Exceptional service in the national interest

Sandia National Laboratories

Accelerating LAMMPS Performance

Stan Moore

2017 LAMMPS Workshop and Symposium

Breakout session: Acceleration Packages

Albuquerque, NM

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

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. SAND2017-8029 C

LAMMPS Resources for Performance Acceleration

Sandia National Laboratories

- **Hardware support**
  - CPU including OpenMP
  - GPU via Cuda
  - KNL via OpenMP
- **Website: Benchmarking page (discussed in this session)**
  - input files, Makefiles, run commands, log files, plots & tables
- **Distro**
  - bench directory
- **Manual**
  - Section 5 = Accelerating LAMMPS performance
  - Section 5.3.1 = GPU package
  - Section 5.3.2 = USER-INTEL package
  - Section 5.3.3 = KOKKOS package
  - Section 5.3.4 = USER-OMP package
  - Section 5.3.5 = OPT package
  - Section 8 = Performance and Scalability

LAMMPS Resources (cont.)

Sandia National Laboratories

- **Packages**
  - GPU, KOKKOS, OPT, USER-INTEL, USER-OMP
- **Makefiles in src/MAKE/OPTIONS dir**
  - Makefile.kokkos, several variants: Cuda, KNL, OpenMP
  - Makefile.intel, several variants: CPU and KNL
  - Makefile.omp
- **Commands**
  - balance, fix balance, processors, run style verlet/split
- **Example dirs**
  - balance

Outline of Topics

Sandia National Laboratories

- **LAMMPS accelerator packages**
  - Overview
  - How and when to use them
- **New benchmarking website**
- **Recent work to improve LAMMPS performance**
- **Other performance considerations**
- **Discussion**

Please feel free to ask questions, give suggestions, or discuss during the presentation

LAMMPS Accelerator Packages

Sandia National Laboratories

- Modern HPC platforms such as multi-core CPUs, Xeon Phis, and GPUs often need to use special code (e.g. OpenMP or CUDA) to allow LAMMPS to perform well
- LAMMPS has 5 accelerator packages that contain specialized code:
  - OPT
  - USER-OMP
  - USER-INTEL
  - GPU
  - Kokkos

OPT Package

Sandia National Laboratories

- Developed by James Fischer (High Performance Technologies), David Richie, and Vincent Natoli (Stone Ridge Technologies)
- Methods rewritten in C++ templated form to reduce the overhead due to if tests and other conditional code
- Code also vectorizes better than the regular CPU version
- Contains 9 pair styles:
  - pair_eam_alloy
  - pair_eam_fs
  - pair_eam
  - pair_lj_charmm_coul_long
  - pair_lj_cut_coul_long
  - pair_lj_cut
  - pair_lj_cut_tip4p_long
  - pair_lj_long_coul_long
  - pair_morse

Compiling and Running OPT Package

Sandia National Laboratories

- In *src* directory, "make yes-opt"
- Compile LAMMPS
- Run with 8 MPI: "mpiexec -np 8 ./lmp_exe -in in.lj -sf opt"
- -sf opt is the *suffix* style: automatically appends /opt onto anything it can
- For example, "pair_style lj/cut" becomes "pair_style lj/cut/opt"

USER-OMP Package

Sandia National Laboratories

- Developed by Axel Kohlmeyer (Temple U)
- Uses OpenMP to enable multithreading on CPUs or Xeon Phis
- Extensive LAMMPS coverage (108 pair styles, 30 fixes, molecular topology bonds, angles, etc., PPPM, Verlet & rRESPA)
- Best for a small number of threads (2-4)
- MPI parallelization in LAMMPS is almost always more effective than OpenMP in USER-OMP on CPUs
- When running with MPI across multi-core nodes, MPI often suffers from communication bottlenecks and using MPI+OpenMP per node can be faster
- The more nodes per job and the more cores per node, the more pronounced the bottleneck and the larger the benefit from MPI+OpenMP

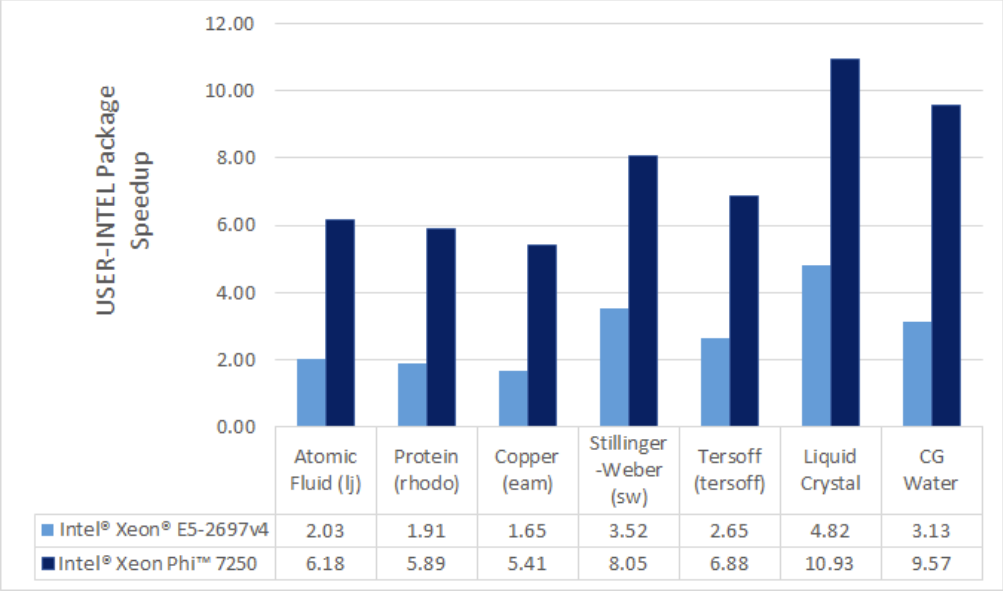
Compiling and Running USER-OMP Package

Sandia National Laboratories

- In *src* directory, "make yes-user-omp"
- Add -fopenmp to the Makefile
- Compile LAMMPS
- Run with 2 MPI and 2 OpenMP threads: "mpiexec -np 2 -v OMP_NUM_THREADS=2 ./lmp_exe -in in.lj -sf omp"

USER-INTEL Package

- Developed by Mike Brown (Intel)
- Allows code to vectorize and run well on both Intel CPUs (with or without threading) and on Xeon Phis
- Can also be used in conjunction with the USER-OMP package
- Supports 11 pair styles, 5 fixes, some bonded styles, PPPM
- Supports single, double, and mixed precision modes



Compiling and Running USER-INTEL Package

Sandia National Laboratories

- Need to use a recent version of the Intel compiler
- Use a Makefile in /src/MAKE/OPTIONS/ such as Makefile.intel_cpu_openmpi
- In /src "make yes-user-intel" and "make yes-user-omp"
- Compile LAMMPS
- To run using 2 MPI and 2 threads on a Intel CPU: "mpiexec -np 2 -v OMP_NUM_THREADS=2 ./lmp_exe -in in.lj -pk intel 0 omp 2 mode mixed -sf intel"
- -pk is the package command

GPU Package

Sandia National Laboratories

- Developed by Mike Brown and Trung Nguyen (ORNL)
- Designed for one or more GPUs coupled to many CPUs
- Pair runs on GPU, fixes/bonds/computes run on CPU
- Atom-based data (e.g. coordinates, forces) move back and forth between the CPU(s) and GPU every timestep
- Supports 49 pair styles, PPPM
- Asynchronous force computations can be performed simultaneously on the CPU(s) and GPU.
- Allows for GPU computations to be performed in single, double precision, or mixed precision mode
- Provides NVIDIA and more general OpenCL support

