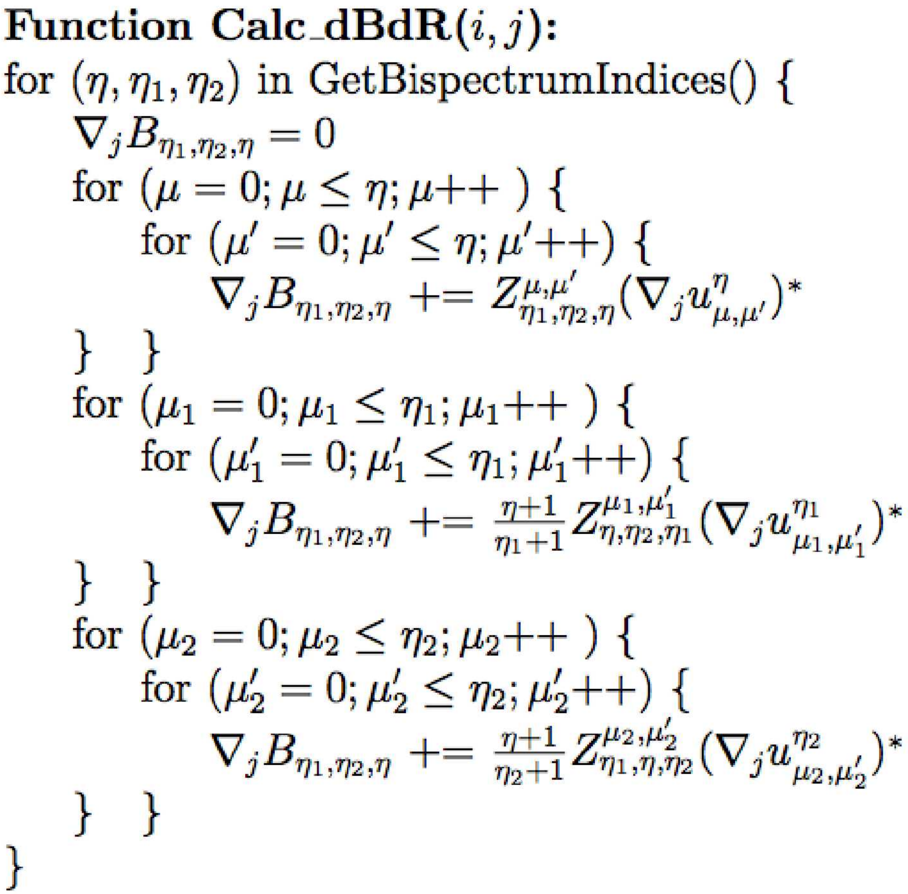
- Density around each atom is expanded in a basis of **4D hyperspherical harmonics**
- Bispectrum components of the 4D hyperspherical harmonic expansion are used as the geometric descriptors of the local environment
  - Preserves universal physical symmetries
  - Invariant to rotation, translation, permutation
  - Size-consistent
- SNAP uses linear regression to fit coefficients to DFT data



$$u^j_{m,m'} = U^j_{m,m'}(0,0,0) + \sum_{r_{ii'}<R_{cut}} f_c(r_{ii'}) w_i U^j_{m,m'}(\theta_0, \theta, \phi)$$

$$B_{j_1,j_2,j} = \sum_{m_1,m'_1=-j_1}^{j_1} \sum_{m_2,m'_2=-j_2}^{j_2} \sum_{m,m'=-j}^{j} (u^j_{m,m'})^* H^{jmm'}_{\substack{j_1 m_1 m'_1 \\ j_2 m_2 m'_2}} \; u^{j_1}_{m_1,m'_1} u^{j_2}_{m_2,m'_2}$$

