



Computer Architecture is Back - The Berkeley View on the Parallel Computing Landscape

David Patterson, Krste Asanović, Kurt Keutzer,
and a cast of thousands
U.C. Berkeley

January, 2007



High Level Message

- Everything is changing
- Old conventional wisdom is out
- We **desperately** need new approach to HW and SW based on parallelism since industry has bet its future that parallelism works □
- Need to create a “watering hole” to bring everyone together to quickly find that solution
 - architects, language designers, application experts, numerical analysts, algorithm designers, programmers, ...



Outline

- Old vs. New Conventional Wisdom
- 7 Questions to Frame Parallel Research
- New Benchmarks for New Architectures
- Hardware Building Blocks
- Human-centric Programming Model
- Innovating at HW/SW interface without Compilers
- Deconstructing Operating Systems
- Building innovative computers without custom chips
- Optimism for Parallel Computing Revolution?
- Where do we go from here?



Conventional Wisdom (CW) in Computer Architecture

- 1. Old CW:* Power is free, but transistors expensive
 - *New CW* is the "Power wall":
Power is expensive, but transistors are "free"
 - Can put more transistors on a chip than have the power to turn on
- 2. Old CW:* Only concern is dynamic power
 - *New CW:* For desktops and servers, static power due to leakage is 40% of total power
- 3. Old CW:* Monolithic uniprocessors are reliable internally, with errors occurring only at pins
 - *New CW:* As chips drop below 65 nm feature sizes, they will have high soft and hard error rates



Conventional Wisdom (CW) in Computer Architecture

4. *Old CW*: By building upon prior successes, continue raising level of abstraction and size of HW designs
 - *New CW*: Wire delay, noise, cross coupling, reliability, clock jitter, design validation, ... stretch development time and cost of large designs at ≤ 65 nm
5. *Old CW*: Researchers demonstrate new architectures by building chips
 - *New CW*: Cost of 65 nm masks, cost of ECAD, and design time for GHz clocks
 \Rightarrow Researchers no longer build believable chips
6. *Old CW*: Performance improves latency & bandwidth
 - *New CW*: BW improves $> (\text{latency improvement})^2$



Conventional Wisdom (CW) in Computer Architecture

7. *Old CW*: Multiplies slow, but loads and stores fast

■ *New CW* is the "Memory wall":
Loads and stores are slow, but multiplies fast

- 200 clocks to DRAM, but even FP multiplies only 4 clocks

8. *Old CW*: We can reveal more ILP via compilers and architecture innovation

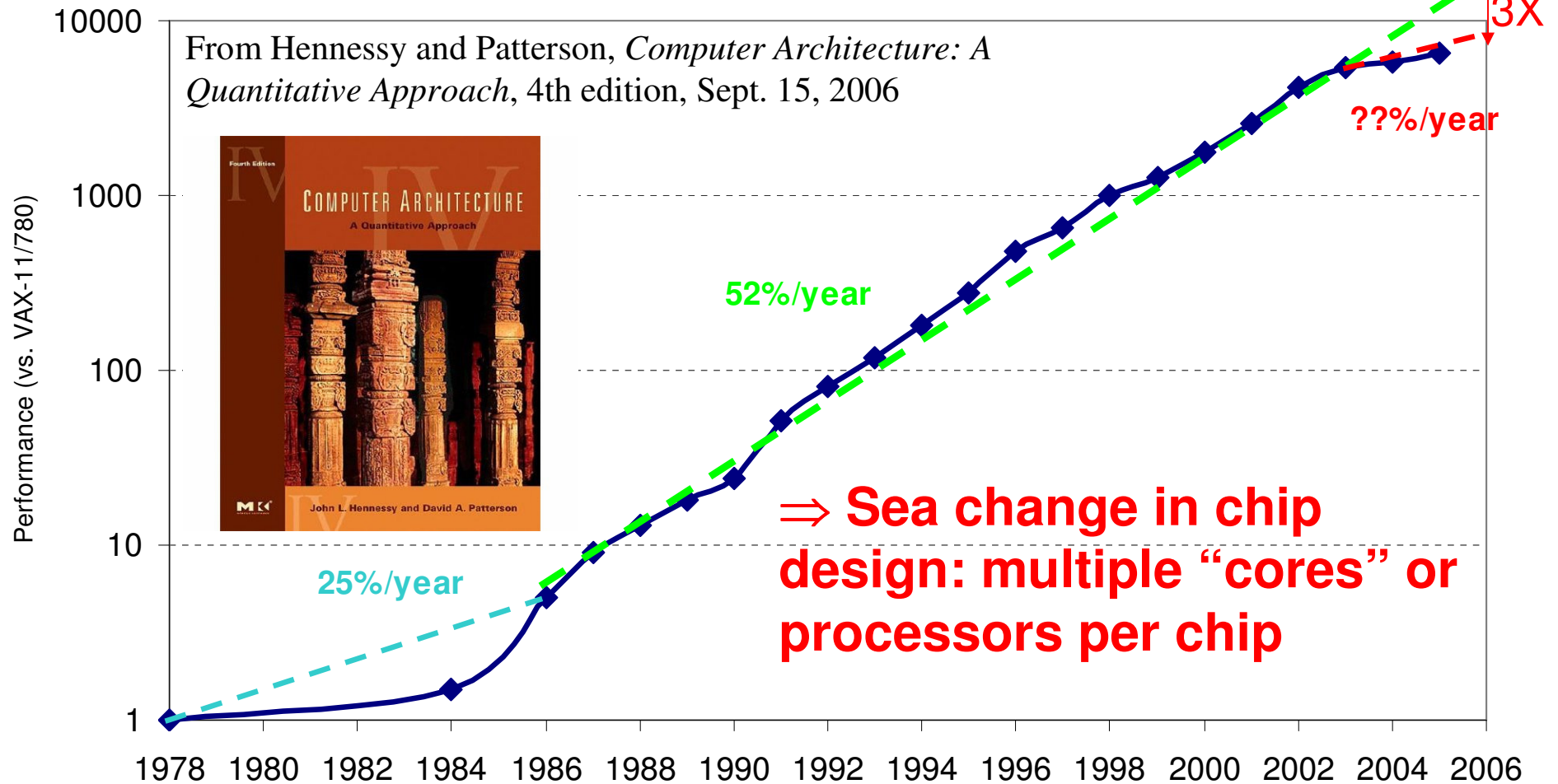
- Branch prediction, OOO execution, speculation, VLIW, ...

■ *New CW* is the "ILP wall":
Diminishing returns on finding more ILP

9. *Old CW*: 2X CPU Performance every 18 months

■ *New CW* is *Power Wall + Memory Wall + ILP Wall*
= *Brick Wall*

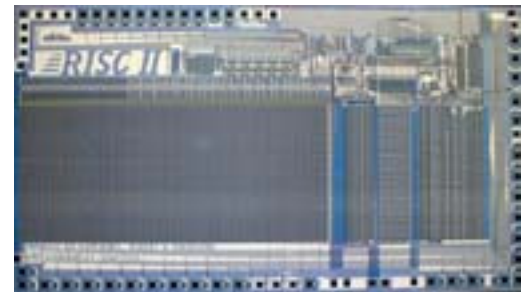
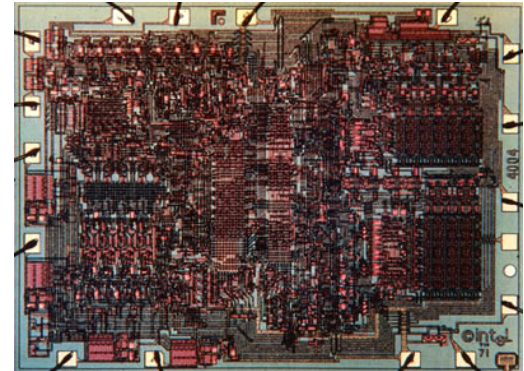
Uniprocessor Performance (SPECint)



- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

Sea Change in Chip Design

- Intel 4004 (1971): 4-bit processor, 2312 transistors, 0.4 MHz, 10 micron PMOS, 11 mm² chip
 - RISC II (1983): 32-bit, 5 stage pipeline, 40,760 transistors, 3 MHz, 3 micron NMOS, 60 mm² chip
 - 125 mm² chip, 0.065 micron CMOS
= 2312 RISC II+FPU+Icache+Dcache
 - RISC II shrinks to ≈ 0.02 mm² at 65 nm
 - Caches via DRAM or 1 transistor SRAM or 3D chip stacking
 - Proximity Communication via capacitive coupling at > 1 TB/s ?
(Ivan Sutherland @ Sun / Berkeley)
- **Processor is the new transistor!**





Conventional Wisdom (CW) in Computer Architecture

- 10. Old CW:* Increasing clock frequency is primary method of performance improvement
 - *New CW:* Processors Parallelism is primary method of performance improvement
- 11. Old CW:* Don't bother parallelizing app, just wait and run on much faster sequential computer
 - *New CW:* No one building 1 processor per chip
 - End of La-Z-Boy Programming Era
- 12. Old CW:* Less than linear scaling for a multiprocessor is failure
 - *New CW:* Given the switch to parallel hardware, even sublinear speedups are beneficial



Parallelism again? What's different this time?

“This shift toward increasing parallelism is not a triumphant stride forward based on breakthroughs in novel software and architectures for parallelism; instead, this **plunge into parallelism is actually a retreat from even greater challenges that thwart efficient silicon implementation of traditional uniprocessor architectures.**”

Berkeley View, December 2006

- HW/SW Industry bet its future that breakthroughs will appear before its too late

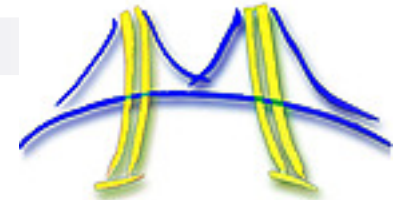


From Multiprogramming to Multithreading

- Multiprogrammed workloads (mix of independent sequential tasks) might obviously benefit from first few generations of multicores
- But how will single tasks get faster on future manycores?

