Autotuning Structured Grid Kernels

EECS
Electrical Engineering and Computer Sciences

Kaushik Datta, Sam Williams, Shoaib Kamil

{ kdatta, samw, skamil } @eecs.berkeley.edu

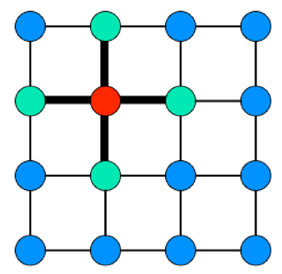
BERKELEY PAR LAB

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

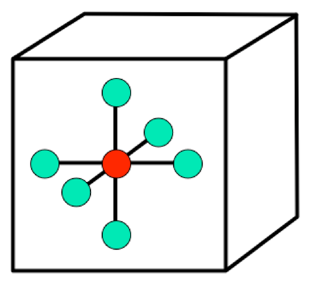
What are structured grids ?

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.



5-point 2D Stencil



7-point 3D Stencil

- 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

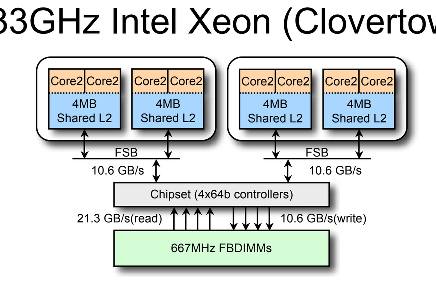
Code Generators

- 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:

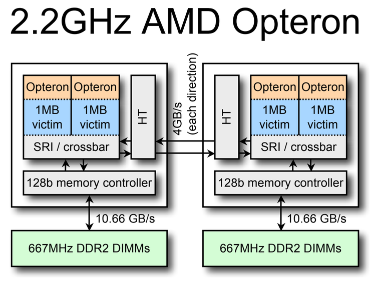
Array Padding	avoids conflicts in the L1/L2
Vectorization	avoids rolling the TLB
Unrolling/DLP	compensates for poor compilers
SW Prefetching	attempts to hide L2 and DRAM latency
SIMDization	compensates for poor compilers, and streaming stores minimize memory traffic

Architectures Evaluated

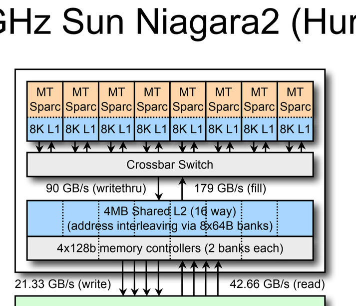
2.33GHz Intel Xeon (Clovertown)



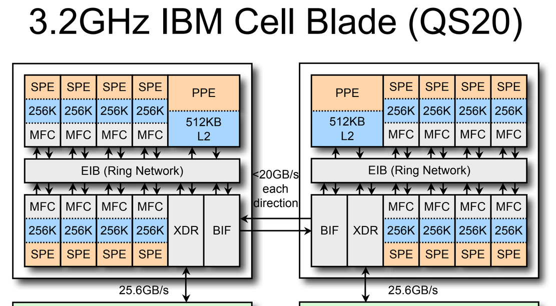
2.2GHz AMD Opteron



1.4GHz Sun Niagara2 (Huron)



3.2GHz IBM Cell Blade (QS20)



Serial Stencil Algorithms

Paper Reference

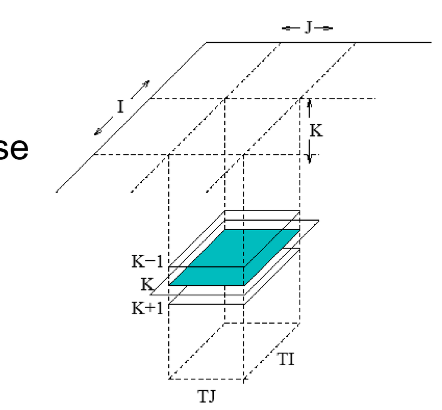
K. Datta, S. Kamil, S. Williams, L. Oliker, J. Shalf, K. Yelick, "Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors", To Appear, SIREV 2008.

