

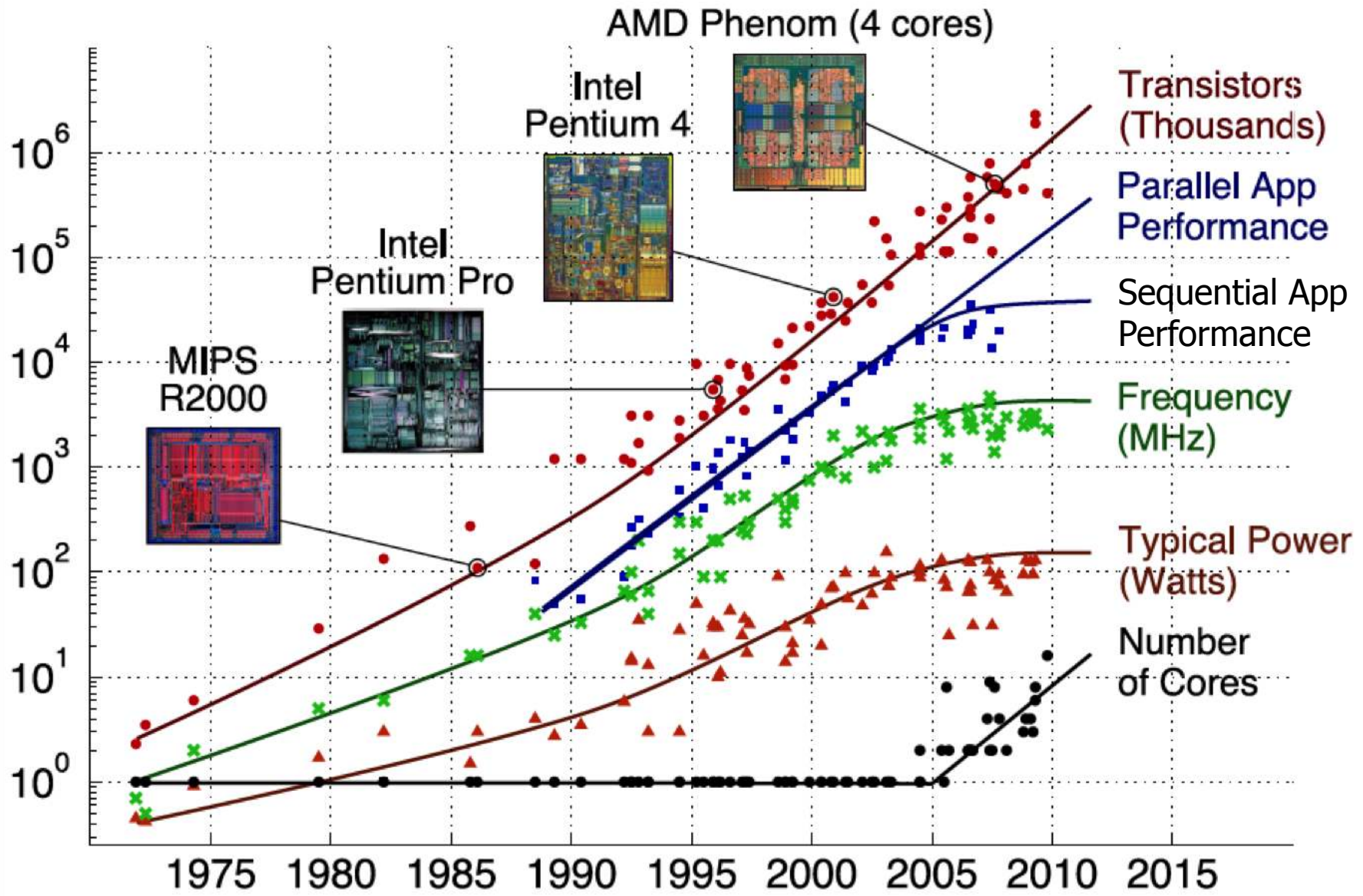
Software Knows Best: Portable Parallelism Requires Standardized Measurements of Transparent Hardware

Sarah Bird, Archana Ganapathi, Kaushik Datta,
Karl Fuerlinger, Shoaib Kamil, Rajesh Nishtala,
David Skinner, Andrew Waterman, Sam Williams,
Krste Asanović, and [Dave Patterson](#)

January 29, 2010

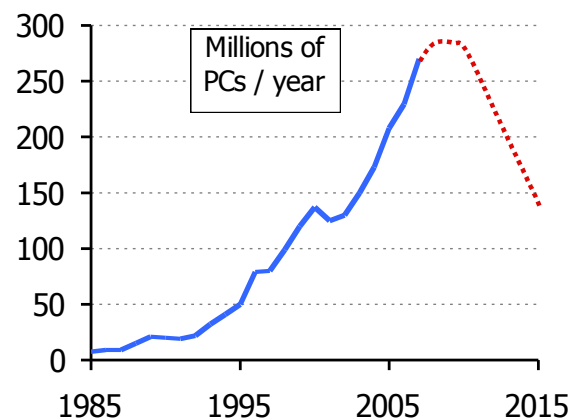
- ❖ Future parallel software adjusts dynamically vs. SPECcpu's statically-linked legacy C code
- ❖ If you expect programmers to continue “Moore’s Law” by doubling amount of portable parallelism in programs every 2 years, need hardware measurement for them to see how well doing
 - During *development* inside an IDE
 - During *runtime* so that app, resource scheduler, and OS can see and adapt
- ❖ Standardized Hardware Measurement may be as important as the IEEE Floating Point Standard

- ❖ Par Lab
 - Motivation, Context, Approach, Apps, SW Stack, Architecture, and Recent Results
- ❖ Case for Hardware Measurement
 - Performance Portability Experiment
 - Parallel Resource Allocation Needs
 - Shortcomings of Current Counters
 - SHOT Architecture and 1st Implementation
 - Potential Concerns
- ❖ Conclusion



Data partially collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond

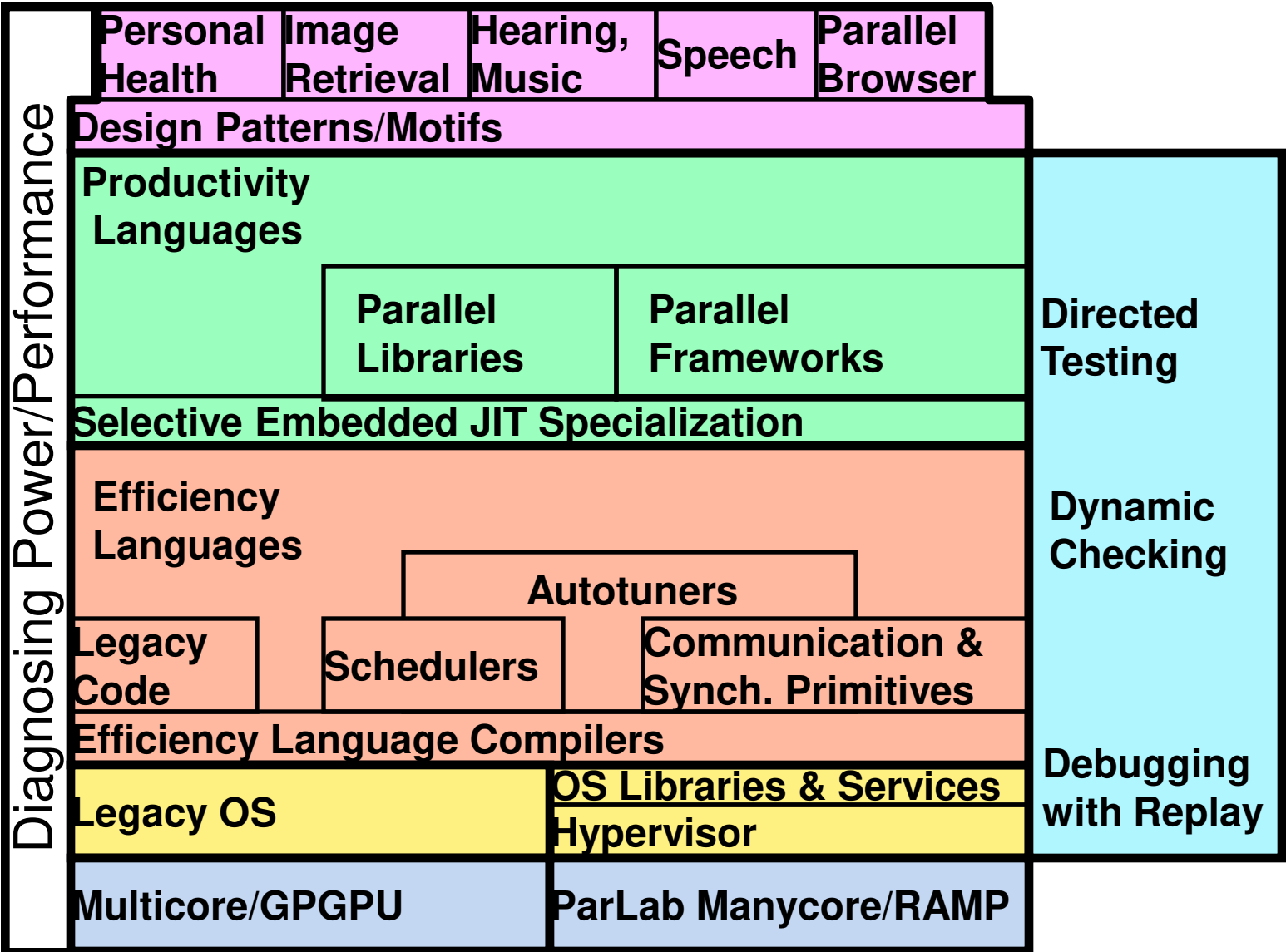
- ❖ John Hennessy, President, Stanford University:
*“...when we start talking about parallelism and ease of use of truly parallel computers, we're talking about a problem that's as hard as any that computer science has faced. ...
 I would be panicked if I were in industry.”*
“A Conversation with Hennessy & Patterson,” ACM Queue Magazine, 1/07.
- ❖ 100% failure rate of Parallel Computer Companies
 - Convex, Encore, Inmos (Transputer), MasPar, NCUBE, Kendall Square Research, Sequent, Silicon Graphics, Thinking Machines
- ❖ What if IT goes from a growth industry to a replacement industry?
 - If SW can't effectively use 32, 64, ... cores per chip
 => SW no faster on new computer
 => Only buy if computer wears out



