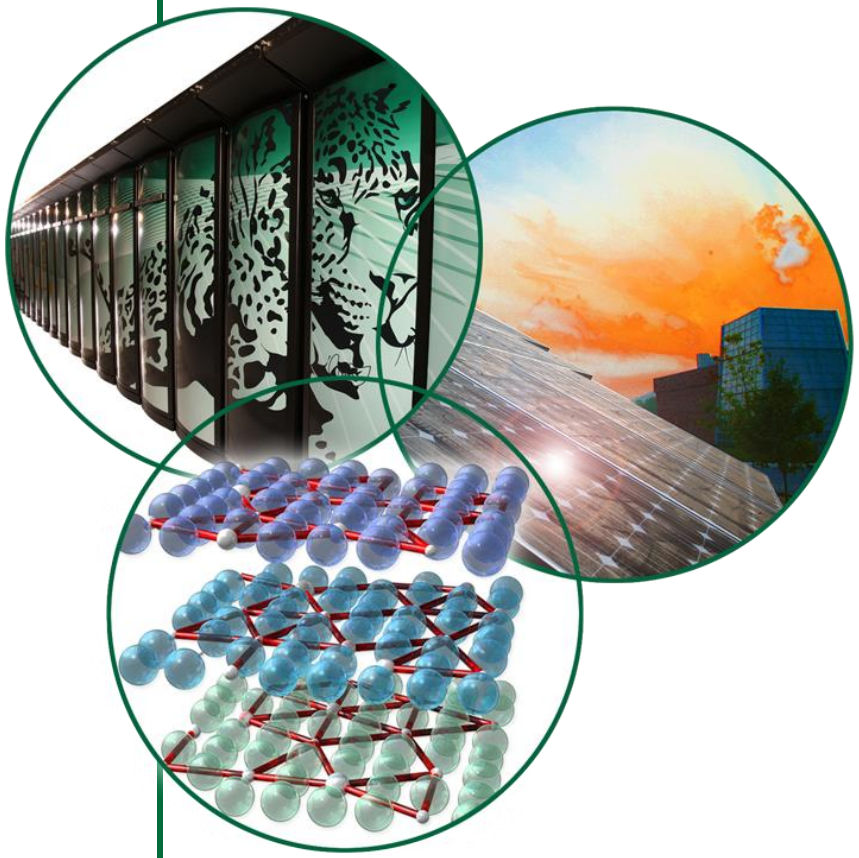
Application Case Studies



Kyle Spafford

Keeneland Tutorial

April 15, 2011

U.S. DEPARTMENT OF ENERGY

Future Technologies Group ornl

OAK RIDGE NATIONAL LABORATORY

MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY

Outline

- Case Study #1:
  - DCA++: Quantum Monte Carlo, Materials Science
  - Jeremy Meredith (ORNL)

- Case Study #2:
  - S3D: Compressible Flow, Turbulent Combustion
  - Kyle Spafford (ORNL), Gregory Reutsch, Norbert Juffa (NVIDIA)

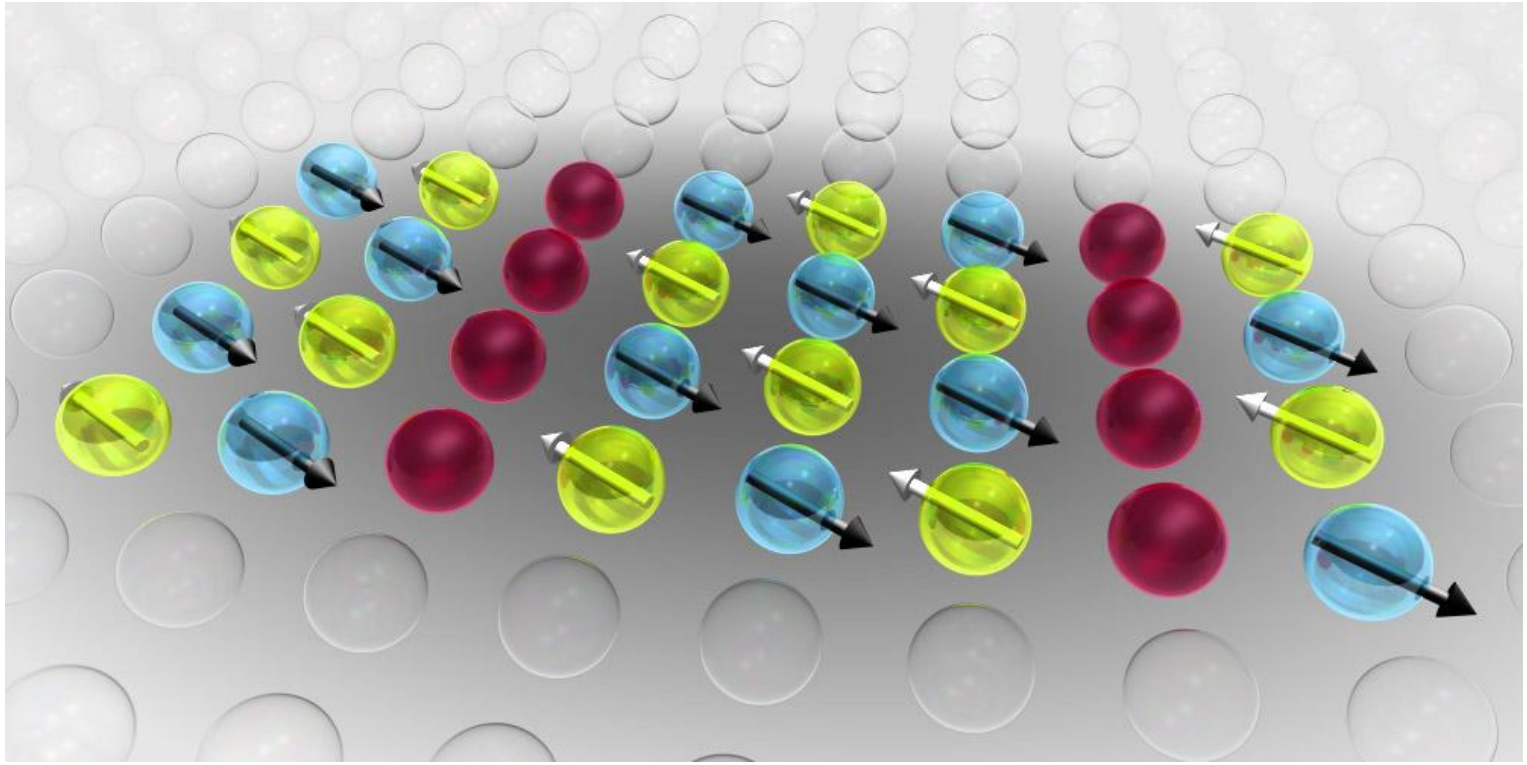
- Case Study #3
  - LAMMPS: Molecular Dynamics
  - Mike Brown, Paul Crozier, Steve Plimpton, Peng Wang

"Recipe" For Accelerating Applications

1. Profile your existing code to find where time is being spent
2. Choose kernels that represent a high fraction of runtime and map well to GPU hardware
3. Extract test harness and test problem
4. Implement and optimize kernels
5. Integrate with application, and minimize PCIe transfers

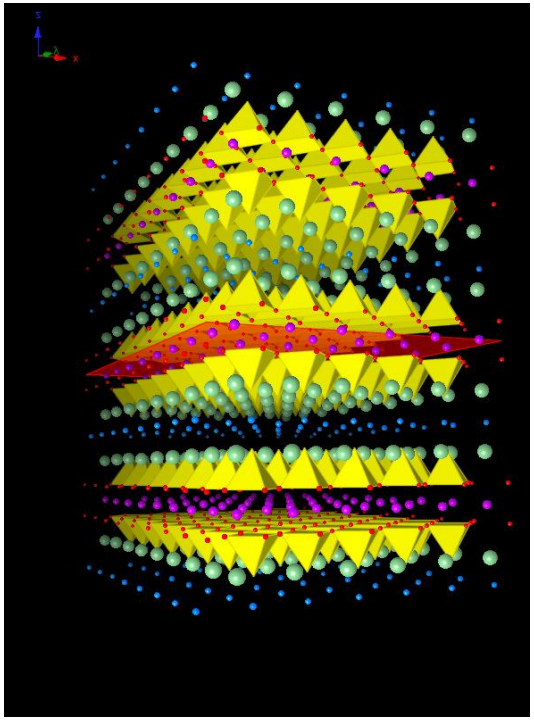
App #1: DCA++

- Quantum Monte Carlo materials science code
- Used for the study of high-temperature superconductors

