SAND2011-1446C

Using GPUs for faster LAMMPS particle simulations

Paul S. Crozier

Exploiting New Computer Architectures in Molecular Dynamics Simulations

March 23, 2011

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Acknowledgements

Ilmenau Univ. of Tech., Germany

Christian Trott & Lars Winterfeld

Sandia National Laboratories

Steve Plimpton & Aidan Thompson

Oak Ridge National Lab

Mike Brown, Scott Hampton & Pratul Agarwal

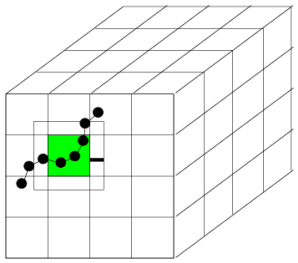
Sandia National Laboratories

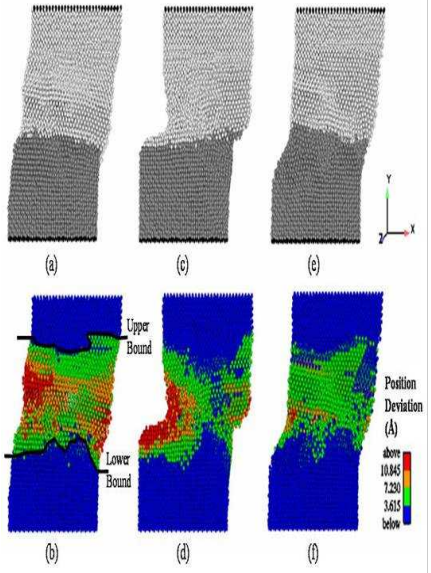
LAMMPS

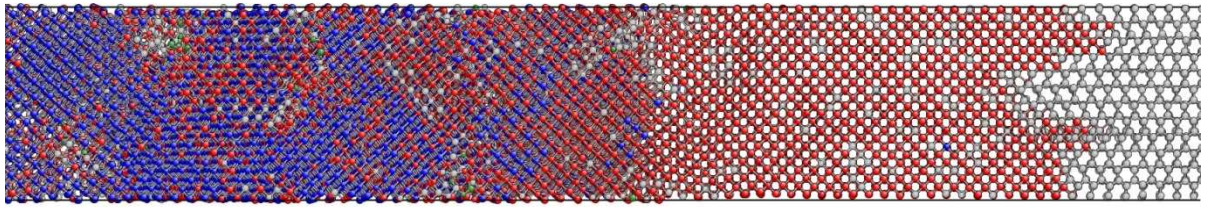
(Large-scale Atomic/Molecular Massively Parallel Simulator)

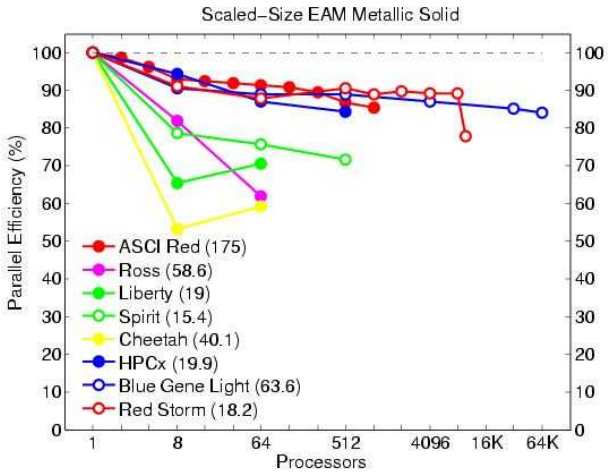
http://lammps.sandia.gov

- Classical MD code.
- Open source, highly portable C++.
- Freely available for download under GPL.
- Easy to download, install, and run.
- Well documented.
- Easy to modify or extend with new features and functionality.
- Active user's e-mail list with over **650** subscribers.
- Since Sept. 2004: over 50k downloads, grown from 53 to 175 kloc.
- Spatial-decomposition of simulation domain for parallelism.
- Energy minimization via conjugate-gradient relaxation.
- Radiation damage and two temperature model (TTM) simulations.
- Atomistic, mesoscale, and coarse-grain simulations.
- Variety of potentials (including many-body and coarse-grain).
- Variety of boundary conditions, constraints, etc.









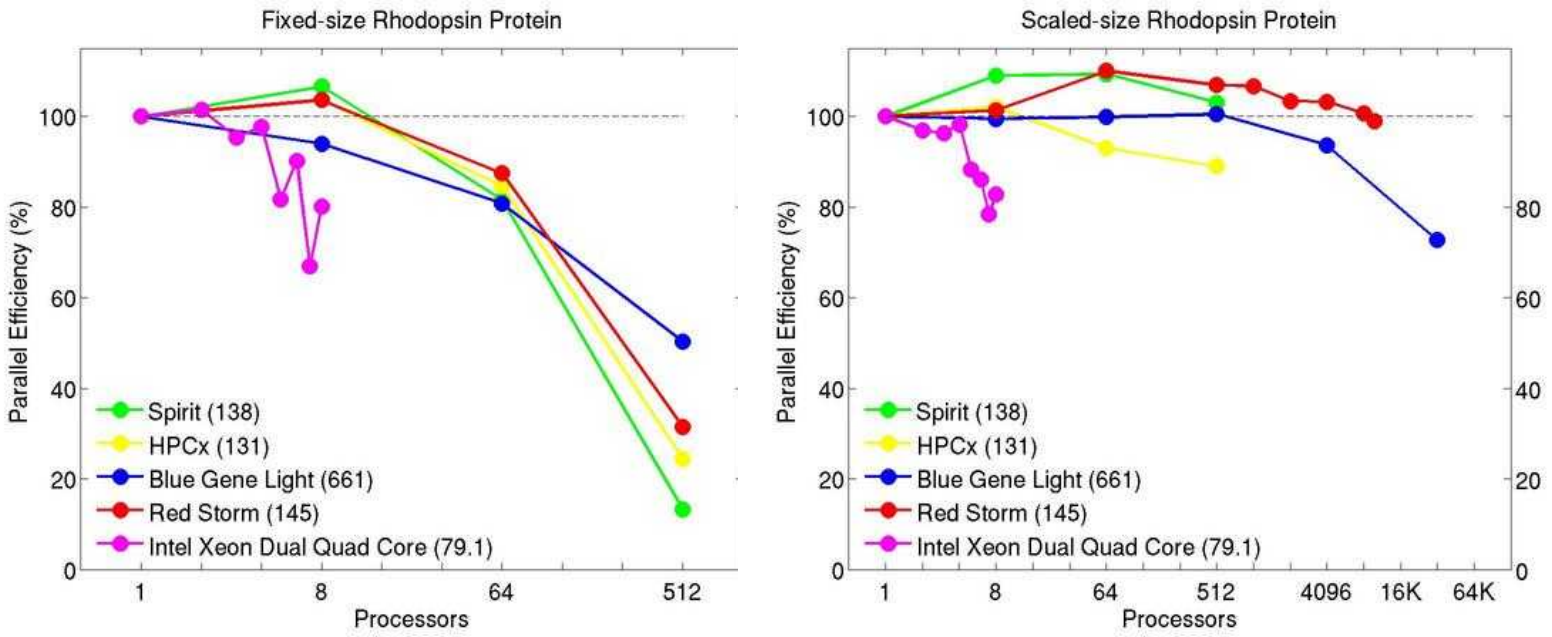
Why Use LAMMPS?

Answer 1: (Parallel) Performance

Protein (rhodopsin) in solvated lipid bilayer

Fixed-size (32K atoms) & scaled-size (32K/proc) parallel efficiencies

Billions of atoms on 64K procs of Blue Gene or Red Storm

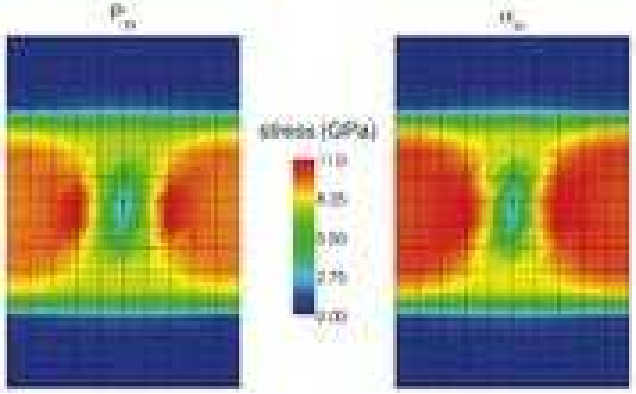


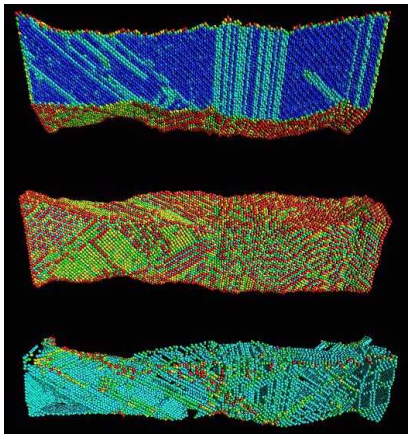
Typical speed: 5E-5 core-sec/atom-step (LJ 1E-6, ReaxFF 1E-3)

Why Use LAMMPS?

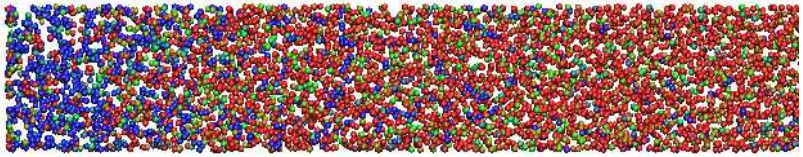
Answer 2: Versatility

Solid Mechanics



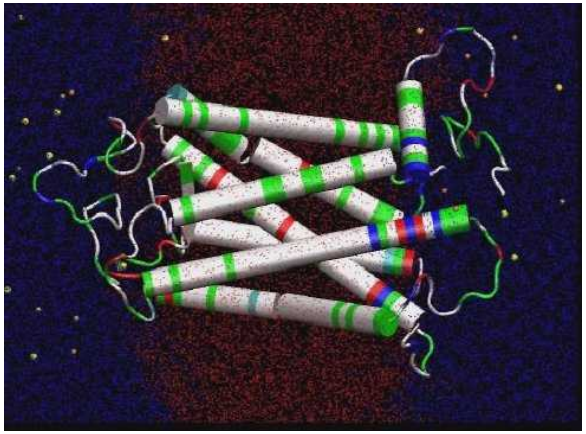


Materials Science

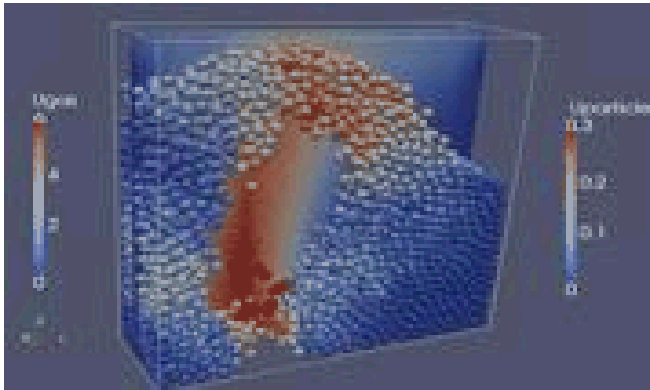


Chemistry

Biophysics



Granular Flow



Why Use LAMMPS?

Answer 3: Modularity

pair_reax.cpp
pair_reax.h

fix_nve.cpp
fix_nve.h

LAMMPS Objects

atom styles: atom, charge, colloid, ellipsoid, point dipole

pair styles: LJ, Coulomb, Tersoff, ReaxFF, AI-REBO, COMB, MEAM, EAM, Stillinger-Weber,

fix_styles: NVE dynamics, Nose-Hoover, Berendsen, Langevin, SLLOD, Indentation,...

compute styles: temperatures, pressures, per-atom energy, pair correlation function, mean square displacemnts, spatial and time averages

Goal: All computes works with all fixes work with all pair styles work with all atom styles

Sandia National Laboratories

Why Use LAMMPS?

Answer 4: Potential Coverage

LAMMPS Potentials
pairwise potentials: Lennard-Jones, Buckingham, ...
charged pairwise potentials: Coulombic, point-dipole
manybody potentials: EAM, Finnis/Sinclair, modified EAM (MEAM), embedded ion method (EIM), Stillinger-Weber, Tersoff, AI-REBO, ReaxFF, COMB
electron force field (eFF)
coarse-grained potentials: DPD, GayBerne, ...
mesoscopic potentials: granular, peridynamics
long-range Coulombics and dispersion: Ewald, PPPM (similar to particle-mesh Ewald)

Sandia National Laboratories

Why Use LAMMPS?

Answer 4: Potentials

LAMMPS Potentials (contd.)
bond potentials: harmonic, FENE,...
angle potentials: harmonic, CHARMM, ...
dihedral potentials: harmonic, CHARMM,...
improper potentials: harmonic, cvff, class 2 (COMPASS)
polymer potentials: all-atom, united-atom, bead-spring, breakable
water potentials: TIP3P, TIP4P, SPC
implicit solvent potentials: hydrodynamic lubrication, Debye
force-field compatibility with common CHARMM, AMBER, OPLS, GROMACS options

Sandia National Laboratories

Why Use LAMMPS?

Answer 4: Range of Potentials

LAMMPS Potentials

Biomolecules: CHARMM, AMBER, OPLS, COMPASS (class 2), long-range Coulombics via PPPM, point dipoles, ...

Polymers: all-atom, united-atom, coarse-grain (bead-spring FENE), bond-breaking, …

Materials: EAM and MEAM for metals, Buckingham, Morse, Yukawa, Stillinger-Weber, Tersoff, AI-REBO, ReaxFF, COMB, eFF...

Mesoscale: granular, DPD, Gay-Berne, colloidal, peri-dynamics, DSMC...

Hybrid: can use combinations of potentials for hybrid systems: water on metal, polymers/semiconductor interface, colloids in solution, …

Sandia National Laboratories

Why Use LAMMPS?

Answer 5: Easily extensible

- **One of the best features of LAMMPS**
  - **–80% of code is "extensions" via styles**
  - **–only 35K of 175K lines is core of LAMMPS**
- **Easy to add new features via 14 "styles"**
  - **–new particle types = atom style**
  - **–new force fields = pair style, bond style, angle style, dihedral style, improper style**
  - **–new long range = kspace style**
  - **–new minimizer = min style**
  - **–new geometric region = region style**
  - **–new output = dump style**
  - **–new integrator = integrate style**
  - **–new computations = compute style (global, per-atom, local)**
  - **–new fix = fix style = BC, constraint, time integration, ...**
  - **–new input command = command style = read_data, velocity, run, …**
- **Enabled by C++**
  - **–virtual parent class for all styles, e.g. pair potentials**
  - **–defines interface the feature must provide**
  - **–compute(), init(), coeff(), restart(), etc**

Sandia National Laboratories

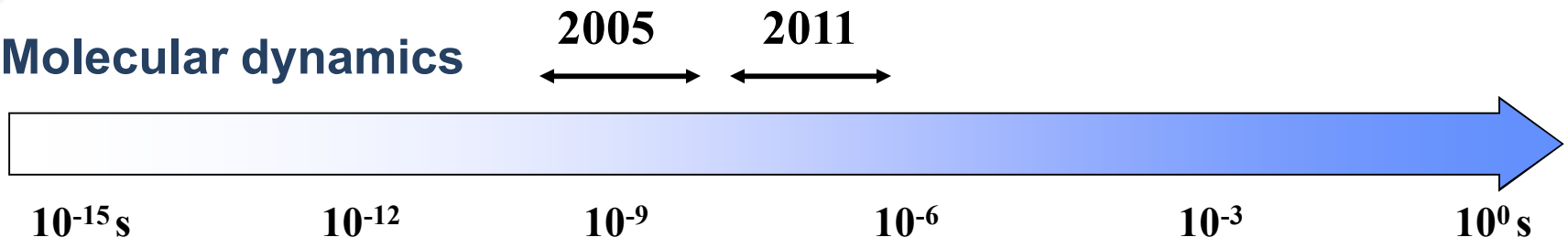
Motivation for adding GPU capabilities to LAMMPS

- User community interest
- The future is many-core, and GPU computing is leading the way
- GPUs and Nvidia's CUDA are the industry leaders
  - Already impressive performance, each generation better
  - More flops per watt
  - CUDA is way ahead of the competition
- Other MD codes are working in this area and have shown impressive speedups
  - HOOMD
  - NAMD
  - Folding@home
  - Others

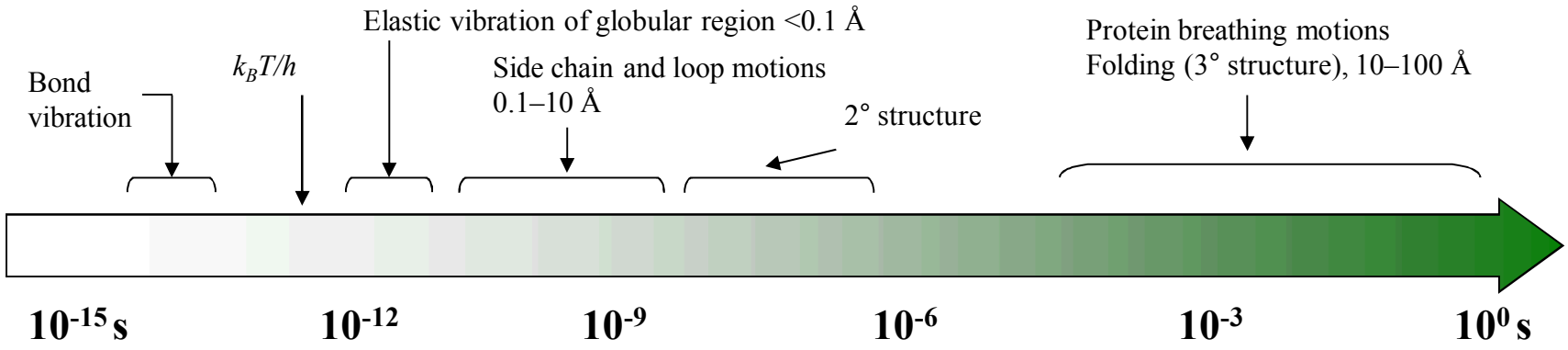
Sandia National Laboratories

Motivation: Better time-scales

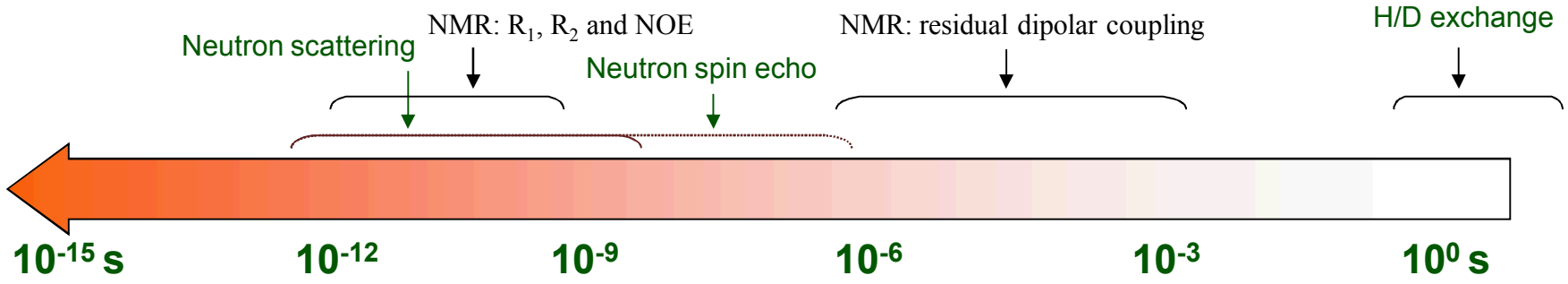
Molecular dynamics



Protein dynamical events

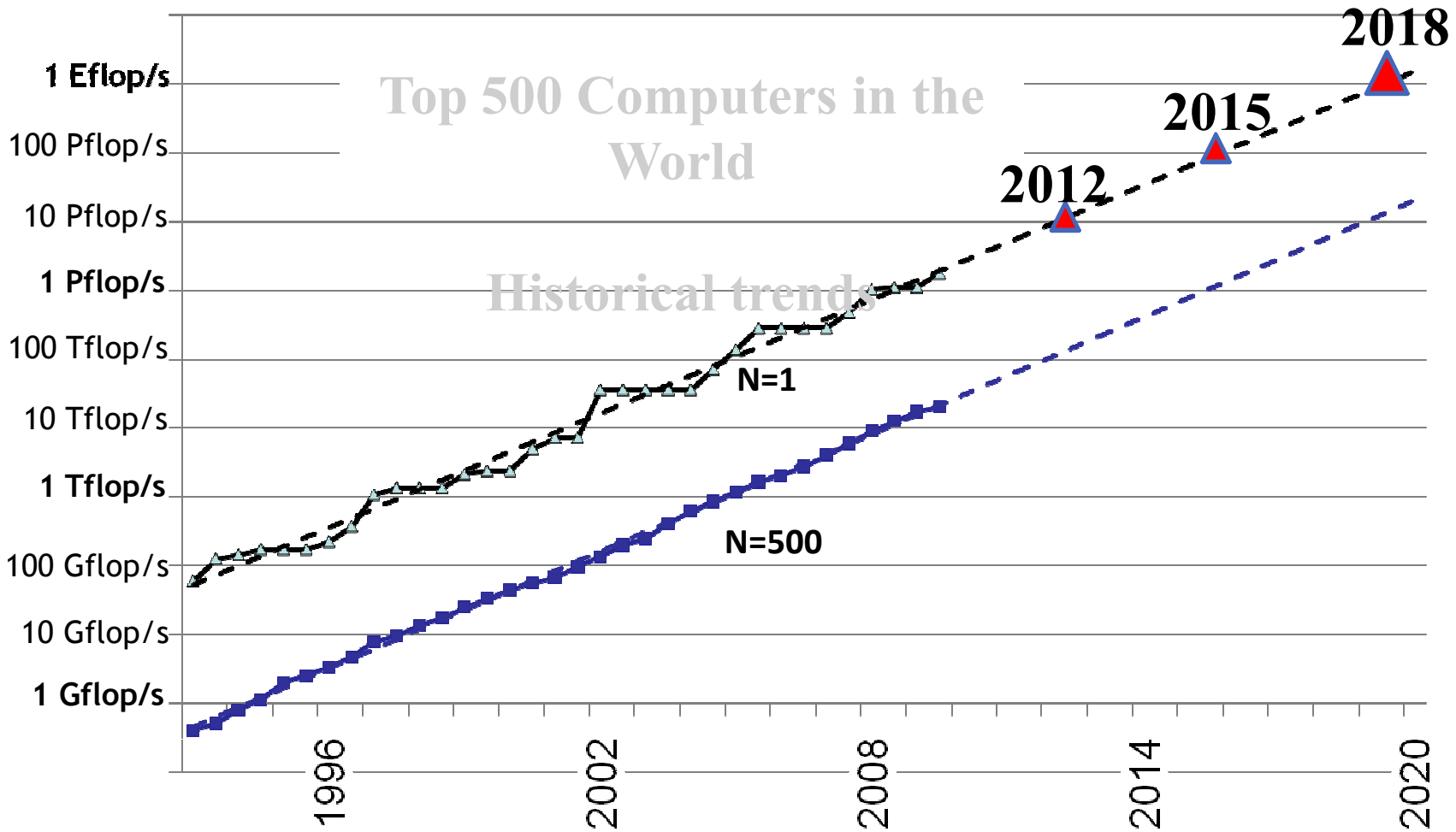


Experimental techniques



Agarwal: *Biochemistry* (2004), **43**, 10605-10618*; Microbial Cell Factories* (2006) **5**:2; *J. Phys. Chem. B* **113**, 16669-80.

