Porting LAMMPS to GPUs

W. Michael Brown, Scott Hampton, Pratul Agarwal, Peng Wang, Paul Crozier, Steve Plimpton

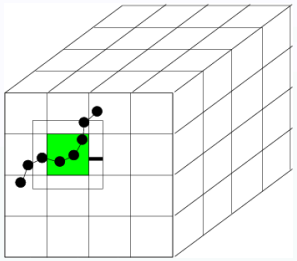
Tuesday, March 9, 2010

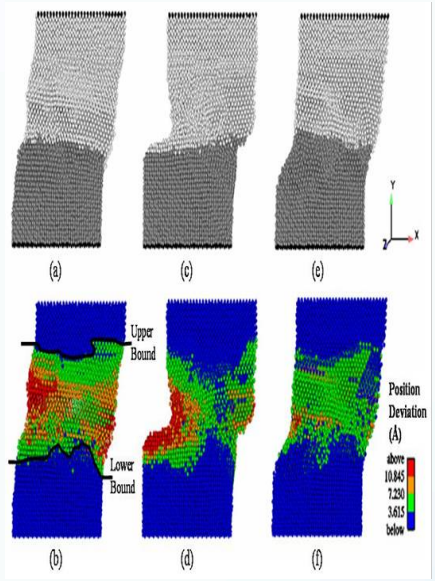
LAMMPS

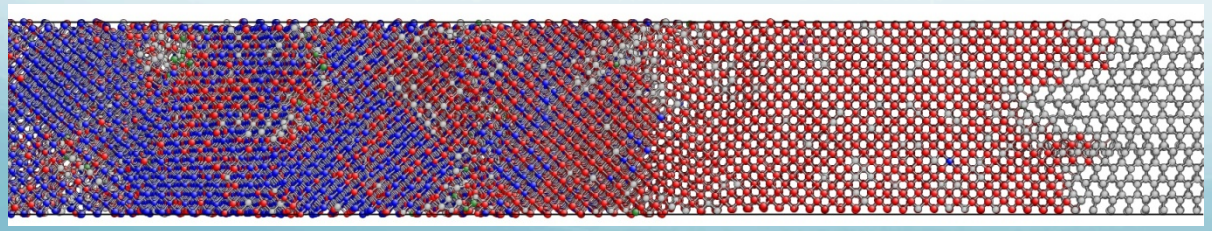
(Large-scale Atomic/Molecular Massively Parallel Simulator)

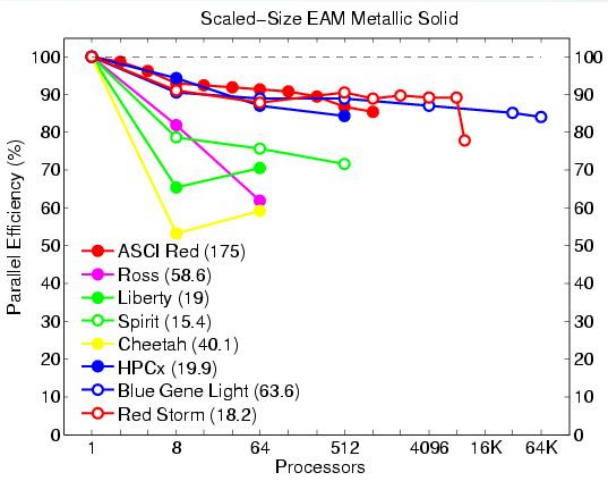
http://lammps.sandia.gov

- **Classical MD code.**
  - **Particles interact with neighbors within some cutoff**
  - **Gradient of the potential energy surface gives forces**
  - **Simulate by integrating equations of motion at timestep**
- 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.







