The Parallel Revolution Has Started: Are You Part of the Solution or Part of the Problem?

Dave Patterson
Parallel Computing Laboratory (Par Lab)
U.C. Berkeley
April 2009
Outline

- What Caused the Revolution?
- Is it Too Late to Stop It?
- Projected Hardware/Software Context?
- Example Coordinated Attack: Par Lab @ UCB
- Roofline: An Insightful Visual Performance Model
- Conclusion
A Parallel Revolution, Ready or Not

- Power Wall = Brick Wall
  End of way built microprocessors for last 40 years

- New Moore’s Law is 2X processors (“cores”) per chip every technology generation, but \( \approx \) same clock rate

- “This shift toward increasing parallelism is not a triumphant stride forward based on breakthroughs ...; instead, this ... is actually a retreat from even greater challenges that thwart efficient silicon implementation of traditional solutions.”

  The Parallel Computing Landscape: A Berkeley View, Dec 2006

- Sea change for HW & SW industries since changing the model of programming and debugging
2005 IT Roadmap Semiconductors

Clock Rate (GHz)

2001 2003 2005 2007 2009 2011 2013

Intel single core

2005 Roadmap
Change in ITS Roadmap in 2 yrs

Clock Rate (GHz)

2001 2003 2005 2007 2009 2011 2013

Intel single core

Intel multicore

2005 Roadmap

2007 Roadmap
Revolution has started!

- While evolution and global warming are “controversial” in scientific circles, belief in need to switch to parallel computing is unanimous in the hardware community.
- AMD, Intel, IBM, Sun, ... now sell more multiprocessor (“multicore”) chips than uniprocessor chips.
  - Plan on little improvement in clock rate (8% / year?)
  - Expect more cores every 2 years, ready or not.
  - Note – they are already designing the chips that will appear over the next 5 years, and they’re parallel.
Need a Fresh Approach to Parallelism

- Berkeley researchers from many backgrounds meeting since Feb. 2005 to discuss parallelism
  - Krste Asanovic, Ras Bodik, Jim Demmel, Kurt Keutzer, John Kubiatowicz, Edward Lee, George Necula, Dave Patterson, Koushik Sen, John Shalf, John Wawrzynek, Kathy Yelick, ...
  - Circuit design, computer architecture, massively parallel computing, computer-aided design, embedded hardware and software, programming languages, compilers, scientific programming, and numerical analysis

- Tried to learn from successes in high performance computing (LBNL) and parallel embedded (BWRC)


- Goal: Productive, Efficient, Correct, Portable SW for 100+ cores & scale as core increase every 2 years (!)
Context: Re-inventing Client/Server

- “The Datacenter is the Computer”
  - Building sized computers: AWS, Google, MS, ...
  - Private and Public

- “The Laptop/Handheld is the Computer”
  - 2007: Number HP laptops > desktops
  - 2009: “Netbook” small cheap laptop
  - 1B+ Cell phones/yr, increasing in function
  - Apple iPhone, Android, Windows Mobile

- Laptop/Handheld as future client, Datacenter as future server
5 Themes of Par Lab

Applications
  Compelling apps drive top-down research agenda

Identify Common Design Patterns and "Bricks"
  Breaking through disciplinary boundaries

Developing Parallel Software with Productivity, Efficiency, and Correctness
  2 Layers + Coordination & Composition Language
  + Autotuning

OS and Architecture
  Composable primitives, not packaged solutions
  Deconstruction, Fast barrier synchronization, Partitions

Diagnosing Power/Performance Bottlenecks
Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore

Applications

Productivity Layer

Efficiency Layer

Diagnosing Power/Performance

Personal Health

Image Retrieval

Hearing, Music

Speech

Parallel Browser

Design Patterns/Dwarfs

Composition & Coordination Language (C&CL)

C&CL Compiler/Interpreter

Parallel Libraries

Parallel Frameworks

Efficiency Languages

Sketching

Autotuners

Composition & Synch. Primitives

Communication & Synch. Primitives

Efficiency Language Compilers

Legacy Code

Schedulers

Legacy OS

Multicore/GPGPU

OS Libraries & Services

Hypervisor

RAMP Manycore

Static Verification

Type Systems

Directed Testing

Dynamic Checking

Debugging with Replay

Language Compilers

Type Systems

Diagnosing Power/Performance

Efficiency Layer

Legacy OS

Multicore/GPGPU

RAMP Manycore

Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore
What’s the Big Idea?
What’s the Big Idea?

