Can Computer Architecture Affect Scientific Productivity?

David A. Patterson
Pardee Professor of Comp. Sci., U.C. Berkeley
President, Association for Computing Machinery

April 19, 2005
The Salishan Conference on High-speed Computing

Scientific Productivity Defined For This Talk

1. Hundreds of computing professionals can make a few codes run faster on brand new large scale computer vs. last one
2. Scientists themselves can make many codes run efficiently on new computers

Conventional Wisdom in Computer Architecture

- Cost per socket goes up in large scale system
  - Interconnection network more expensive for good 1000-way system than good 100-way systems
  - Good bandwidth, good latency, avoid hot spots
- Yet sell more small systems than big systems
- Supercomputing market too small to justify large investment for high-end custom chips, systems

Conventional Wisdom in Computer Architecture

- 1960s Conventional Wisdom (CW):
  Hardware is hard to change, Software is flexible
- New CW:
  Hardware is flexible, Software is hard to change
- Old CW: Commodity microprocessors have fixed instruction sets that can't be changed
- New CW: Commodity microprocessors backwards compatible, but expand instruction set regularly
  - Since 1997, Intel added 36 instructions per year to IA-32, or 3 instructions per month!

Conventional Wisdom in Computer Architecture

- Old CW: Parallel buses connect components
- New CW: Serial lines + switches connect them
  - Network (finally) connected directed to microprocessor, lowering HW communication overhead
  - PCI Express @ Intel, Hypertransport @ AMD, Ethernet @ Sun Niagra, …
- Old CW: Performance improves uniformly
- New CW:
  Bandwidth improvement > (latency improvement)^2

To learn more, see "Latency. Lags Bandwidth" David Patterson, CACM (Oct. 2004)

Conventional Wisdom in Computer Architecture

- Old CW: Power is free, Transistors are expensive
- New CW: Power is expensive, Transistors are free
  - Can put more on chip than can afford to turn on
- Old CW: MPU manufacturers don’t want new ideas
- New CW: Power wall + Memory wall = Brick wall
  - New idea receptive environment
- Old CW: Uniprocessor performance 2X / 1.5 yrs
- New CW: 2X CPUs per socket / ~ 3 to 4 years
  - More simpler processors more power efficient

Conventional Wisdom in Computer Architecture

- Old CW: Researchers can't build hardware in a timely fashion that would convince others
- New CW: FPGA, open source HW
  $\Rightarrow$ rapid development and validation of innovative systems even if run ~ 10X to 30X slower
- Old CW: Only 2 or 3 companies can build state-of-the-art microprocessors
- New CW: Semiconductor technology slowing
  $\Rightarrow$ longer generations $\Rightarrow$ more use before old
  + not clear what to build

Conventional Wisdom in Scientific Programming

- Old CW: Dense linear algebra ‘R’ Us
- New CW: Future computing is 7 dwarves + Search, Sort, and Event-Driven simulation?
  - Trends in evolution of 7 dwarves
    - Dense $\Rightarrow$ Sparse $\Rightarrow$ Adaptive
    - SIMD $\Rightarrow$ More elaborate control flow
      - One big linear system $\Rightarrow$ Put together lots of small linear systems
  - Different fields simultaneously at different levels of sophistication on trend line
    - Biology on left, Physics on right?
- Advance computer design by looking forward (7 ± 3 dwarves) vs. looking backward (SPEC200x)

Horst Simon, William Kramer, et al, Science-Driven System Architecture: A New Process for Leadership Class Computing, Journal of the Earth Simulator, Vol 2, p 1-9, January 2005.

Phillip Colella's "Seven dwarves"

High-end simulation in the physical sciences consists of seven algorithms:

1. Structured Grids (including locally structured grids, e.g. AMR)
2. Unstructured Grids
3. Fast Fourier Transform
4. Dense Linear Algebra
5. Sparse Linear Algebra
6. Particles
7. Monte Carlo

Well-defined targets from algorithmic and software standpoint.