A look at history

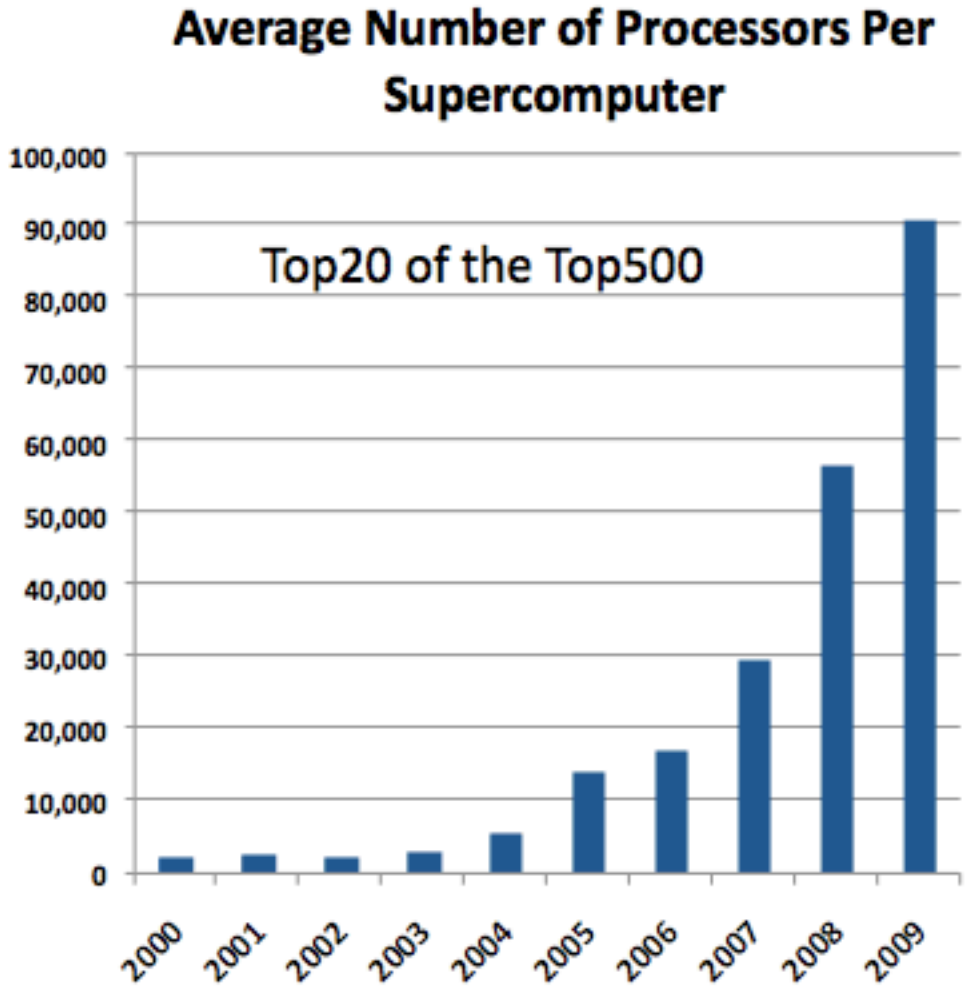


Courtesy: Al Geist (ORNL), Jack Dongarra (UTK)

Sandia National Laboratories

Concurrency

Fundamental assumptions of system software architecture and application design did not anticipate exponential growth in parallelism



Courtesy: Al Geist (ORNL)

Sandia National Laboratories

Future architectures

- Fundamentally different architecture
  - Different from the traditional MPP (homogeneous) machines
- Very high concurrency: Billion way in 2020
  - Increased concurrency on a single node
- Increased Floating Point (FP) capacity from accelerators
  - Accelerators (GPUs/FPGAs/Cell/? etc.) will add heterogeneity

Significant Challenges:

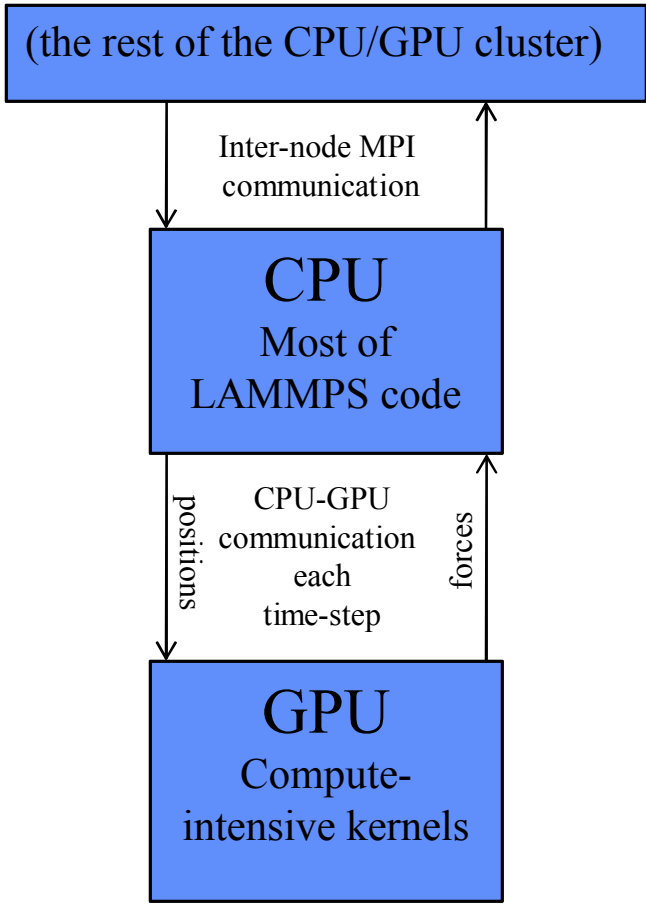
Concurrency, Power and Resiliency

2012 → 2015 → 2018 ...

GPU-LAMMPS project goals & strategy

Goals

- Better time-to-solution on particle simulation problems we care about.
- Get LAMMPS running on GPUs (other MD codes don't have all of the cool features and capabilities that LAMMPS has).
- Maintain all of LAMMPS functionality while improving performance (roughly 2x – 100x speedup).
- Harness next-generation hardware, specifically CPU+GPU clusters.



Strategy

- Enable LAMMPS to run efficiently on CPU+GPU clusters. Not aiming to optimize for running on a single GPU.
- Dual parallelism
  - Spatial decomposition, with MPI between CPU cores
  - Force decomposition, with CUDA on individual GPUs
- Leverage the work of internal and external developers.

Sandia National Laboratories

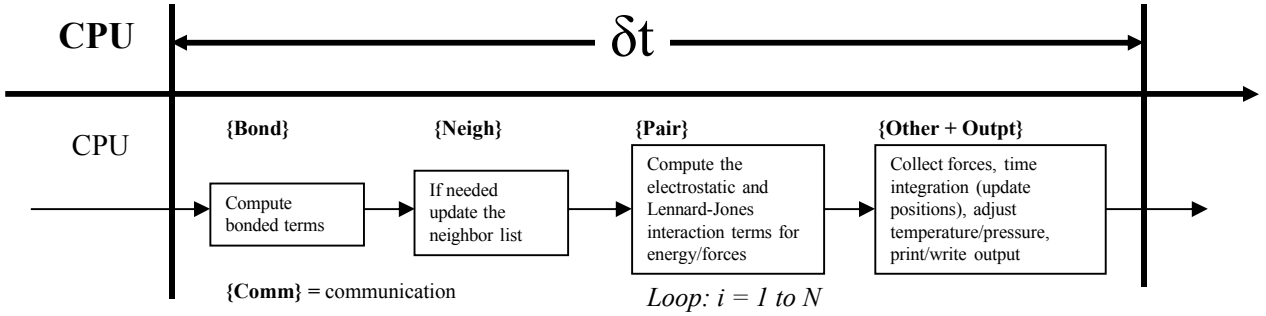
Our Approach:
Gain from multi-level parallelism

- Off-loading: improving performance in strong scaling
- Other Alternative: Entire (or most) MD run on GPU
- *Computations are free, data localization is not*
  - Host-GPU data exchange is expensive
  - In a multiple GPU and multi-node: much worse for entire MD on GPU

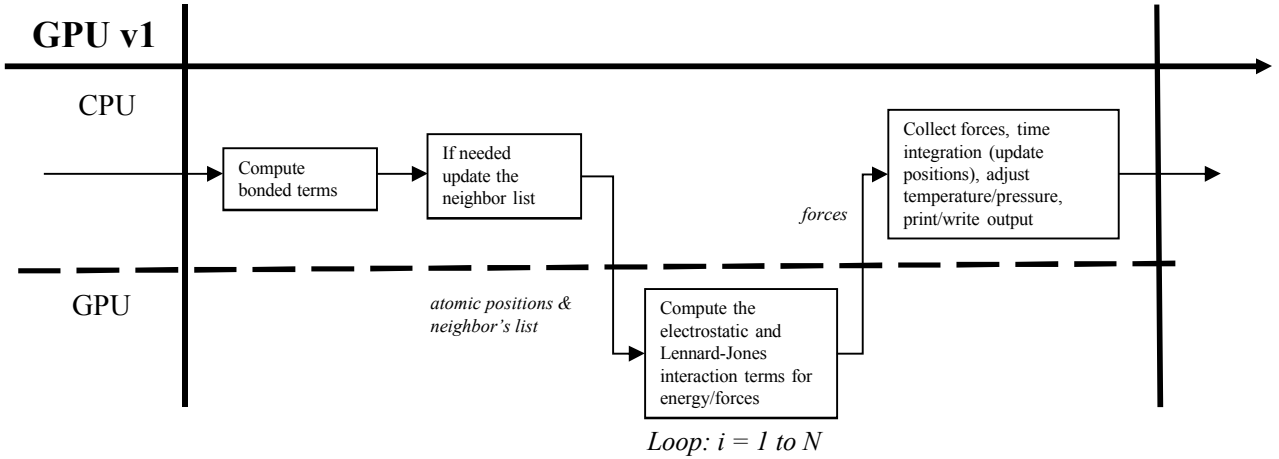
We propose/believe:

- Best time-to-solution will come from not using a single resource but most (or all) heterogeneous resources
- Keep the parallelism that already LAMMPS has
  - Spatial decomposition: Multi-core processors/MPI

Host only

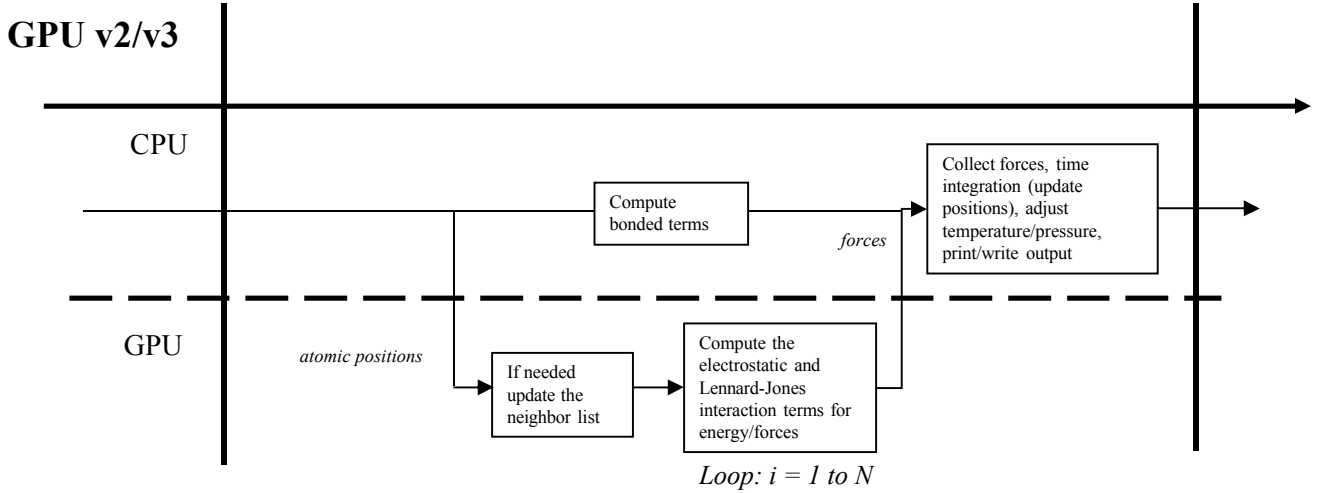


GPU as a Co-processor



GPU as an Accelerator

With concurrent computations on CPU & GPU



GPU-LAMMPS project structure

Developers team

Paul Crozier (1435)
Mike Brown (ORNL)
Arnold Tharrington (ORNL)
Scott Hampton (ORNL)
Axel Kohlmeyer (Temple)
Christian Trott (Ilmenau)
Lars Winterfeld (Ilmenau)
Duncan Poole (Nvidia)
Peng Wang (Nvidia)
~10 others (PSU, RPI, Ga Tech, Mellanox)

Publicity

Presentations at SC09, SC10, LCI HPC conference, SOS14, LAMMPS workshop, ASC threading workshop, ORNL biomolecular sims workshop
Best paper award @ HPCS 2010
Several more publications in preparation
Application for Thuringia, Germany research prize, €17.5k

Collaboration infrastructure

http://lammps.sandia.gov
http://code.google.com/p/gpulammps/
External Subversion repository, wiki pages
E-mail list: gpulammps@sandia.gov
Monthly telecons
Periodic face-to-face meetings
- Nvidia HQ (Aug 2009)
- SC09 in Portland (Nov 2009)
- LAMMPS workshop @ CSRI (Feb 2010)

Funding

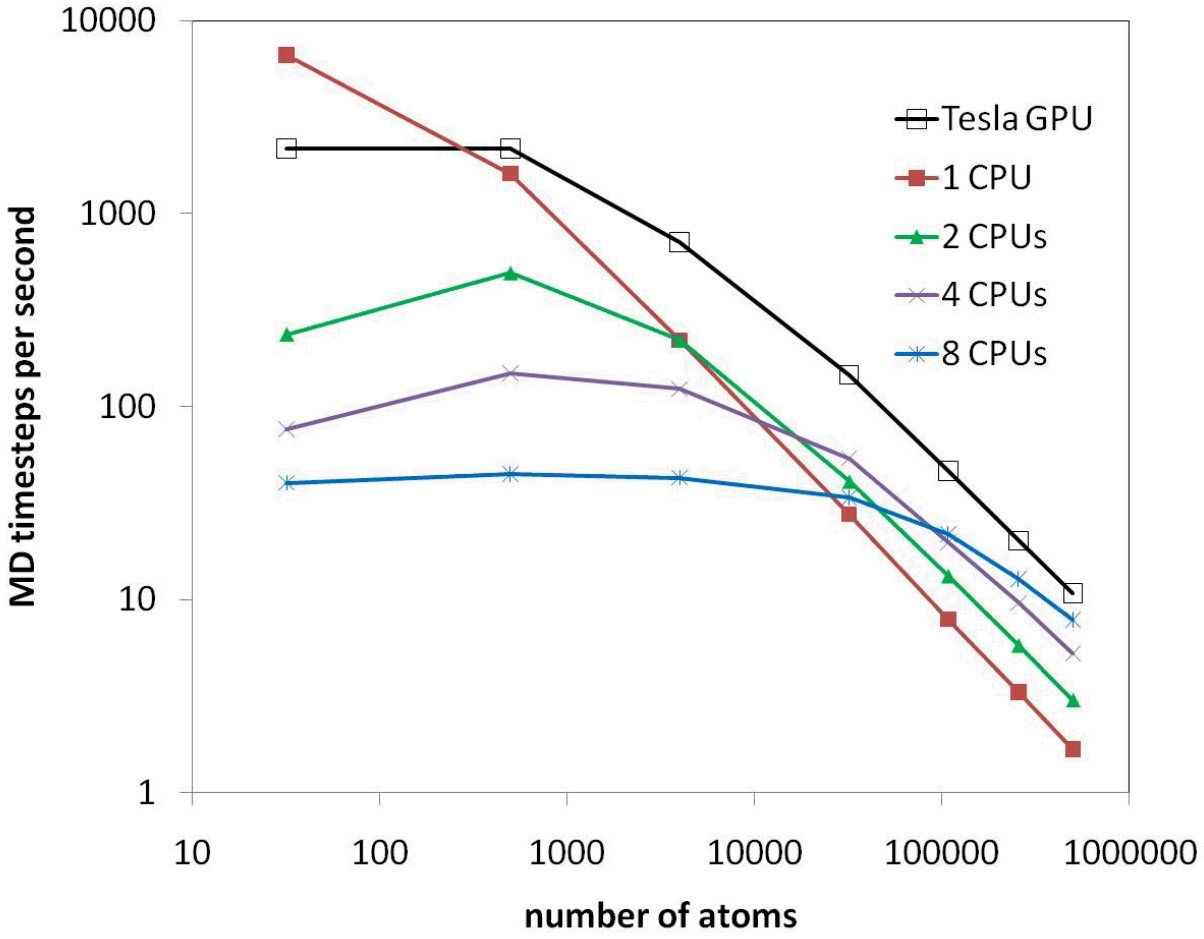
OLCF-3 Apps Readiness
Mantevo project led by Mike Heroux (Sandia)
Most team members have their own minimal funding

Force field coverage

- pairwise potentials: Lennard-Jones, Buckingham, Morse, Yukawa, soft, class 2 (COMPASS), tabulated
- charged pairwise potentials: Coulombic, point-dipole
- manybody potentials: EAM, Finnis/Sinclair EAM, modified EAM (MEAM), Stillinger-Weber, Tersoff, AI-REBO, ReaxFF
- coarse-grained potentials: DPD, GayBerne, REsquared, colloidal, DLVO, cg/cmm
- mesoscopic potentials: granular, Peridynamics
- polymer potentials: all-atom, united-atom, bead-spring, breakable
- water potentials: TIP3P, TIP4P, SPC
- implicit solvent potentials: hydrodynamic lubrication, Debye
- long-range Coulombics and dispersion: Ewald, PPPM (similar to particle-mesh Ewald), Ewald/N for long-range Lennard-Jones
- force-field compatibility with common CHARMM, AMBER, OPLS, GROMACS options

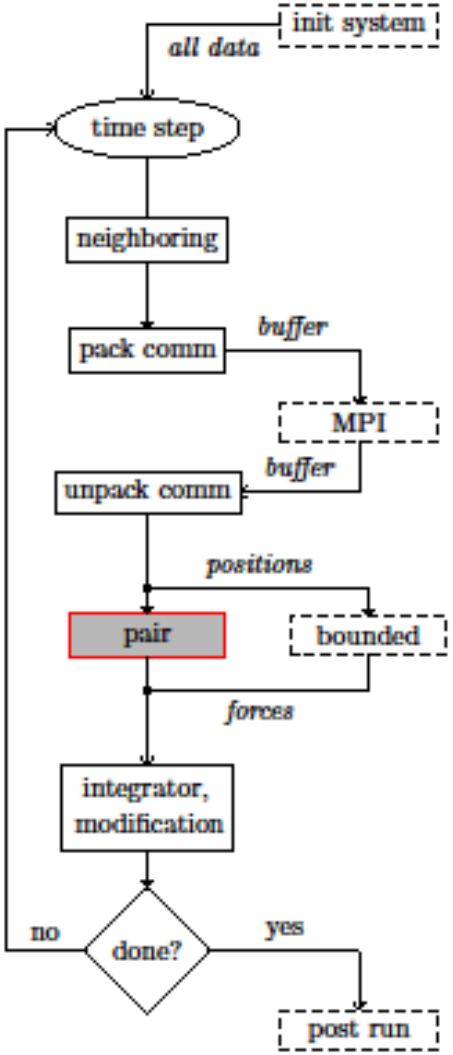
Results for LAMMPS LJ benchmark

- Dual Quad Core Intel® Xeon® Processors X5560 2.8GH
- 1 Tesla C1060 GPU
- Use of Tesla is almost always faster than not using it on this machine.
- Tesla 3.2x faster than Dual Quad Core for 4000 atom system.



