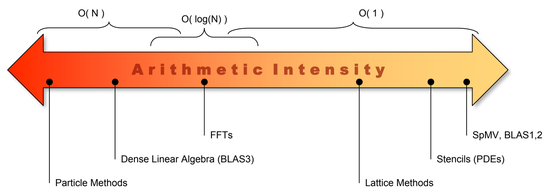
Autotuning Scientific Kernels on Multicore Systems

Sam Williams*§, Jonathan Carter*, James Demmel§, Leonid Oliker*§,
David Patterson*§, John Shalf*, Katherine Yelick*§, Richard Vuduc‡

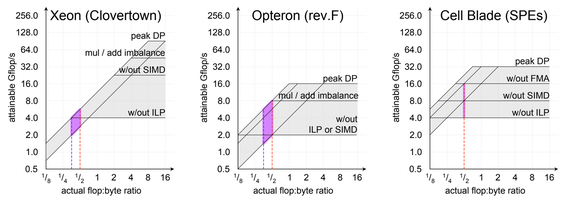
*Lawrence Berkeley National Laboratory, §UC Berkeley, ‡Georgia Institute of Technology

Overview

Mulicore architecture are paradigm for next decade
- Offer best tradeoffs: performance, energy, reliability
- However, diverse CMP choices raise difficult questions
  - Performance, application suitability, optimization strategy



Progress to date:
- Explored broadest set of CMP systems in HPC literature
- Examined two diverse scientific algorithms: SPMV and LBMHD
- Developed performance portable auto-tuners
- Achieved highest intra-node performance in the literature
- Continuing effort with additional platforms and algorithms
- Developing rooftop model which offers insight into CMP performance
- Model ties together:
  floating point performance, arithmetic intensity, memory performance



Insights:
- Paradoxically, the most complex/advanced architectures required the most tuning, and delivered the lowest performance
- Niagara2 delivered both very good performance and productivity
- Cell delivered extremely good performance and efficiency (processor and power)
- Our multicore-specific autotuning is required to achieve top performance
- Architectural transparency is invaluable in optimizing code
- Results suggest CMPs designs should *maximize effective bandwidth utilization with increasing cores, even at cost of single core performance*

What is Autotuning?

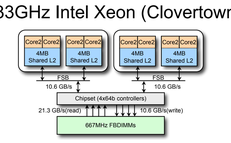
Idea
- There are too many complex architectures with too many possible code transformations to optimize over.
- An optimization on one machine may slow another machine down.
- Need a general, automatic solution

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
- Examples include:

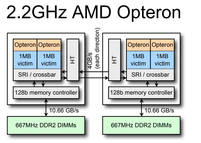
Array Padding	avoids conflicts in the L1/L2
Vectorization	avoids rolling the TLB
Unrolling/DLP	compensates for poor compilers
SW Prefetching	attempts to hide L2 and DRAM latency
SIMDization	compensates for poor compilers, and streaming stores minimize memory traffic

Architectures Evaluated

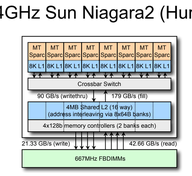
2.33GHz Intel Xeon (Clovertown)



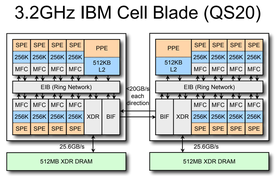
2.2GHz AMD Opteron



1.4GHz Sun Niagara2 (Huron)



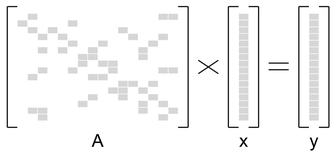
3.2GHz IBM Cell Blade (QS20)



Sparse Matrix-Vector Multiplication

Sparse Matrices
- Most entries are 0.0
- Significant performance advantage in only storing/operating on the nonzeros

Evaluate y=Ax
- A is a sparse matrix
- x & y are dense vectors



Challenges
- Difficult to exploit ILP and DLP
- Irregular memory access to the source vector
- Often difficult to load balance
- Very low computational intensity (often >6bytes/flop)
- **Likely memory bound**

Dataset: the matrices

- Pruned original SPARSITY suite down to 14 matrices of interest
- None should fit in cache12-135MB (memory intensive benchmark)
- 4 categories
- Rank ranging from 2K to 1M

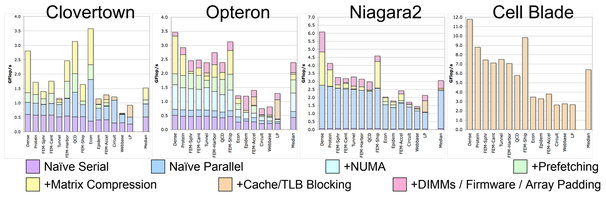
Name	Dense	Protein	FEM / Spheres	FEM / Cantilever	Wind Tunnel	FEM / Harbor	QCD	FEM / Ship	Eco-nomics	Epidem	FEM / Accel	Circuit	webbase	LP
Dimension	2K	36K	83K	62K	218K	47K	49K	141K	207K	526K	121K	171K	1M	4K x 1M
Nonzeros (nonzeros/row)	4.0M (2K)	4.3M (119)	6.0M (72)	4.0M (65)	11.6M (53)	2.37M (50)	1.90M (39)	3.98M (28)	1.27M (6)	2.1M (4)	2.62M (22)	959K (6)	3.1M (3)	11.3M (2825)
CSR footprint	48MB	52MB	72MB	48MB	140MB	28MB	23MB	48MB	16MB	27MB	32MB	12MB	41MB	135MB
Spyplot														

Initial Multicore SpMV Experimentation

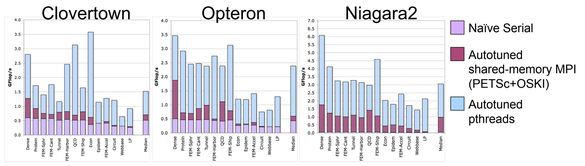
Reference
S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, J. Demmel, "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms", Supercomputing (SC), 2007.