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

Conventional Wisdom in Scientific Programming

- Old CW: Programming is hard
- New CW: Parallel programming is really hard
  - 2 kinds of Scientific Programmers
    - Those using single processor
    - Those who can use up to 100 processors
  - Big steps for programmers
    - From 1 processor to 2 processors
    - From 100 processors to 1000 processors
  - Can computer architecture make many processors look like fewer processors, ideally one?
- Old CW: Who cares about I/O in Supercomputing?
- New CW: Supercomputing
  = Masssive data + Massive Computation

Organize rest of this talk? My Most Popular Talks

- “How to Give a Bad Talk”
- “How to Have a Bad Career”
- And so today’s subtitle is

Outline

- CW in Technology and Scientific Programming
- **Part I: Key Advice for Computer Architects Wishing to Hurt Scientific Productivity**
  - Aim High (& Ignore Amdahl's Law)
  - Promote Mystery (& Hide Thy Real Performance)
  - Be "Interesting" (& Have a Quirky Personality)
  - Accuracy and Reliability are for Wimps (Speed Kills)
- Part II: 8 Options to Enhance Productivity
  - Flight Data Recorder,128-bit Floating Point, Transactional Memory, Vector Architectures, "Virtual CPUs," Innovative Memory, Research Prototypes, Social Science to Settle Debates
- Summary and Conclusion
- ACM feedback (if time permits)

Bad Computer Architecture #1: Aim High (and Ignore Amdahl's Law)

- Peak Performance Sells
  - + Increases employment of computer scientists at companies trying to get larger fraction of peak
- Examples
  - – Very deep pipeline / very high clock rate
  - – Relaxed write consistency
  - – Out-Of-Order message delivery

Bad Computer Architecture #2: Promote Mystery (and Hide Thy Real Performance)

- Predictability suggests no sophistication
  - + If its unsophisticated, how can it be expensive?
- Examples
  - – Out-of-order execution processors
  - – Memory/disk controllers with secret prefetch algorithms
  - – N levels of on-chip caches, where N ~ (Year – 1975) / 10

Bad Computer Architecture #3: Be “Interesting” (and Have a Quirky Personality)

- Programmers enjoy a challenge
  - + Job security since must rewrite application with each new generation
- Examples
  - – Message-passing clusters composed of shared address multiprocessors
  - – TLBs exceptions if access all cache memory on chip
  - – Complicated, undocumented branch predictors
  - – Computing using Graphical Processor Units

Bad Computer Architecture #4: Accuracy & Reliability are for Wimps (Speed Kills Competition)

- Don’t waste resources on accuracy, reliability
  + Probably blame crashes on OS anyways
- Examples
  – Cray et al 754 Floating Point Format, yet not compliant, so get different results from desktop
  – No ECC on L2 Cache of Sun UltraSPARC 2
  – No ECC on Virginia Tech Apple G5 cluster
  – “Error Free” intercommunication networks make error checking in messages “unnecessary”

Alternatives to Hurting Productivity

- Aim High (& Ignore Amdahl’s Law)?
  - No! Delivered productivity >> Peak performance
- Promote Mystery (& Hide Thy Real Performance)?
  - No! Promote a simple, understandable model of execution and performance
    - Should be able to explain to Ph.D.s in physics!
- Be “Interesting” (& Have a Quirky Personality)
  - No programming surprises!
- Accuracy & Reliability are for Wimps? (Speed Kills)
  - No! You’re not going fast if you’re headed in the wrong direction

Enhancing Scientific Productivity: 1) “Flight Data Recorder” for Replay and Debugging

- Mark Hill *et al* propose “Flight Data Recorder” to replay recent execution deterministically, even programs with data races, including OS interactions
- FDR records cache coherence traffic in multiprocessors and thread ordering information in main memory
  - Logs memory state whenever memory updated to starting state
  - Records (minimal subset) of outcome of data races
- 4 processor system to replay last 1 second takes 7% of system memory at just 2% performance cost

To learn more, see “A ‘flight data recorder’ for enabling full-system multiprocessor deterministic replay,” Min Xu, Rastislav Bodik, Mark D. Hill, Proc.. Int’l Symp. Computer Architecture, 2003.
Also see, "Application Development Productivity Challenges for High-End Computing", Vivek Sarkar, Clay Williams, Kemal Ebcioglu, P-PHEC (Workshop on Productivity and Performance in High-End Computing), 2004.