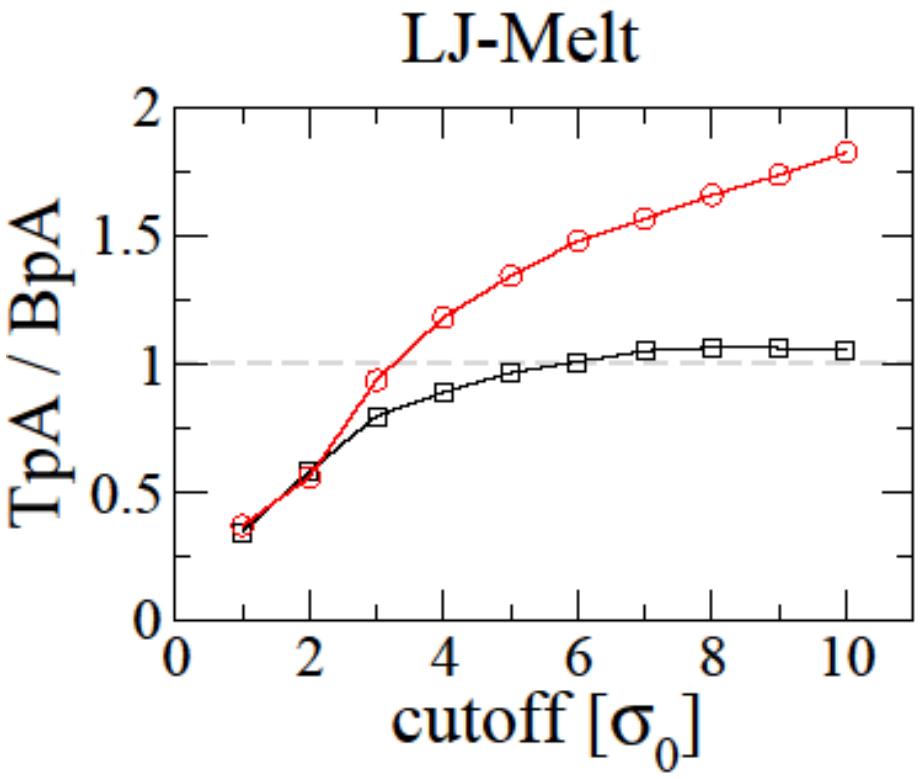
Sandia National Laboratories

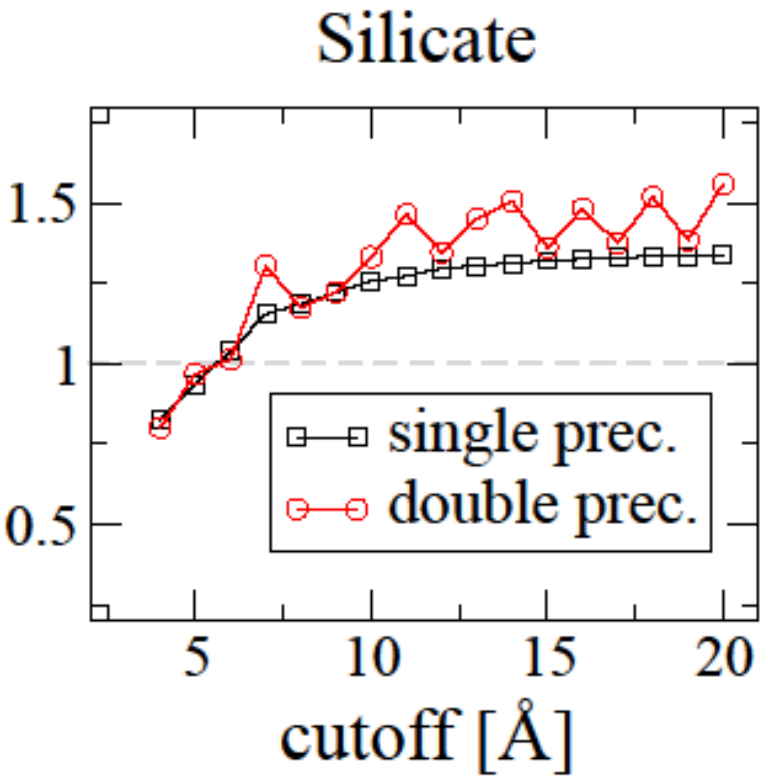
Workflow



Sandia National Laboratories

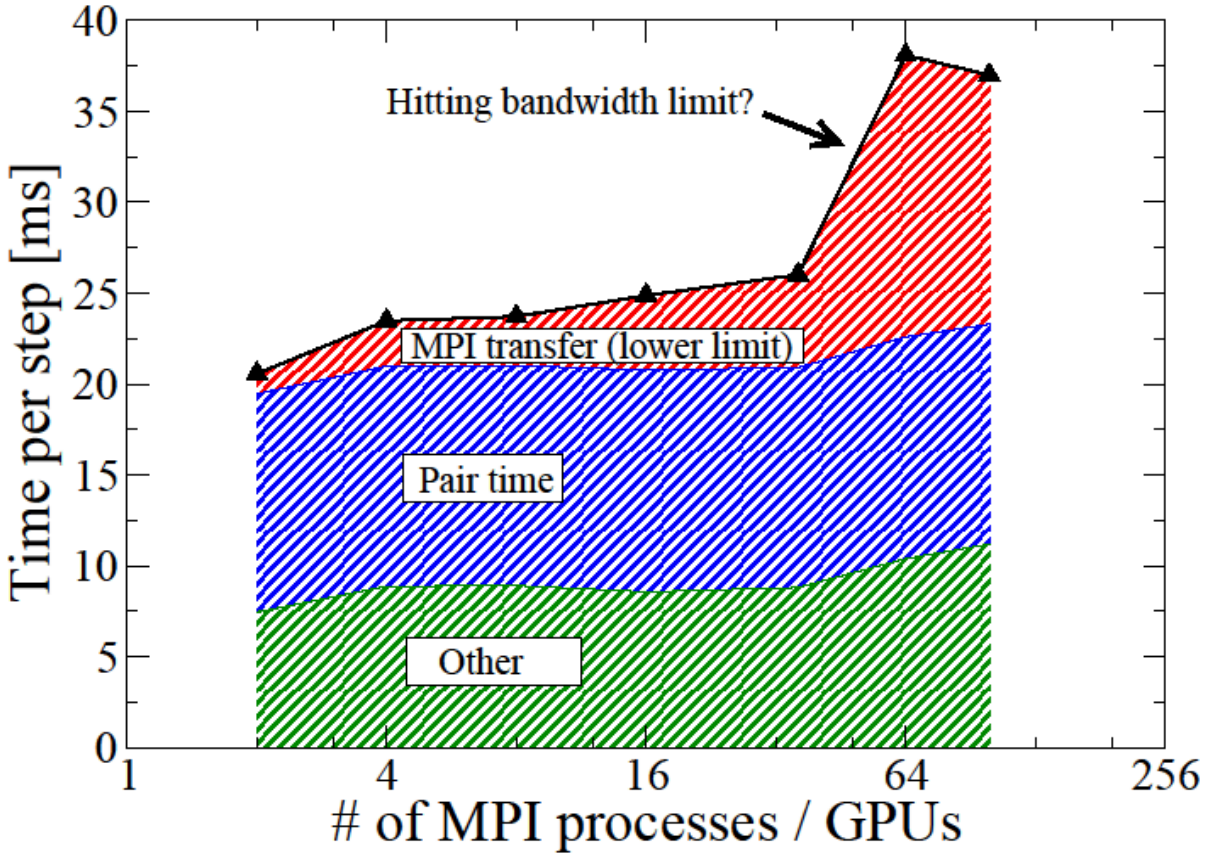
Thread-per-atom vs Block-per-atom





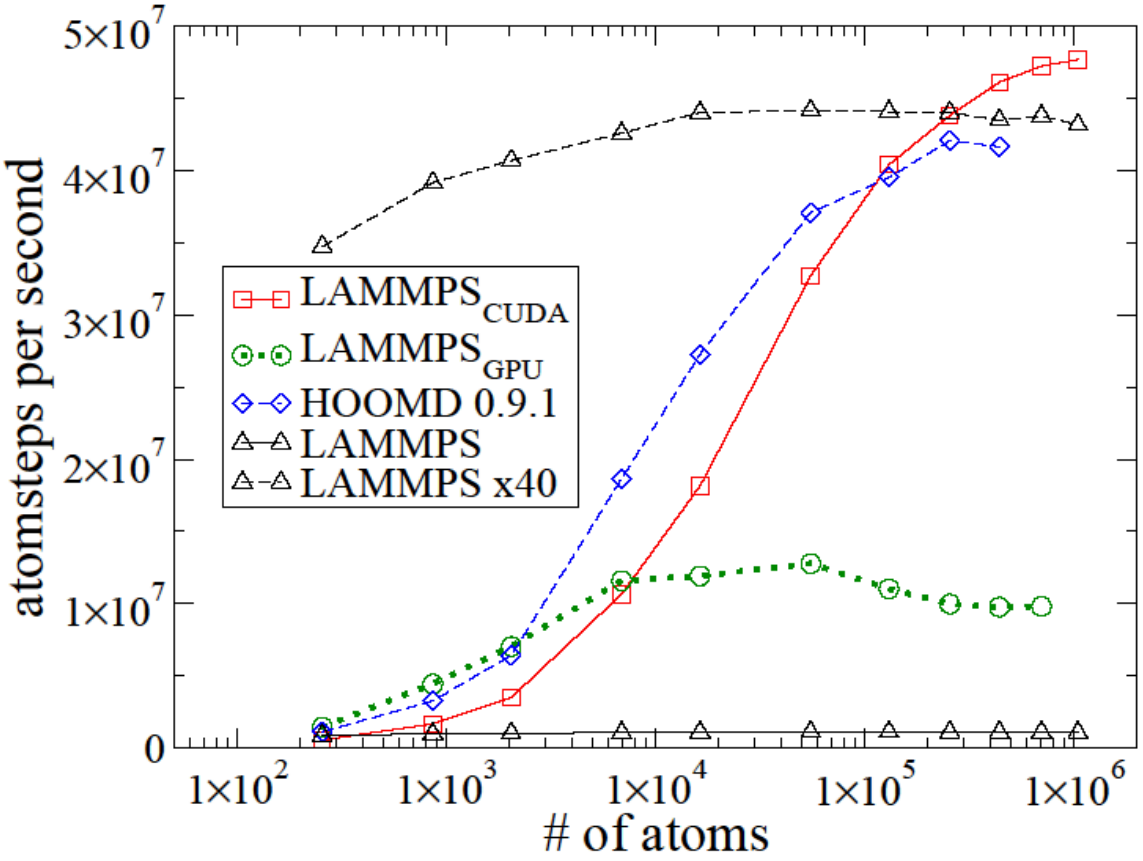
Sandia National Laboratories

Scaling problem: communications costs



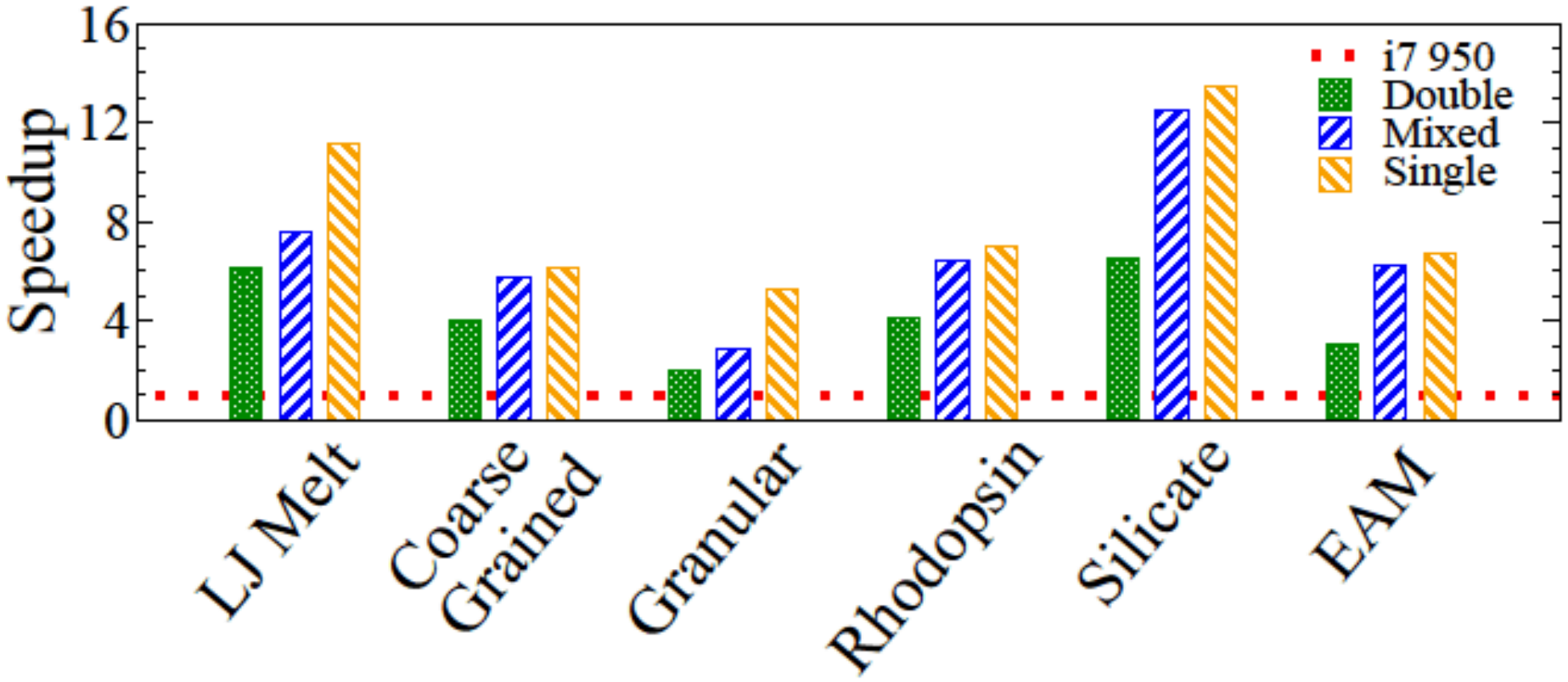
Sandia National Laboratories

Simulation speed vs system size



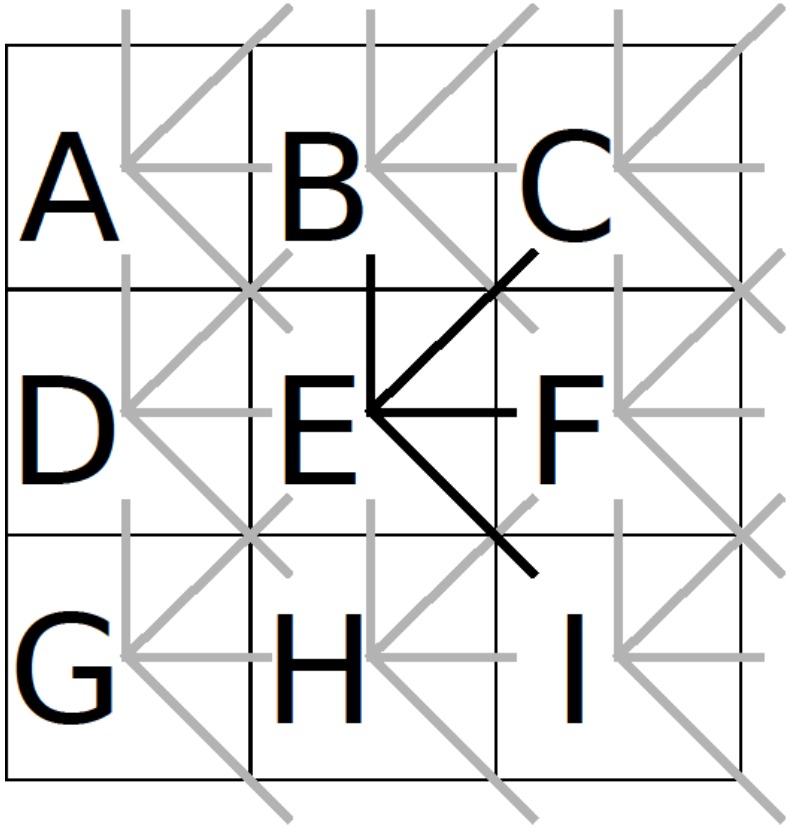
Sandia National Laboratories

GPU speedups on several classes of materials



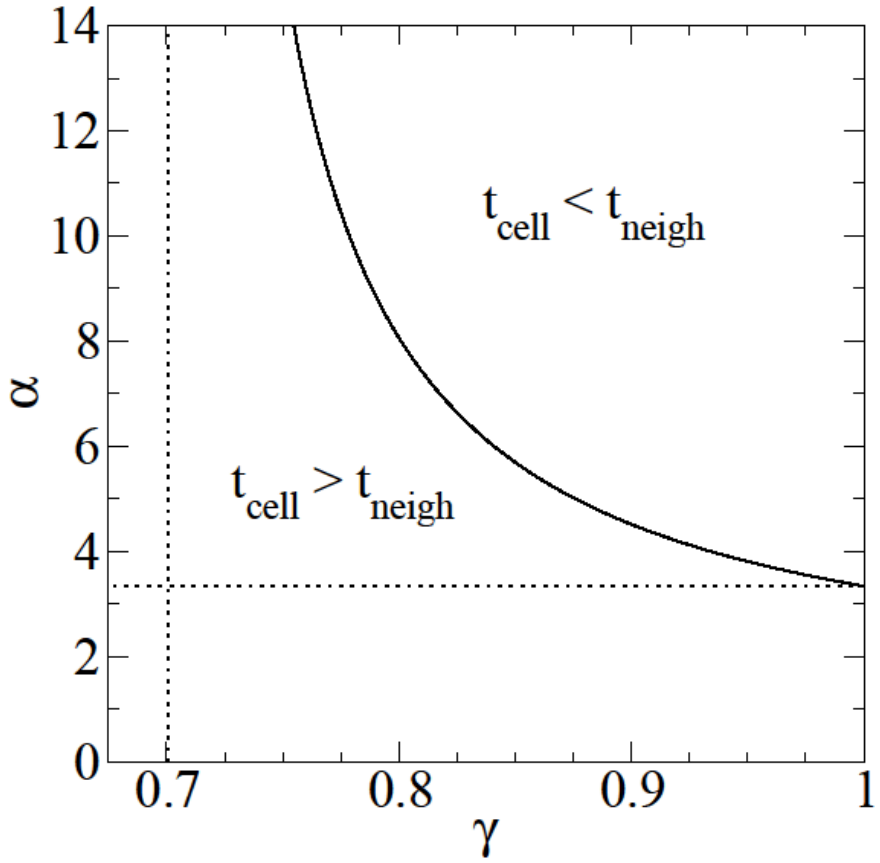
Sandia National Laboratories

Using Newton's 3rd law to cut work



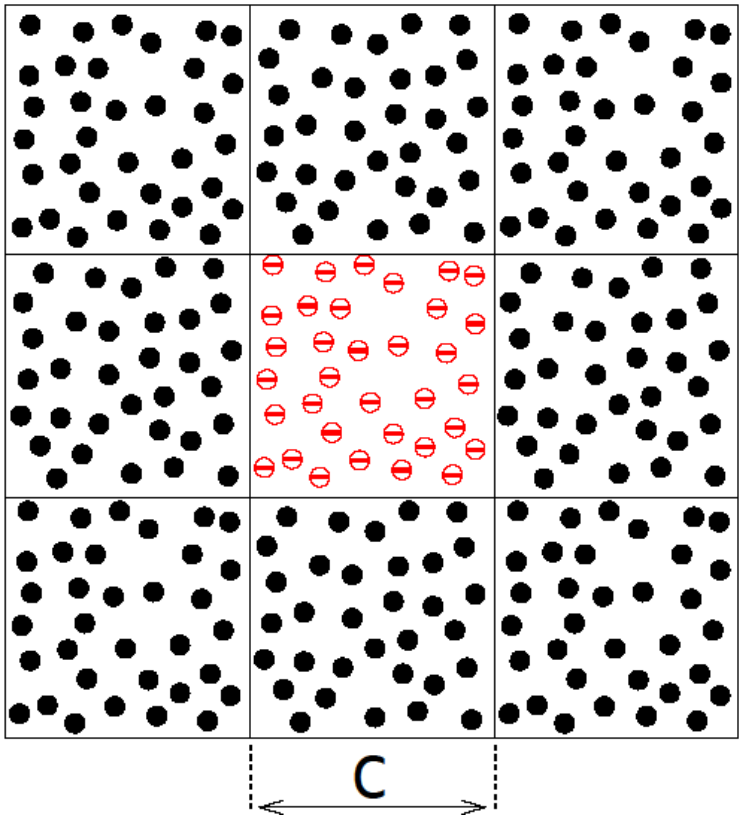
Sandia National Laboratories

Cell list vs neighbor list