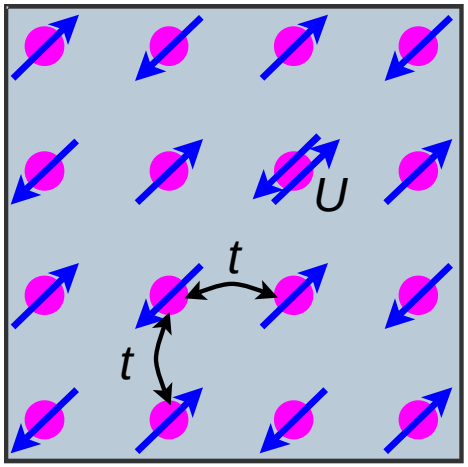


Quantum Cluster Approximation in the DCA++ code

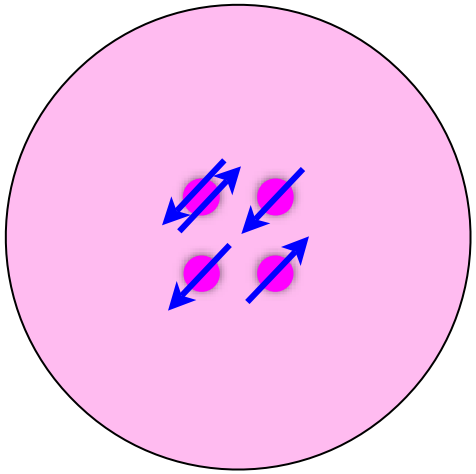
HTSC: 10^{23}
interacting electrons



2D Hubbard model
for CuO planes



Dynamic cluster
approximation: Map
Hubbard model onto
embedded cluster



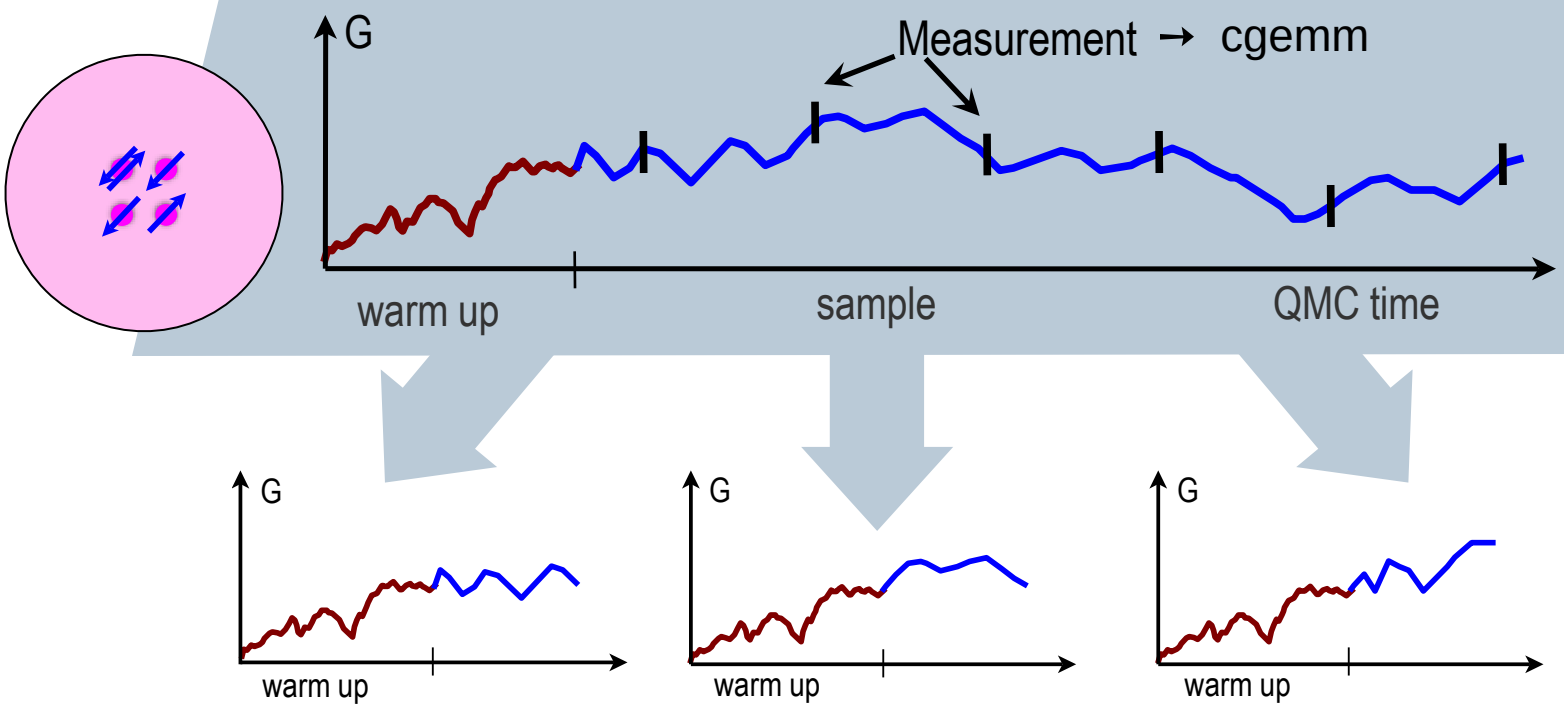
Quantum cluster theories: T.A. Maier *et al.*, RMP '05

Hirsch-Fye auxiliary field QMC cluster solver

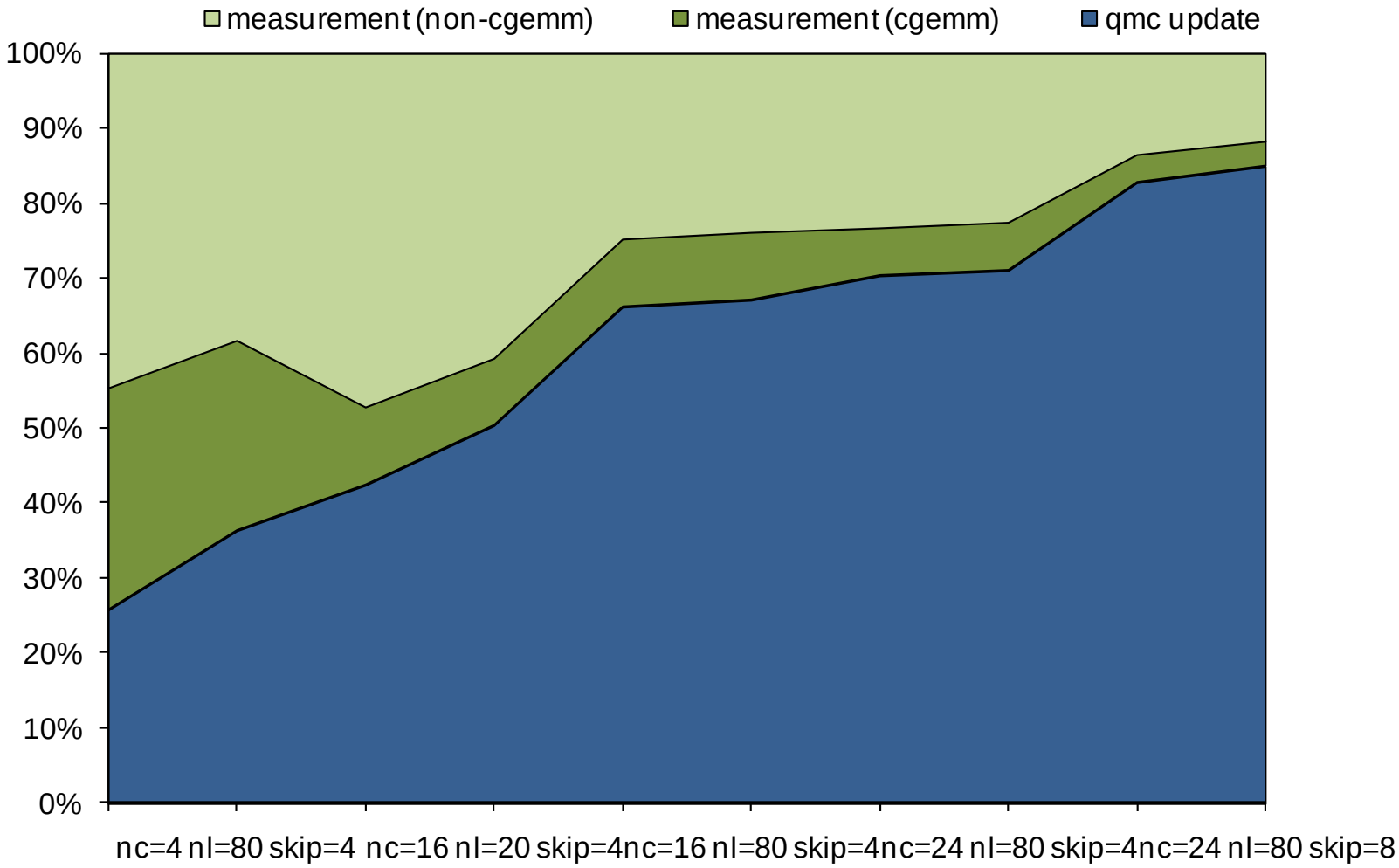
$\hat{G}' = \hat{G} + \vec{a} \times \vec{b}$ (vector outer product) → dger (N=4480)

or delay updating:

$\hat{G}' = \hat{G} + A\,B$ (matrix-matrix multiply) → dgemm (4480 x 32)

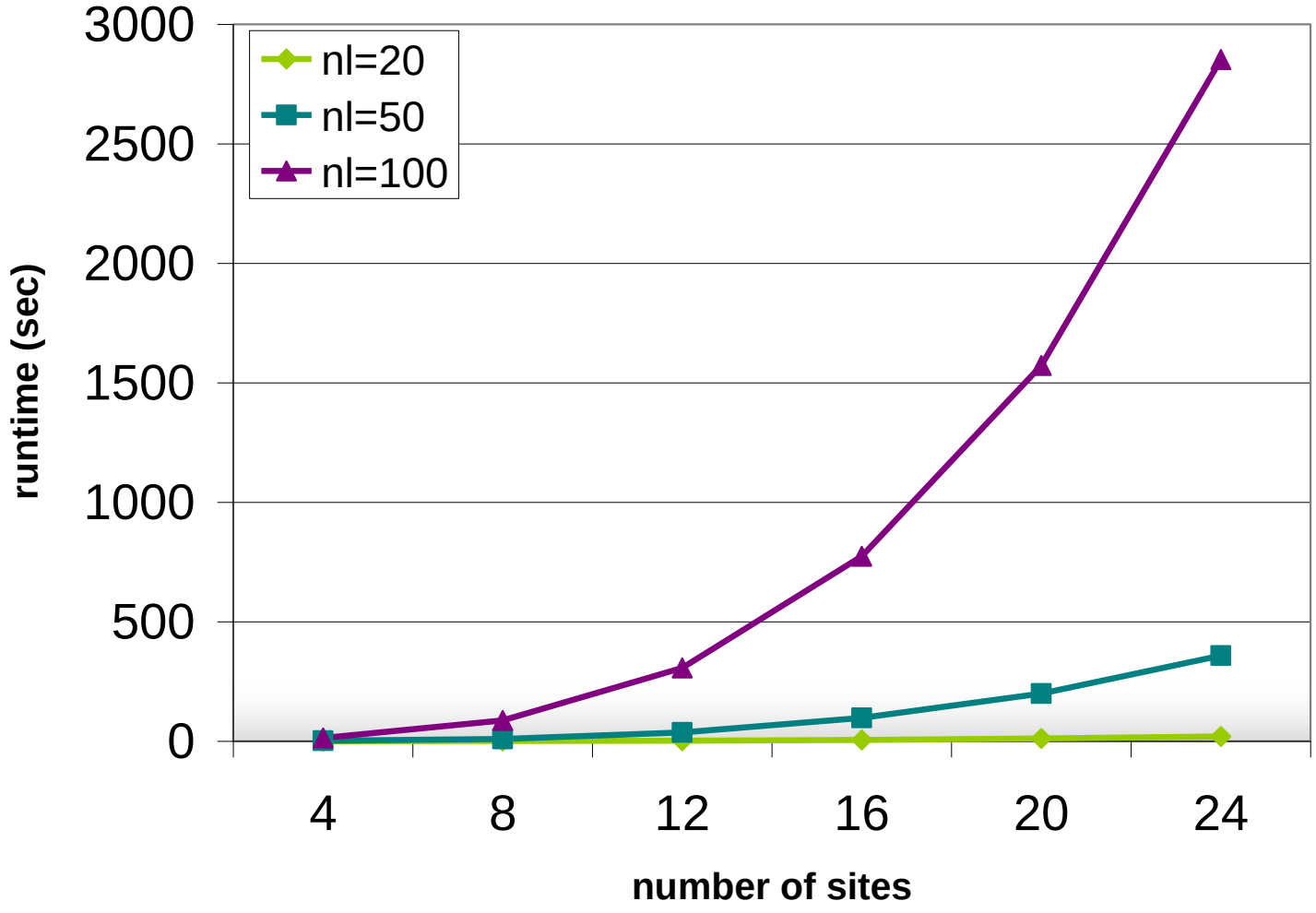


Finding a Target for Acceleration



- **32-task MPI, CPU single-precision**

CPU-only Performance (linear scale)