Compiling and Running GPU Package

Sandia National Laboratories

- First compile GPU library in lib/gpu (make -f Makefile.linux.mixed)
- In *src* directory, "make yes-gpu"
- Compile LAMMPS
- Run with 16 MPI and 4 GPUs: "mpiexec -np 16 ./lmp_exe -in in.lj -sf gpu -pk gpu 4"

Kokkos

Sandia National Laboratories

- Abstraction layer between programmer and next-generation platforms
- Allows the same C++ code to run on multiple hardwares (GPU, Xeon Phi, etc.)
- Core developers are Carter Edwards and Christian Trott (Sandia)
- 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

- Used on top of existing MPI parallelization (MPI + X)
- Open-source, can be downloaded at https://github.com/kokkos/kokkos

Kokkos Package

Sandia National Laboratories

- Developed by Christian Trott, Stan Moore, Ray Shan (Sandia) and others
- Supports OpenMP and GPUs
- Scales to many OpenMP threads
- Designed for one-to-one GPU to CPU ratio
- Designed so that everything (pair, fixes, computes, etc.) runs on the GPU, minimal data transfer from GPU to CPU
- Currently only double precision is supported
- Supports only newer NVIDIA GPUs

LAMMPS Kokkos Package

Sandia National Laboratories

- 6 atom styles: angle, atomic, bond, charge, full, molecular
- 34 pair styles: buck/coul/cut, buck/coul/long, buck, coul/cut, coul/debye, coul/dsf, coul/long, coul/wolf, eam/alloy, eam/fs, eam, lj/charmm/coul/charmm/implicit, lj/charmm/coul/charmm, lj/charmm/coul/long, lj/class2/coul/cut, lj/class2/coul/long, lj/class2, lj/cut/coul/cut, lj/cut/coul/debye, lj/cut/coul/dsf, lj/cut/coul/long, lj/cut, lj/expand, lj/gromacs/coul/gromacs, lj/gromacs, lj/sdk, morse, sw, reax/c, table, tersoff, tersoff/mod, tersoff/zbl, vashishta
- 12 fix styles: deform, langevin, momentum, nph, npt, nve, nvt, qeq/reax, reaxc/bonds, reaxc/species, setforce, wall/reflect
- 1 compute style: temp
- 2 bond styles: fene, harmonic
- 2 angle styles: charmm, harmonic
- 2 dihedral styles: charmm, opls
- 1 improper style: harmonic
- 1 kspace style: pppm

Kokkos Package Options

Sandia National Laboratories

- Using a half neighbor list with netwon flag on is usually better for CPUs but requires atomics when using more than one thread
- For pairwise potentials, using a full neighbor list doubles the computation but doesn't require thread atomics and can reduce communication (often better for GPU and sometimes Xeon Phi)
- Using threaded communication (packing/unpacking buffers) is faster on the GPU since it avoids host/device memory transfer but can be slower on the CPU or Xeon Phi
- These differences are implemented as options in the LAMMPS Kokkos package

Compiling and Running Kokkos Package

Sandia National Laboratories

- Need c++11 compiler (gcc 4.7.2 or higher, intel 14.0 or higher, CUDA 6.5 or higher)
- In /src directory, "make yes-kokkos"
- Build with /src/MAKE/OPTIONS/Makefile.kokkos_omp or Makefile.kokkos_cuda_openmpi
- Run with 4 MPI and 4 GPUs: "mpiexec -np 4 ./lmp_exe -in in.lj -k on g 4 -sf kk"
- Run with 4 OpenMP threads: "./lmp_exe -in in.lj -k on t 4 -sf kk -pk kokkos newton on neigh half"
- Kokkos package documentation will be updated soon

Comparison of Kokkos to Other LAMMPS Packages

Sandia National Laboratories

- **USER-OMP**
  - Kokkos uses atomics or a full neighbor list to avoid write conflicts, while USER-OMP uses memory duplication
  - USER-OMP is typically faster for a few number of threads, while Kokkos is more thread-scalable
- **GPU package**
  - GPU package only runs the pair style and a few other computations on the GPU and works best when coupled with many CPUs
  - Kokkos package tries to run everything (including fixes, bonds, etc.) on the GPU
- **USER-INTEL**
  - USER-INTEL supports single, double and mixed precision, Kokkos currently only supports double precision
  - USER-INTEL vectorizes better

Accelerator Package Rules of Thumb

Sandia National Laboratories

CPUs and Xeon Phis

- Use USER-INTEL if available
- Otherwise if you are using a few threads, use USER-OMP or OPT, otherwise use Kokkos serial or Kokkos

GPUs

- If all/most of the fix styles are in the Kokkos package, use the Kokkos package
- If many fixes are not yet in the Kokkos package, use the GPU package
- If you want to use many more CPUs than GPUs, use the GPU package
- For single or mixed precision, use the GPU package

New Benchmark Website

Sandia National Laboratories

- Very non-trivial to get optimal performance on modern HPC platforms
- Current LAMMPS benchmarking page is outdated
- New LAMMPS benchmarking website will show performance plots for different accelerator packages on different hardware
- Will also include links to:
  - Tables of time for each run
  - Makefiles used for compiling LAMMPS
  - List of modules loaded
  - Exact MPI run command used, along with affinity settings
  - LAMMPS logfiles for each run

Benchmark Problems

Sandia National Laboratories

- **Lennard-Jones** = atomic fluid with Lennard-Jones potential
- **EAM** = metallic solid with EAM potential
- **Tersoff** = semiconductor solid with Tersoff potential
- **Chain** = bead-spring polymer melt of 100-mer chains
- **Granular** = chute flow of spherical granular particles
- Still to be added: **Rhodopsin** (solvated protein in bilayer), **ReaxFF**, **GayBerne**

Accelerator Packages used for Benchmarks

Sandia National Laboratories

- **For acceleration on a CPU/Intel KNL:**
  - CPU = reference implementation, no package, no acceleration (CPU)
  - OPT package with generic optimizations for CPUs (OPT)
  - USER-OMP package with OpenMP support (OMP)
  - USER-INTEL package with CPU and precision options (Intel/CPU)
  - KOKKOS package with OMP option for OpenMP (Kokkos/OMP)
  - KOKKOS package with serial option (Kokkos/serial)
- **For acceleration on an NVIDIA GPU:**
  - GPU package, with precision options (GPU)
  - KOKKOS package with CUDA option (Kokkos/Cuda)

Benchmark Machines

Sandia National Laboratories

- **chama** = Intel SandyBridge CPUs
  - 1232 nodes
  - One node = dual Sandy Bridge:2S:8C @ 2.6 GHz, 16 cores, no hyperthreading
  - interconnect = Qlogic Infiniband 4x QDR, fat tree
- **serrano** = Intel Broadwell CPUs
  - 1122 nodes
  - one node = dual Broadwell 2.1 GHz CPU E5-2695, 36 cores + 2x hyperthreading
  - interconnect = Omni-Path

Benchmark Machines

Sandia National Laboratories

- **mutrino** = Intel Haswell CPUs and Intel KNLs
  - ~100 CPU nodes
    - one node = dual Haswell 2.3 GHz CPU, 32 cores + 2x hyperthreading
  - ~100 KNL nodes
    - node = single Knight's Landing processor, 64 cores + 4x hyperthreading
  - interconnect = Cray Aries Dragonfly