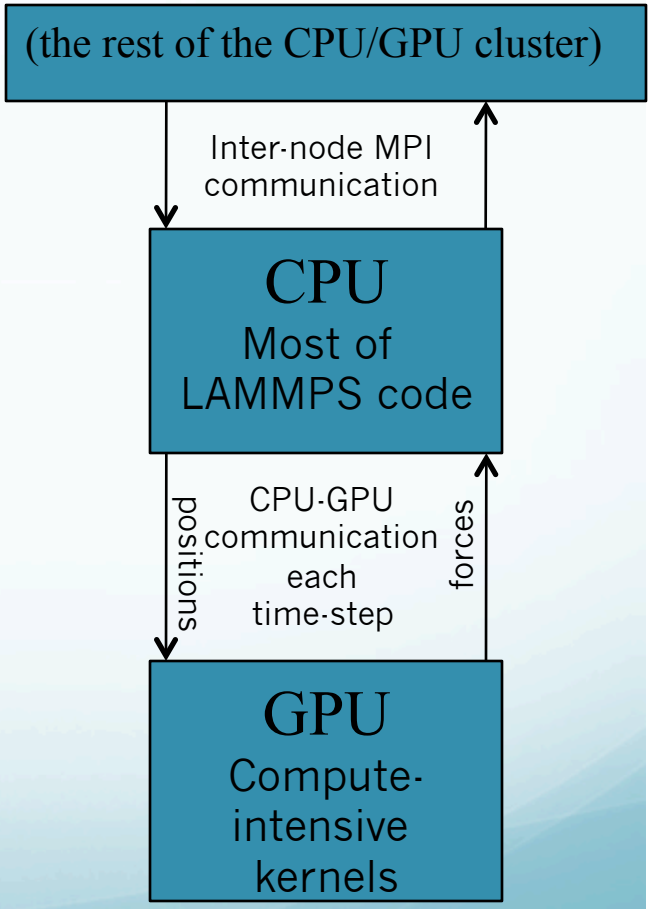


Extending LAMMPS via Styles

- In hindsight, this is best feature of LAMMPS
  - 80% of code is "extensions" via styles
  - only 35K of 175K lines is core of LAMMPS
- Easy for us and others 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

GPU-LAMMPS strategy

- Enable LAMMPS to run efficiently on future CPU-based clusters that have GPU accelerators.
- Not aiming for running on a single GPU.
- Not aiming to rewrite all of LAMMPS in CUDA.
- Rewrite the most compute-intensive LAMMPS kernels in CUDA.
- At each time-step, ship particle positions from CPU to GPU, compute forces on the GPU, and then ship forces back to the CPU.
- Domain decomposition across processes; force decomposition on GPU

