Auto-tuning and the Roofline Model

Sam Williams, David Patterson, Kathy Yelick, Jim Demmel,
Andrew Waterman, Rich Vuduc, Lenny Oliker, John Shalf, Jonathan Carter, …
samw@eecs.berkeley.edu

EECS Electrical Engineering and Computer Sciences

BERKELEY PAR LAB

PARALLEL COMPUTING LABORATORY

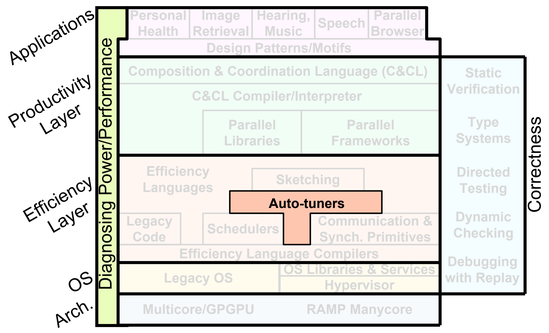
Where does this fit in the ParLab ?

ParLab

- Multi- and manycore is the only foreseeable solution to improve performance or reduce power.
- Vertically and horizontally integrated lab focused on manycore.
- Within each layer are multiple research groups.
- Within each group are multiple projects.
- Two groups (analysis and verification) dive through all layers

My Work

- Fits into two different, but related layers:
  - The Roofline model is a template for analyzing performance
  - Auto-tuning computational motifs is buried in the Efficiency Layer.
- Uses existing multicore SMPs as proxies for next generation multicore computers
  e.g. Victoria Falls with 128 threads is like a 128 core machine



What is Auto-tuning?

Basic Idea

- Provides performance portability across the breadth and evolution of multicore architectures.
- There are too many complex architectures with too many possible code transformations to hand optimize every kernel for every architecture.
- An optimization on one machine may slow another machine down.
- Need a general, automated solution.

Code Generators

- Kernel-specific
- Perl scripts generate 1000's of code variations for various optimizations:
  - **NUMA-Aware** collocates data with the threads processing it
  - **Array Padding** avoids conflicts in the L1/L2
  - **Register Blocking** in the sparse motif, data structure is hierarchically blocked for locality
  - **Cache Blocking** minimizes cache misses and memory traffic
  - **Vectorization** avoids thrashing 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

Auto-tuners

- Search over all possible code variants for best performance.
- Often, an exhaustive search is intractable.
  - The trend is to use heuristics to guide the search.
  - The future is to use performance models to guide the search.

Isn't this just compilation? No.

- Auto-tuners are dataset aware, where compilers are oblivious.
- Auto-tuners are motif-oriented, not code-oriented.
- Auto-tuners can change the data structures, loop structures or even the algorithm at runtime to achieve better performance

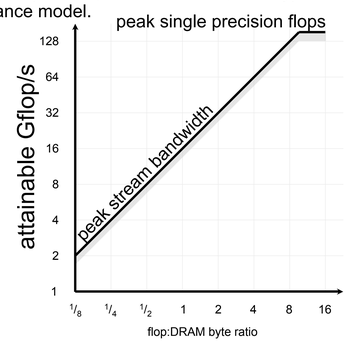
The Roofline Model

Reference

Samuel Williams, Andrew Waterman, David Patterson, "Roofline: An Insightful Multicore Performance Model", (submitted to) Communications of the ACM, 2008

Introduction

- Use bound and bottleneck analysis to distill the key components of architecture and performance (computation, communication, locality) into a visually-intuitive performance model.
- Allow programmers to model, predict, and analyze a kernel's performance.
- Here we restrict the model to memory-intensive SPMD floating-point kernels.

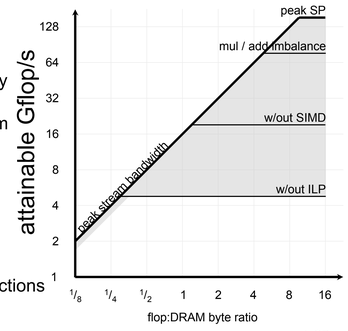


Naïve Roofline Model

- Well known formalism.
- Base on microbenchmarks and optimization manuals.
- Combines communication, computation, and locality into a single figure.