Benchmark Machines

Sandia National Laboratories

- **ride80** = IBM Power8 CPUs and NVIDIA K80 GPUs
  - 11 nodes
  - one node = dual Power8 3.42 GHz CPU (Firestone), 16 cores + 8x hyperthreading
  - each node has 2 Tesla K80 GPUs (each K80 is "dual" with 2 internal GPUs)
  - interconnect = Infiniband
- **ride100** = IBM Power8 CPUs and NVIDIA P100 GPUs
  - 8 nodes
  - one node = dual Power8 3.42 GHz CPU (Garrison), 16 cores + 8x hyperthreading
  - each node has 4 Pascal P100 GPUs
  - interconnect = Infiniband

Parameter Sweep

Sandia National Laboratories

- Don’t know optimal number of MPI tasks vs OpenMP threads or number of hyperthreads to use *a priori*
- For GPU package, don’t know optimal number of CPUs per GPU
- Use a parameter sweep to find optimal settings for the different packages
- Only best results for each package included on the website

Types of Runs

Sandia National Laboratories

- **Fixed number of timesteps (i.e. 100)**
  - For cheap potentials like LJ, run may be too short, which leads to high variance in the results
  - For expensive potentials or large number of atoms, run may take a long time
- **Fixed time (i.e. 30 seconds)**
  - Use fix halt to set an approximate time limit
  - Can use fixed number of timesteps for the first parameter sweep and then refine results with fixed time

Types of Scaling

Sandia National Laboratories

- Single core
- Single node
- Multi-node strong scaling up to 64 nodes (fixed problem size)
- Multi-node weak scaling up to 64 nodes (fixed problem size per node)
- Also have some data for KNL scaling up to 8192 nodes

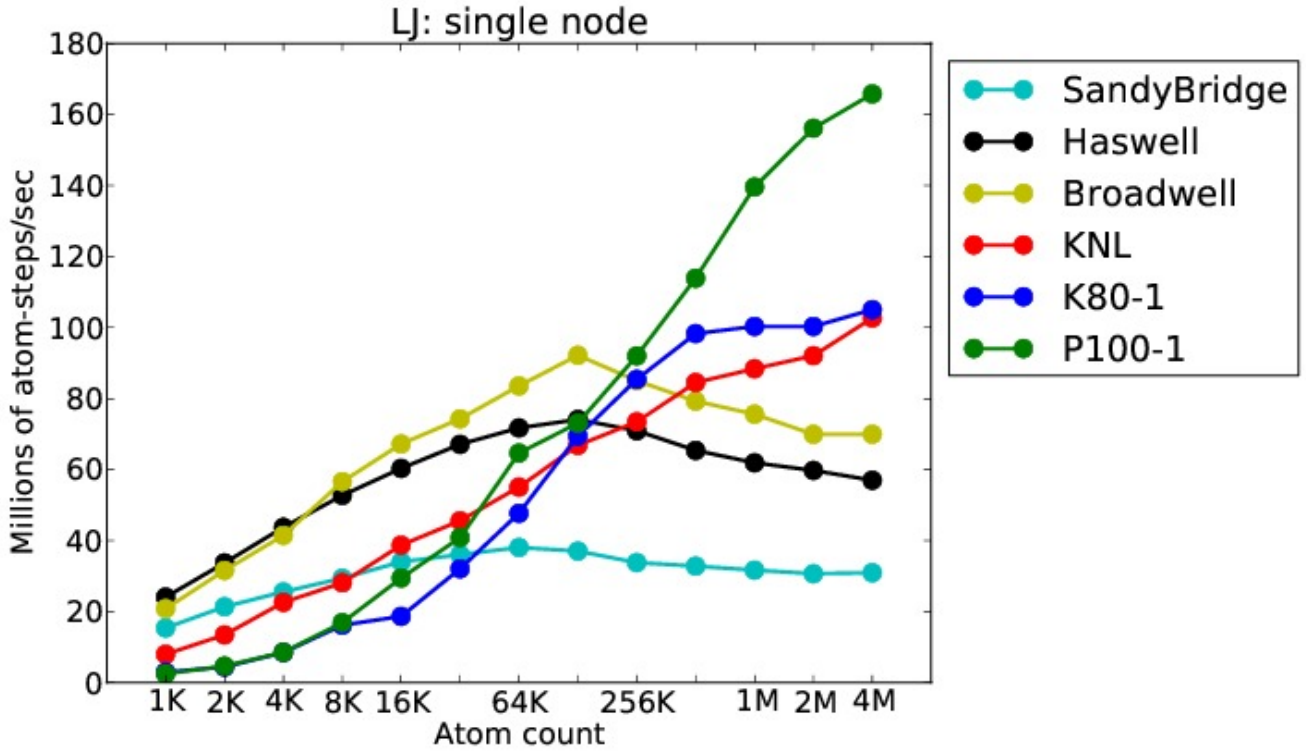
Automation

Sandia National Laboratories

- Python script is created for every machine and every model
- Python scripts work together to generate batch scripts for each accelerator package and model
- Batch scripts are submitted to the job queue on each machine
- Python script post-process logfiles to generate tables of timings, finds “best” time in sweep of parameters
- Python scripts generate plots from tables and then generates webpage
- LAMMPS is constantly being improved; easy to rerun the benchmarks and regenerate the webpage with updated results

Information Hierarchy

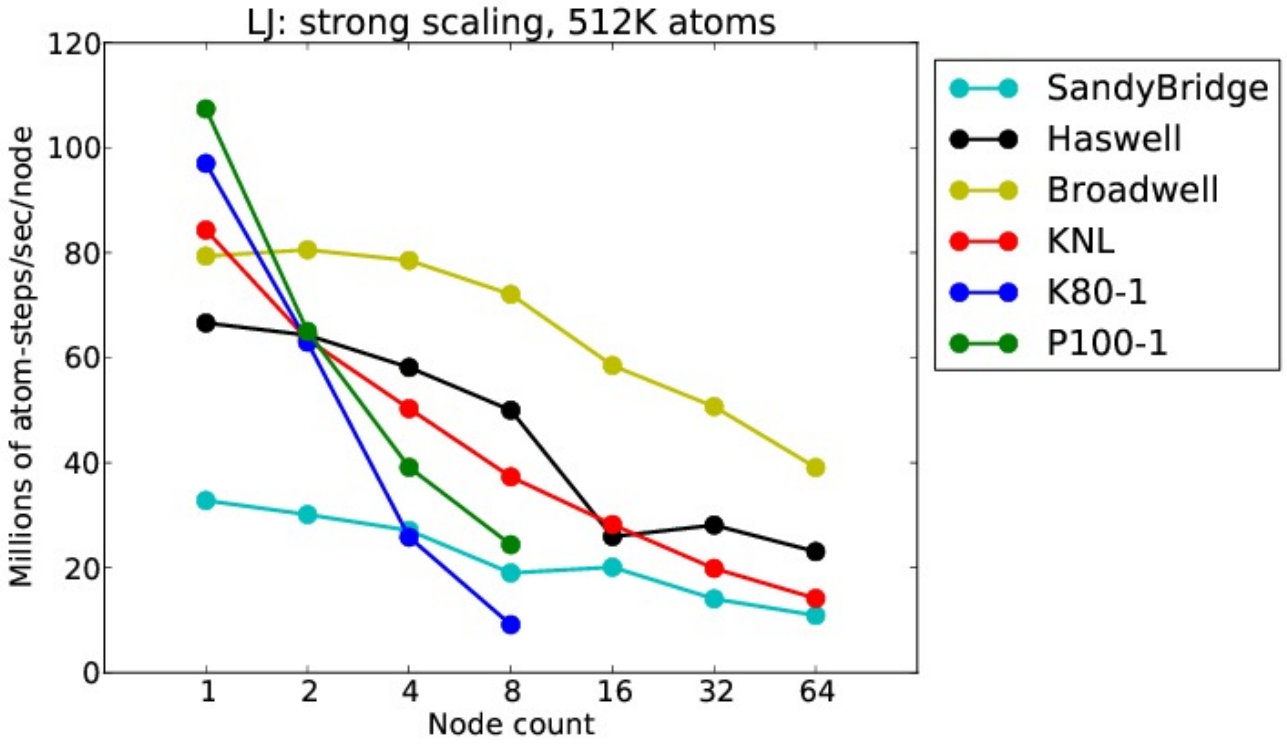
- For each model and scaling type (node, weak etc.), show
  - Overall best performance for each machine using any accelerator package



