Refactoring EXAALT MD for Emerging Architecture

Aidan Thompson, Stan Moore

Sandia National Laboratories

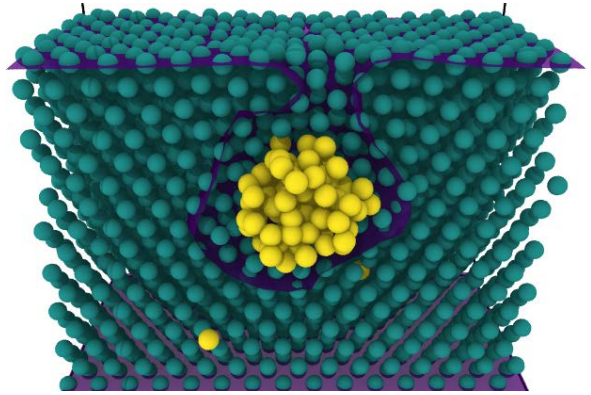
Rahul Gayatri

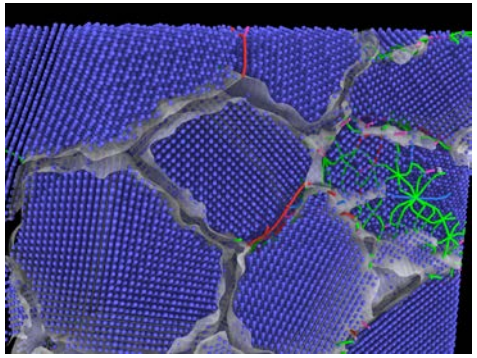
NERSC, Lawrence Berkeley National Laboratory

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.

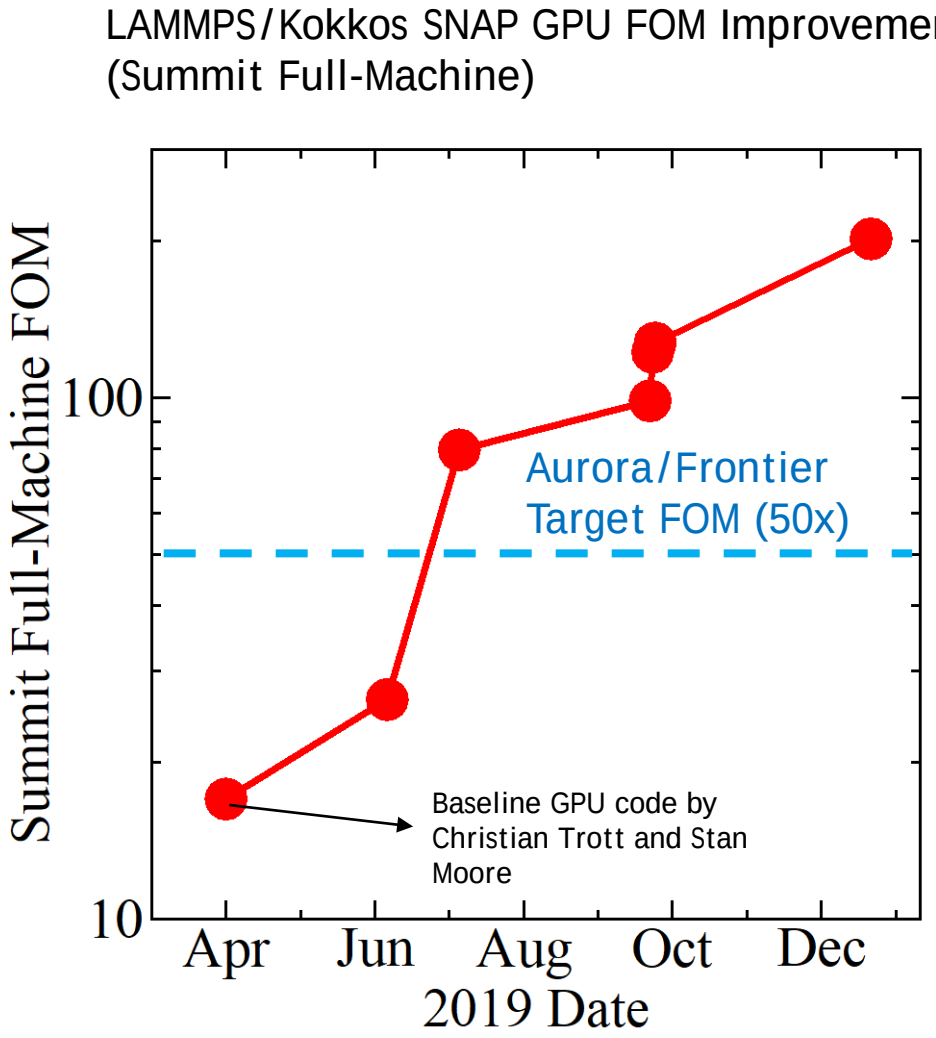
Connections between CoPA & EXAALT & LAMMPS

- ECP EXAALT project seeks to extend accuracy, length, and time scales of material science simulations for fission/fusion reactors using LAMMPS MD
- 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)
- Primary KPP target is MD of nuclear fusion materials that uses the SNAP interatomic potential in LAMMPS
- Performance directly depends on single-node performance for SNAP
- ECP CoPA codesign project targeting MD as one of its "sub-motifs"





Summary of EXAALT KPP Improvements



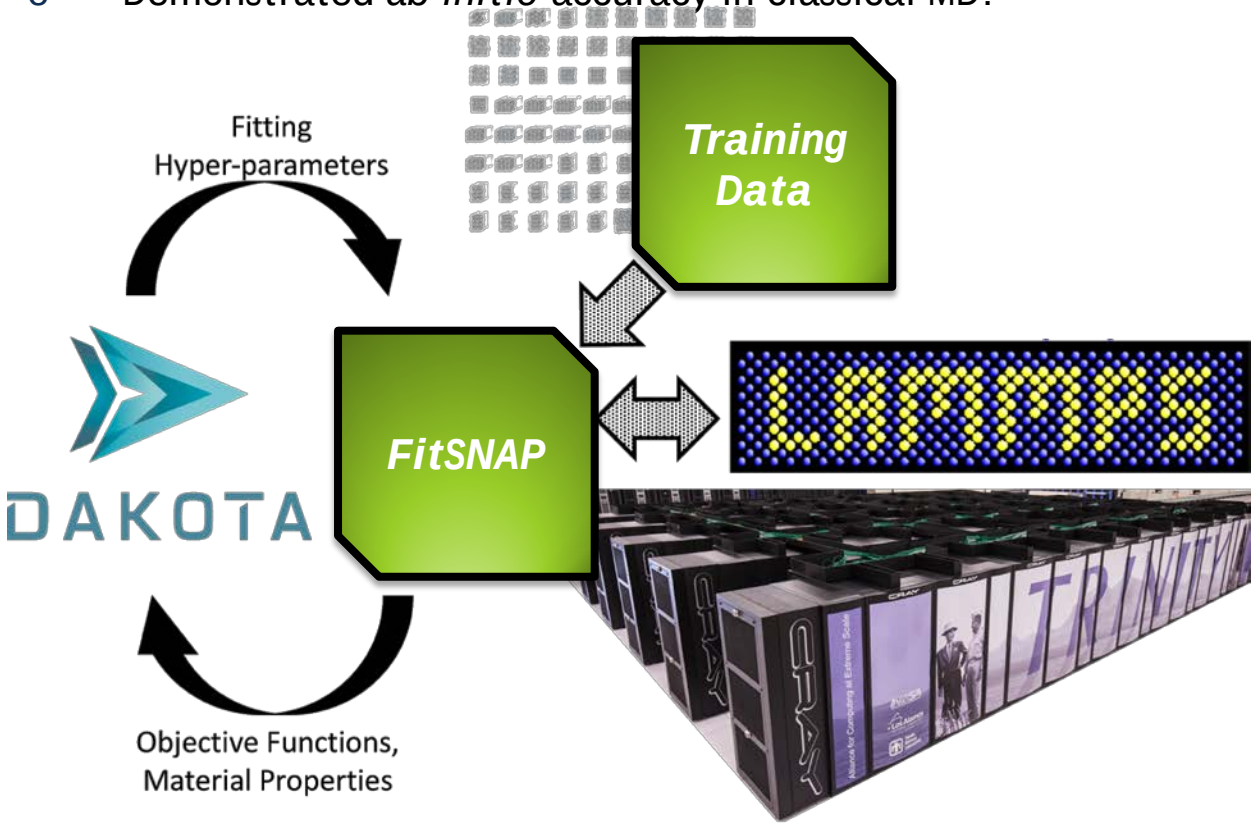
http://exascaleproject.org
"Episode 44: ECP Team Reengineers Materials Simulation Code, Achieves Atypical Performance Increase"