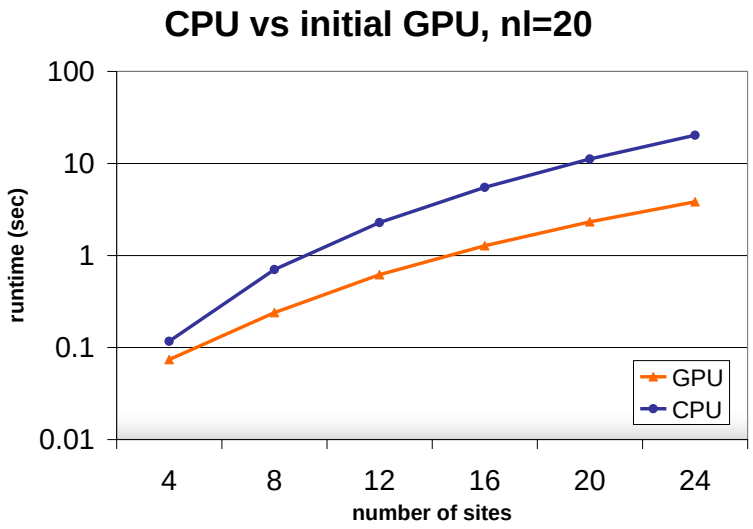


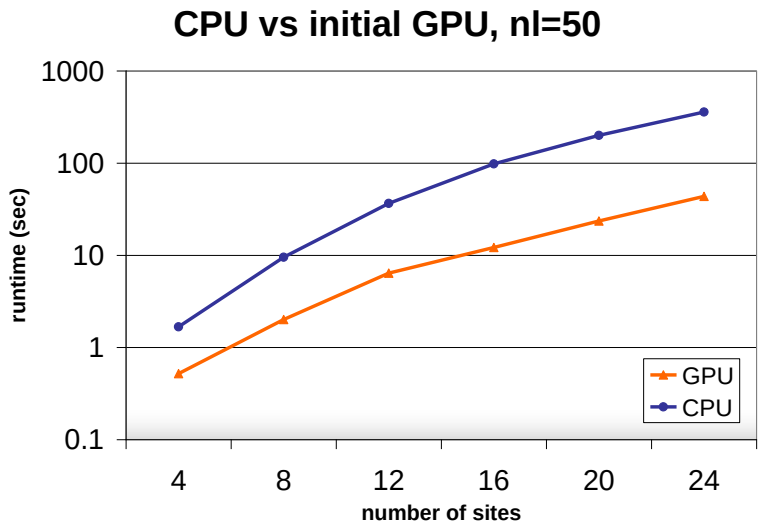
Initial GPU port

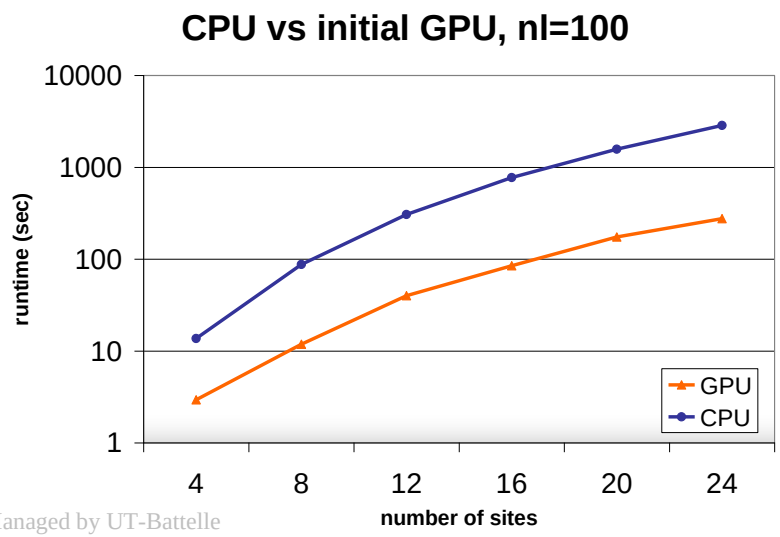
- Target the matrix-matrix multiply (GEMM)
  - majority of runtime, more so for large configurations
  - We'll use the cublas library (linear algebra routines on the GPU)

- GPU memory exists on the card itself
  - Must send arrays over PCI-Express bus
  - Example:
    - Send A,B,C to GPU over PCIe
    - Perform GPU-based GEMM on A,B,C
    - Read result C from GPU over PCIe

Initial GPU port: GEMM only, naïve data transfer







- **Speedups:**
  - 1.6x (nl=20 nc=4)
  - 10.3x (nl=100 nc=24)
- **Effort:**
  - minimal LOC
  - compilation environment

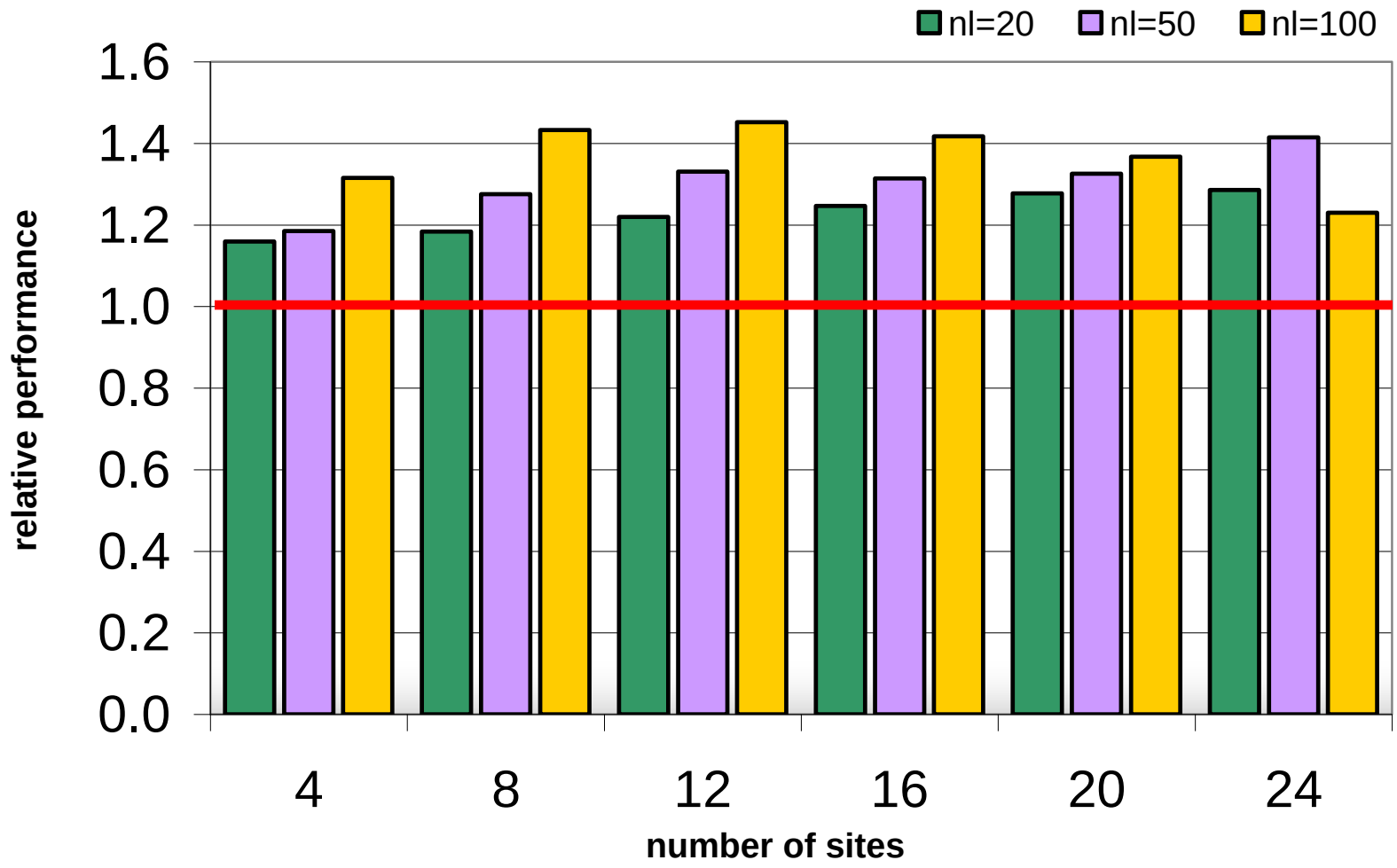
Memory Management

- Worst approach
  - Allocate and transfer to GPU for each call
  - Completely transparent to application
- Slightly less naïve approach
  - Allocate once and transfer to GPU for each call
  - Internal API changes or CPU-GPU mapping
- Lazy transfers
  - Only transfer to/from GPU when needed
  - “Real” effort

Smart Data Transfers

- Required implementing an "Array Handler"
  - Track when arrays are modified
  - Track when arrays will be needed
  - Encapsulate CPU/GPU array mapping
  - Transfer only when something has changed
  - Instrument application to utilize it