- Berkeley researchers from many backgrounds meeting since Feb. 2005 to discuss parallelism
 - Krste Asanovic, Ras Bodik, Jim Demmel, Kurt Keutzer, John Kubiawicz, 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)
- Led to “Berkeley View” Tech. Report 12/2006 and new Parallel Computing Laboratory (“Par Lab”)
- From Top 25 CS Depts, Intel/MS award UCB \$10M
- Goal: Productive, Efficient, Correct, Portable SW for 100+ cores & scale as core increase every 2 years (!)

Easy to write portable code that runs efficiently on manycore

Applications
Productivity Layer
Efficiency Layer
OS Arch.



Correctness



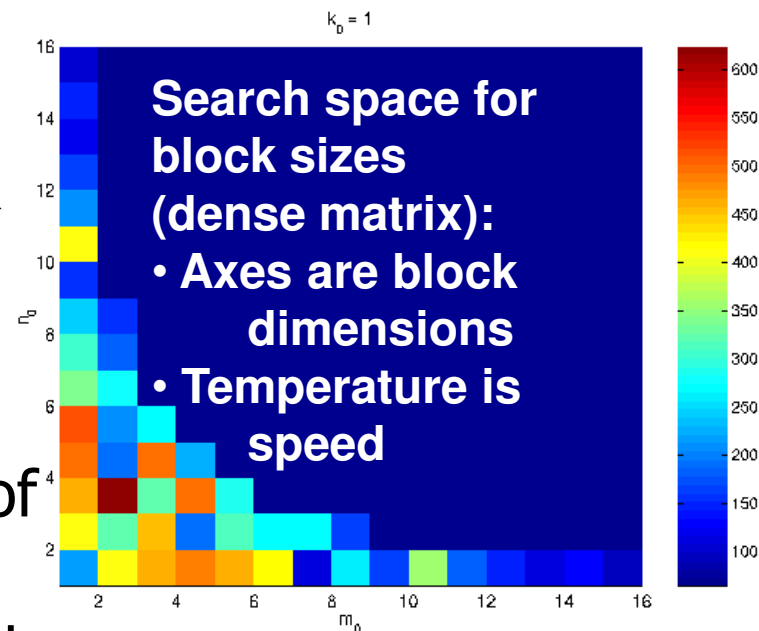
- ❖ Data Center or Cloud ("Server")
- ❖ Handheld/Tablet/Laptop ("Mobile Client")
- ❖ Both together ("Server+Client")
- ❖ Apps of the future are partly in the Cloud and partly in the Mobile Client, and functions may shift depending on platforms, connectivity, conditions

- ❖ **What are the compelling future workloads?**
 - Need apps of future vs. legacy to drive agenda
 - Improve research even if not the real killer apps
- ❖ **Computer Vision:** [Segment-Based Object Recognition](#), Poselet-Based Human Detection
- ❖ **Health:** [MRI Reconstruction](#), Stroke Simulation
- ❖ **Music:** 3D Enhancer, Hearing Aid, Novel UI
- ❖ **Speech:** Automatic Meeting Diary
- ❖ **Video Games:** Analysis of Smoke 2.0 Demo
- ❖ **Computational Finance:** Value-at-Risk Estimation, Crank-Nicolson Option Pricing
- ❖ **Parallel Browser:** Layout, Scripting Language

- ❖ Examining our applications and future platforms it's clear..
 1. Users want full-featured computationally-intensive responsive applications
 2. Power is very important for the cloud
 3. Battery life (energy) is very important for client

- ❖ **Optimizing for performance is still the best way to get good energy efficiency which solves all 3 goals**

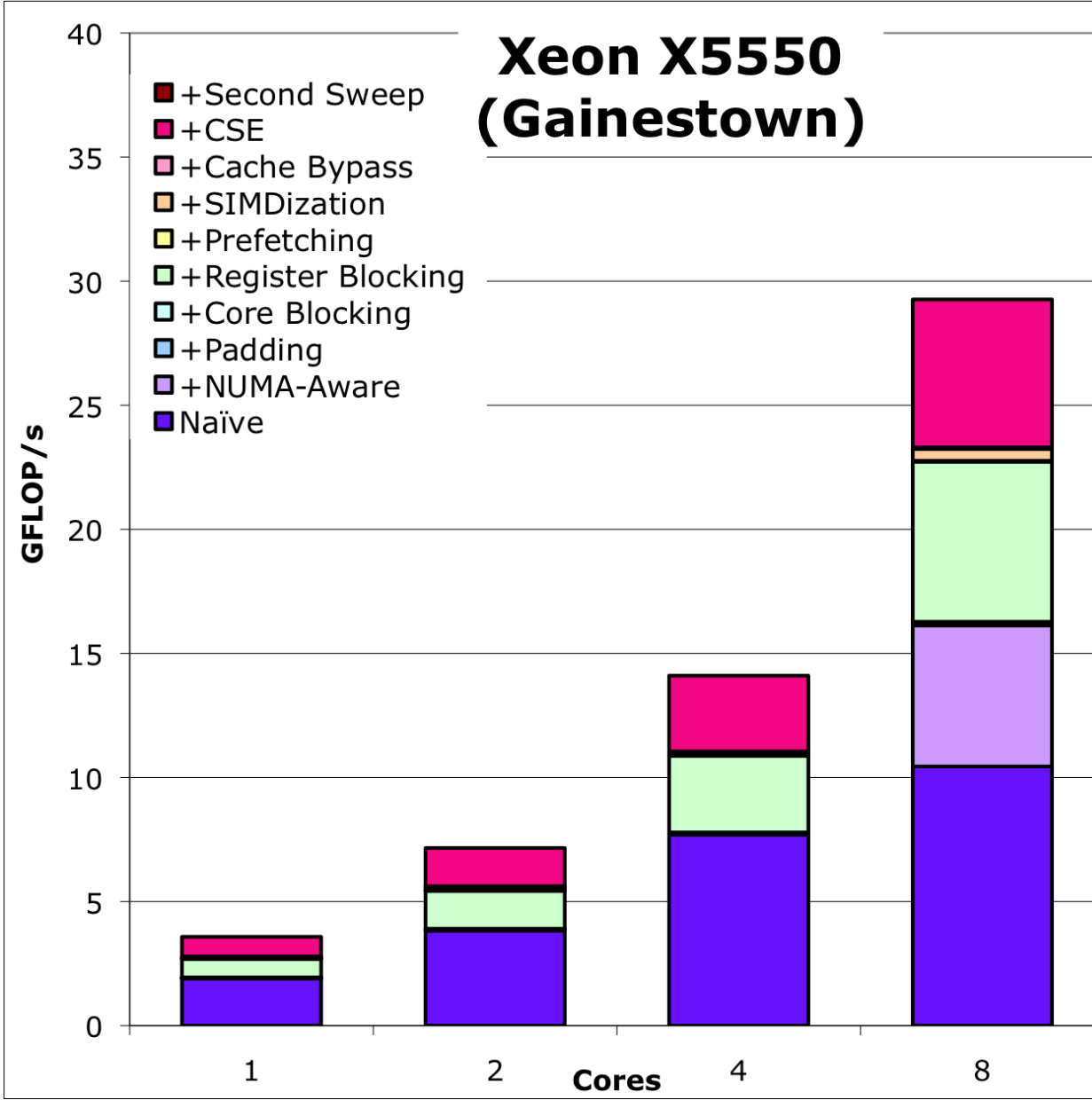
- Problem: generating optimal code like searching for needle in haystack
- Manycore → even more diverse
- New approach: “Auto-tuners”
 - 1st generate program variations of combinations of optimizations (blocking, prefetching, ...) and data structures
 - Then compile and run to heuristically search for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