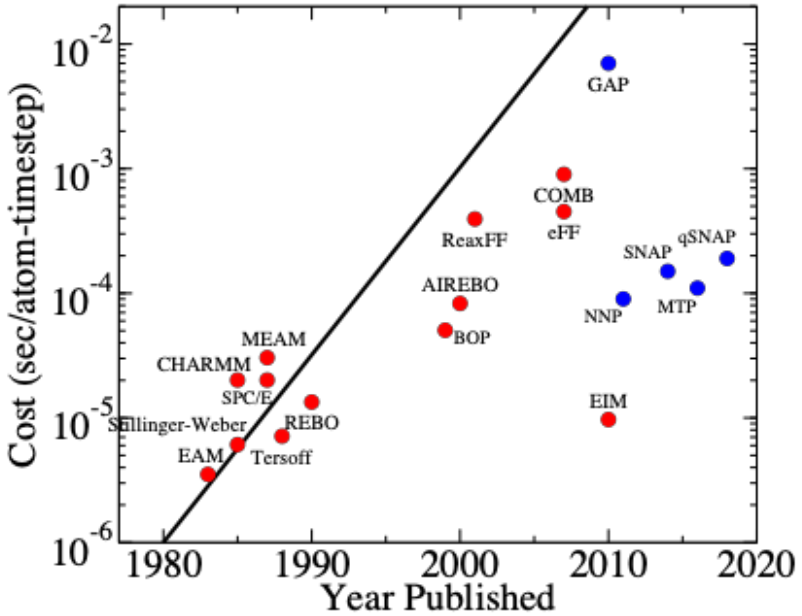
EXAALT Performance Improvements

- Joint effort by Aidan Thompson (EXAALT), Stan Moore (CoPA), Rahul Gayatri (NESAP), Sarah Anderson (Cray), Evan Weinberg (NVIDIA)
- Created stripped-down proxy code (TestSNAP)
- Completely rewrote TestSNAP to reduce flops and memory
- Explored many different GPU strategies, using OpenACC and CUDA
- Ported best implementation back to production code with Kokkos
- Other kernels required for ParSplice (time stepping, minimization) also implemented on accelerators
- These kernels account for essentially all of the flops in base challenge

SNAP: Spectral Neighbor Analysis Potential

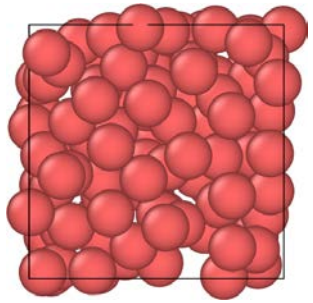
- ML interatomic potential (IAP) have three critical parts:
  - Descriptors of the local environment
  - Energy and force functions expressed in the descriptors
  - Training(regression method) on large amount of 'ground truth' energies and forces
- Demonstrated *ab initio* accuracy in classical MD!





Chemistry	Application	# Configs	Year
Ta	Plasticity	363	2014
In,P	Intrinsic Defects	665	2015
W,He	Fusion Energy	2,800	2017
W,Be	Fusion Energy	25,000	2018
Actinides	Shock Physics	20,000	2018
W,H	Fusion Energy	40,000	2019
C	Shock Physics	10,000	2019
HEA	Add. Manufact.	>20,000	2019
W,N	Fusion Energy	35,000	2019

SNAP: Spectral Neighbor Analysis Potential



Geometric descriptors of atomic environments

Energy as a function of geometric descriptors

$$E^{SNAP} = \sum_{i=1}^{N} E_i^{SNAP} + \sum_{j<i}^{N} \phi_{ij}^{rep}\left(r_{ij}\right)$$

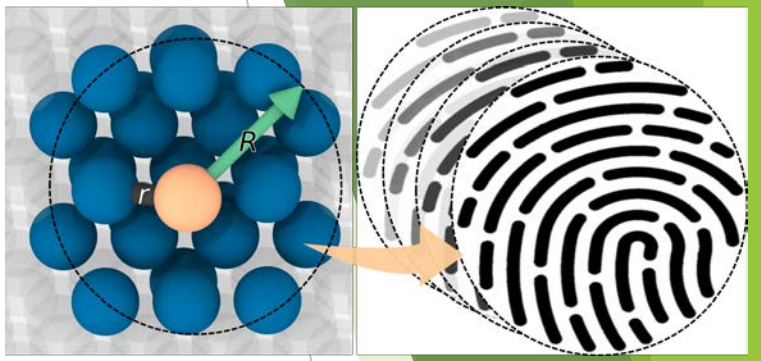
$$E_i^{SNAP} = \beta_0 + \sum_{k \in \{J<J_{\max}\}} \beta_k B_k^i$$

- **GAP (Gaussian Approximation Potential):** Bartok, Csanyi et al., *Phys. Rev. Lett, 2010.* Uses 3D neighbor density bispectrum and **Gaussian process regression**
- **SNAP (Spectral Neighbor Analysis Potential):** Our SNAP approach uses GAP's neighbor bispectrum, but replaces Gaussian process with **linear regression**.
  - More robust
  - Lower computational cost (training and predicting)
  - Decouples MD speed from training set size
  - Enables large training data sets, more bispectrum coefficients
  - Straightforward sensitivity analysis
  - Fast

SNAP Bispectrum Components

- Neighbors of each atom are mapped onto unit sphere in 4D

$$3D\ Ball\colon (r,\theta,\phi), r < R_{cut} \implies 4D\ Sphere\colon (\theta_0,\theta,\phi), \theta_0 = \frac{r}{R_{cut}}\pi$$



- Expand density around each atom in a basis of 4D hyperspherical ***harmonics***,

$$\rho_i(\mathbf{r}) = \delta(\mathbf{0}) + \sum_{r_{i'}<R_{cut}} f_c(r_{i'}) w_{i'} \delta(\mathbf{r}_{i'})$$