Introduction

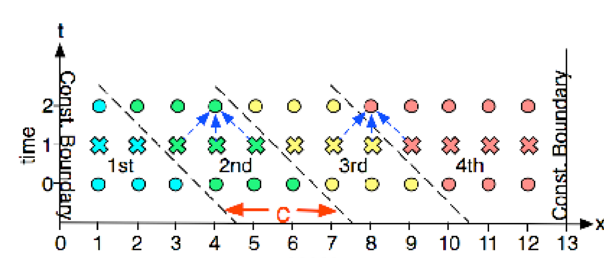
- Investigation of stencil optimizations for a simple 7-point heat equation
- Constructed memory models for single-timestep cache blocking and time skewed blocking

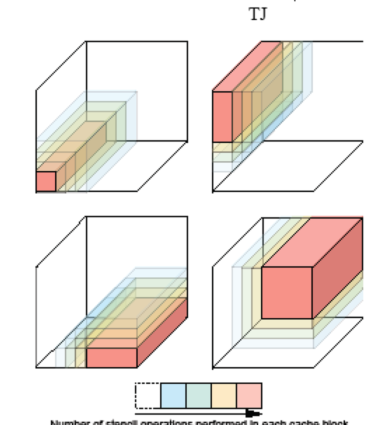
Single-Timestep Cache Blocking

- 2D Cache Blocking algorithm in 3D with reuse only in space
- Largest-stride dimension is unblocked
- Early work (on Pentium III-class machines) by Rivera et al. showed performance Improvements



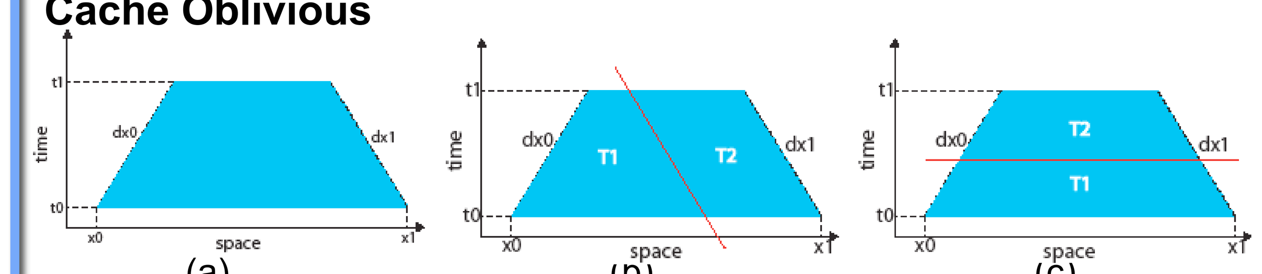
Multiple-Iteration Time Skewing





- Extends cache blocking to reuse points over multiple sweeps of the grid
- Diagram above left shows the shape of each block and the order they are executed in 1D. Above right shows a 3D version of the block execution order.
- Block shape takes into account the inter-point dependencies of the stencil

