Autotuning Structured Grid Kernels

Kaushik Datta, Sam Williams, Vasily Volkov, Mark Murphy
{ kdatta, samw, volkov, mjmurphy } @eecs.berkeley.edu

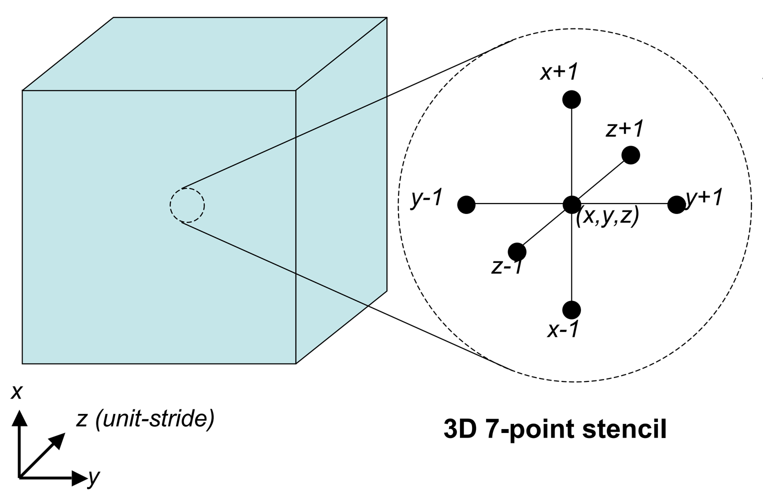
EECS Electrical Engineering and Computer Sciences
BERKELEY PAR LAB

P A R A L L E L   C O M P U T I N G   L A B O R A T O R Y

What are structured grids ?

Structured Grids

- Data is arranged in regular multidimensional grids (usually 2-4 dimensions)
- Computation is series of grid update steps
- Neighbor addressing is implicit based on each point's coordinates
- For a given point, a ***stencil*** is a pre-determined set of nearest neighbors (possibly including itself)
- A ***stencil code*** updates every point in a regular grid with a common stencil.




```
Next[x,y,z]=
  C0 * Current[x,y,z]+
  C1 *(Current[x-1,y,z]+
       Current[x+1,y,z]+
       Current[x,y-1,z]+
       Current[x,y+1,z]+
       Current[x,y,z-1]+
       Current[x,y,z+1]);
```

3D 7-point stencil | Inner Loop Pseudocode

- There are several structured grid kernels, including:
  - Basic Poisson solver (e.g., Jacobi and Gauss-Seidel iterations)
  - Multigrid
  - Mesh Refinement
  - Adaptive Mesh Refinement (AMR)
  - Lattice Methods (including LBMHD)