Enhancing Scientific Productivity: 2) 128-bit Floating Point

- Yelick *et al* experience with rewriting scientific codes: numerical bugs every week, one parallel bug per year
- Although math-rigorous error analysis makes most computation fit in 64 bits, such analysis takes time $\Rightarrow$ lowered productivity
  - Parallel programming already really hard
- Simulating 128-bit FP on 64-bit is very slow: 20X
- Adding less than 10% to 64-bit FPU allows 4X slowdown for 128-bit Fl. Pt. (wider registers…)
  - 128-bit Fl. Pt. standard already set (IEEE 754R)
- Also, user-level Fl. Pt. traps reduce programming concerns of denorms, divide by zero, …

To learn more, see How Futile are Mindless Assessments of Roundoff in Floating-Point Computation? http://www.cs.berkeley.edu/~wkahan/Mindless.pdf

Enhancing Scientific Productivity: 3) Transactional Memory

- New research: Kozyrakis and Olukotun at Stanford
- Rather than critical sections, assume that there is no data sharing race conditions, but periodically check and redo threads that conflict
- Buffer writes until "check points," then broadcast addresses before committing writes to memory
  - For cache-coherent, shared address computers
- Trades bursts of network bandwidth for simpler parallel programming
  - Early work with small multiprocessors encouraging

(To learn more, see http://tcc.stanford.edu/)

Enhancing Scientific Productivity: 4) Vector Architectures [Part I]

- 1 instruction operates on vectors of data
- Vector loads get data from memory into big register files, operate, and then vector store
  - E.g., Indexed load, store for sparse matrix
  - Invent new vector load, store for adaptive matrix?
- Easy to add vector to commodity instruction set
  - E.g., Morph SIMD into vector in ~ 30 instructions
  - About the same CPU requirements as SMT

Enhancing Scientific Productivity: 4) Vector Architectures [Part II]

- Compiler technology parallelizes code, hides latency + gives hints how to make even better
- Easy to understand performance model, can avoid quirks, delivers large fraction of peak performance
  - Scientists successfully vectorize their codes
- Need memory to support vector
  - Short term, can deliver vector from L2 cache like Cray X1
  - Long term, memory on same chips as CPU
  - If memory latency is THE problem, innovate there vs. CPU?
- Vectorized codes used to justify novel, non-vector architecture; often all it runs; so just use vector!

To learn more,see http://www.iram.berkeley.edu

Enhancing Scientific Productivity: 5) "Virtual Central Processing Unit"

- Multithreaded occasionally useful now, but exacerbates programming challenge in future
  - Making 64 processors look like 1024 a good idea?
- Clearly, commercial microprocessors will have many processors on chip for commercial workloads
  - Can we make them more attractive building blocks for scientific computing at low cost?
- A different approach (for Berkeley): "Virtual CPU" makes many physical processors look like **fewer** CPUs, ideally one
  - Just as virtual memory invented to make small physical memory look to programmer like a lot of memory

Enhancing Scientific Productivity: Virtual Vector CPU

- For example, collection of processors on a chip $\Rightarrow$ selectable mode where they act as 1 vector computer
- Each processor has one portion of vector unit ("lane") and subset of vector register file
- Memory unit, TLB modified to handle many requests from vector loads and vectors stores
- Common clock makes synchronization easier
- Single chips makes extra paths to exchange information between "vector units" plausible
  - Broadcast vector instructions, Coordination between units, Passing results for reductions, …

Horst Simon, William Kramer, et al, Science-Driven System Architecture: A New Process for Leadership Class Computing, Journal of the Earth Simulator, Vol 2, p 1-9, January 2005.

Enhancing Scientific Productivity: 6) Innovate in Memory vs. CPU

- If Memory Wall is a major performance bottleneck, why not look beyond main memory of DRAM SIMM modules?
  - Why use same memory as in desktops?
  - Simplified programming via low latency has value too?
- SRAM for portion of Main Memory?
- Logic in Memory?
- Stencil in memory?

Enhancing Scientific Productivity: 7) Build Research Prototypes

- DOE universe of all possible architectures
  ≈ what a few manufacturers are willing to propose to a small market
- Instead, fund research prototypes that rapidly, cheaply explore more alternatives?
  - FPGAs to quickly simulate innovative designs while running 7 dwarves with operating systems
  - Lower risk innovation for companies and DOE
    - Millions now to save 10s of millions later takes vision, but sensible
  - Market for reliable processing with massive storage may be larger than market for I/O-starved, accuracy-challenged, undependable, low productivity but high peak performance conventional supercomputer

Enhancing Scientific Productivity: 8) Social Science to Resolve Debates

- CS&E pretends that can’t do experiments with programmers and learn anything significant
  - Yet ~ 25% of university campus does social science experiments on human subjects (under strict guidelines)
- Do a valid, large scale social science experiment to get real measurements on productivity of programming languages and architectures
  - Science to settle debates of Scientific Computing?
  - Compared to supercomputer, doesn’t cost much