Example on Intel Xeon X5500 for 27 Point Stencil

❖ For 8 cores,
autotuning gives
~3X improvement
over naïve code

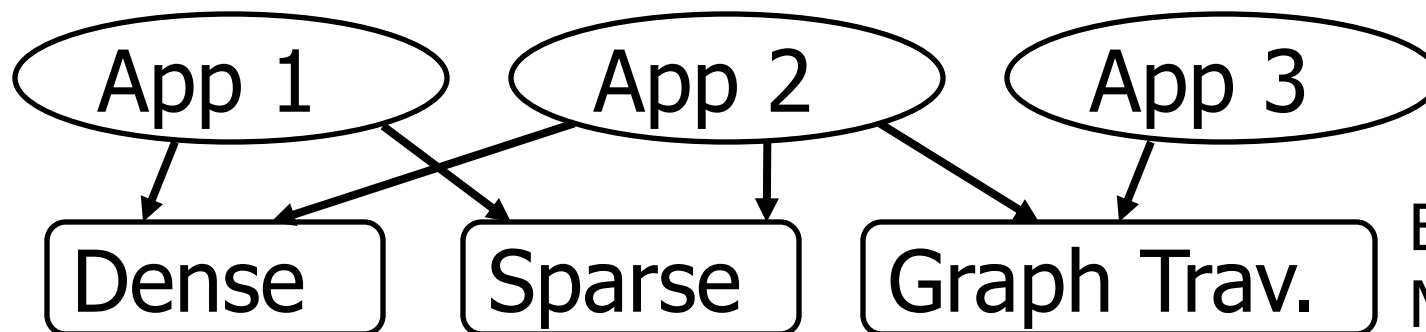
- Common Subexpression Elimination
- SIMDization
- Core Blocking
- NUMA aware



- ❖ *Autotuning has great potential for achieving good performance for applications*
- ❖ Unfortunately,
 - They take an expert a long time to write
 - There isn't a good framework for reusing them or for others to deploy them in ordinary code
 - They tune statically for a fixed platform — concurrently running applications violate this assumption
 - The search space is large—taking a lot of cycles and a long time to explore






- ❖ Libraries? Can be helpful, but brittle
 - Situation off a little from what you need and you can't use library

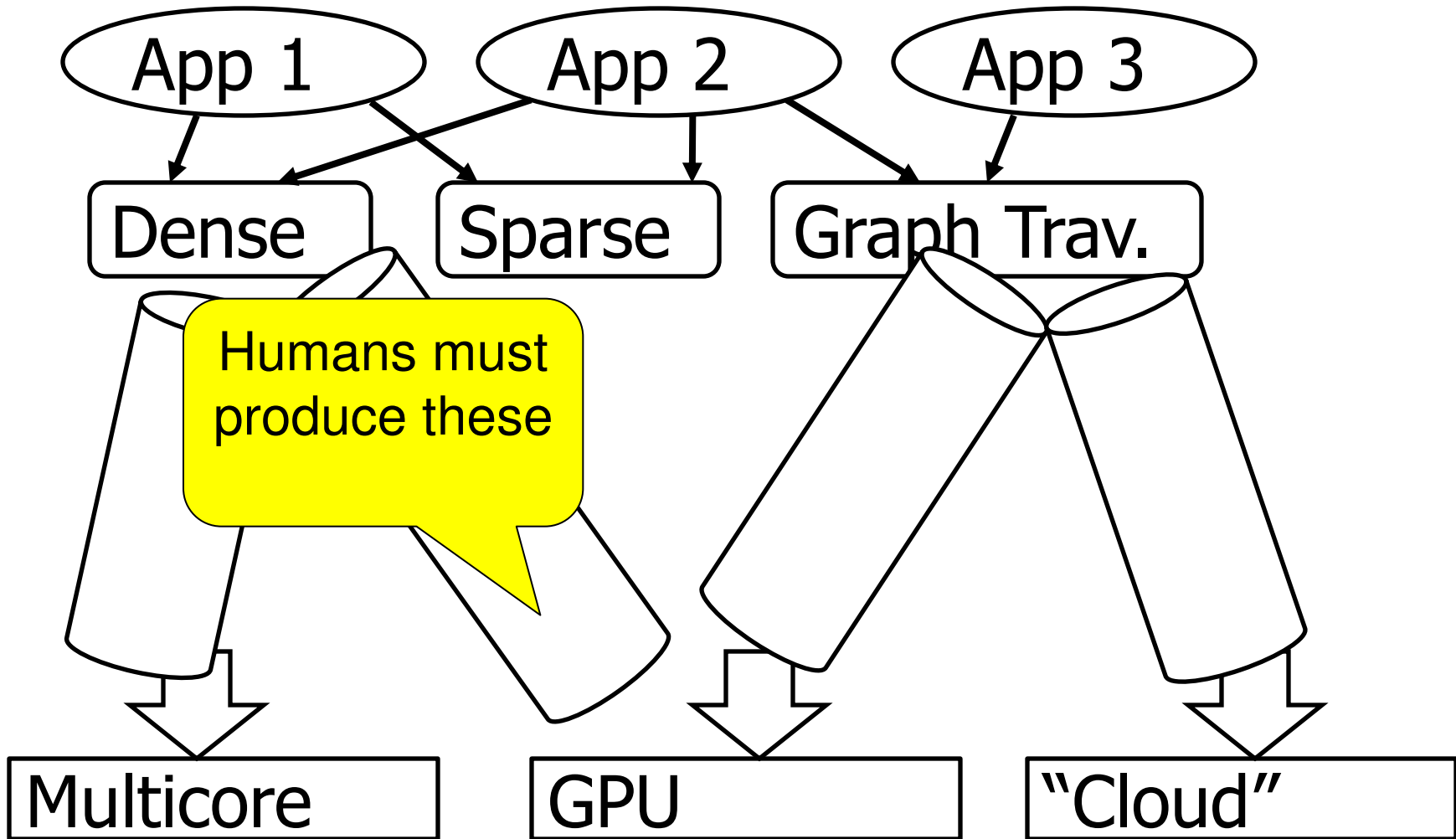
- ❖ *Productivity level language* (PLL): Python, Ruby
 - high-level abstractions well-matched to application domain => 5x faster development and 3-10x fewer lines of code
 - >90% of programmers
- ❖ *Efficiency level language* (ELL): C/C++, CUDA, OpenCL
 - >5x longer development time
 - potentially 10x-100x better performance by exposing HW model
 - <10% of programmers
- ❖ 5x development time \neq 10x-100x performance!
Raise level of abstraction and get performance?