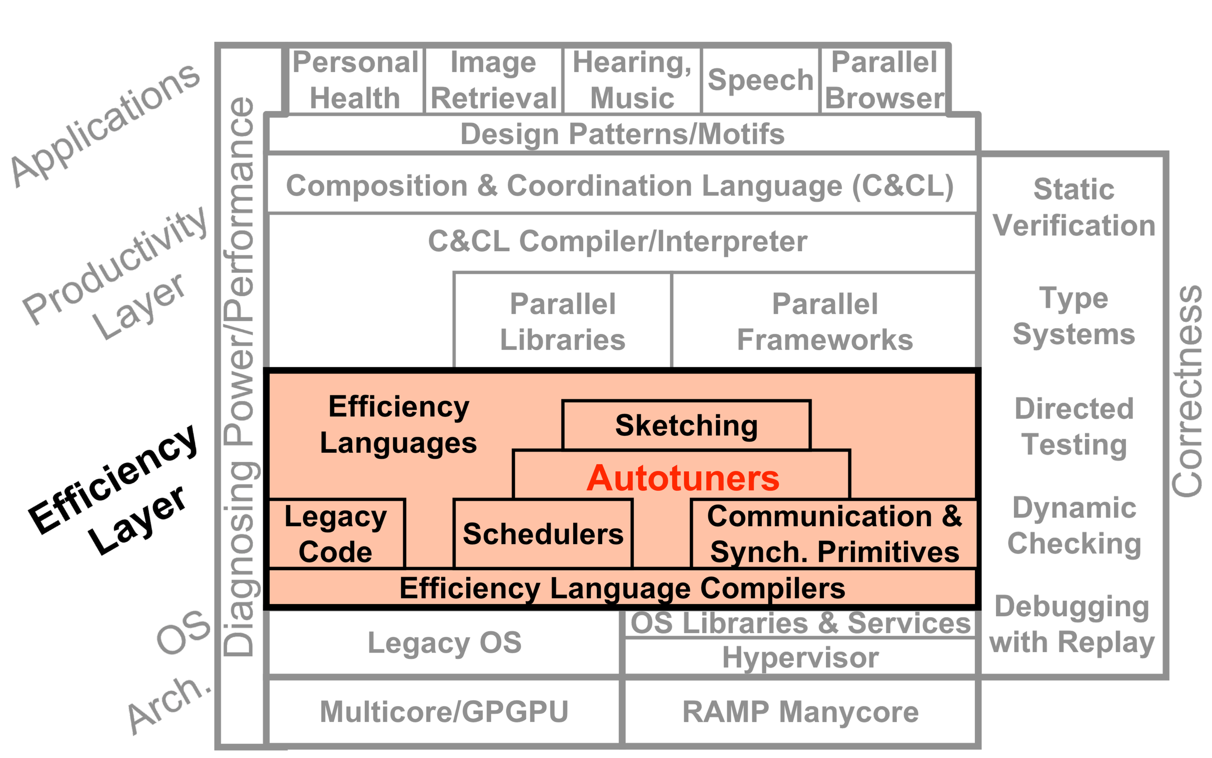
What is Autotuning?

Idea

- There are too many complex architectures with too many possible code transformations to optimize over.
- An optimization on one machine may slow another machine down.
- Need a general, automatic solution

Code Generators

- Kernel-specific
- Perl script generates 1000's of code variations
- Autotuner searches over all possible implementations (sometimes guided by a performance model to prune the space) to find the optimal configuration
- Optimizations included in this work:
  - **NUMA-Aware** collocates data with the threads processing it
  - **Array Padding** avoids conflicts in the L1/L2
  - **Thread/Cache Blocking** minimizes cache misses and memory traffic
  - **Vectorization** avoids rolling the TLB
  - **Unrolling/DLP** compensates for poor compilers
  - **SW Prefetching** attempts to hide L2 and DRAM latency
  - **SIMDization** compensates for poor compilers, and streaming stores minimize memory traffic



Autotuning Stencils (Cache-based Machines)

Paper Reference

K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, K.Yelick, "Stencil Computation Optimization and Autotuning on State-of-the-Art Multicore Architectures", Submitted to Supercomputing 2008.

Solving Poisson's Equation

- A common PDE arising in nature (e.g., electrostatics, heat diffusion) is Poisson's equation:

$$\nabla^2\varphi = f$$

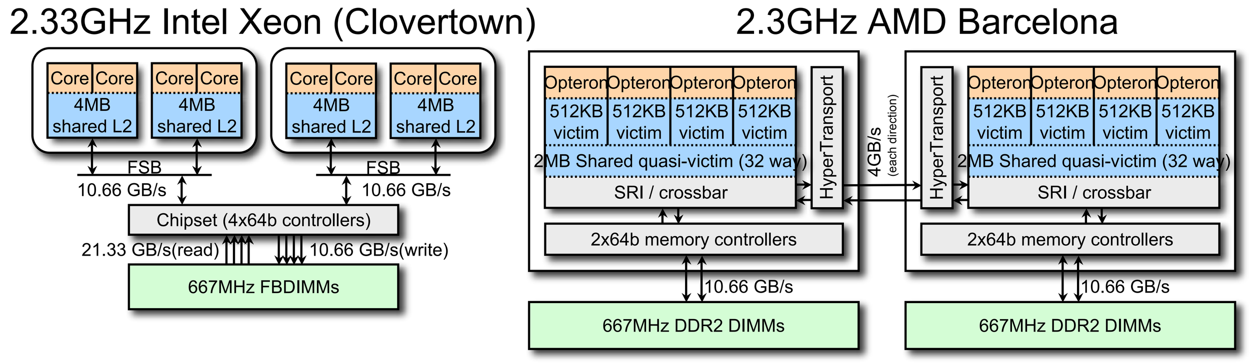
- By discretizing the volume and performing finite differences for the derivatives, the problem transforms into a stencil code

Stencil Code Description

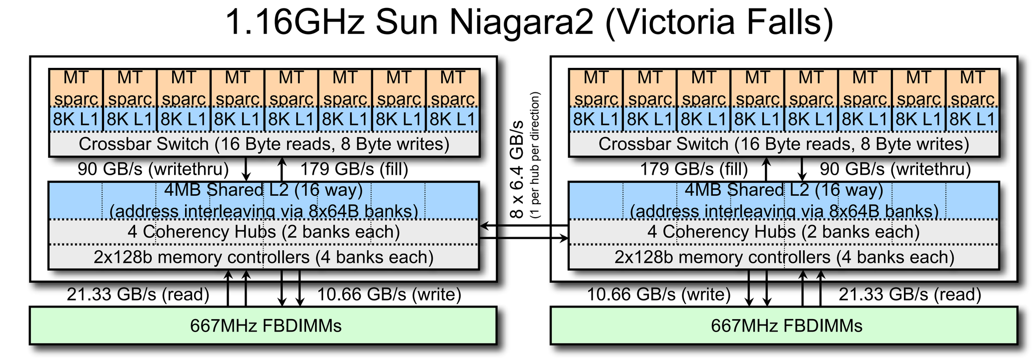
- We tuned an out-of-place (Jacobi) 7-point 3D stencil
- Ideally each update requires 8 flops and 16 Bytes (flop:byte of 0.5)
- Most cache-based machines will yield a flop:byte ratio of 0.33
- Ran a 256^3 (256 MB) problem

Architectures Evaluated

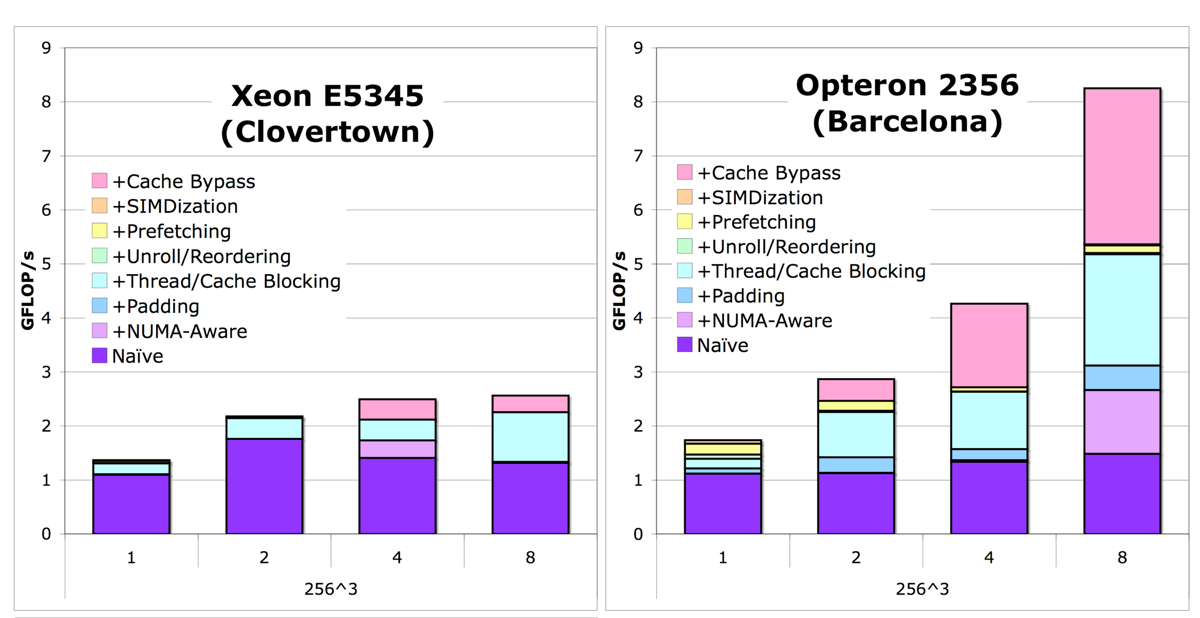
2.33GHz Intel Xeon (Clovertown) | 2.3GHz AMD Barcelona