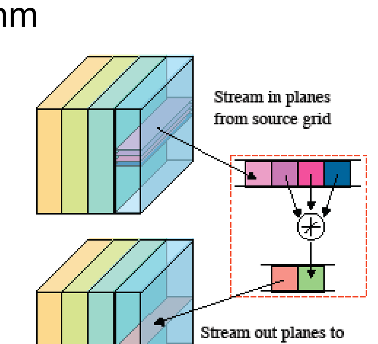
Cache Oblivious



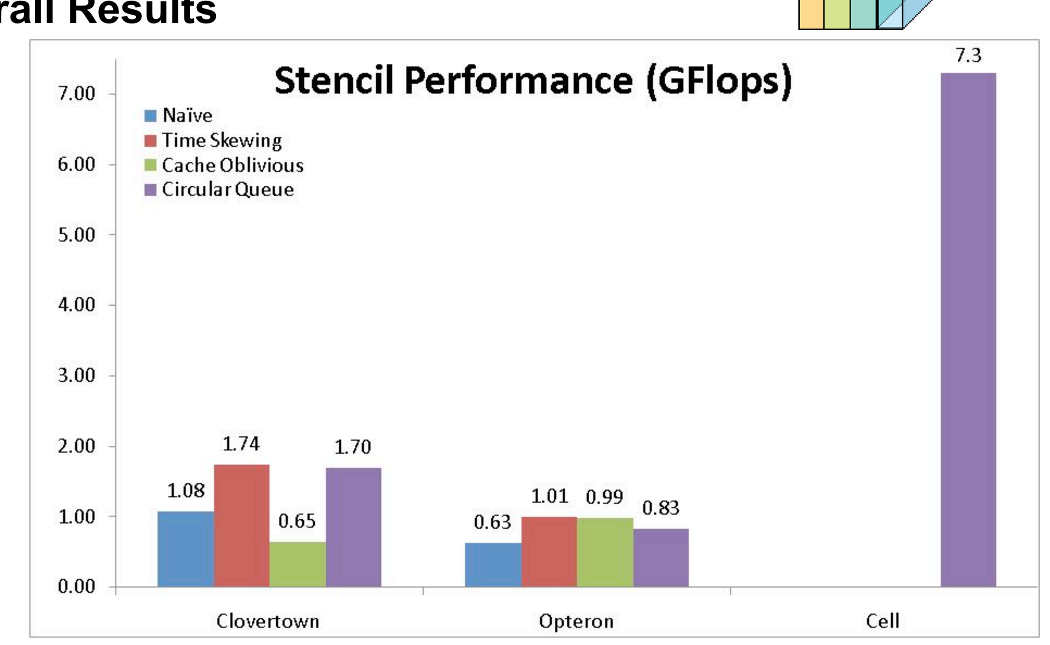
- Recursive algorithm that does not use cache size as a parameter
- Cuts a spacetime trapezoid such as the 1D example in Figure (a) in either time (c) or space (b), preserving point dependencies
- Extensive effort by our group to optimize this algorithm

Circular Queue

- Designed to pipeline planes of a stencil into a local store or cache and perform stencil operations
- Originally for Cell local stores using DMA operations



Overall Results



Stencil Probe Release

- A self-contained benchmark suite with implementations of all the above algorithms
- Targeted release date: Jan 21, 2008

Autotuning Stencils

Solving Poisson's Equation

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

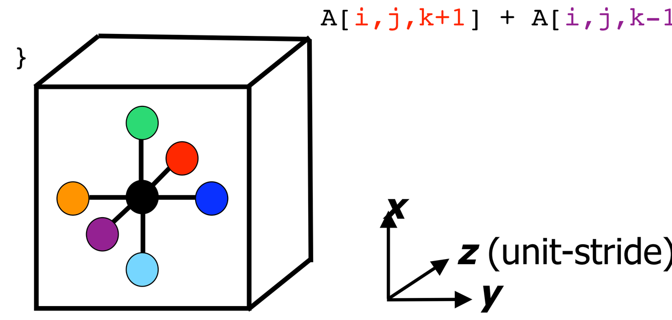
$$\nabla^2\varphi = f$$

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

Stencil Code Description

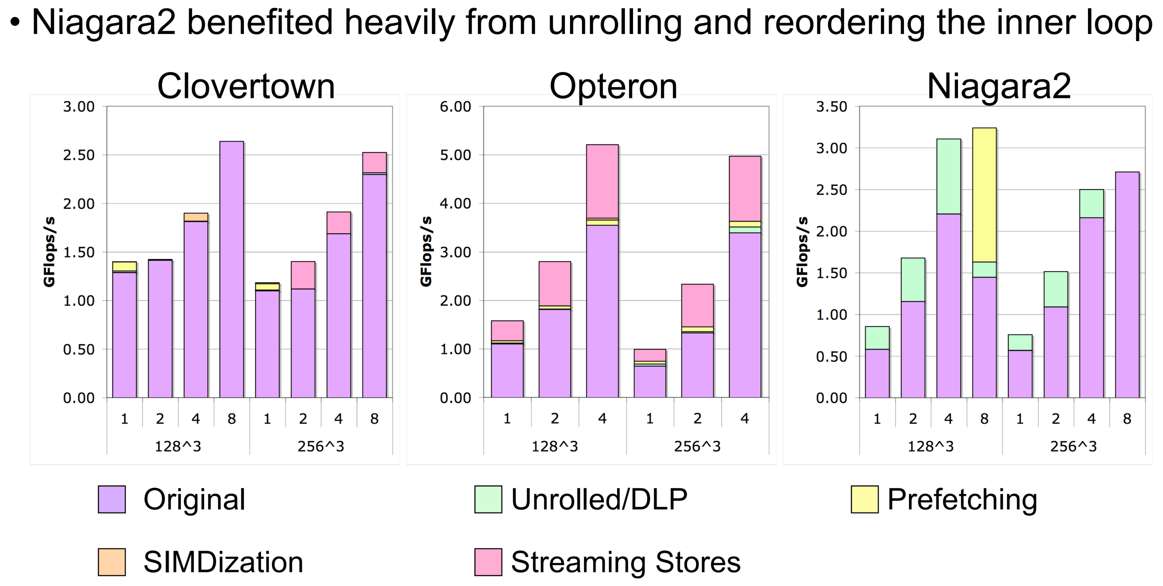
- We tuned an out-of-place (Jacobi) 7-point 3D stencil
- Ideally each update requires 8 flops and 16 Bytes (flop:byte of 0.5)
- Most cache-based machines will yield a flop:byte ratio of 0.33
- 2 Problem Sizes: 128^3 (32 MB) and 256^3 (256 MB)
- Some pseudo-code:

```
void stencil3d(double A[], double B[], int nx, int ny, int nz) {
            for all i in x-dim {
               for all j in y-dim {
                  for all k in z-dim {
                     B[center] = S0* A[i,j,k] +
                                 S1*(A[i+1,j,k] + A[i-1,j,k] +
                                     A[i,j+1,k] + A[i,j-1,k] +
                                     A[i,j,k+1] + A[i,j,k-1]);
                  }
               }
            }
}
```



