Steps Towards Heterogeneity and the UC Berkeley Parallel Computing Lab

Krste Asanović, Ras Bodik, Eric Brewer,
Jim Demmel, Armando Fox,
Tony Keaveny, Kurt Keutzer,
John Kubiatowicz, Nelson Morgan,
Dave Patterson, Koushik Sen,
David Wessel, and Kathy Yelick

UC Berkeley Par Lab

June, 2011

Power is the Problem

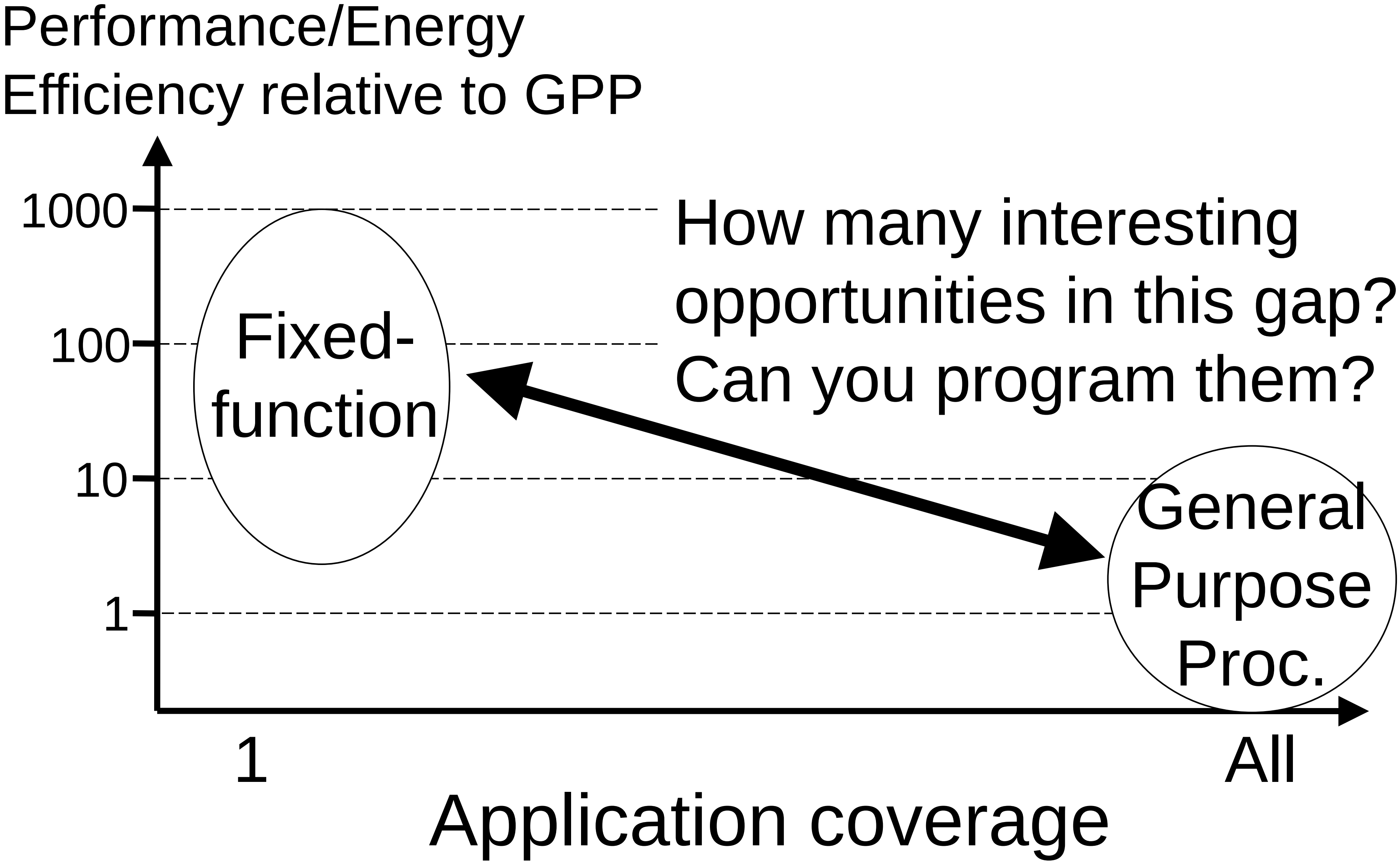
- Given limited power budget and slowly improving transistors, how can we continue increase performance enabled by Moore's Law?
  - "*This shift toward increasing parallelism is not a triumphant stride forward based on breakthroughs in novel software and architectures for parallelism; instead, this plunge into parallelism is actually a retreat from even greater challenges that thwart efficient silicon implementation of traditional uniprocessor architectures.*"*
- Same motivation for transition from homogenous multicore to heterogeneous multicore
- Lower energy at same performance as interesting as more performance?
- Do multicore advances make heterogeneity feasible?