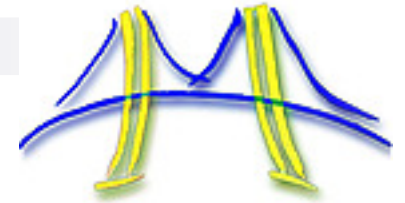


Need a New Approach



- Berkeley researchers from many backgrounds met between February 2005 and December 2006 to discuss parallelism
 - Circuit design, computer architecture, massively parallel computing, computer-aided design, embedded hardware and software, programming languages, compilers, scientific programming, and numerical analysis
- Krste Asanović, Rastislav Bodik, Bryan Catanzaro, Joseph Gebis, Parry Husbands, Kurt Keutzer, Dave Patterson, William Plishker, John Shalf, Samuel Williams, Katherine Yelick + others
- Tried to learn from successes in embedded and high performance computing
- Led to 7 Questions to frame parallel research

7 Questions for Parallelism



■ Applications:

1. What are the apps?
2. What are kernels of apps?

■ Hardware:

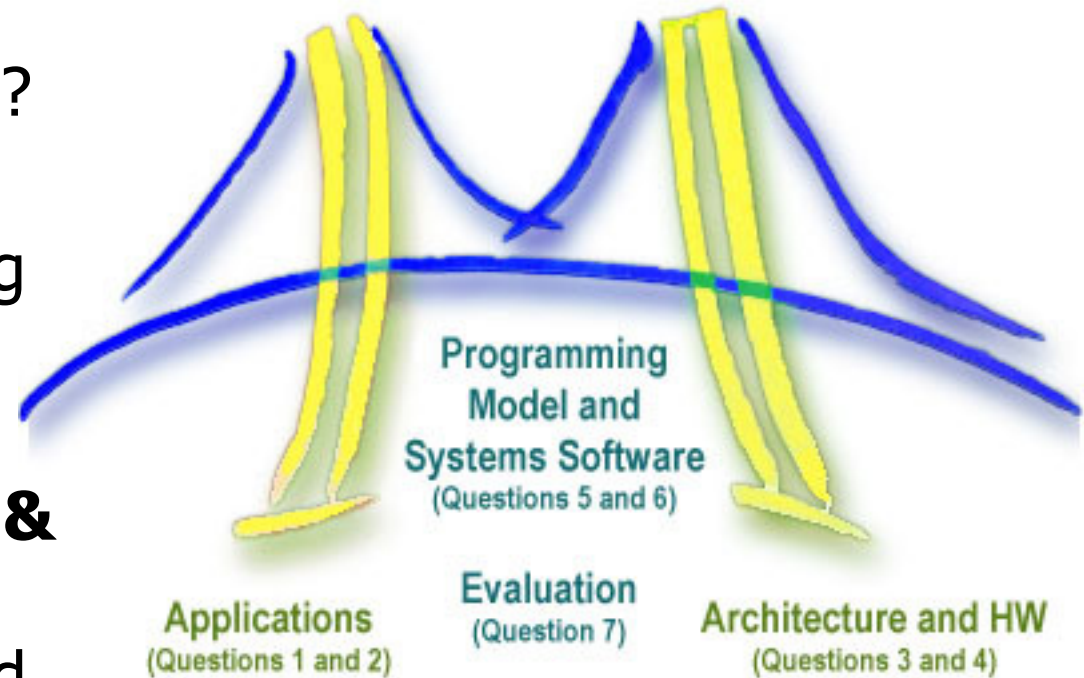
3. What are the HW building blocks?
4. How to connect them?

■ Programming Model & Systems Software:

5. How to describe apps and kernels?
6. How to program the HW?

■ Evaluation:

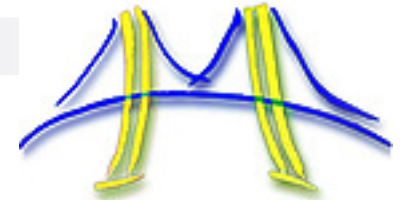
7. How to measure success?




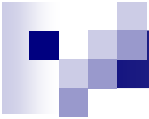
(Inspired by a view of the Golden Gate Bridge from Berkeley)



Apps and Kernels



- Old CW: Since cannot know future programs, use old programs to evaluate future computers
 - e.g., SPEC2006, EEMBC
- What about parallel codes?
 - Few, tied to old models, languages, architectures, ...
- New approach: Design future computers for patterns of computation and communication important in the future
- Claim: 13 “dwarfs” are key for next decade, so design for them!
 - Representative codes may vary over time, but these dwarfs will be important for > 10 years



Phillip Colella's "Seven dwarfs"



High-end simulation in the physical sciences = 7 numerical methods:

1. Structured Grids (including locally structured grids, e.g. Adaptive Mesh Refinement)
 2. Unstructured Grids
 3. Fast Fourier Transform
 4. Dense Linear Algebra
 5. Sparse Linear Algebra
 6. Particles
 7. Monte Carlo
- A dwarf is a pattern of computation and communication
 - Dwarfs are well-defined targets from algorithmic, software, and architecture standpoints

Slide from "Defining Software Requirements for Scientific Computing", Phillip Colella, 2004

Do dwarfs work well outside HPC?

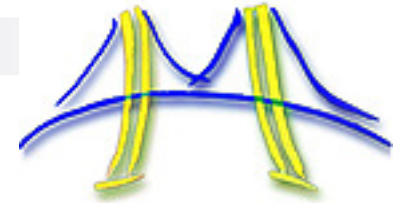


- Examine effectiveness 7 dwarfs elsewhere
 1. Embedded Computing (EEMBC benchmark)
 2. Desktop/Server Computing (SPEC2006)
 3. Machine Learning
 - Advice from Mike Jordan and Dan Klein of UC Berkeley
 4. Games/Graphics/Vision
 5. Data Base Software
 - Advice from Jim Gray of Microsoft and Joe Hellerstein of UC
- Result: Added 7 more dwarfs, revised 2 original dwarfs, renumbered list



13 Dwarfs (so far)

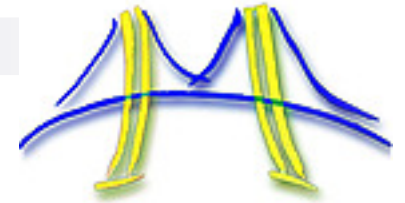
1. Dense Linear Algebra
 2. Sparse Linear Algebra
 3. Spectral Methods
 4. N-Body Methods
 5. Structured Grids
 6. Unstructured Grids
 7. MapReduce
 8. Combinational Logic
 9. Graph Traversal
 10. Dynamic Programming
 11. Back-track/Branch & Bound
 12. Graphical Model Inference
 13. Finite State Machine
- Claim is that parallel architecture, language, compiler ... that do these well will run parallel apps of future well
 - Note: MapReduce is embarrassingly parallel; perhaps FSM is embarrassingly sequential?



Dwarf Popularity (Red Hot → Blue Cool)

	HPC	Embed	SPEC	ML	Games	DB
1 Dense Matrix	Red	Red	Red	Red	Red	Yellow
2 Sparse Matrix	Red	Yellow	Yellow	Red	Red	Light Blue
3 Spectral (FFT)	Red	Yellow	Light Blue	Yellow	Yellow	Light Blue
4 N-Body	Red	Light Blue	Yellow	Light Blue	Yellow	Light Blue
5 Structured Grid	Red	Red	Red	Light Blue	Yellow	Light Blue
6 Unstructured	Red	Light Blue	Light Blue	Yellow	Yellow	Light Blue
7 MapReduce	Red	Light Blue	Green	Red	Light Blue	Red
8 Combinational	Light Blue	Red	Light Blue	Green	Light Blue	Green
9 Graph Traversal	Light Blue	Red	Yellow	Red	Yellow	Yellow
10 Dynamic Prog	Light Blue	Yellow	Light Blue	Red	Light Blue	Red
11 Backtrack/ B&B	Light Blue	Light Blue	Light Blue	Red	Light Blue	Yellow
12 Graphical Models	Light Blue	Light Blue	Light Blue	Red	Light Blue	Yellow
13 FSM	Light Blue	Red	Red	Yellow	Yellow	Red

7 Questions for Parallelism



Applications:

1. What are the apps?
2. What are kernels of apps?

■ **Hardware:**

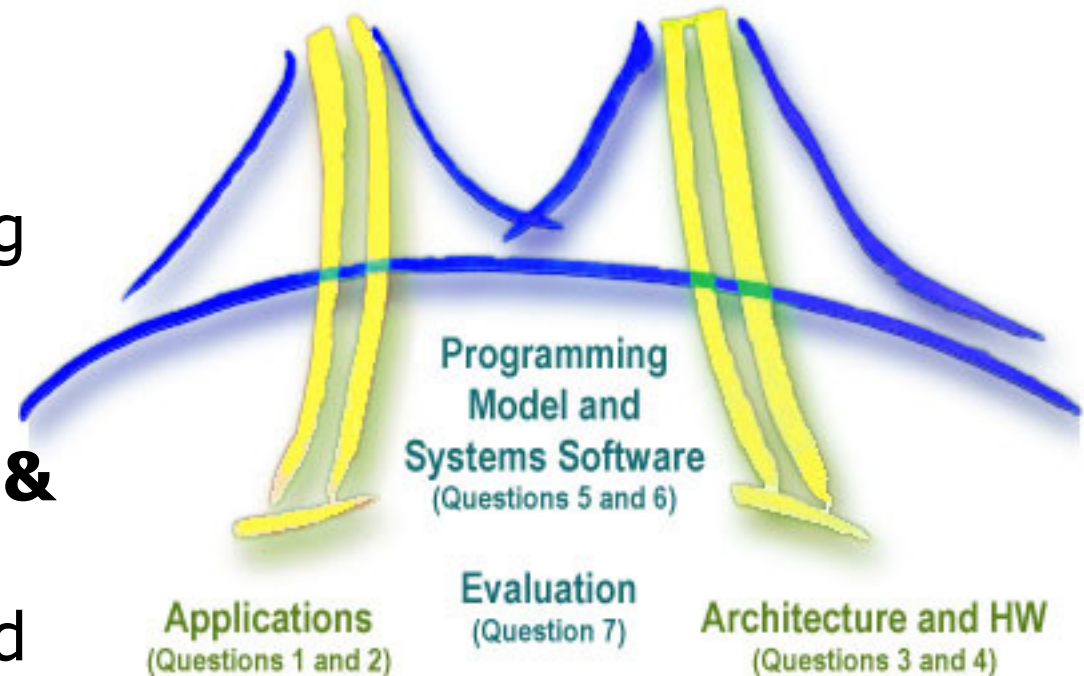
3. What are the HW building blocks?
4. How to connect them?

■ **Programming Model & Systems Software:**

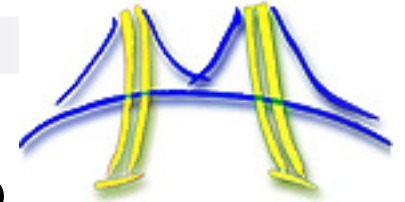
5. How to describe apps and kernels?
6. How to program the HW?

■ **Evaluation:**

7. How to measure success?

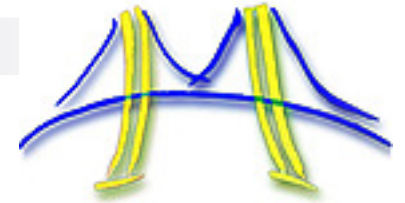


(Inspired by a view of the Golden Gate Bridge from Berkeley)



HW: What are the problems?