Two threaded implementations:
- Cache-based Pthreads, and Cell local-store based.
- 1D parallelization by rows
- Progressively expanded the autotuner (constant levels of productivity)

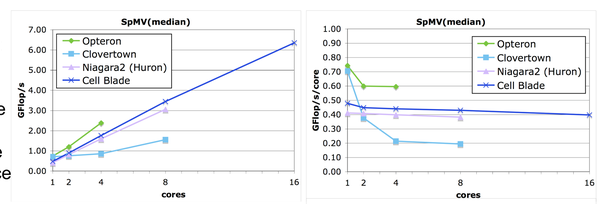
Autotuning
- For x86 machines Naïve parallelization didn't even double performance, but delivered 40x on Niagara2.
- Prefetching(exhaustive search)/DMA was helpful on both x86 machines (more so after compression)
- NUMA is essential on Opteron/Cell
- Matrix Compression(heuristic) often delivered performance better than the reduction in memory traffic
- There is no naïve Cell SPE implementation
- Although local store blocking is essential on Cell for correctness, it is rarely beneficial on cache-based machines
- Machines running a large number of threads need large numbers of DIMMs (not just bandwidth)



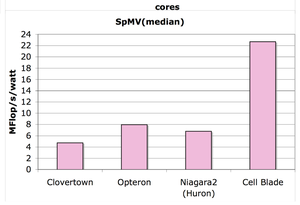
Comparison with MPI
- Compared autotuned pthread implementation with autotuned shared memory MPI (MPICH) (PETSc+OSKI autotuned implementation)
- For many matrices, MPI was only slightly faster than serial on the x86 machines
- MPI rarely scaled beyond 16 threads on Niagara2 (still under investigation)
- Autotuned pthread implementation was often 2-3x faster than autotuned MPI



Performance and Scalability
- All machines showed good multisocket scaling (bandwidth per socket was the limiting factor)
- Clovertown showed very poor multicore scaling (low data utilization on FSB)
- Machines with simpler cores, and more bandwidth delivered better performance

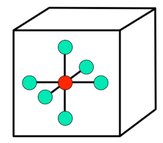


System Power Efficiency
- Used a digital power meter to measure sustained power under load
- 16 FBDIMMs on Niagara2 drove sustained power to 450W, while others required around 300W
- Clovertown delivered lower performance and required more power than Opteron (thus substantially lower power efficiency)
- Cell delivered good performance at moderate power

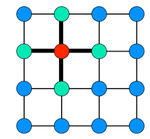


What are structured grids ?

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



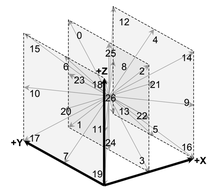
7-point 3D Stencil

- 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)

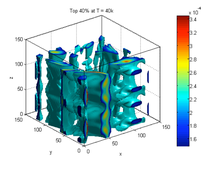
Autotuning Lattice Methods

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.
Winner Best Paper Award Application track

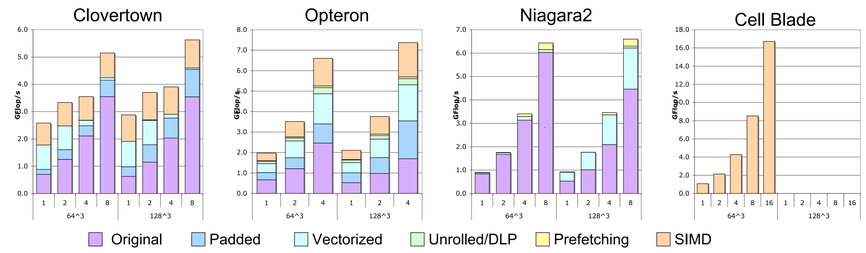
Lattice-Boltzmann Methods
- Out-of-place (Jacobi) style structured grid code
- Popular in CFD
- Simplified kinetic model that maintains the macroscopic quantities
- Distribution functions
  (e.g. 27 velocities per point in space) are used to reconstruct macroscopic quantities



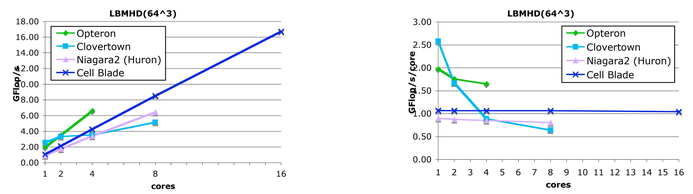
Lattice-Boltzmann Magneto-hydrodynamics (LBMHD)
- Simulates plasma turbulence
- Couples CFD and Maxwell's Equations
- Thus it requires:
  a Momentum (27 component) distribution and
  a Magnetic (45 component) distribution
  7 macroscopic quantities
  (density, momentum, magnetic field)
- Two phases to 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



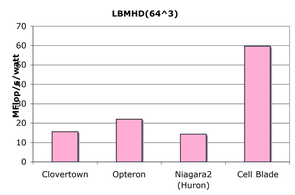
Autotuning LBMHD
- Autotuning dramatically improved performance on the Opteron (4x)
- Became important when the problem could no longer be mapped with Niagara2's 4MB pages
- Although prefetching showed little benefit, SIMD and streaming stores helped significantly.
- Cell was not autotuned, and only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



Office of Science
U.S. DEPARTMENT OF ENERGY