**The Landscape of Parallel Computing Research: A View From Berkeley*, Dec 2006

What next?

- Future advancements in energy/op needs more than just parallelism
- Voltage-Frequency scaling of limited benefit in future technologies
  - Not much difference between Vdd and Vt
- Move to simpler general-purpose cores is a one-time gain
  - In smart phones, cores were already relatively simple
- More transistors per die than we can power at the same time (“*Utilization Wall* ”)

Efficiency versus Generality

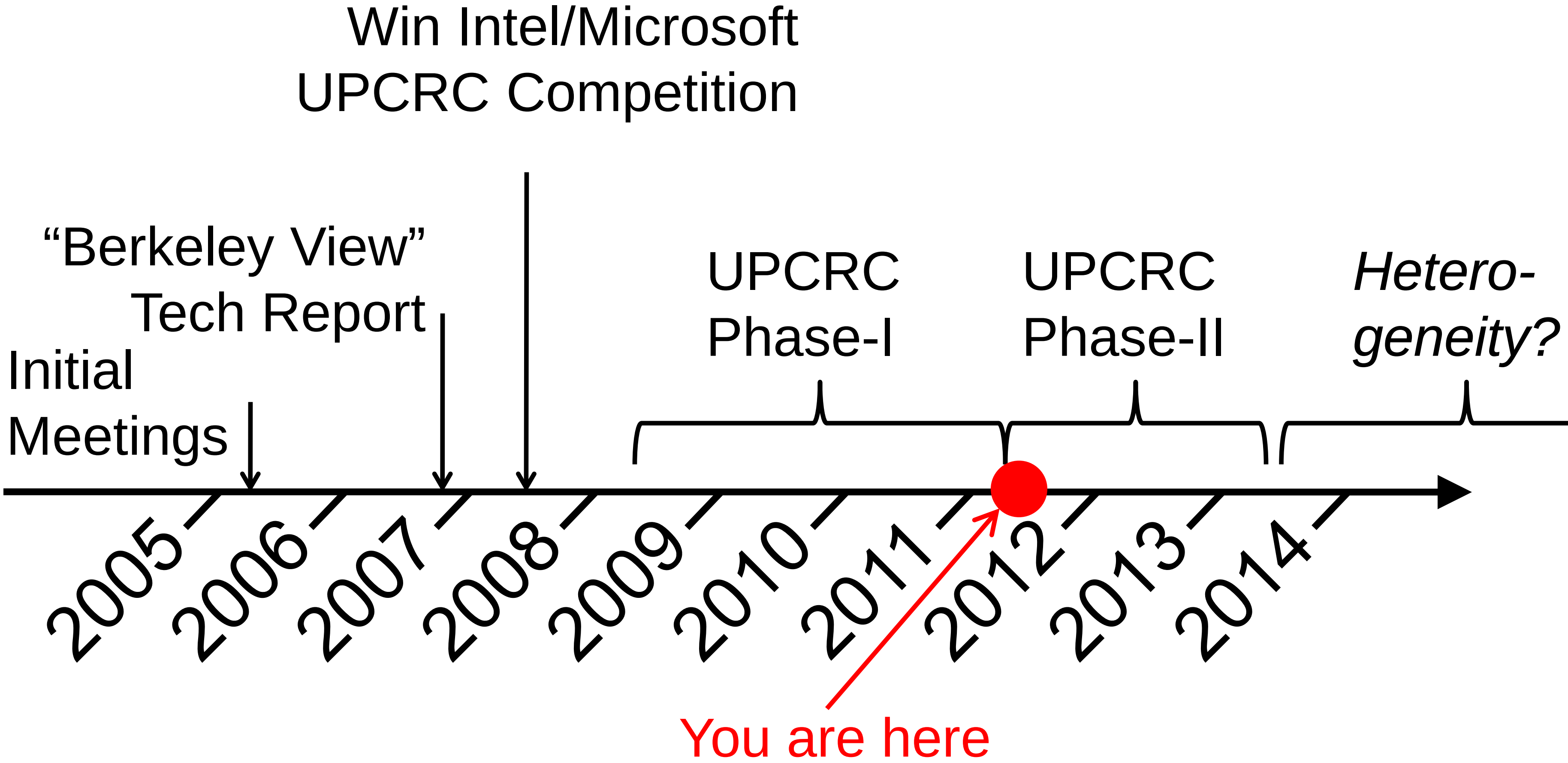


How many interesting opportunities in this gap? Can you program them?

Outline

- Why Heterogeneity?
- Quick Summary of Some Par Lab Advances
- Berkeley Hunch on Heterogeneity

Par Lab Timeline



Win Intel/Microsoft UPCRC Competition

"Berkeley View" Tech Report

Initial Meetings

UPCRC Phase-I

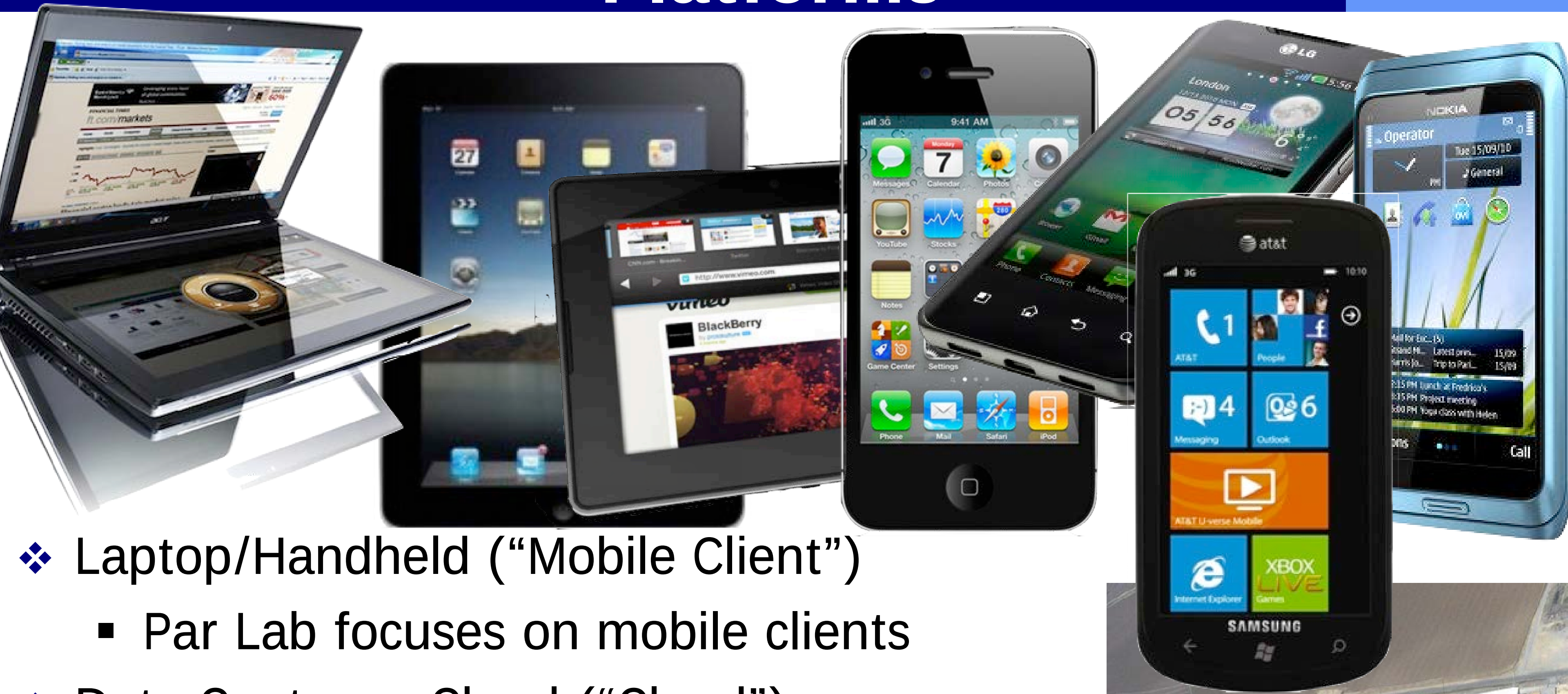
UPCRC Phase-II

Hetero-geneity?