Smart Data Transfers: Advantage over Naïve Transfers

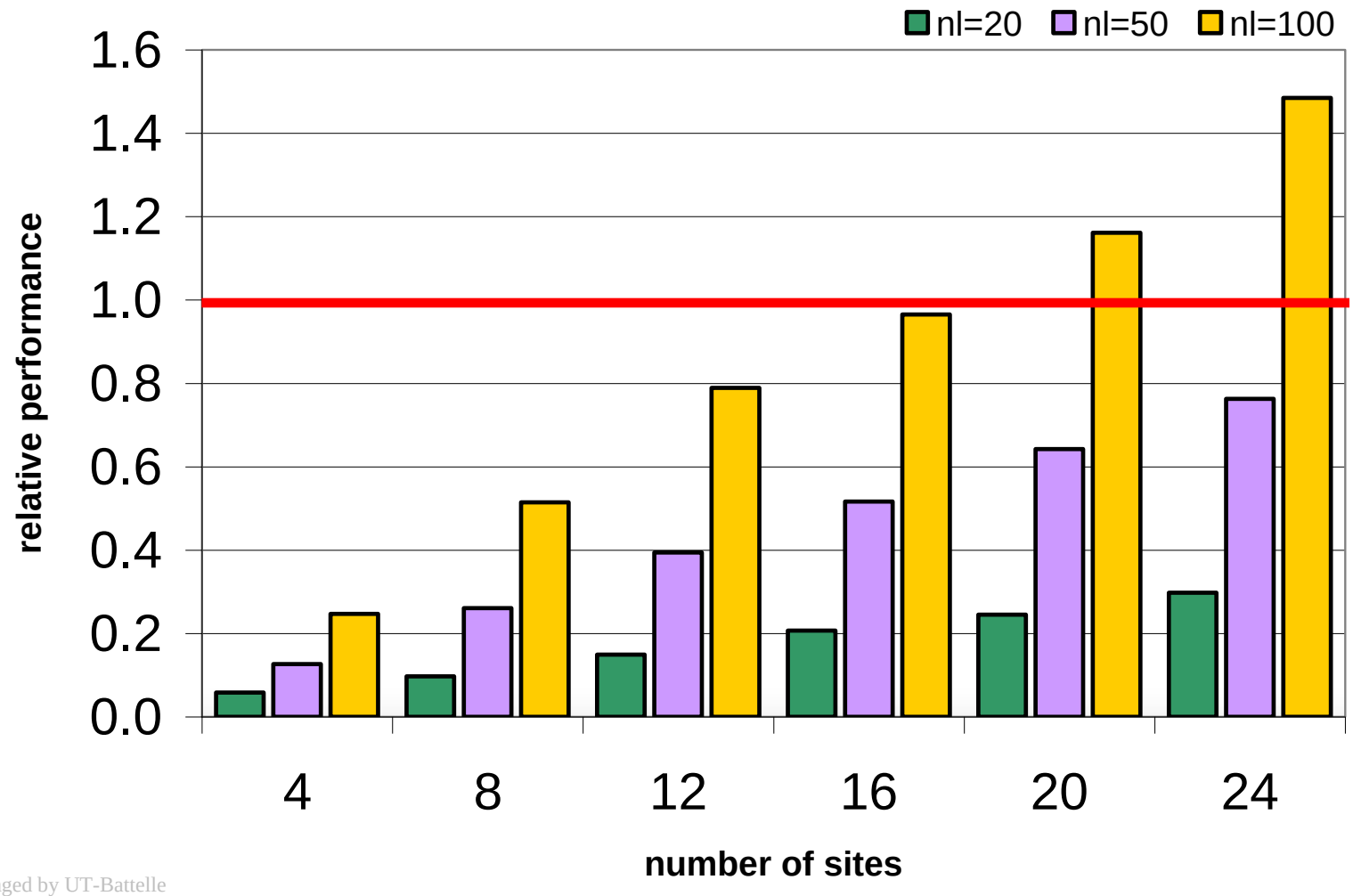


Minimizing PCIe Transfers

- Functions that are otherwise insignificant become bottlenecks (because PCIe is so slow)
  - If they require data to be on the CPU, they may be causing extra data transfers
  - Exploring functions that may even be a poor match for GPU acceleration is worthwhile

- Disadvantage: It's tons of work to port over large code bases

Accelerating Minor Functions: Advantage over GEMM-only



Accelerating Minor Functions

- Speedups:
  - 17x slower on small problems
  - 1.5x faster on large problems

- Effort:
  - more significant
    - these functions aren't necessarily a good match for GPUs
    - which makes them harder to port
  - and more dangerous
    - if they are not a good match, they can slow your code down

Performance Summary for QMC Update

CPU: single 2.0GHz Opteron

GPU: single 500MHz/96SP G80 (8800GTS 320MB)

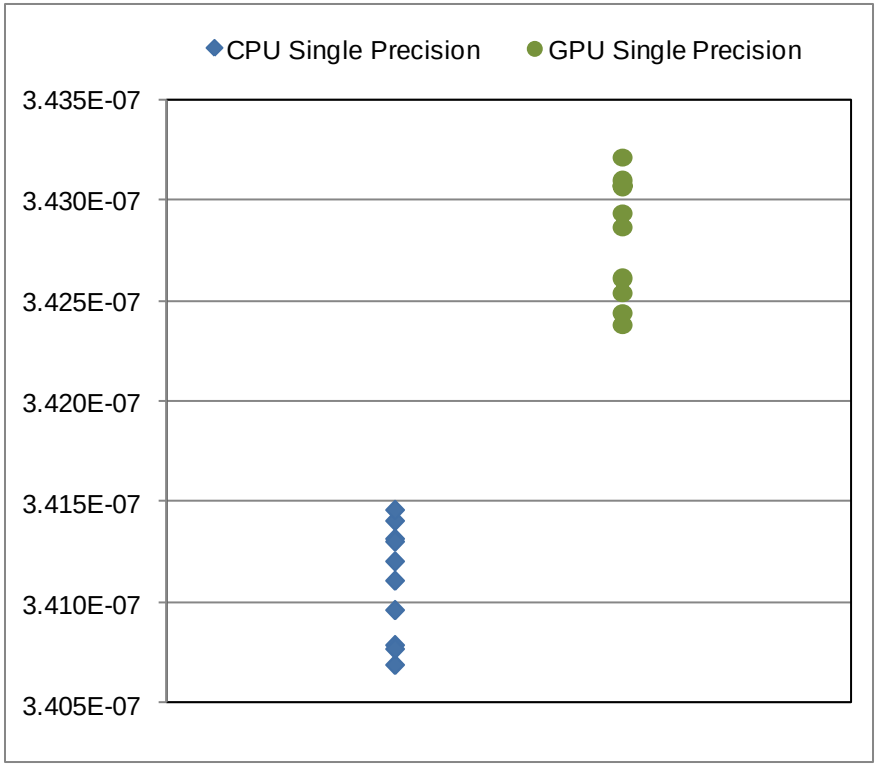
- 10x speedup
  - Offload BLAS to GPU, transfer all data
- 13x speedup
  - Offload BLAS, transfer only “dirty” matrices
- 19x speedup
  - Offload BLAS + additional functions, allowing data to remain on GPU longer

Whole-Code, Parallel Speedups

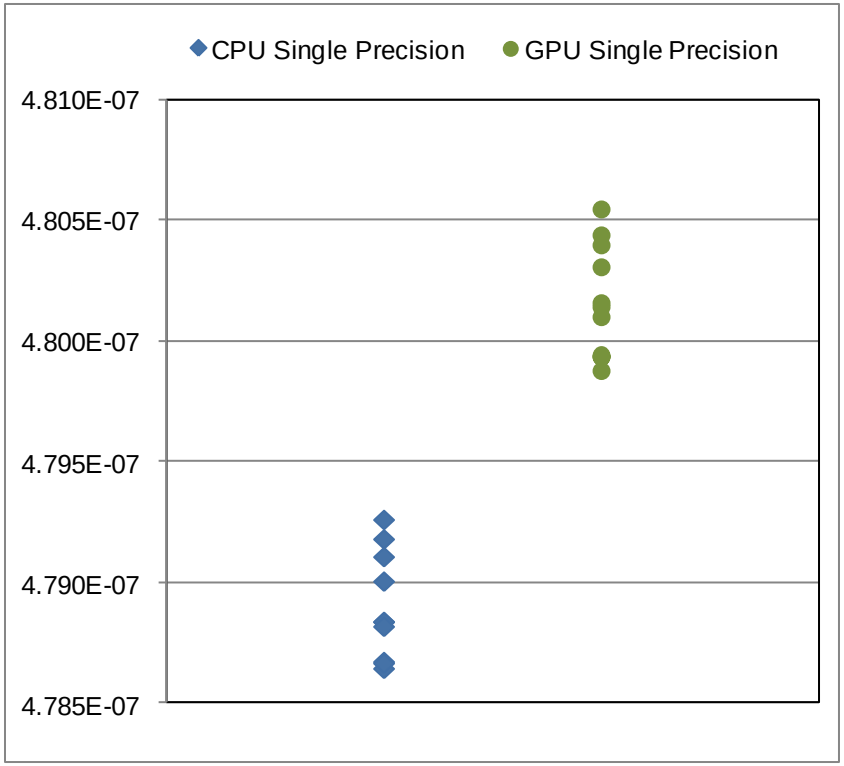
	Median Speedup	*Max Speedup*
Warmup	13.46x	17.28x
One QMC Step	13.32x	17.10x
One Measurement	1.00x	1.00x
One Iteration	4.96x	5.56x
Whole Code	**5.35x**	