- Big Idea: No (Preconceived) Big Idea!
What’s the Big Idea?

- Big Idea: No (Preconceived) Big Idea!
- In past, apps considered at end of project
What’s the Big Idea?

- Big Idea: No (Preconceived) Big Idea!
- In past, apps considered at end of project
- Instead, work with domain experts at beginning to develop compelling applications
  - Lots of ideas now (and more to come)
What’s the Big Idea?

- Big Idea: No (Preconceived) Big Idea!
- In past, apps considered at end of project
- Instead, work with domain experts at beginning to develop compelling applications
  - Lots of ideas now (and more to come)
- Apps determine in 3-4 yrs which ideas are big
  - Give promising results so far
Compelling Laptop/Handheld Apps (David Wessel)

- Musicians have an insatiable appetite for computation + real-time demands
  - More channels, instruments, more processing, more interaction!
  - Latency must be low (5 ms)
  - Must be reliable (No clicks)

Music Enhancer

- Enhanced sound delivery systems for home sound systems using large microphone and speaker arrays
- Laptop/Handheld recreate 3D sound over ear buds

Hearing Augmenter

- Laptop/Handheld as accelerator for hearing aide

Novel Instrument User Interface

- New composition and performance systems beyond keyboards
- Input device for Laptop/Handheld

Berkeley Center for New Music and Audio Technology (CNMAT) created a compact loudspeaker array: 10-inch-diameter icosahedron incorporating 120 tweeters.
Stroke diagnosis and treatment

- 3rd deaths after heart, cancer
- No treatment >4 hours after
- Rapid Patient-specific 3D Fluid-Structure Interaction analysis of Circle of Willis
  - CoW 80% life-threatening strokes
  - Need highly-accurate simulations in near real-time
  - To evaluate treatment options while minimizing damage > 4 hrs after stroke
Content-Based Image Retrieval

(Kurt Keutzer)

- Query by example
- Image Database
- Relevance Feedback
- Similarity Metric
- Candidate Results
- Final Result

- Built around Key Characteristics of personal databases
  - Very large number of pictures (>5K)
  - Non-labeled images
  - Many pictures of few people
  - Complex pictures including people, events, places, and objects
Compelling Laptop/Handheld Apps

- Meeting Diarist
- Laptops/Handhelds at meeting coordinate to create speaker identified, partially transcribed text diary of meeting
Parallel Browser

- Web 2.0: Browser plays role of traditional OS
  - Resource sharing and allocation, Protection
- Goal: Desktop quality browsing on handhelds
  - Enabled by 4G networks, better output devices
- Bottle-necks to parallelize
  - Parsing, Rendering, Scripting
- "SkipJax"
  - Parallel replacement for JavaScript/AJAX
  - Based on Brown’s FlapJax
Compelling Apps in a Few Years

- **Name Whisperer**
  - Built from Content Based Image Retrieval
  - Like Presidential Aid
- Handheld scans face of approaching person
- Matches image database
- Whispers name in ear, along with how you know him
Theme 2. What to compute?

- Look for common computations across many areas
  Embedded Computing (42 EEMBC benchmarks)
  Desktop/Server Computing (28 SPEC2006)
  Data Base / Text Mining Software
  Games/Graphics/Vision
  Machine Learning / Artificial Intelligence
  Computer Aided Design
  High Performance Computing (Original “7 Dwarfs”)
- Result: 12 Dwarfs
**“Dwarf” Popularity**

How do compelling apps relate to 12 dwarfs?