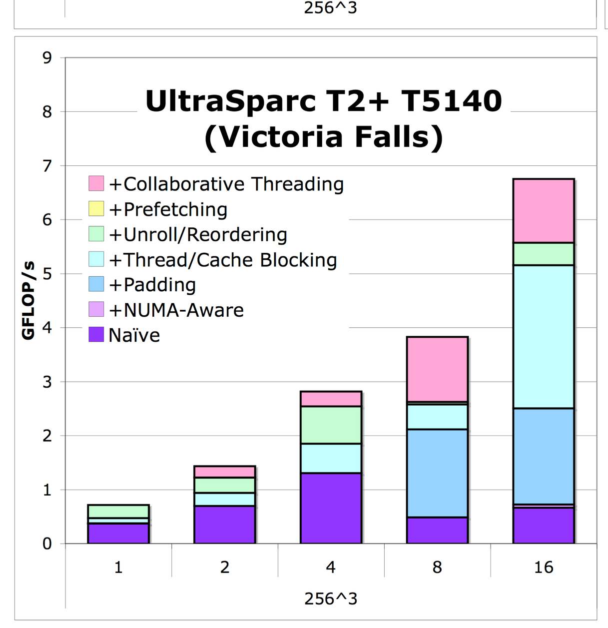


1.16GHz Sun Niagara2 (Victoria Falls)



Autotuning the Stencil Code





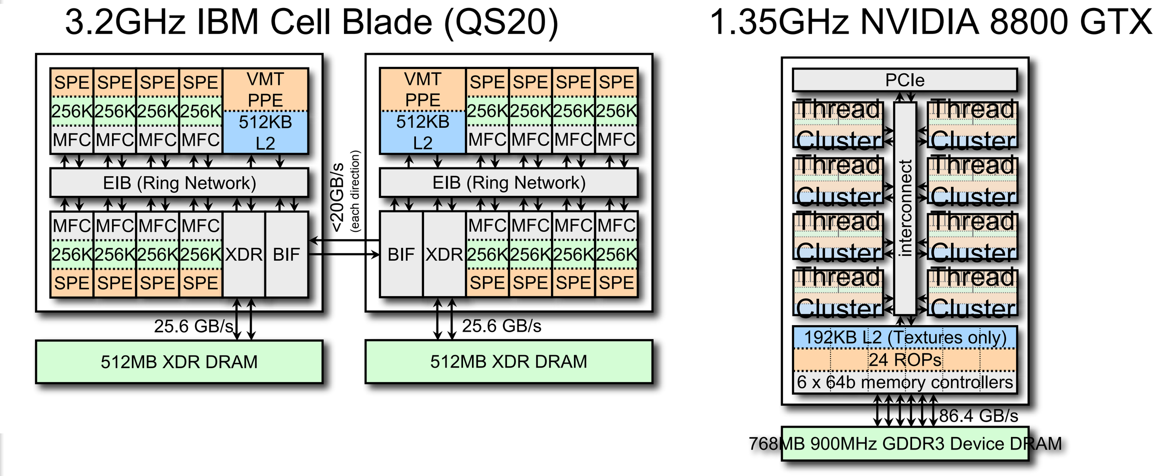
- **Clovertown**: single socket performance still limited by FSB bandwidth, dual socket perhaps by DRAM bandwidth
- **Opteron**: Cache bypass optimization was extremely useful (changed the flop:byte ratio from 0.33 to 0.50)
- **Niagara2**: benefited heavily from padding and thread/cache blocking