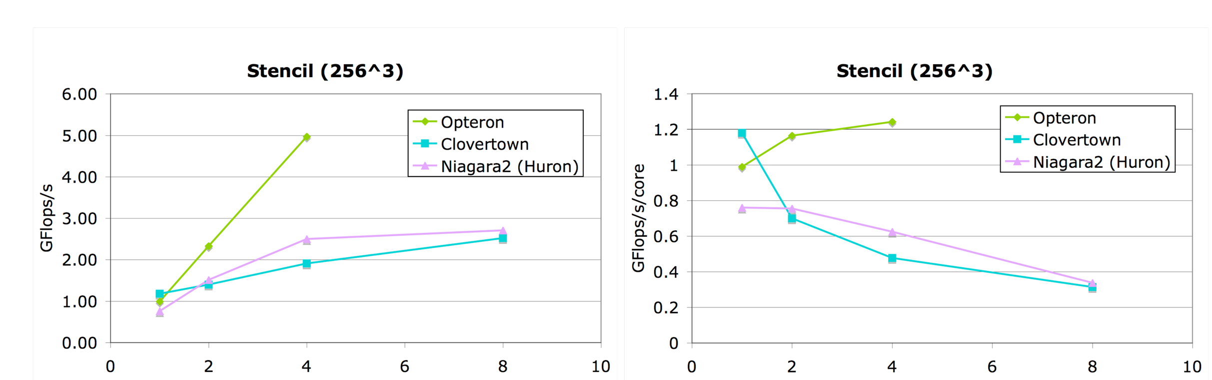
Autotuning the Stencil Code

- Streaming Store optimization was extremely useful on the Opteron (changed the flop:byte ratio from 0.33 to 0.50)
- Clovertown single socket performance still limited by FSB bandwidth, two socket perhaps by DRAM bandwidth
- Niagara2 benefited heavily from unrolling and reordering the inner loop



Scalability and Performance Comparison

- Clovertown has problems with both multicore and multisocket scaling
- Opteron shows superlinear speedup (likely cache effects)
- Niagara2 performance drops off when all 64 threads (8 cores) are used (still under investigation)
- Opteron performs the best



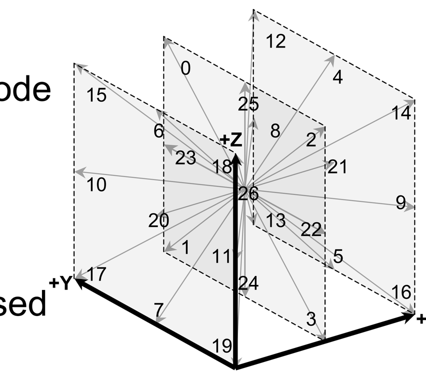
Autotuning Lattice Methods

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) (to appear), 2008.