<table>
<thead>
<tr>
<th></th>
<th>Embed</th>
<th>SPEC</th>
<th>DB</th>
<th>Games</th>
<th>ML</th>
<th>CAD</th>
<th>HPC</th>
<th>Health</th>
<th>Image</th>
<th>Speech</th>
<th>Music</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Finite State Mach.</td>
<td>![Health]</td>
<td>![Image]</td>
<td>![Speech]</td>
<td>![Music]</td>
<td>![Browser]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Circuits</td>
<td>![Health]</td>
<td>![Image]</td>
<td>![Speech]</td>
<td>![Music]</td>
<td>![Browser]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Graph Algorithms</td>
<td>![Health]</td>
<td>![Image]</td>
<td>![Speech]</td>
<td>![Music]</td>
<td>![Browser]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Structured Grid</td>
<td>![Health]</td>
<td>![Image]</td>
<td>![Speech]</td>
<td>![Music]</td>
<td>![Browser]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Dense Matrix</td>
<td>![Health]</td>
<td>![Image]</td>
<td>![Speech]</td>
<td>![Music]</td>
<td>![Browser]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Spectral (FFT)</td>
<td>![Health]</td>
<td>![Image]</td>
<td>![Speech]</td>
<td>![Music]</td>
<td>![Browser]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Dynamic Prog</td>
<td>![Health]</td>
<td>![Image]</td>
<td>![Speech]</td>
<td>![Music]</td>
<td>![Browser]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Particle Methods</td>
<td>![Health]</td>
<td>![Image]</td>
<td>![Speech]</td>
<td>![Music]</td>
<td>![Browser]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Backtrack/ B&amp;B</td>
<td>![Health]</td>
<td>![Image]</td>
<td>![Speech]</td>
<td>![Music]</td>
<td>![Browser]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Graphical Models</td>
<td>![Health]</td>
<td>![Image]</td>
<td>![Speech]</td>
<td>![Music]</td>
<td>![Browser]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Unstructured Grid</td>
<td>![Health]</td>
<td>![Image]</td>
<td>![Speech]</td>
<td>![Music]</td>
<td>![Browser]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Themes 1 and 2 Summary

- Application-Driven Research (top down) vs. CS Solution-Driven Research (bottom up)
  - Bet is not that every program speeds up with more cores, but that we can find some compelling ones that do
- Drill down on (initially) 5 app areas to guide research agenda
- Dwarfs + Design Patterns to guide design of apps through layers
Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore
Theme 3: Developing Parallel SW

- 2 types of programmers ➔ 2 layers
  - **Efficiency Layer** (10% of today’s programmers)
    - Expert programmers build Frameworks & Libraries, Hypervisors, ...
    - “Bare metal” efficiency possible at Efficiency Layer
  - **Productivity Layer** (90% of today’s programmers)
    - Domain experts / Naïve programmers productively build parallel apps using frameworks & libraries
    - Frameworks & libraries composed to form app frameworks
  - Effective composition techniques allows the efficiency programmers to be highly leveraged; major challenge
Ensuring Correctness

**Productivity Layer**
- Enforce independence of tasks using decomposition (partitioning) and copying operators
- Goal: Remove chance for concurrency errors (e.g., nondeterminism from execution order, not just low-level data races)

**Efficiency Layer: Check for subtle concurrency bugs (races, deadlocks, and so on)**
- Mixture of verification and automated directed testing
- Error detection on frameworks with sequential code as specification
- Automatic detection of races, deadlocks
21st Century Code Generation

- Problem: generating optimal code like searching for needle in haystack
- Manycore ➔ even more diverse
- New approach: “Auto-tuners”
  - 1st generate program variations of combinations of optimizations (blocking, prefetching, ...) and data structures
  - Then compile and run to heuristically search for best code for that computer
- Examples: PHiPAC (BLAS), Atlas (BLAS), Spiral (DSP), FFT-W (FFT)
Theme 3: Summary