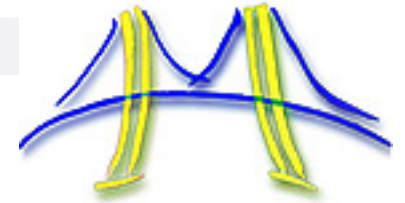
- Power limits leading edge chip designs
 - Intel Tejas Pentium 4 cancelled due to power issues
- Yield on leading edge processes dropping dramatically
 - IBM quotes yields of 10 – 20% on 8-processor Cell
- Design/validation leading edge chip is becoming unmanageable
 - Verification teams > design teams on leading edge processors



HW Solution: Small is Beautiful

- Expect modestly pipelined (5- to 9-stage) CPUs, FPUs, vector, SIMD PEs
 - Small cores not much slower than large cores
- Parallel is energy efficient path to performance: CV^2F
 - Lower threshold and supply voltages lowers energy per op
- Redundant processors can improve chip yield
 - Cisco Metro 188 CPUs + 4 spares; Sun Niagara sells 6 or 8 CPUs
- Small, regular processing elements easier to verify
- One size fits all?
 - Amdahl's Law \Rightarrow Heterogeneous processors?

Heterogeneous Processors?

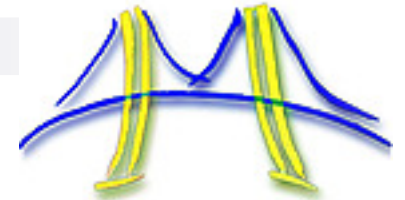


- Suppose to run the code 2X faster 1 core needs 10X resources (power, caches, ...)
- Amdahl's Law: Assume 10% time program gets no faster on manycore computer (e.g. OS)


-geneity?	Slow Cores	Fast Cores	Speedup
Homo-	100	0	9.2
Homo-	0	10	10.5
Hetero-	90	1	16.7

Heterogeneous same area but 1.6X to 1.8X faster

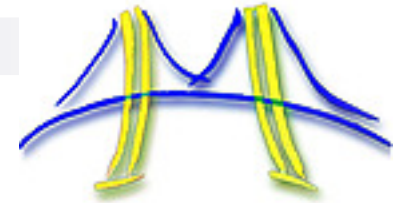
Number of Cores/Socket



- We need revolution, not evolution
- Software or architecture alone can't fix parallel programming problem, need innovations in both
- "Multicore" 2X cores per generation: 2, 4, 8, ...
- "Manycore" 100s is highest performance per unit area, and per Watt, then 2X per generation: 128, 256, 512, 1024 ...
- **Multicore architectures & Programming Models good for 2 to 32 cores won't evolve to Manycore systems of 1000's of processors**
⇒ **Desperately need HW/SW models that work for Manycore or run out of steam | (as ILP ran out of steam at 4 instructions)**

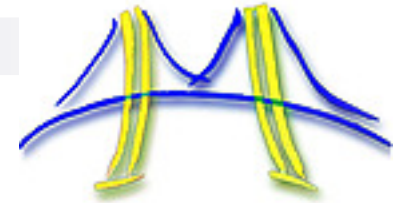


Some obvious (but neglected) recommendations for hardware



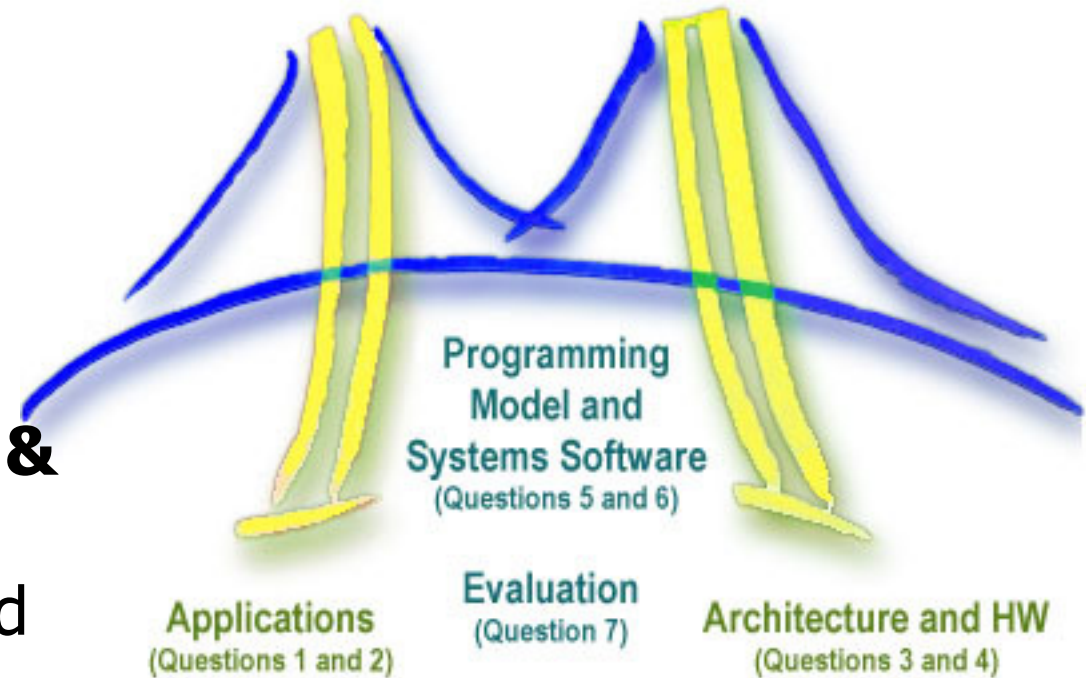
- Counters and other instrumentation more important than in the past
 - Needed for Feedback directed applications
 - Since energy is limit, include energy counters as well as performance counters
- Include counters that work!
 - In past low priority, so ship even if counters broken, or don't slow processor to measure it
 - If can't measure feature, won't use it effectively
- Don't include features that significantly affect performance or energy if programmers cannot accurately measure their impact

7 Questions for Parallelism



Applications:

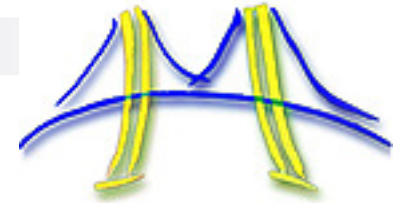
1. What are the apps?
2. What are kernels of apps?
 - Hardware:
3. What are the HW building blocks?
4. How to connect them?
- **Programming Model & Systems Software:**
5. How to describe apps and kernels?
6. How to program the HW?
 - **Evaluation:**
7. How to measure success?



(Inspired by a view of the Golden Gate Bridge from Berkeley)

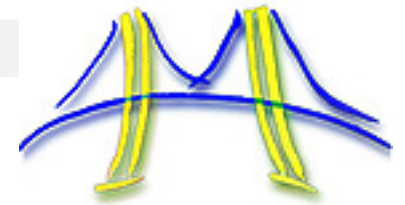


How to Connect Processors?

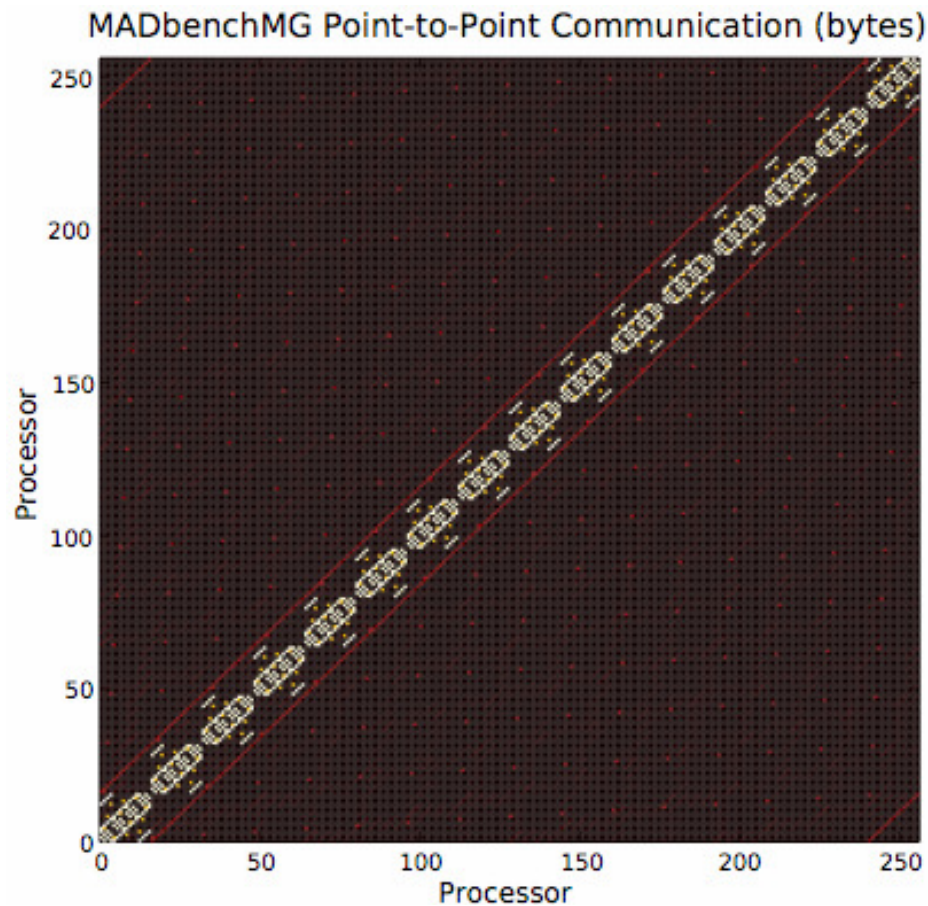


- **Topic wide open! (HW/SW innovations ASAP!)**
- 13 Dwarfs to gain insight into Networks On a Chip
 - Sparse connectivity for dwarfs; crossbar is overkill
 - No single best topology
- A Bandwidth-oriented network for data
 - Most point-to-point message are large and BW bound
- Separate Latency-oriented network for collectives
 - Given BW improves $> (\text{latency improvement})^2$
 - E.g., Thinking Machines CM-5, Cray T3D, IBM BlueGene/L&P
- Virtual circuit switch??
- Synchronization??
 - Transactional memory, full-empty bits, barriers???
 - Is cache coherency all we need to coordinate cores?

