SAND2017-9006C

SPARTA Kokkos: A Massively Parallel and Multithreaded DSMC Code

Stan Moore, Dan Ibanez

2017 DSMC Workshop

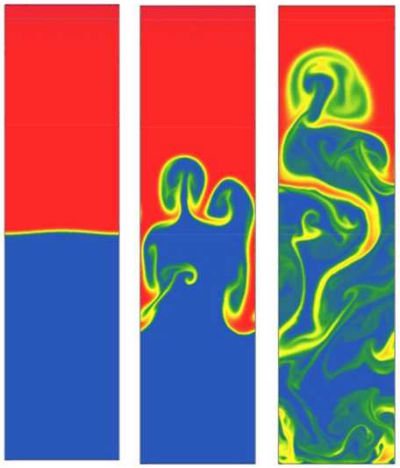
Santa Fe, NM

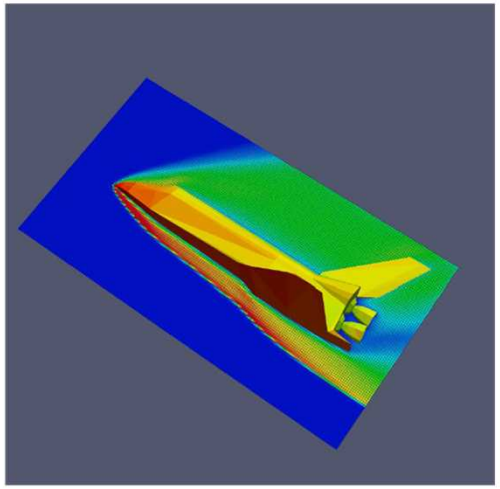
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. SAND NO. 2011-XXXXP

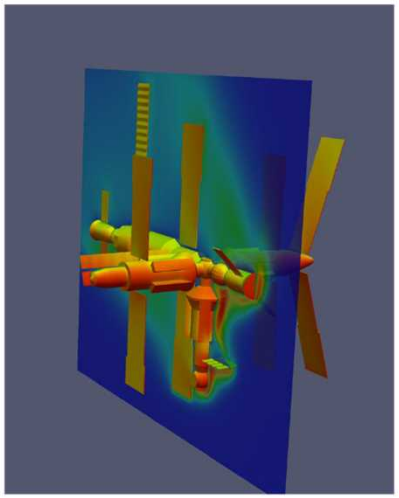
SPARTA

(**S**tochastic **PA**rallel **R**arefied-gas **T**ime-accurate **A**nalyzer)

- Direct Simulation Monte Carlo (DSMC) code
- Core developers are Steve Plimpton and Michael Gallis (Sandia National Labs)
- Open-source, http://sparta.sandia.gov

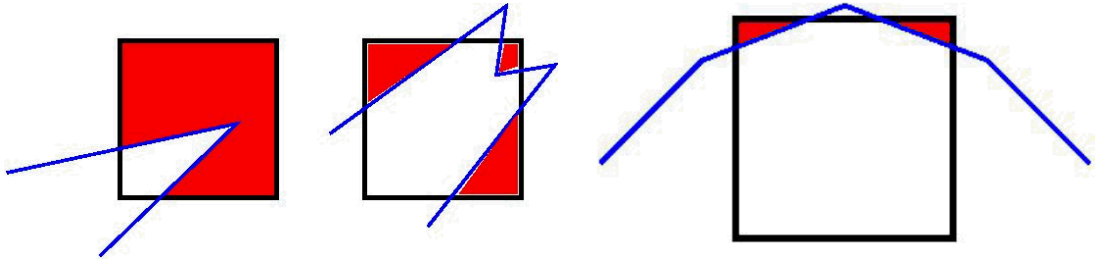


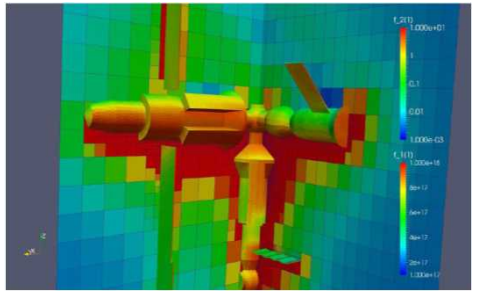




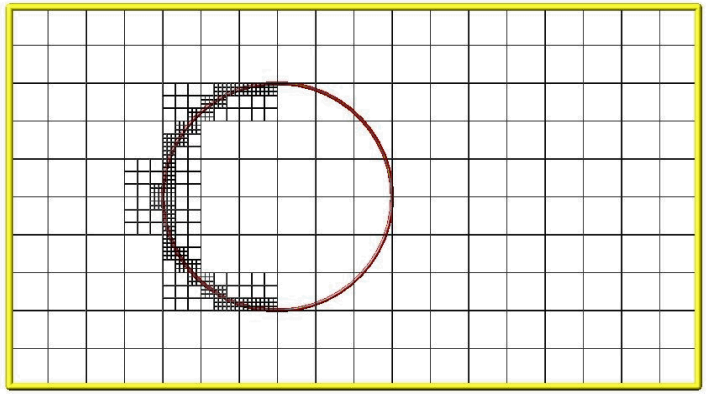
SPARTA Features

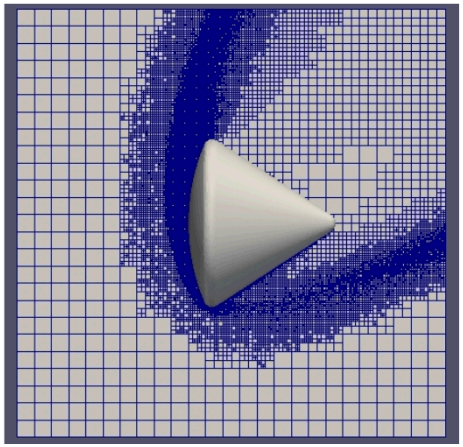
- Structured grids with complex surfaces via cut and split cells





- Hierarchal grids with adaptive mesh refinement

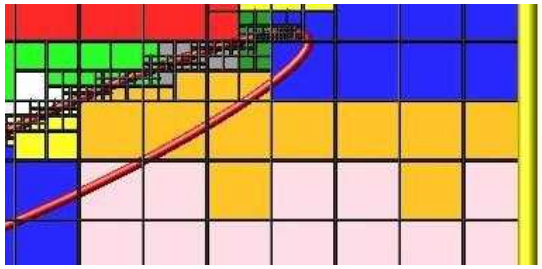


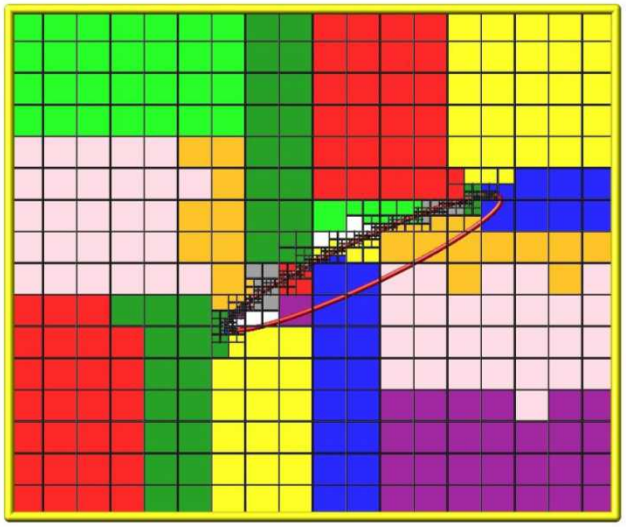


- MPI parallelism using highly scalable domain decomposition (trillion particles simulated using the Sequoia supercomputer)

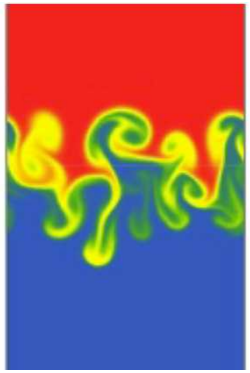
SPARTA Features (cont.)

- Load balancing (static and dynamic)





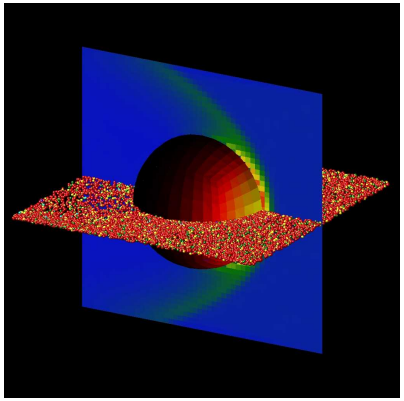
- Gas-phase collisions and chemistry

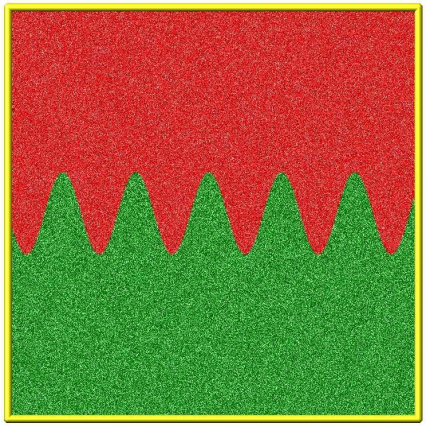


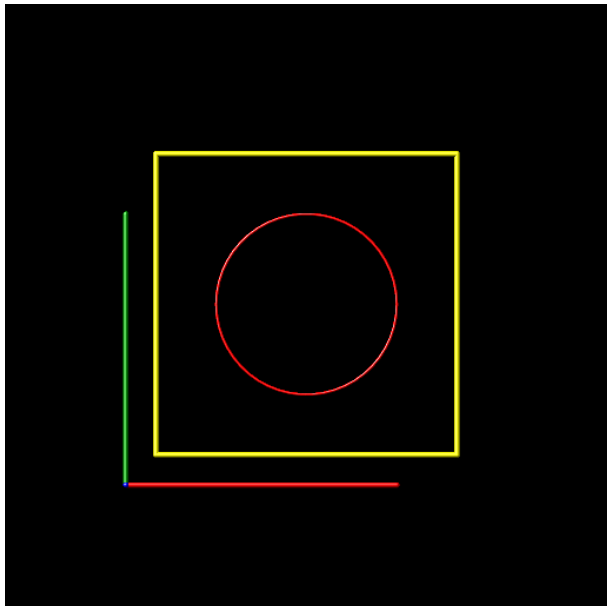
- Surface collisions and chemistry
- Grid cell weighting of particles

SPARTA Features (cont.)