- **16 2.2GHz Barcelona nodes**
- **32 MPI tasks**
- **32 8800GTX GPUs**

Accuracy of SGEMM



1024x1024

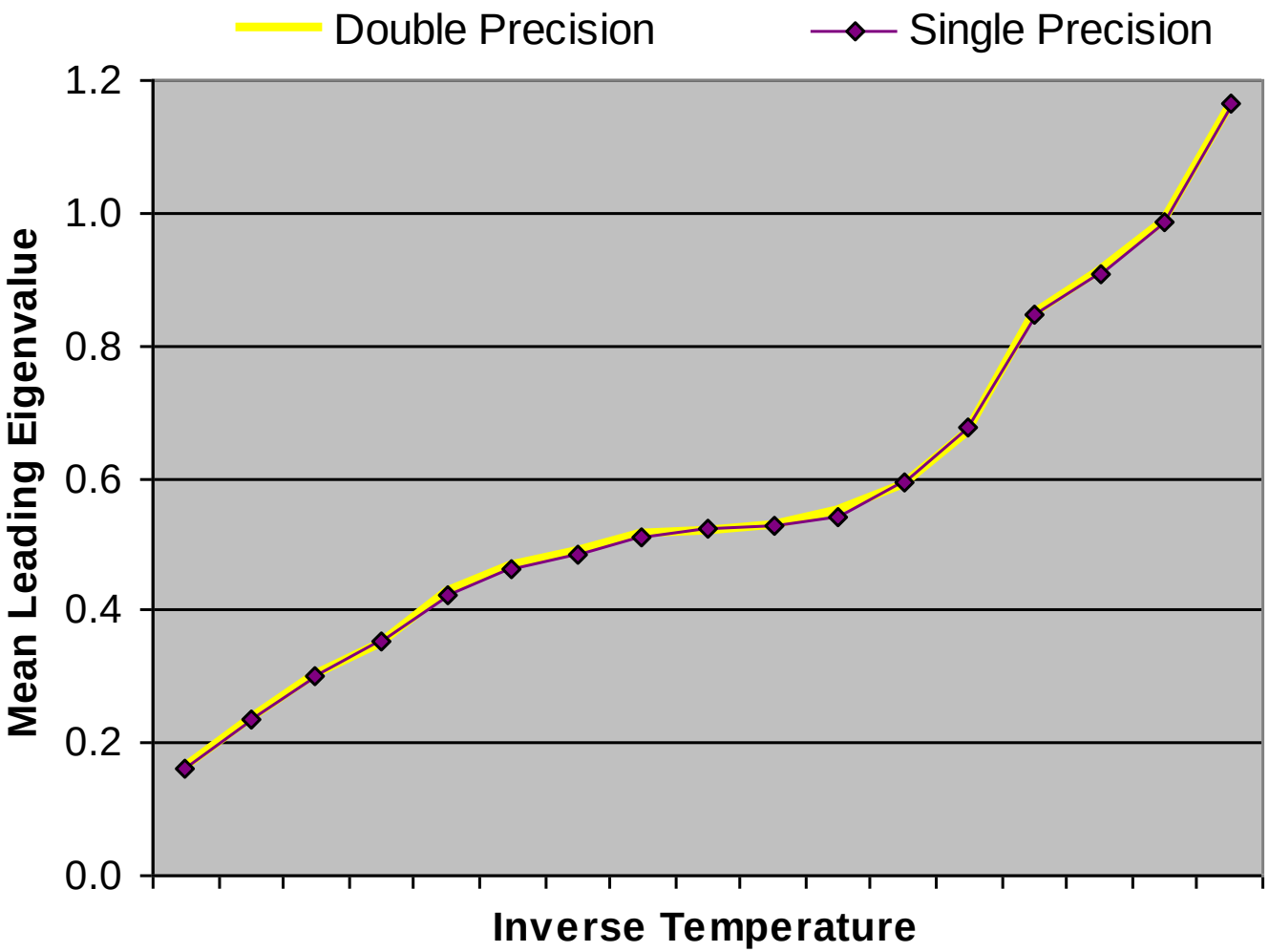


2048x2048

Accuracy: What's sufficient in QMC

- How to measure sufficient accuracy when the results depend on randomness?
  - run with same random seed?
    - doesn't help – decisions eventually change
  - record and replay decisions?
    - good, but probably best for short runs
  - run many times and check final distribution?
    - scientifically meaningful

Accuracy: Tolerance to Single Precision

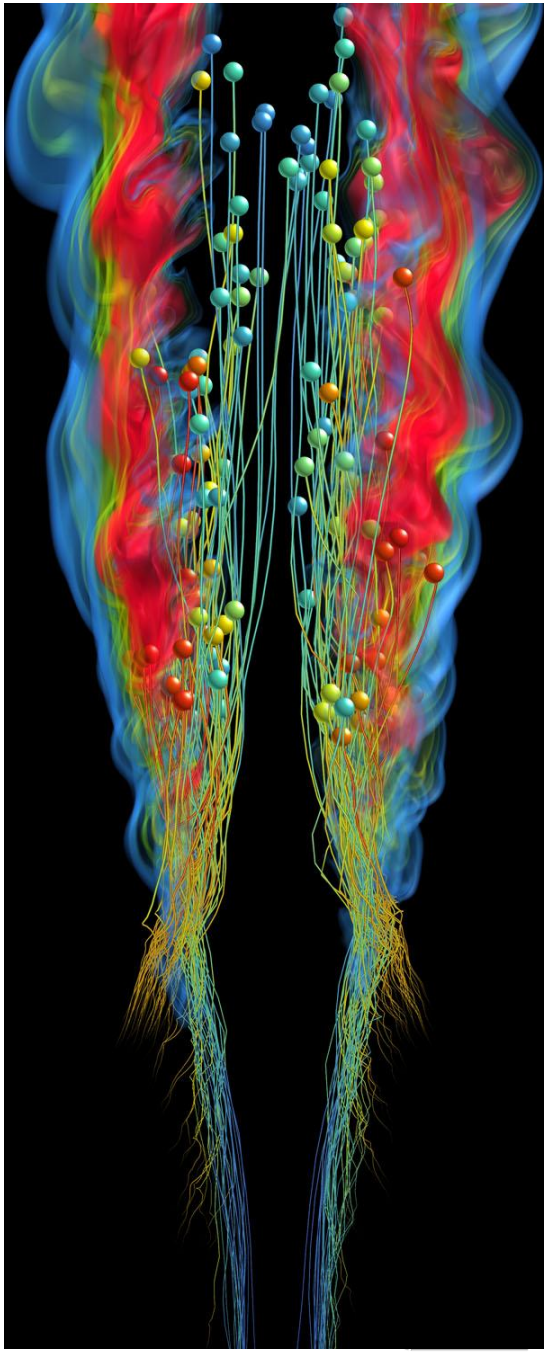


Outline

- Case Study #1:
  - DCA++: Quantum Monte Carlo, Materials Science
  - Jeremy Meredith (ORNL)
- Case Study #2:
  - S3D: Compressible Flow, Turbulent Combustion
  - Kyle Spafford (ORNL), Gregory Reutsch, Norbert Juffa (NVIDIA)
- Case Study #3
  - LAMMPS: Molecular Dynamics
  - Mike Brown, Paul Crozier, Steve Plimpton, Peng Wang

S3D

- S3D is a massively parallel direct numerical solver (DNS) for the full compressible Navier-Stokes, total energy, species and mass continuity equations coupled with detailed chemistry.
- US Department of Energy currently using it to simulate combustion of biofuels
- Highly scalable code (150k cores on Jaguar), written entirely in Fortran\MPI



Challenges: Tailoring S3D to the GPU

- **Main Tasks**
  - S3D has a huge code base written in Fortran, porting the entire application without proof-of-concept is infeasible—must choose a kernel for acceleration
  - Expose fine-grained data parallelism
  - Modify algorithms to exploit the GPUs memory hierarchy
  - Validate accuracy

- **Profiling Results:**
  - 50-60% of runtime spent on calculation of chemical rates

GetRates Kernel

- Based on temperature, pressure, and mass fraction, GetRates computes the speed at which chemical reactions are occurring
- Comprises roughly 45-60% of runtime, varying with grid size
- Operates on a regular, three dimensional grid

Parallelism in GetRates

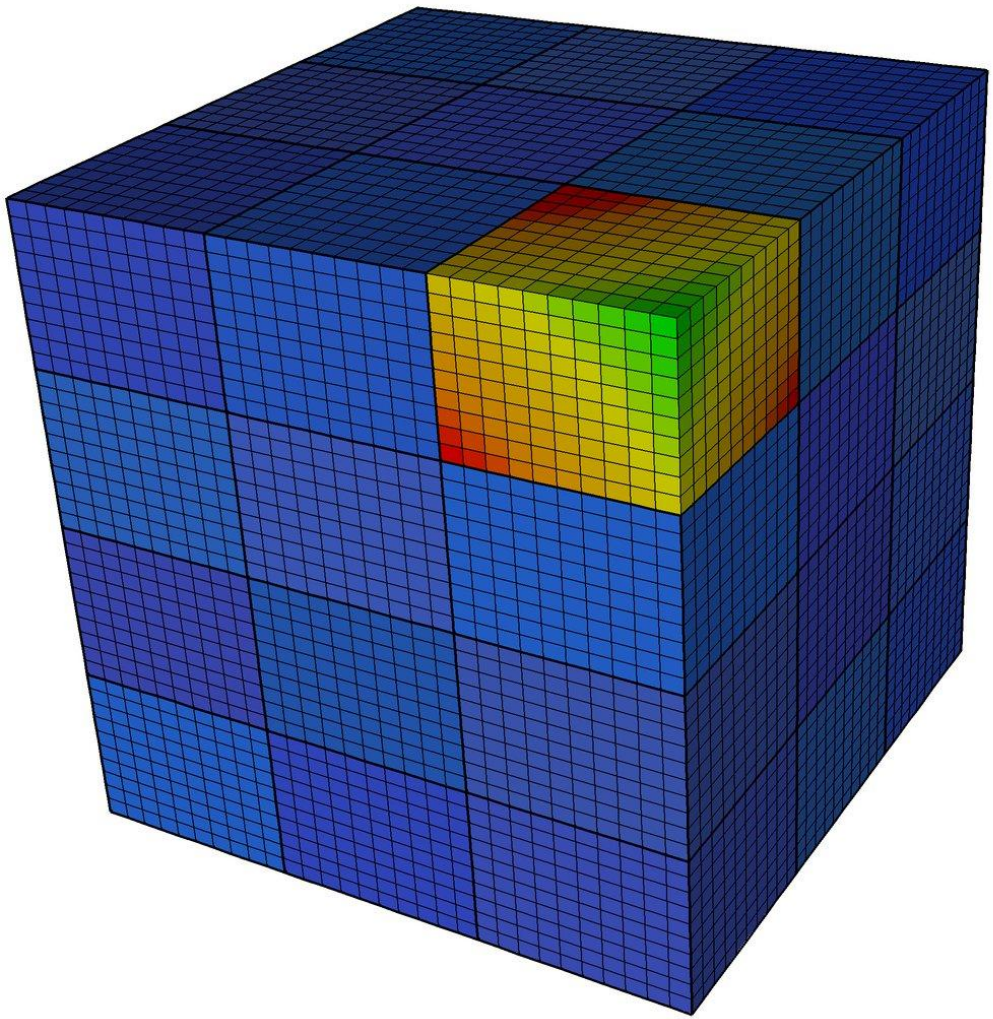
```
for x = 1 to cube_length
  for y = 1 to cube_length
    for z = 1 to cube_length
      for n = 1 to nspecies
          grid[x][y][z][n] =
          F(grid[x][y][z][1:nspecies])
```

- Outer 3 loops are parallel
- F has heavy computation (transcendentals, etc.)