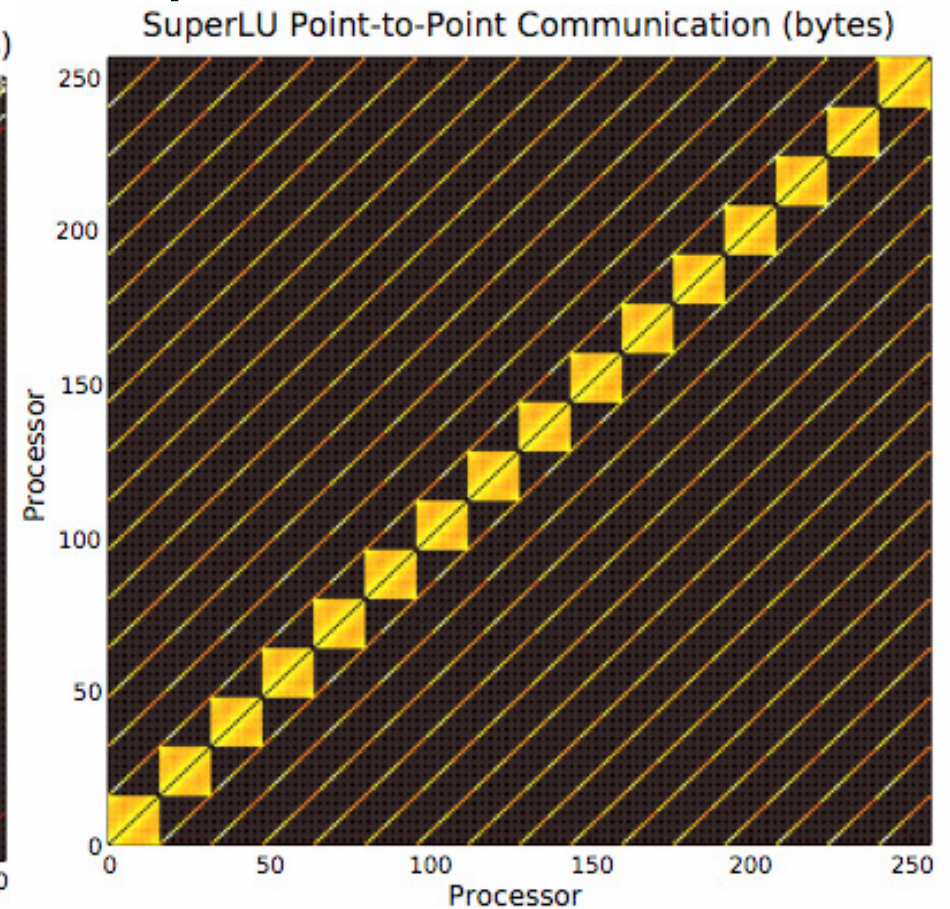
Example: Dwarf Communication Patterns



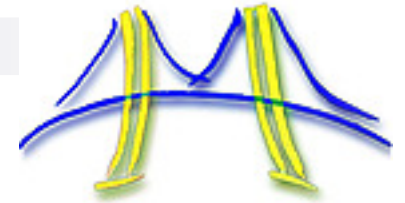
■ Dense Matrix



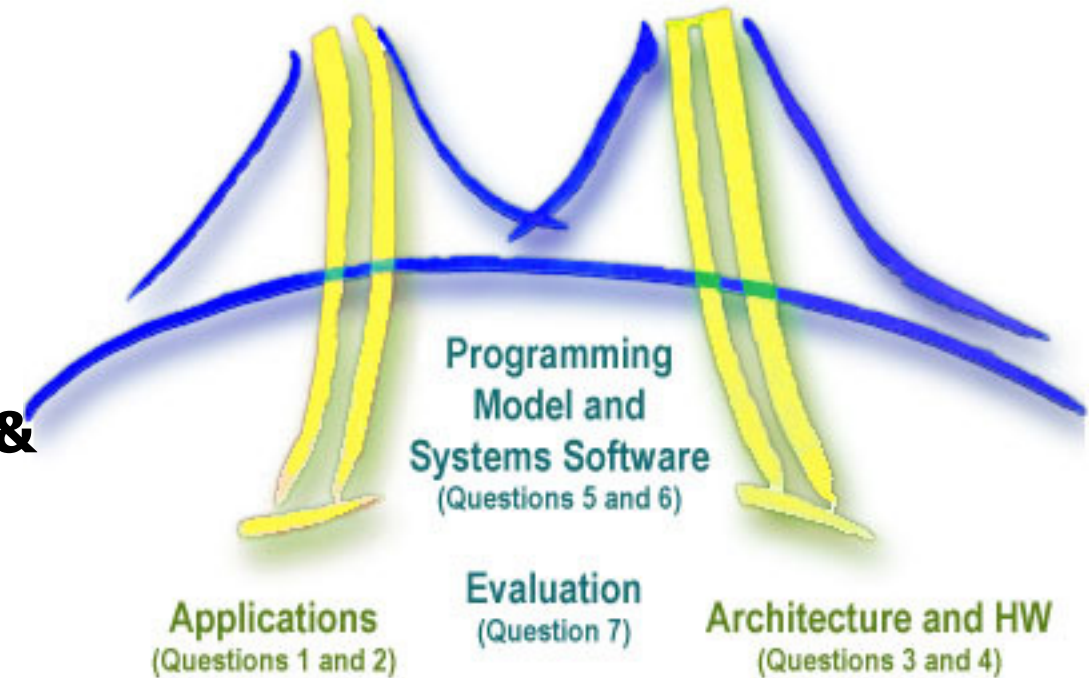
■ Sparse Matrix



7 Questions for Parallelism



- **Applications:**
 1. What are the apps?
 2. What are kernels of apps?
- **Hardware:**
 3. What are the HW building blocks?
 4. How to connect them?
- **Programming Model & Systems Software:**
 5. How to describe apps and kernels?
 6. How to program the HW?
- **Evaluation:**
 7. How to measure success?



(Inspired by a view of the Golden Gate Bridge from Berkeley)



Programming Model

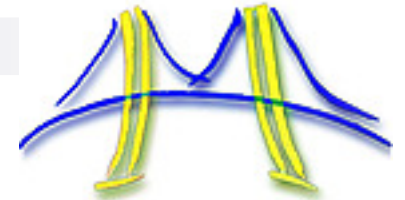
- Programming model must allow programmer to balance competing goals of *productivity* and *implementation efficiency*
 - Biggest challenge facing manycore systems
- ⇒ **Programming Model high priority**
- Past foci of parallel Programming Models:
 1. Hardware-centric (e.g., C-variants)
 2. Application-centric (e.g., MatLab)
 3. Formalism-centric (e.g., Sisal)



Human-centric Programming Model

- Obviously, success of models strongly affected by human beings who use them
- Efficiency, productivity, correctness important, but no impact of research from psychology???
 - Used in HCI, but not in language design or SW engineering
 - E.g., a rich theory of human errors [Kantowitz and Sorkin 1983] [Reason 1990]
- E.g., Transactional Memory helps with human errors
 - System ensures correctness even when programmers make incorrect assumptions about safety of parallelizing code
- Integrating research on human psychology and problem solving critical to developing successful parallel Programming Model and environments

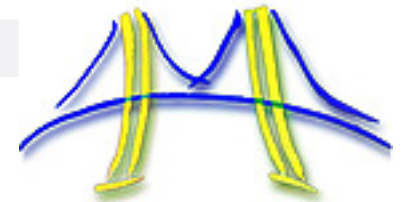
Testing Human-centric Models



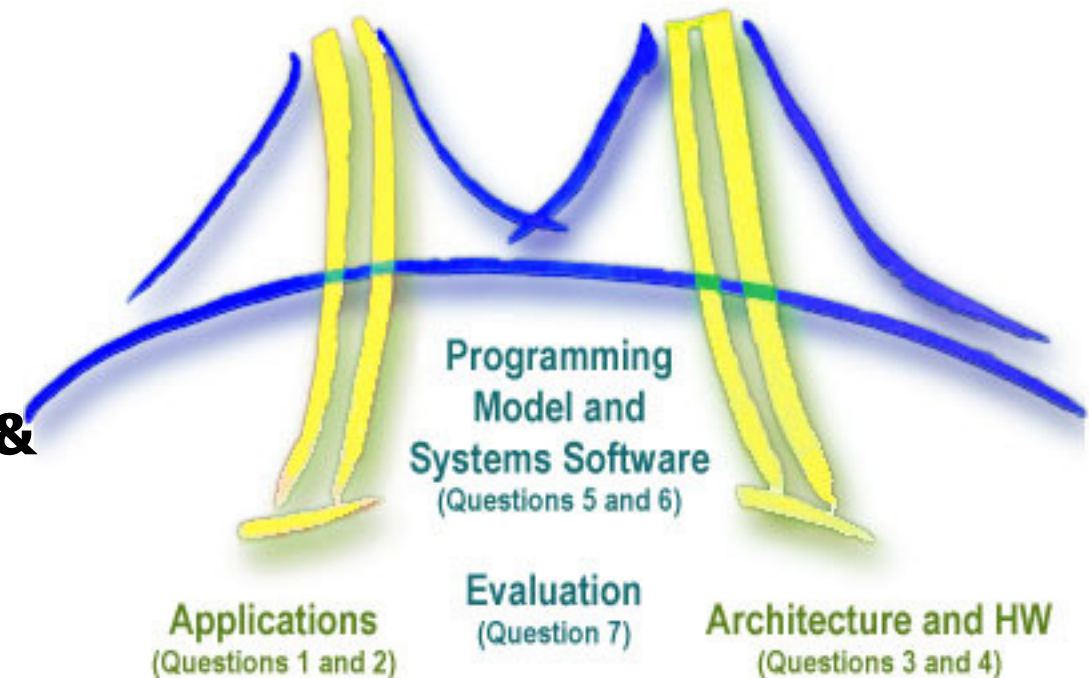
- Follow psychological method to resolve controversies: Human-subject experiments
 - E.g., Efficiency/productivity of small programs of shared memory vs. message passing by novice programmers: OpenMP (shared memory) >> MPI (message passing)*
- To make real progress in programming manycore systems efficiently, use human subject experiments to resolve open issues vs. maintaining strong opinions

* L. Hochstein, J. Carver, F. Shull, S. Asgari, V.R. Basili, J.K. Hollingsworth, M. Zelkowitz. "Parallel Programmer Productivity: A Case Study of Novice Parallel Programmers," *International Conference for High Performance Computing, Networking and Storage (SC'05)*. Nov. 2005.

7 Questions for Parallelism

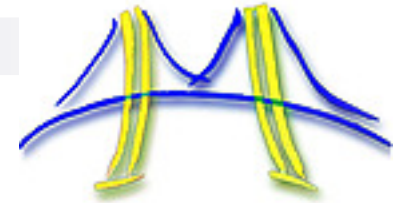


- **Applications:**
 1. What are the apps?
 2. What are kernels of apps?
- **Hardware:**
 3. What are the HW building blocks?
 4. How to connect them?
- **Programming Model & Systems Software:**
 5. How to describe apps and kernels?
 6. How to program the HW?
- **Evaluation:**
 7. How to measure success?