2005 2006 2007 2008 2009 2010 2011 2012 2013 2014

You are here

Dominant Application Platforms



- Laptop/Handheld (“Mobile Client”)
  - Par Lab focuses on mobile clients
- Data Center or Cloud (“Cloud”)
  - RAD Lab/AMP Lab focuses on Cloud
- Both together (“Client+Cloud”)
  - ParLab-AMPLab collaborations

Par Lab's original "bets"

- ***Let compelling applications drive research agenda***
- Software platform: data center + mobile client
- Identify common programming patterns
- Productivity versus efficiency programmers
- Autotuning and software synthesis
- Build-in correctness + power/performance diagnostics
- OS/Architecture support applications, provide flexible primitives not pre-packaged solutions
- FPGA simulation of new parallel architectures: RAMP
- Co-located integrated collaborative center

Above all, no preconceived big idea
- see what works driven by application needs.

“Post Conceived” Big Ideas

- Communication-Avoiding Algorithms
  - *Large speedup of highly-polished algorithms by concentrating on data movement vs. FLOPs*
- Structural Patterns for Parallel Composition
  - *Good software architecture vs. invent new lang*
- Selective Embedded Just-In-Time Specialization (SEJITS)
  - *Productivity of Python with Efficiency of C++*
- Higher-level Hardware Description Lang (Chisel)
  - *More rapidly explore HW design space*
- Theme: Specialized HW requires Specialized SW

Communication-Avoiding Algorithms (Demmel, Yelick, Keutzer)

- Past algorithms: FLOPs expensive, Moves cheap
- From architects, numerical analysts interacting, learn that now Moves expensive, FLOPs cheap
- New theoretical lower bound of moves to FLOPs
- Success of theory and practice: real code now achieves lower bound of moves to great results
- Even Dense Matrix: >10X speedup over Intel MKL Multicore Nehalem and >10X speedup over GPU libraries for tall-skinny matrices (IPDPS 2011)
- Widely applicable: all linear algebra, Health app…

Types of Programming (or “types of programmer”)

	Example Languages	Example Activities
Domain-Level (No formal CS)	Max/MSP, SQL, CSS/Flash/Silverlight, Matlab, Excel	Builds app with DSL and/or by c nizing app f
Productivity-Level (Some CS cours	Python/Ruby/L Has # ala	ogramming rameworks, writes application frameworks (or apps)
Effici ncy-Level (MS in CS)	Java/C# C/C++/FORTRAN assembler	Uses hardware/OS primitives, builds programming frameworks (or apps)
Hardware/OS		Provides hardware primitives and OS services

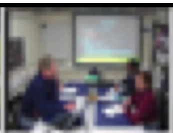
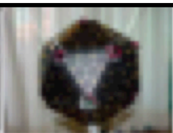
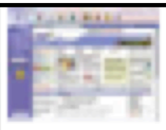
Where & how to make parallelism visible?

How to make parallelism visible?

- In a new general-purpose parallel language?
  - An oxymoron?
  - Won’t get adopted
  - Most big applications written in >1 language
- **Par Lab is betting on Computational and Structural Patterns at all levels of programming (Domain thru Efficiency)**
  - Patterns provide a good vocabulary for domain experts
  - Also comprehensible to efficiency-level experts or hardware architects
  - *Lingua franca* between the different levels in Par Lab

Motif (nee "Dwarf") Popularity (Red Hot → Blue Cool)

How do compelling apps relate to 12 motifs?