Berkeley View
Motifs
("Dwarfs")

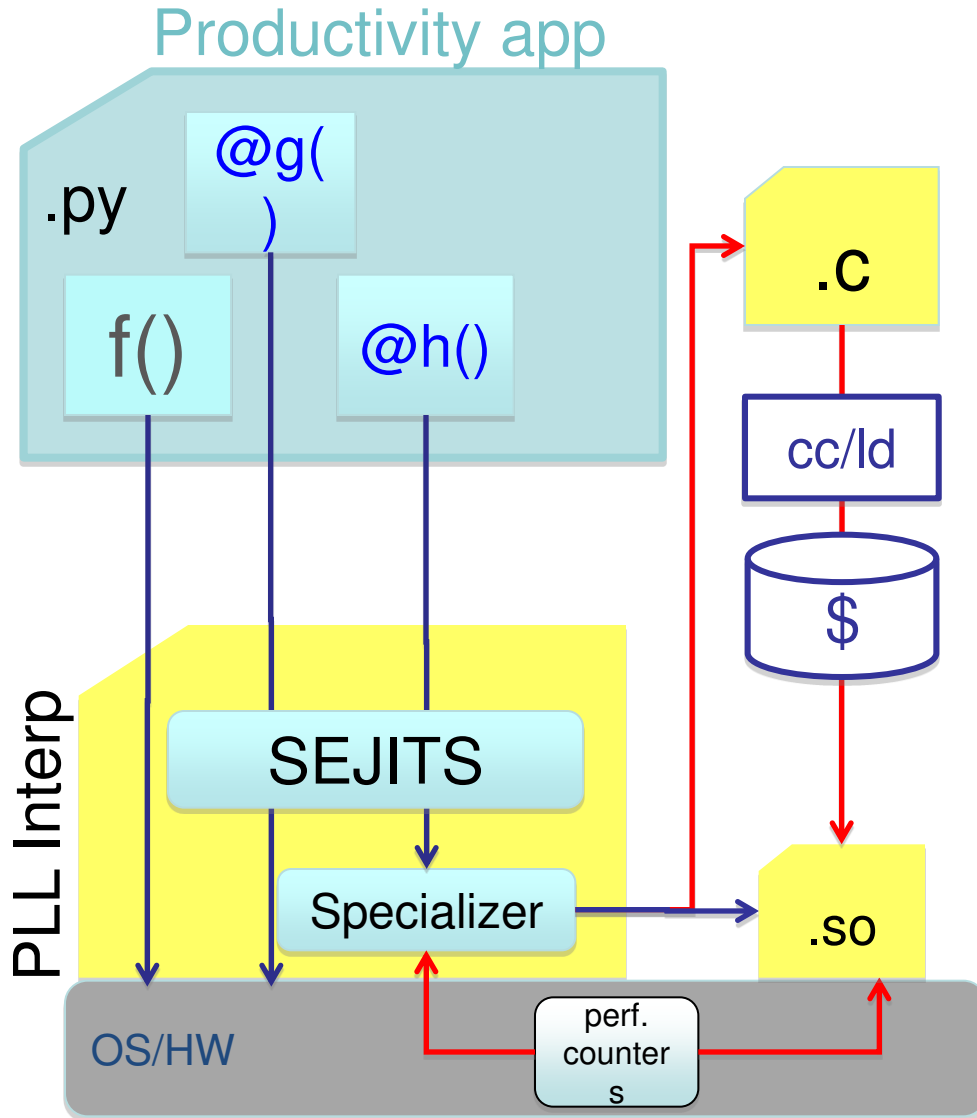
□ How do compelling apps relate to 12 motifs?

	Embed	SPEC	DB	Games	ML	CAD	HPC	 Health	 Image	 Speech	 Music	 Browser
1 Finite State Mach.	Red	Red	Red	Yellow	Yellow	Yellow	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue	Red
2 Circuits	Red	Light Blue	Green	Light Blue	Green	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue	Red
3 Graph Algorithms	Red	Yellow	Yellow	Yellow	Red	Red	Light Blue	Red	Light Blue	Red	Green	Green
4 Structured Grid	Red	Red	Light Blue	Yellow	Light Blue	Light Blue	Red	Light Blue	Red	Light Blue	Light Blue	Light Blue
5 Dense Matrix	Red	Red	Yellow	Red	Red	Red	Light Blue	Red	Red	Red	Red	Light Blue
6 Sparse Matrix	Yellow	Yellow	Light Blue	Red	Red	Red	Light Blue	Red	Light Blue	Light Blue	Red	Light Blue
7 Spectral (FFT)	Yellow	Light Blue	Light Blue	Yellow	Yellow	Yellow	Red	Light Blue	Green	Red	Red	Red
8 Dynamic Prog	Yellow	Light Blue	Red	Light Blue	Red	Red	Light Blue	Light Blue	Light Blue	Yellow	Light Blue	Red
9 Particle Methods	Light Blue	Yellow	Light Blue	Yellow	Light Blue	Light Blue	Red	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue
10 Backtrack/ B&B	Light Blue	Light Blue	Yellow	Light Blue	Red	Red	Light Blue	Light Blue	Light Blue	Light Blue	Yellow	Light Blue
11 Graphical Models	Light Blue	Light Blue	Yellow	Light Blue	Red	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue	Red	Light Blue
12 Unstructured Grid	Light Blue	Light Blue	Light Blue	Yellow	Yellow	Yellow	Red	Red	Light Blue	Light Blue	Red	Light Blue

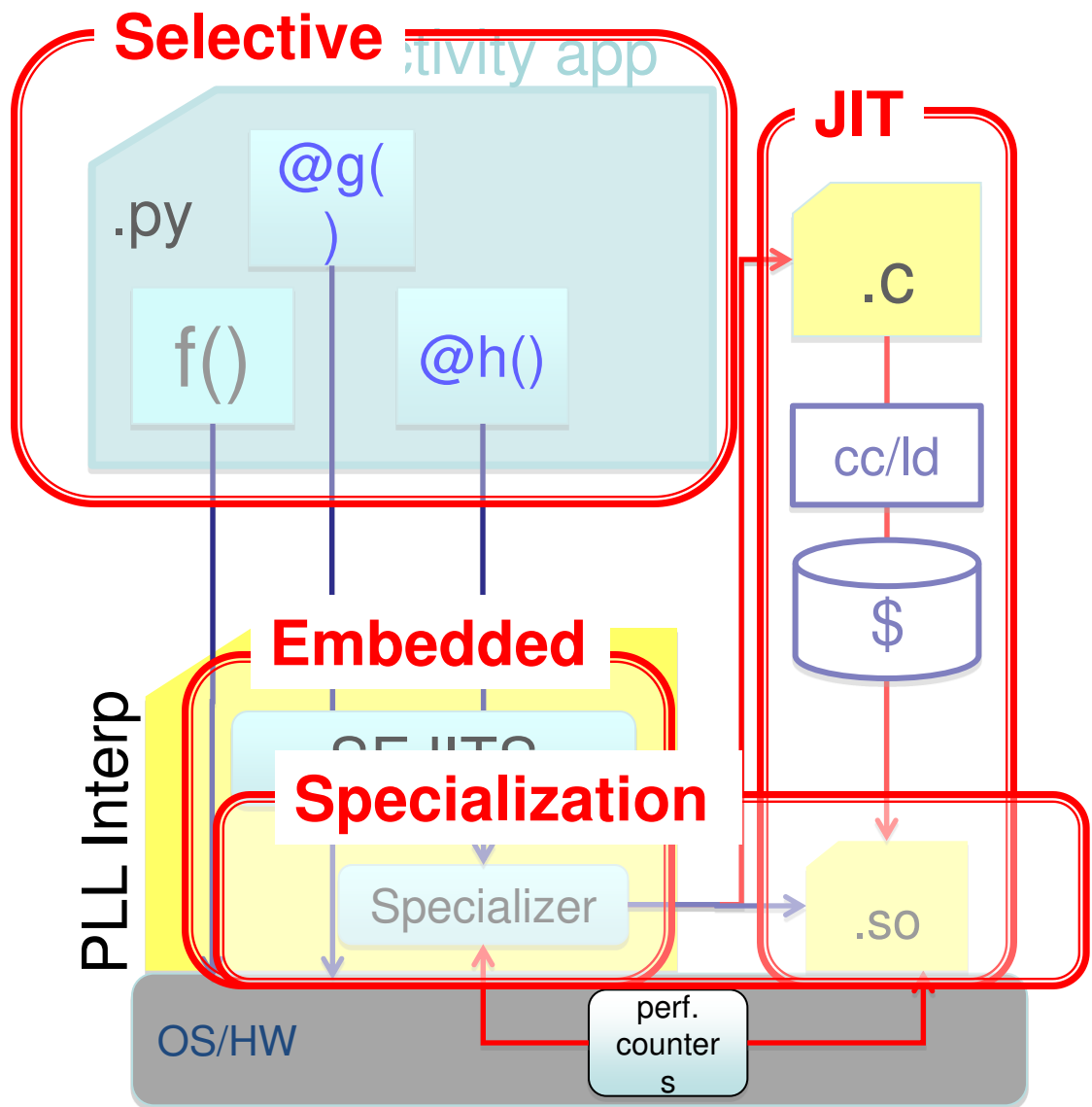


- ❖ Productivity programmers write in *general purpose, modern, high level* PLL
- ❖ SEJITS infrastructure *Specializes* (optimizes, tunes) computation motifs *Selectively* at runtime
- ❖ Specialization uses runtime info to *generate* and *JIT-compile* ELL code targeted to hardware
- ❖ *Embedded* because PLL's own machinery enables (vs. extending PLL interpreter)