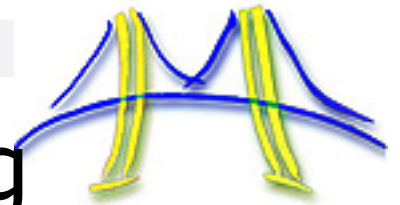
(Inspired by a view of the Golden Gate Bridge from Berkeley)

21st Century Code Generation



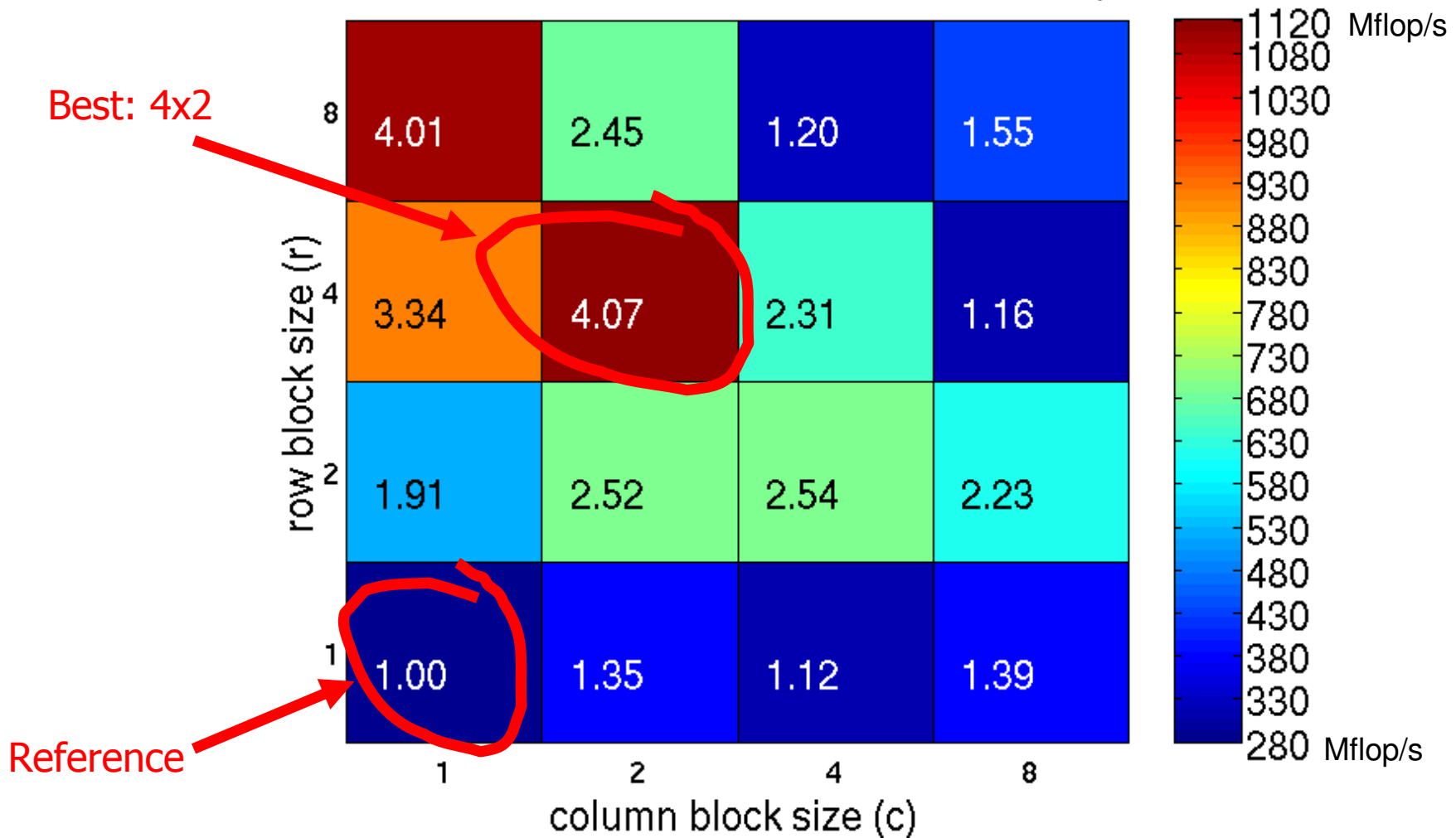
- Takes a decade for compiler innovations to show up in production compilers?
- New approach: “**Auto-tuners**” 1st run variations of program on computer to find best combinations of optimizations (blocking, padding, ...) and algorithms, then produce C code to be compiled for *that* computer
 - E.g., PHiPAC (BLAS), Atlas (BLAS), Sparsity (Sparse linear algebra), Spiral (DSP), FFT-W
 - Can achieve 10X over conventional compiler
- One Auto-tuner per kernel or dwarf?
 - Exist for Dense Linear Algebra, Sparse Linear Algebra, Spectral

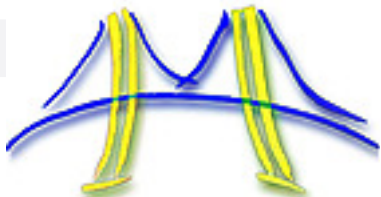

Sparse Matrix – Search for Blocking



For finite element problem (BCSR) [Im, Yelick, Vuduc, 2005]

900 MHz Itanium 2, Intel C v8: ref=275 Mflop/s





Some obvious (but neglected) recommendations for models

- Be independent of number of processors
- Support of proven styles of parallelism
 - Task/Thread-level parallelism, data-level parallelism, and bit/gate-level parallelism
- Have richer set of data types/sizes than were found in 40-year old IBM 360 architecture!
 - 1 bit (Boolean)
 - 8 bits (Integer, ASCII)
 - 16 bits (Integer, DSP fixed point, Unicode)
 - 32 bits (Integer, Single-precision floating point, Unicode)
 - 64 bits (Integer, Double-precision floating point)
 - 128 bits (Integer, Quad-Precision floating point)
 - Large integer (>128 bits) (Crypto)

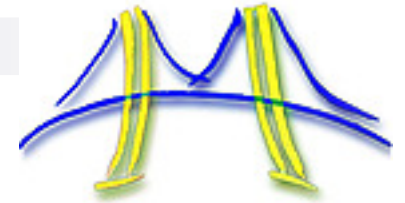


Deconstructing Operating Systems



- Resurgence of interest in virtual machines
 - Traditional OSes brittle & too large (AIX GBs DRAM)
 - VM monitor thin SW layer btw guest OS and HW
- Advantages
 - Security via isolation
 - VMs move from failing processor
- Mendel Rosenblum: future of OSes could be libraries where only functions needed are linked into app, on top of thin VMM layer providing protection and sharing of resources
 - Everywhere, but great match to 1000s of processors

7 Questions for Parallelism



■ Applications:

1. What are the apps?
2. What are kernels of apps?

■ Hardware:

3. What are the HW building blocks?
4. How to connect them?

4. How to connect them?

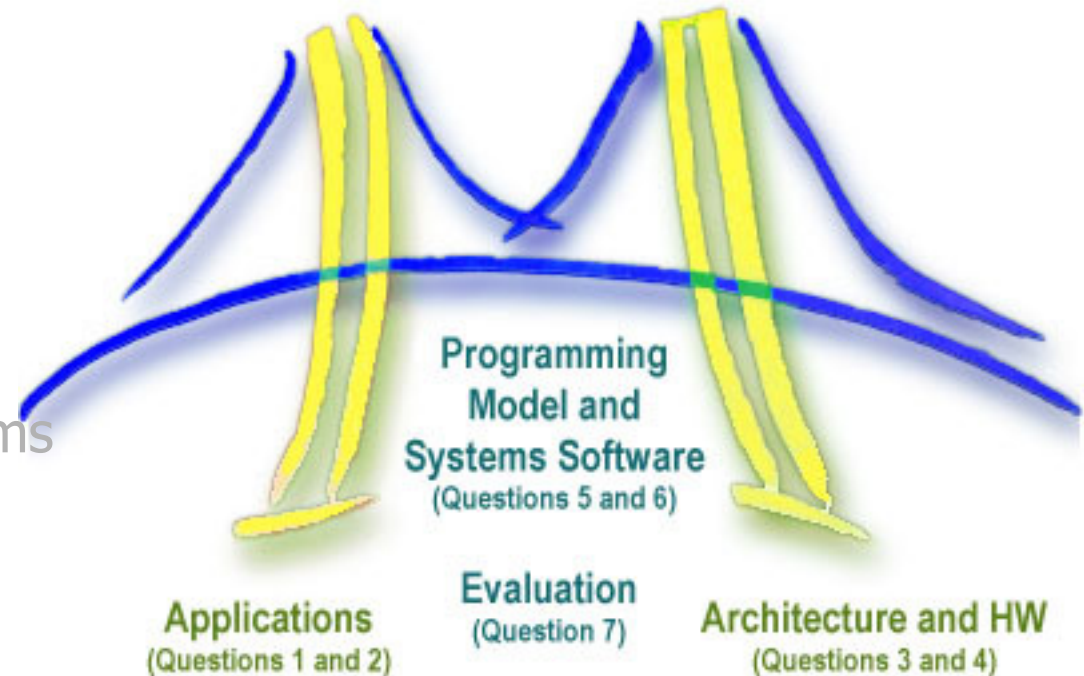
Programming Model & Systems Software:

5. How to describe apps and kernels?
6. How to program the HW?

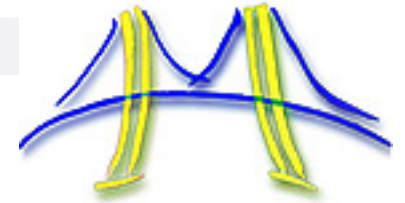
6. How to program the HW?

■ Evaluation:

7. How to measure success?

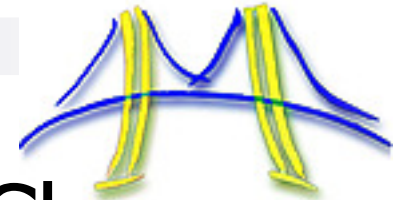


(Inspired by a view of the Golden Gate Bridge from Berkeley)



How to measure success?

- Easy to write programs that execute efficiently on manycore computing systems
 1. Maximizing programmer productivity
 2. Maximizing application performance and energy efficiency
- Challenges
 - Conventional Serial Performance Issues
 - Minimizing Remote Accesses
 - Balancing Load
 - Granularity of Data Movement and Synchronization



Problems with “Manycore” Sea Change

1. Algorithms, Programming Languages, Compilers, Operating Systems, Architectures, Libraries, ... not ready for 1000 CPUs / chip
2. \approx Only companies can build HW, and it takes years
3. Software people don't start working hard until hardware arrives
 - 3 months after HW arrives, SW people list everything that must be fixed, then we all wait 4 years for next iteration of HW/SW
4. How get 1000 CPU systems in hands of researchers to innovate in timely fashion on in algorithms, compilers, languages, OS, architectures, ... ?
5. Can avoid waiting years between HW/SW iterations?