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

EECS
Electrical Engineering and Computer Sciences

"Our" Pattern Language (OPL-2010) (Kurt Keutzer, Tim Mattson)

BERKELEY PAR LAB

A plications

Structural Patterns

Pipe-and-Fi

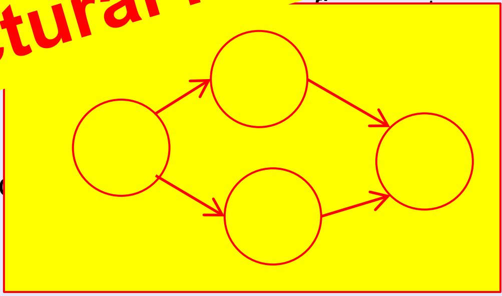
 Control

Event-Based/Implic
Invocation

Puppeteer

 ph

Structural Patterns



Computational Pa

Grap

 -State-Machines

 ch-and-

 ense-Linear-

Sparse-Linear

Unstructured-

Structured-Grids

 s

 ds

Monte-Carlo

Computational Patterns

$A = M \times V$

Concurrent Algorithm Strateg

Task-Parallelism
Divide and Conquer

 iscrete-Event
 eometric-Decomposition
 peculation

Refine Towards Implementation

Implementation Strategy Pa

SPMD
Data-Par/index-space

Fork/Join
Actors

Tas

 -Queue
 hared-map
Partitioned Graph

Distributed-Array
Shared-Data

Program structure

Data structure

Parallel Execution Patterns

MIMD
SIMD

Thread-Pool
Task-Graph

Transactions

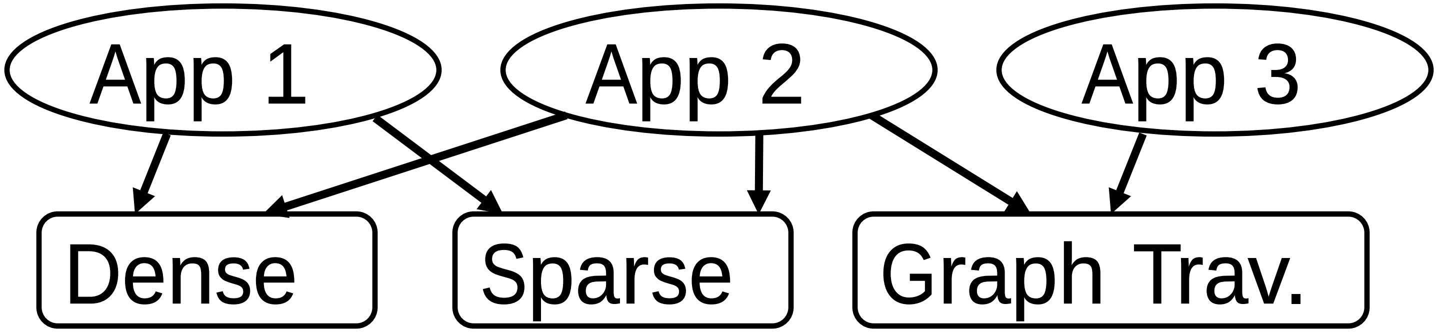
Concurrency Foundation constructs (not expressed as patterns)

Thread creation/destruction
Process creation/destruction

Message-Passing
Collective-Comm.

Point-To-Point-Sync. (mutual exclusion)
collective sync. (barrier)

Mapping Patterns to Hardware


App 1
App 2
App 3
Dense
Sparse
Graph Trav.