Mapping to CUDA

- Regular 3D mesh decomposed spatially into blocks

- Each block is further divided into threads, which solve for an individual point in space

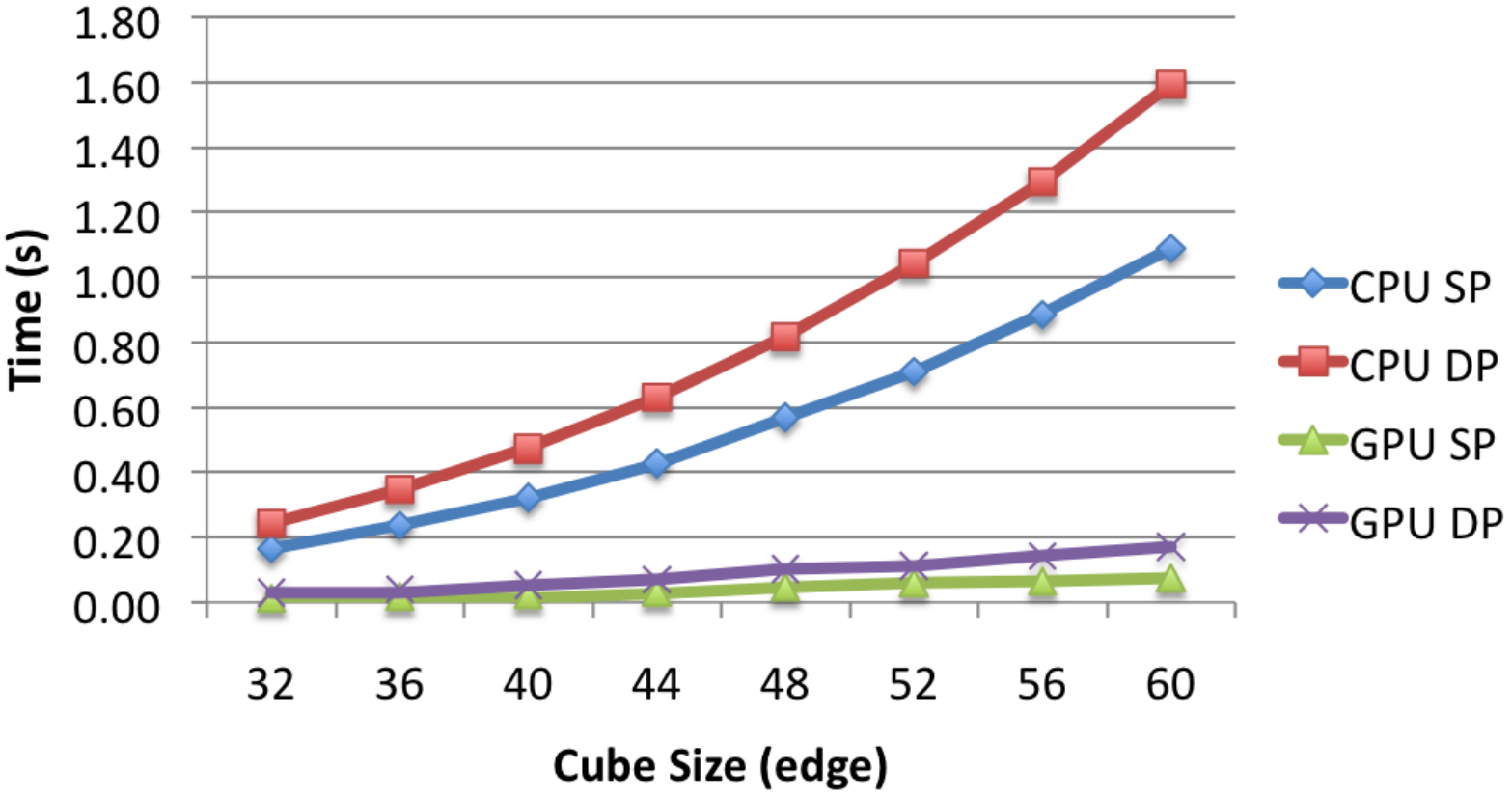


Exploiting the GPU Memory Hierarchy

- Coalesce reads/writes to global memory
  - Single most important factor for performance
- Interleave accesses to global memory with intense calculation
- Use shared memory when possible
- Constant memory: almost as fast as register access on cache hit

Kernel Speedups

SP: 13-14.5x DP: 8.1x-9.3x
Tesla C1060 vs Opteron (Barcelona)



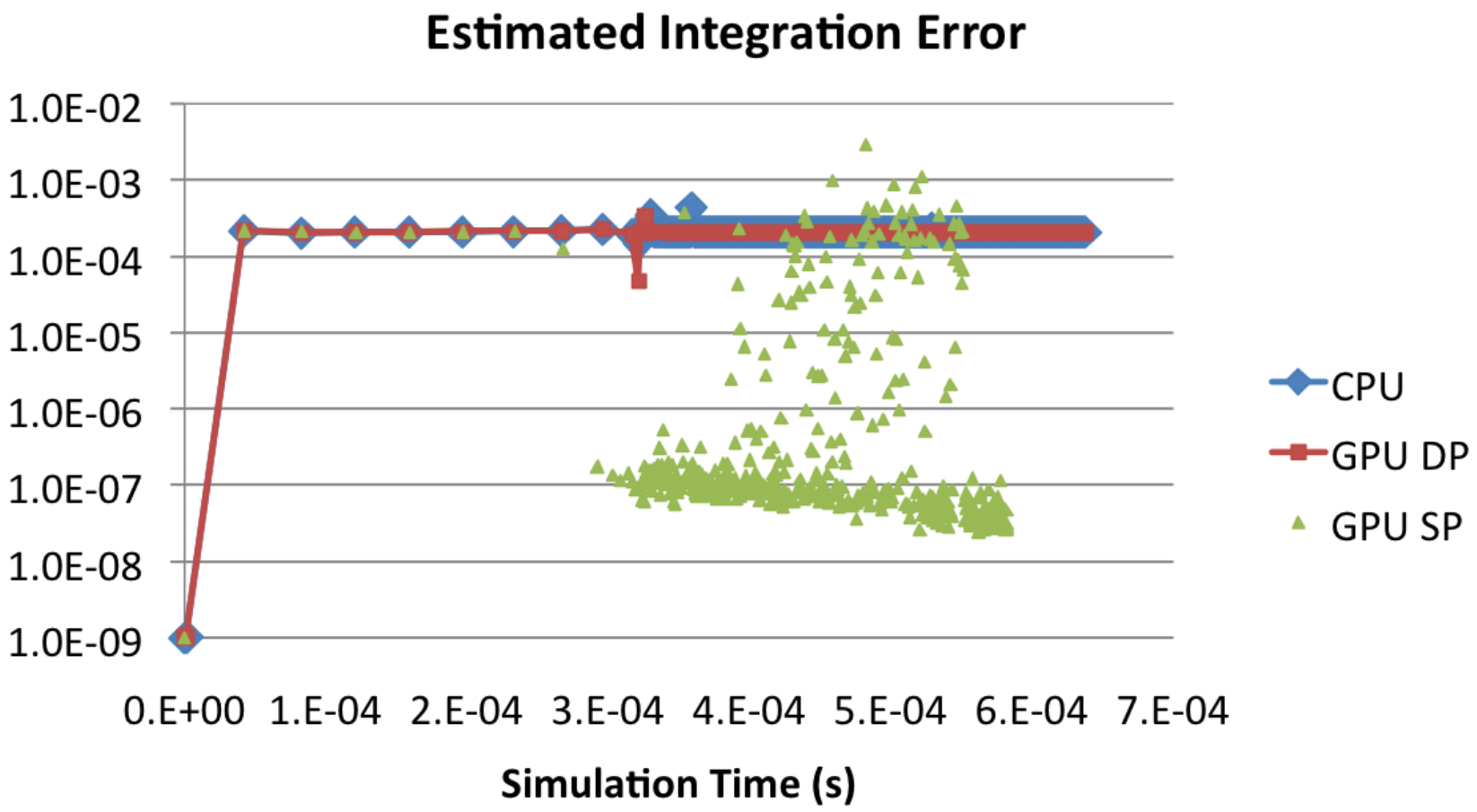
Exploiting the GPU Memory Hierarchy

- Some Limitations
  - Intermediate values (> 500 temporary variables per thread)
    - Where to put them?
    - Even at a small block size, still spill into global memory
  - PCIe bus
    - Data transfers can kill performance

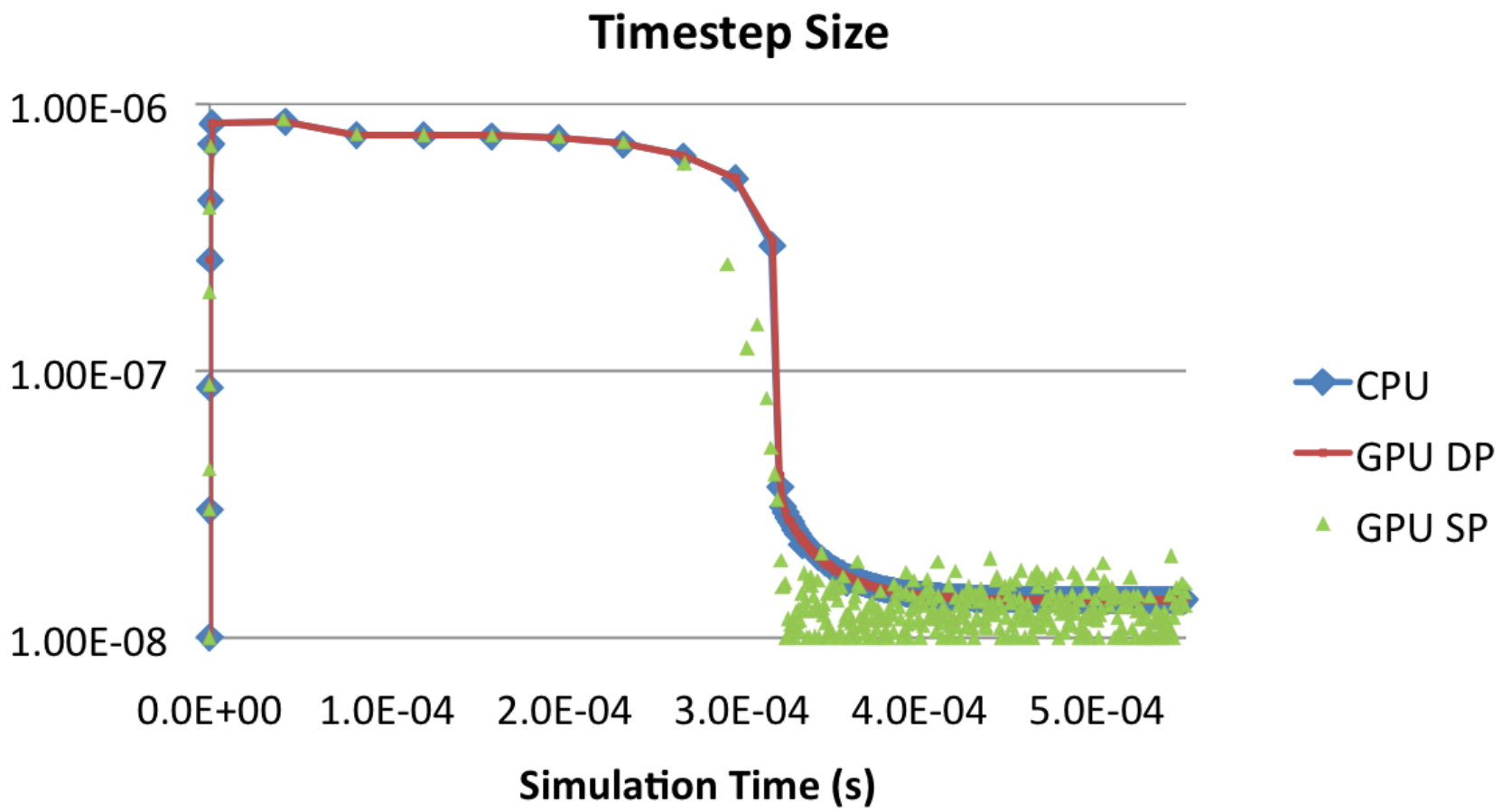
Accuracy vs. Performance Tradeoff

- Peak Flops
  - 933 SP vs 78 DP Tesla C1060
  - (much improved in Fermi)
- Double, Single, Fast-Single Intrinsics
  - __sinf(), sinf(), or sin() ?
- Timestep Size
  - Large timestep – faster but more susceptible to error
  - Automatically adjusted based on estimated error

