Irregular Memory Access
Optimizations for Gyrokinetic PIC
applications on multicore processors

Kamesh Madduri\textsuperscript{1} Sam Williams\textsuperscript{1}
Lenny Oliker\textsuperscript{1} John Shalf\textsuperscript{1} Erich Strohmaier\textsuperscript{1}
Jimmy Su\textsuperscript{2} Filip Blagojevic\textsuperscript{1} Kathy Yelick\textsuperscript{1,2}
Stephane Ethier\textsuperscript{3}

\textsuperscript{1} CRD/NERSC, Lawrence Berkeley National Laboratory
\textsuperscript{2} Computer Science Division, UC Berkeley
\textsuperscript{3} Princeton Plasma Physics Laboratory
Talk Outline

• Introduction
  – Particle-In-Cell simulations
  – GTC (Gyrokinetic Toroidal Code) Overview
  – The particle-mesh interpolation step

• Tuning parallel particle-mesh interpolation on multicore systems
  – Grid decomposition schemes
  – Synchronization strategies

• Performance results


• Future work
Particle-In-Cell (PIC) simulations

• Popular method for numerical simulation of many-body systems.
• Often implemented from first principles without the need of an approximate equation of state
• Applications: plasma modeling, Astrophysics

Grid/mesh overlaying particles to measure charge and current densities

Generic PIC Schematic
Plasma Microturbulence Simulations & GTC

- **ITER**: International collaboration to build the first fusion science experiment of a self-sustaining fusion reaction, the “burning plasma”.
- #1 priority in DoE’s science facility investment plan (2003)

Donut-shaped ("tokamak") reactor

- **GTC**: Code developed for Gyrokinetic particle-in-cell simulations of ion temperature gradient (ITG) turbulence.
- Scientists study effects of low-frequency microturbulence in fusion plasmas.
- Developed at the Princeton Plasma Physics laboratory.
- One of the numerical tools to predict efficiency of energy confinement in ITER.
Parallel Gyrokinetic Toroidal Code (GTC)

- Two levels of parallelization in the GTC MPI Implementation
  - 1D domain decomposition (along the torus, typically 64- to 128-way)
  - Particle decomposition in each toroidal domain
- Particle decomposition among MPI processes in each domain requires poloidal plane replication.

IBM Blue Gene/L run (2007) on 32,768 cores:
- # toroidal planes: 64
- 512-way particle decomposition
- 1024 particles/cell
- 32768 poloidal grid points

- Particle-Grid interpolation steps ("scatter" and "gather") constitute 80% of the execution time in simulations.
- "Scatter" step main source of inefficiency due to poor locality.
Grid memory accesses depend on the order in which particles are processed.

In a multithreaded implementation with a shared grid, multiple threads update grid locations in parallel.

The case of random particle positions and parallel updates is similar to the GUPS benchmark. However, implementations usually exploit the fact that PIC is a physical many-body simulation method.
Illustration of GTC “gyrokinetic” charge deposition step: Irregular memory accesses

1. Gyrating motion of a charged particle (ion) replaced by a moving ring.

2. Scatter step: ring approximated by four points, each assigned a quarter of the charge to deposit on neighboring grid coordinates.

3. The charge at 8-32 distinct grid points updated by each ion.

4. Gyrokinetic radius (Larmor radius) of a particle varies in simulation.
GTC PIC charge deposition kernel optimization on multicore systems

- **Problem:** Large memory footprint due to poloidal plane replication (in the 1D grid decomposition scheme) hinders GTC scaling for studying ITER-sized devices.
- **Problem:** Flat-MPI implementation does not effectively exploit multicore nodes.

**Our Contributions:**
- Memory-efficient multicore optimizations for the GTC charge deposition kernel.
- Exploration of several different grid decomposition and synchronization strategies.
- Experimental study and performance tuning on emerging cache-based multi-socket, multicore systems.
  - Intel Nehalem, AMD Barcelona, Sun Victoria Falls

Multithreaded parallelization

- Partition particle updates among threads.
- Thread parallelization complemented by grid decomposition and efficient synchronization, to reduce shared array update overhead and enhance locality.

For each particle do

#1. Get position
(5 loads from particle arrays, 3 loads from grid arrays, 2 stores, 1 sqrt, ~20 flops)

#2. Perform four-point gyrokinetic averaging
(8-32 updates to charge density grid array, 20 stores to auxiliary particle arrays, 40 loads from grid arrays, ~160 flops)
Shared grid vs. Full replication

**Shared**: One grid shared by all threads. Most memory-efficient approach.

- Requires synchronization for charge updates.
- Simplest Pthreads approach.

**Full replication**: Each thread maintains a private copy of the grid. Requires a P-way reduction to get final result.

- 1 shared
- P private copies (replicates)

- No synchronization required.
- Similar to reference GTC MPI implementation.
Grid partitioning

- Assume a radially-binned particle ordering (particle locality in $\psi$).
  - Motivates a radial partitioning of grid.
- The particle Larmor radius is less than the # of grid flux surfaces.
  - Motivates a radial grid partitioning + ghost flux surfaces to reduce shared grid updates.