In-core Parallelism

- Current architectures achieve high performance through many forms of in-core parallelism.
- A lack of exploitation of any form of in-core parallelism will degrade performance.
- Delineate performance levels = **in-core ceilings**

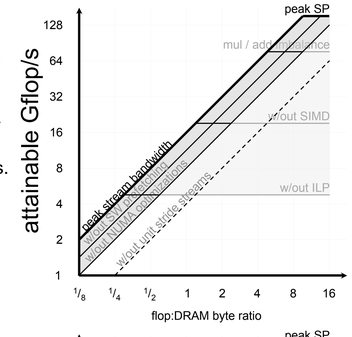


Instruction Mix

- Large numbers of integer instructions can limit FP performance.

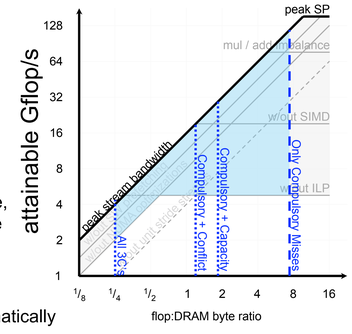
Memory Bandwidth

- High memory bandwidth comes from hiding latency and exploiting parallelism.
- HW prefetchers hide latency for unit-stride access patterns.
- SW prefetchers supplement this.
- Multisocket SMPs require careful placement of data (NUMA optimizations).
- A lack of any of these will degrade memory bandwidth = **bandwidth ceilings**



Locality

- Think 3C's of caches.
- All kernels have compulsory cache misses.
- Caches are finite = capacity misses
- Caches aren't fully associative = conflict misses
- If software doesn't handle these, arithmetic intensity will degrade = **arithmetic intensity walls**



Future Work

- Performance counters will dramatically enhance the Roofline Model.
- Expanding the model to other communication and computation metrics

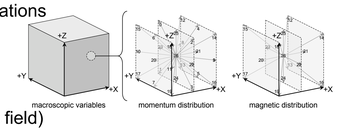
Auto-tuning the Structured Grid Motif

Reference

Samuel Williams, Jonathan Carter, Leonid Oliker, John Shalf, Katherine Yelick, "Lattice Boltzmann Simulation Optimization on Leading Multicore Platforms", International Parallel & Distributed Processing Symposium (IPDPS) (to appear), 2008.
Best Paper, Application Track

Lattice-Boltzmann Magneto-hydrodynamics (LBMHD)

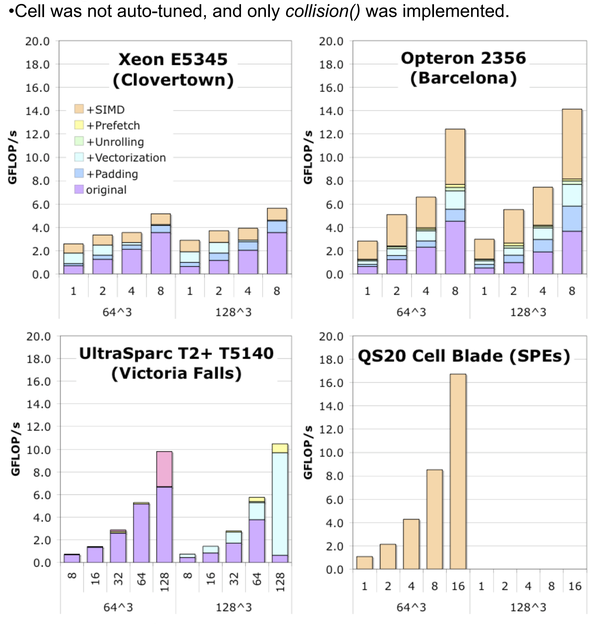
- Simulates plasma turbulence via LBM
- Couples CFD with Maxwell's Equations
- Thus it requires:
  27pt Momentum distribution
  15pt Magnetic distribution
  7 macroscopic quantities
  (density, momentum, magnetic field)