Build Academic Manycore from FPGAs

- As ≈ 16 CPUs will fit in Field Programmable Gate Array (FPGA), 1000-CPU system from ≈ 64 FPGAs?
 - 8 32-bit simple “soft core” RISC at 100MHz in 2004 (Virtex-II)
 - FPGA generations every 1.5 yrs; $\approx 2X$ CPUs, $\approx 1.2X$ clock rate
- HW research community does logic design (“gate shareware”) to create out-of-the-box, Manycore
 - E.g., 1000 processor, standard ISA binary-compatible, 64-bit, cache-coherent supercomputer @ ≈ 150 MHz/CPU in 2007
 - RAMPants: 10 faculty at Berkeley, CMU, MIT, Stanford, Texas, and Washington
- “Research Accelerator for Multiple Processors” as a vehicle to attract many to parallel challenge

Why Good for Research Manycore?



	SMP	Cluster	Simulate	RAMP
Scalability (1k CPUs)	C	A	A	A
Cost (1k CPUs)	F (\$40M)	C (\$2-3M)	A+ (\$0M)	A (\$0.1-0.2M)
Cost of ownership	A	D	A	A
Power/Space (kilowatts, racks)	D (120 kw, 12 racks)	D (120 kw, 12 racks)	A+ (.1 kw, 0.1 racks)	A (1.5 kw, 0.3 racks)
Community	D	A	A	A
Observability	D	C	A+	A+
Reproducibility	B	D	A+	A+
Reconfigurability	D	C	A+	A+
Credibility	A+	A+	F	B+/A-
Perform. (clock)	A (2 GHz)	A (3 GHz)	F (0 GHz)	C (0.1 GHz)
GPA	C	B-	B	A-

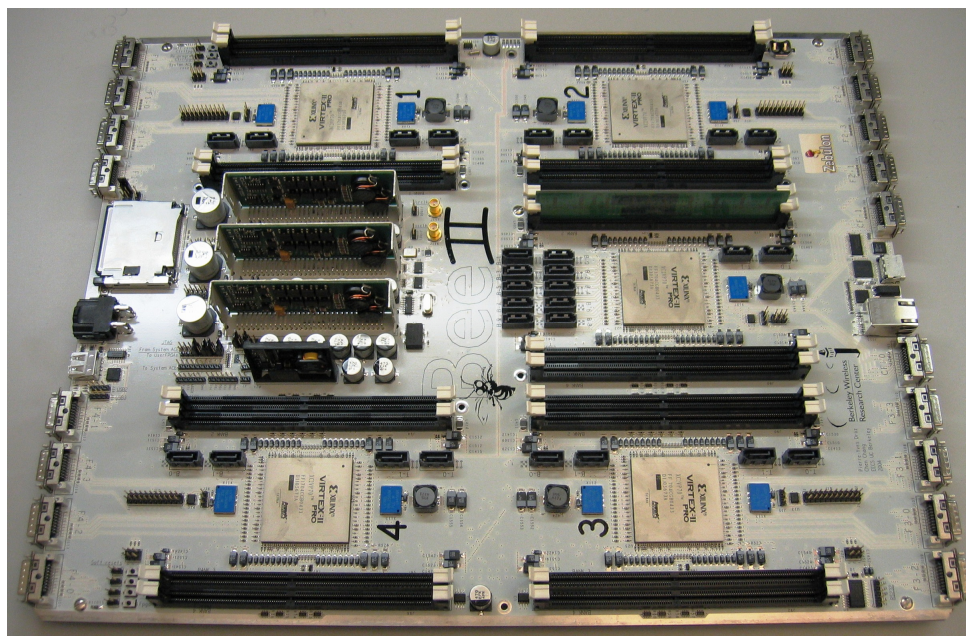
RAMP 1 Hardware

- Completed Dec. 2004 (14x17 inch 22-layer PCB)

Board:

5 Virtex II FPGAs, 18 banks DDR2-400 memory, 20 10GigE conn.

\$10,000 / board



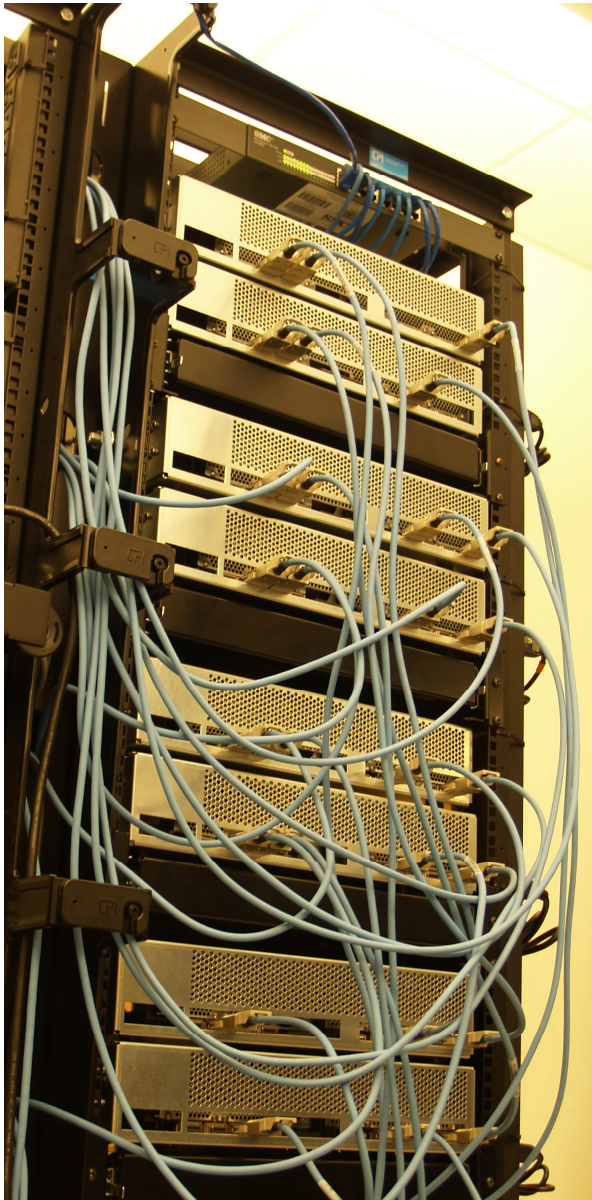
BEE2: Berkeley Emulation Engine 2

By John Wawrzynek and Bob Brodersen with students Chen Chang and Pierre Droz

Transactional Memory / RAMP Red

- 8 CPUs with 32KB L1 data-cache with Transactional Memory support (Kozyrakis, Olukotun... at Stanford)
 - CPUs are hardcoded PowerPC405, Emulated FPU
 - UMA access to shared memory (no L2 yet)
 - Caches and memory operate at 100MHz
 - Links between FPGAs run at 200MHz
 - CPUs operate at 300MHz
- A separate, 9th, processor runs OS (PowerPC Linux)
- It works: runs SPLASH-2 benchmarks, AI apps, C-version of SpecJBB2000 (3-tier-like benchmark)
- **1st Transactional Memory Computer!**
- **Transactional Memory RAMP runs 100x faster than simulator on a Apple 2GHz G5 (PowerPC)**

256 CPU Message Passing/RAMP Blue



- 8 MicroBlaze cores / FPGA
- 8 BEE2 modules (32 “user” FPGAs) x 4 FPGAs/module = 256 cores @ 100MHz
 - \$10k/board
- Full star-connection between modules
- It works; runs NAS benchmarks in UPC
- Cores are softcore MicroBlazes (32-bit Xilinx RISC)
- Schultz, Krasnov, Wawrzynek at Berkeley

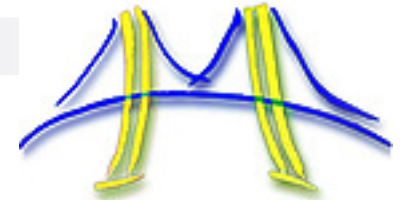
Multiprocessing Watering Hole



Parallel file system Dataflow language/computer Data center in a box
Fault insertion to check dependability Router design Compile to FPGA
Flight Data Recorder Security enhancements Transactional Memory
Internet in a box 128-bit Floating Point Libraries Parallel languages

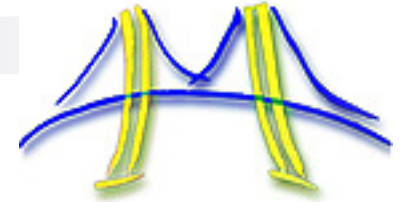
- Killer app: \approx All CS Research, Advanced Development
- RAMP attracts many communities to shared artifact
⇒ Cross-disciplinary interactions
- RAMP as next Standard Research/AD Platform?
(e.g., VAX/BSD Unix in 1980s)

Reasons for Optimism towards Parallel Revolution this time



- End of sequential microprocessor/faster clock rates
 - No looming sequential juggernaut to kill parallel revolution
- SW & HW industries fully committed to parallelism
 - End of La-Z-Boy Programming Era
- Moore's Law continues, so soon can put 1000s of simple cores on an economical chip
- Communication between cores within a chip at very low latency and very high bandwidth
 - Processor-to-Processor fast even if Memory slow
- All cores equal distance to shared main memory
 - Less data distribution challenges
- Open Source Software movement means that SW stack can evolve more quickly than in past
- RAMP as vehicle to ramp up parallel research



Summary



- 13 Dwarfs as stand-ins for future parallel apps
 - Important patterns of computation & communication
- Simple processors! Manycore beats Multicore
 - Most efficient MIPS/watt, MIPS/area, MIPS/development \$
 - Multicore (2-32) solutions may fail at Manycore (250-1000)
- Goal: easy-to-program, efficient manycore
- Need human-centric programming models
- Use Autotuners and Deconstructed OSes on VMs
- Use RAMP to accelerate HW/SW generations
 - FPGAs to emulate + Architects aid colleagues via gateware
- If you liked the movie, read the book:
“The Landscape of Parallel Computing Research: A View from Berkeley,” UCB TR EECS-2006-183, 12/18/06



Change directions of research funding?

Historically:
Get leading
experts per
discipline
(across US)
working
together
to work on
parallelism

		CMU		Stanford	...
Application					
Language					
Compiler					
Libraries					
Networks					
Architecture					
Hardware					
CAD					

Change directions of research funding?