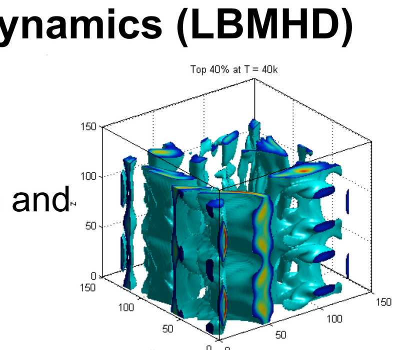
Lattice-Boltzmann Methods

- 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



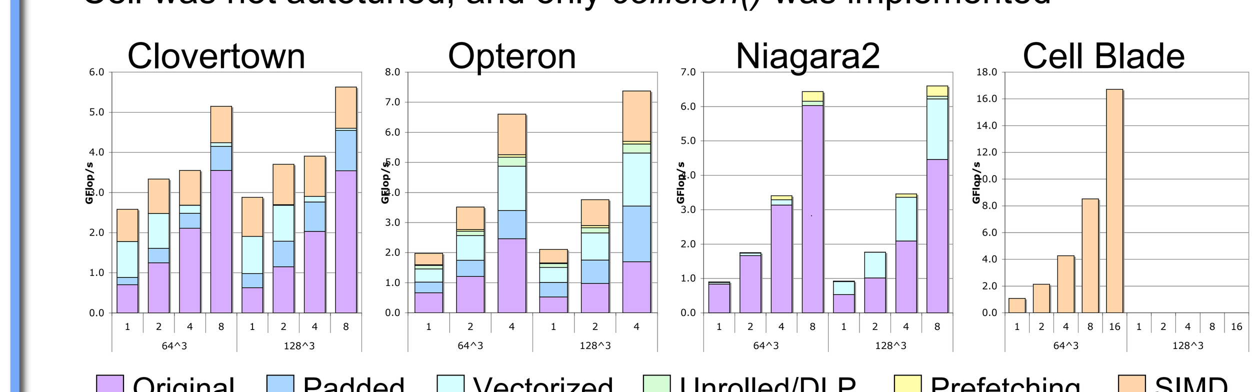
Lattice-Boltzmann Magneto-hydrodynamics (LBMHD)

- 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



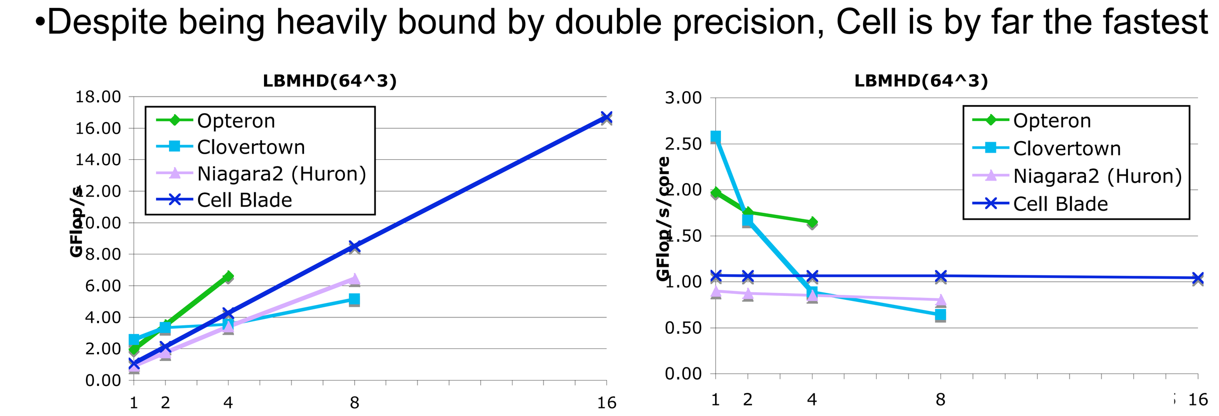
Autotuning LBMHD

- 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



Scalability and Performance Comparison

- Clovertown has problems with both multicore and multisocket scaling
- Niagara2 delivered performance between Opteron and Clovertown
- Despite being heavily bound by double precision, Cell is by far the fastest



System Power Efficiency

- Used a digital power meter to measure sustained system power
- Niagara2 system required 50% more power than other systems