- Bispectrum components of the 4D hyperspherical harmonic expansion are used as the geometric descriptors of the local environment
  - Preserves universal physical symmetries
  - Rotation, translation, permutation
  - Size-consistent (extensible)

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

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

SNAP Force Calculation

Function Calc_dBdR(i, j):

for (η, η_1, η_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 sizes <= 14

Original Kokkos Version of SNAP

- Christian Trott (SNL) created the original Kokkos version in the ExaMiniMD proxy app
- Used advanced Kokkos features: three levels of hierarchical parallelism and shared scratchpad memory (global)
- Very memory compact
- Stan Moore (SNL) ported this version to the LAMMPS KOKKOS package
- Not clear of the possible improvements, if any…

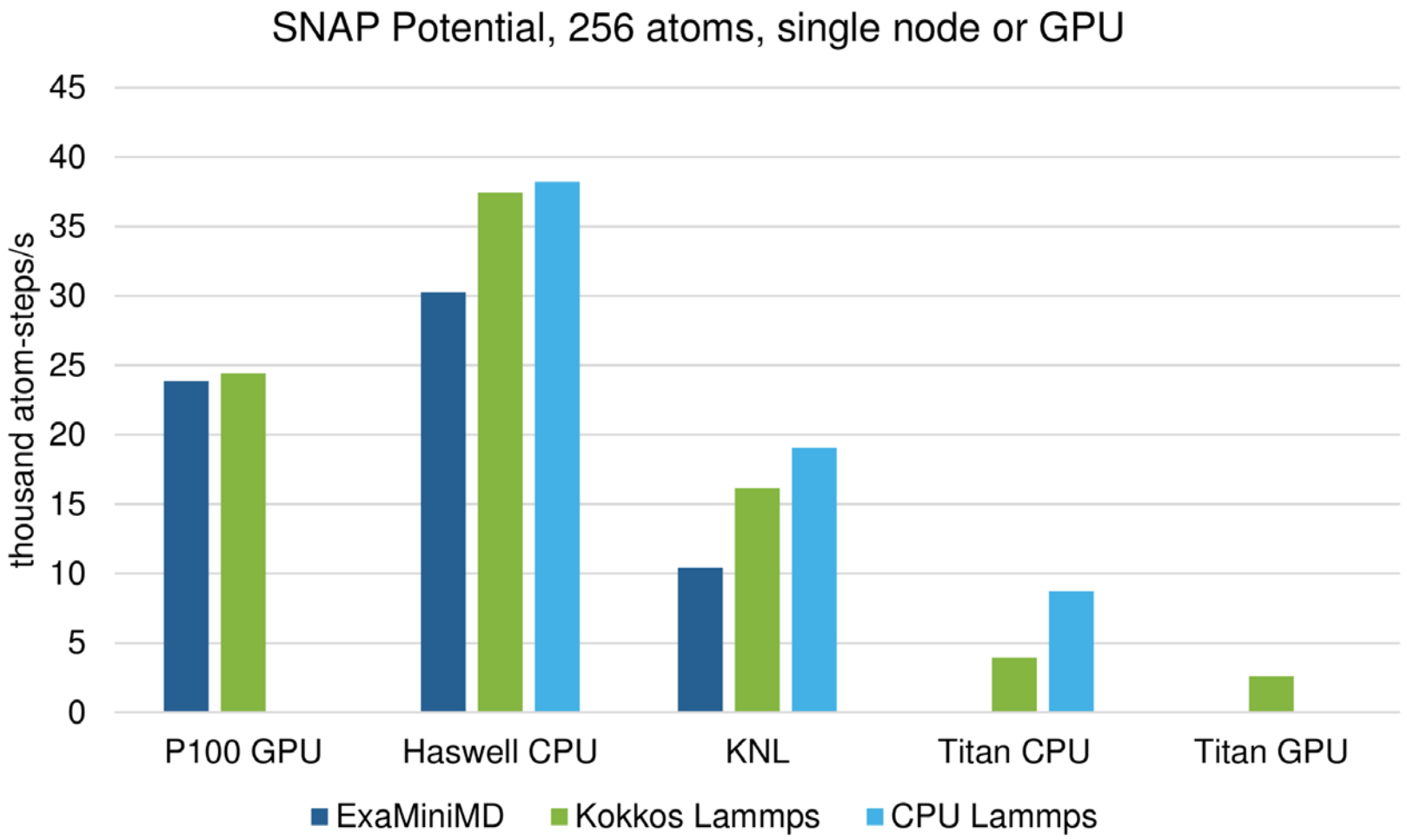
```
parallel_for(num_atoms) {  // Use TeamPolicy (level 1) here
 // Count Neighbors in Cutoff
 parallel_reduce(num_neighs)  // Use TeamThreadRange (level 2) here
 // Build reduced NeighborList
 if (team_rank==0)
  parallel_scan(num_neighs)  // Use ThreadVectorRange (level 3) here
 team_barrier
 // Compute U_i
 compute_ui ()
 team_barrier
 // Compute Z_i
 compute_zi ()
 team_barrier

 parallel_for(num_neighs_reduced) {  // Use TeamThreadRange (level 2) here
  // Compute derivative of U_i
  compute_duidrj()  // Use ThreadVectorRange (level 3) inside
  // Compute derivative of B_i
  compute_dbidrj()  // Use ThreadVectorRange (level 3) inside
  // Update force on i and j
  for(k<ncoeff)
   fij += dbvec(k)

  f(i) += fij;
  f(j) += fij;
 }
 ... // Compute energy
}
```

```
compute_zi() {
 parallel_for(num_idx) { // Use TeamThreadRange (level 2) here
  parallel_for(jxj) { // Use ThreadVectorRange (level 3) here
   for(ma1<ma1_max)
    for(mb1<mb1_max) ...
  }
 }
}
```

SNAP Performance before Jan2019 Hackathon



SNAP Benchmarking on Summit

- EXAALT FOM benchmark uses 205 bispectrum coefficients, tungsten crystal
- Mira (IBM BG/Q) FOM baseline: 0.182 Katoms-steps/s/node * 49152 Mira nodes
- On Summit, run 6 GPU + 1 CPU (36 cores) replicates per node
- 2018 LAMMPS performance on Summit: 33.7 Katom-steps/s/node * 4608 Summit nodes: projected **17.4x faster than Mira baseline**

KPP: Challenges

- SNAP fraction-of-peak performance has been steadily declining on most hardware
- Recommendation from 2018 EXAALT review was to focus on SNAP GPU performance
- Developed a collaboration between EXAALT, CoPA, and NERSC/NESAP in order to address this risk

Architecture	Year	Normalized fraction of peak
Intel SandyBridge/Chama	2012	1.0
IBM PowerPC/Mira	2012	0.23
AMD CPU/Titan	2013	0.71
NVIDIA K20X/Titan	**2013**	**0.037**
Intel Haswell/Trinity	2016	0.47
Intel KNL/Trinity	**2016**	**0.080**
NVIDIA P100/SNL testbed	**2016**	**0.077**
Intel Broadwell/Serrano	2017	0.39
NVIDIA V100/SNL testbed	**2018**	**0.093**

Benchmarks for 2000 SNAP atoms

TestSNAP – standalone independent SNAP module

- Standalone SNAP kernel mini app derived from CPU version (90% similar)
- Proxy in memory and computation
- Included correctness check
- Blank slate to try something new without biases
- Initial OpenACC port very slow