  - Results in this presentation are preliminary and may be improved

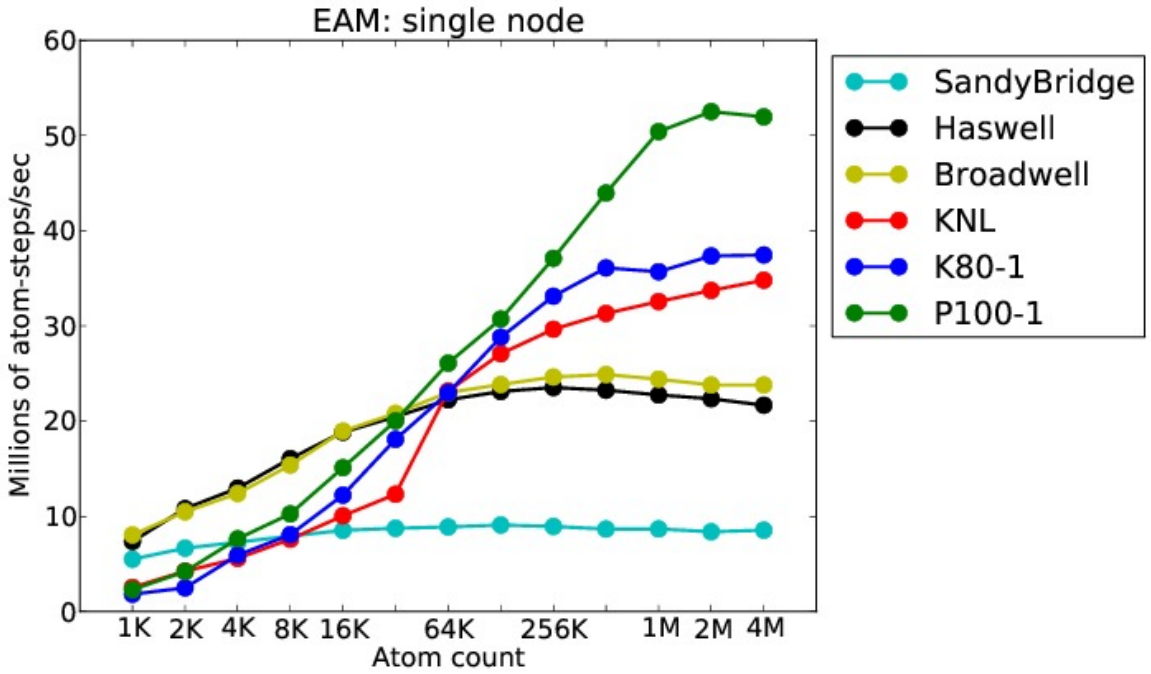
Information Hierarchy

- For each model and scaling type (node, weak etc.), show
  - Overall best performance for each machine using any accelerator package



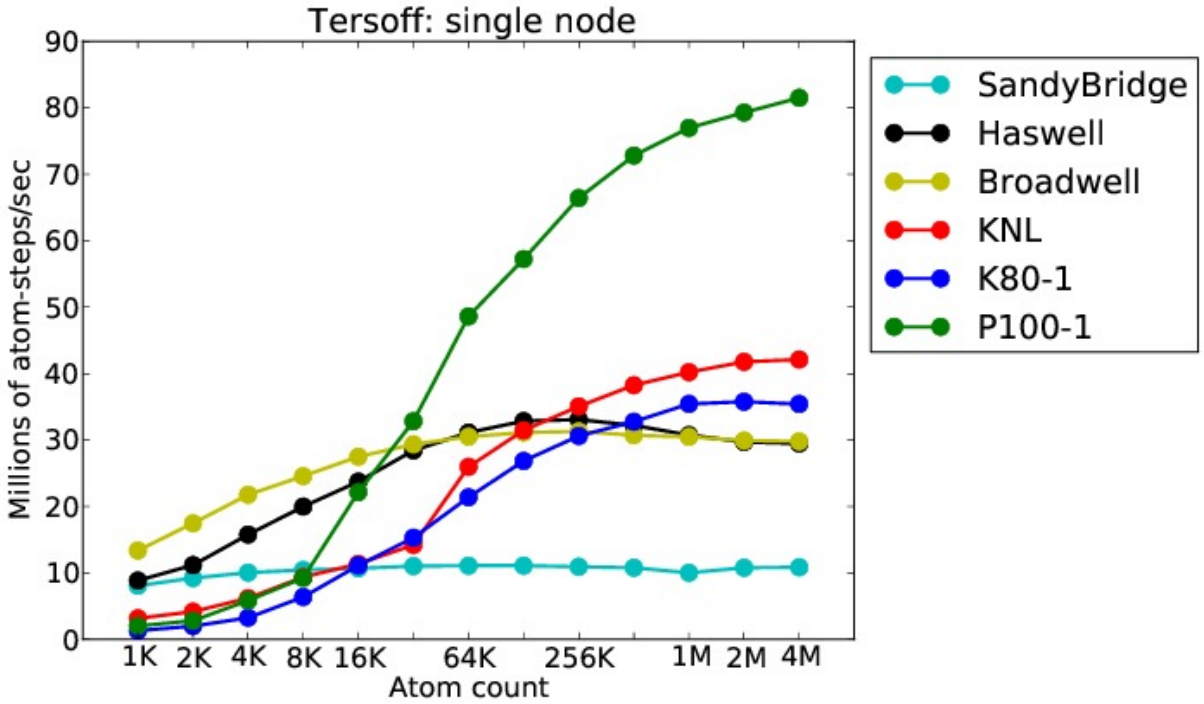
Information Hierarchy

- For each model and scaling type (node, weak etc.), show
  - Overall best performance for each machine using any accelerator package



Information Hierarchy

- For each model and scaling type (node, weak etc.), show
  - Overall best performance for each machine using any accelerator package



Information Hierarchy

- For each model and scaling type, also show
  - Table of performance for each machine using any accelerator package
  - Links to LAMMPS logfiles

Single node performance, LJ benchmark
Performance in millions of atom-timesteps / second