- Diagnostics
  - global boundary statistics
  - per grid cell statistics
  - per surface element statistics
  - time-averaging of global, grid, surface statistics
- In-Situ Visualization







- And more

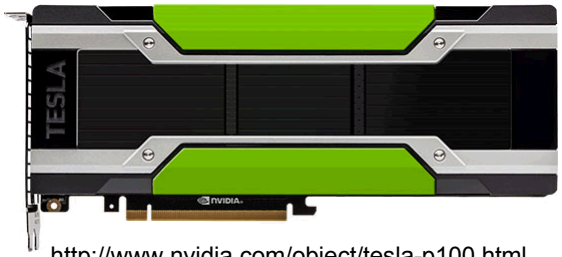
Current Trends in HPC Hardware

Sandia National Laboratories

- Nodes are getting wider (more CPU cores/node)
  - Intel Sandy Bridge: 16 cores/node
  - Intel Haswell: 32 cores/node
  - Intel Broadwell: 36 cores/node
  - Intel Sky Lake: 48 cores/node
  - Intel Knight's Landing (KNL): 68 cores/node
- CPU vector processor width is increasing: KNL and Sky Lake can process 8 double-precision operations at a time
- Multiple hardware (hw) threads per core (i.e. hyperthreading)
  - Intel KNL: 4 hw threads/core
  - IBM Power8: 8 hw threads/core

GPU Acceleration

- Currently, two out of the top ten supercomputers use NVIDIA GPUs, according to the June 2017 Top500 List (https://www.top500.org)
- In the future, another two large supercomputers (Summit at ORNL and Sierra at LLNL) will also have GPUs
- GPUs have 100s-1000s of "cores" (but frequency of GPU core is lower than CPU core)
- Special code (e.g. CUDA) required to run on GPUs
- Pointers in CPU memory are NOT accessible on the GPU, and visa versa, unless using unified virtual memory (UVM)



http://www.nvidia.com/object/tesla-p100.html

Parallelization Approach

Sandia National Laboratories

- MPI Domain decomposition
  - Each processor owns a portion of the simulation domain and particles therein
  - Halo of “ghost” grid cells to reduce communication cost
- MPI used for node to node communication
- Can use either MPI or threading between processors inside a CPU node (shared memory)
- Thread over significant loops of work inside of MPI domains, i.e. loops over grid cells, particles, etc.
- For GPUs, a host CPU launches kernels of work on the GPU

MPI vs OpenMP: The Reality

Sandia National Laboratories

- Using threading requires code to be thread safe, which can lead to additional overhead vs MPI only
  - Order of execution of a threaded loop is not guaranteed and can change from run to run
  - No two threads can write to the same memory at the same time
  - May need to rewrite algorithm to avoid “race” conditions or use thread atomics (memory locks)
- Benefits of MPI + OpenMP vs MPI only may only be visible at large scale (i.e. thousands of nodes, where MPI communication time or MPI memory consumption is high)
- MPI only doesn’t work on GPUs, must use threading

Kokkos Library

Sandia National Laboratories

- Modern HPC hardware is complicated
- The Kokkos library is an abstraction layer between the programmer and these platforms
- Core developers of Kokkos are Carter Edwards, Christian Trott, and Daniel Sunderland (Sandia National Laboratories)
- Kokkos consists of two main parts:
  1. Parallel dispatch—threaded kernels are launched and mapped onto backend languages such as CUDA, OpenMP, or Pthreads
  2. Kokkos views—multidimensional arrays with polymorphic memory layouts that can be optimized for a specific hardware (such as C-style layout right vs Fortran-style layout left)
- Used on top of existing MPI parallelization (MPI + X)
- Open-source, can be downloaded from https://github.com/kokkos/kokkos

Kokkos (cont.)

Sandia National Laboratories

- C++ code is written once using Kokkos abstractions in a form that looks independent of the target hardware
- Target-specific code (i.e. OpenMP or CUDA) is generated based on compile-time options
- Helps developers deal with a wide variety of ever-changing hardware (for new hardware, change Kokkos not SPARTA)
- Protects developers from having to maintain multiple versions of the code
- In practice, Kokkos allows SPARTA to run on GPUs and use OpenMP threading on multicore CPUs/Xeon Phis
- Goal is performance portability: good performance across many different platforms

SPARTA KOKKOS Package

Sandia National Laboratories

- Implemented as an optional add-on package to SPARTA
- Developed by Stan Moore, Dan Ibanez, and Tim Fuller (Sandia National Laboratories)

<u>Algorithms ported to Kokkos:</u>

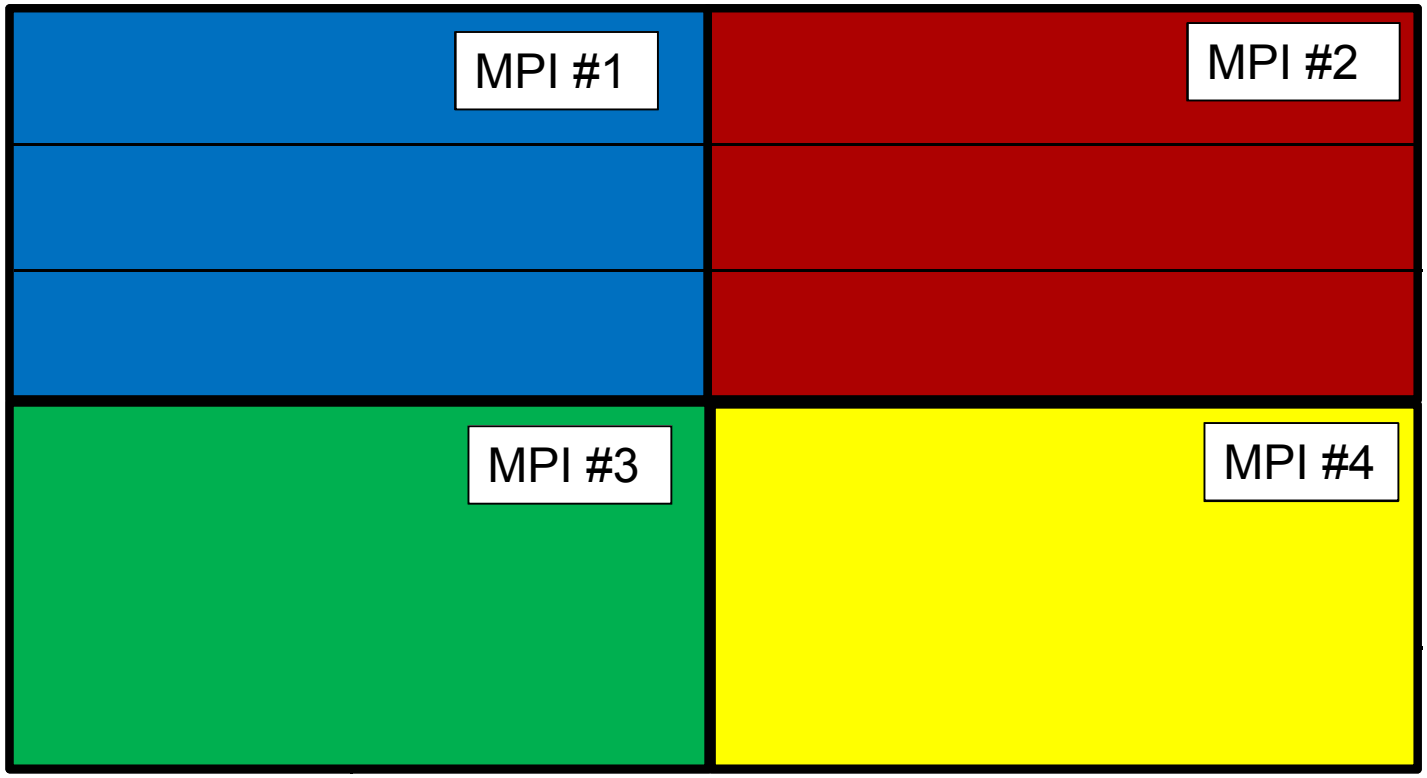
- Particle move, sort, and collide (reactions not yet supported)
- Communication: pack/unpack buffers, compress particle list
- Diagnostics
  - Temperature computation
  - Averaging of grid quantities
- Particle emission from cell faces
- Surface collisions (diffuse, specular, and vanish)

Incremental Approach

- Many diagnostics haven't yet been ported to Kokkos, how can they still work with Kokkos?
  - Certain Kokkos arrays are aliased with legacy data structures (must use C-style layout right)
  - Automatic data movement between host CPU and device GPU for non-Kokkos parts of the code such as diagnostics (not using UVM)
- Initialization done on host CPU without threading (assumes run time is large compared to initialization time)
- Allows incremental porting approach

MPI Parallelization Approach

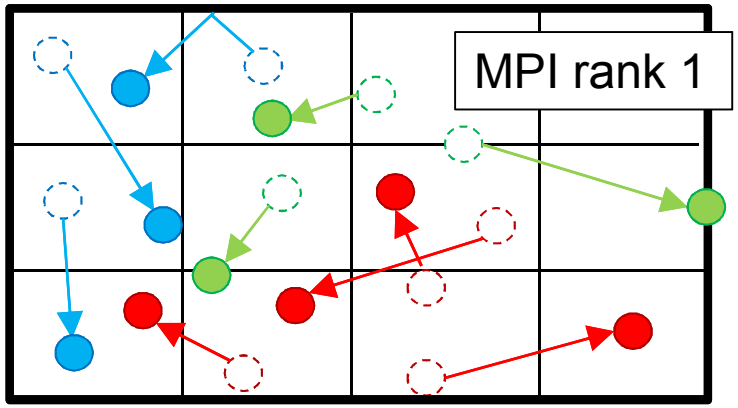
- Domain decomposition: each processor owns a portion of the simulation domain and particles therein