SNAP Force Calculation



```
Function Calc_dBdR($i, j$):
for $(\eta, \eta_1, \eta_2)$ in GetBispectrumIndices() {
    $\nabla_j B_{\eta_1,\eta_2,\eta} = 0$
    for $(\mu = 0; \mu \leq \eta; \mu{+}{+}$ ) {
        for $(\mu' = 0; \mu' \leq \eta; \mu'{+}{+})$ {
            $\nabla_j B_{\eta_1,\eta_2,\eta} \mathrel{+}= Z^{\mu,\mu'}_{\eta_1,\eta_2,\eta} (\nabla_j u^{\eta}_{\mu,\mu'})^*$
    }   }
    for $(\mu_1 = 0; \mu_1 \leq \eta_1; \mu_1{+}{+}$ ) {
        for $(\mu_1' = 0; \mu_1' \leq \eta_1; \mu_1'{+}{+})$ {
            $\nabla_j B_{\eta_1,\eta_2,\eta} \mathrel{+}= \frac{\eta+1}{\eta_1+1} Z^{\mu_1,\mu_1'}_{\eta,\eta_2,\eta_1} (\nabla_j u^{\eta_1}_{\mu_1,\mu_1'})^*$
    }   }
    for $(\mu_2 = 0; \mu_2 \leq \eta_2; \mu_2{+}{+}$ ) {
        for $(\mu_2' = 0; \mu_2' \leq \eta_2; \mu_2'{+}{+})$ {
            $\nabla_j B_{\eta_1,\eta_2,\eta} \mathrel{+}= \frac{\eta+1}{\eta_2+1} Z^{\mu_2,\mu_2'}_{\eta_1,\eta,\eta_2} (\nabla_j u^{\eta_2}_{\mu_2,\mu_2'})^*$
    }   }
}
```

- Deeply nested loops
- Loop structure not regular
- Loop bounds relatively small

SNAP Performance Improvements

SNAP effort part of the NESAP NERSC-9 (Perlmutter) program at NERSC and a OLCF GPU Hackathon

Aidan Thompson (Sandia) took the SNAP CPU code out of LAMMPS → **TestSNAP** stand-alone (realistic) force kernel, includes correctness check

Idea from Nick Lubbers (LANL) →Aidan made algorithmic improvements that reduced FLOP count and eliminated some intermediate storage → ~2x speedup on CPUs

Aidan reduced memory use by collapsing multidimension arrays into compact lists

Rahul Gayatri (NERSC):

1. wrote a CUDA/OpenACC version of **TestSNAP**
2. broke up the one monster kernel into many smaller kernels, reduces register pressure and allows tailoring launch parameters for each kernel, but blows up the memory
3. inverted loops and changed data layouts to improve memory access

Also had help from Sarah Anderson (Cray) and Evan Weinberg (NVIDIA)