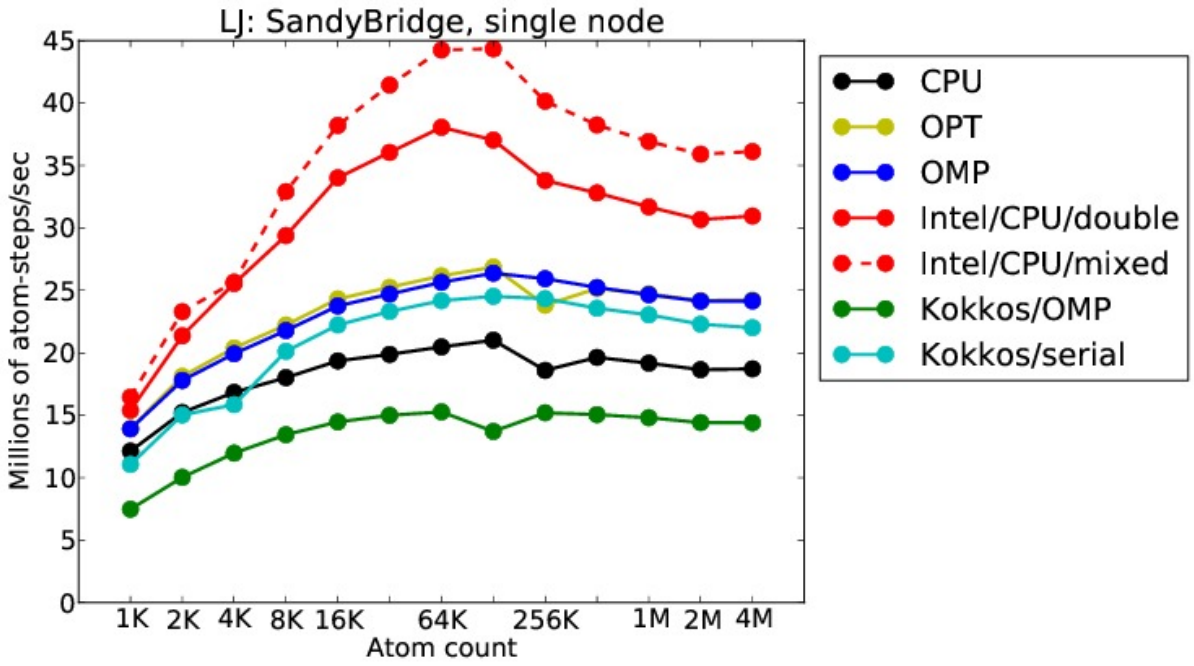
Natoms	SandyBridge	Haswell	Broadwell	KNL	K80-1	P100-1
1000	15.39 (Intel/CPU,mpi=16)	24.07 (OPT,mpi=32,hyper=1)	20.82 (Intel/CPU,mpi=36,hyper=1)	7.98 (Kokkos/KNL,mpi=8,thread=8,hyper=1)	2.96 (GPU,mpi=2,hyper=1)	2.49 (GPU,mpi=1,hyper=1)
2000	21.37 (Intel/CPU,mpi=16)	33.71 (Intel/CPU,mpi=32,hyper=1)	31.58 (Intel/CPU,mpi=36,hyper=2)	13.44 (Intel/KNL,mpi=32,thread=2,hyper=1)	4.381 (GPU,mpi=2,hyper=1)	4.657 (GPU,mpi=1,hyper=1)
4000	25.55 (Intel/CPU,mpi=16)	43.72 (Intel/CPU,mpi=32,hyper=1)	41.46 (Intel/CPU,mpi=36,hyper=2)	22.6 (Intel/KNL,mpi=64,thread=1,hyper=1)	8.492 (GPU,mpi=2,hyper=1)	8.532 (GPU,mpi=1,hyper=1)
8000	29.39 (Intel/CPU,mpi=16)	52.72 (Intel/CPU,mpi=32,hyper=1)	56.59 (Intel/CPU,mpi=36,hyper=2)	28.11 (Intel/KNL,mpi=32,thread=2,hyper=1)	16.17 (GPU,mpi=2,hyper=1)	16.89 (GPU,mpi=1,hyper=1)
16000	34.01 (Intel/CPU,mpi=16)	60.33 (Intel/CPU,mpi=32,hyper=1)	67.21 (Intel/CPU,mpi=36,hyper=1)	38.72 (Intel/KNL,mpi=64,thread=1,hyper=1)	18.67 (GPU,mpi=2,hyper=1)	29.44 (Kokkos/Cuda,mpi=1)
32000	36.05 (Intel/CPU,mpi=16)	67.1 (Intel/CPU,mpi=32,hyper=1)	74.27 (Intel/CPU,mpi=36,hyper=1)	45.58 (Intel/KNL,mpi=64,thread=1,hyper=1)	32.06 (Kokkos/Cuda,mpi=2)	40.84 (Kokkos/Cuda,mpi=1)
64000	38.05 (Intel/CPU,mpi=16)	71.72 (Intel/CPU,mpi=32,hyper=1)	83.55 (Intel/CPU,mpi=72,hyper=1)	55.06 (Intel/KNL,mpi=128,thread=1,hyper=2)	47.7 (Kokkos/Cuda,mpi=2)	64.65 (Kokkos/Cuda,mpi=1)
128000	37.03 (Intel/CPU,mpi=16)	74.09 (Intel/CPU,mpi=32,hyper=1)	92.17 (Intel/CPU,mpi=72,hyper=2)	66.81 (Intel/KNL,mpi=128,thread=1,hyper=2)	69.42 (Kokkos/Cuda,mpi=2)	73.07 (Kokkos/Cuda,mpi=1)
256000	33.8 (Intel/CPU,mpi=16)	70.93 (Intel/CPU,mpi=64,hyper=2)	85.01 (Intel/CPU,mpi=36,hyper=2)	73.44 (Intel/KNL,mpi=64,thread=2,hyper=2)	85.4 (Kokkos/Cuda,mpi=2)	92 (Kokkos/Cuda,mpi=1)
512000	32.8 (Intel/CPU,mpi=16)	65.39 (OPT,mpi=64,hyper=2)	79.26 (Intel/CPU,mpi=72,hyper=2)	84.5 (Intel/KNL,mpi=128,thread=1,hyper=2)	98.27 (Kokkos/Cuda,mpi=2)	113.9 (GPU,mpi=16,hyper=1)
1024000	31.68 (Intel/CPU,mpi=16)	61.9 (OPT,mpi=64,hyper=2)	75.59 (Intel/CPU,mpi=72,hyper=1)	88.38 (Intel/KNL,mpi=128,thread=1,hyper=2)	100.3 (GPU,mpi=16,hyper=1)	139.6 (GPU,mpi=16,hyper=1)
2048000	30.67 (Intel/CPU,mpi=16)	59.74 (Intel/CPU,mpi=64,hyper=2)	69.92 (Intel/CPU,mpi=72,hyper=2)	92.06 (Intel/KNL,mpi=128,thread=1,hyper=2)	100.3 (GPU,mpi=16,hyper=1)	156.1 (GPU,mpi=16,hyper=1)
4096000	30.95 (Intel/CPU,mpi=16)	56.98 (OMP,mpi=64,hyper=2,thread=1)	69.94 (Intel/CPU,mpi=72,hyper=1)	102.7 (Intel/KNL,mpi=256,thread=1,hyper=4)	105 (GPU,mpi=16,hyper=1)	165.8 (GPU,mpi=16,hyper=1)