Threaded Move

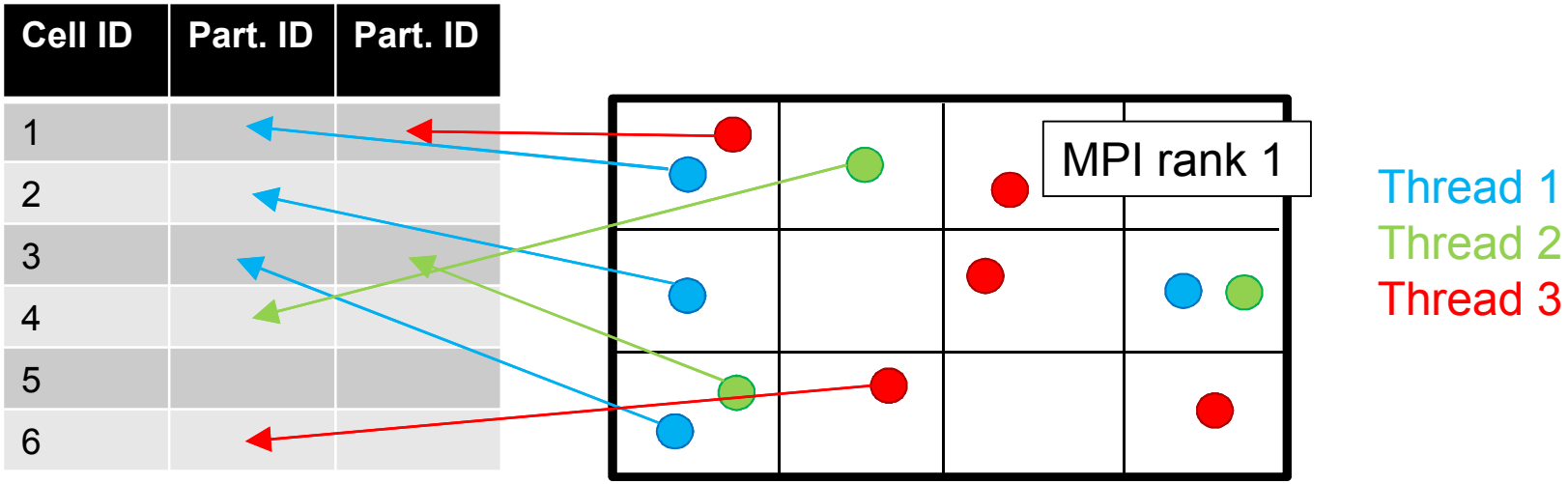
- One thread pushes particles for a timestep or micro-iteration
- Intermediate grid crossings are found
- Statistical accumulators (i.e. number of moves, number of surface collisions, etc.) use either a parallel reduction or an atomic reduction on a global variable

Thread 1
Thread 2
Thread 3



Threaded Sort

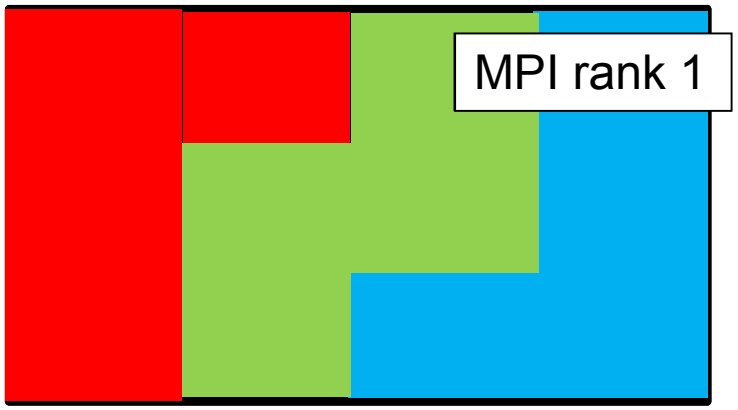
- Threads loop over particles to sort by grid cell
- 2D array of grid cells vs particle IDs is created, along with 1D array of counts of particles in each cell
- Requires thread atomics to avoid write conflicts
- If 2D array is too small, increase second dimension, realloc, and try again
- Tried parallel scan with CRS graph, but 2D array was faster for uniform particles/cell



Threaded Collide

- Each thread processes all the collisions in a grid cell
- Nearest neighbor algorithm also supported

Thread 1
Thread 2
Thread 3



Compiling and Running Kokkos Package

Sandia National Laboratories

- Start with a Kokkos Makefile included with SPARTA Kokkos
- Install the KOKKOS package using “make yes-kokkos” and compile
- Designed so that no changes to input script are needed to use threading
- Run with 4 OpenMP threads: “./spa_exe -in in.lj -k on t 4 -sf kk”
- Run with 4 MPI and 4 GPUs: “mpiexec -np 4 ./spa_exe -in in.colide -k on g 4 -sf kk”

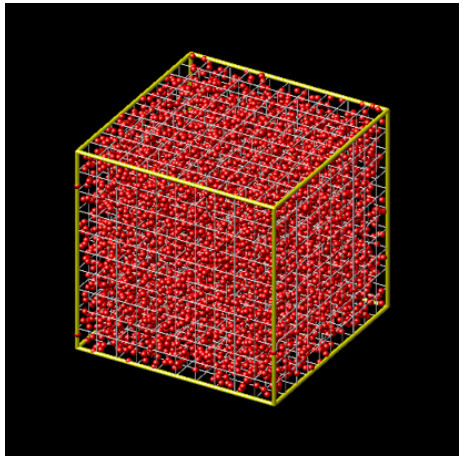
Algorithmic Specializations

Sandia National Laboratories

- Different algorithms may work better on different hardware
  - Typically atomic reduction of statistical accumulators is faster on GPUs, while parallel reduction is faster on CPUs/Xeon Phis
  - Threaded packing and unpacking of communication buffers can be faster on GPUs since it avoids host/device memory transfer, non-threaded is normally faster on CPUs/Xeon Phis
  - Reordering the particle list to align with grid cells can be faster on GPUs, but slower on CPUs
- These options are implemented in SPARTA Kokkos and can be toggled at the command line/input script

Benchmarking

- Collide benchmark
  - Particles advect though a uniform grid in a box with specular walls (argon at room temperature)
  - Variable soft sphere (VSS) collisions
  - 10 particles/grid cell on average
  - Can be scaled to arbitrary sizes
  - Equilibrate for 30 steps using 70 ns timestep, then benchmark for 100 steps using 7 ns timestep



Benchmark Machines

- **chama** = Intel SandyBridge CPUs
  - 1232-node cluster
  - 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

- **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, 68 cores + 4x hyperthreading
  - interconnect = Cray Aries Dragonfly
  - testbed meant to represent the Trinity supercomputer at LANL

Benchmark Machines

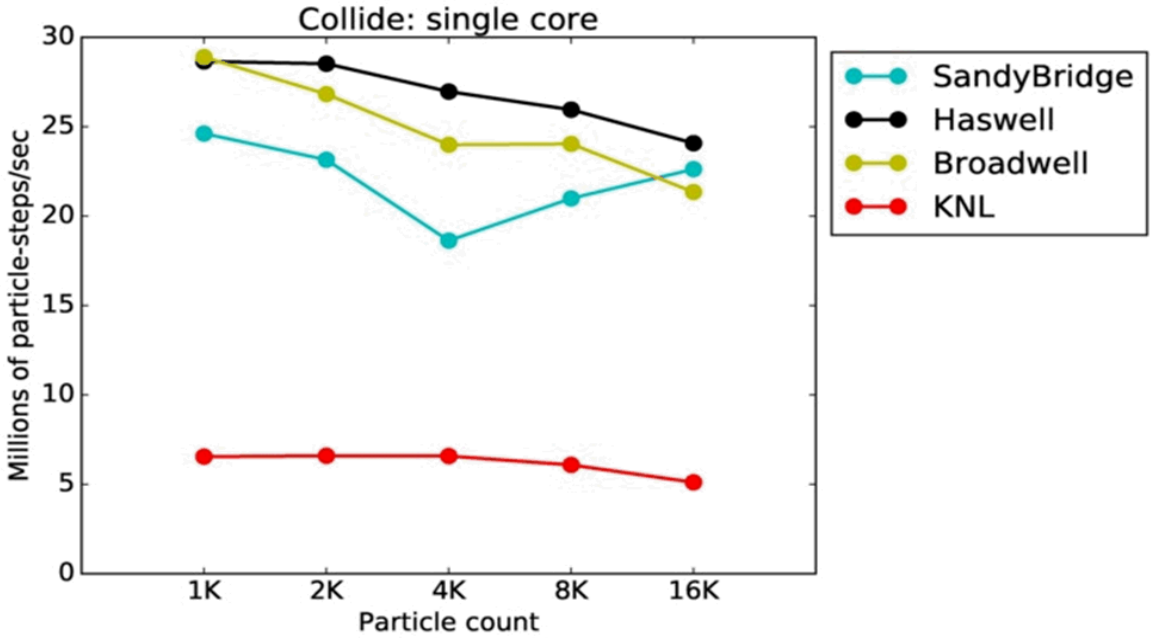
- **ride80** = IBM Power8 CPUs and NVIDIA K80 GPUs
  - 11 nodes
  - one node = dual Power8 3.42 GHz CPU (Firestone), 16 cores + 8x hyperthreading
  - each node has 2 Tesla K80 GPUs (each K80 is "dual" with 2 internal GPUs)
  - interconnect = Infiniband
- **ride100** = IBM Power8 CPUs and NVIDIA P100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

- Don’t know optimal number of MPI tasks vs OpenMP threads or number of hyperthreads to use *a priori*
- Use a parameter sweep to find optimal settings for the different packages
- Only best results of the parameter sweep included in the results shown here