Tuning Stencils (All Architectures)

Additional Architectures Evaluated

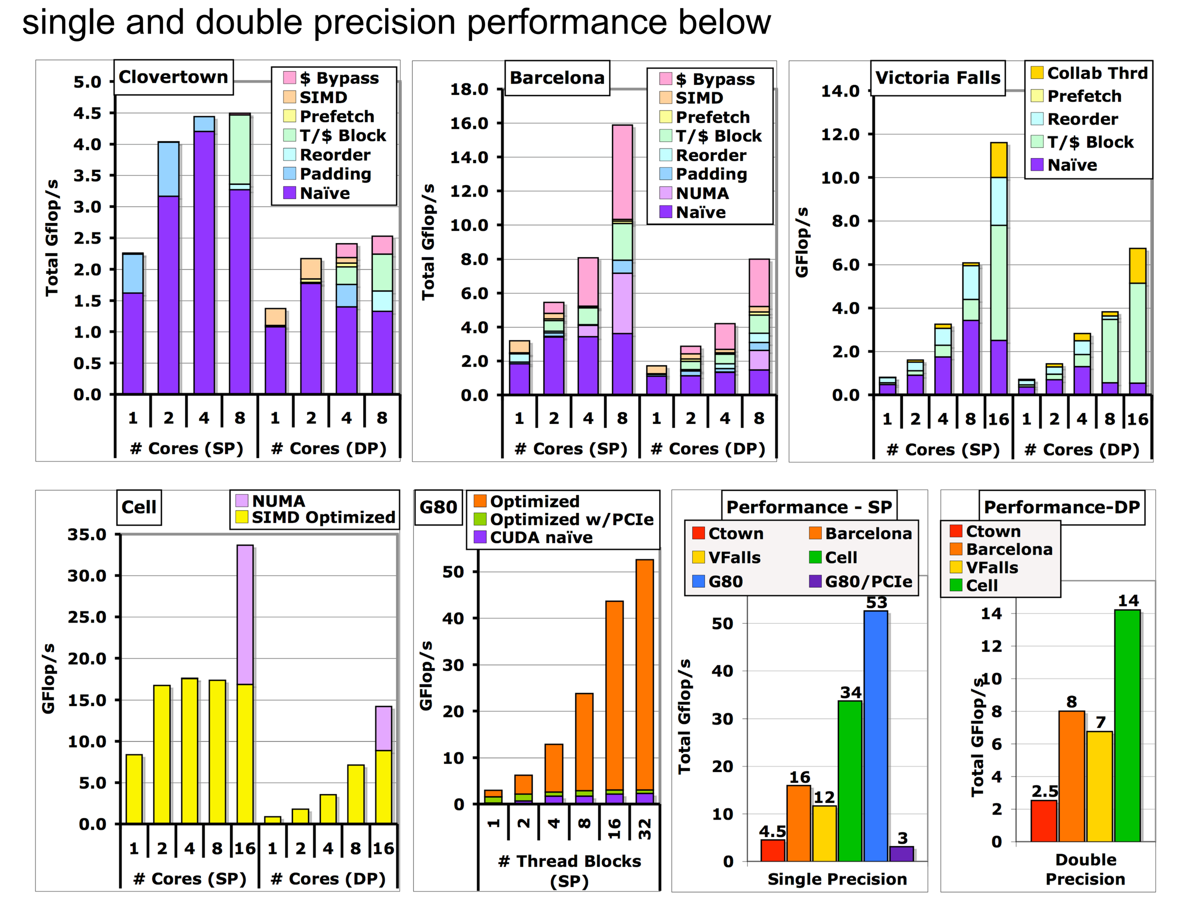
- In addition to the three cache-based architectures in the previous panel, we add two novel multicore architectures:

3.2GHz IBM Cell Blade (QS20) | 1.35GHz NVIDIA 8800 GTX



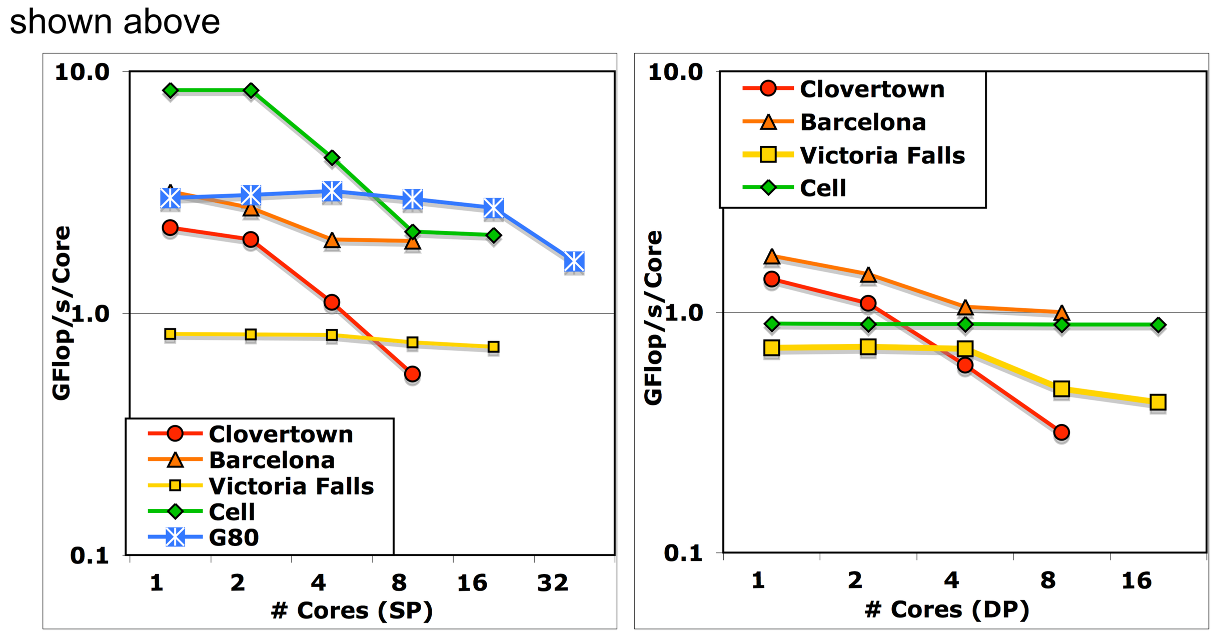
Tuned Stencil Code Performance

- Since the NVIDIA GPU only supports single precision, we compare both single and double precision performance below