Run commands and logfile links for column SandyBridge

1000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 5 -v y 5 -v z 10 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=1K.node=1.mpi=16
2000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 5 -v y 10 -v z 10 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=2K.node=1.mpi=16
4000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 10 -v y 10 -v z 10 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=4K.node=1.mpi=16
8000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 10 -v y 10 -v z 20 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=8K.node=1.mpi=16
16000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 10 -v y 20 -v z 20 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=16K.node=1.mpi=16
32000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 20 -v y 20 -v z 20 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=32K.node=1.mpi=16
64000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 20 -v y 20 -v z 40 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=64K.node=1.mpi=16
128000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 20 -v y 40 -v z 40 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=128K.node=1.mpi=16
256000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 40 -v y 40 -v z 40 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=256K.node=1.mpi=16
512000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 40 -v y 40 -v z 80 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=512K.node=1.mpi=16
1024000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 40 -v y 80 -v z 80 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=1M.node=1.mpi=16
2048000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 80 -v y 80 -v z 80 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=2M.node=1.mpi=16
4096000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -sf intel -pk intel 0 mode double -v x 80 -v y 80 -v z 160 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=intel_cpu.precision=double.kind=node.size=4M.node=1.mpi=16

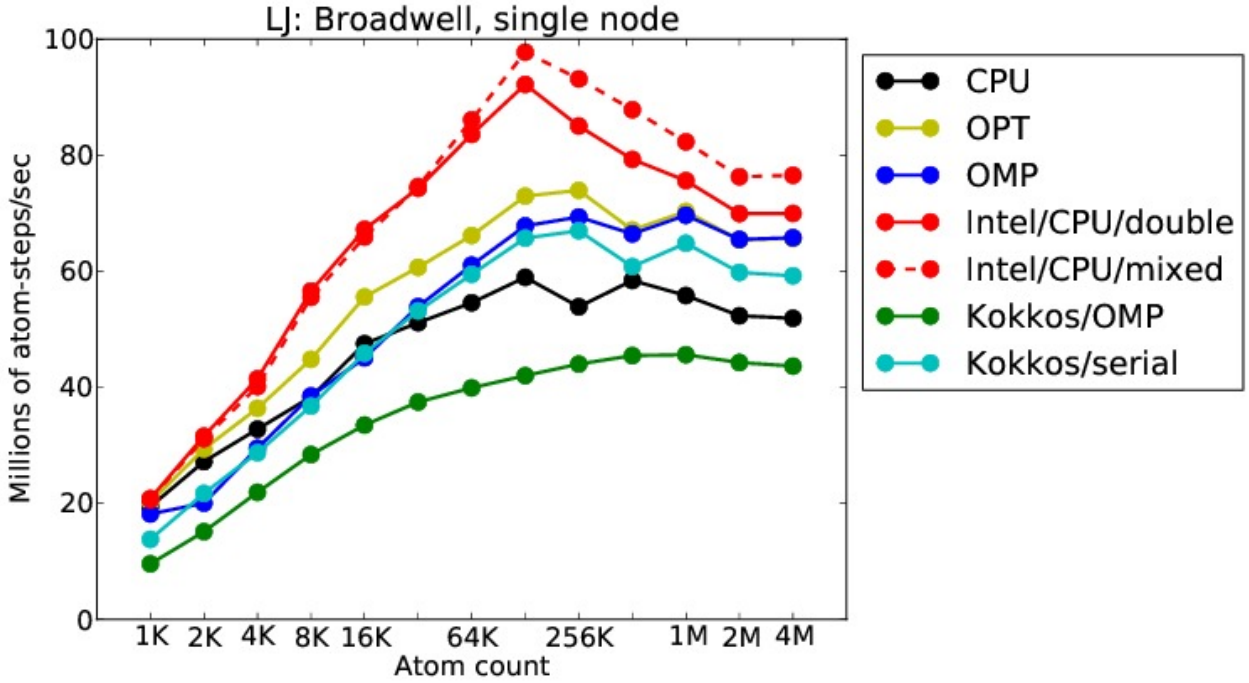
Information Hierarchy

- For each machine, model, and scaling type, show
  - Performance for each accelerator package (best out of parameter sweep)



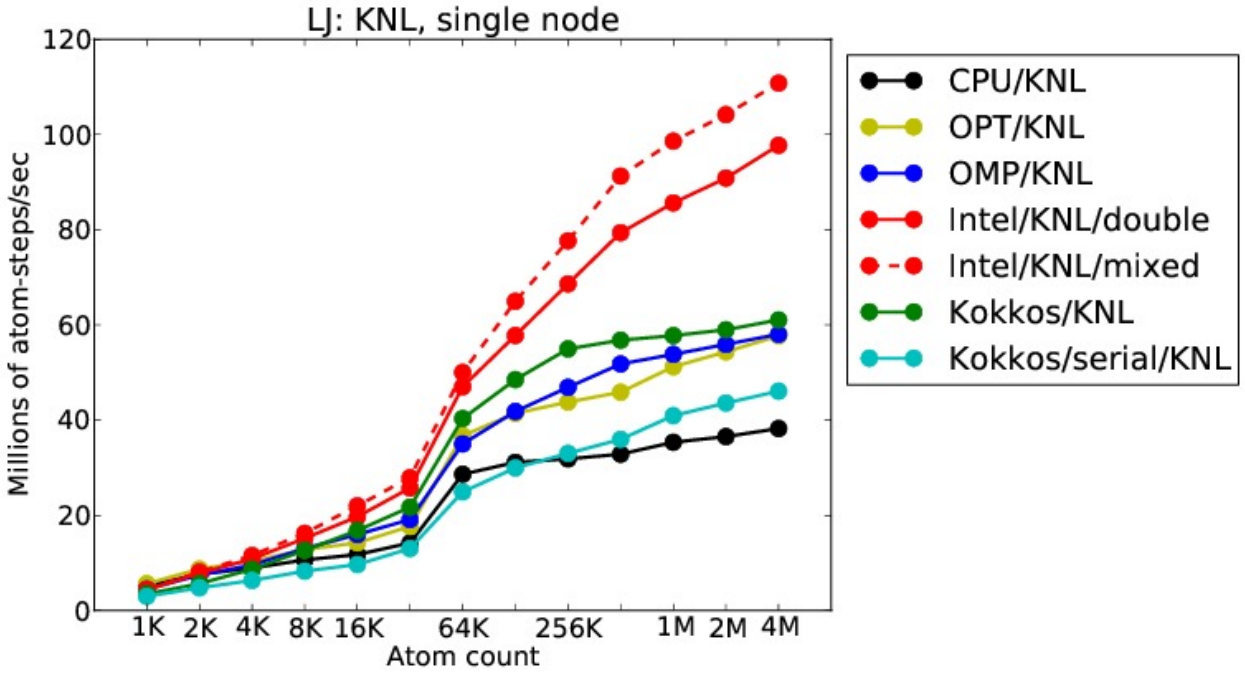
Information Hierarchy

- For each machine, model, and scaling type, show
  - Performance for each accelerator package (best out of parameter sweep)