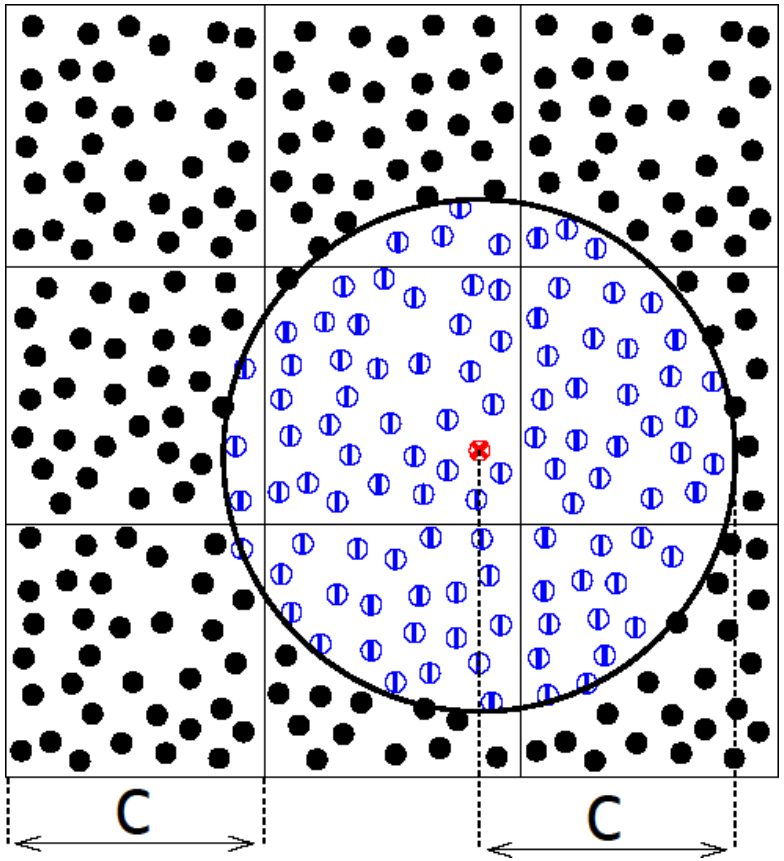
Sandia National Laboratories

Particles in a cell



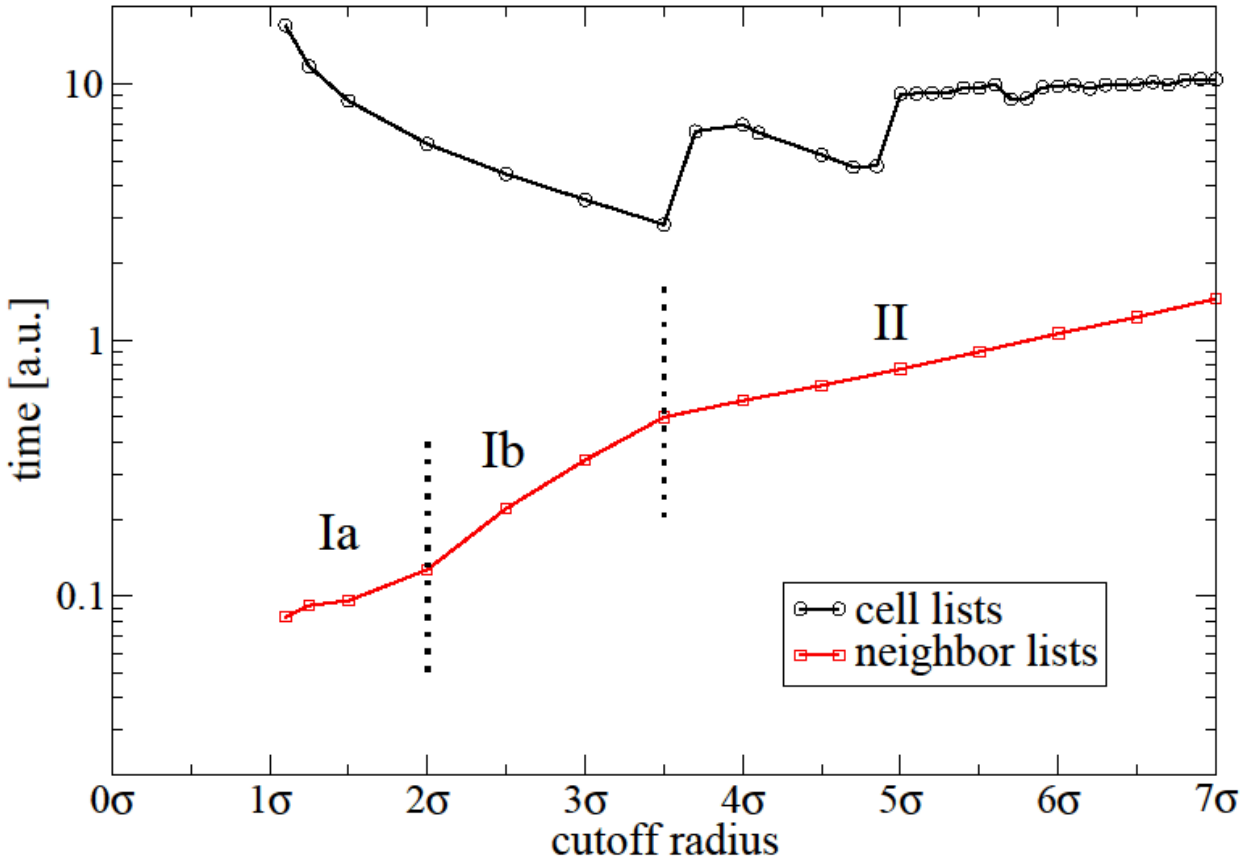
Sandia National Laboratories

Particles in a cell list



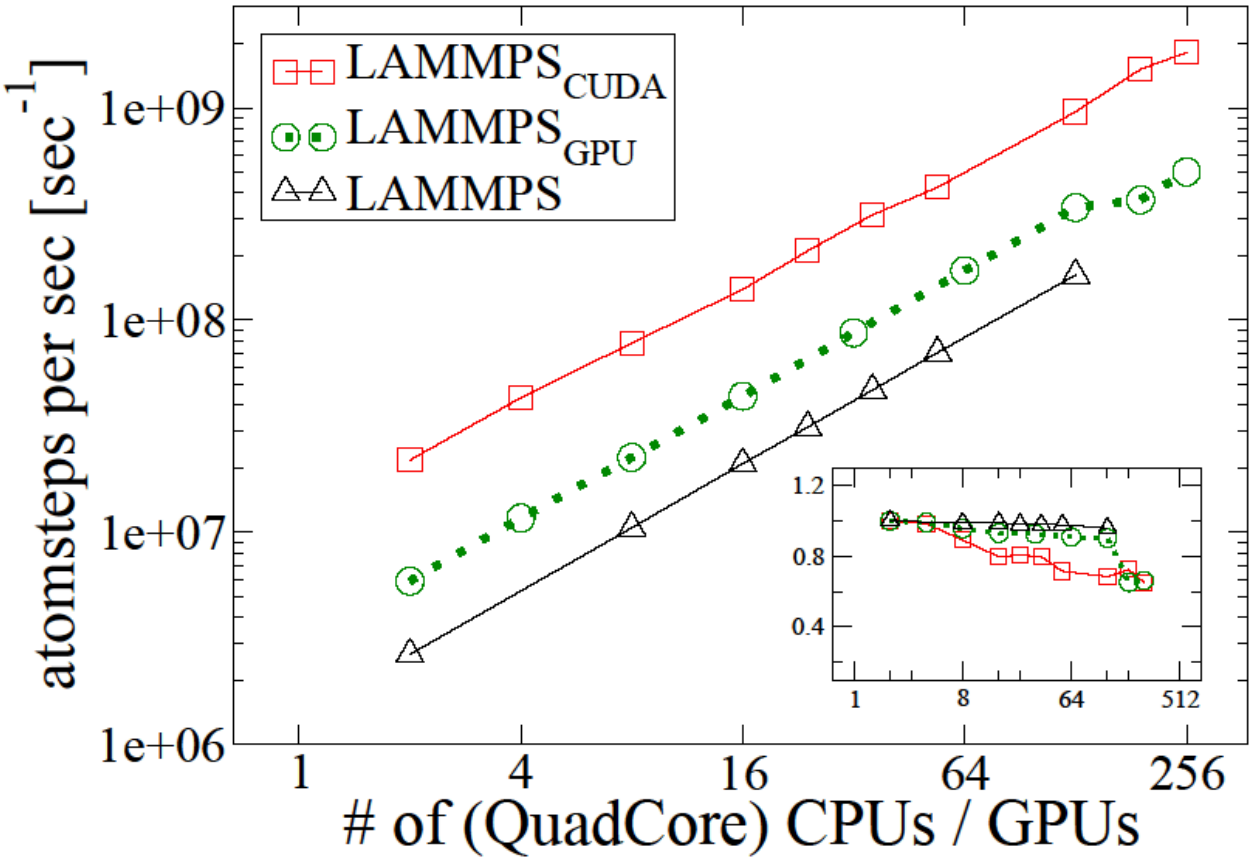
Sandia National Laboratories

Cell list vs neighbor lists



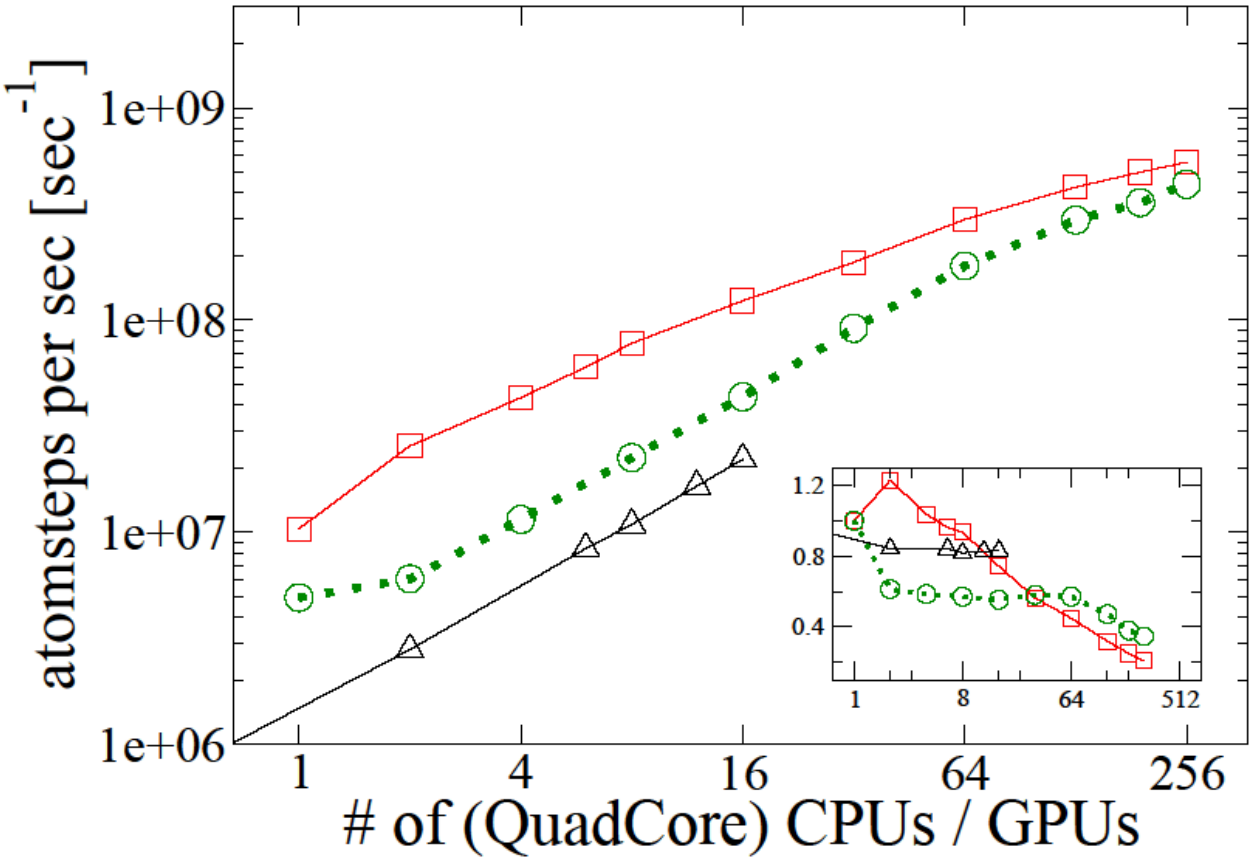
Sandia National Laboratories

LJ, fixed # atoms per node (weak scaling)



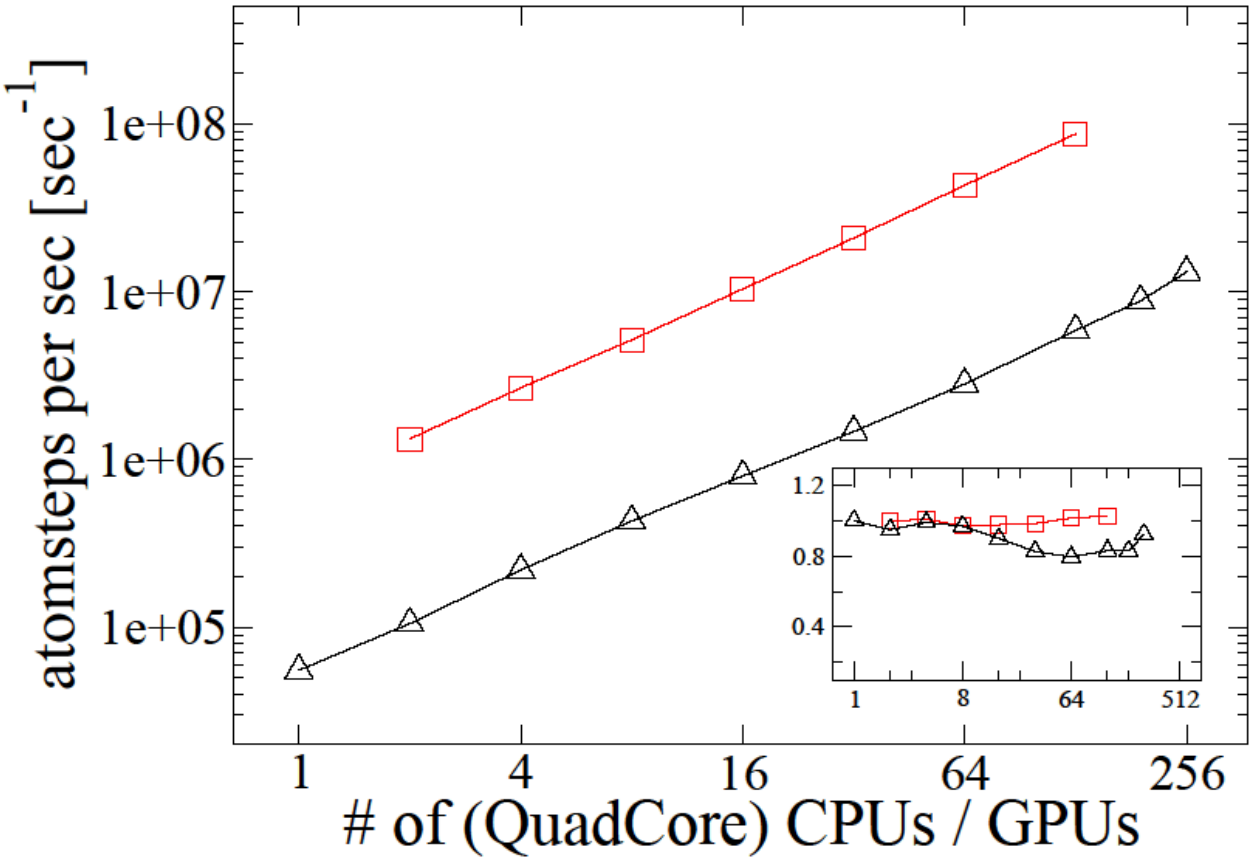
Sandia National Laboratories

LJ, fixed size (strong scaling)



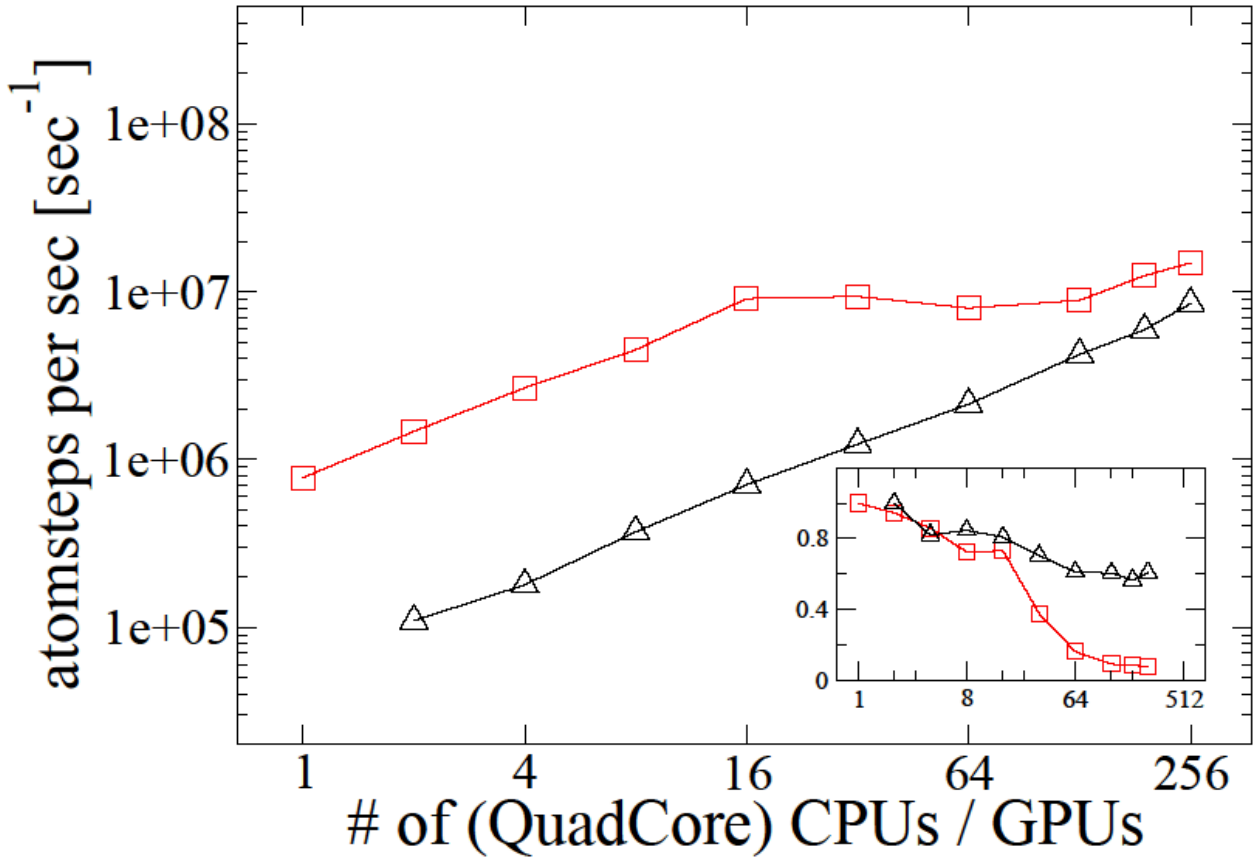
Sandia National Laboratories

Silicate, fixed # atoms per node (weak scaling)

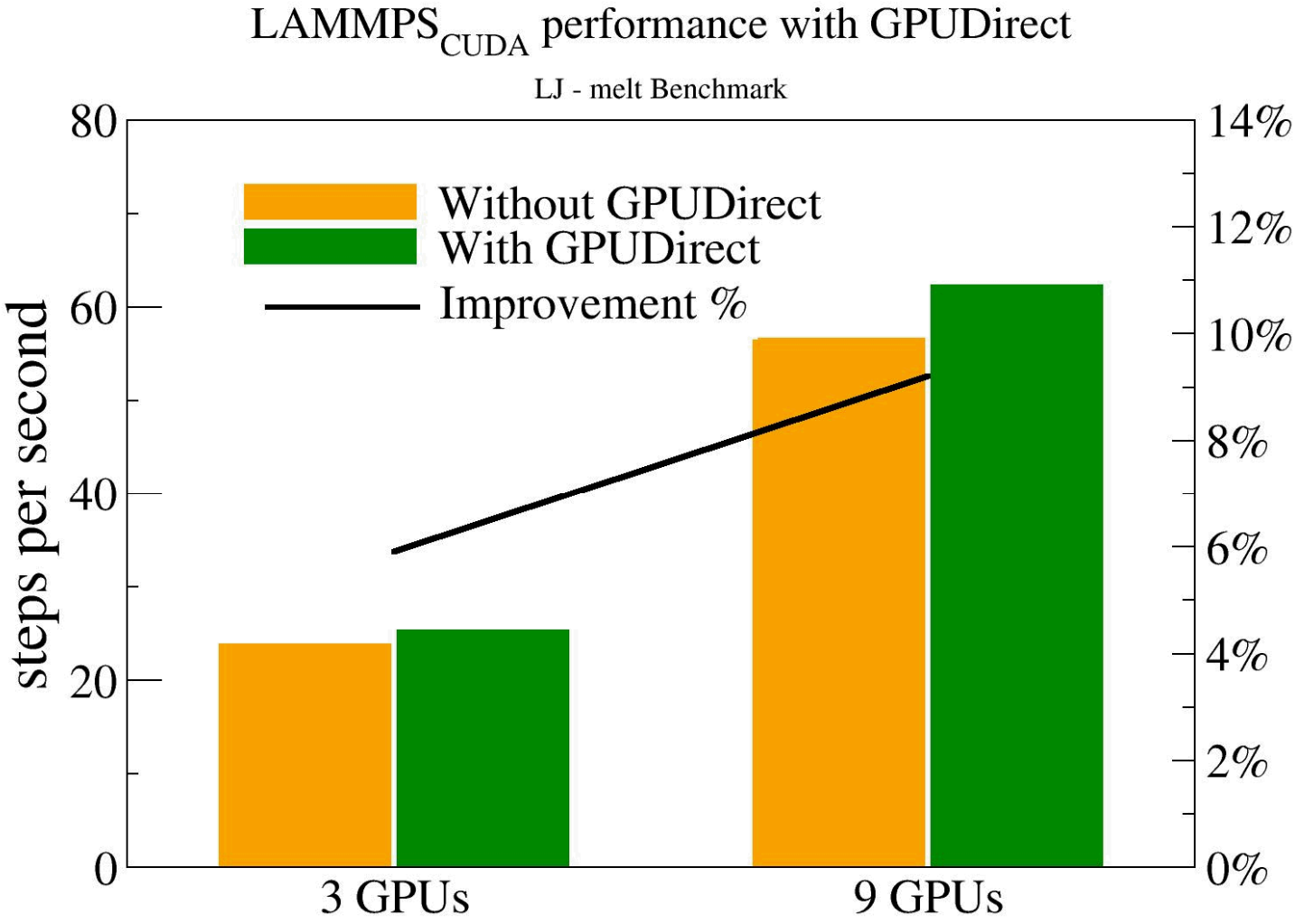


Sandia National Laboratories

Silicate system, fixed size (strong scaling)