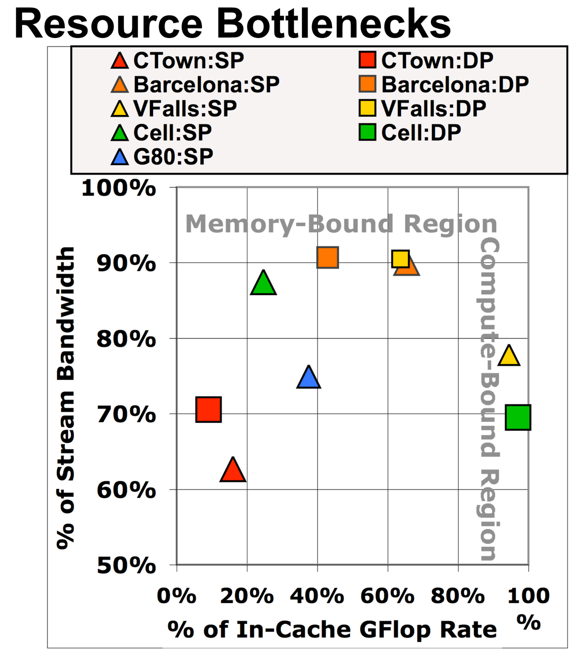
Core Scalability

- Tuned core scalability, below, is *much* better than untuned scalability, as shown above



Single Precision | Double Precision

Resource Bottlenecks



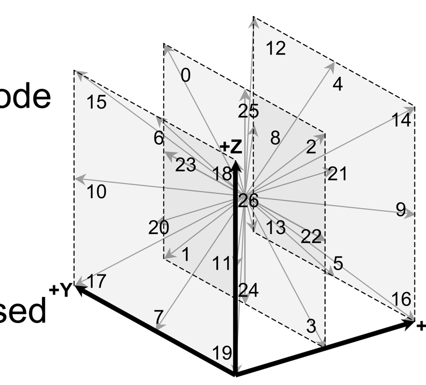
- This graph tries to determine whether we are exhausting either memory bandwidth or in-core performance
- Most architectures come close to maximizing one or both resources
- Clovertown performance is poor due to substantial cache coherence traffic on the FSB
- G80 is the other outlier- this will require further exploration

Autotuning Lattice Methods

Paper Reference

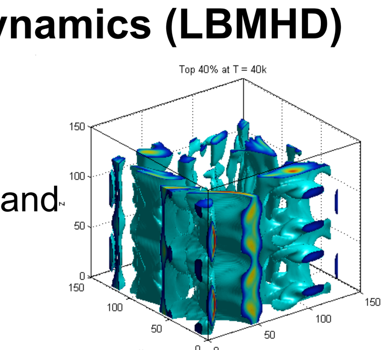
S. Williams, J. Carter, L. Oliker, J. Shalf, K. Yelick, "Lattice Boltzmann Simulation Optimization on Leading Multicore Platforms", International Parallel & Distributed Processing Symposium (IPDPS) 2008.

Lattice-Boltzmann Methods



- Out-of-place (Jacobi) style structured grid code
- Popular in CFD
- Simplified kinetic model that maintains the macroscopic quantities
- Distribution functions (e.g. 27 velocities per point in space) are used to reconstruct macroscopic quantities

Lattice-Boltzmann Magneto-hydrodynamics (LBMHD)



- Simulates plasma turbulence
- Couples CFD and Maxwell's Equations
- Thus it requires:
  - a Momentum (27 component) distribution and
  - a Magnetic (45 component) distribution
  - 7 macroscopic quantities (density, momentum, magnetic field)
- Two phases to the code:
  - ***collision()*** advances the grid one time step
  - ***stream()*** handles the boundary conditions (periodic for benchmark)
- Each cell update requires ~1300 flops and ~1200 bytes of data
- flop:byte ~ 1.0(ideal), ~0.66(cache-based machines)
- 2 Problem Sizes: 64^3(330MB), and 128^3(2.5GB)
- Currently utilize Structure-of-Arrays data layout to maximize locality

Autotuning LBMHD

- Autotuning dramatically improved performance on the Opteron (4x)
- Became important when the problem could no longer be mapped with Niagara2's 4MB pages
- Although prefetching showed little benefit, SIMD and streaming stores helped significantly.
- Cell was not autotuned, and only *collision()* was implemented