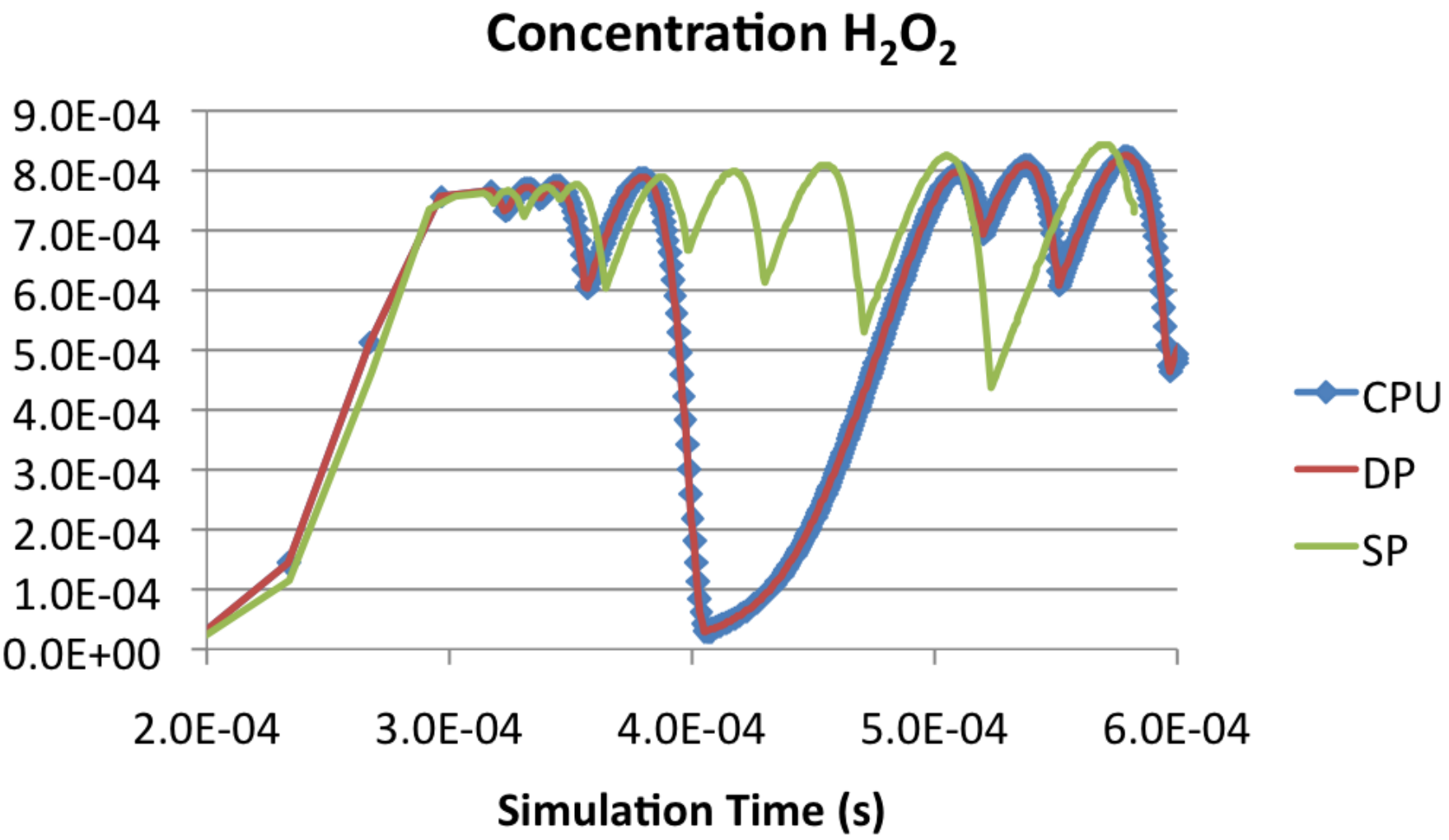
Estimated Error at Each Timestep



As error becomes erratic…



More Questions About Accuracy



Performance

Size	Kernel Speedup S	Kernel Speedup D	% of Total	Amdahl's Limit	Actual Speedup S	Actual Speedup D	Normalized Cost (ms) CPU	Normalized Cost (ms) GPU DP	Normalized Cost (ms) GPU SP
32	13.05x	8.17x	46.01%	1.82x	1.78x	1.61x	16.7	10.44	9.49
60	14.56x	9.32x	58.02%	2.38x	2.32x	2.27x	13.53	5.95	5.97

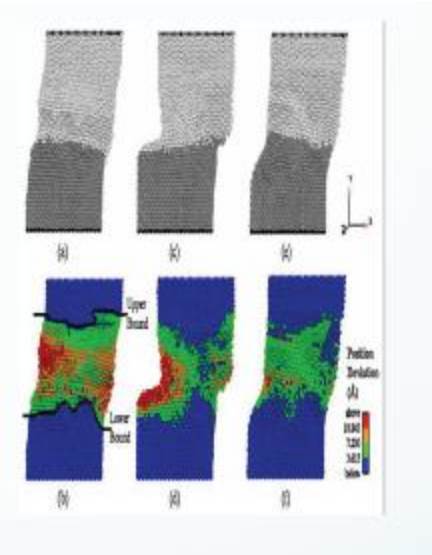
- **Amdahl's Limit – Theoretical speedup with an unlimited number of processors**
- **Normalized Cost – wallclock time to simulate a single point in space per nanosecond**
- **Scientists can now simulate larger domains, at greater resolution**

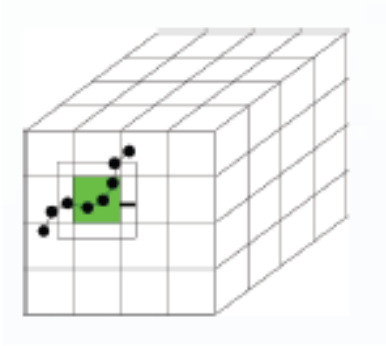
Outline

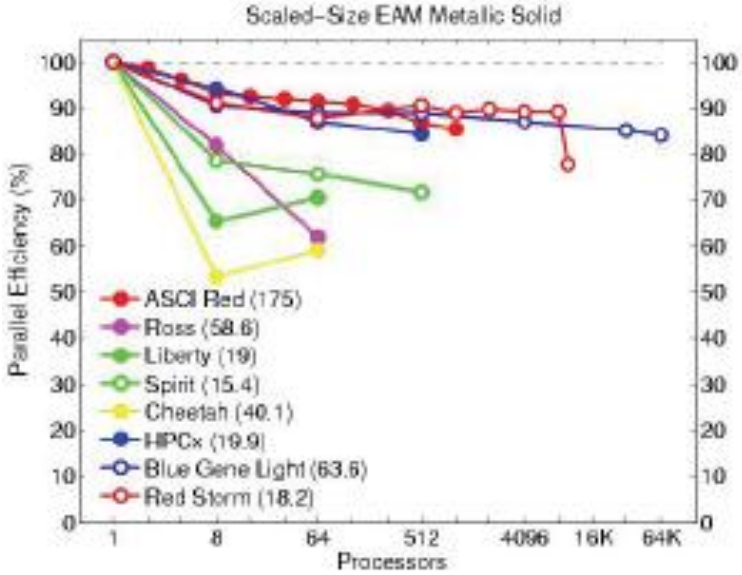
- Case Study #1:
  - DCA++: Quantum Monte Carlo, Materials Science
  - Jeremy Meredith (ORNL)
- Case Study #2:
  - S3D: Compressible Flow, Turbulent Combustion
  - Kyle Spafford (ORNL), Gregory Reutsch, Norbert Juffa (NVIDIA)
- Case Study #3
  - LAMMPS: Molecular Dynamics
  - Mike Brown, Paul Crozier, Steve Plimpton(ORNL/Sandia, Peng Wang (NV)

LAMMPS

- **Large-Scale Atomic/Molecular Massively Parallel Simulator**
  - **Classic MD Code, written in C++**
  - **175k+ lines of code**
  - **Spatial Decomposition**
  - **Distributed with MPI – has run on Jaguar, many DOE platforms**







LAMMPS – "Styles"

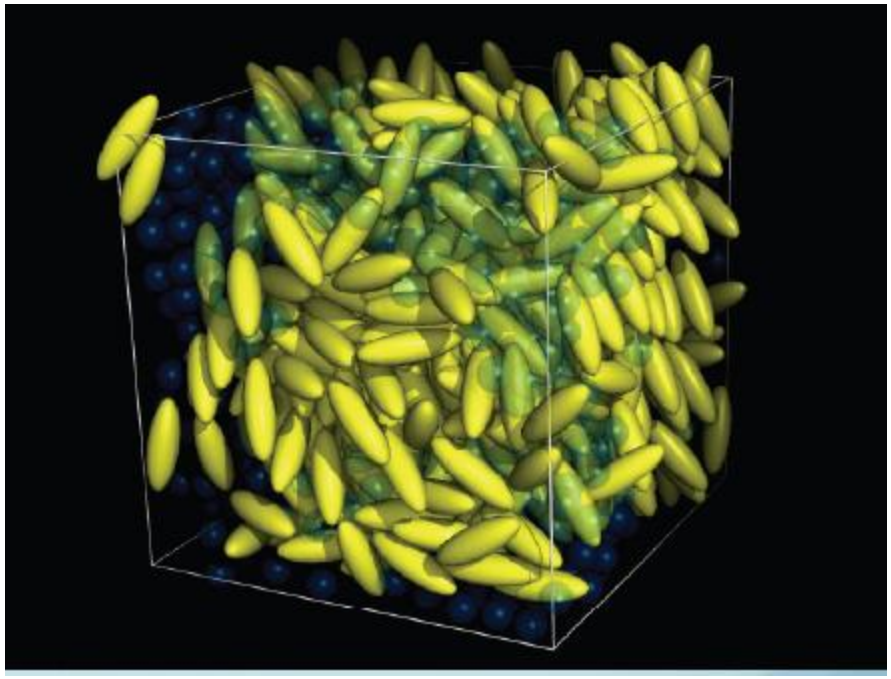
- LAMMPS is extensible via styles
  - These are basically plugins.
  - Add new potentials
  - Add new ranges, geometries, etc.