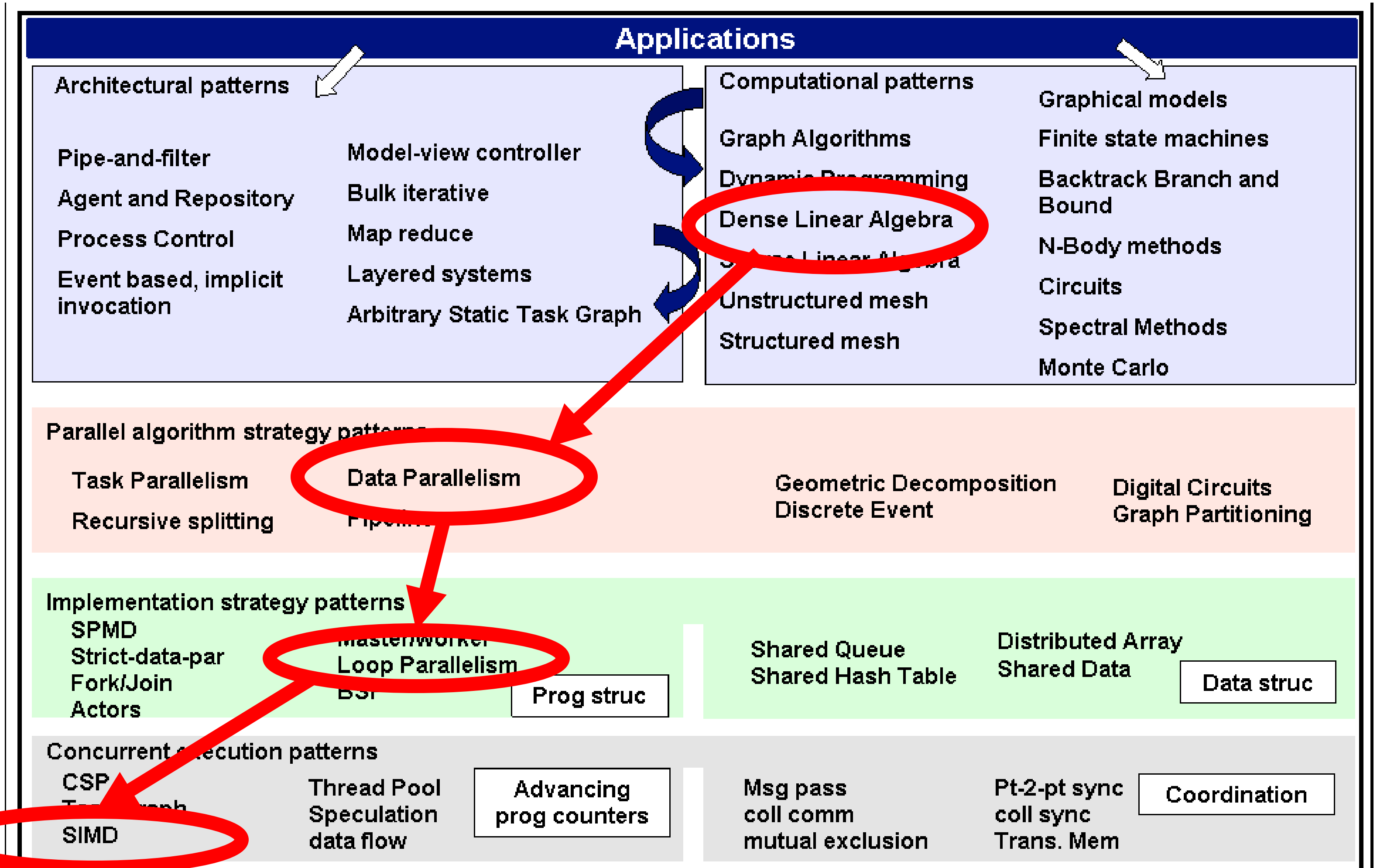
Only a few types of hardware platform

Multicore

GPU

“Cloud”

High-level pattern constrains space of reasonable low-level mappings



Applications

Architectural patterns

Pipe-and-filter
Agent and Repository
Process Control
Event based, implicit invocation
Model-view controller
Bulk iterative
Map reduce
Layered systems
Arbitrary Static Task Graph

Computational patterns

Graph Algorithms
Dynamic Programming
Dense Linear Algebra
Sparse Linear Algebra
Unstructured mesh
Structured mesh
Graphical models
Finite state machines
Backtrack Branch and Bound
N-Body methods
Circuits
Spectral Methods
Monte Carlo

Parallel algorithm strategy patterns

Task Parallelism
Recursive splitting
Data Parallelism
Pipeline
Geometric Decomposition
Discrete Event
Digital Circuits
Graph Partitioning

Implementation strategy patterns

SPMD
Strict-data-par
Fork/Join
Actors
Master/worker
Loop Parallelism
BSP
Prog struc
Shared Queue
Shared Hash Table
Distributed Array
Shared Data
Data struc