Single Core

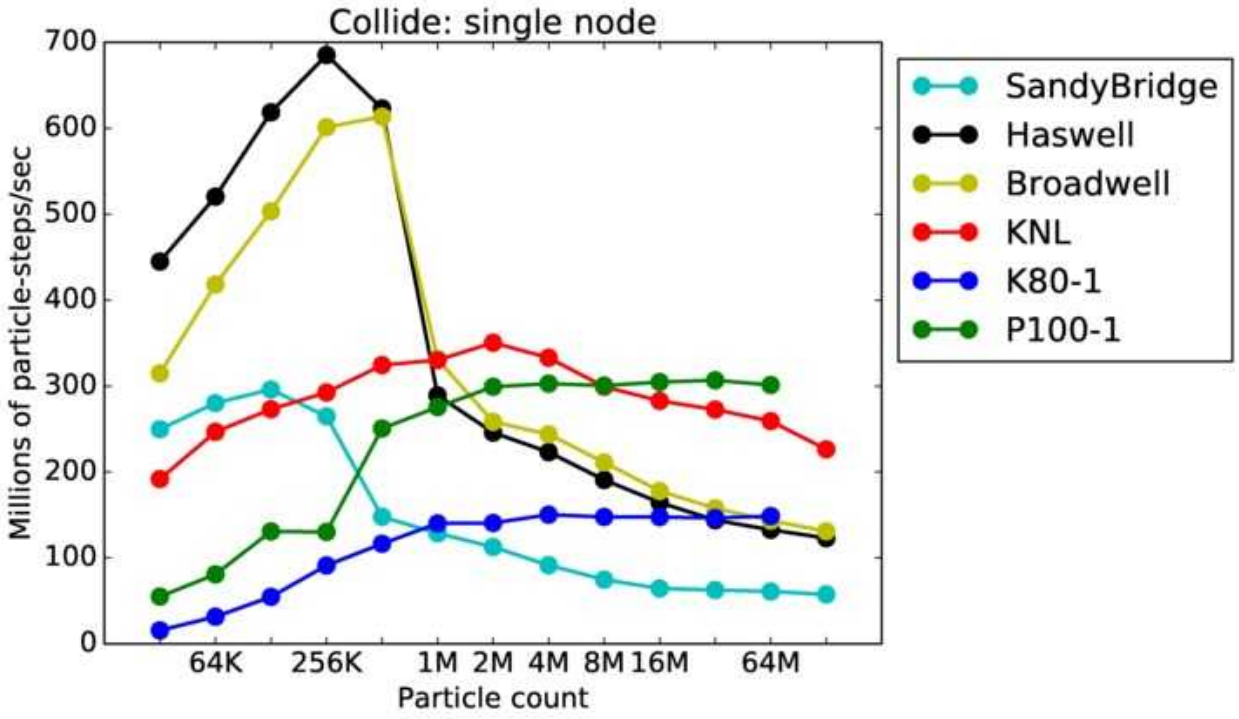
- 1 MPI rank with 1 OpenMP thread
- Best performance using either Kokkos or MPI only
- Not possible to run on 1 GPU core



*Results shown in this presentation may be improved in the future

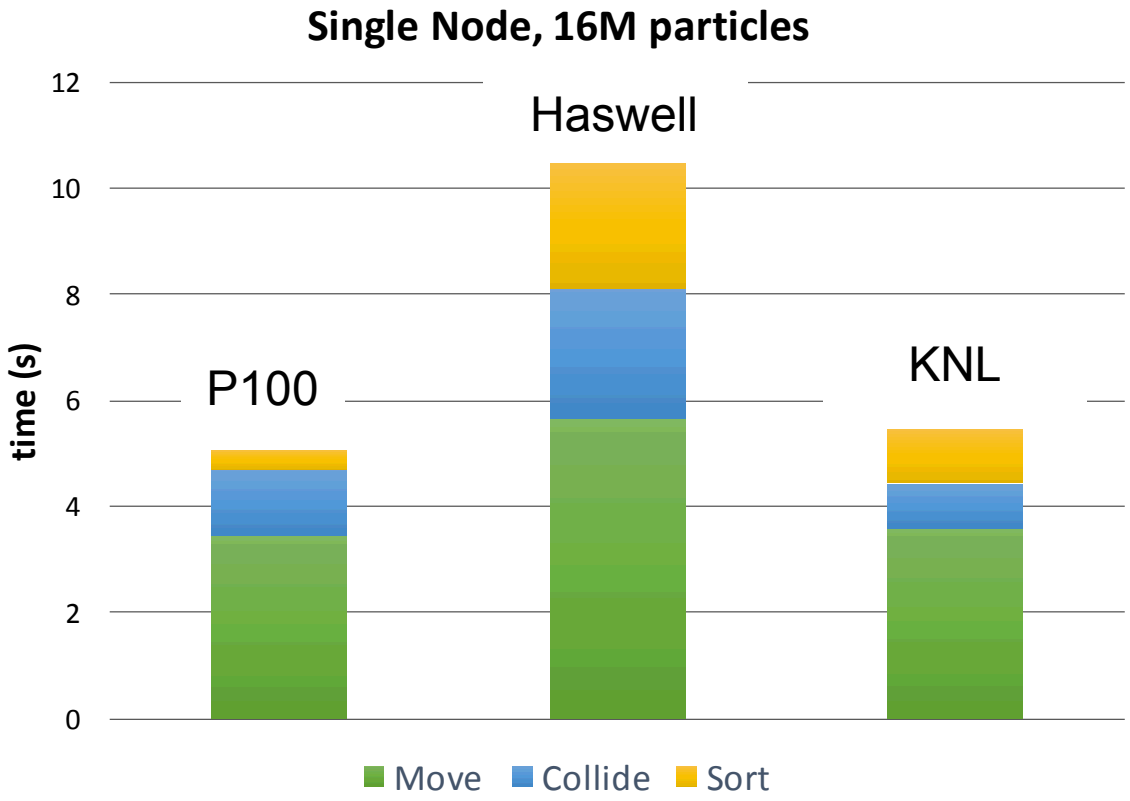
Single Node

- 1 CPU node or 1 GPU (2 logical GPUs for K80)
- Best performance using either Kokkos or MPI only
- Large cache effect for small problem sizes



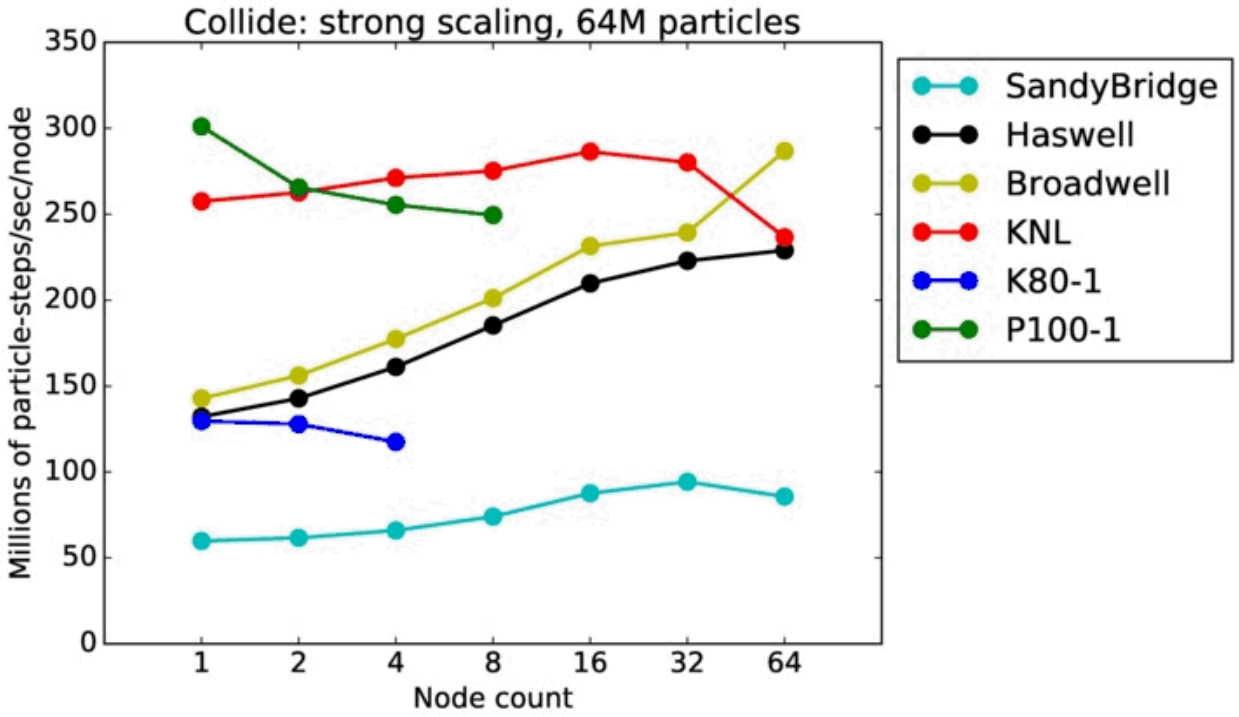
Timing Breakdown

- 1 CPU node or 1 GPU (2 logical GPUs for K80)