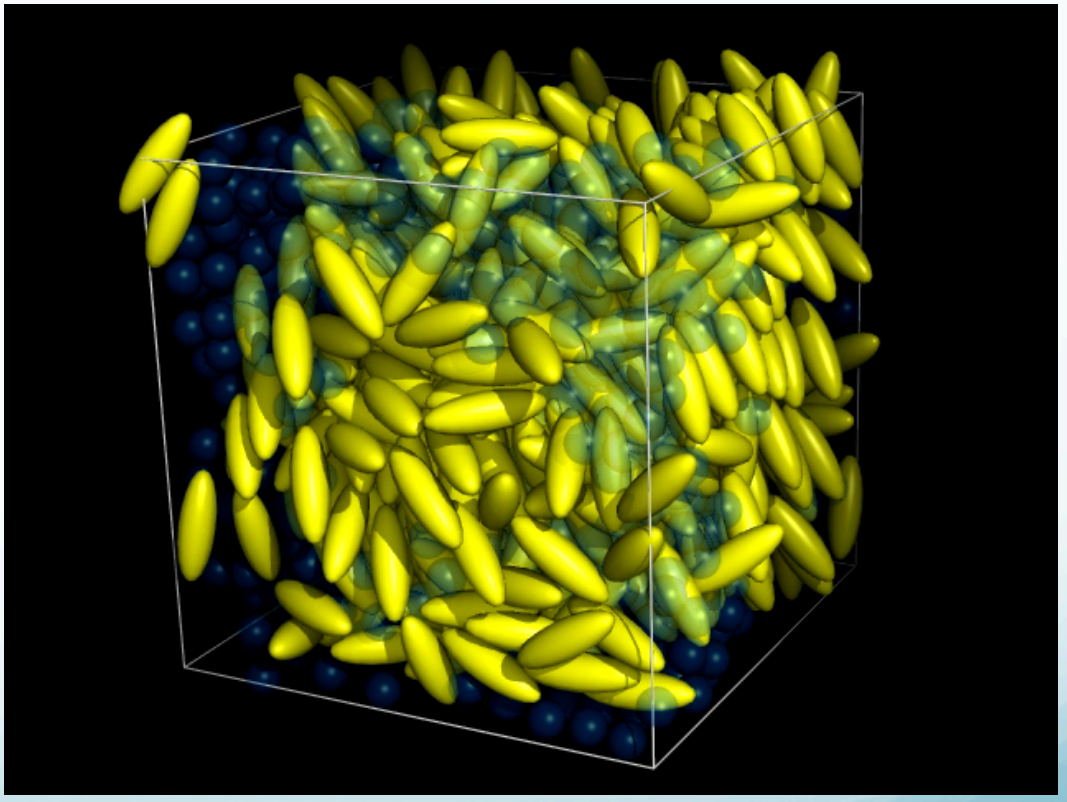


Example

Aspherical Particle Simulation

Why Aspherical Particles?

- 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)



Gay-Berne Potential

- Single-site potential for biaxial ellipsoids
- *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

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

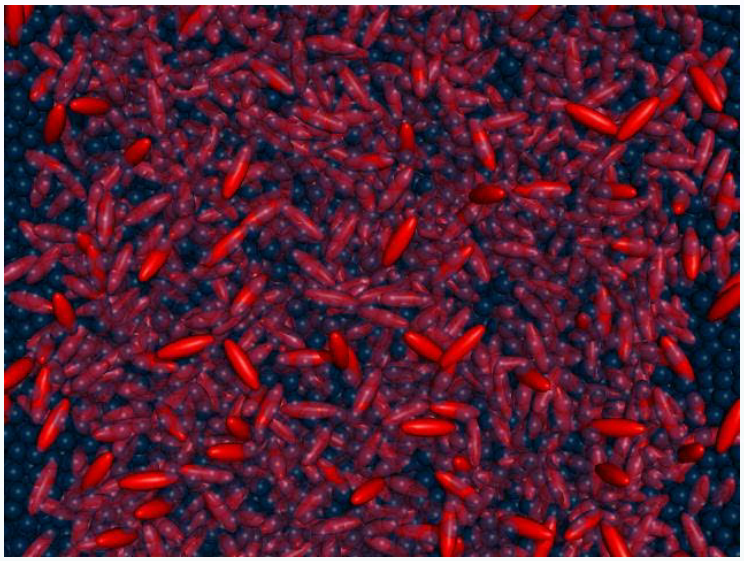
$$U_r = 4\varepsilon\left[\left(\frac{\sigma}{h_{12}+\gamma\sigma}\right)^{12} - \left(\frac{\sigma}{h_{12}+\gamma\sigma}\right)^{6}\right]$$

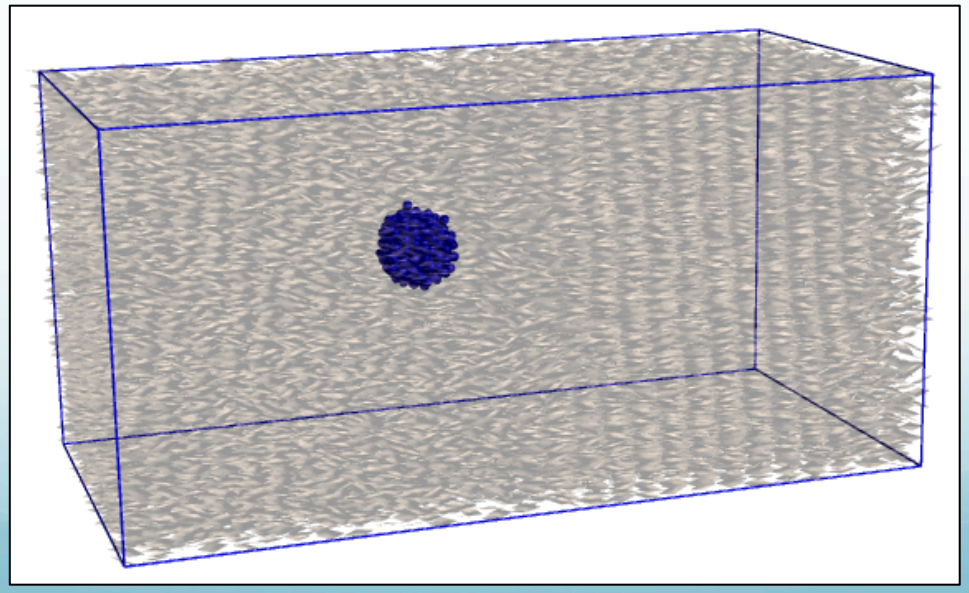
$$\eta_{12} = \left[\frac{2s_1s_2}{\det\left[\mathbf{A}_1^T\mathbf{S}_1^2\mathbf{A}_1 + \mathbf{A}_2^T\mathbf{S}_2^2\mathbf{A}_2\right]}\right]^{\upsilon/2}$$