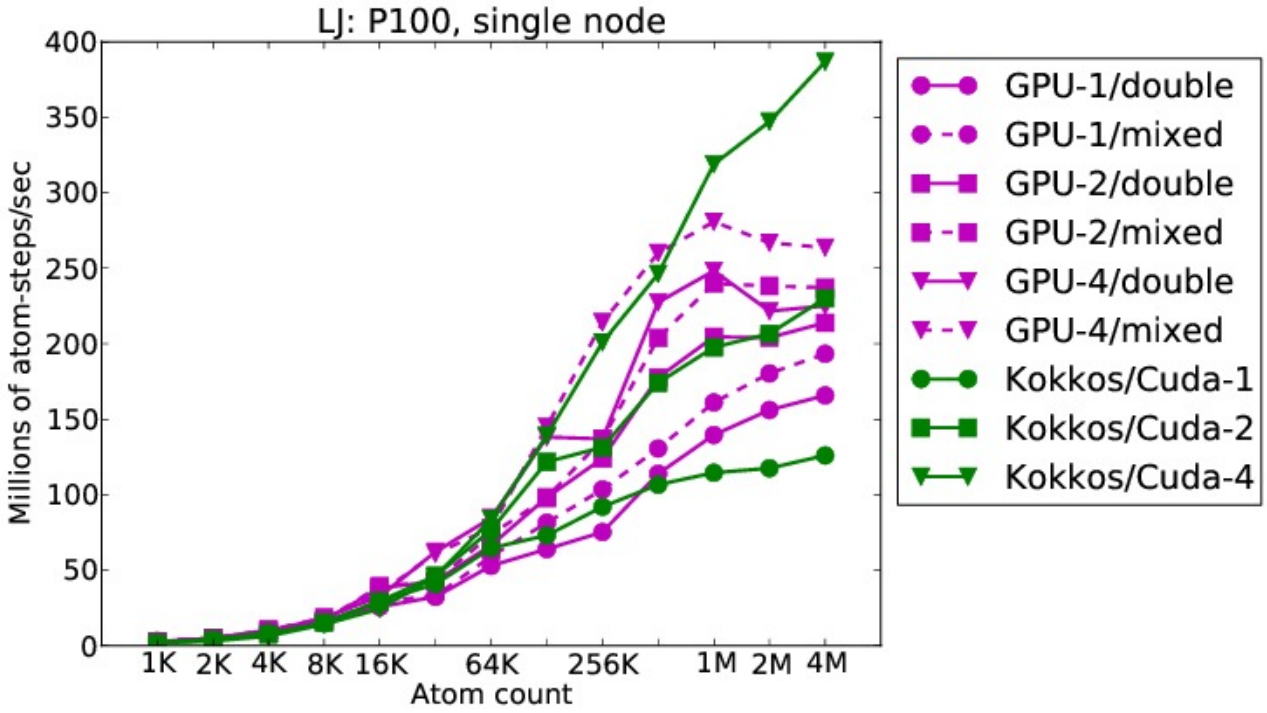
Information Hierarchy

- For each machine, model, and scaling type, show
  - Performance for each accelerator package (best out of parameter sweep)



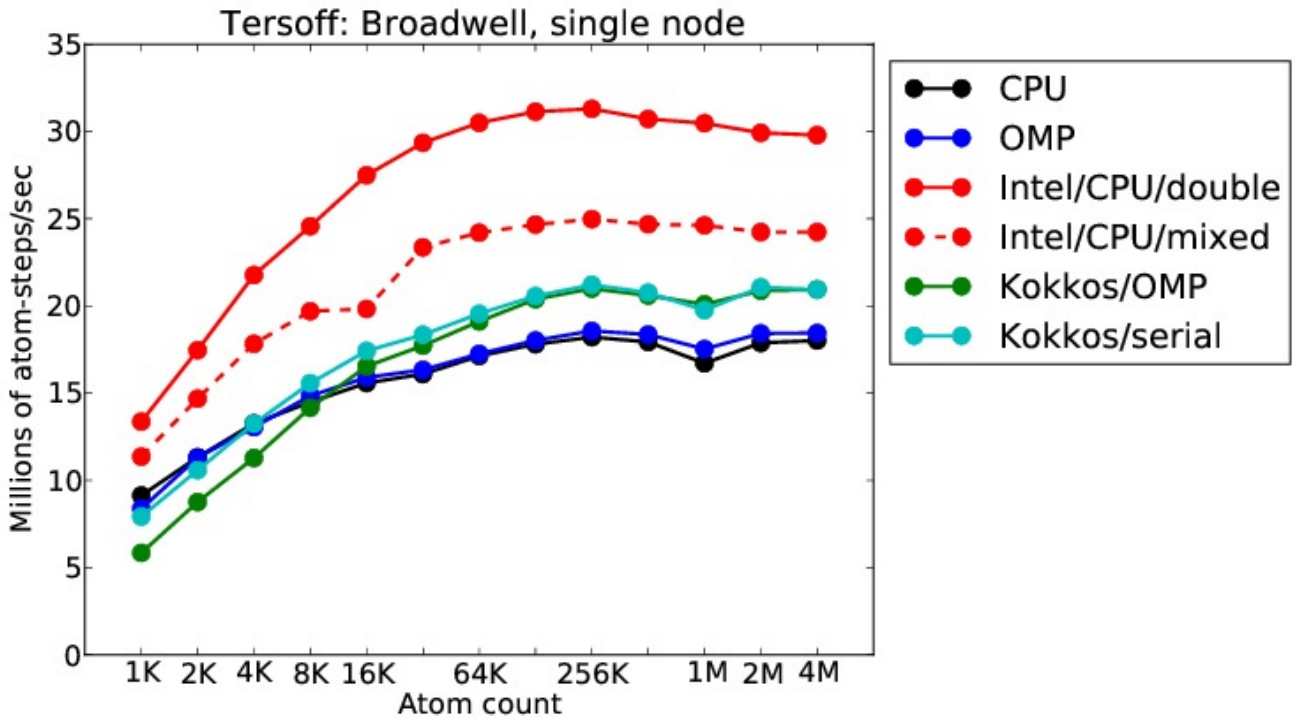
Information Hierarchy

- For each machine, model, and scaling type, show
  - Performance for each accelerator package (best out of parameter sweep)



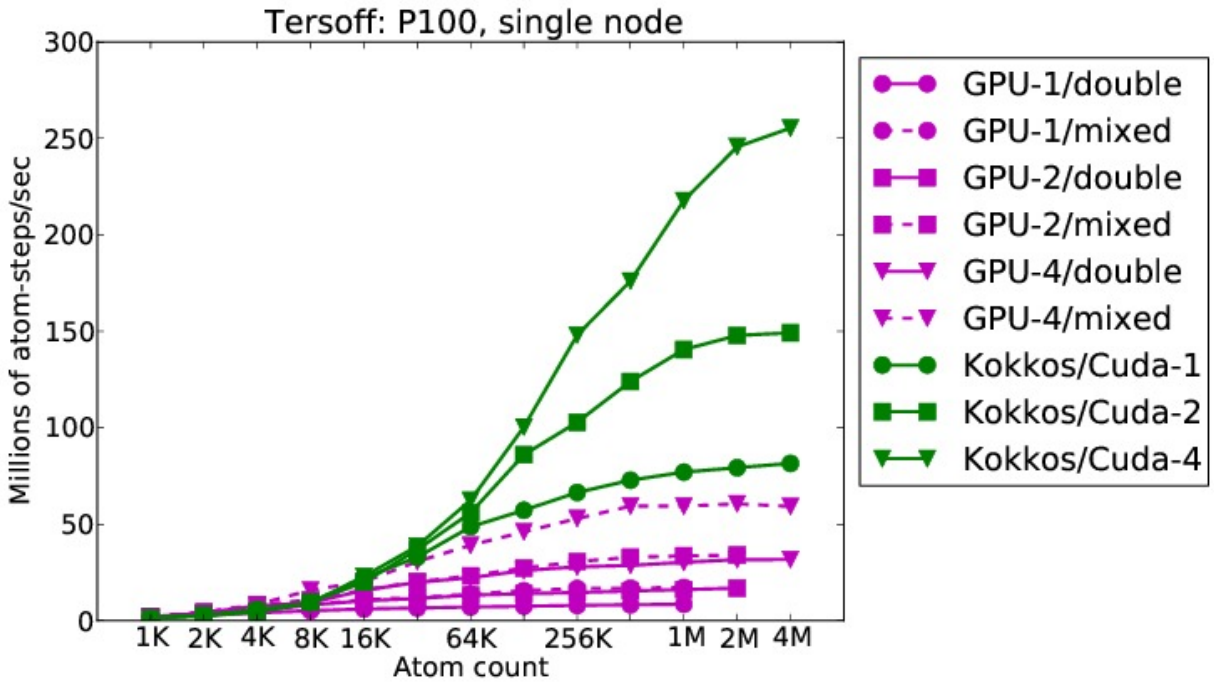
Information Hierarchy

- For each machine, model, and scaling type, show
  - Performance for each accelerator package (best out of parameter sweep)