- Autotuning vs. Static Compiling
- Productivity Layer & Efficiency Layer
- Composability of Libraries/Frameworks
- Libraries and Frameworks to leverage experts
Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore

- Applications
  - Personal Health Image Retrieval Hearing, Music Speech Parallel Browser
- Design Patterns/Dwarfs
  - Composition & Coordination Language (C&CL)
  - C&CL Compiler/Interpreter
    - Parallel Libraries
    - Parallel Frameworks
- Efficiency Languages
- Legacy Code
  - Schedulers
  - Communication & Synch. Primitives
- Efficiency Language Compilers
- OS
  - Legacy OS
  - OS Libraries & Services
    - Hypervisor
- Arch.
  - Multicore/GPGPU
  - RAMP Manycore
- Productivity Layer
- Correctness
  - Static Verification
  - Type Systems
  - Directed Testing
  - Dynamic Checking
  - Debugging with Replay
Theme 4: OS and Architecture
(Krste Asanovic, Eric Brewer, John Kubiatowicz)

- HW Solutions: Small is Beautiful
  - Expect many modestly pipelined (5- to 9-stage) CPUs, FPUs, vector, SIMD Proc. Elmts
  - Reconfigurable Memory Hierarchy
  - Offer HW partitions with 1-ns Barriers

- Deconstructing Operating Systems
  - Resurgence of interest in virtual machines
  - Leverage HW partitioning for thin hypervisors
    ⇒ Allow SW full access to HW in partition
1008 Core “RAMP Blue”

- 1008 = 12 32-bit RISC cores / FPGA, 4 FGPAs/board, 21 boards
  - Simple MicroBlaze soft cores @ 90 MHz
    - Full star-connection between modules
- NASA Advanced Supercomputing (NAS) Parallel Benchmarks (all class S)
  - UPC versions (C plus shared-memory abstraction)
Recent Results: RAMP Gold

- Zhangxi Tan
  - Advisors Krste Asanovic, David Patterson
- FPGA: RAM easy, Mux hard, DRAM fast
  - Pipeline forwarding? Accurate timing?
- RAMP Gold: Standard ISA + Timing Model + highly multithreaded, nonpipelined core
  - Multithreads use RAM, switch threads/cycle
  - Variable Timing Model to explore cache hierarchies, DRAM speeds, processors speeds, ...  
- 64 Threads SPARC v8 @100 MHz: 1 BIPS/FPGA
  - Boots BSD UNIX
Par Lab Domain Expert Deal

- Get help developing application on latest commercial multicores / GPUs and legacy OS
  + Develop using many fast, recent, stable computers
  + Develop on preproduction version of new computers
  – Conventional architectures and OS, but many types

- Will help port app to innovative Par Lab Arch
Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore

- Personal Health
- Image Retrieval
- Hearing, Music
- Speech
- Parallel Browser

Design Patterns/Dwarfs

- Composition & Coordination Language (C&CL)

C&CL Compiler/Interpreter

- Parallel Libraries
- Parallel Frameworks

Efficiency Languages

- Legacy Code
- Schedulers

Communication & Synch. Primitives

Sketching

Autotuners

Efficiency Language Compilers

OS Libraries & Services

Legacy OS

Multicore/GPGPU

RAMP Manycore

Static Verification

Type Systems

Directed Testing

Dynamic Checking

Debugging with Replay

Diagnosing Power/Performance

Efficiency Layer

Productivity Layer

Applications
Theme 5: Diagnosing Power/Performance Bottlenecks (Demmel)

- Measure Power/Performance bottlenecks
  - Aid scheduler, OS, autotuner in adapting system

- Turn into info to help efficiency-level programmer?
  - Am I using 100% of memory bandwidth?

- Turn into info to help productivity programmer?
  - If I change it like this, impact on Power/Performance?