Kernel refactor

```
for atom i {
    for neighbor j {
        function_1()
        function_2()
        ...
    }
}
```

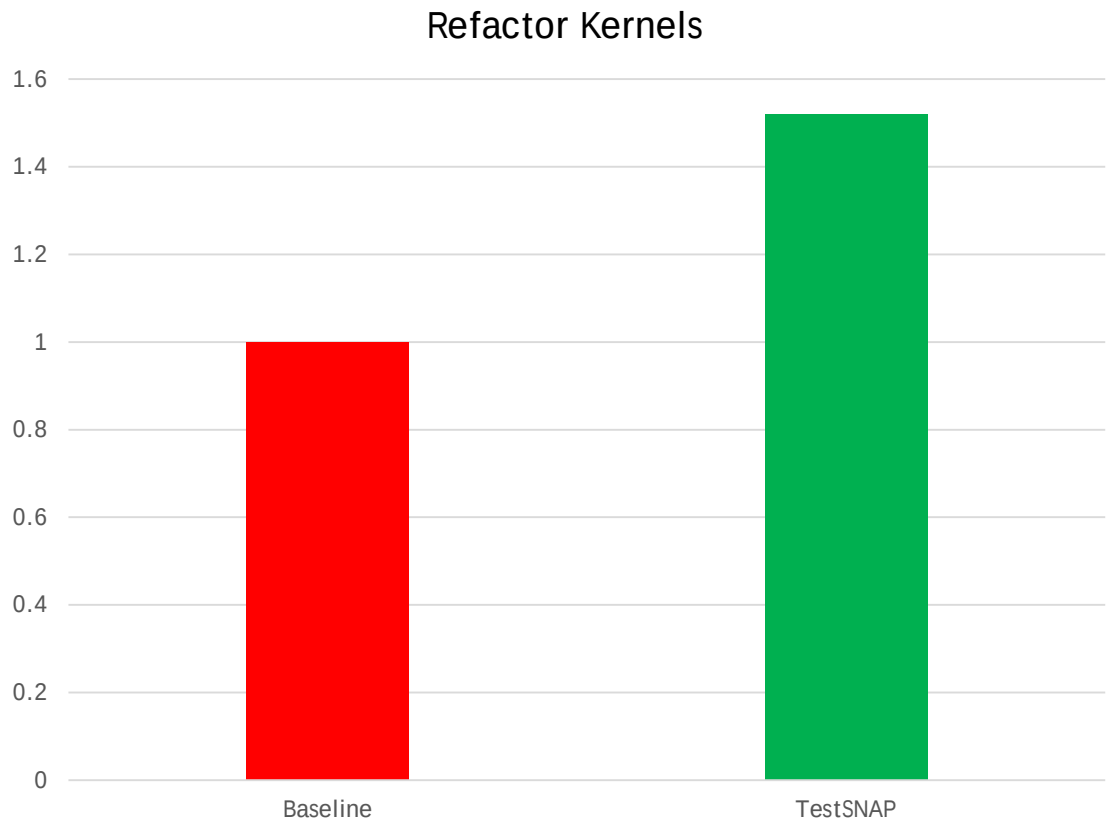
```
for atom i {
    for neighbor j {
        function_1()
    }
}
```

```
for atom i {
    for neighbor j {
        function_2()
    }
}
```

```
...
```

- Broke one large kernel into many smaller kernels, helps reduce register pressure
- Reordered loop structure
- Works because kernel launch latency negligible compared to kernel execution time
- Atypical: usually need to **fuse** kernels instead
- Greatly increases memory footprint, must store intermediate results

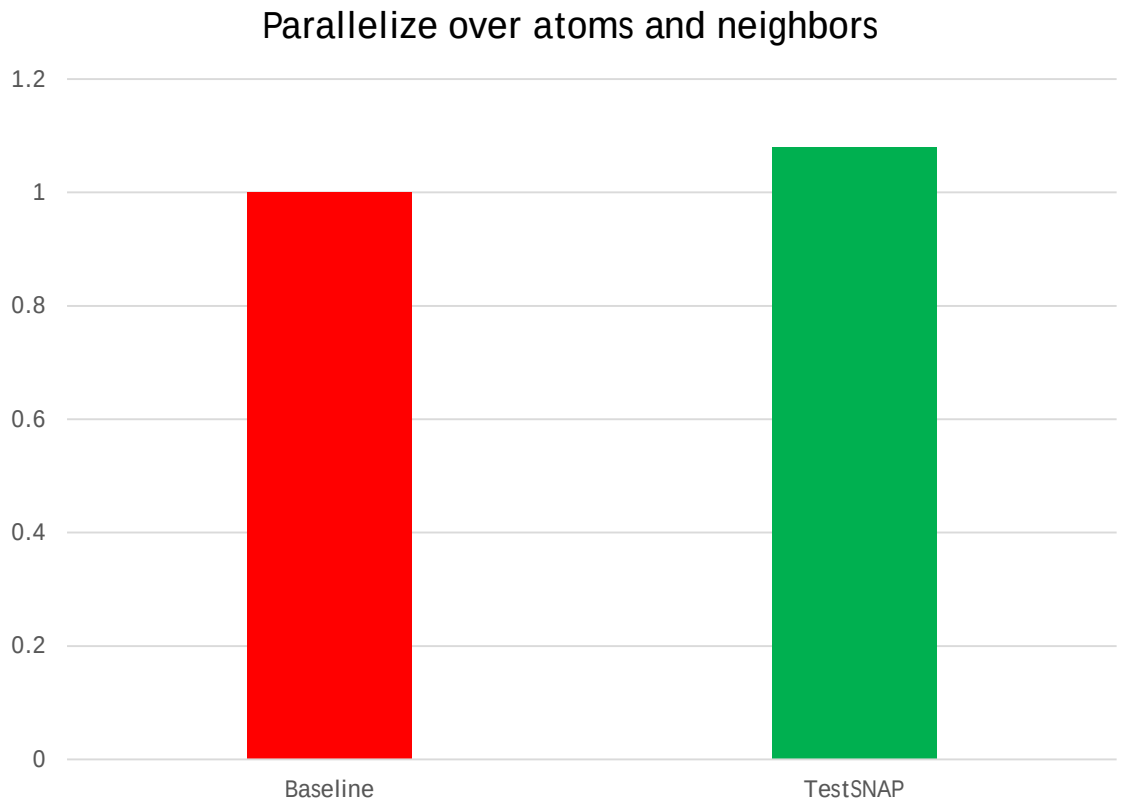
During Jan~2019 hackathon



(Results for V100 GPU)

- Break up the compute kernels
- Store atom specific information across kernels
- Increases memory footprint
- Distribute the atom specific work in each kernel over the threadblocks and threads of a threadblock

Parallelize over atoms and neighbors



Could only fit the smaller problem size in the GPU memory (16GBs)

Post Jan-2019 hackathon

- Nick Lubbers (EXAALT, LANL) suggested re-arranging the order of summation (Y-array trick)
- Memory footprint was reduced by 20x by compacting multi-dimensional arrays in to simple lists
- Y-array trick was first applied to the sequential memory and merged into Kokkos-LAMMPS in June 2019
- Allowed the FOM 2J14 benchmark into the memory