To increase cross-disciplinary bandwidth, get experts **per site** working together on parallelism

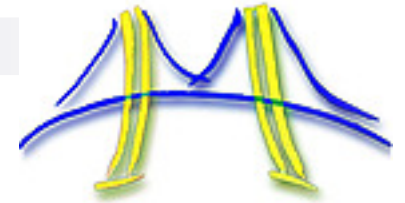
		CMU		Stanford	.
Application					
Language					
Compiler					
Libraries					
Networks					
Architecture					
Hardware					
CAD					



Where to go from here?

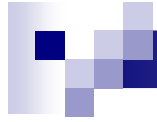
- What bold new applications will manycore enable?
- Can we design architectures that make parallel programming easier?
- Can we develop highly-productive programming models that harness the performance of manycore?
- Berkeley and Stanford are ideal places to do the cross-disciplinary research needed to save the IT industry's desperate bet on parallelism

Acknowledgments



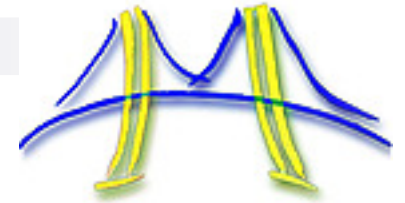
- Berkeley View material comes from discussions with:
 - Professors Krste Asanović, Ras Bodik, Jim Demmel, Kurt Keutzer, John Wawrzynek, and Kathy Yelick
 - UCB Grad students Bryan Catanzaro, Jike Chong, Joe Gebis, William Plishker, and Sam Williams
 - LBNL: Parry Husbands, Bill Kramer, Lenny Oliker, John Shalf
- **See view.eecs.berkeley.edu**
- RAMP based on work of RAMP Developers:
 - Krste Asanović (MIT/Berkeley), Derek Chiou (Texas), James Hoe (CMU), Christos Kozyrakis (Stanford), Shih-Lien Lu (Intel), Mark Oskin (Washington), David Patterson (Berkeley, Co-PI), and John Wawrzynek (Berkeley, PI)
- **See ramp.eecs.berkeley.edu**

RAMP



Backup Slides for Q&A

Really 1000s of processors/chip?



- Cisco Metro 188 cores in 130 nm technology
 - 18mm by 18mm, dissipates 35W at a 250MHz clock rate
 - Aggregate 50 billion instructions per second
 - Embedded in routers
- 5-stage pipelined Tensilica processors with very small caches, size/core is 0.5 mm²
- Scaling from Moore's Law ⇒
752 cores in 45 nm and 1504 cores in 30 nm
 - Intel has taped out a 45-nm technology chip in 2006

Example: Vary memory latency, BW

- Target system: TPC-C, Oracle, Linux on 1024 CPUs @ 2 GHz, 64 KB L1 I\$ & D\$/CPU, 16 CPUs share 0.5 MB L2\$, shared 128 MB L3\$
 - Latency: L1 1 - 2 cycles, L2 8 - 12 cycles, L3 20 - 30 cycles, DRAM 200 - 400 cycles
 - Bandwidth: L1 8 - 16 GB/s, L2 16 - 32 GB/s, L3 32 - 64 GB/s, DRAM 16 - 24 GB/s per port, 16 - 32 DDR3 128b memory ports
- Host system: TPC-C, Oracle, Linux on 1024 CPUs @ 0.1 GHz, 32 KB L1 I\$, 16 KB D\$
 - Latency: L1 1 cycle, DRAM 2 cycles
 - Bandwidth: L1 0.1 GB/s, DRAM 3 GB/s per port, 128 64b DDR2 ports
 - Use cache models and DRAM to emulate L1\$, L2\$, L3\$ behavior

Accurate Clock Cycle Accounting

- Key to RAMP success is cycle-accurate emulation of parameterized target design
 - As vary number of CPUs, CPU clock rate, cache size and organization, memory latency & BW, interconnect latency & BW, disk latency & BW, Network Interface Card latency & BW, ...
 - Least common divisor time unit to drive emulation?
 - 1. For research results to be credible
 - 2. To run standard, shrink-wrapped OS, DB, ...
 - Otherwise fake interrupt times since devices relatively too fast
- ⇒ Good target clock cycle accounting is high priority for RAMP project

RAMP Philosophy

- Build vanilla out-of-the-box examples to attract software community
 - Multiple industrial ISAs, real industrial operating systems, 1000 processors, accurate clock cycle accounting, reproducible, traceable, parameterizable, cheap to buy and operate, ...
- But RAMPants have grander plans (will share)
 - Data flow computer ("Wavescalar") – Oskin @ U. Washington
 - 1,000,000-way MP ("Transactors") – Asanovic @ MIT
 - Distributed Data Centers ("RAD Lab") – Patterson @ Berkeley
 - Transactional Memory ("TCC") – Kozyrakis @ Stanford
 - Reliable Multiprocessors ("PROTOFLEX") – Hoe @ CMU
 - X86 emulation ("UT FAST") – Chiou @ Texas
 - Signal Processing in FPGAs ("BEE2") – Wawrzynek @ Berkeley

Related Approaches

■ Quickturn, Axis, IKOS, Thara:

- FPGA- or special-processor based gate-level hardware emulators
- HDL mapped to array for cycle and bit-accurate netlist emulation
- No DRAM memory since modeling CPU, not system
- Doesn't worry about speed of logic synthesis: 1 MHz clock
- Uses small FPGAs since takes many chips/CPU, and pin-limited
- Expensive: \$5M

■ RAMP's emphasis is on emulating high-level system behaviors

- More DRAMs than FPGAs: BEE2 has 5 FPGAs, 96 DRAM chips
- Clock rate affects emulation time: >100 MHz clock
- Uses biggest FGPAs, since many CPUs/chip
- Affordable: \$0.1 M



RAMP's Potential Beyond Manycore *RAMP*

- **Attractive Experimental Systems Platform:
Standard ISA + standard OS + modifiable
+ fast enough + trace/measure anything**
 - Generate long traces of full stack: App, VM, OS, ...
 - Test hardware security enhancements in the wild
 - Inserting faults to test availability schemes
 - Test design of switches and routers
 - SW Libraries for 128-bit floating point
 - App-specific instruction extensions (\approx Tensilica)
 - Alternative Data Center designs
 - Akamai vs. Google: N centers of M computers

Potential to Accelerate Manycore

- **With RAMP: Fast, wide-ranging exploration of HW/SW options + head-to-head competitions to determine winners and losers**
 - Common artifact for HW and SW researchers ⇒ innovate across HW/SW boundaries
 - Minutes vs. years between “HW generations”
 - Cheap, small, low power ⇒ Every dept owns one
 - FTP supercomputer overnight, check claims locally
 - Emulate any Manycore ⇒ aid to teaching parallelism
 - If HP, IBM, Intel, M/S, Sun, ...had RAMP boxes
 - ⇒ Easier to carefully evaluate research claims
 - ⇒ Help technology transfer
- **Without RAMP: One Best Shot + Field of Dreams?**

RAMP Supporters:

- Gordon Bell (Microsoft)
- Ivo Bolsens (Xilinx CTO)
- Jan Gray (Microsoft)
- Norm Jouppi (HP Labs)
- Bill Kramer (NERSC/LBL)
- Konrad Lai (Intel)
- Craig Mundie (MS CTO)
- Jaime Moreno (IBM)
- G. Papadopoulos (Sun CTO)
- Jim Peek (Sun)
- Justin Rattner (Intel CTO)
- Michael Rosenfield (IBM)
- Tanaz Sowdagar (IBM)
- Ivan Sutherland (Sun Fellow)
- Chuck Thacker (Microsoft)
- Kees Vissers (Xilinx)
- Jeff Welser (IBM)
- David Yen (Sun EVP)
- *Doug Burger (Texas)*
- *Bill Dally (Stanford)*
- *Susan Eggers (Washington)*
- *Kathy Yelick (Berkeley)*

RAMP Participants: Arvind (MIT), Krste Asanović, Derek Chiou (Texas), James Hoe (CMU), Christos Kozyrakis (Stanford), Shih-Lien Lu (Intel), Mark Oskin (Washington), David Patterson (Berkeley, Co-PI), Jan Rabaey (Berkeley), and John Wawrzynek (Berkeley, PI)

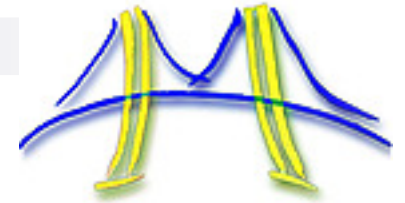
RAMP Project Status

- NSF infrastructure grant awarded 3/06
 - 2 staff positions (NSF sponsored), no grad students
- IBM Faculty Awards to RAMPants 6/06
 - Krste Asanovic (MIT), Derek Chiou (Texas), James Hoe (CMU), Christos Kozyrakis (Stanford), John Wawrzynek (Berkeley)
- 3-day retreats with industry visitors
 - "Berkeley-style" retreats 1/06 (Berkeley), 6/06 (ISCA/Boston), 1/07 (Berkeley), 6/07 (ISCA/San Diego)
- RAMP 1/RDL short course
 - 40 people from 6 schools 1/06

Why RAMP More Credible?

- Starting point for processor is debugged design from Industry in HDL
- Fast enough that can run more software, more experiments than simulators
- Design flow, CAD similar to real hardware
 - Logic synthesis, place and route, timing analysis
- HDL units implement operation vs. a high-level description of function
 - Model queuing delays at buffers by building real buffers
- Must work well enough to run OS
 - Can't go backwards in time, which simulators can unintentionally
- Can measure anything as sanity checks