SEJITS makes tuning decisions *per-function* (not per-app)



SEJITS makes tuning decisions *per-function* (not per-app)

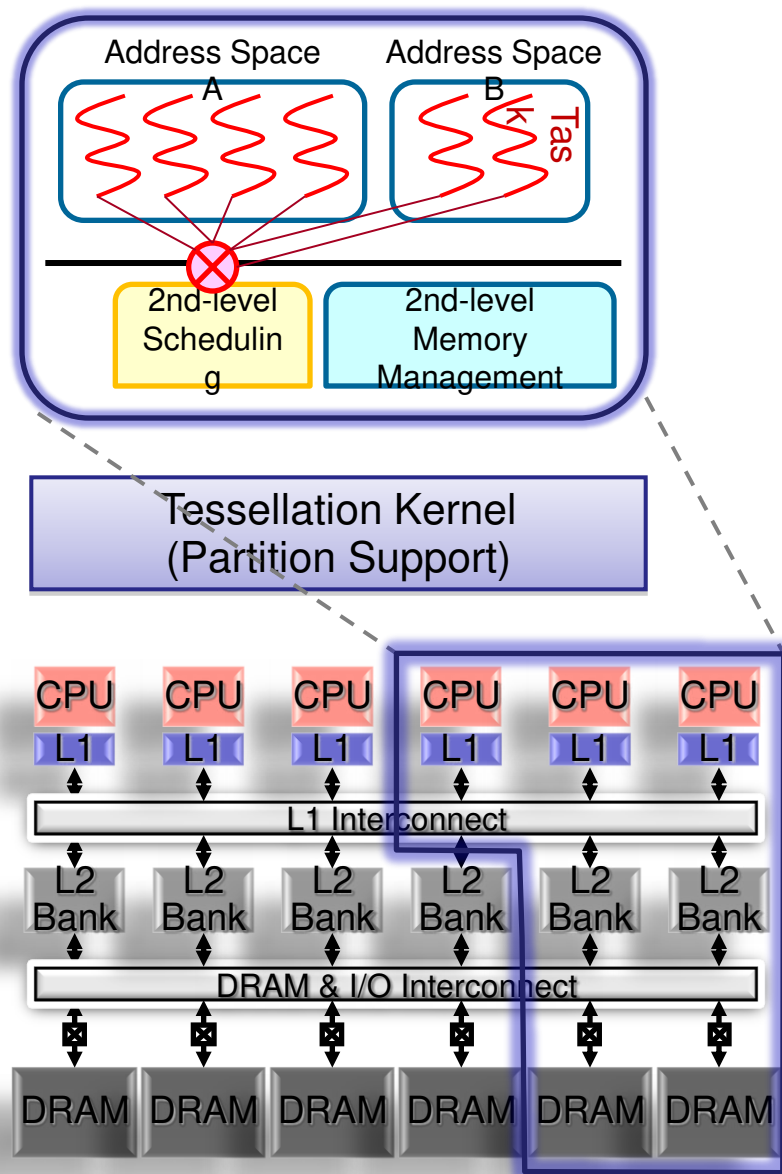


- ❖ Want to benchmark autotuner, JIT, compiler adapting to the hardware being used at install time as well as during run time
- ❖ Statically linked legacy C programs irrelevant to multicore future
 - Good idea in 1980s not so much in 2010s

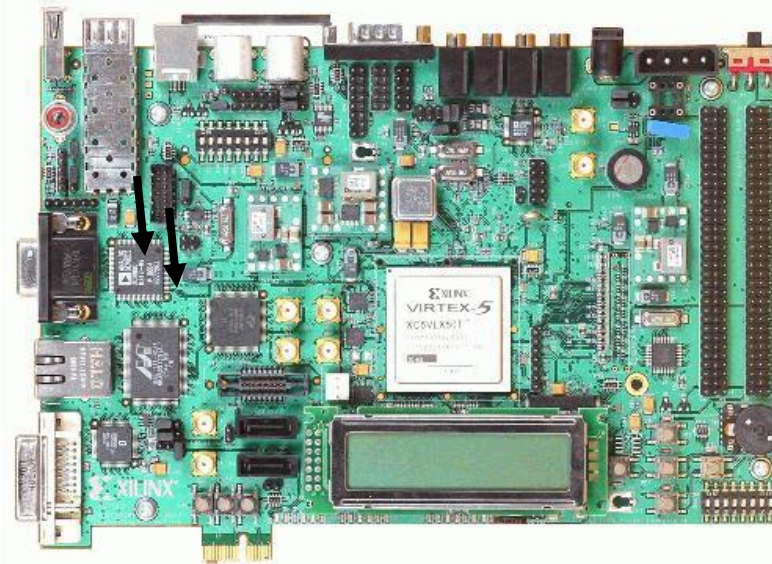
- ❖ *Autotuning has great potential for achieving good performance for applications*
- ❖ Unfortunately,
 - They take an expert a long time to write — **Still True**
 - ~~There isn't a good framework for reusing them or for others to deploy them in ordinary code — **SEJITS**~~
 - They tune statically for a fixed platform — concurrently running applications violate this — **Adaptive Applications and OS + Hardware Measurement?**
 - The search space is large—taking a lot of cycles to explore and a long time — **Machine Learning + Hardware Measurement (Later in Talk) to democratize autotuning**

- ❖ Real-time apps adapt to resources available
- ❖ Not enough resources:
 - Lower quality of audio synthesis so no clicks in music
 - Reduce quality of graphics or realism of physics simulations to get steady frame rate
 - Reduce complexity of web pages served to meet response times SLO under heavy load
- ❖ Too many resources:
 - Release resources back to OS to preserve battery life in client or save power in cloud

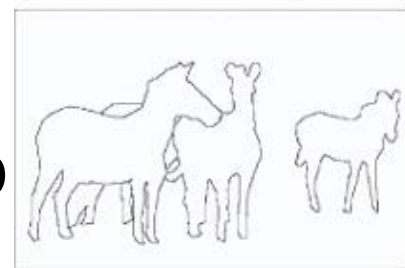
- ❖ *Space-Time Partitioning*
 - *Provides performance isolation to applications*
 - *Strict QoS guarantees*
 - *Makes performance tuning/autotuning more effective*
 - *Can adapt partition sizes for current mix of applications to meet performance and energy goals for the system*



- ❖ Rapid accurate simulation of manycore architectural ideas using FPGAs
- ❖ Initial version models 64 cores of SPARC v8 with shared memory system on \$750 board
- ❖ Hardware FPU, MMU, boots OS
- ❖ 250X faster than SW simulator



	Cost	Performance (MIPS)	Simulations per day
Software Simulator	\$2,000	0.1 - 1	1
RAMP Gold	\$2,000 + \$750	50 - 100	100



- ❖ Bryan Catanzaro: Parallelizing Computer Vision (image segmentation)
- ❖ Problem: Malik's highest quality algorithm was 5.5 minutes / image on new PC
- ❖ Good SW architecture + talk within Par Lab on to use new algorithms, data structures
 - Bor-Yiing Su, Yunsup Lee, Narayanan Sundaram, Mark Murphy, Kurt Keutzer, Jim Demmel, Sam Williams
- ❖ Current result: 1.8 seconds / image on manycore
- ❖ ~ 150X speedup
 - Factor of 10 quantitative change is a qualitative change
- ❖ Malik: "This will revolutionize computer vision."