Y-array Trick

- Old Algorithm: Dominated by neighbor loop

$$\mathbf{Z}^j_{j_1 j_2} = \mathbf{U}^{j_1} \cdot \mathbf{H}^j_{j_1 j_2} \cdot \mathbf{U}^{j_2},$$

$$\nabla_j E^i_{SNAP} = \sum_{j_1 j_2 j} \beta^j_{j_1 j_2} \mathbf{Z}^j_{j_1 j_2} : \nabla \mathbf{U}^j$$

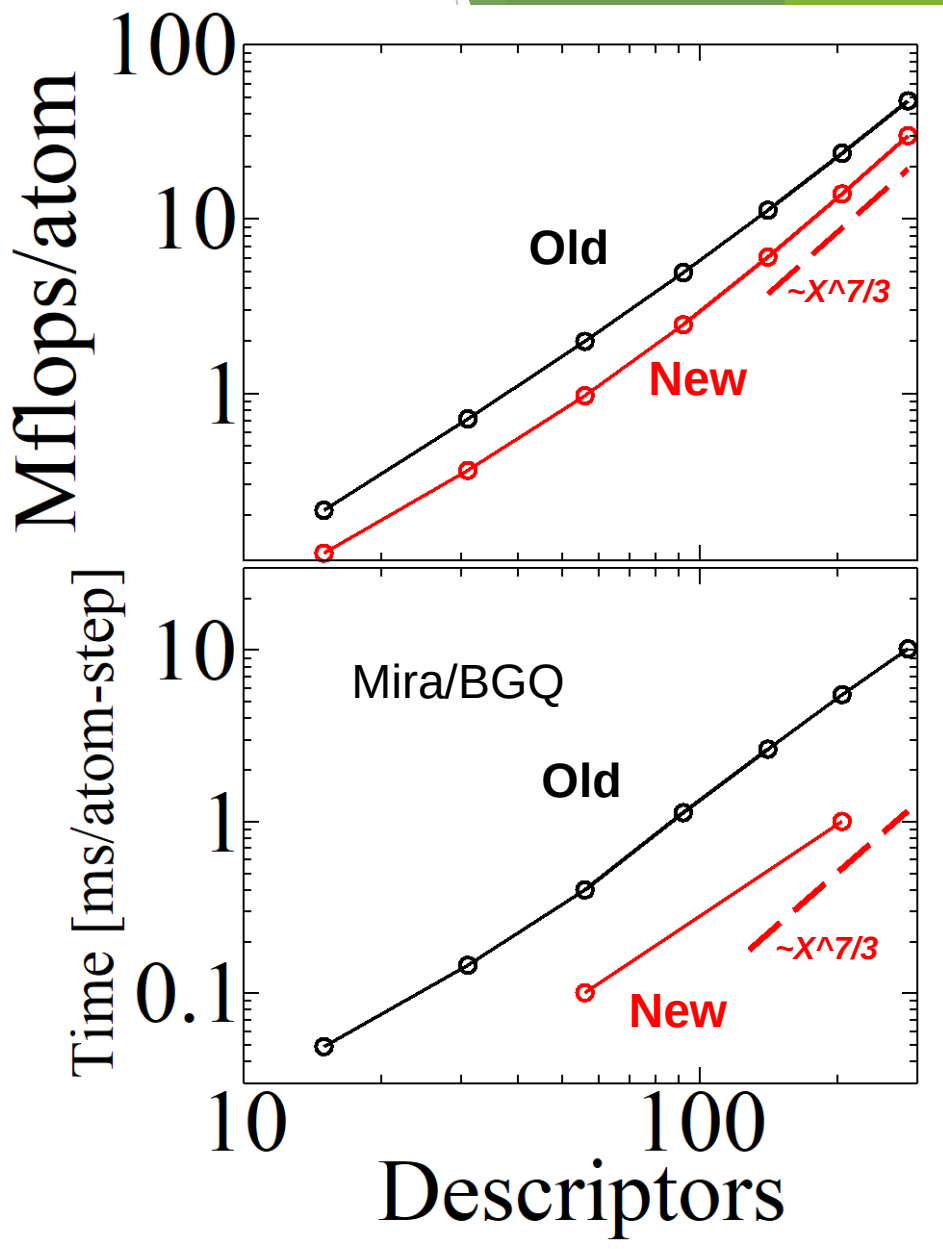
O(J^5) x Neighs

- New algorithm: Pre-compute Y factors outside neighbor loop

$$\mathbf{Y}^j = \sum_{j_1 j_2} \beta^j_{j_1 j_2} \mathbf{Z}^j_{j_1 j_2},$$

$$\nabla_j E^i_{SNAP} = \sum_j \mathbf{Y}^j : \nabla \mathbf{U}^j$$ **O(J^3) x Neighs**

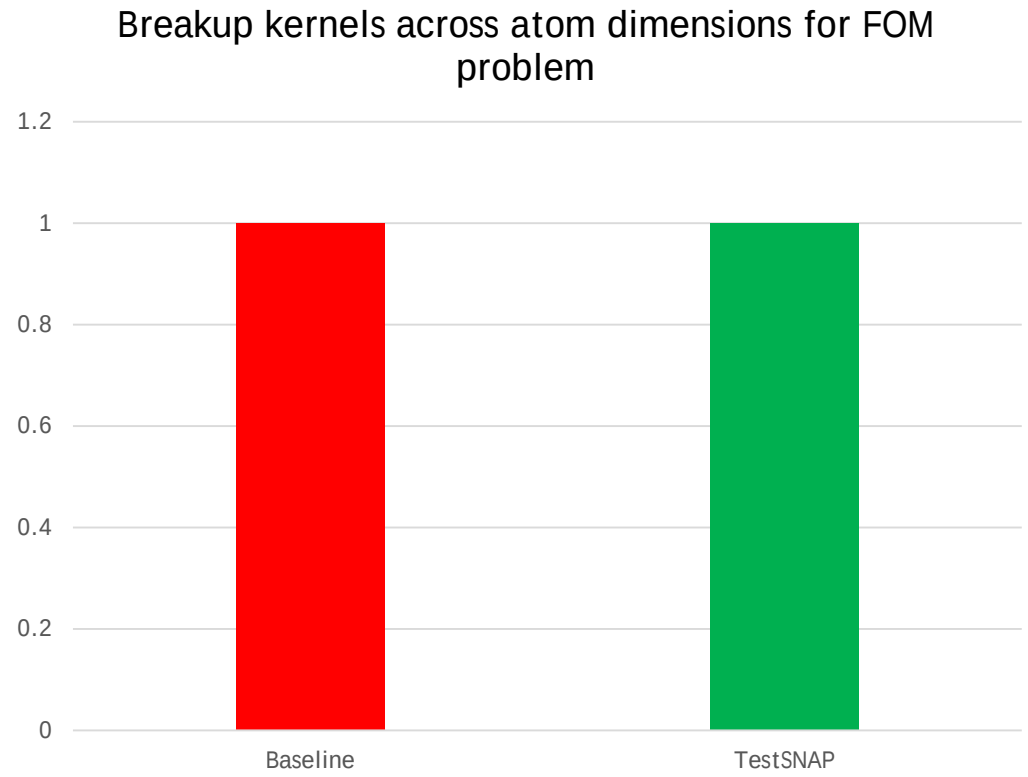
- Cost of neighbor loop is now negligible
- Dominated by Y pre-compute O(J^7)
- 2x reduction in Flops/atom
- 5.5x speedup on Mira/BGQ!!