- An IEEE Counter Standard for all multicores?
  - => Portable monitoring tool kit, OS/scheduling aid
  - Measuring utilization accurately => New Optimization
  - RAMP Gold 1st implementation, help evolve standard
Recent Results: Active Testing
Recent Results: Active Testing

- Pallavi Joshi, Chang-Seo Park,
- Advisor Koushik Sen
Recent Results: Active Testing

- Pallavi Joshi, Chang-Seo Park,
  - Advisor Koushik Sen

- Problem: Concurrency Bugs
Recent Results: Active Testing

- Pallavi Joshi, Chang-Seo Park,
  - Advisor Koushik Sen
- Problem: Concurrency Bugs
- Actively control the scheduler to force potentially buggy schedules: Data races, Atomicity Violations, Deadlocks
Recent Results: Active Testing

- Pallavi Joshi, Chang-Seo Park,
  - Advisor Koushik Sen

- Problem: Concurrency Bugs

- Actively control the scheduler to force potentially buggy schedules: Data races, Atomicity Violations, Deadlocks

- Found parallel bugs in real OSS code: Apache Commons Collections, Java Collections Framework, Java Swing GUI framework, and Java Database Connectivity (JDBC)
Results: Making Autotuning “Auto”
Results: Making Autotuning “Auto”

- Archana Ganapathi & Kaushik Datta
- Advisors Jim Demmel and David Patterson
Results: Making Autotuning “Auto”

- Archana Ganapathi & Kaushik Datta
  - Advisors Jim Demmel and David Patterson
- Problem: need expert in architecture and algorithm for search heuristics
Results: Making Autotuning “Auto”

- Archana Ganapathi & Kaushik Datta
  - Advisors Jim Demmel and David Patterson
- Problem: need expert in architecture and algorithm for search heuristics
- Instead, Machine Learning to Correlate Optimizations and Performance
Results: Making Autotuning “Auto”

- Archana Ganapathi & Kaushik Datta
  - Advisors Jim Demmel and David Patterson
- Problem: need expert in architecture and algorithm for search heuristics
- Instead, Machine Learning to Correlate Optimizations and Performance
- Evaluate in 2 hours vs. 6 months
Results: Making Autotuning “Auto”

- Archana Ganapathi & Kaushik Datta
  - Advisors Jim Demmel and David Patterson
- Problem: need expert in architecture and algorithm for search heuristics
- Instead, Machine Learning to Correlate Optimizations and Performance
- Evaluate in 2 hours vs. 6 months
- Match or Beat Expert for Stencil Dwarfs
Results: Fast Dense Linear Algebra
Results: Fast Dense Linear Algebra

- Mark Hoemmen: LINPACK benchmark made dense linear algebra seem easy
  - If solve impractically large problems \((10^6 \times 10^6)\)
  - Advisor Jim Demmel
Results: Fast Dense Linear Algebra

- Mark Hoemmen: LINPACK benchmark made dense linear algebra seem easy
  - If solve impractically large problems \((10^6 \times 10^6)\)
  - Advisor Jim Demmel

- Problem: Communication limits perf. for non-huge matrices and increasing core counts
Results: Fast Dense Linear Algebra

- Mark Hoemmen: LINPACK benchmark made dense linear algebra seem easy
  - If solve impractically large problems ($10^6 \times 10^6$)
  - Advisor Jim Demmel

- Problem: Communication limits perf. for non-huge matrices and increasing core counts

- New way to panel matrix to minimize comm.
  - “Tall Skinny” QR factorization
Results: Fast Dense Linear Algebra

- Mark Hoemmen: LINPACK benchmark made dense linear algebra seem easy
  - If solve impractically large problems ($10^6 \times 10^6$)
  - Advisor Jim Demmel

- Problem: Communication limits perf. for non-huge matrices and increasing core counts
  - New way to panel matrix to minimize comm.
    - "Tall Skinny" QR factorization

- IBM BlueGene/L, 32 cores: up to 4× faster
Results: Fast Dense Linear Algebra

- Mark Hoemmen: LINPACK benchmark made dense linear algebra seem easy
  - If solve impractically large problems (10^6 × 10^6)
  - Advisor Jim Demmel