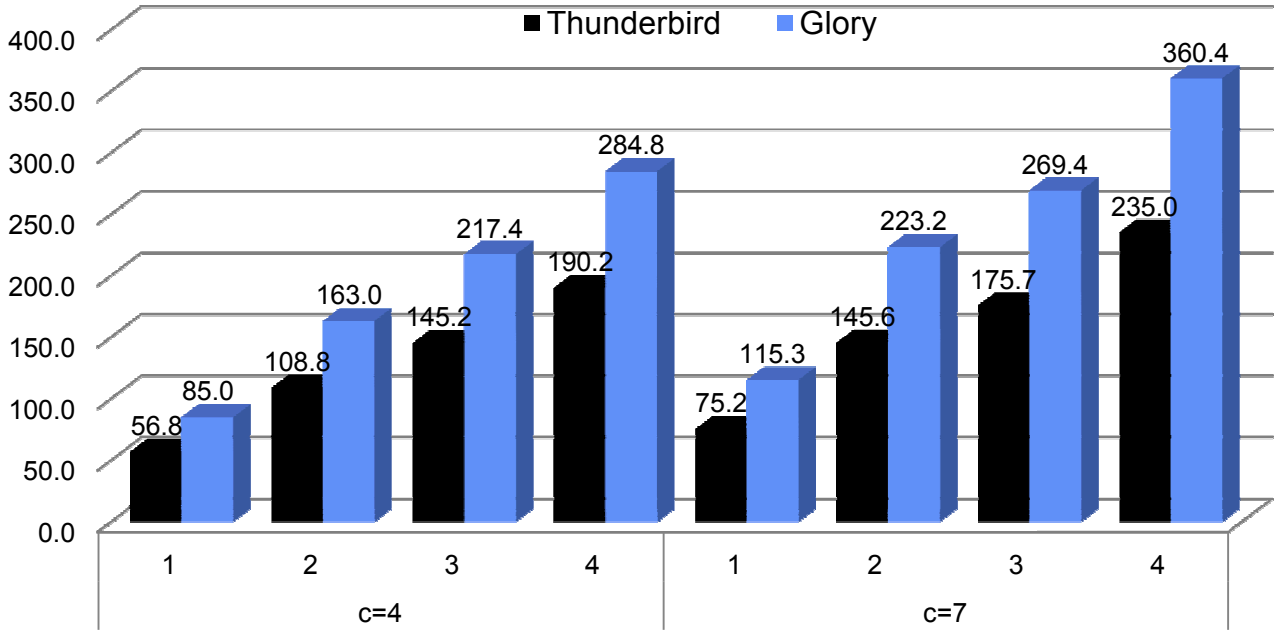
Sandia National Laboratories



Sandia National Laboratories

GPGPU Times Speedup vs 1 Core

(*c*=cutoff, 32768 particles)

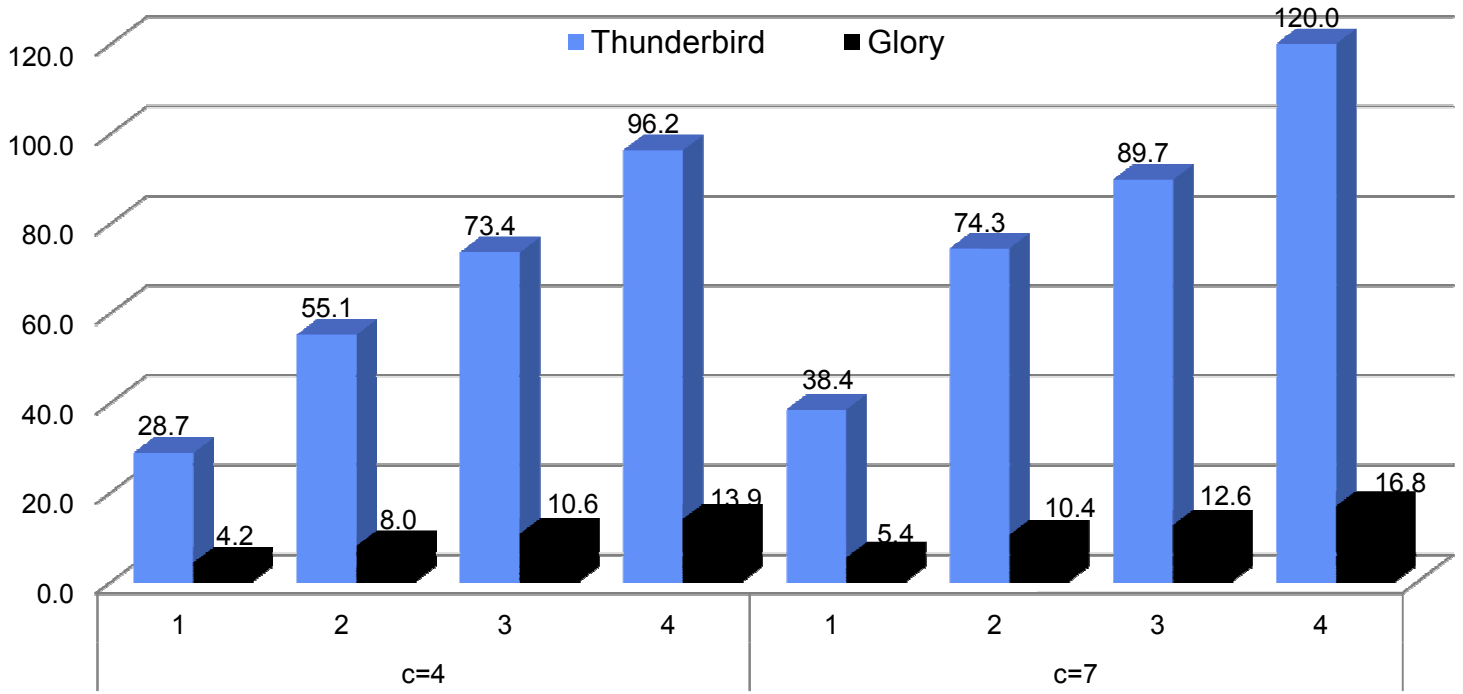


GPGPU: 1, 2, 3, or 4 NVIDIA, 240 core, 1.3 GHz Tesla C1060 GPU(s)
Thunderbird: *1 core of Dual 3.6 GHz Intel EM64T processors*
Glory: *1 core of Quad Socket/Quad Core 2.2 GHz AMD*

Sandia National Laboratories

GPGPU Times Speedup vs 1 Node

(*c*=cutoff, 32768 particles)

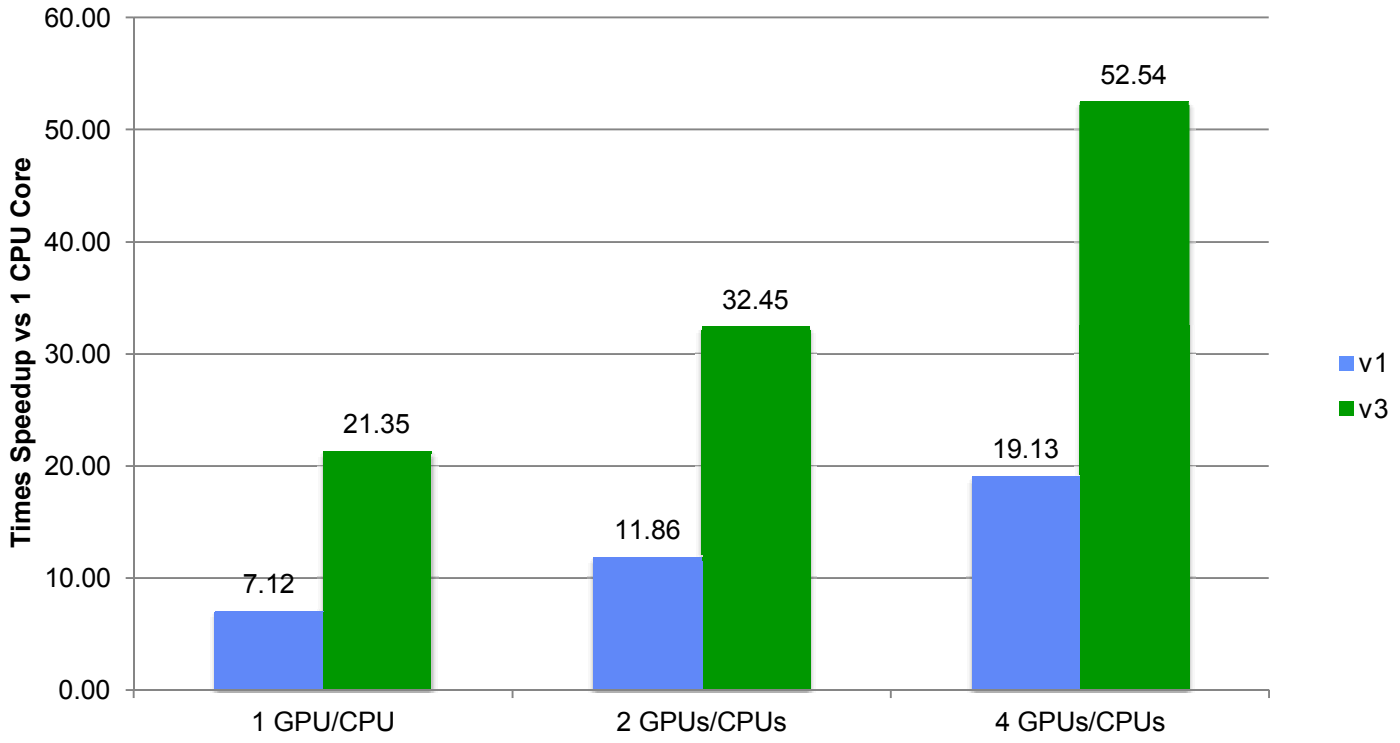


GPGPU: 1, 2, 3, or NVIDIA, 240 core, 1.3 GHz Tesla C1060 GPU(s)
Thunderbird: *2 procs, Dual 3.6 GHz Intel EM64T processors*
Glory: *16 procs, Quad Socket/Quad Core 2.2 GHz AMD*

Sandia National Laboratories

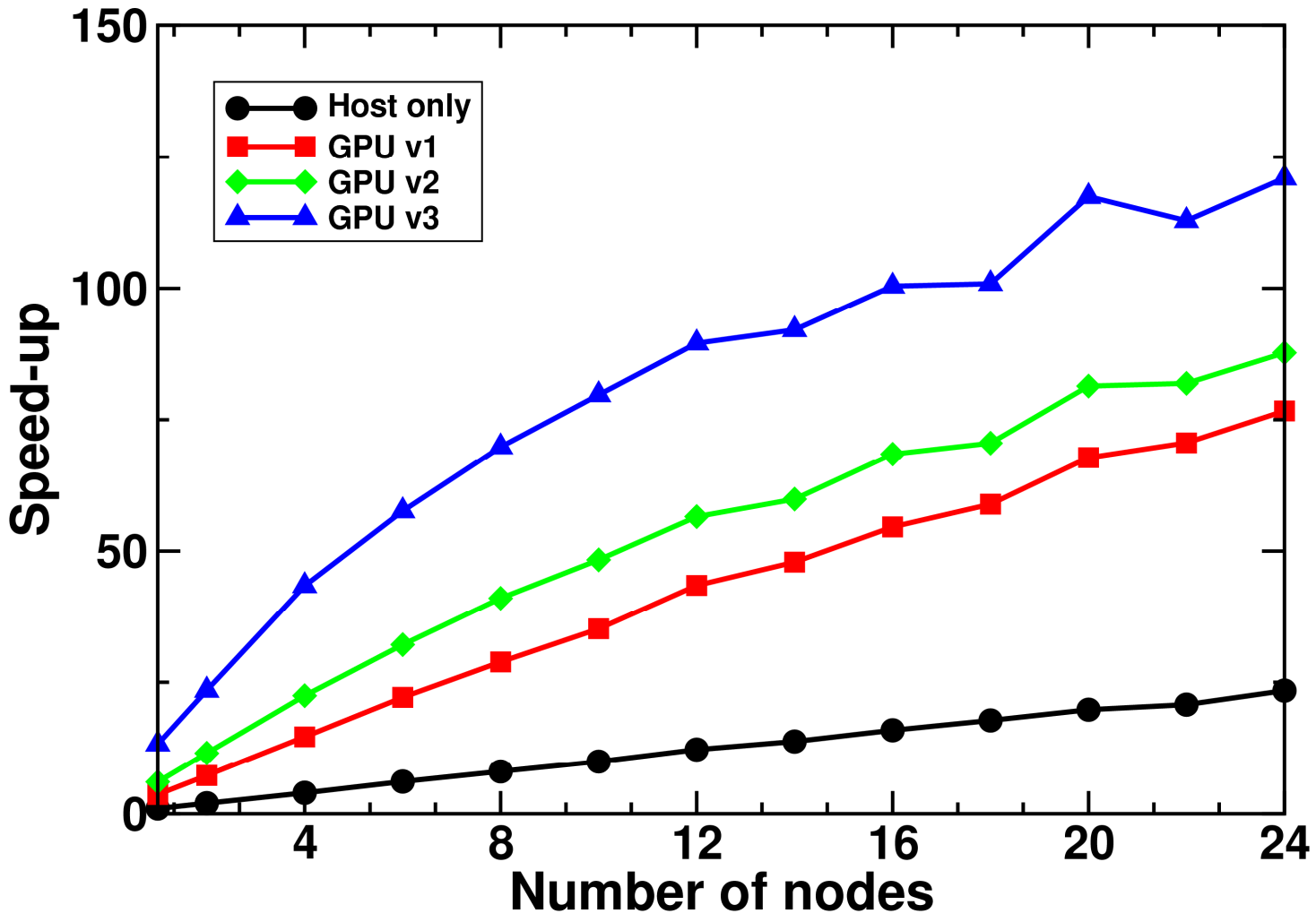
Biomolecular simulation with GPU-LAMMPS

Joint-AMBER-CHARMM Benchmark
Intel Xeon E5540 (2.53 GHz) with Tesla C1060
23,558 atoms



Sandia National Laboratories

GPU-LAMMPS bio-MD speedup on a multi-node CPU/GPU cluster



Sandia National Laboratories

LAMMPS (no GPUs, GPU package, CUDA package) vs. HOOMD on an LJ benchmark

Benchmark parameters

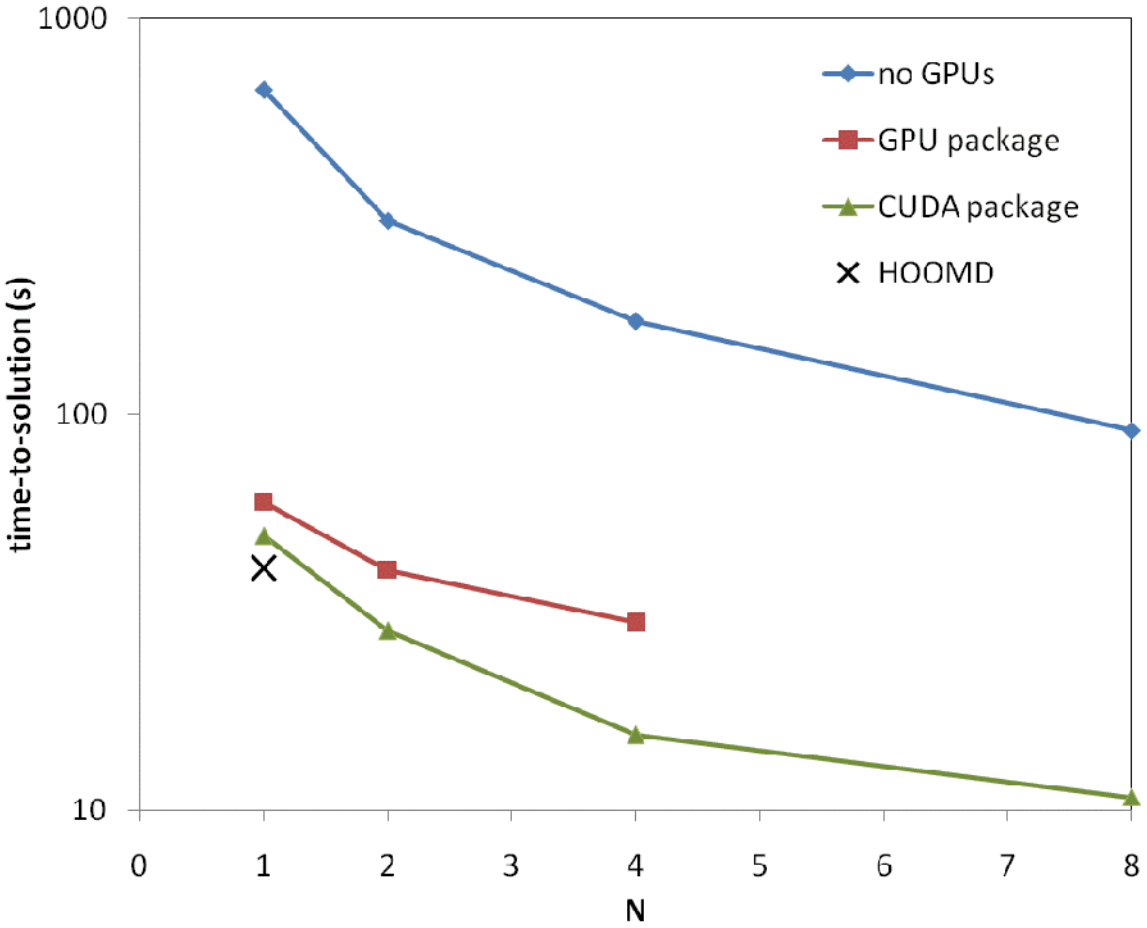
- 442,368 LJ atoms
- 2000 timesteps
- ρ*= 0.4, T* = 0.5

Hardware differences

- "no GPUs" case: N = # of CPUs
- "GPU package" case: N = # of C1060s & CPUs
- "CUDA package" case: N = # of C1060s
- "HOOMD" case: N = # of GTX 280s (slightly faster than C1060s)

Other caveats

- HOOMD would likely perform better with fewer particles and a longer run due to memory access considerations
- Additional parameter tuning might improve performance in several cases



Sandia National Laboratories

Results: *JAC benchmark*
Single workstation

Serial JAC	Intel Xeon E5540 with Tesla C1060 CPU	GPU v1	GPU v2	GPU v3
Pair (%)	*310.8 (93.9)*	*12.91(27.8)*	*13.6 (40.4)*	*8.4 (54.0)*
Bond (%)	5.2 (1.6)	5.1 (10.9)	5.2 (15.3)	5.0 (32.4)
Neigh (%)	12.9 (3.9)	26.5 (57.0)	12.9 (38.1)	0.1 (0.4)
Comm (%)	0.2 (0.1)	0.2 (0.5)	0.2 (0.7)	0.2 (1.5)
Other (%)	1.9 (0.6)	1.8 (3.9)	1.9 (5.5)	1.8 (11.6)
Total (s)	**331.0**	**46.5**	**33.8**	**15.5**

⇐ Non-bonded

Joint AMBER-CHARMM
23,558 atoms
(Cut-off based)

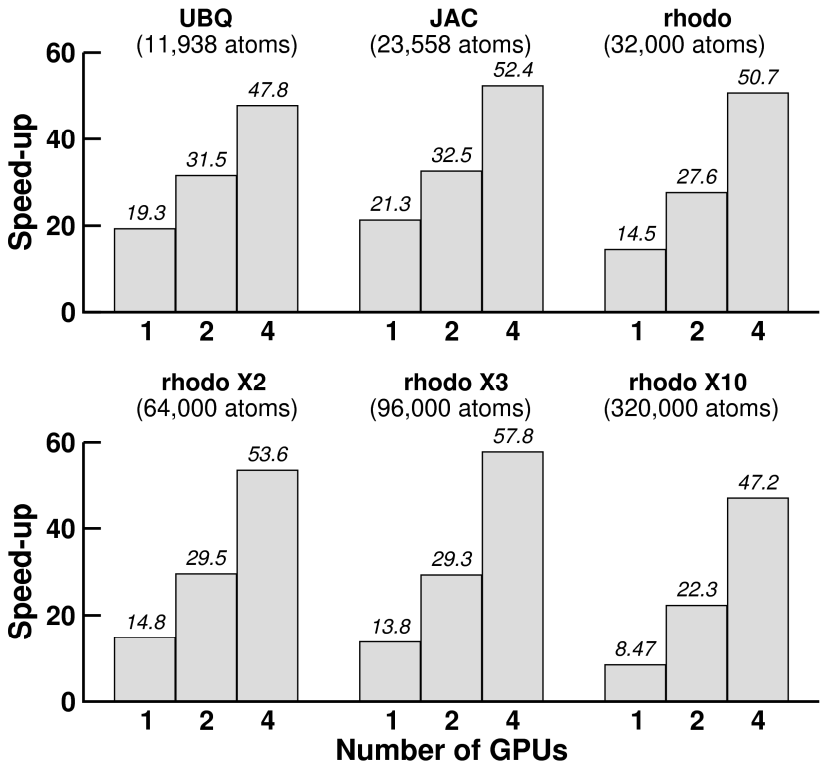
Intel Xeon E5540
2.53 GHz

2 GPUs JAC	Intel Xeon E5540 with Tesla C1060 CPU	GPU v1	GPU v2	GPU v3
Pair (%)	*162.6 (78.1)*	*6.5 (23.4)*	*7.0 (34.9)*	*4.6 (45.0)*
Bond (%)	2.9 (1.4)	2.6 (9.2)	2.6 (12.7)	2.6 (25.2)
Neigh (%)	6.7 (3.2)	12.9 (46.4)	6.4 (31.5)	0.0 (0.3)
Comm (%)	34.8 (16.7)	4.7 (16.9)	3.1 (15.2)	1.9 (18.3)
Other (%)	1.2 (0.6)	1.1 (4.1)	1.1 (5.7)	1.1 (11.1)
Total (s)	**208.2**	**27.9**	**20.2**	**10.2**