Y-array trick in TestSNAP

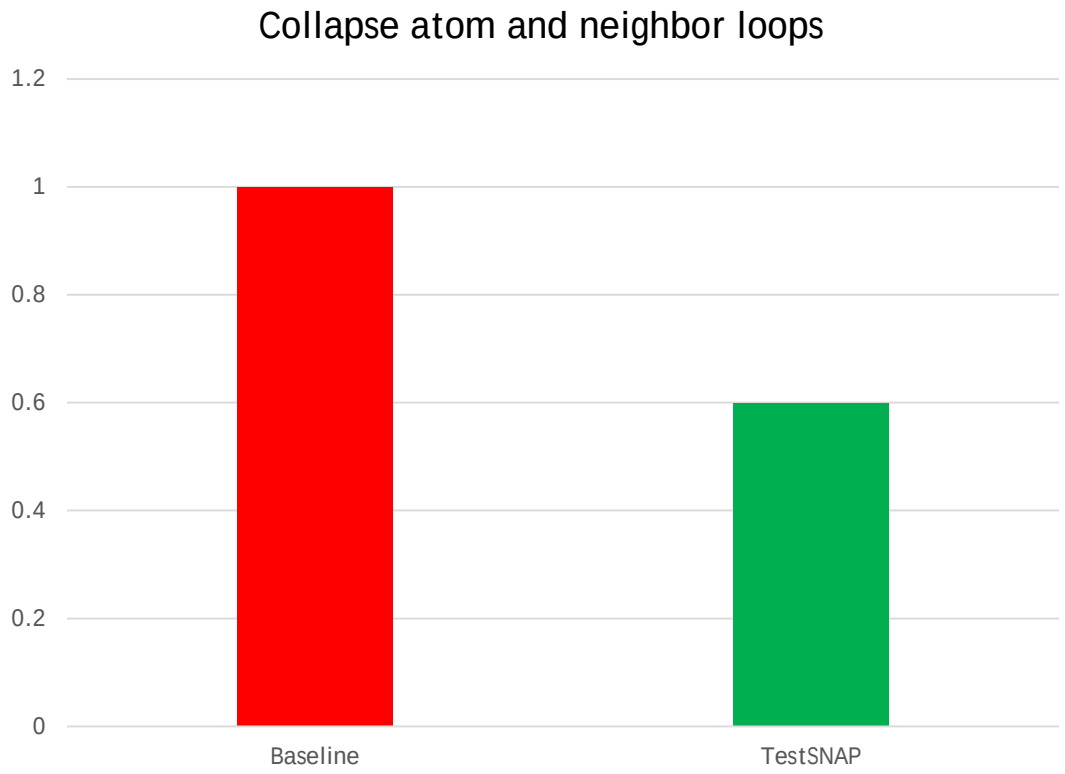
- New version can now fit in the memory for large problem size
- Exposed parallelism initially via
  1. OpenACC
  2. Cuda (to expose higher degree of parallelism with multi dimensional grid generation)
  3. Kokkos (Moved to a kokkos implementation to be consistent with the LAMMPS implementation)

Apply smaller problem optimizations on bigger problem size



- With the Y-array trick, 2J14 could now be fit in the GPU memory
- We applied the 2J8 tricks to the FOM (2J14) benchmark
- Initially parallelized over atom loops

Collapse atom and neighbor loops

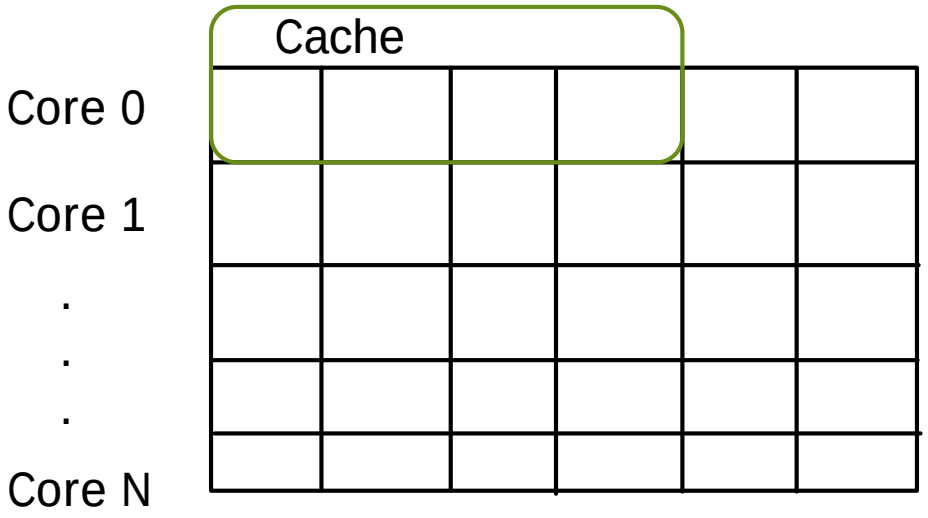


- Distribute the works across atom and neighbor loops
  - The memory-footprint was now reduced to 12GB