$$s = \left[a_ib_i + c_ic_i\right]\left[a_ib_i\right]^{1/2}$$

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

Liquid Crystal Simulations





Accelerated Gay-Berne in LAMMPS

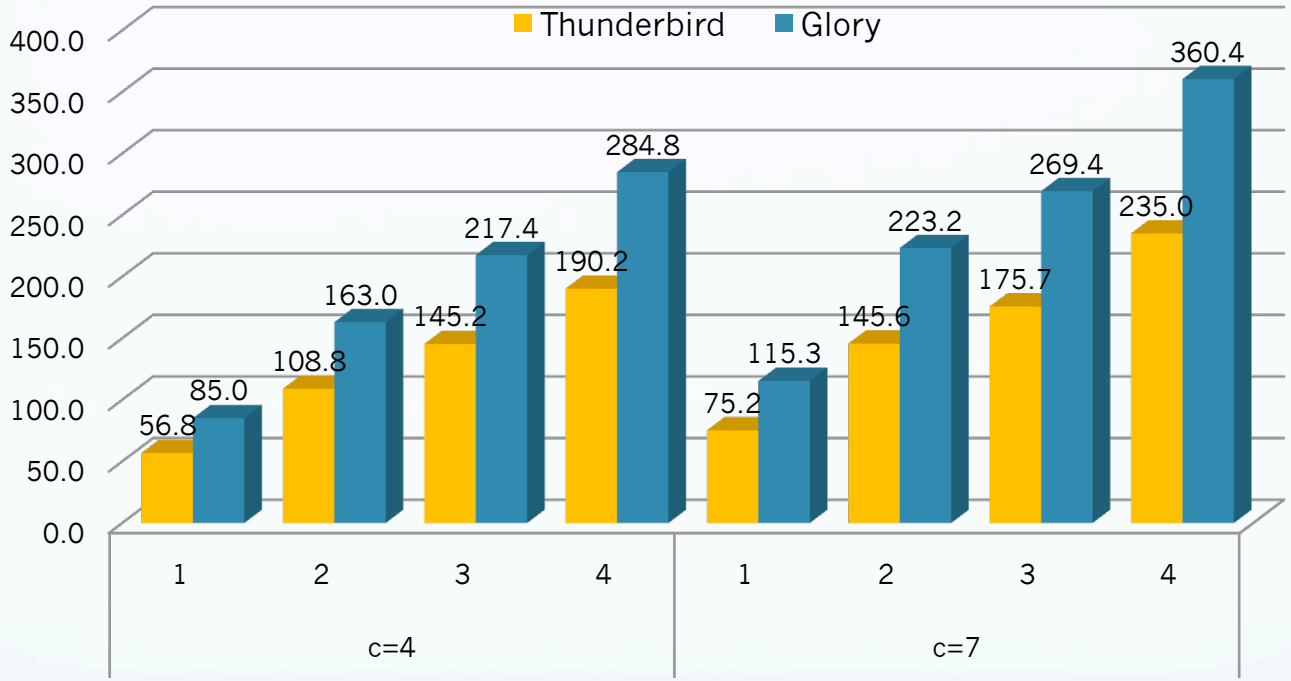
- Good candidate for GPU acceleration
  - *Very* expensive force calculation
- Available in the GPU package (make yes-asphere yes-gpu)
  - Can run on multiple GPUs on a single node or in a cluster
  - Multiple precision options: Single, Single/Double, and Double
  - Can simulate millions of particles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