4 GPUs	Intel Xeon E5540 with Tesla C1060			
Pair (%)	*76.4 (58.1)*	*3.3 (19.1)*	*3.7 (29.5)*	*2.7 (43.3)*
Bond (%)	1.3 (1.0)	1.3 (7.3)	1.3 (10.2)	1.3 (20.0)
Neigh (%)	3.1 (2.4)	6.2 (35.8)	3.1 (24.6)	0.0 (0.3)
Comm (%)	50.1 (38.0)	5.8 (33.6)	3.8 (30.1)	1.6 (25.1)
Other (%)	0.7 (0.6)	0.7 (4.0)	0.7 (5.6)	0.7 (11.1)
Total (s)	**131.6**	**17.3**	**12.6**	**6.3**

Performance: Single-node (multi-GPUs)

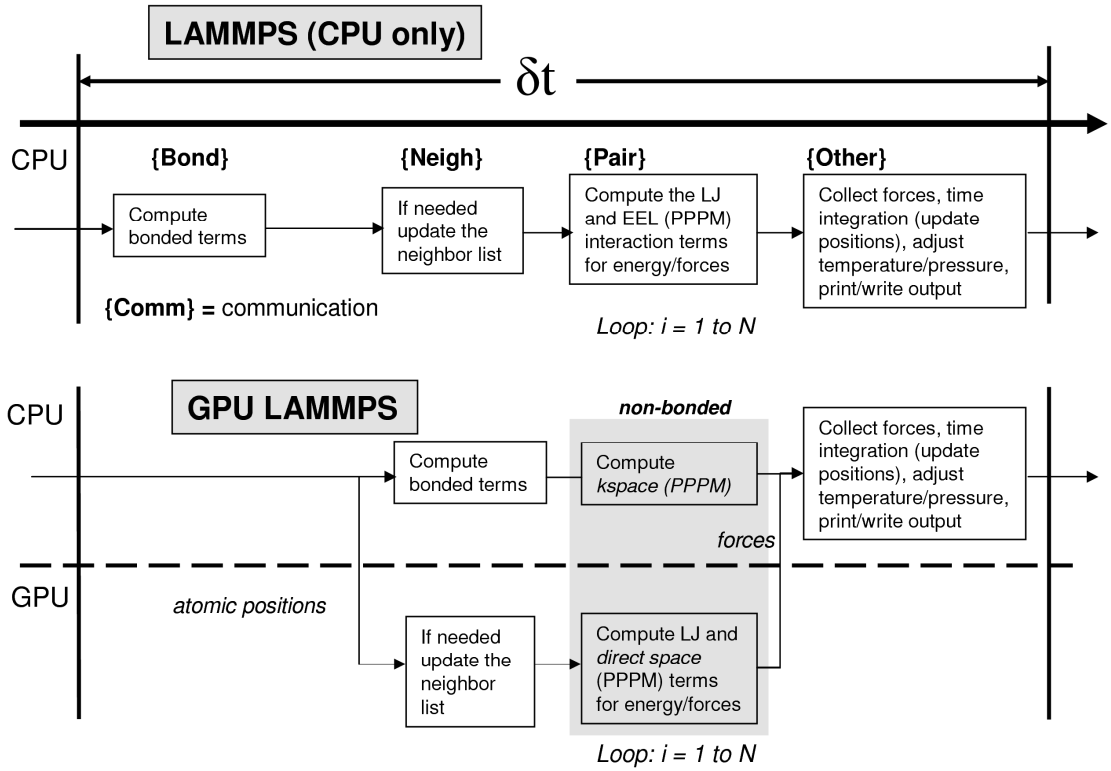
- Single workstation with 4 *Tesla C1060* cards
- 10-50X speed-ups, larger systems – data locality
- Super-linear speed-ups for larger systems
- Beats 100-200 cores of ORNL Cray XT5 (#1 in Top500)

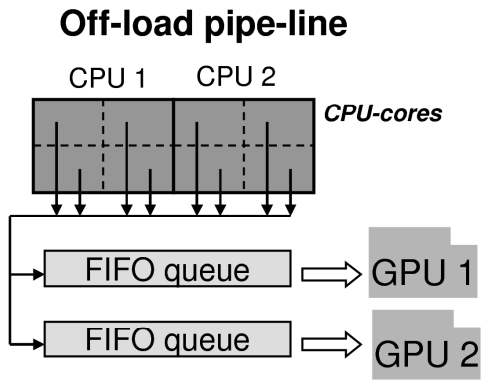


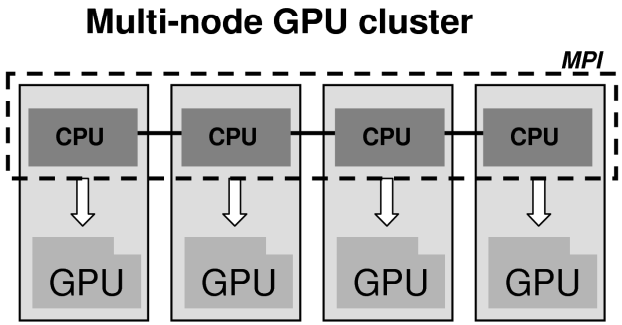
Performance metric (ns/day)

CPU-cores/ GPUs	UBQ (11,938 atoms)		JAC (23,558 atoms)		Rhodo (32,000 atoms)	
	GPU	XT5	GPU	XT5	GPU	XT5
1	15.11		8.80		2.94	
2	23.45		13.35		5.51	
4	35.12		21.58		10.50	
32		16.95		6.63		7.67
64		24.88		11.58		14.11
128		32.75		19.94		22.99
256		39.15		30.11		29.97
Cores/ GPUs	**rhodoX2 (64,000 atoms)**		**rhodoX3 (96,000 atoms)**		**rhodoX10 (320,000 atoms)**	
1	1.32		0.79		0.12	
2	2.79		1.71		0.36	
4	5.03		3.61		0.88	
32		4.09		2.76		0.82
64		7.64		5.38		1.63
128		13.98		9.72		3.32

Pipelining: scaling on multi-core/multi-node with GPUs







Challenge: How do all cores use a single (or limited) GPUs?

Solution: Use pipelining strategy*

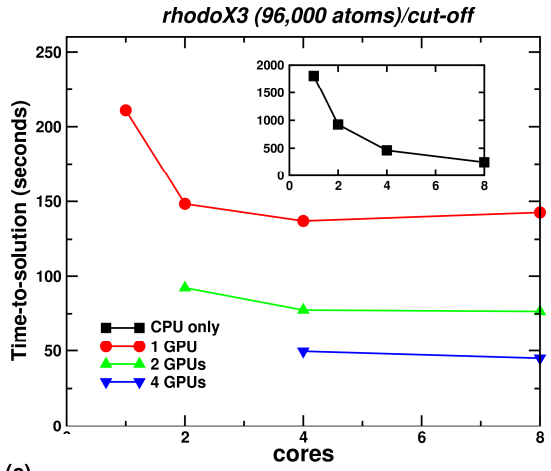
* = Hampton, S. S.; Alam, S. R.; Crozier, P. S.; Agarwal, P. K. (2010), Optimal utilization of heterogeneous resources for biomolecular simulations. *Supercomputing 2010.*

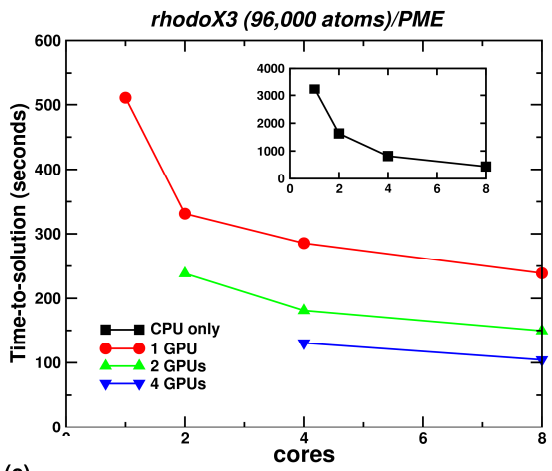
Sandia National Laboratories

Results: Communications overlap

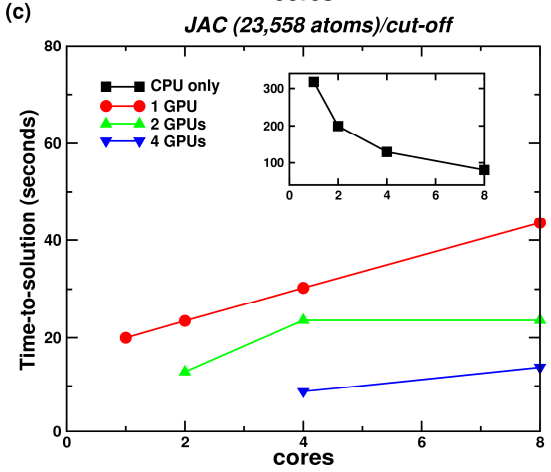
Need to overlap off-node/on-node communications

– Very important for strong scaling mode

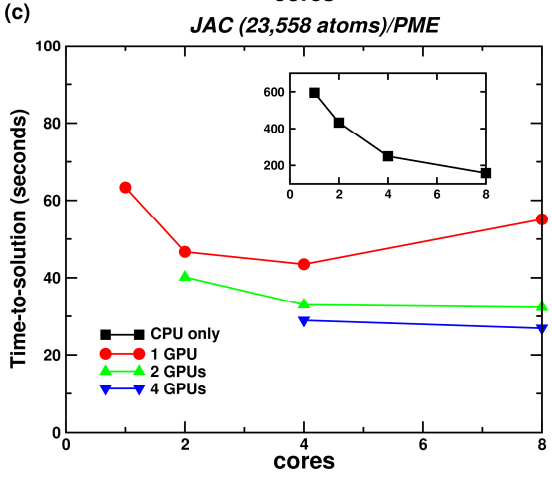




(c)



(c)



2 Quad-core Intel Xeon E5540 2.53 GHz

Important lesson: Off/on node communication overlap

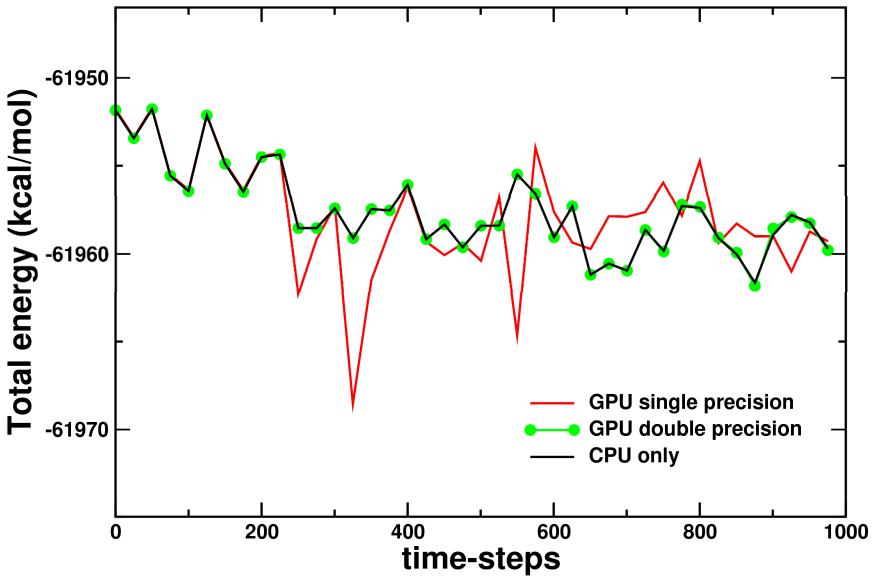
less off-node communication

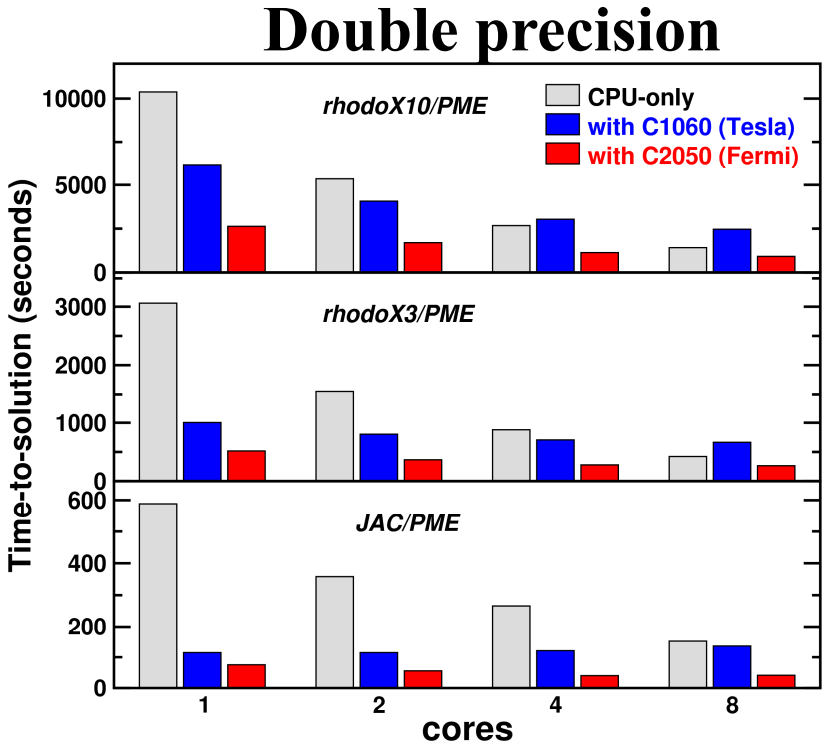
more off-node communication

Sandia National Laboratories

Results: NVIDIA's *Fermi*

- Early access to *Fermi* card: single vs double precision
- *Fermi*: ~6X more double precision capability than Tesla series
- Better and more stable MD trajectories





Intel Xeon E5520 (2.27 GHz)

Sandia National Laboratories

Results: GPU cluster

- 24-nodes Linux cluster: 4 quad CPUs + 1 Tesla card per node
  - AMD Opteron 8356 (2.3 GHz), Infiniband DDR
- Pipelining allows all up to 16 cores to off-load to 1 card
- Improvement in time-to-solution

Protein in water
320,000 atoms
(Long range
electrostatics)

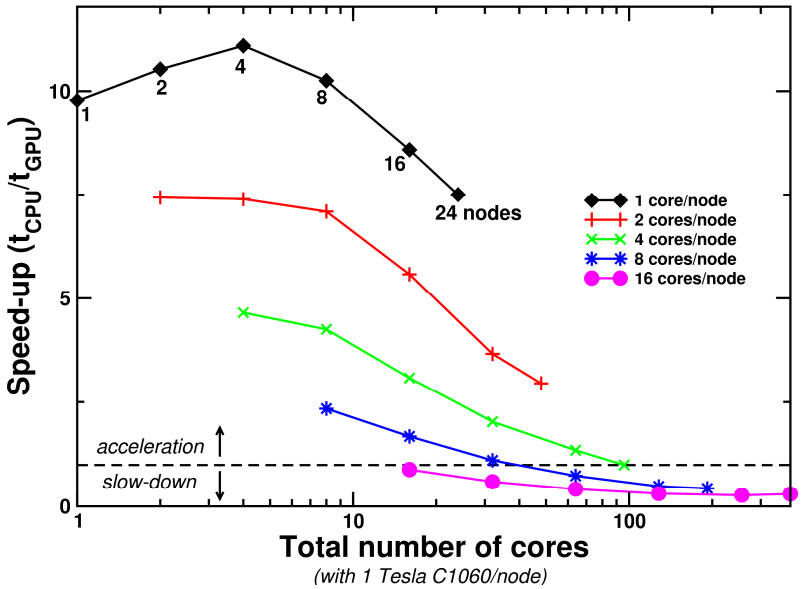
LAMMPS (CPU-only)					
Nodes	1 c/n	2 c/n	4 c/n	8 c/n	16 c/n
1	15060.0	7586.9	3915.9	2007.8	1024.1
2	7532.6	3927.6	1990.6	1052.9	580.5
4	3920.3	1948.1	1028.9	559.2	302.4
8	1956.0	1002.8	528.1	279.5	192.6
16	992.0	521.0	262.8	168.5	***139.9****
24	673.8	335.0	188.7	145.1	214.5
GPU-enabled LAMMPS (1 C1060/node)					
Nodes	**1 c/n**	**2 c/n**	**4 c/n**	**8 c/n**	**16 c/n**
1	3005.5	1749.9	1191.4	825.0	890.6
2	1304.5	817.9	544.8	515.1	480.6
4	598.6	382.2	333.3	297.0	368.6
8	297.9	213.2	180.1	202.0	311.7
16	167.3	*126.7*	*118.8*	176.5	311.1
24	*111.1*	*89.2**	*108.3*	196.3	371.1

Sandia National Laboratories

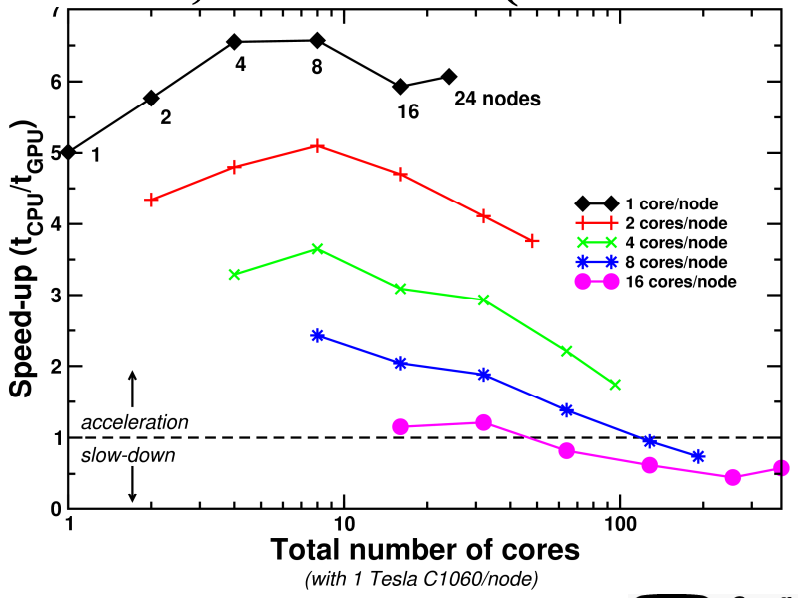
Results: GPU cluster

- Optimal use: matching algorithm with hardware
- The best time-to-solution comes from multi-level parallelism
  - Using CPUs AND GPUs
- Data locality makes a significant impact on the performance

96,000 atoms (cut-off)



320,000 atoms (PPPM/PME)

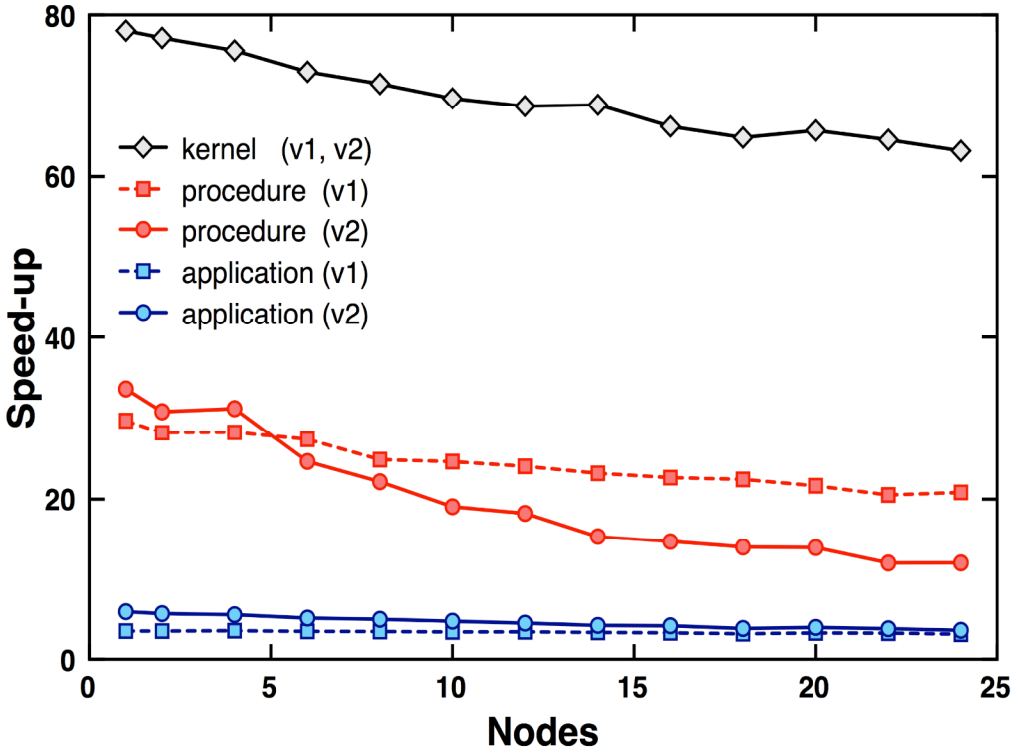


Cut-off = short range forces only

PME = Particle mesh Ewald method for long range

Sandia National Laboratories

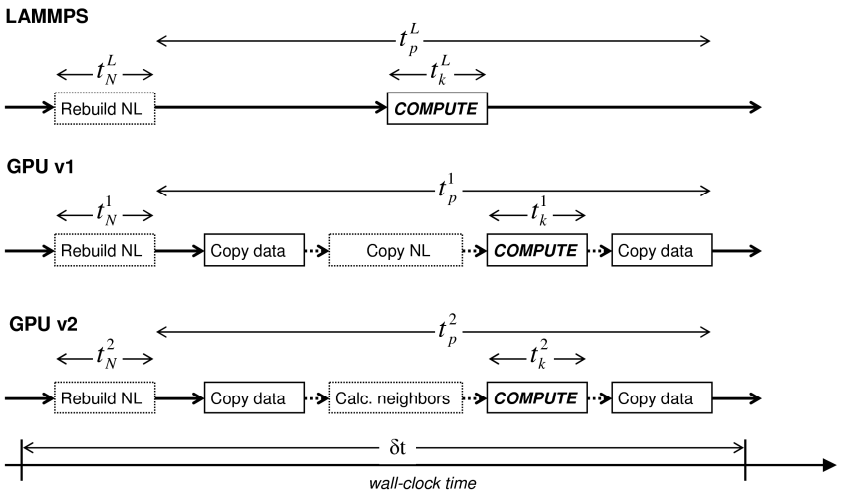
Performance modeling



Theoretical limit (computations only)

Off-loading non-bonded (data transfer included)

Entire simulation



- Kernel speedup = time to execute the doubly nested for loop of the compute() function (without cost of data transfer)
- Procedure speedup = time for the compute() function to execute, including data transfer costs in the GPU setup
- Overall speedup = run-time (CPU only) / run-time (CPU + GPU)

Sandia National Laboratories

Long range electrostatics

- Gain from multi-level hierarchy
- Matching the hardware features with software requirements
- On GPUs
  - Lennard-Jones and short range terms (direct space)
  - Less communication and more computationally dense
- Long range electrostatics
  - Particle mesh Ewald (PME)
  - Requires fast Fourier transforms (FFT)
  - Significant communication
  - Keep it on the multi-core

Sandia National Laboratories

Should a block or a thread be assigned to each atom?