- Two phases to the code:
  collision() advances the grid one time step
  stream() handles the boundary conditions (periodic for benchmark)
- Each lattice update requires ~1300 flops and ~1200 bytes of data
- flop:byte ~ 1.0(ideal), ~0.7(cache-based machines)

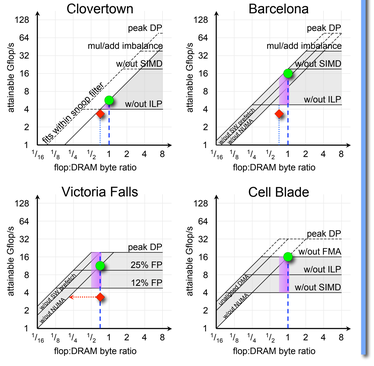
Auto-tuning LBMHD

- Auto-tuning 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 auto-tuned, and only *collision()* was implemented.



Using the Roofline Model

- Out-of-the-box code touches too many arrays per loop iteration.
- Cache bypass improves the arithmetic intensity
- Dataset is too large for the snoop filter to ever work.
- Clearly, no room for improvement on Clovertown, Barcelona, or Cell.



◆ Original performance
● Auto-tuned performance

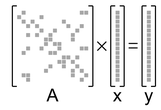
Auto-tuning the Sparse Linear Algebra Motif

Reference

Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, James Demmel, "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms", Supercomputing (SC), 2007.

What's a Sparse Matrix?

- Like a dense matrix,
- but most of the entries are 0.0
- Huge performance advantage in storing/operating on the nonzeros
- CSR is the standard representation
- Requires significant meta data

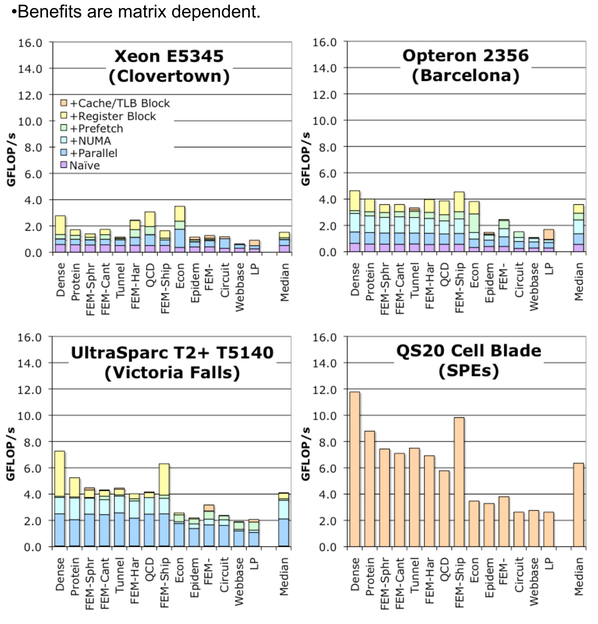


Sparse Matrix Vector Multiplication (SpMV)

- Evaluate **y=Ax**
- **A** is a sparse matrix
- **x** & **y** are dense vectors
- No ILP, DLP, and very low flop:byte ratio (<0.166)

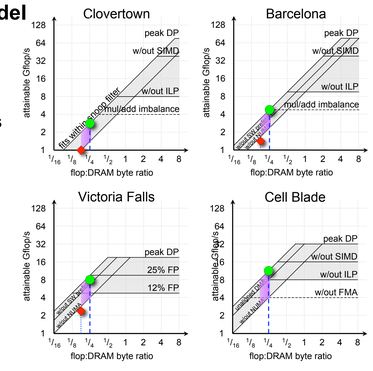
Auto-tuning SpMV

- Register, Cache, and TLB blocking result in hierarchical data structures.
- Exhaustive search for optimal prefetch distance.
- Memory traffic minimization heuristic improves flop:byte ratio (<0.25)
- Dramatic increases in performance across all machines
- SIMDization of little/no value.
- Benefits are matrix dependent.



Using the Roofline Model

- Plot only the **Dense** matrix stored in sparse format
- Clearly heavily memory-bound on all computers
- Register blocking amortizes meta data
- Prefetching/NUMA often essential in delivering higher bandwidth



◆ Original performance
● Auto-tuned performance