Two-way reduction required.

Three-way reduction required.
Possible Synchronization Strategies

Coarse-grained locks
- Lock all $\theta$-$\zeta$ for a given $\psi$
  - # of lock calls per particle: 8
  - Contention dependent on # of particles per cell and particle distribution.
  - Requires radial binning of particles for ensuring low contention

Medium-grained locks
- Lock all $\zeta$ for a given $\theta$-$\psi$
  - # of lock calls per particle: 16
  - Contention dependent only on # of particles per cell.

Fine-grained locks
- Individually lock one grid point
  - # of lock calls per particle: 32

Atomics
- Atomically increment one grid point
  - # of lock calls per particle: 32

Other Performance Optimizations

• Fused the address calculation and charge deposition loops to improve temporal cache locality.
• Data structure changes to reduce misses due to cache line fragmentation.
• Fine partitioning of grid points among threads.
• NUMA-aware particle initialization.
• Affinity binding: Pinning threads to cores, static scheduling of particle charge decomposition work.
• Partial SIMDization of charge updates.
We assume a radially-binned particle distribution.
- Rate of change of particle position least in the radial direction.
- Periodic radial binning essential for efficient performance.

We experiment with several different grid sizes and particles-per-cell configurations.

### GTC problem configurations

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>mzeta</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>mpsi</td>
<td>90</td>
<td>192</td>
<td>384</td>
<td>768</td>
</tr>
<tr>
<td>mthetamax</td>
<td>640</td>
<td>1408</td>
<td>2816</td>
<td>5632</td>
</tr>
<tr>
<td>mgrid</td>
<td>32449</td>
<td>151161</td>
<td>602695</td>
<td>2406883</td>
</tr>
</tbody>
</table>

- Total Particles (micell=5)
  - 0.16M 0.76M 3M 12M
- Total Particles (micell=100)
  - 3M 15M 60M 241M

# of particles per cell

# of poloidal grid points
Architectural Details of Parallel Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Double-precision peak GFlop/s</th>
<th>DRAM Pin Bandwidth (GBytes/s)</th>
<th># of threads of execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Opteron 2356</td>
<td>73.6</td>
<td>21.33</td>
<td>8</td>
</tr>
<tr>
<td>Intel Gainestown X5550</td>
<td>85.3</td>
<td>51.2</td>
<td>16</td>
</tr>
<tr>
<td>Sun UltraSparc T5140</td>
<td>18.7</td>
<td>42.66 (r) + 21.33 (w)</td>
<td>128</td>
</tr>
</tbody>
</table>
Parallel performance with various optimizations

Problem size B (150K grid points), 5 particles per cell.

- Nehalem performance is ~ 2x of Barcelona.
- Pthreads implementations significantly faster than the MPI reference code.
- Partitioned grid approaches result in substantial performance improvement over shared grid variants.
- Atomic increments best-performing synchronization strategy.
- Thread pinning most beneficial on Nehalem.
Parallel Scaling

Problem size B (150K grid points), 5 particles per cell.

Strong scaling results (results normalized to single-threaded/single-process performance)

- SMT gives substantial benefit on Nehalem for the Pthreads variants.
- MPI implementation does not scale beyond 1 process/core.
- Sharp drop in Victoria Falls performance due to load imbalance from Larmor radius variation.
Parallel performance with problem size variation

Nehalem performance in GFlop/s

Performance drop on increasing grid size

Performance improvement on increasing # of particles per cell
Best “memory-efficient” Pthreads variant vs. reference MPI implementation

Barcelona

Nehalem

Niagara2

Ratio of the GFlop/s values of memory-efficient Pthreads variant & MPI implementation indicated.

- Substantial speedup for small particles-per-cell values on all processors.
- Reduction cost reduces as the number of particles per cell increases.
- Nehalem speedup due to efficient utilization of SMT.
- Niagara runs suffer due to load imbalance from avg. gyroradius variation.
Memory Footprint

Problem size B (150K grid points), 5 particles per cell.

Nehalem

Niagara2

- Memory requirements of the fully replicated version prohibitive!
- Naively using multicore is infeasible.
Conclusions

• Our memory-efficient charge deposition approaches (with grid decomposition and synchronization) enable solution to large problem instances on current multicore systems.

• For small problem sizes, we achieve a performance improvement of 1.5-4.4x over the optimized MPI implementation.

• Maximum Larmor radius value scales as \((\# \text{ of radial flux surfaces})/16\), leading to load imbalance in the partitioned grid approaches on the Niagara2.

• Pipelined atomic intrinsic support would improve performance of the memory-efficient Pthreads approach.
Future Work

• Explore middle ground between the *partitioned grid* and the *P-way reduction (full replication)* approaches.

• Fast implementation of a periodic *radial binning* strategy to enhance locality.

• Investigate charge deposition parallelization on local-store based multicores.

• Identify parameters to tune for, algorithmic variants to automatically choose.
Thank you!

Questions?