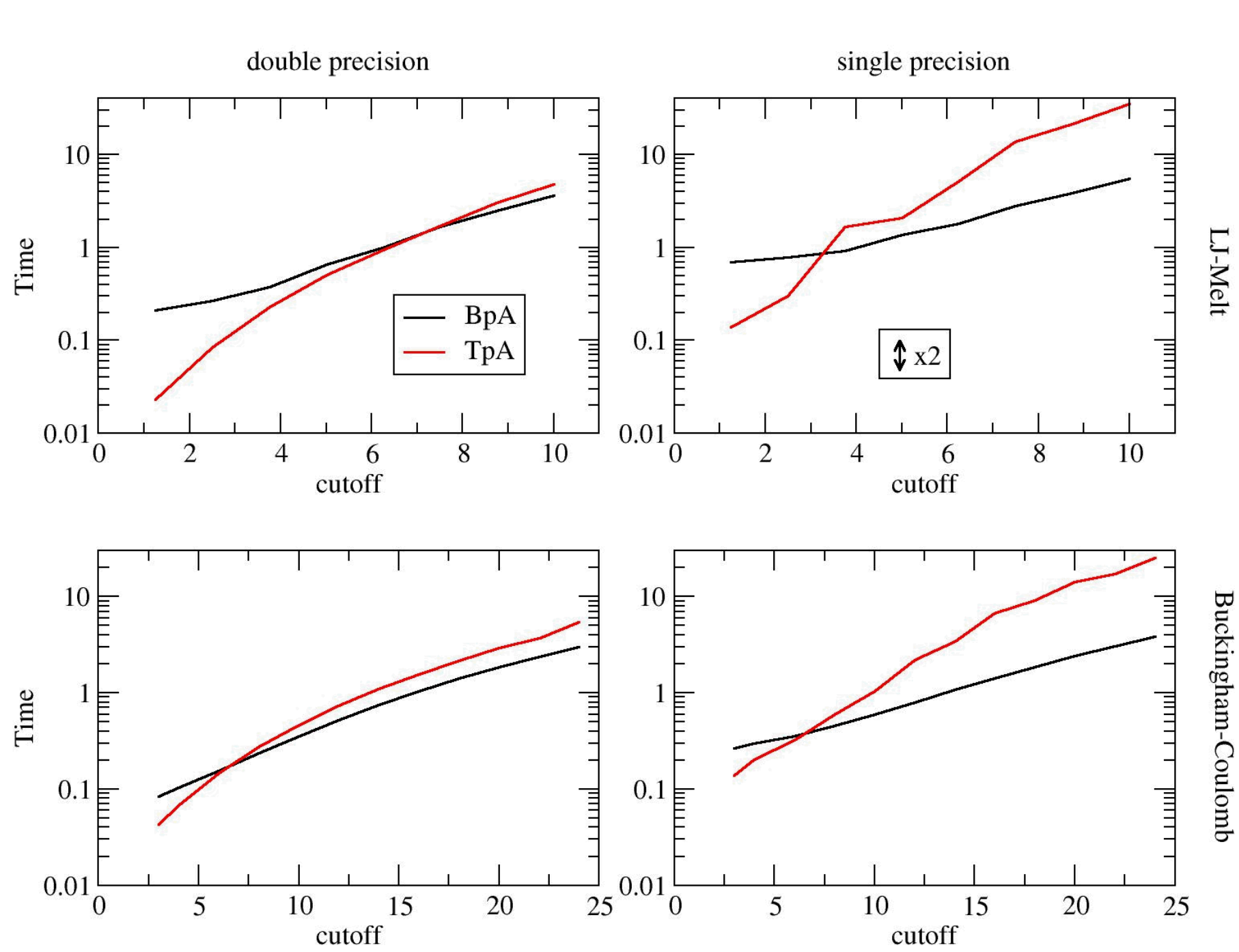
(Note that this question wouldn't even be asked for a legacy CPU code.)

BpA = Block per Atom
- A block of threads is assigned to each atom
- Each thread then computes a single atom-atom interaction

TpA = Thread per Atom
- A single thread is assigned to each atom
- Each thread then computes all of that atom's interactions with other atoms

➢ Which approach will be faster?

Sandia National Laboratories


double precision
single precision
LJ-Melt
Buckingham-Coulomb
Time
cutoff
BpA
TpA
x2
0.01
0.1
1
10
0
2
4
5
6
8
10
15
20
25

Results of BpA vs TpA study

- Speed differences up to 6.5x observed
- TpA is faster when below a critical cutoff distance, $rcut_{critical}$
- $rcut_{critical}$ depends on force field, precision, hardware, and N_{atoms}

Why is TpA faster when atoms have fewer neighbors?

- BpA has more overhead than TpA (you have to start one block per atom instead only one thread per atom).
- BpA method does not flush out data you need on other atoms as fast as the TpA method.

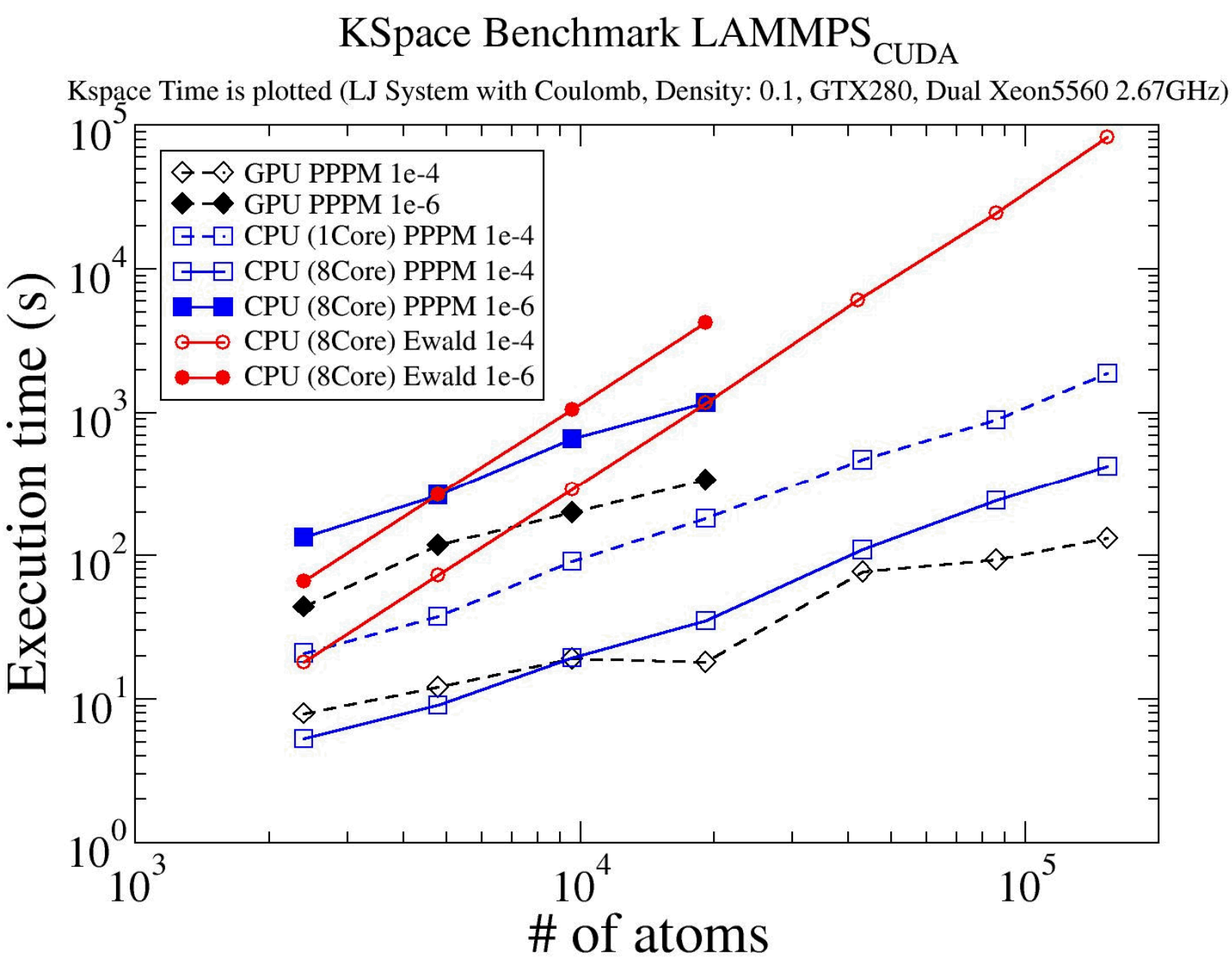
Sandia National Laboratories

Conclusions for GPU-LAMMPS project:

- Make both options available to GPU-LAMMPS users.
- Have LAMMPS pick the faster option via a short test run, but allow user to over-ride LAMMPS's choice.

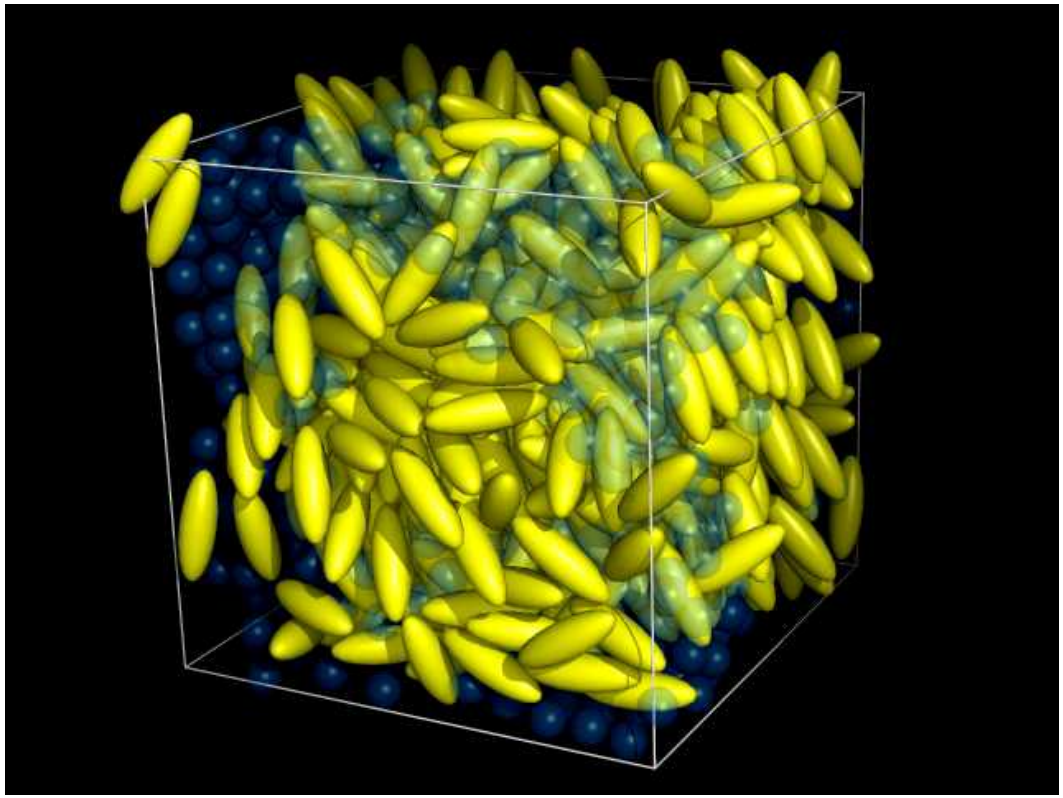
Broader conclusions:

- Writing fully-optimized many-core code will likely require a lot of additional work.
- Porting legacy code to run efficiently on exascale machines will be non-trivial.


KSpace Benchmark LAMMPS$_{CUDA}$
Kspace Time is plotted (LJ System with Coulomb, Density: 0.1, GTX280, Dual Xeon5560 2.67GHz)
GPU PPPM 1e-4
GPU PPPM 1e-6
CPU (1Core) PPPM 1e-4
CPU (8Core) PPPM 1e-4
CPU (8Core) PPPM 1e-6
CPU (8Core) Ewald 1e-4
CPU (8Core) Ewald 1e-6
Execution time (s)
# of atoms
10^0
10^1
10^2
10^3
10^4
10^5
10^3
10^4
10^5

Aspherical Particle Simulations

- Particles in nature and manufacturing often have highly irregular shapes
- Liquid crystal simulations
- Coarse Graining
- Majority of computational particle mechanics (CPM) simulators treat only spherical particles
- Need a parallel and scalable implementation to attack realistic problems (LAMMPS)



Sandia National Laboratories

Gay-Berne Potential

- **Single-site potential for asphericals**
- ***h* is the distance of closest approach**
- **S is the shape matrix**
- **The E matrix characterizes the relative well depths of side-to-side, face-to-face, and end-to end interactions**
- **~30 times the cost of an LJ interaction**

$$U = U_r\left(\mathbf{A}_1,\mathbf{A}_2,\mathbf{r}_{12}\right)\eta_{12}\left(\mathbf{A}_1,\mathbf{A}_2\right)\chi_{12}\left(\mathbf{A}_1,\mathbf{A}_2,\hat{r}_{12}\right)$$

$$U_r = 4\varepsilon\left[\left(\frac{\sigma}{h_{12}+\gamma\sigma}\right)^{12} - \left(\frac{\sigma}{h_{12}+\gamma\sigma}\right)^{6}\right]$$

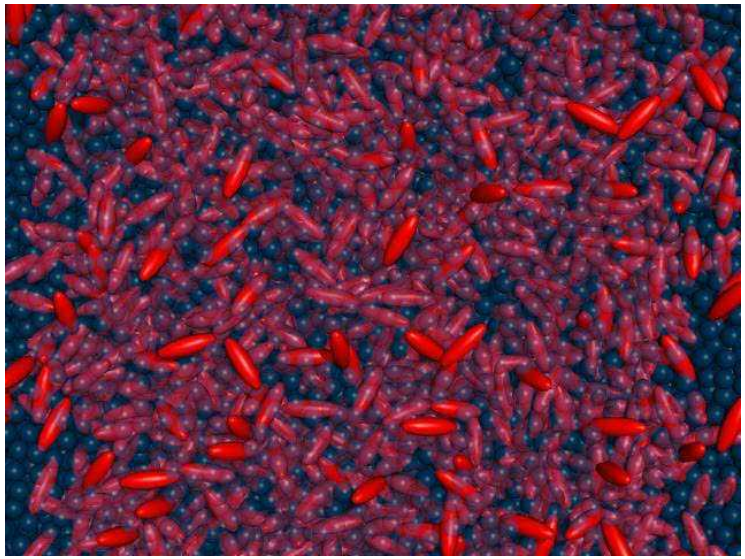
$$\eta_{12} = \left[\frac{2s_1 s_2}{\det\left[\mathbf{A}_1^T\mathbf{S}_1^2\mathbf{A}_1 + \mathbf{A}_2^T\mathbf{S}_2^2\mathbf{A}_2\right]}\right]^{\upsilon/2}$$

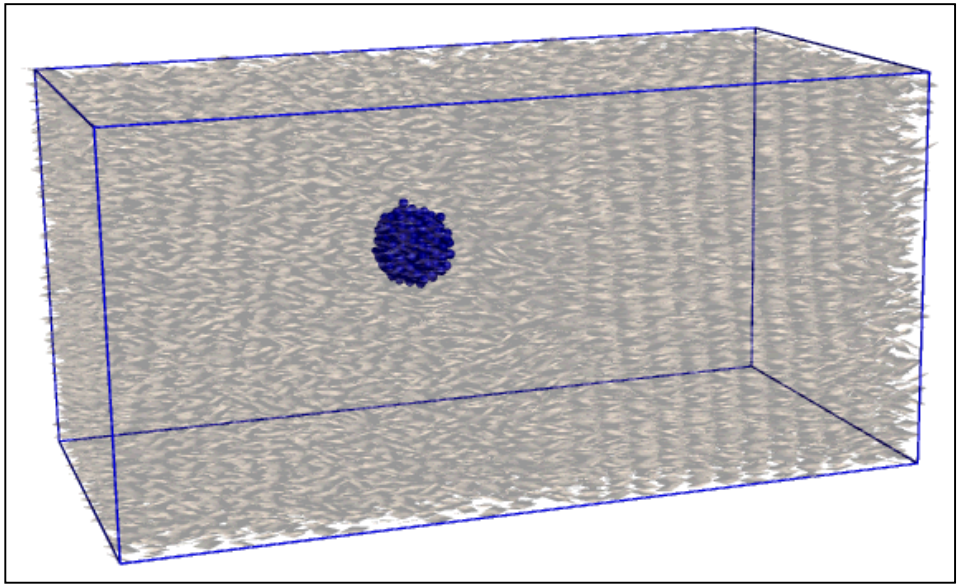
$$s = \left[a_i b_i + c_i c_i\right]\left[a_i b_i\right]^{1/2}$$

$$\chi_{12} = \left[2\hat{\mathbf{r}}_{12}^T\left(\mathbf{A}_1^T\mathbf{E}_1\mathbf{A}_1 + \mathbf{A}_2^T\mathbf{E}_2\mathbf{A}_2\right)^{-1}\hat{\mathbf{r}}_{12}\right]^{\mu}$$

Sandia National Laboratories

Liquid Crystal Simulations





Sandia National Laboratories

Accelerated Gay-Berne in LAMMPS

- Good candidate for GPU acceleration
  - *Very* expensive force calculation
- Available in the GPU package (make yes-asphere yes-gpu)
  - Can run on multiple GPUs on a single node or in a cluster
  - Multiple precision options: Single, Single/Double, and Double
  - Can simulate millions of particles per GPU