- Problem: Communication limits perf. for non-huge matrices and increasing core counts
- New way to panel matrix to minimize comm.
  - “Tall Skinny” QR factorization

- IBM BlueGene/L, 32 cores: up to 4× faster
- Pentium III cluster, 16 cores: up to 6.7× faster
  - vs. Parallel LINPACK (ScaLAPACK) on 10^5 × 200 matrix
Recent Results: App Acceleration
Recent Results: App Acceleration

- Bryan Catanzaro: Parallelizing Computer Vision (image segmentation) using GPU
Recent Results: App Acceleration

- Bryan Catanzaro: Parallelizing Computer Vision (image segmentation) using GPU
- Problem: Malik’s highest quality algorithm is 7.8 minutes / image on PC
Recent Results: App Acceleration

- Bryan Catanzaro: Parallelizing Computer Vision (image segmentation) using GPU
- Problem: Malik’s highest quality algorithm is 7.8 minutes / image on PC
- Invention + talk within Par Lab on parallelizing phases using new algorithms, data structures
  - Bor-Yiing Su, Yunsup Lee, Narayanan Sundaram, Mark Murphy, Kurt Keutzer, Jim Demmel, and Sam Williams
Recent Results: App Acceleration

- Bryan Catanzaro: Parallelizing Computer Vision (image segmentation) using GPU
- Problem: Malik’s highest quality algorithm is 7.8 minutes / image on PC
- Invention + talk within Par Lab on parallelizing phases using new algorithms, data structures
  - Bor-Yiing Su, Yunsup Lee, Narayanan Sundaram, Mark Murphy, Kurt Keutzer, Jim Demmel, and Sam Williams
- Current GPU result: 2.1 seconds / image
Recent Results: App Acceleration

- Bryan Catanzaro: Parallelizing Computer Vision (image segmentation) using GPU
- Problem: Malik’s highest quality algorithm is 7.8 minutes / image on PC
- Invention + talk within Par Lab on parallelizing phases using new algorithms, data structures
  - Bor-Yiing Su, Yunsup Lee, Narayanan Sundaram, Mark Murphy, Kurt Keutzer, Jim Demmel, and Sam Williams
- Current GPU result: 2.1 seconds / image
- > 200X speedup
  - Factor of 10 quantitative change is a qualitative change
Par Lab Summary

- Try Apps-Driven vs. CS Solution-Driven Research
- Design patterns + Dwarfs
- Efficiency layer for \( \approx 10\% \) today’s programmers
- Productivity layer for \( \approx 90\% \) today’s programmers
- Autotuners vs. Compilers
- OS & HW: Primitives vs. Solutions
- Verification Directed Testing
- Counter Standard to

Easy to write correct programs that run efficiently and scale up on manycore

- Personal Health
- Image Retrieval
- Hearing, Music
- Speech
- Parallel Browse
- Design Patterns/Dwarfs
- Composition & Coordination Language (C&CL)
- C&CL Compiler/Interpreter
- Parallel Libraries
- Parallel Frameworks
- Sketching
- Communication & Synch. Primitives
- Efficiency Languages
- Autotuners
- Legacy Code
- Schedulers
- Efficiency Language Compilers
- OS Libraries & Services
- Legacy OS
- Hypervisor
- Multicore/GPGPU
- RAMP Manycore

Efficiency Language Compilers
- Legacy OS
- OS Libraries & Services
- Hypervisor
- Multicore/GPGPU
- RAMP Manycore

Correctness
- Static Verification
- Type Systems
- Directed Testing
- Dynamic Checking
- Debugging with Replay
Outline

- What Caused the Revolution?
- Is it Too Late to Stop It?
- Projected Hardware/Software Context?
- Why Might We Succeed (this time)?
- Example Coordinated Attack: Par Lab @ UCB
- Roofline: An Insightful Visual Performance Model
- Conclusion
Why Multicore Performance Model?

- No consensus on multicore architecture
  - Number cores, thick vs. thin, SIMD/Vector or not,
    Caches vs. Local Stores, homogeneous or not, ...