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

GPGPU Times Speedup vs 1 Core

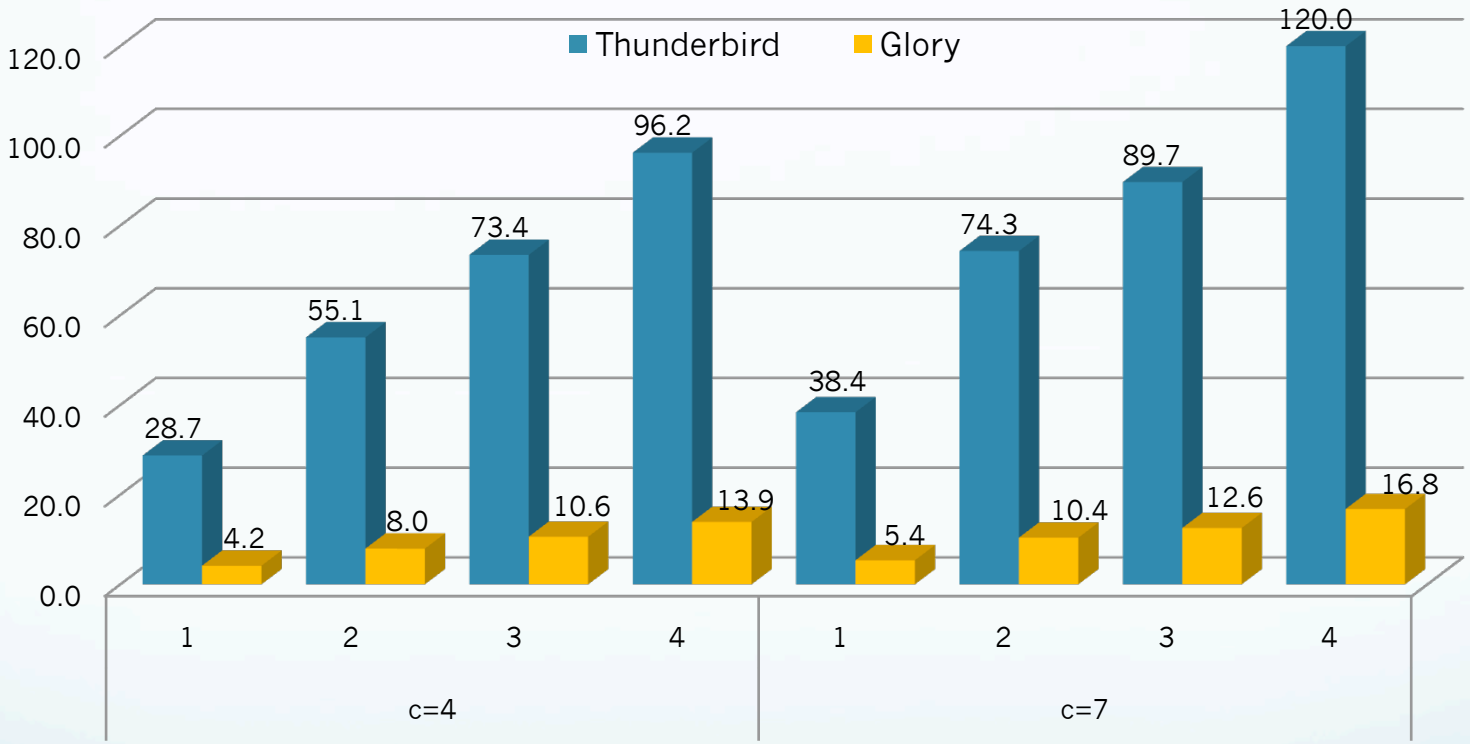
(c=cutoff, 32768 particles)