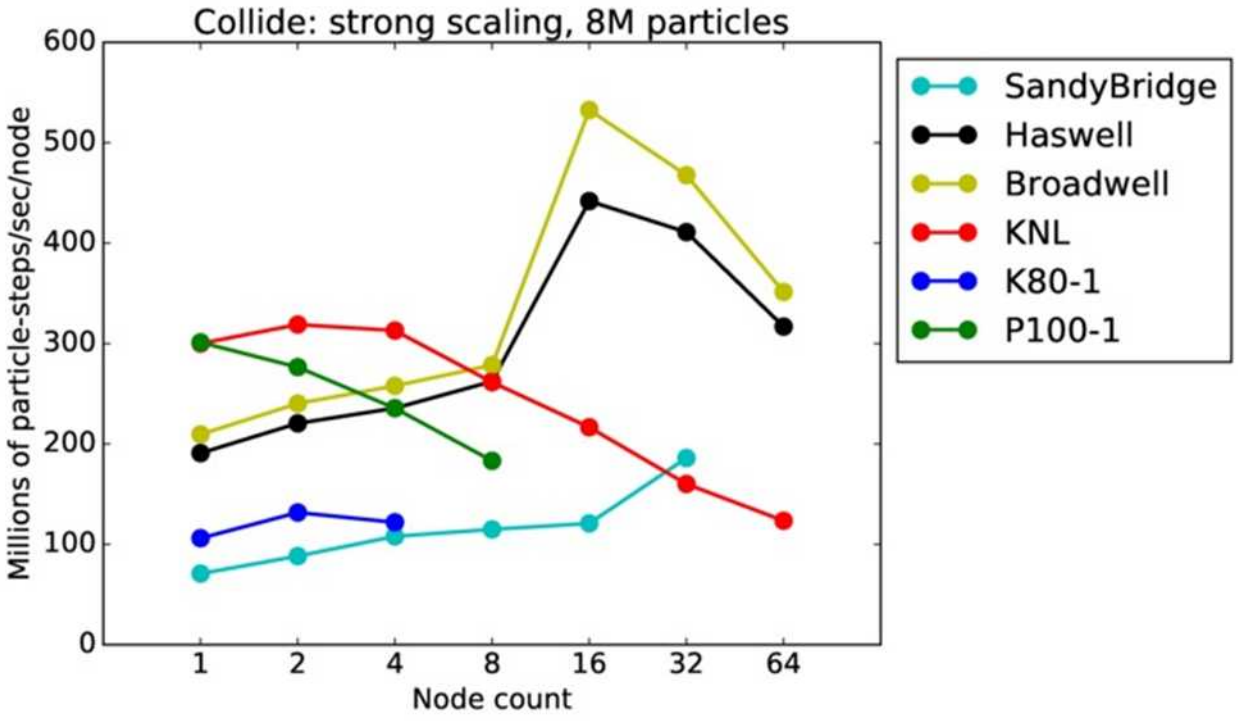
Strong Scaling, 64M particles

- Best performance using either Kokkos or MPI only



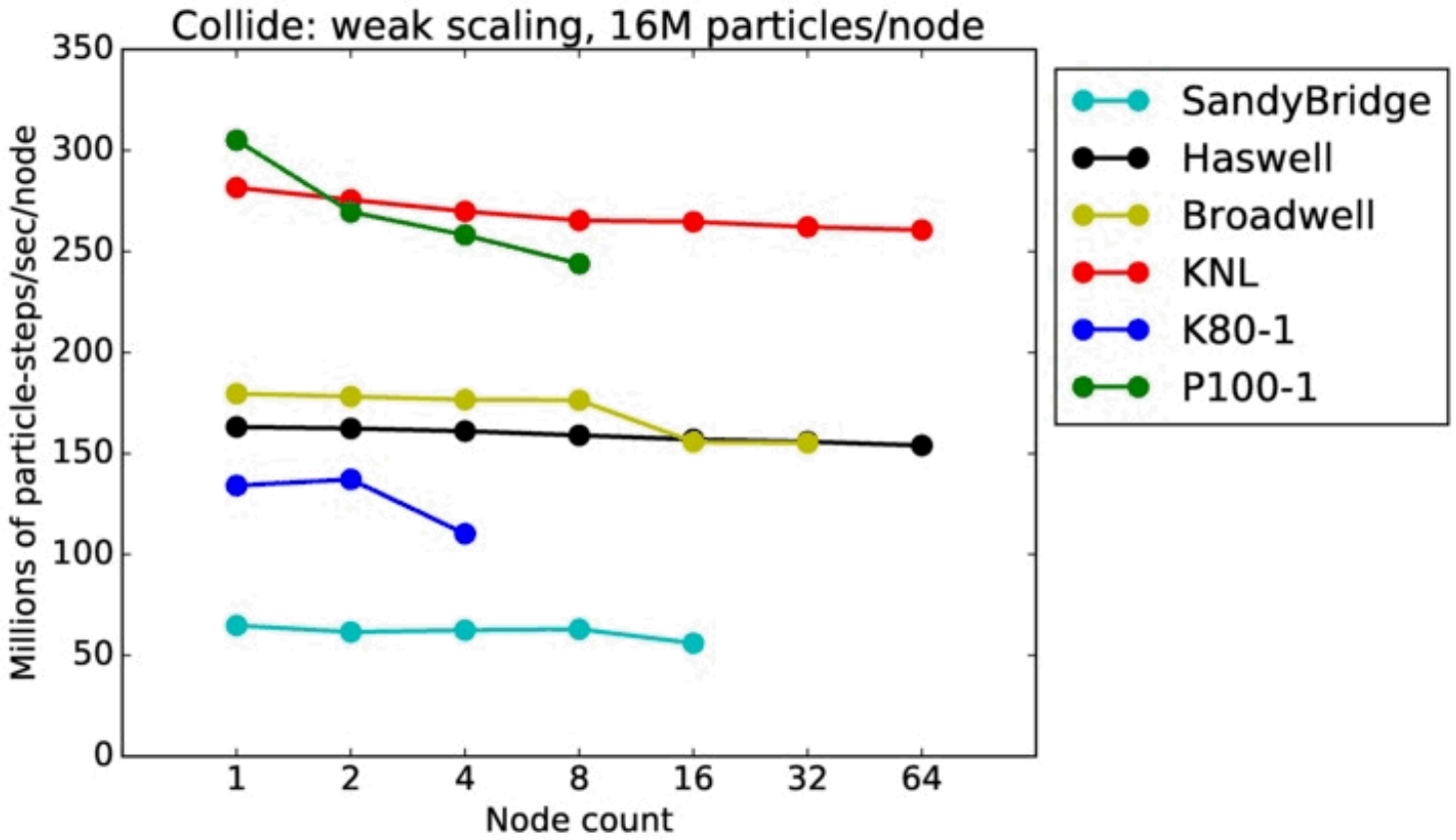
Strong Scaling, 8M particles

- Best performance using either Kokkos or MPI only



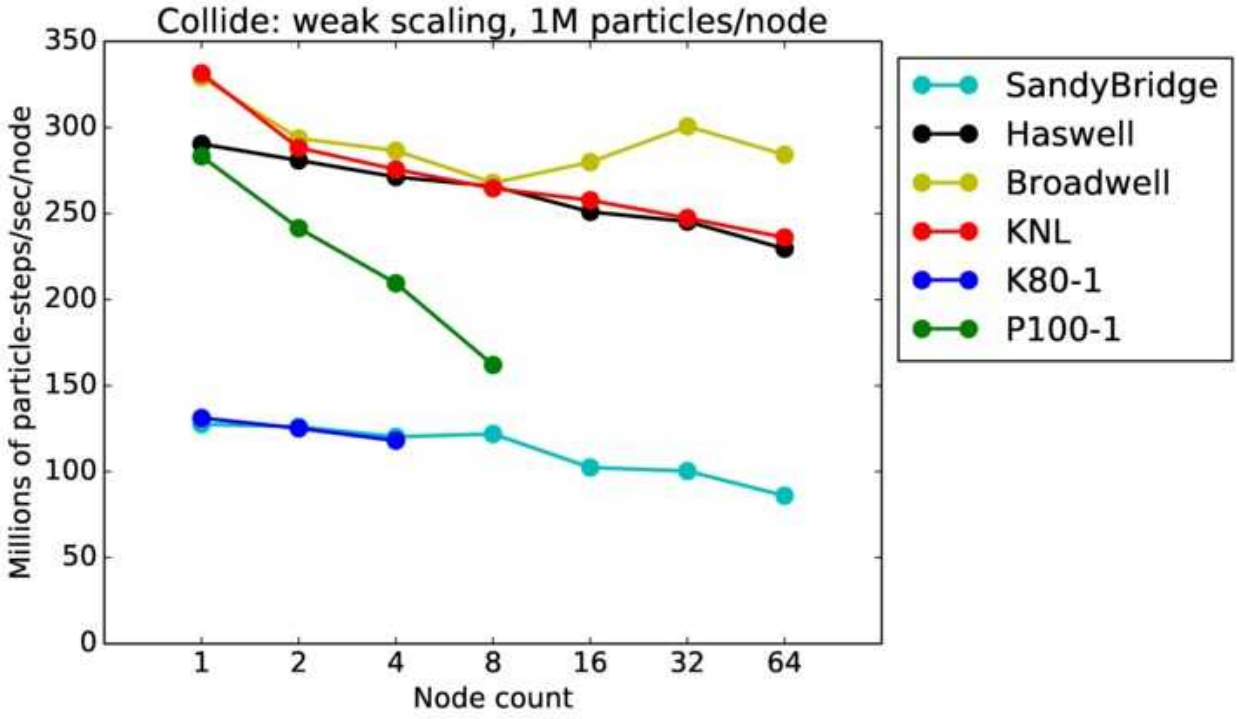
Weak Scaling, 16M particles/node

- Best performance using either Kokkos or MPI only



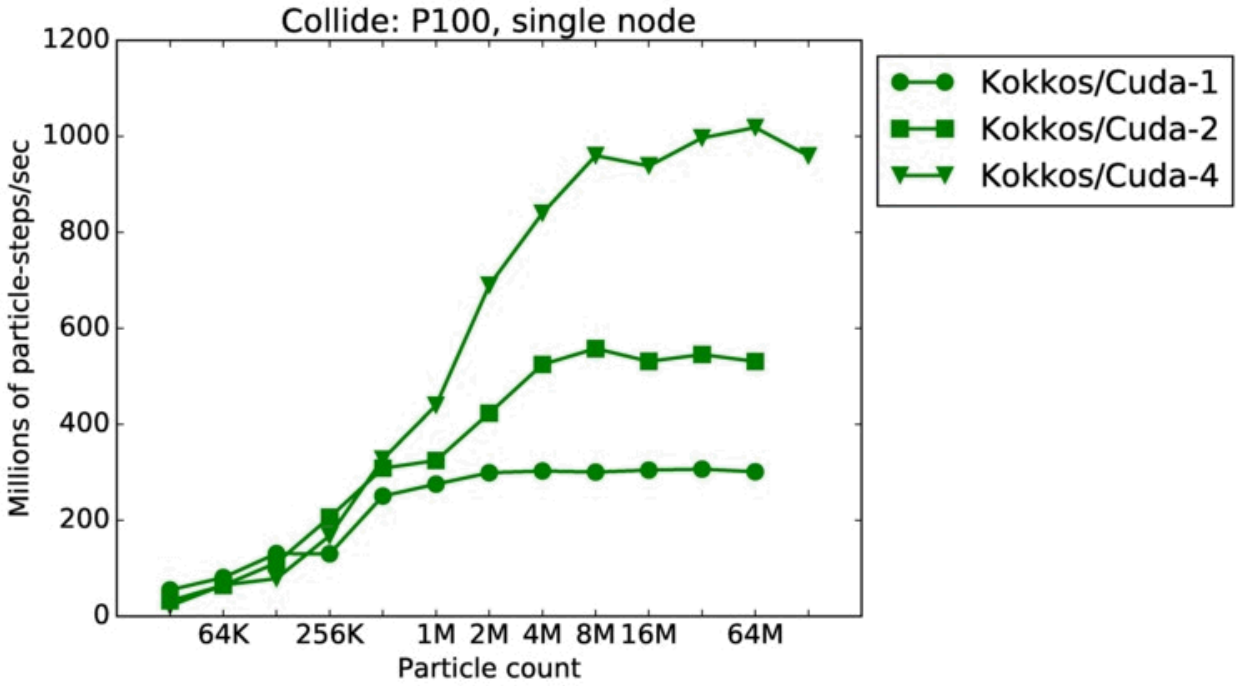
Weak Scaling, 1M particles/node