- Pediatric MRI is difficult
 - Children cannot keep still or hold breath
 - Low tolerance for long exams
 - Must put children under anesthesia: risky & costly
- Need techniques to accelerate MRI acquisition (sample & multiple sensors)
- Reconstruction must also be fast, or time saved in acquisition is lost in compute
 - Current reconstruction time: 2 hours
 - Non-starter for clinical use
 - Mark Murphy (Par Lab) reconstruction: 1 minute on manycore
 - Fast enough for radiologist to make critical decisions
 - Dr. Shreyas Vasanawala (Lucille Packard Children's Hospital) put into use Feb 2010 for further clinical study

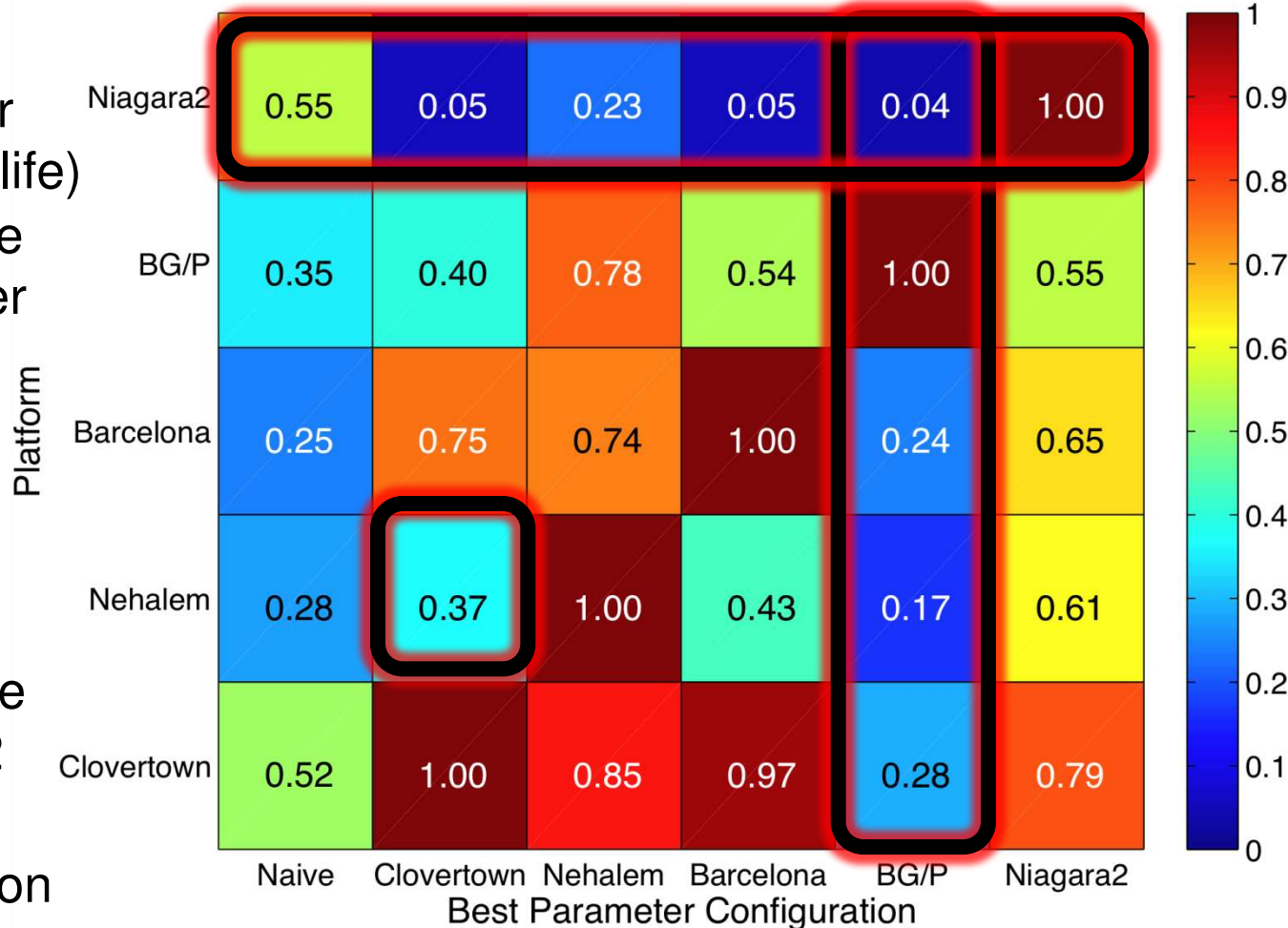


- ❖ *Let compelling applications drive research agenda*
- ❖ Software platform: mobile client + cloud
 - Apps that dynamically shift functions between client & client depending on conditions
- ❖ Identify common programming patterns to reveal parallelism
- ❖ Productivity versus efficiency programmers
- ❖ Autotuning and software synthesis
- ❖ OS/Architecture support multiple applications running simultaneously that adapt to save energy
- ❖ FPGA simulation of new parallel architectures: RAMP
- ❖ Build power/performance measurement into stack to help autotuning, SEJITS, scheduling, energy efficiency

- ❖ Par Lab
 - ❖ Motivation, Context, Approach, Apps, SW Stack, Architecture, and Recent Results
- ❖ Case for Hardware Measurement
 - Performance Portability Experiment
 - Parallel Resource Allocation Needs
 - Shortcomings of Current Counters
 - SHOT Architecture and 1st Implementation
 - Potential Concerns
- ❖ Conclusion

- ❖ Writing parallel code is hard
- ❖ Only reasons are performance or energy efficiency
 - Otherwise write sequential code
- ❖ To become mainstream, parallel code must be portable
- ❖ Hence parallel HW/SW must support *performance-portable parallel software*
- ❖ Yet HW getting more diverse (multicore, mobile platforms, cloud) and SW getting more dynamic (autotuning, SEJITS, acquiring/releasing resources to save energy, client-cloud shifting)

27-Point Stencil [Fraction of platform best]



- ❖ Code tuned for another machine
~ 1.5X to 3X slower
(terrible for battery life)
- ❖ Code tuned for Blue Gene always slower than naïve code
- ❖ Naïve code for Niagara 2 always faster than code tuned for another
- ❖ Code tuned for Blue Gene on Niagara 2 25X slower
- ❖ Even next generation Intel MPU is ~3X slower if tuned to old architecture

7 Shortcomings of Current Counters

1. Essential metrics are not measurable
 - Not able to compute memory traffic on an Opteron or POWER5 because prefetches not measurable by an accessible counter
2. Many metrics are strongly tied to microarchitectural details
 - SiCortex has performance counters for stalls in each pipeline stage but hard to know what is happening in each stage

3. High access overheads

- Some systems require serialization of the pipeline in order to access counters
- Can't put measurement inside functions and too expensive to support adaptation on the fly

4. Limited number of counters that can be used simultaneously

- IBM Blue Gene can measure + and -, or \times and \div , but not both at the same

5. No support for multiple applications

- AMD Barcelona: One core's programming of shared L3 cache counters can be over-ridden by another core, and no way to prohibit it

6. Not standardized

- Not consistently available on enough MPUs for apps and OSes to rely on them

7. Not correct or not functional

- R12000 instructions decoded counter off 25%
- Counters not thought a critical component to verify since intended only for chip engineers

- ❖ **Standardized Hardware Operation Tracker: SHOT**
- ❖ Since some counters are per core, SW must read all counters as if on same clock edge
 - e.g., via distributed latches loaded simultaneously
 - Don't need to be perfect counts, just consistent: accuracy $\pm 1\%$ OK
- ❖ Low latency reads so deployed in production code
- ❖ Can be read by OS and by user apps
- ❖ To be used by virtual machines, must be able to save and restore as part of context switch