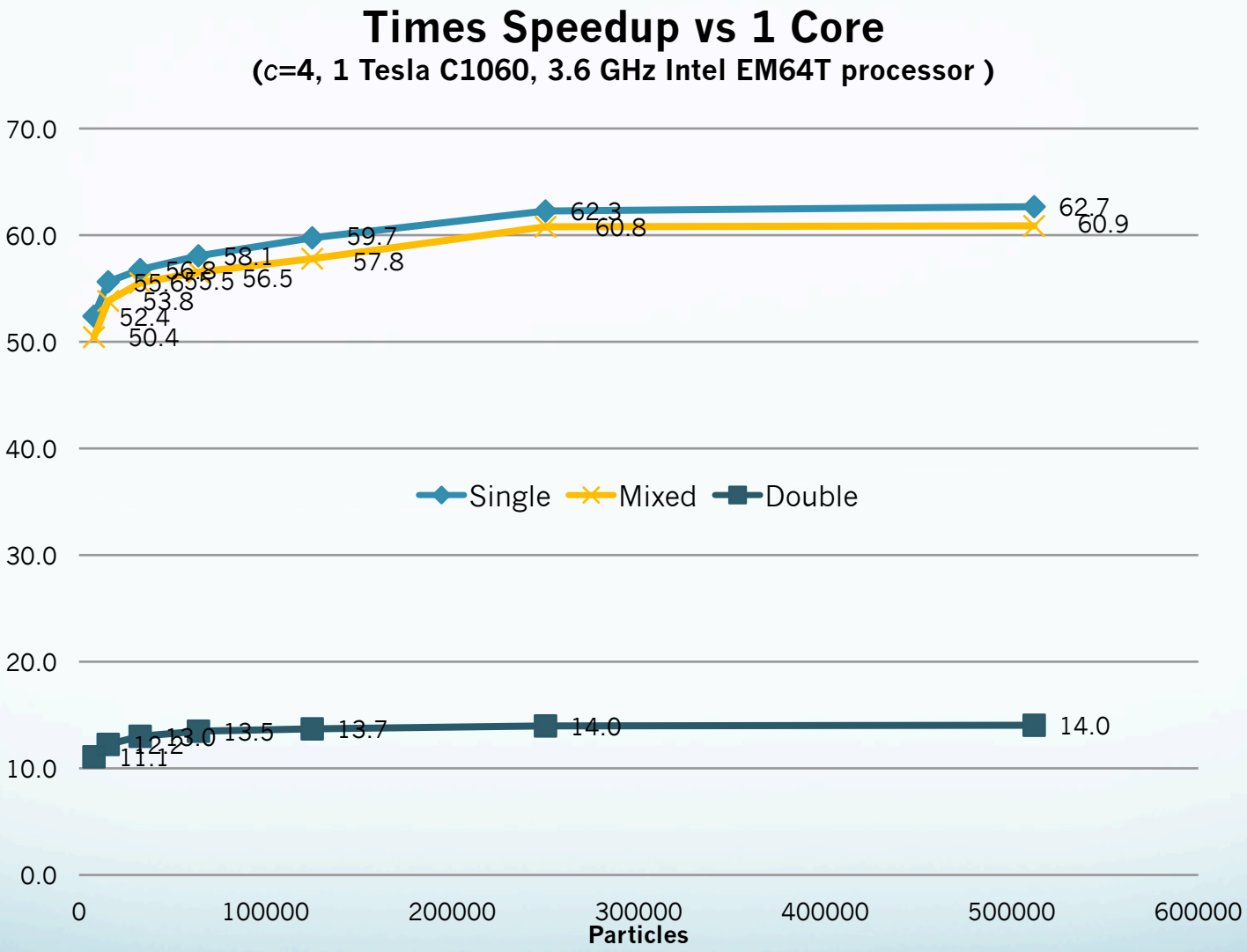
GPGPU: 1, 2, 3, or 4 NVIDIA, 240 core, 1.3 GHz Tesla C1060 GPU(s)
Thunderbird: *1 core of Dual 3.6 GHz Intel EM64T processors*
Glory: *1 core of Quad Socket/Quad Core 2.2 GHz AMD*

GPGPU Times Speedup vs 1 Node

(c=cutoff, 32768 particles)