These improvements were ported to Kokkos SNAP in LAMMPS by Stan Moore

SNAP Benchmarking on Summit

On Summit node, run 6 GPU + 1 CPU (36 cores) replicates for EXAALT

EXAALT benchmark uses 205 bispectrum coefficients, tungsten crystal

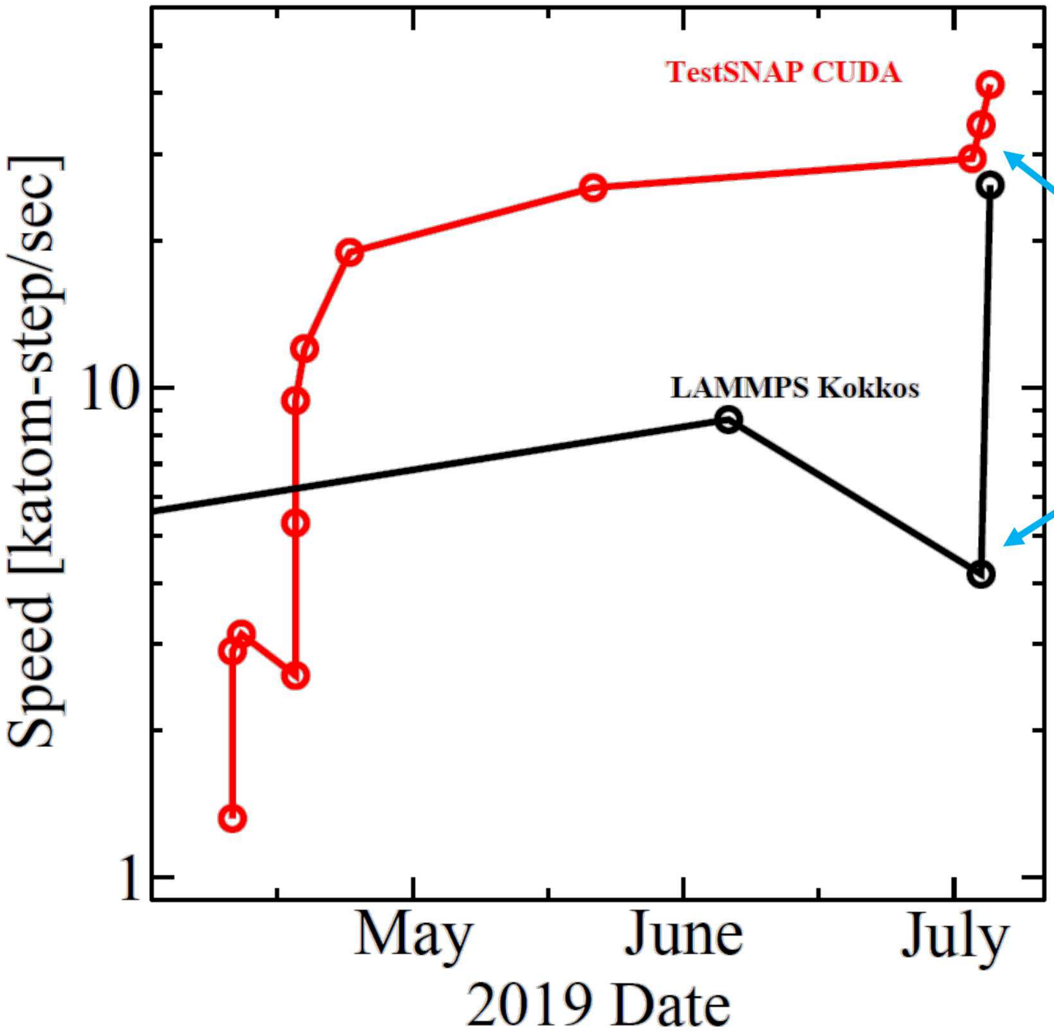
2018 LAMMPS on CPUs: 59.9 seconds

New LAMMPS on CPUs: 24.7 seconds: **2.4x faster**

2018 LAMMPS on 1 V100: 39.6 seconds

New LAMMPS on 1 V100: 7.2 seconds: **5.5x faster**

SNAP GPU Performance Over Time



OLCF GPU Hackathon

EXAALT FOM/KPP Projection for Summit

Mira (IBM BG/Q) FOM baseline: 0.182 Katoms-steps/s/node * 49152 Mira nodes

2018 LAMMPS performance on Summit: 33.7 Katom-steps/s/node * 4608 Summit nodes: projected **17.4x faster than Mira baseline**

New LAMMPS performance on Summit: 175.1 Katom-steps/s/node * 4608 Summit nodes: projected **90.2x faster than Mira baseline**

Recently ported energy minimization in LAMMPS to Kokkos, which is needed by ParSplice

Danny Perez (LANL) planning to validate these projections with large-scale Summit run soon

SNAP Benchmarking on Summit

More improvements discovered during GPU hackathon and implemented in **TestSNAP**, but not yet in LAMMPS:

- Transpose data layout part way through the algorithm to optimize access patterns (easy)
- Write code in a way to convince compiler to use 128 bit memory read/writes for complex data types (medium hard)

Rahul has been learning Kokkos and is still working on improving SNAP

More speedup to come!

Looking Forward to FY20

LAMMPS Kokkos package only supports double-precision (FP64)

Single (FP32) and mixed precision (FP32 for intermediate calculations and FP64 for accumulation) can significantly improve performance on both GPUs and CPUs (when reduced accuracy can be tolerated)

Kokkos host views for atom data are aliased to legacy host data structures (LayoutRight, FP64). If a non-Kokkos style is invoked, atom data is transferred down to CPU, calculation is performed, and data is transferred back up to GPU

Want to use FP32, LayoutLeft views on GPUs for better performance: add transpose and cast as an intermediate step when syncing between host/device views in Kokkos DualView

Adding single/mixed precision support not a trivial undertaking but gives significant benefit and is necessary for LAMMPS Kokkos performance to be competitive with other MD codes