- LAMMPS now has "lammps-gpu"
  - Extends LAMMPs with CUDA-accelerated kernels
    - LJ potential
    - Neighbor List construction

Aspherical Particles

- Most existing methods handle only spherical particles
- Need a way to handle more realistic scenarios

- Scientists know how to handles these particles, but it's about 30x more intensive
- Instead of LJ, use a more elaborate potential, Gay-Berne



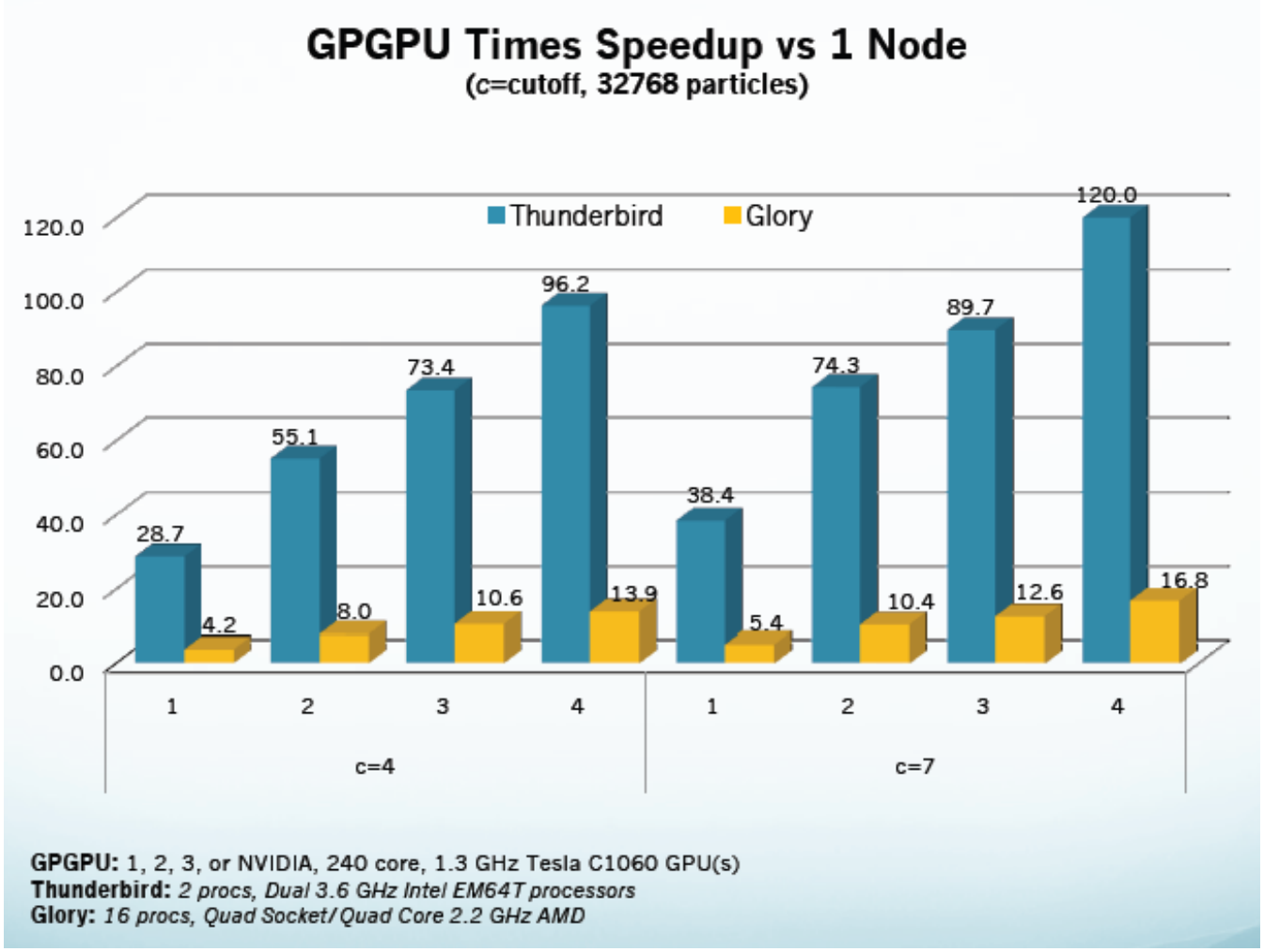
Gay-Berne Potential on GPUs

- Great candidate
  - High FP intensity
  - As “regular” memory access as LJ
  - Multi-precision friendly (single, double, mixed)

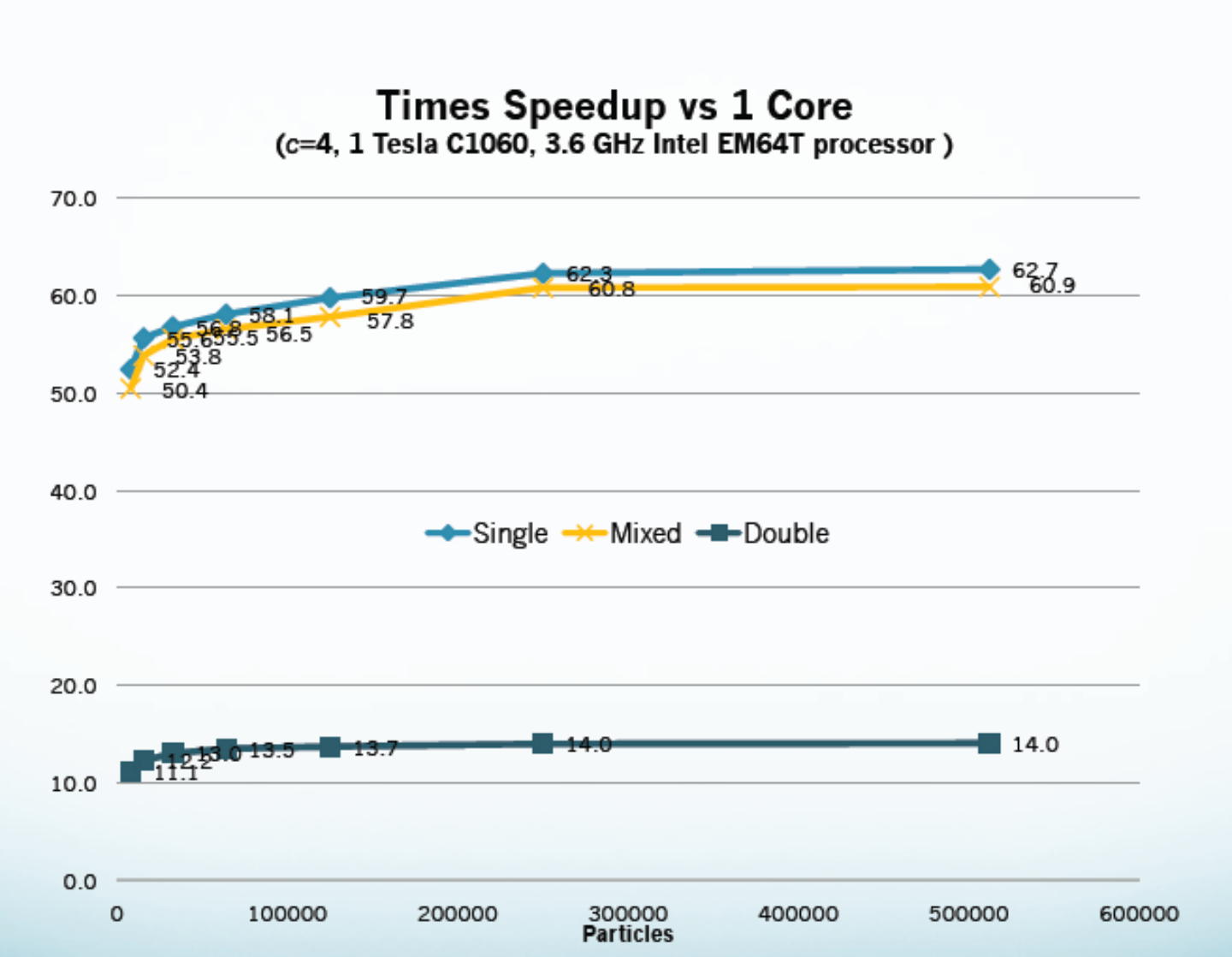
Basic Algorithm

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

Performance - Node



Mixed Precision



Coding Issues

- Difficult to keep everything in registers
  - In a real app, atoms have associated have more constants and other associated data than in benchmark codes
- Manual loop unrolling important for performance
- Huge number of implementations to draw from
  - Derived from the wide array of data structures used in MD

"Recipe" For Accelerating Applications

1. Profile your existing code to find where time is being spent
2. Choose kernels that represent a high fraction of runtime and map well to GPU hardware
3. Extract test harness and test problem
4. Implement and optimize kernels
5. Integrate with application, and minimize/overlap PCIe transfers
6. Check Accuracy
7. Repeat 2-6 as necessary

Thanks!

kys@ornl.gov

For More Information

- Keeneland: National Institute for Experimental Computing
  - http://keeneland.gatech.edu/
- "The Scalable HeterOgeneous Computing (SHOC) Benchmark Suite"
  - A. Danalis, G. Marin, C. McCurdy, J.S. Meredith, P.C. Roth, K. Spafford, V. Tipparaju, J.S. Vetter
  - Third Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU-3), 2010
  - http://ft.ornl.gov/doku/shoc/start
- "Accelerating S3D: A GPGPU Case Study"
  - K. Spafford, J. Meredith, J.S. Vetter, J. Chen, R. Grout, R. Sankaran
  - Seventh International Workshop on Algorithms, Models, and Tools for Parallel Computing on Heterogeneous Platforms (HeteroPar), 2009
- "Accuracy and Performance of Graphics Processors: A Quantum Monte Carlo Application Case Study"
  - J.S. Meredith, G. Alvarez, T.A. Maier, T.C. Schulthess, J.S. Vetter
  - Parallel Computing, Issue 35, Volume 3, March 2009