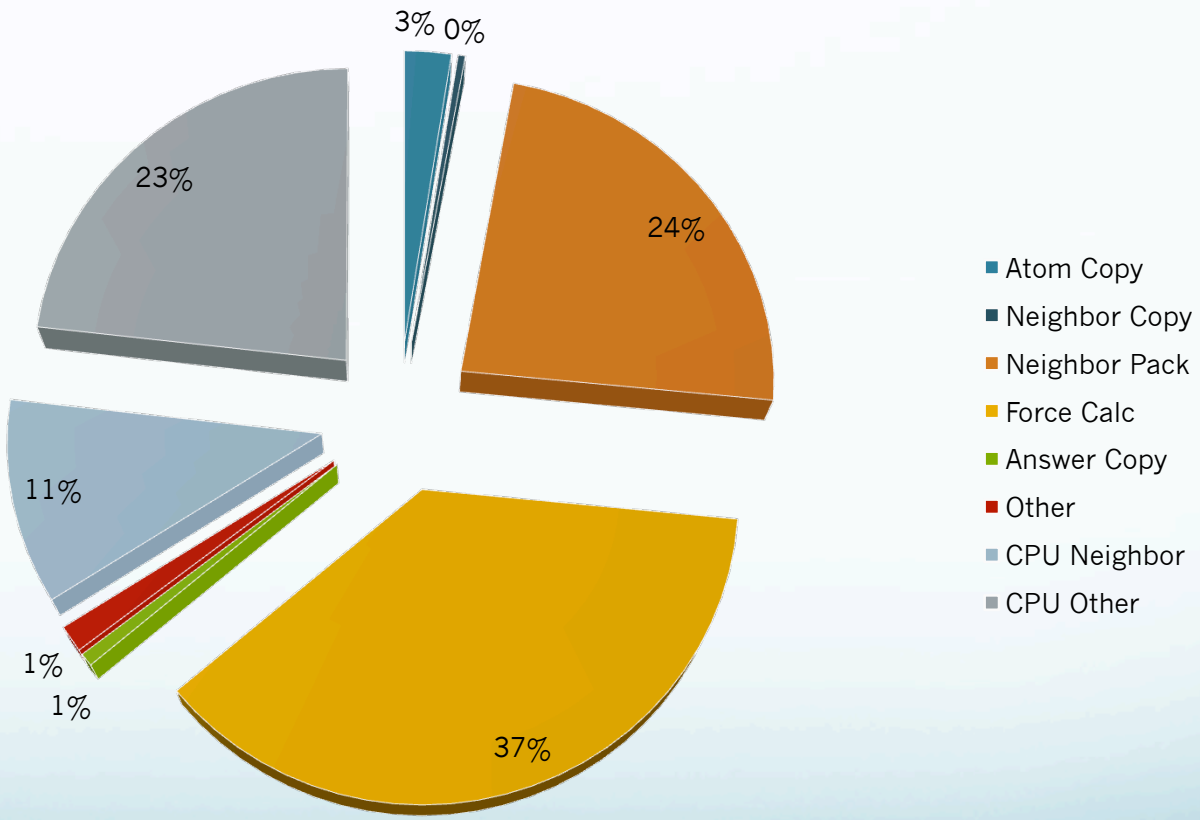


GPGPU: 1, 2, 3, or NVIDIA, 240 core, 1.3 GHz Tesla C1060 GPU(s)
Thunderbird: *2 procs, Dual 3.6 GHz Intel EM64T processors*
Glory: *16 procs, Quad Socket/Quad Core 2.2 GHz AMD*