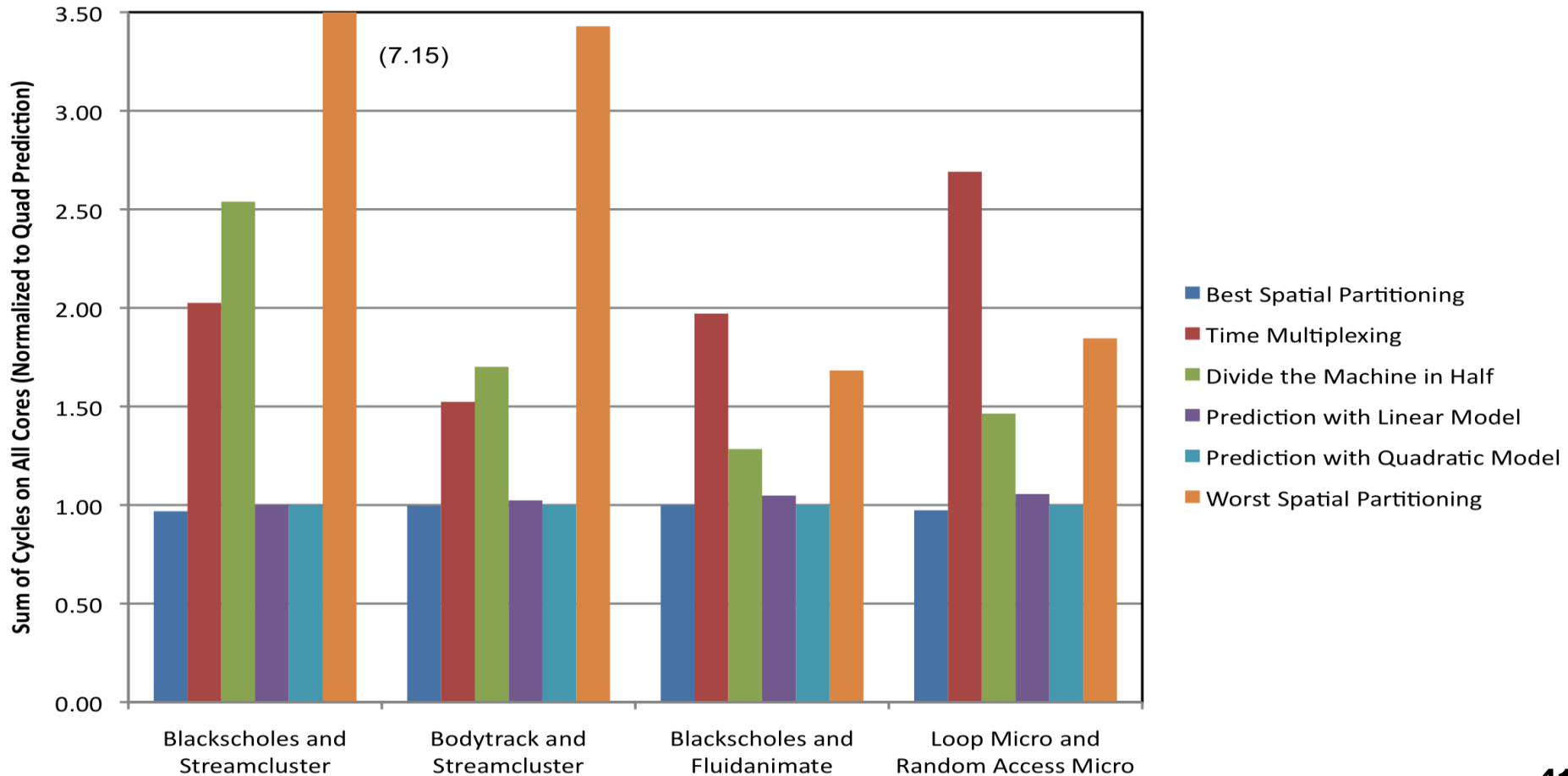
1. **Global real time clock** (vs. count clock cycles)
 - Since clock rate varies due to Dynamic Voltage and Frequency Scaling (DVFS)
 - ~ 100 MHz (fast enough for apps)
2. **Number instructions retired per core**
 - Measure computation throughput
3. **Off-chip memory traffic** (including prefetching)
 - Key to performance and energy
 - ❖ **Standard so apps and OS can rely on them**
 - ❖ **Implemented on RAMP Gold FPGA Simulator**

- ❖ Desirable, but not part of minimum standard
- 4. Energy consumption per task of SW visible components (cores, caches)
- 5. Instructions executed by type
 - Floating point, integer, load, store, control
- 6. Cache traffic by category
 - Speculative, compulsory, capacity miss, conflict miss, write allocate, write back, coherency
- 7. Time spent in each power state for each component

- ❖ Operating System
 - Adjust resources between apps – **Runtime**
 - Co-schedule applications with disjoint resource requirements – **Runtime**
- ❖ Library, Framework, and Autotuner Writers
 - Runtime performance to adjust thread scheduling, make algorithmic changes, and release resources – **Install Time & Runtime**
- ❖ Efficiency Programmers as part of IDE tools
 - **Development Time**
- ❖ Productivity Programmers
 - Not directly - benefit from OS and Library use

- ❖ Autotuning problem: The search space is large—taking a lot of cycles to explore and a long time
- ❖ Search Full Parameter Space
 - More than 180 Days
- ❖ Using machine learning + few performance counters to democratize autotuning
 - 12 minutes to find solution
- ❖ ~As good or even beat the expert!
 - -1% and 16% for a 7-pt Stencil
 - -2% and 15% for a 27-pt Stencil
 - 18% and 50% for dense matrix
- ❖ Enables even greater range of optimizations than we imagined

- ❖ Runtime OS schedule 2 programs via prediction using counters within 3% optimal, 1.7X – 2X faster than dividing machine or time multiplexing



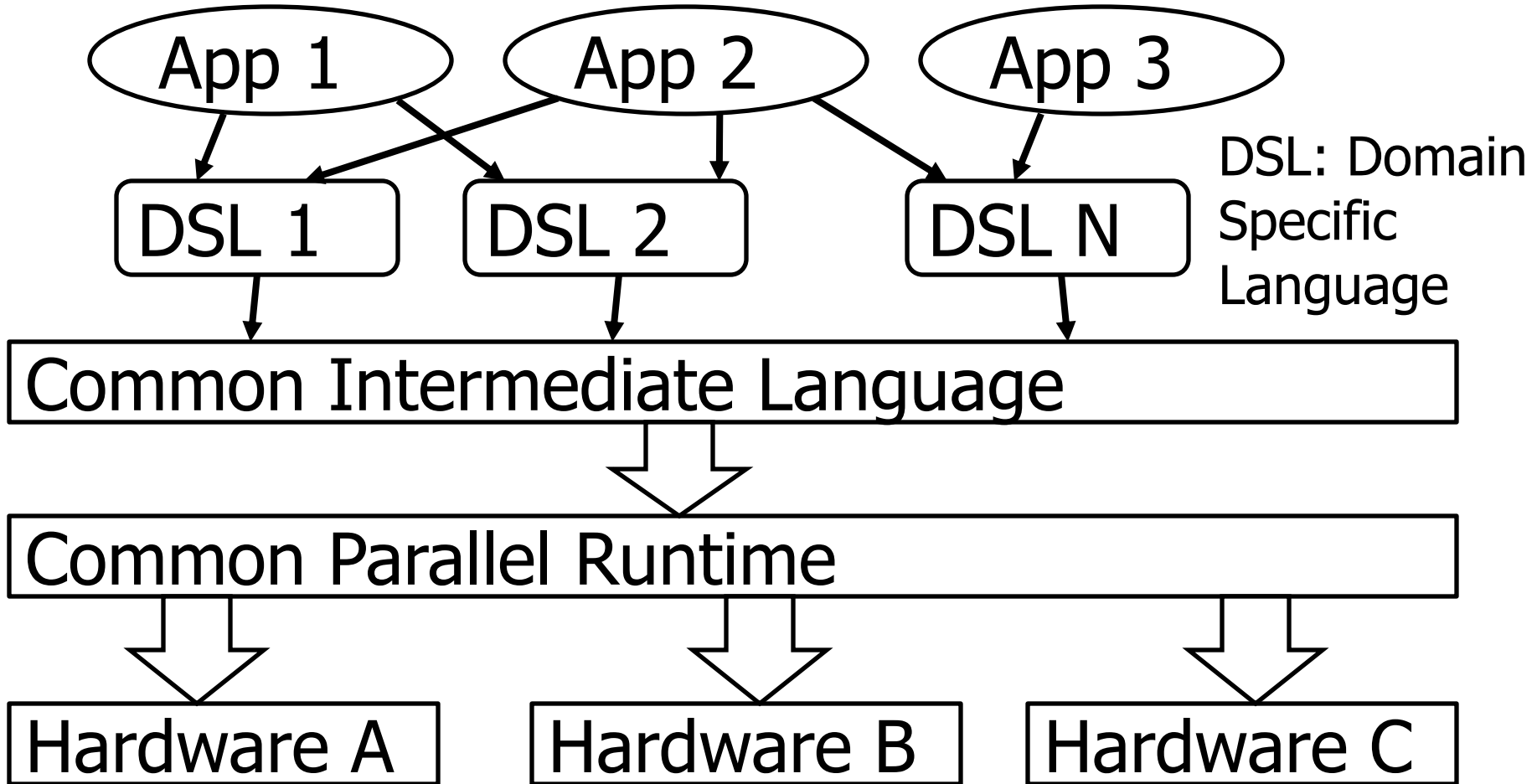
1. Given that current MPUs have 100s of events they can count, it is impossible to select a useful architecture-independent set of metrics
 - Detailed microarchitectural runtime info from 100s of events is wrong level of performance abstraction for parallel software
 - Just need a few, top-down measurements
2. Such measurement hardware is too expensive
 - Counters can be made small and low power, accuracy $\pm 1\%$ OK
 - SiCortex's performance counters account for 0.05% of the transistors on chip