- Best performance using either Kokkos or MPI only



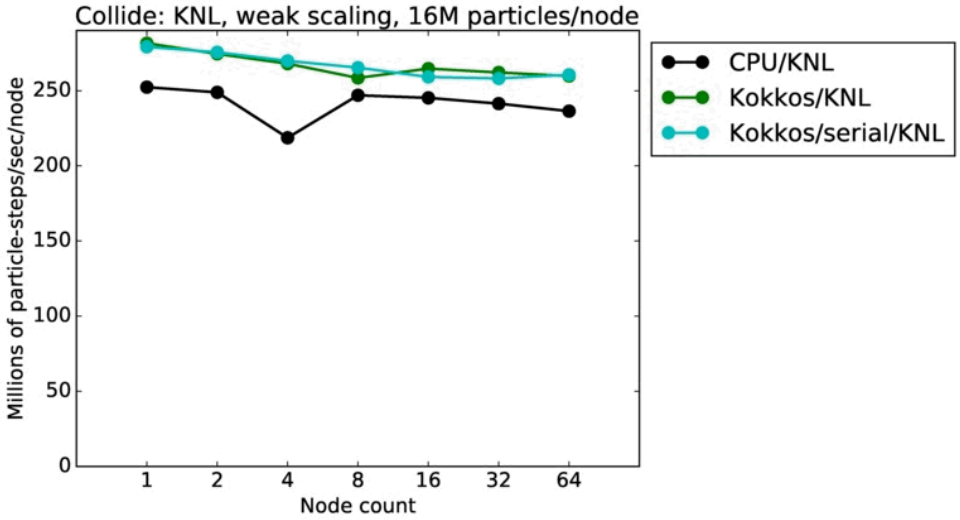
GPU Node Scaling

- Use 1, 2, or 4 P100 GPUs with Kokkos CUDA



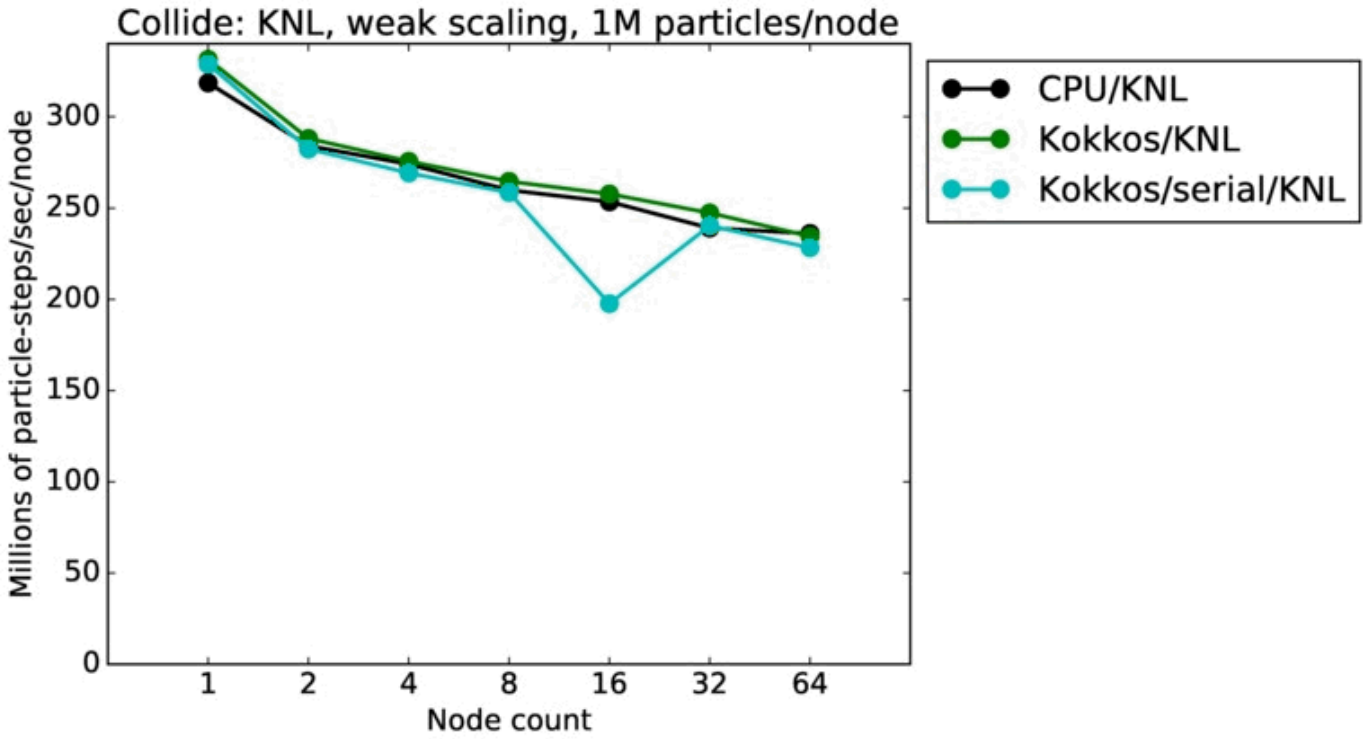
Kokkos OpenMP vs MPI Only

- KNL weak scaling
- CPU is MPI only, Kokkos is OpenMP, Kokkos/serial is MPI only using the Kokkos code
- Allowing MPI on the hyperthreads
- OpenMP threading has little to no benefit, may have more benefit at full scale (1000s of nodes)