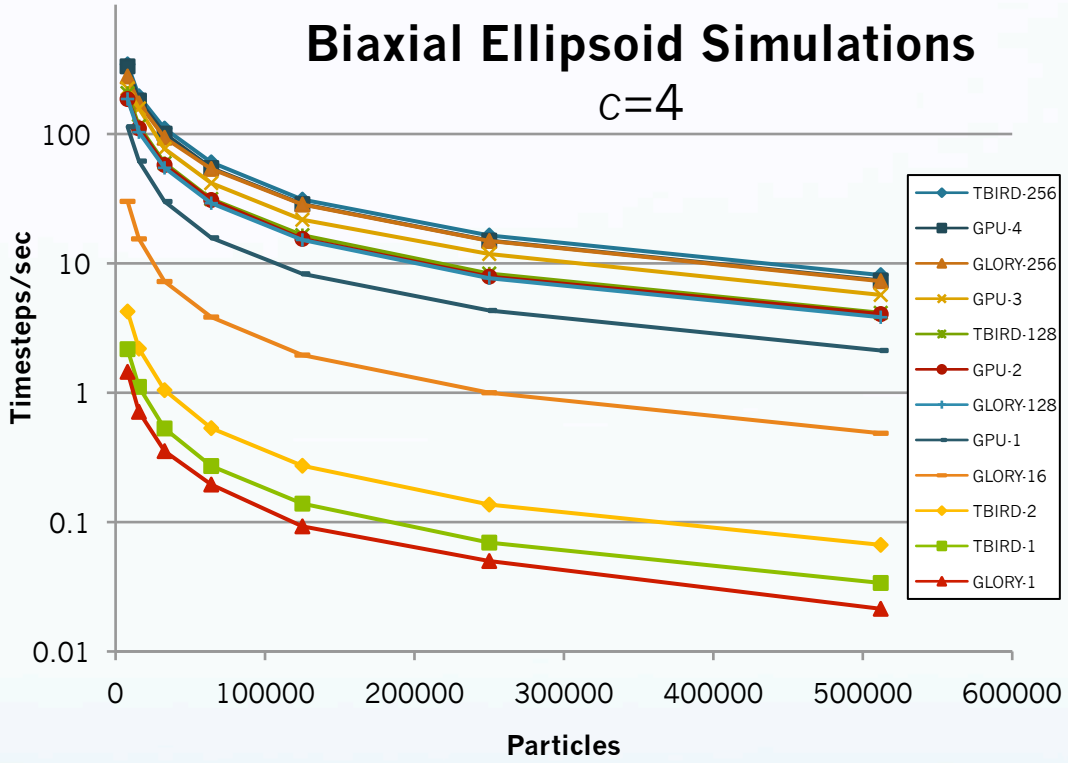

Times Speedup vs 1 Core
(c=4, 1 Tesla C1060, 3.6 GHz Intel EM64T processor)
70.0
60.0
50.0
40.0
30.0
20.0
10.0
0.0
0
100000
200000
300000
400000
500000
600000
Particles
Single
Mixed
Double
52.4
53.8
55.6
55.5
56.5
57.8
59.7
60.8
62.3
62.7
60.9
50.4
58.1
11.1
12.2
13.0
13.5
13.7
14.0
14.0

Simulation Time Breakdown

32K Particles, Cutoff=4, 1 Tesla C1060



HPC Comparison



- A single 4-GPU accelerated node can run a simulation in the same amount of time as 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

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

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 particl 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

Future Work

- Currently available in the main LAMMPS distribution
  - Lennard-Jones and Gay-Berne
- Adding more potentials
  - 3-body, MEAM, etc.
- Long range electrostatics

Porting LAMMPS to GPUs

- Contact Paul Crozier (pscrozi@sandia.gov)
- Still largely a research effort

Marc Adams (Nvidia)
Pratul Agarwal (ORNL)
Sarah Anderson (Cray)
Mike Brown (Sandia)
Paul Crozier (Sandia)
Massimiliano Fatica (Nvidia)
Scott Hampton (ORNL)
Ricky Kendall (ORNL)
Hyesoon Kim (Ga Tech)
Axel Kohlmeyer (Temple)
Doug Kothe (ORNL)
Scott LeGrand (Nvidia)
Ben Levine (Temple)
Christian Mueller (UTI Germany)
Steve Plimpton (Sandia)
Duncan Poole (Nvidia)
Steve Poole (ORNL)
Jason Sanchez (RPI)
Arnold Tharrington (ORNL)
John Turner (ORNL)
Peng Wang (Nvidia)
Lars Winterfeld (UTI Germany)
Andrew Zonenberg (RPI)

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.**

http://www.cs.sandia.gov/~wmbrown/geryon

Questions