Conclusion

- Chips limited in power, memory, implicit parallelism, so processors per socket will grow: 2X / 3 to 4 years
- For years, computer designers neglected productivity
  - No excuse for 21st century scientific computing to be based on untrustworthy, mysterious, I/O-starved, quirky hardware where peak performance is king and must rewrite applications
  - Larger market if 21st century supercomputer ≠ 20th century?
- Let's design processors for 21st century programs (7 ± 3 dwarves) vs. 20th century programs (SPEC200x)
- 8 Options that may help: Flight Data Recorder, 128-bit Fl. Pt., Vector Instructions, Virtual CPUs, Transactional Memory, Innovative Main Memory, Research Prototypes, Social Science to Settle Debates

(From “Why Join ACM?,” David Patterson, *CACM*, Feb ’05, p. 14)

Why Join ACM?

Joiners

Value-seekers

It’s the right thing to do!

Join a worldwide organization with 80,000+ members on 6 continents

Benefit from 35 Special Interest Groups (SIGs): big SIGs like SIGGRAPH, SIGPLAN + small SIGs from SIGACCESS (Accessible Computing for those with disabilities) to SIGWEB (Hypertext, Hypermedia and Web)

Learn from 100+ research conferences, 21+ journals, 4+ magazines, and Digital Library (DL): 58 years of ACM papers online

Sponsor outreach via new Computer Science Teachers Association (K-12) +ACM-Women, Coalition to Diversify Computing (joint with IEEE-CS & CRA), International Collegiate Programming Contest, 500+ student chapters, 100+ local (regional) chapters

Aid advocacy (US-ACM), such as new policy on E-voting machines

Support education via Curriculum Standards and Accreditation

Help honor computing via ACM Fellows + 12 Awards, including it’s “Nobel Prize,”

It’s a good return on your investment!

$99 / year gets **Professional Development Centre** (PDC), magazines, Email redirect

Lifelong learning is up to you in this fast moving field, so stay current via
- •Magazines like *Queue* (featured in Slashdot almost every month)

•PDC with 450+ free IT Courses and 400+ free Online IT Books with search, bookmarking, …
(PDC is *not* part of a DL site license)

Lifelong email forwarding
- •Doesn’t replace favorite email service
- •*YourName*@acm.org can use forever

•ACM 58 years old and growing; Free email in 2020?

Saves money if you go to ≥ 1 ACM conf. / year as ACM discount > $99 or if you take ≥ 1 PDC course / year as course outside ACM costs > $99

May be free as some employers (e.g., Google, Microsoft) pay membership, yet you save them money if go to ≥ 1 ACM conf. or take ≥ 1 PDC course/year

Backup Slides

3 Scientific Programming Cultures: NSF Centers, DOE Labs, Academia

1. At NSF Centers
   - a. Lone wolf amateur scientists / programming experts
   - b. Lone wolf amateur programmers / science experts
   - c. Consortia: groups of 12 to 50 oriented around a problem and set of code
     - i. Separate user group who can use old version
     - ii. Limited number who control the code and get the first crack at using it to advance the science
     - iii. Enough resources so that can hire a few professional programmers
     - iv. Properly motivated to do good science and efficiently use machines
   - d. Users of canned systems e.g., NAMD

3 Scientific Programming Cultures: NSF Centers, DOE Labs, Academia

2. At DOE labs
   a. Unclassified DOE and NSF centers share similarities
   b. New group: “hired guns” expert programmers paid to fix codes, make things run well
3. Academic scientists outside of labs
   a. Do-it-yourselfers trying to get work done on clusters
   b. Theoreticians who gave up on parallel programming, waiting for desktop to get faster
   – Matlab is lingua franca for publications, pseudo code

Size of Machine

- What parallelism achievable with good or bad architectures, good or bad algorithms?
  - 32-way: anything goes
  - 100-way: good architecture and bad algorithms
    or bad architecture and good algorithms
  - 1000-way: good architecture and good algorithm

100 Processor Plateau

- Why does computation stay at ~ 100 processors?
- Scientists improve accuracy, efficiency of computation to improve solution but generally reduce parallelism
  - E.g., adjustable matrix resolution
- Programmers recode to improve parallelism