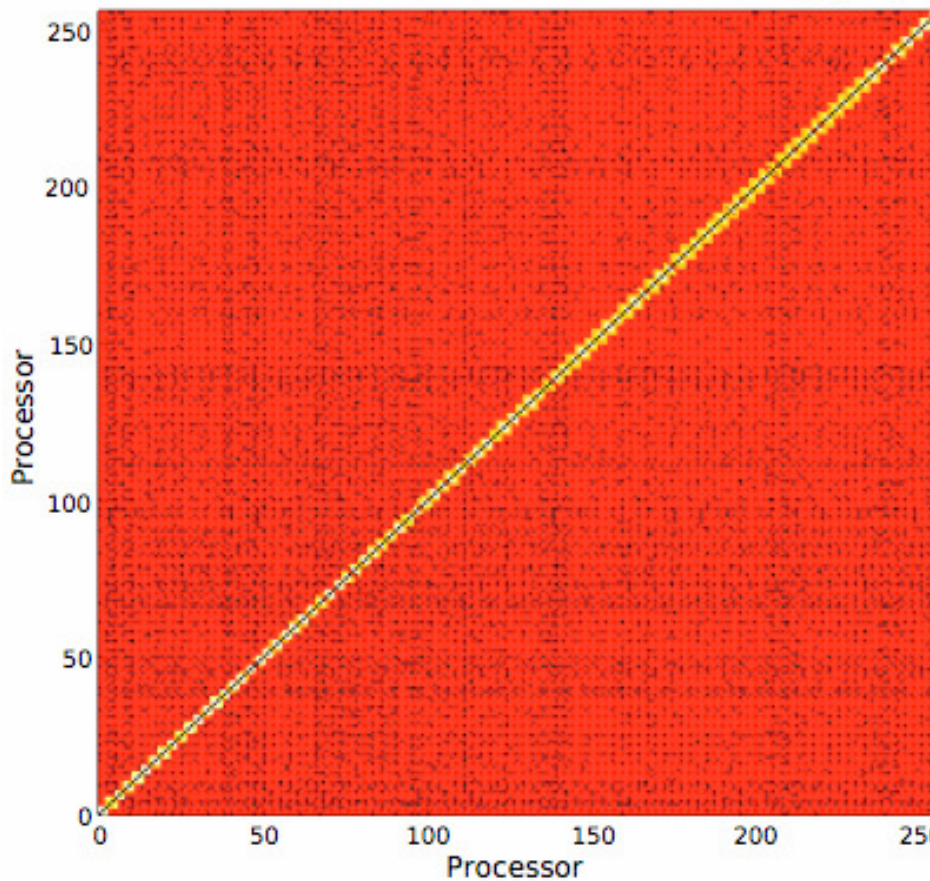
Dwarf Communication Patterns



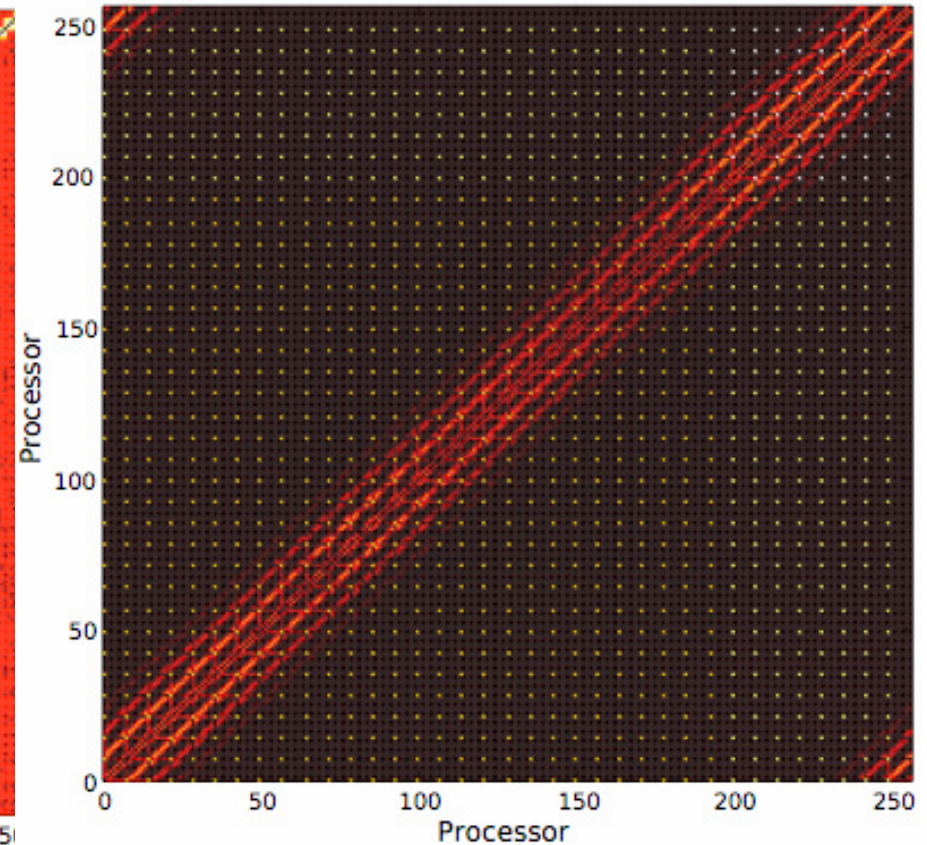
■ Spectral (e.g., FFT)

■ N-Body Methods

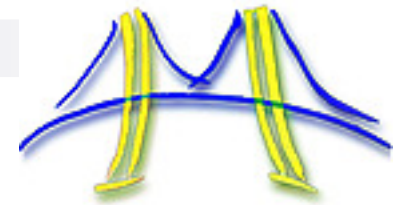
PARATEC Point-to-Point Communication (bytes)



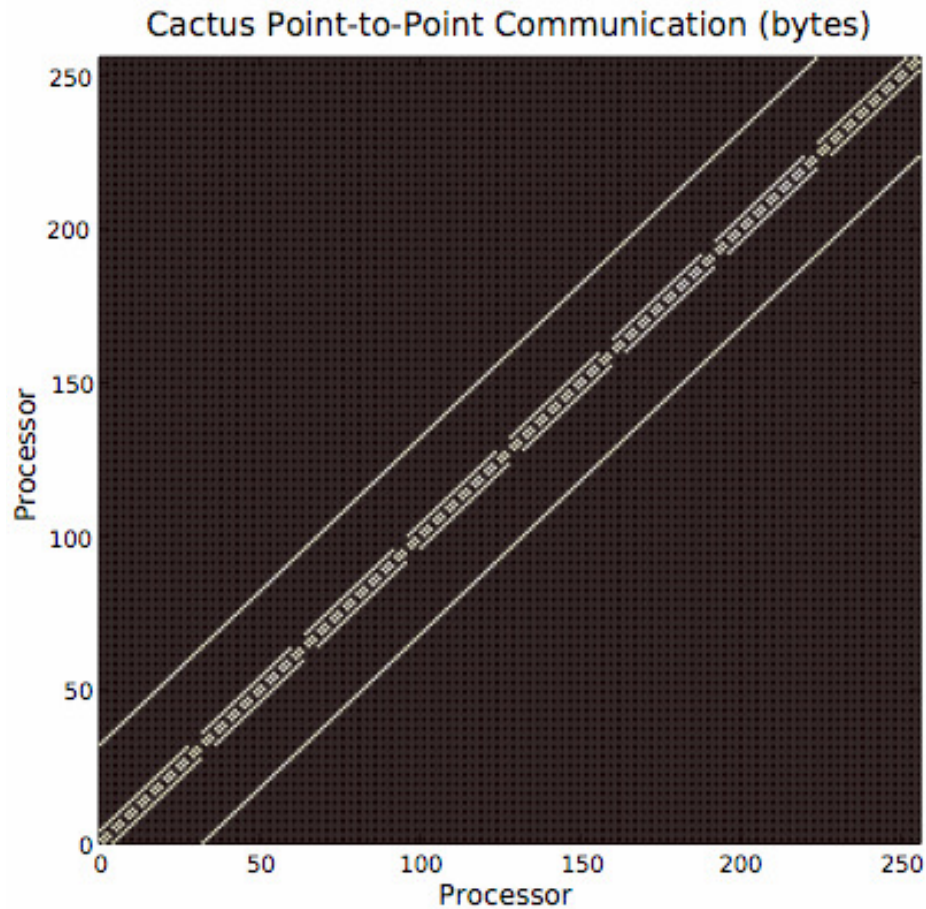
PMEMD Point-to-Point Communication (bytes)



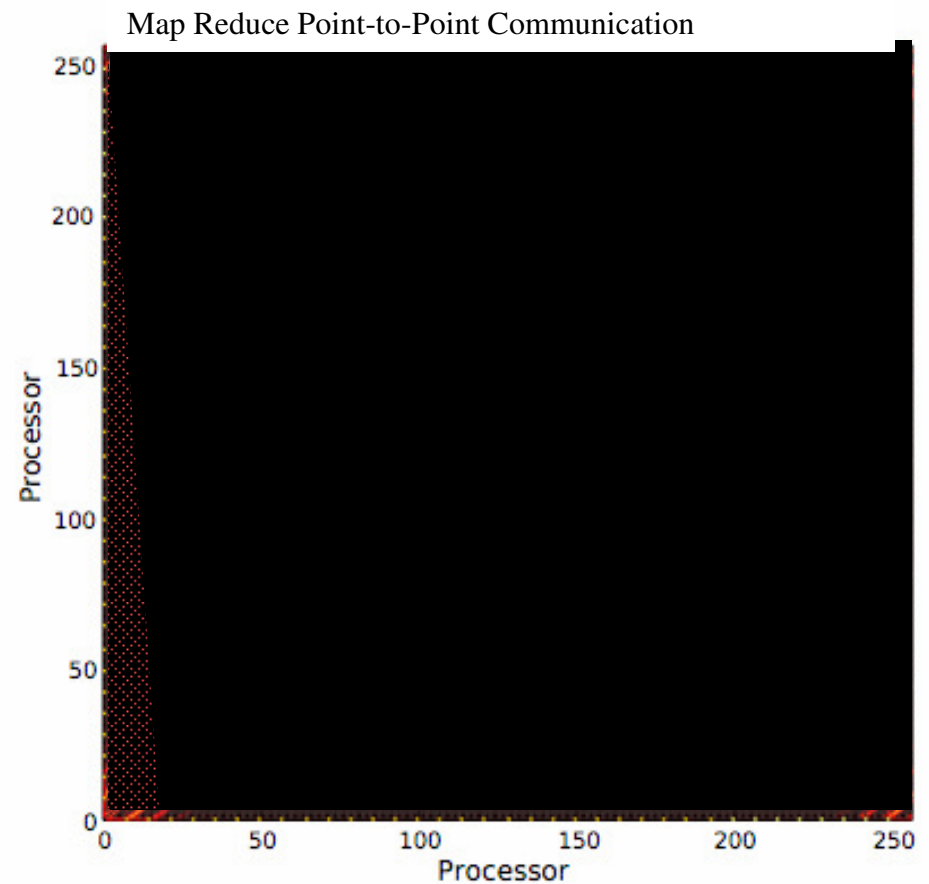
Dwarf Communication Patterns



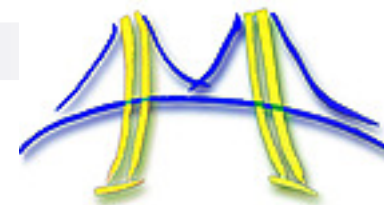
■ Structured Grid



■ MapReduce



Best Sparse Blocking for 8 Computers



		Intel Pentium M		Sun Ultra 2, Sun Ultra 3, AMD Opteron
8				
4	IBM Power 4, Intel/HP Itanium	Intel/HP Itanium 2	IBM Power 3	
2				
1				
	1	2	4	8
	column block size (c)			

- All possible column block sizes selected for 8 computers; How could compiler know?