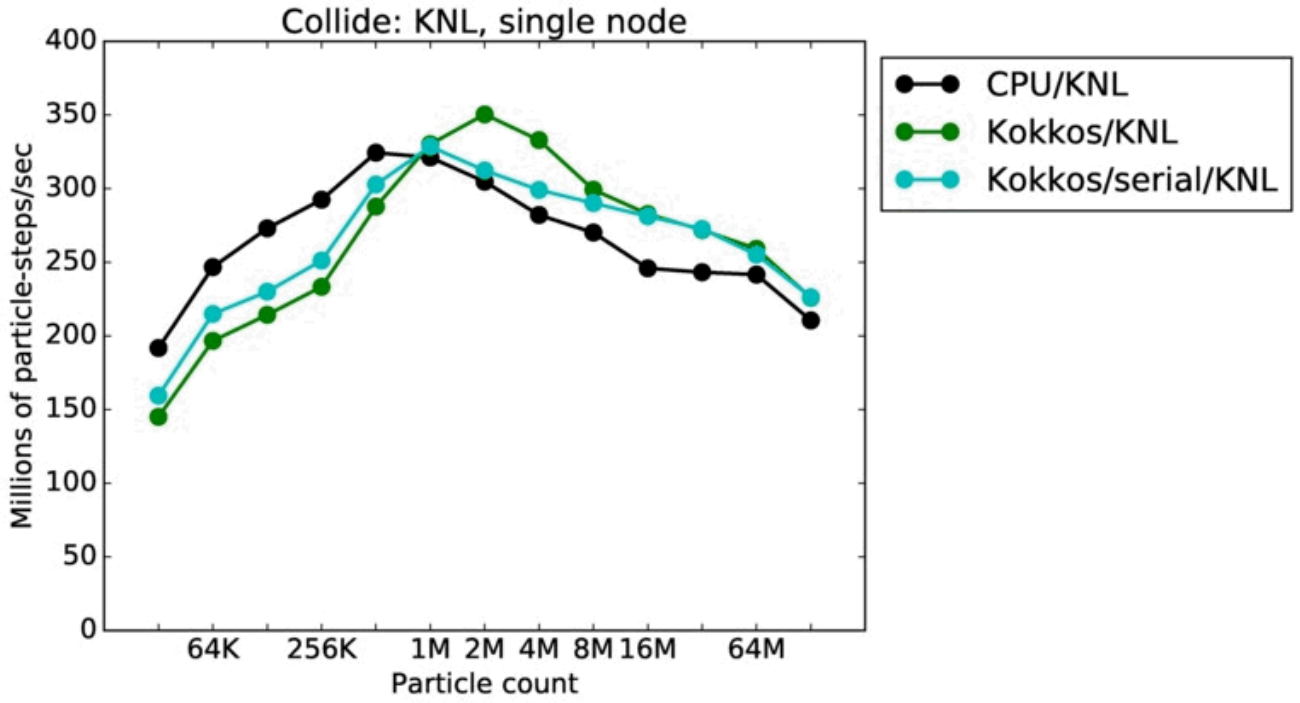
Kokkos OpenMP vs MPI Only

- KNL weak scaling



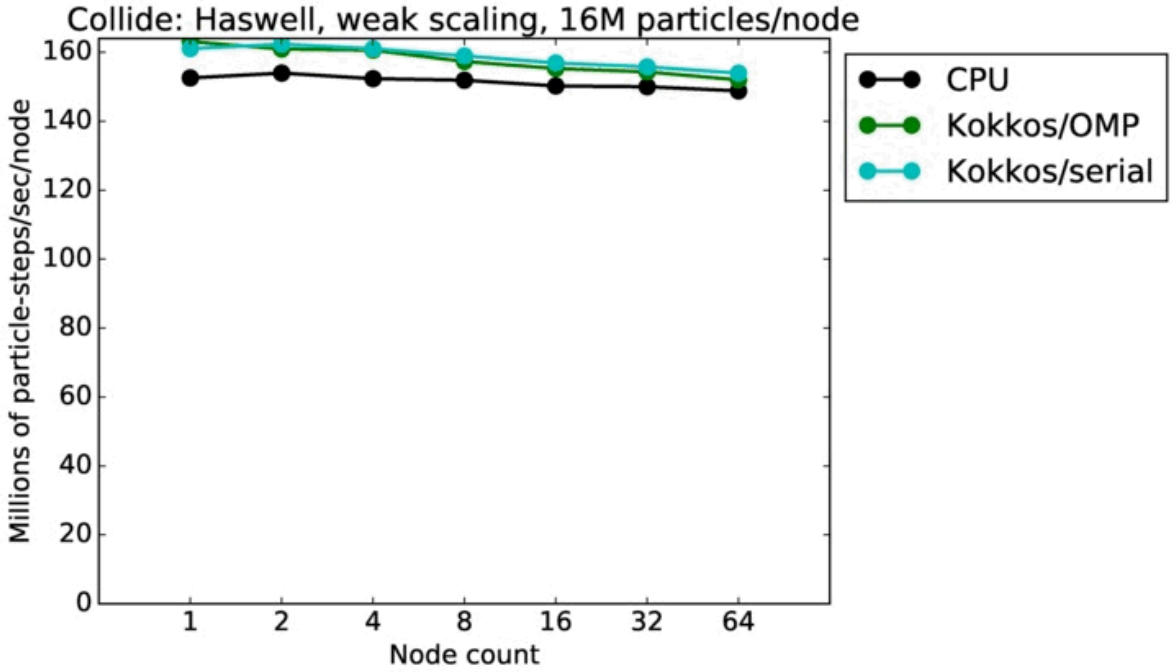
Kokkos OpenMP vs MPI Only

- Single Node KNL



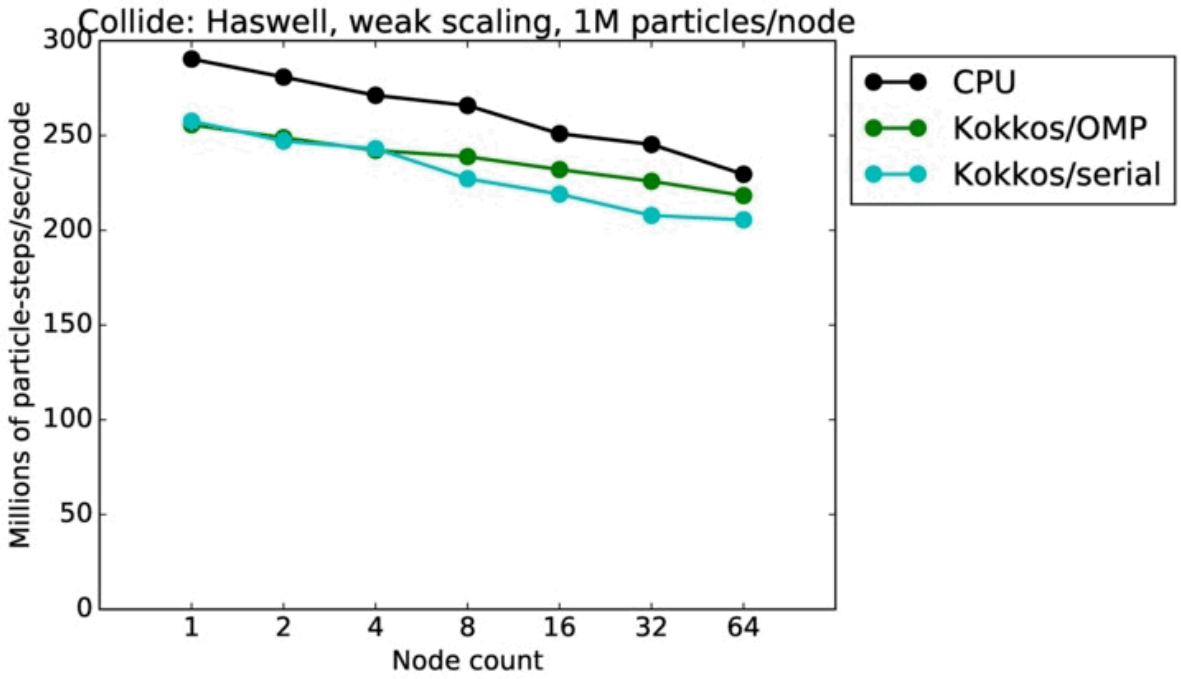
Kokkos OpenMP vs MPI Only

- Haswell weak scaling



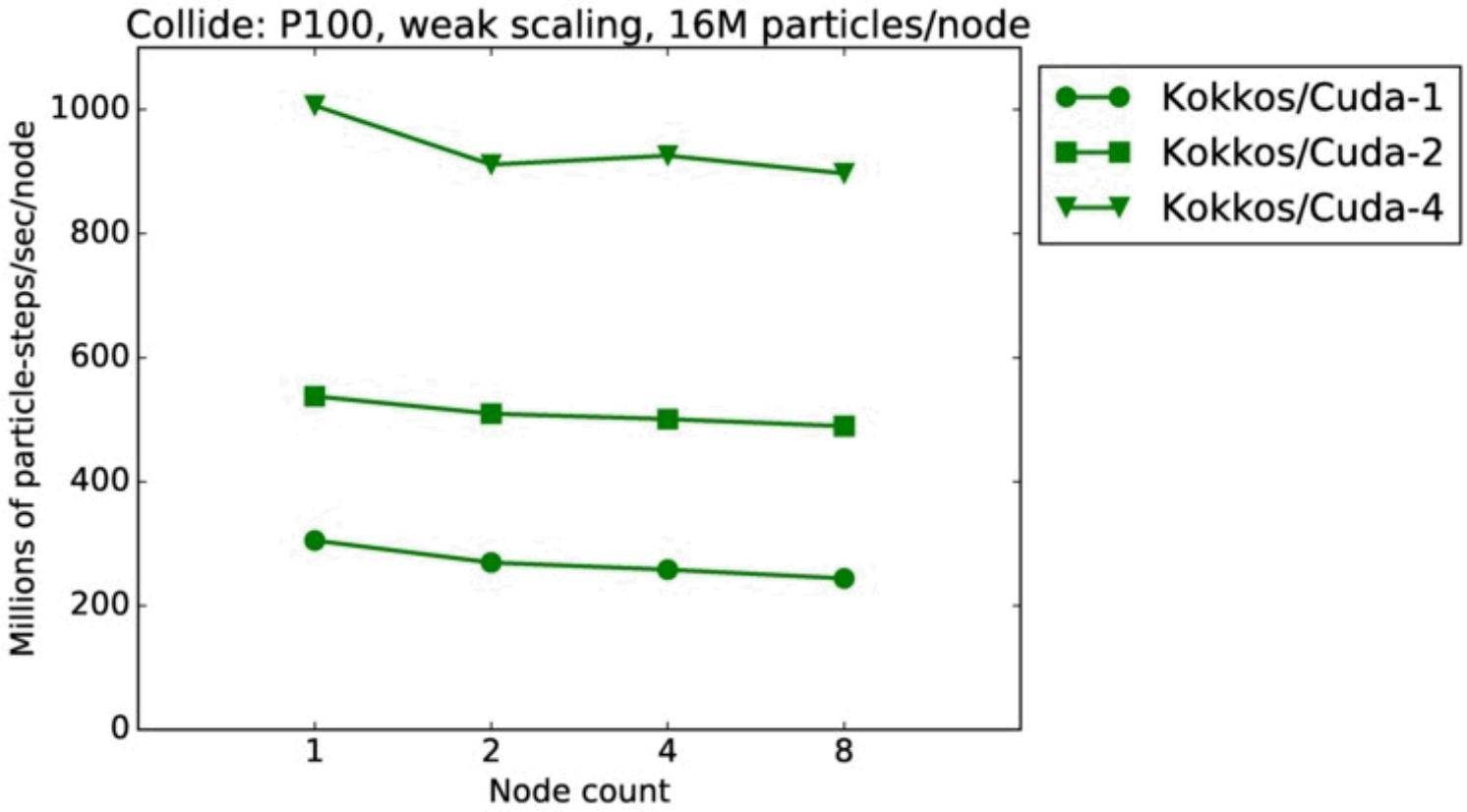
Kokkos OpenMP vs MPI Only

- Haswell weak scaling



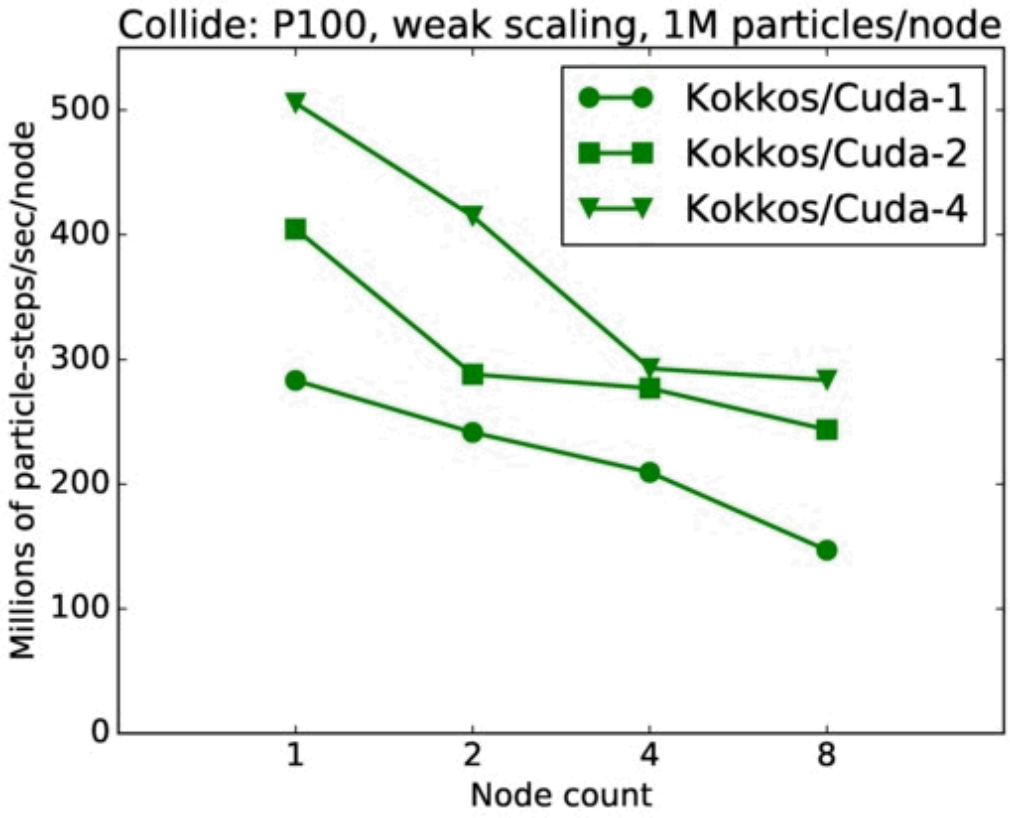
Kokkos OpenMP vs MPI Only

- P100 weak scaling using 1, 2, and 4 GPUs per node



Kokkos OpenMP vs MPI Only

- P100 weak scaling



New Benchmark Website

Sandia National Laboratories

- Very non-trivial to get optimal performance on modern HPC platforms
- New SPARTA benchmarking website will show performance plots for different hardware
  - Free molecular flow
  - Collisional flow
  - Flow around a sphere
- Will also include links to:
  - Tables of time for each run
  - Makefiles used for compiling SPARTA on different platforms
  - List of modules loaded
  - Exact MPI run command used, along with affinity settings
  - SPARTA logfiles for each run

New Benchmark Website (cont.)

- Screenshot of benchmarking website:

Single node performance, Collide benchmark
Performance in millions of particle-timesteps / second

Nparticles	SandyBridge	Haswell	Broadwell	KNL	K80-1	P100-1
32000	249.8 (CPU,mpi=16)	444.9 (CPU,mpi=32,hyper=1)	314.8 (CPU,mpi=36,hyper=1)	191.8 (CPU/KNL,mpi=64,hyper=1)	15.72 (Kokkos/Cuda,mpi=2)	55.04 (Kokkos/Cuda,mpi=1)
64000	280.3 (CPU,mpi=16)	520.4 (CPU,mpi=64,hyper=2)	418.2 (CPU,mpi=36,hyper=2)	246.8 (CPU/KNL,mpi=64,hyper=1)	31.62 (Kokkos/Cuda,mpi=2)	80.94 (Kokkos/Cuda,mpi=1)
128000	296.1 (CPU,mpi=16)	618.6 (CPU,mpi=64,hyper=2)	503.3 (CPU,mpi=72,hyper=2)	273.1 (CPU/KNL,mpi=64,hyper=1)	54.74 (Kokkos/Cuda,mpi=2)	130.8 (Kokkos/Cuda,mpi=1)
256000	264.8 (CPU,mpi=16)	685.2 (CPU,mpi=64,hyper=2)	601.3 (CPU,mpi=64,hyper=2)	292.5 (CPU/KNL,mpi=128,hyper=2)	91.27 (Kokkos/Cuda,mpi=2)	130.2 (Kokkos/Cuda,mpi=1)
512000	147.9 (CPU,mpi=16)	623 (CPU,mpi=64,hyper=2)	613.4 (CPU,mpi=72,hyper=2)	324.4 (CPU/KNL,mpi=128,hyper=2)	116.6 (Kokkos/Cuda,mpi=2)	250.7 (Kokkos/Cuda,mpi=1)
1024000	128.7 (CPU,mpi=16)	289.2 (CPU,mpi=64,hyper=2)	331.2 (CPU,mpi=72,hyper=2)	330.2 (Kokkos/KNL,mpi=64,thread=4,hyper=4)	140.2 (Kokkos/Cuda,mpi=2)	275.5 (Kokkos/Cuda,mpi=1)
2048000	112.7 (CPU,mpi=16)	245.9 (Kokkos/serial,mpi=64,hyper=2)	258.4 (CPU,mpi=72,hyper=2)	350.4 (Kokkos/KNL,mpi=64,thread=4,hyper=4)	140.7 (Kokkos/Cuda,mpi=2)	299.2 (Kokkos/Cuda,mpi=1)
4096000	91.56 (CPU,mpi=16)	223.1 (Kokkos/OMP,mpi=64,hyper=2,thread=1)	243.9 (Kokkos/serial,mpi=72,hyper=2)	333 (Kokkos/KNL,mpi=64,thread=4,hyper=4)	150.3 (Kokkos/Cuda,mpi=2)	302.9 (Kokkos/Cuda,mpi=1)
8192000	74.74 (Kokkos/serial,mpi=16)	190.8 (Kokkos/serial,mpi=64,hyper=2)	211.2 (Kokkos/serial,mpi=72,hyper=2)	299.3 (Kokkos/KNL,mpi=32,thread=8,hyper=4)	147.7 (Kokkos/Cuda,mpi=2)	300.5 (Kokkos/Cuda,mpi=1)
16384000	64.46 (CPU,mpi=16)	164.5 (Kokkos/serial,mpi=64,hyper=2)	177.9 (Kokkos/serial,mpi=72,hyper=2)	282.8 (Kokkos/KNL,mpi=256,thread=1,hyper=4)	147.6 (Kokkos/Cuda,mpi=2)	304.7 (Kokkos/Cuda,mpi=1)