Sandia National Laboratories

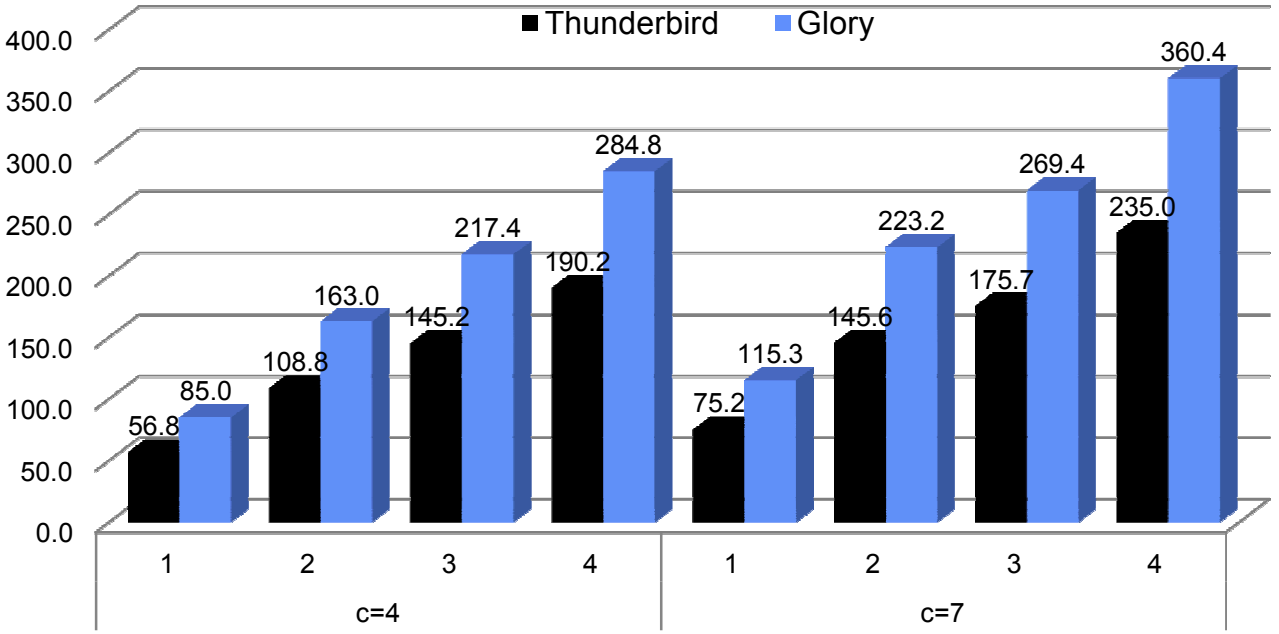
Algorithm

1. Copy atom positions and quaternions to device
2. Did reneighbor occur ? copy neighbor list to device
3. Call neighbor_pack kernel
   - 1 Atom per GPU Core
   - Perform cutoff check for all neighbors and store for coalesced access
   - *This limits thread divergence for the relatively expensive force computation*
4. Call force computation kernel
   - 1 Atom per GPU Core
     - Use full neighbor lists (double the amount of computations versus the CPU)
       - *No collisions with this approach*
   - Compute force, torque, energies, and virial terms
5. Copy forces, torques, energies, and virial terms to host

Sandia National Laboratories

GPGPU Times Speedup vs 1 Core

(*c*=cutoff, 32768 particles)

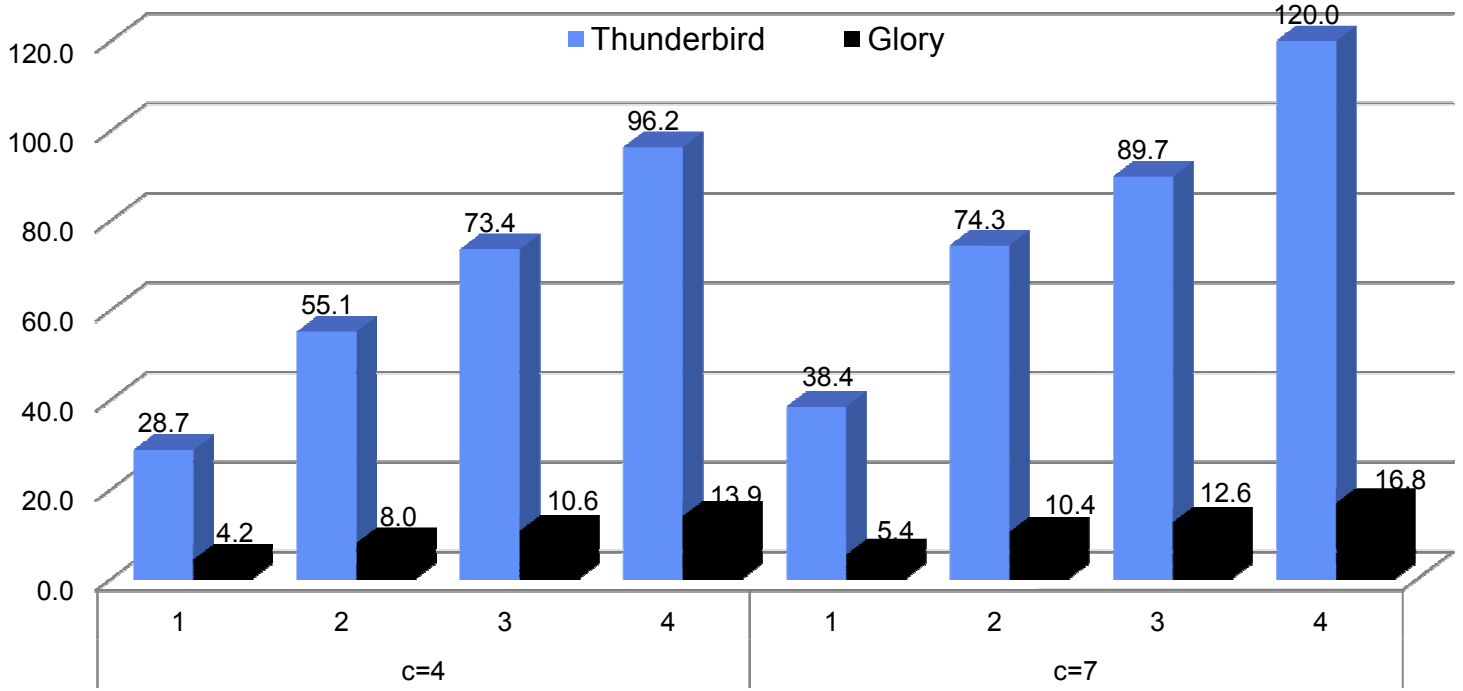


GPGPU: 1, 2, 3, or 4 NVIDIA, 240 core, 1.3 GHz Tesla C1060 GPU(s)
Thunderbird: *1 core of Dual 3.6 GHz Intel EM64T processors*
Glory: *1 core of Quad Socket/Quad Core 2.2 GHz AMD*

Sandia National Laboratories

GPGPU Times Speedup vs 1 Node

(*c*=cutoff, 32768 particles)

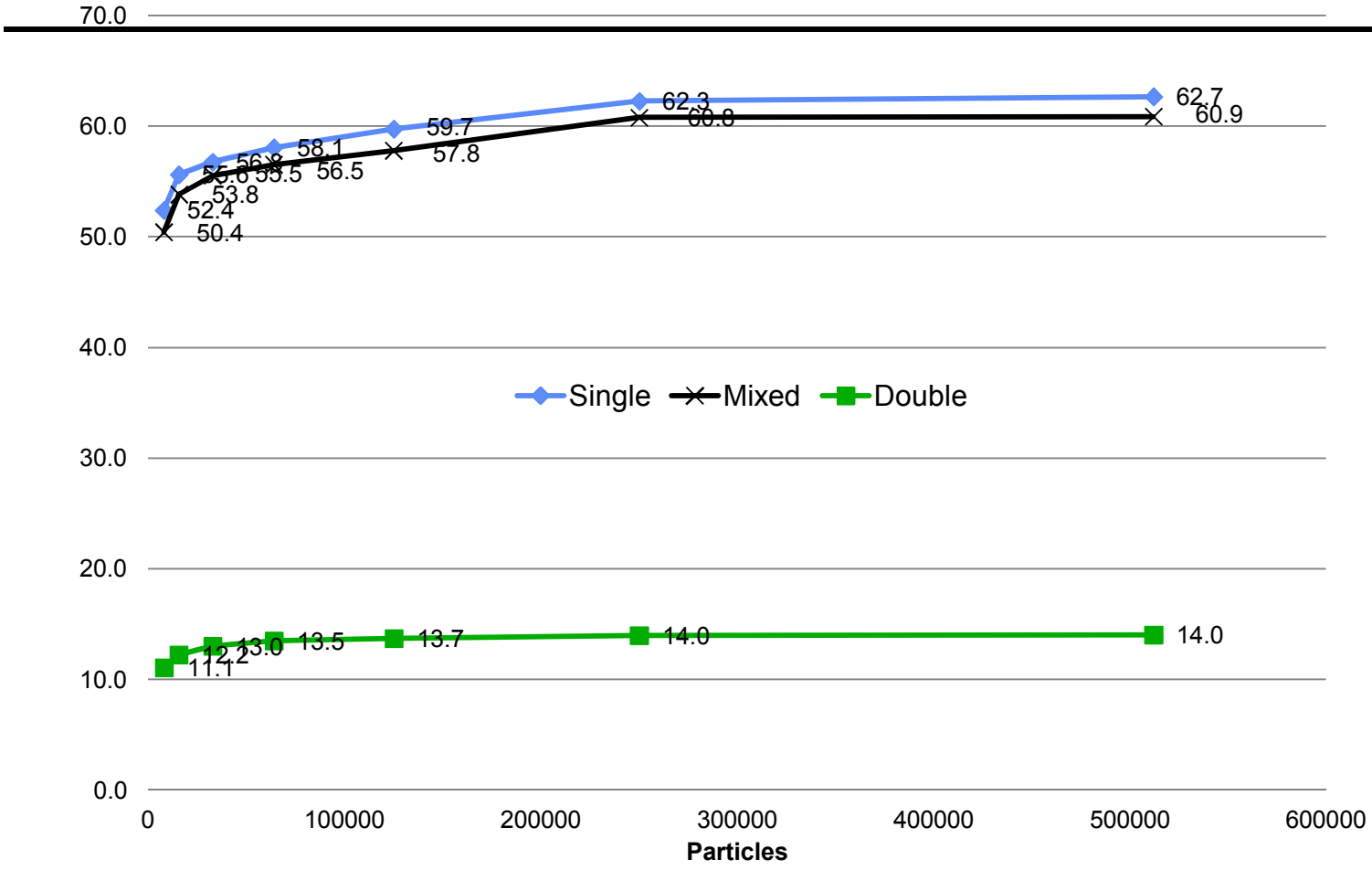


GPGPU: 1, 2, 3, or NVIDIA, 240 core, 1.3 GHz Tesla C1060 GPU(s)
Thunderbird: *2 procs, Dual 3.6 GHz Intel EM64T processors*
Glory: *16 procs, Quad Socket/Quad Core 2.2 GHz AMD*

Sandia National Laboratories

Times Speedup vs 1 Core

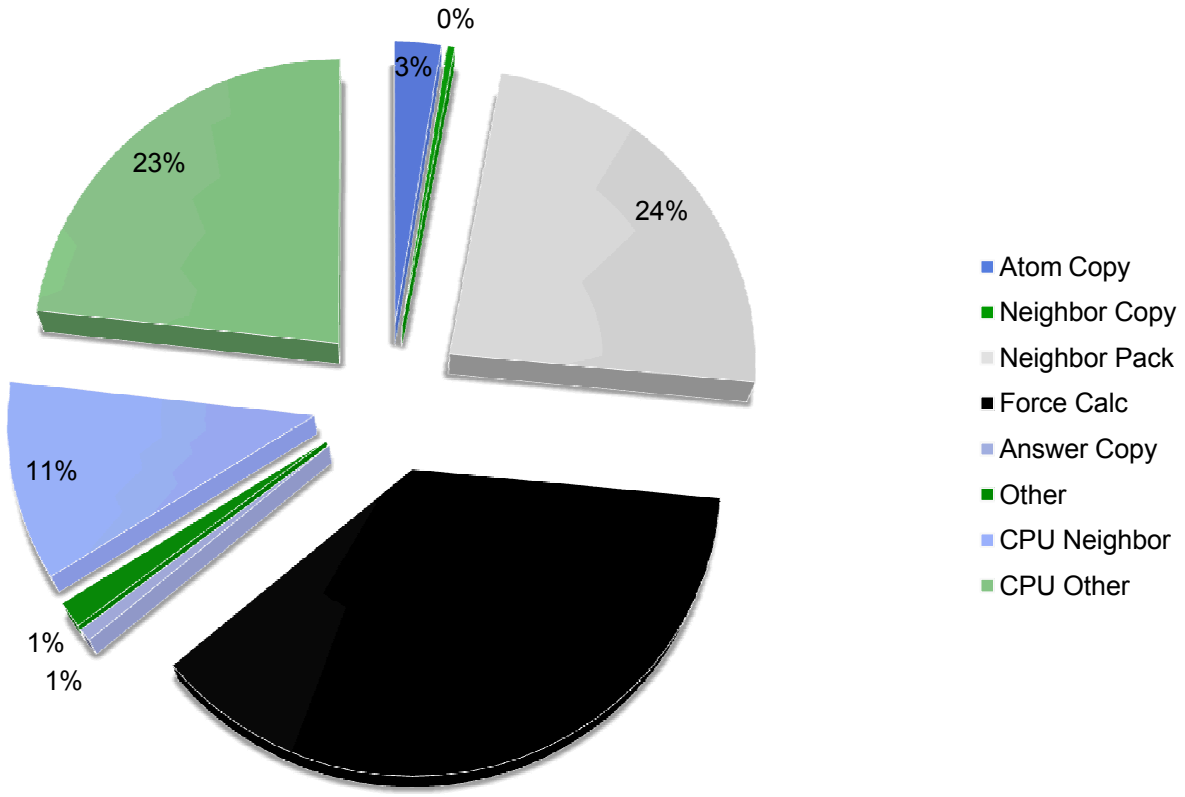
(*c*=4, 1 Tesla C1060, 3.6 GHz Intel EM64T processor)



Sandia National Laboratories

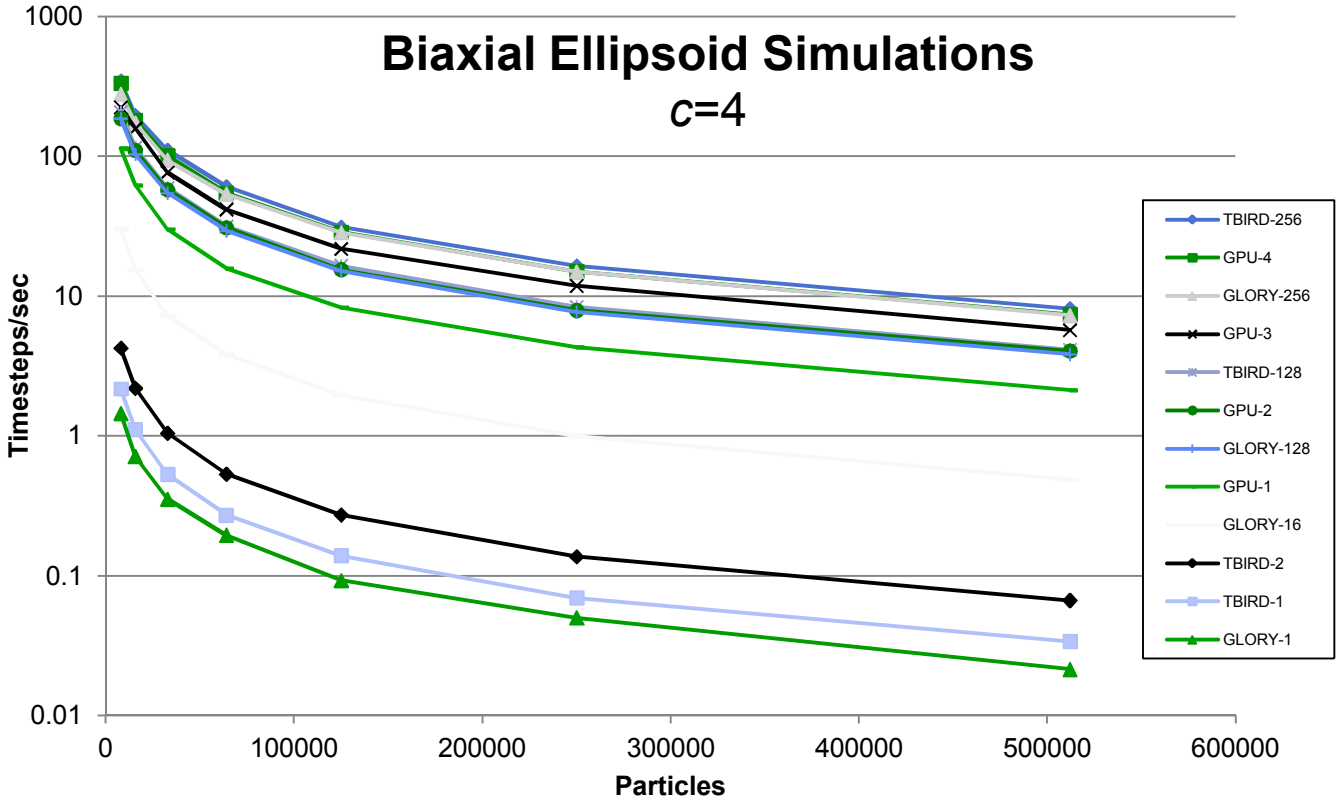
Simulation Time Breakdown

32K Particles, Cutoff=4, 1 Tesla C1060



Sandia National Laboratories

HPC Comparison



- **A single 4-GPU accelerated node can run a simulation faster than a 256-core simulation on Thunderbird or Glory.**
- **The power requirements for the GPU accelerated run were <1.2 kW versus 11.2 kW on Glory or 44.8 kW on Thunderbird**

Sandia National Laboratories

Coding Issues

- Difficult to keep force computation in registers
  - Had to manually scope variables to fit single precision in registers
  - Double precision goes to global memory
- Had to manually unroll Gaussian elimination loop
  - Compiler could not figure out array pointer arithmetic
- Complicated memory management can lead to separate implementations
  - e.g. what if a given simulation has atom type constants that do not fit in shared memory?
  - Many GPU implementations are “benchmark codes” meant for publication, not real use

Sandia National Laboratories

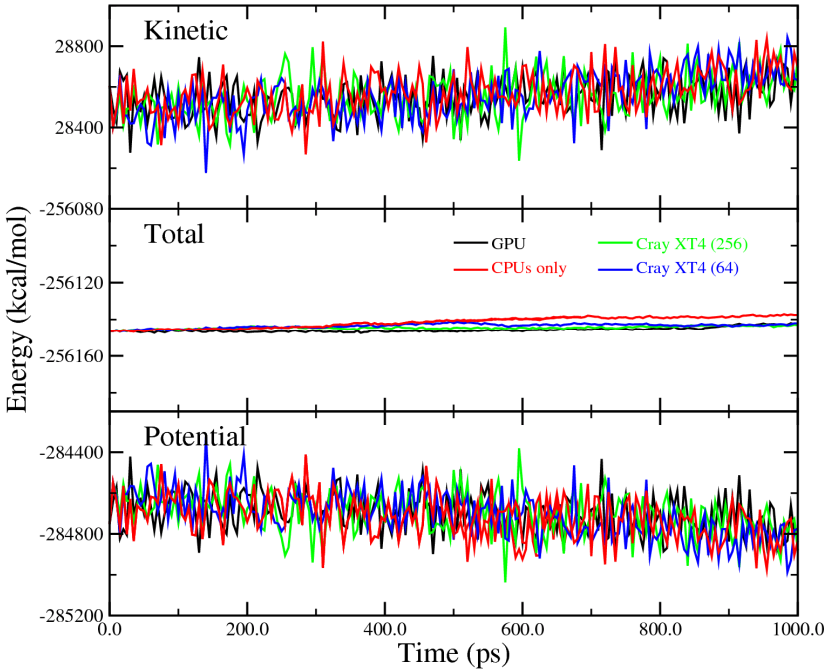
Alternative Algorithms

- Forces divided evenly among GPU cores (as opposed to per atom)
  - Need atomic operations to avoid collisions
  - No floating point atomic operations on current hardware
  - Slower for large simulations
  - *>20x speedup for a 128 particle simulation with Gay-Berne (<1 particle per core)*
- Neighbor list computation on the GPU
  - For Lennard-Jones, the simulation time is halved using a GPU cell list implementation
- Concurrent CPU execution
  - Multithreaded force decomposition (OpenMP)
  - Domain decomposition (separate MPI process for GPU and CPU computations)
  - Multiple threads/processes utilizing same GPU
  - For Gay-Berne, the upper-bound for concurrent execution performance gains is small
    - Overhead (full neighbor lists, thread creation, domain sizes)
  - For some potentials, concurrent execution may be needed in order to achieve good speedups

Sandia National Laboratories

Mixed Precision Results

LAMMPS *rhodo* benchmark (32,000 atom system, 1 million time-steps)



	Cray XT4	*Cray XT4*	*Linux cluster*	*GPU cluster*
CPU-cores	**64 (16 x 4)[b]**	**256 (64 x 4)**	**64 (4 x 16)**	**8 (8 x 1)**
GPUs				**8 (8 x 1)**
Compiler	**PGI (v 8.0-3)**	**PGI (v 8.0-3)**	**Gcc (4.1.1)**	**gcc (4.1.1) +CUDA (2.2)**
Precision	**Double**	**Double**	**Double**	**Mixed**

Sandia National Laboratories

OpenCL, CUDA-Driver, CUDA-Runtime?

- OpenCL offers a general API that is supported by many vendors and allows the potential to run kernels efficiently on the CPU in addition to coprocessor devices.
- CUDA Driver is a more mature GPGPU programming API with stable compilers, freedom in the choice of host compilers, and can potentially generate the most efficient code for Nvidia devices.
- CUDA Runtime offers a more succinct API and support for GPU code integrated with host code.
- Geryon – Software library that allows a single code to compile using any of the 3 APIs. Change namespace to change API.

Lessons learned

- Legacy codes can not be automatically “translated” to run efficiently on GPUs.
  - GPU memory management is tricky to optimize and strongly affects performance.
  - Host-device communication is a bottleneck.
  - Moving more of the calculation to the GPU improves performance, but requires more code conversion, verification, and maintenance.
  - Optimal algorithm on CPU is not necessarily the optimal algorithm on GPU or on a hybrid cluster (i.e. neighbor lists, long-range electrostatics).
  - Mixed- or single-precision is OK in some cases, and considerably faster. Preferable to give users the choice of precision.
  - Optimal performance requires simultaneous use of available CPUs, GPUs, and communication resources.

Remaining challenges

1. Funding limitations
2. Merging of disparate efforts and “visions”
3. Better coverage: port more code (FFs, other) to GPU
4. Better performance (better algorithms & code)
5. Quality control (code correctness and readability)
6. Code maintenance
    a. Synching with standard LAMMPS
    b. User support
    c. Bug fixes

Sandia National Laboratories