3. Exposing power and performance information is a competitive disadvantage
 - E.g., could show customers that 1 core runs slower, hotter due to process variation
 - E.g., could give away microarchitectural details that are a competitive advantage
 - But *not exposing a disadvantage* since apps, libraries, frameworks, runtimes and OSes that use them will run more efficiently on a competitor's chip that implements SHOT

4. Standardization can be done entirely in SW
 - SW standard intractable
 - PAPI started 1999, not portable, and developers say situation getting worse
5. SHOT creates an Information Side Channel that can be a security threat
 - Much of this info can already be approximated
 - Difficult in practice because adversarial code must also know if victim app is running, what other programs are sharing the resource
 - So many simpler attacks that this is not high on security experts list of concerns

- ❖ SW adapts more at runtime than in the past
 - Client-Cloud, Energy saving, Autotuning, SEJITS, scheduler, OS
- ❖ Parallel HW even more diverse than sequential
 - Code for other platform runs ~1.5X-3X slower
- ❖ Multicore challenge hardest for CS in 50 years
 - Performance portability is one of main obstacles
- ❖ For programmers to sustain “Moore’s Law,” architects must make HW measurable to different SW layers during development *and* during runtime
- ❖ SHOT as big impact on portable parallel code as

- ❖ Asanović, K., R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiawicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, K. Yelick., "A View of the Parallel Computing Landscape," *Communications of the ACM*, vol. 52, no. 10, October 2009.
- ❖ Bird, S., A. Ganapathi, K. Datta, K. Fuerlinger, S. Kamil, R. Nishtala, D. Skinner, A. Waterman, S. Williams, K. Asanović, D. Patterson, "Software Knows Best: Portable Parallelism Requires Standardized Measurements of Transparent Hardware," submitted for publication.
- ❖ Catanzaro, B., A. Fox, K. Keutzer, D. Patterson, B-Y. Su, M. Snir, K. Olukotun, P. Hanrahan, and H. Chafi, "Ubiquitous Parallel Computing from Berkeley, Illinois and Stanford," *IEEE Micro*, to appear, March/April 2010.
- ❖ Catanzaro, B., S. Kamil, Y. Lee, K. Asanović, J. Demmel, K. Keutzer, J. Shalf, K. Yelick, and A. Fox, "SEJITS: Getting Productivity and Performance with Selective Embedded JIT Specialization," *1st Workshop on Programmable Models for Emerging Architecture (PMEA) at the 18th International Conference on Parallel Architectures and Compilation Techniques*, Raleigh, North Carolina, November 2009.
- ❖ Korn, W., P. Teller, and G. Castillo. Just how accurate are performance counters? *IEEE International Conference on Performance, Computing, and Communications*, p. 303–310, April 2001.
- ❖ Tan, Z., A. Waterman, S. Bird, H. Cook, K. Asanović, and D. A. Patterson, "A Case for FAME: FPGA Architecture Model Execution," submitted for publication.



Domains: Too many, too dynamic

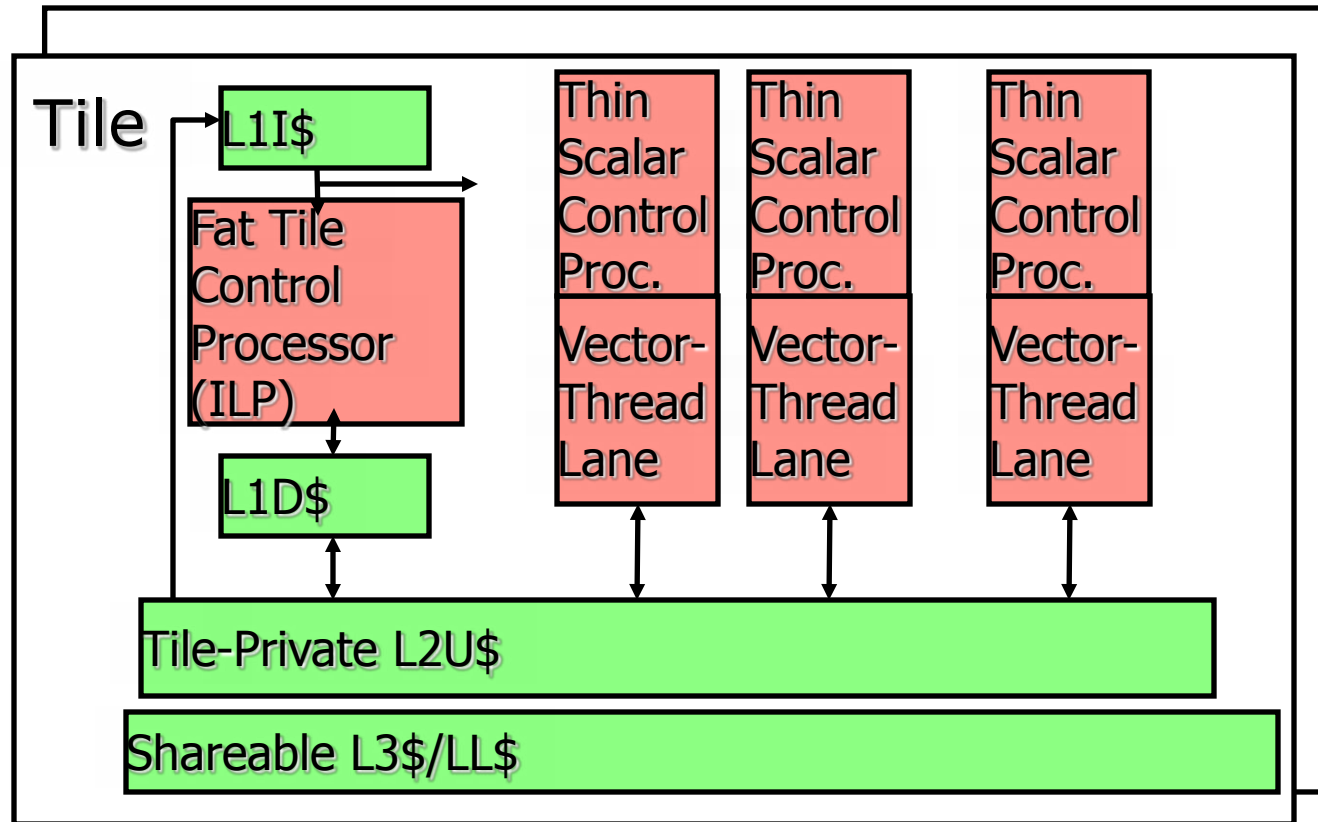
- ❖ New domain per app?
- ❖ Multiple domains in one app? Learn new syntax?

Layers: Abstraction loses important information

- ❖ Can't encode all relevant knowledge about code above, or machine below

- ❖ Use PLL introspection & dynamic features:
 - intercept entry to “potentially specializable” function
 - inspect abstract syntax tree (AST) of computation looking for specializable *computation patterns*
 - (lookup in *catalog* of specializers)
- ❖ If a specializer is found, it can:
 - manipulate/traverse AST of the function
 - emit & JIT-compile ELL source code
 - dynamically link compiled code to PLL interp
- ❖ Fallback: just continue in PLL
- ❖ *Necessary features present in modern PLL's, but absent from older widely-used PLL's*

- ❖ Single “Fat” ILP-focused Tile Control Processor
- ❖ Multiple “Thin” Lane Control Processors embedded in vector-thread lane



- Tile Control Processor, Lane Control Processor, and Vector-Thread microthreads all run the same ISA, but microarchs optimized for different forms of parallelism