Row major vs Column Major

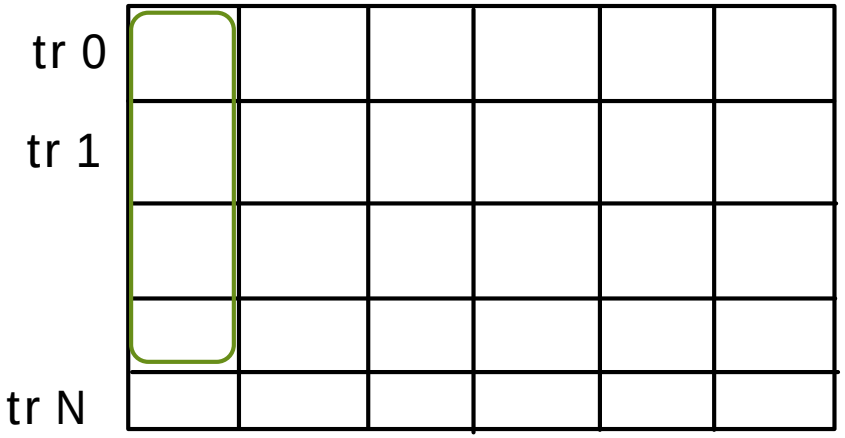
Row major

Avoid false sharing in cache && improve cache utilization



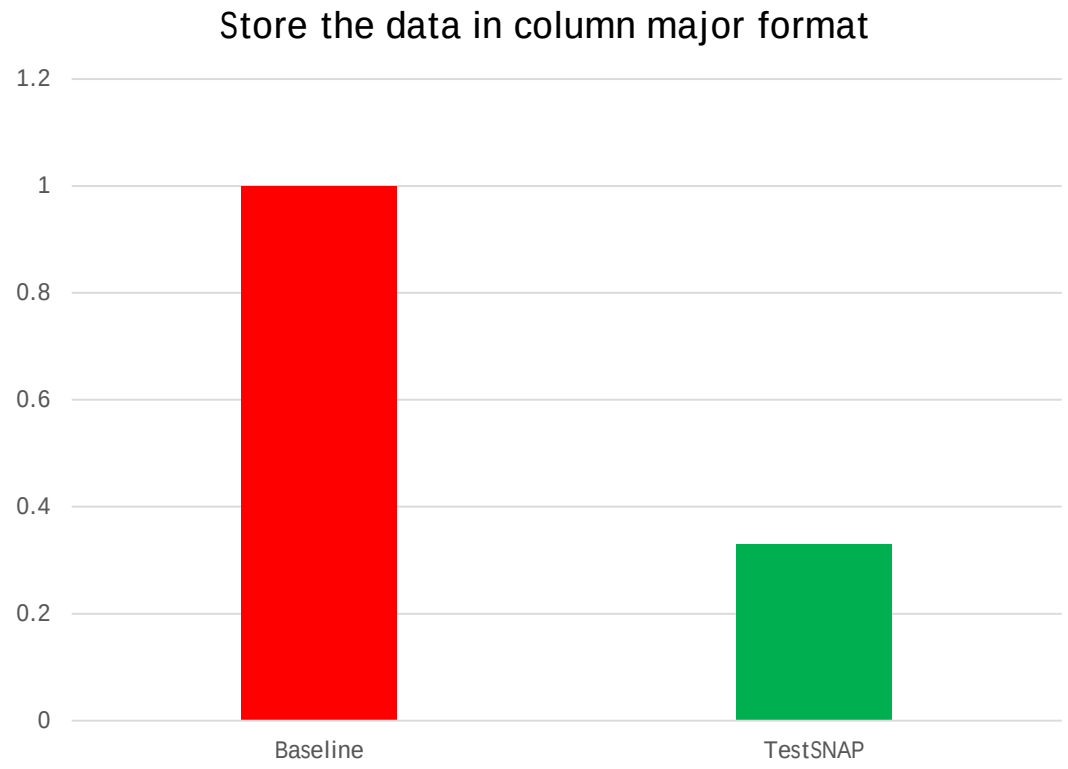
Column major

Promote memory coalescing



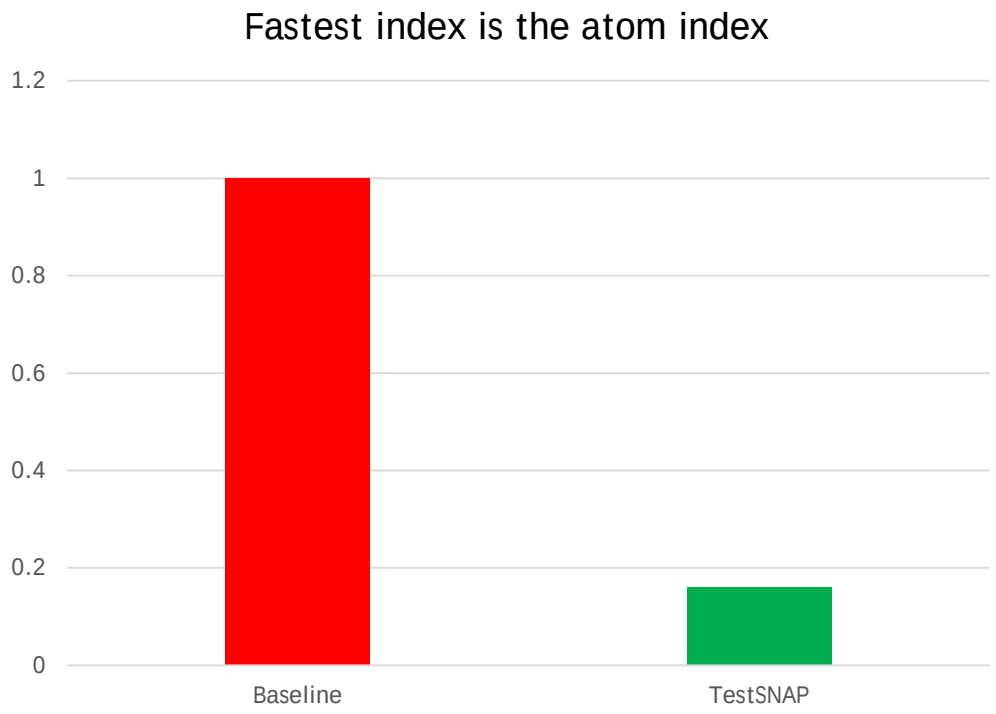
- Memory access patterns
  - Row major - C style access, optimal for CPUs
  - Column Major – FORTRAN style access, optimal for GPUs
    - Gave a 2X performance improvement

Column major data access pattern



- Accessing the data in a column major fashion gave us a ~2X performance boost

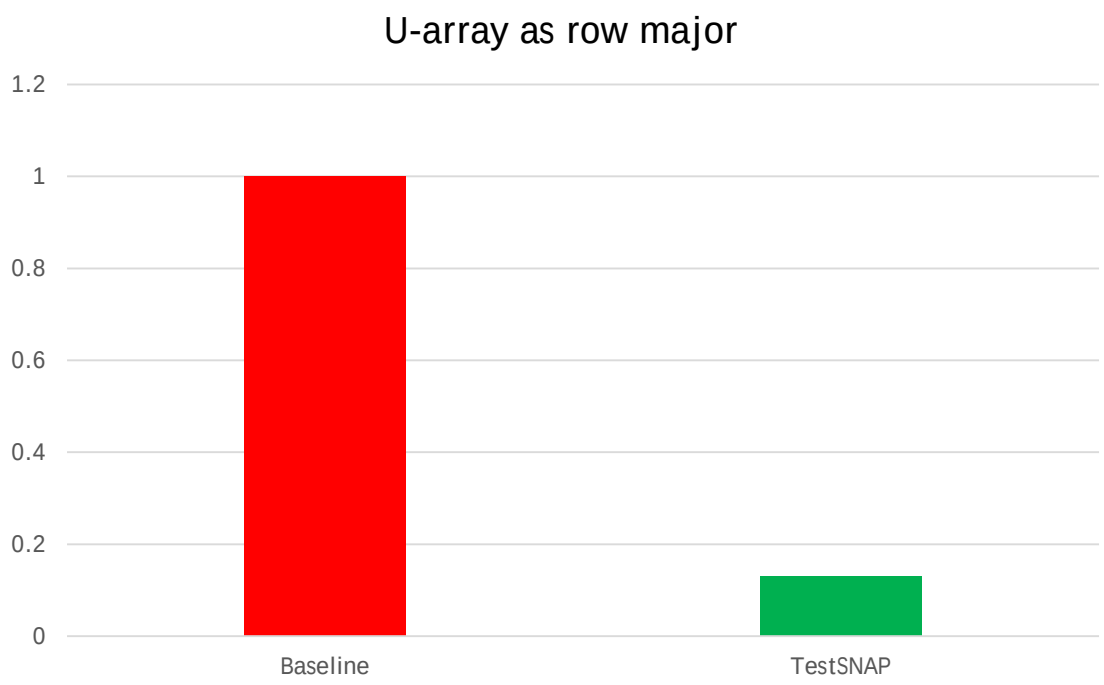
Reverse the loop order




```
for neighbor j {
    for atom i {
        ...
    }
}
```