- Must Programmer become expert on application and all new computers to deliver good performance on multicore?
  - 1% programmers know both?
Multicore SMPs (All Dual Sockets)

**Intel Xeon E5345 (Clovertown)**

- 2.33 GHz, 8 Fat cores, Memory Bus

**AMD Opteron 2356 (Barcelona)**

- 2.30 GHz, 8 Fat cores

**Sun T2+ T5140 (Victoria Falls)**

- 1.17 GHz, 16 thin 8-way MT cores

**IBM QS20 Cell Blade**

- 3.20 GHz, 16 thin SIMD cores
Assumptions for New Model

- Focus on 13 Dwarfs
  - Use floating point versions here; others in future
- Bound and Bottleneck Model good enough; e.g. Amdahl’s Law
  - Don’t need accuracy to 10% to understand which optimizations to try to get next level of performance
- Can use SPMD programming model so parallelization, load balancing not issues
  - Have looked at accommodating load balancing, parallelization, but not shown today
Bounds/Bottlenecks?

- For Floating Point Dwarfs: Peak Floating Point Performance Bound
- For Dwarfs that don’t fit all in cache: DRAM Memory Performance Bound
- **Operational Intensity** = Average number of Floating Point Operations per Byte to DRAM
  - Between cache and DRAM vs. processor and cache
  - Varies by multicore design (cache org.) and dwarf
Can Graph Performance Model?

- For Floating Point Dwarfs, Y axis is performance or GFLOPs/second
- Log Scale
Can Graph Performance Model?

- Suppose X-axis was Operational Intensity?
  - FLOPs per Bytes to DRAM
  - Log scale

- Can plot Memory BW Bound (GBytes/sec) since
  \[
  \frac{\text{GFLOPs/sec}}{\text{FLOPs/Byte}} = \text{GBytes/sec}
  \]
Can Graph Performance Model?

- Suppose X-axis was Operational Intensity?
  - FLOPs per Bytes to DRAM
  - Log scale
- Can plot Memory BW Bound (GBytes/sec) since
  \[
  \frac{\text{GFLOPs/sec}}{\text{FLOPs/Byte}} = \text{GBytes/sec}
  \]

“Roofline”
Roofline Visual Performance Model

![Graph showing the Roofline model with a Ridge Point]

**Ridge Point**

- Peak memory bandwidth (stream)
- Peak floating-point performance

**Axes:**
- Operational Intensity (Flops/Byte)
- Attainable GFlops/s
Roofline Visual Performance Model

- "Ridge Point": minimum Operational Intensity to get Peak Performance
  - Operational Intensity is avg. FLOPs/Byte per dwarf
  - Compute Bound?
  - Memory Bound?
Roofline Visual Performance Model

- **“Ridge Point”**: minimum Operational Intensity to get Peak Performance
  - Operational Intensity is avg. FLOPs/Byte per dwarf
  - Compute Bound?
  - Memory Bound?
Roofline Visual Performance Model

- **“Ridge Point”:** minimum Operational Intensity to get Peak Performance
  - Operational Intensity is avg. FLOPs/Byte per dwarf
  - Compute Bound?
  - Memory Bound?
"Ridge Point": minimum Operational Intensity to get Peak Performance
- Operational Intensity is avg. FLOPs/Byte per dwarf
- Compute Bound?
- Memory Bound?

What do Real Rooflines Look Like?
Where do Real Dwarfs map on Roofline?
Roofline Models of Real Multicores

- **Intel Clovertown**
  - Peak: 75 GFLOPS
  - Ridge Point: 6.7

- **AMD Barcelona**
  - Peak: 74 GFLOPS
  - Ridge Point: 4.4

- **IBM Cell Blade**
  - Peak: 29 GFLOPS
  - Ridge Point: 0.65

- **Sun Victoria Falls**
  - Peak: 19 GFLOPS
  - Ridge Point: 0.33
Roofline Models of Real Multicores