Information Hierarchy

- For each machine, model, and scaling type, show
  - Performance for each accelerator package (best out of parameter sweep)



Information Hierarchy (cont.)

Sandia National Laboratories

- For each machine, model, and scaling type, also show
  - Table of performance for each accelerator package (best out of parameter sweep)

Single node performance, LJ benchmark, SandyBridge
Performance in millions of atom-timesteps / second

Natoms	CPU (mpi)	OPT (mpi)	OMP (mpi,thread)	Intel/CPU/double (mpi)	Intel/CPU/mixed (mpi)	Kokkos/OMP (mpi,thread)	Kokkos/serial (mpi)
1000	12.13 (16)	13.93 (16)	13.91 (16,1)	15.39 (16)	16.45 (16)	7.477 (16,1)	11.06 (16)
2000	15.22 (16)	18.11 (16)	17.79 (16,1)	21.37 (16)	23.29 (16)	10.03 (16,1)	15.02 (16)
4000	16.83 (16)	20.39 (16)	19.92 (16,1)	25.55 (16)	25.66 (16)	11.97 (16,1)	15.85 (16)
8000	18.01 (16)	22.22 (16)	21.78 (16,1)	29.39 (16)	32.91 (16)	13.44 (16,1)	20.12 (16)
16000	19.35 (16)	24.31 (16)	23.75 (16,1)	34.01 (16)	38.2 (16)	14.47 (16,1)	22.23 (16)
32000	19.87 (16)	25.24 (16)	24.7 (16,1)	36.05 (16)	41.44 (16)	14.99 (16,1)	23.31 (16)
64000	20.47 (16)	26.16 (16)	25.64 (16,1)	38.05 (16)	44.25 (16)	15.27 (16,1)	24.18 (16)
128000	21.01 (16)	26.86 (16)	26.37 (16,1)	37.03 (16)	44.34 (16)	13.7 (16,1)	24.52 (16)
256000	18.6 (16)	23.81 (16)	25.93 (16,1)	33.8 (16)	40.15 (16)	15.2 (16,1)	24.34 (16)
512000	19.63 (16)	25.15 (16)	25.23 (16,1)	32.8 (16)	38.24 (16)	15.05 (16,1)	23.57 (16)
1024000	19.17 (16)	24.72 (16)	24.65 (16,1)	31.68 (16)	36.92 (16)	14.8 (16,1)	23.04 (16)
2048000	18.65 (16)	24.19 (16)	24.13 (16,1)	30.67 (16)	35.91 (16)	14.41 (16,1)	22.29 (16)
4096000	18.71 (16)	24.22 (16)	24.15 (16,1)	30.95 (16)	36.11 (16)	14.4 (16,1)	22.01 (16)

Run commands and logfile links for column CPU

1000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 5 -v y 5 -v z 10 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=1K.node=1.mpi=16
2000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 5 -v y 10 -v z 10 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=2K.node=1.mpi=16
4000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 10 -v y 10 -v z 10 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=4K.node=1.mpi=16
8000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 10 -v y 10 -v z 20 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=8K.node=1.mpi=16
16000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 10 -v y 20 -v z 20 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=16K.node=1.mpi=16
32000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 20 -v y 20 -v z 20 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=32K.node=1.mpi=16
64000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 20 -v y 20 -v z 40 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=64K.node=1.mpi=16
128000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 20 -v y 40 -v z 40 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=128K.node=1.mpi=16
256000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 40 -v y 40 -v z 40 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=256K.node=1.mpi=16
512000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 40 -v y 40 -v z 80 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=512K.node=1.mpi=16
1024000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 40 -v y 80 -v z 80 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=1M.node=1.mpi=16
2048000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 80 -v y 80 -v z 80 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=2M.node=1.mpi=16
4096000	mpirun -n 16 -N 16 --bind-to core lmp_chama_cpu -v x 80 -v y 80 -v z 160 -v t 30 -v tfreq 100 -in in.lj -log log.lammps.date=31Mar17.model=lj.machine=chama.pkg=cpu.kind=node.size=4M.node=1.mpi=16

Recent Performance Work

Sandia National Laboratories

- USER-INTEL added full neighbor list with newton off, can be better for simple pair-wise potentials on Xeon Phi
- Added “short” neighbor list to CPU, OpenMP, Kokkos and GPU (not yet released) many-body potentials (sw, tersoff, and vashishta)
- KOKKOS package improved EAM and ReaxFF performance on GPUs
- USER-OMP added multithreaded ReaxFF

ReaxFF

Sandia National Laboratories

- 4 versions in LAMMPS:
  - USER-REAXC
  - Fortran
  - KOKKOS
  - USER-OMP
- KOKKOS version more memory robust, should be used with GCMC
- KOKKOS serial version faster than USER-REAXC, at least in some cases
- KOKKOS version can run on NVIDIA GPUs
- USER-OMP version brand new, probably better for OpenMP on Xeon Phi/CPU (need to benchmark performance)

Performance Regression Testing

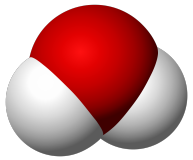
Sandia National Laboratories

- Currently have automated “code correctness” regression testing for LAMMPS
- But no performance regression tests
- Changes to the code could slow down performance without developers knowledge
- Could add automated performance regression tests

Long-Range Electrostatics

Sandia National Laboratories

- Truncation doesn't work well for charged systems due to long-ranged nature of Coulombic interactions
- Use Kspace style to add long-range electrostatics:
  - PPPM—usually fastest, uses FFTs
  - Ewald—potentially most accurate, but slow for large systems
  - MSM—multigrid method that also works for non-periodic systems
- Usually specify a relative accuracy (1e-4 or 1e-5 typically used)
- Example syntax (for periodic systems): kspace_style pppm 1.0e-4
- Use pair_style *coul/long such as lj/cut/coul/long



Accelerating LRE

Sandia National Laboratories

- 2-FFT PPPM (kspace_modify diff ad)
- Staggered PPPM
- Single vs double precision PPPM
- Partial charge PPPM
- Verlet/split run style--can overlap pair computation with Kspace

Other Performance Considerations

Sandia National Laboratories

- Processor command for MPI grid layout, can map to numa regions
- Load-balancing
  - balance command
  - fix balance
- Affinity is important and complicated, see examples on new benchmark website

Sandia National Laboratories

Questions?

Discussion/Suggestions?