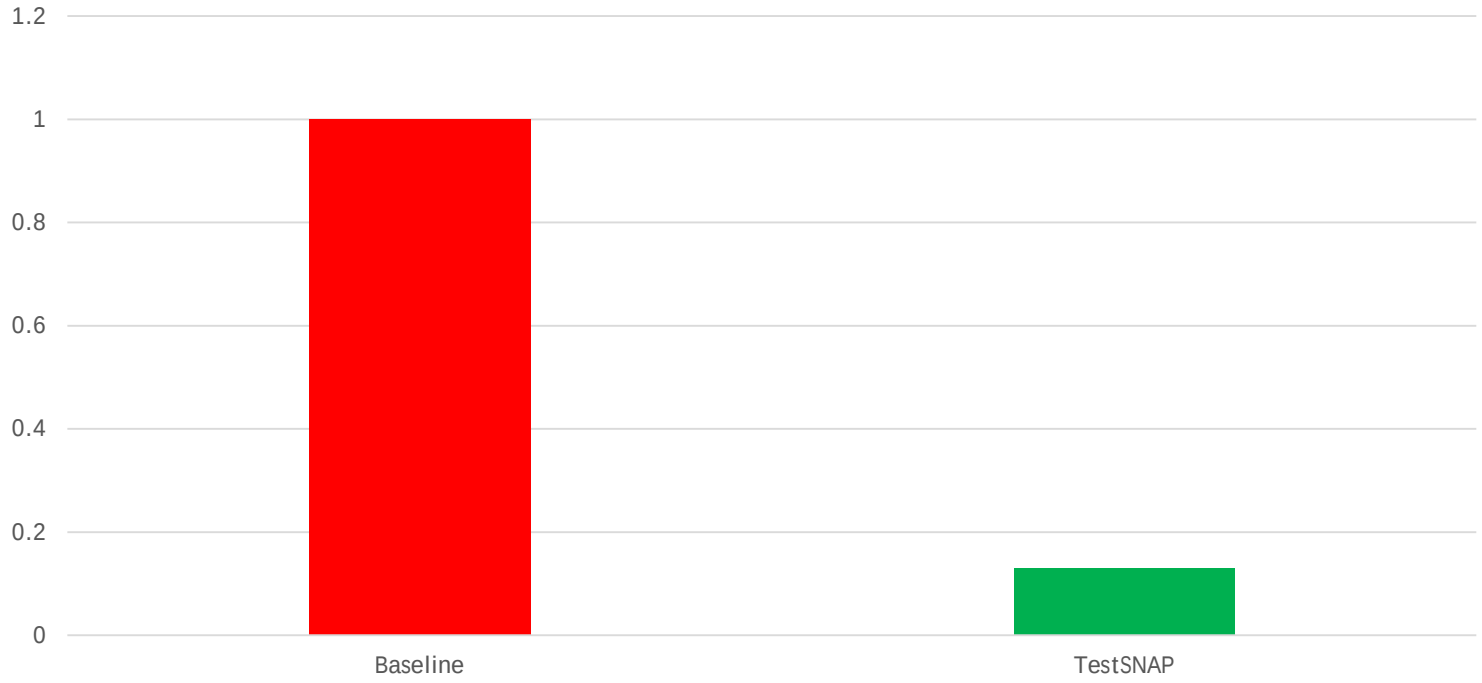
- Reverse the loops to make atom index as the fastest moving index
  - Gave a 2x performance boost

Row major access for one of the data structures



- Increased runtime of kernel with atomics by 10%
- Improved runtime of another kernel by 25%
- Overall gave a 15% improvement

TestSNAP was ~7.5x faster than Baseline LAMMPS implementation of SNAP



Status at the end of the July 2019 hackathon

July2019 Hackathon

- Integrated several optimizations from TestSNAP into the Kokkos module in LAMMPS
  - Broke up kernels: actually made the code significantly slower at first (could be due to suboptimal memory layout)
  - Requires storing intermediate results between kernels for atom/neighbor pairs (extreme memory overhead that requires chunking up the loops)
  - Column major
  - Loop reorder: led to significant speedup, but not possible without breaking up the kernels
  - Removed advanced Kokkos features: used flat parallelism instead of hierarchical and no shared scratchpad memory
- New performance on Summit: 175.1 Katom-steps/s/node * 4608 Summit nodes: projected **90x faster than Mira baseline (5.2x speedup)**

post July2019 Hackathon

- Integrated more optimizations from TestSNAP into the Kokkos module in LAMMPS
  - Refactored loop indices data structures to use complex numbers and be muti-dimensional arrays instead of arrays of structs
  - Replaced more hierarchal parallelism with flat parallelism and exposed additional parallelism by collapsing loops
  - Changed data layout between kernels via transpose
- Current master LAMMPS on Summit: 262.0 Katom-steps/s/node * 4608 Summit nodes: projected **134x faster than Mira baseline (7.7x speedup)**

post July2019 Hackathon (TestSNAP)

- Implemented a transpose routine for U-array to take advantage of column major updates for kernels with atomics and row major updates for other kernels
- Used double2 vector type from CUDA to optimize on 128 bit load/stores
- Improved the algorithm and reduced the memory footprint to 3.5GBs
- **> 2x speedup in TestSNAP since July 2019 hackathon**

Looking Forward

- Evan Weinberg (NVIDIA) has added additional optimizations to Kokkos SNAP in LAMMPS:
  - refactoring algorithms to avoid thread atomics
  - use of Kokkos hierarchal parallelism and scratch memory
- New version being tested by CoPA and EXAALT project members
- Not yet released, but should be merged into master LAMMPS soon
- Unreleased LAMMPS on Summit: 407.7 Katom-steps/s/node * 4608 Summit nodes: projected **210x faster than Mira baseline (12x speedup)**
- **And we are not done yet!**

Lessons Learned

- Performance gains came from both algorithmic improvements (i.e. changes to the serial code) and implementation improvements targeting specific hardware (i.e. changes to the CUDA/Kokkos code)
- Small proxy app with correctness check is invaluable, allows rapid prototyping
- Profiling helps to know where to focus next
- Pay attention to memory access on GPUs

Supplementary Material

Source Code

TestSNAP: https://gitlab.com/NESAP/EXAALT/q1-2019-hack-a-thon

LAMMPS: https://github.com/lammps/lammps/tree/master/src/SNAP

FitSNAP: https://github.com/FitSNAP/FitSNAP

Papers

C.R. Trott, S.D. Hammond, A. P. Thompson, "SNAP: Strong scaling high fidelity molecular dynamics simulations on leadership-class computing platforms," *Supercomputing, (Lecture Notes in Computer Science)*, **8488** 19 (2014).

A. P. Thompson , L.P. Swiler, C.R. Trott, S.M. Foiles, and G.J. Tucker, "Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials," *J. Comp. Phys.*, **285** 316 (2015).

M. A. Wood, and A. P. Thompson, "Extending the accuracy of the SNAP interatomic potential form," *J. Chem. Phys.*, **148** 241721 (2018).

M. A. Wood, M. A. Cusentino, B.D. Wirth and A.P. Thompson, Phys. Rev. B 99, 184305 (2019)

Reports

"Episode 44: ECP Team Reengineers Materials Simulation Code, Achieves Atypical Performance Increase", ECP Highlight, September 2019, http://exascaleproject.org, https://soundcloud.com/exascale-computing-project/episode-44-ecp-team-reengineers-materials-simulation-code-achieves-atypical-performance-increase

ECP CoPA Milestone Report, "Deploy ExaMiniMD optimizations into LAMMPS, Stan Moore, Christian Trott, Steve Plimpton, June (2018)