- Intel Clovertown
  Peak: 75 GFLOPS
  Ridge Point: 6.7
- AMD Barcelona
  Peak: 74 GFLOPS
  Ridge Point: 4.4
- IBM Cell Blade
  Peak: 29 GFLOPS
  Ridge Point: 0.65
- Sun Victoria Falls
  Peak: 19 GFLOPS
  Ridge Point: 0.33

<table>
<thead>
<tr>
<th>Op. Int.</th>
<th>1/4</th>
<th>1/2</th>
<th>1.07</th>
<th>1.64</th>
</tr>
</thead>
<tbody>
<tr>
<td>dwarf</td>
<td>SpMV</td>
<td>Stencil</td>
<td>LBMHD</td>
<td>3-D FFT</td>
</tr>
</tbody>
</table>
What if Performance < Roofline?

- Measure computational and memory optimizations in advance
- Order computation optimizations and memory optimizations as “ceilings” below Roofline
  - Need to perform optimization to break thru ceiling
- Use Operational Intensity to pick whether do memory or computation optimization (or both)
Adding Computational Ceilings

- Has separate multipliers and adders: * = + ?
- SIMD?
  (2 Flops/Instr)
- 4 instructions per clock cycle?
Memory + Comp Ceilings

- Memory Optimizations
- Prefetching
- NUMA optimizations (use DRAM local to socket)
Memory+Comp Ceilings

- Partitions expected perf. into 3 optim. regions:
  - Compute only
  - Memory only
  - Compute+ Memory

![Graph showing memory-processor ceilings with partitions into three optim regions: compute only, memory only, and compute+ memory.](image)
Memory+Comp Ceilings

- Partitions expected perf. into 3 optim. regions:
  - Compute only
  - Memory only
  - Compute+ Memory
Status of Roofline Model

- Used for 2 other kernels on 4 other multicores
  - Evaluate 2 financial PDE solvers
  - Intel Penryn & Larrabee + NVIDIA B80 & GTX280
  - Version 1 fit in L1$, enough BW for peak throughput
  - Version 2 didn’t fit, Roofline helped figure out cache blocking to reach peak throughput

- Non-floating point kernels would be interesting
  - e.g., Sort (potential exchanges/sec vs GB/s), Graph Traversal (nodes traversed/sec vs. GB/s)

- Opportunities for others to help investigate: many kernels, multicores, metrics
Conclusion

- Power wall ➔ Parallelism
  - Can’t build a 15 GHz, 100 W uniprocessor
- Industry bet its future on parallel computing
  - Hardest CS Problem in 50 years
- Once in a career opportunity to reinvent whole hardware/software stack if can make it easy to write correct, efficient, portable, scalable parallel programs
- Failure is not the sin; the sin is not trying
Conclusion

- Power wall $\rightarrow$ Parallelism
  - Can’t build a 15 GHz, 100 W uniprocessor
- Industry bet its future on parallel computing
  - Hardest CS Problem in 50 years
- Once in a career opportunity to reinvent whole hardware/software stack if can make it easy to write correct, efficient, portable, scalable parallel programs
- Failure is not the sin; the sin is not trying
- Are you going to be part of the problem or part of the solution?
Acknowledgments

- Learn more about Par Lab: 5 Par Lab papers in HotPar09
  - www.usenix.org/event/hotpar09
  - ML-Autotune, OS, Browser Design, Library composition, Communication

- Learn more about Roofline:

- Thanks to Faculty, Students, and Staff in Par Lab
  - See parlab.eecs.berkeley.edu

- Thanks to RAMP Consortium
  - See ramp.eecs.berkeley.edu

- Thanks to Intel & Microsoft as Par Lab founding sponsors; Samsung, NEC, National Instruments Affiliates
  - If interested in Par Lab affiliate sponsorship, contact me
New Par Lab: Opened Dec 1, 2008

- 5th Floor South Soda Hall
- Founding Partners: Intel and Microsoft
- Affiliate Partners: Samsung, National Inst, NEC