32768000	62.67 (Kokkos/serial,mpi=16)	144 (Kokkos/serial,mpi=64,hyper=2)	157.8 (Kokkos/serial,mpi=72,hyper=2)	272.7 (Kokkos/serial/KNL,mpi=256,hyper=4)	146.5 (Kokkos/Cuda,mpi=2)	306.7 (Kokkos/Cuda,mpi=1)
65536000	60.96 (Kokkos/serial,mpi=16)	132.7 (Kokkos/serial,mpi=64,hyper=2)	143.5 (Kokkos/serial,mpi=72,hyper=2)	259.2 (Kokkos/KNL,mpi=256,thread=1,hyper=4)	148.7 (Kokkos/Cuda,mpi=2)	301.2 (Kokkos/Cuda,mpi=1)
131072000	57.58 (Kokkos/serial,mpi=16)	123 (Kokkos/serial,mpi=64,hyper=2)	131 (Kokkos/OMP,mpi=72,hyper=2,thread=1)	226.4 (Kokkos/serial/KNL,mpi=256,hyper=4)	None	None

Run commands and logfile links for column SandyBridge

32000	mpirun -n 16 -N 16 --bind-to core spa_chama_cpu -v x 16 -v y 10 -v z 20 -v t 100 -in in.collide.steps -log log.sparta.date=8Aug17.model=collide.machine=chama.pkg=cpu.kind=node.size=32K.node=1.mpi=16
64000	mpirun -n 16 -N 16 --bind-to core spa_chama_cpu -v x 16 -v y 20 -v z 20 -v t 100 -in in.collide.steps -log log.sparta.date=8Aug17.model=collide.machine=chama.pkg=cpu.kind=node.size=64K.node=1.mpi=16
128000	mpirun -n 16 -N 16 --bind-to core spa_chama_cpu -v x 32 -v y 20 -v z 20 -v t 100 -in in.collide.steps -log log.sparta.date=8Aug17.model=collide.machine=chama.pkg=cpu.kind=node.size=128K.node=1.mpi=16
256000	mpirun -n 16 -N 16 --bind-to core spa_chama_cpu -v x 32 -v y 20 -v z 40 -v t 100 -in in.collide.steps -log log.sparta.date=8Aug17.model=collide.machine=chama.pkg=cpu.kind=node.size=256K.node=1.mpi=16
512000	mpirun -n 16 -N 16 --bind-to core spa_chama_cpu -v x 32 -v y 40 -v z 40 -v t 100 -in in.collide.steps -log log.sparta.date=8Aug17.model=collide.machine=chama.pkg=cpu.kind=node.size=512K.node=1.mpi=16

Automation

- 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
- SPARTA Kokkos is constantly being improved; can rerun the benchmarks and regenerate the webpage with updated results

Regression Testing

Sandia National Laboratories

- Currently 9 regression tests for Kokkos SPARTA run on:
  - multiple CPU cores using multiple OpenMP threads
  - multiple GPUs
  - multiple CPU cores using a single thread in “exact” mode (uses Kokkos but gives the exact same output as the non-Kokkos version of SPARTA)
- Testing is automated using the Jenkins continuous integration server
- Regression testing helps improve the quality and reliability of Kokkos SPARTA

Open Source Release

- MPI only version of SPARTA is available at http://sparta.sandia.gov
- Kokkos version (along with documentation) should be released soon (by the end of 2017)
- New benchmarking website also to be released

Conclusions

- Kokkos abstractions provide performance portability by allowing SPARTA to run on GPUs and use OpenMP threads on CPUs/Xeon Phis
- OpenMP threading has little benefit for low node counts, may have more benefit when using 1000s of nodes
- Current and future work:
  - Multithreaded support for complex surfaces, load balancing, adaptive grids, etc. in progress
  - Improve performance on CPU and KNL through vectorization
  - Increase coverage of the KOKKOS package in SPARTA

Sandia National Laboratories

Thank you

Questions?