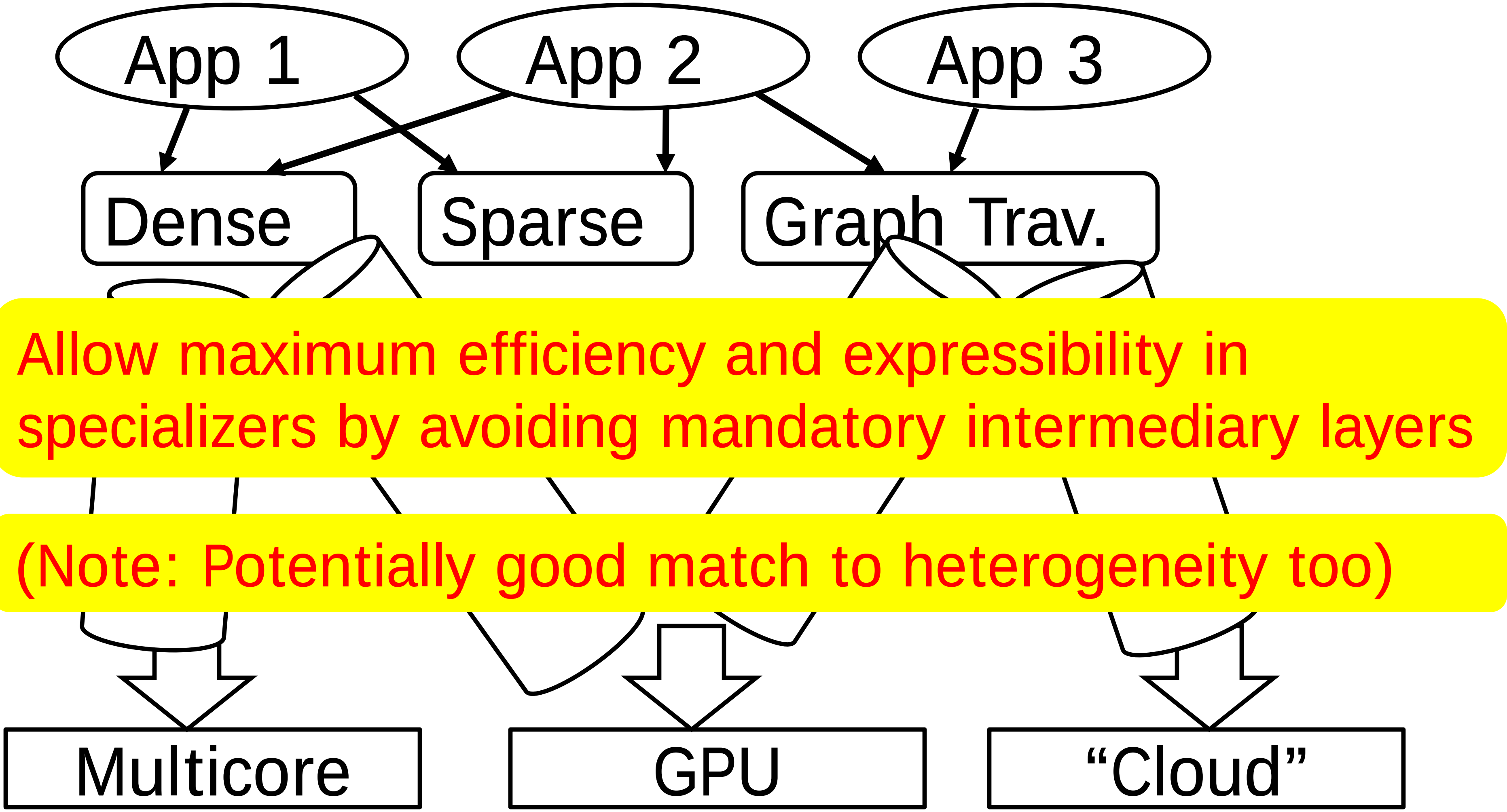
Concurrent execution patterns

CSP
Task graph
SIMD
Thread Pool
Speculation
data flow
Advancing prog counters
Msg pass
coll comm
mutual exclusion
Pt-2-pt sync
coll sync
Trans. Mem
Coordination

Figure 1: overall structure of OPL showing the five layer model. Implementation strategy patterns are divided into 2 sets; one describing a program's structure and the other data structures. The concurrent execution patterns are broken down into a set of patterns that "advance a program counter" and a set that coordinates the execution of parallel threads.

Specializers: Pattern-specific and platform-specific compilers

*aka. "**Stovepipes**"*



Allow maximum efficiency and expressibility in specializers by avoiding mandatory intermediary layers

(Note: Potentially good match to heterogeneity too)

Autotuning for Code Generation (Demmel, Yelick)

□ Problem: generating optimized code is like searching for needle in haystack; use computers rather than humans

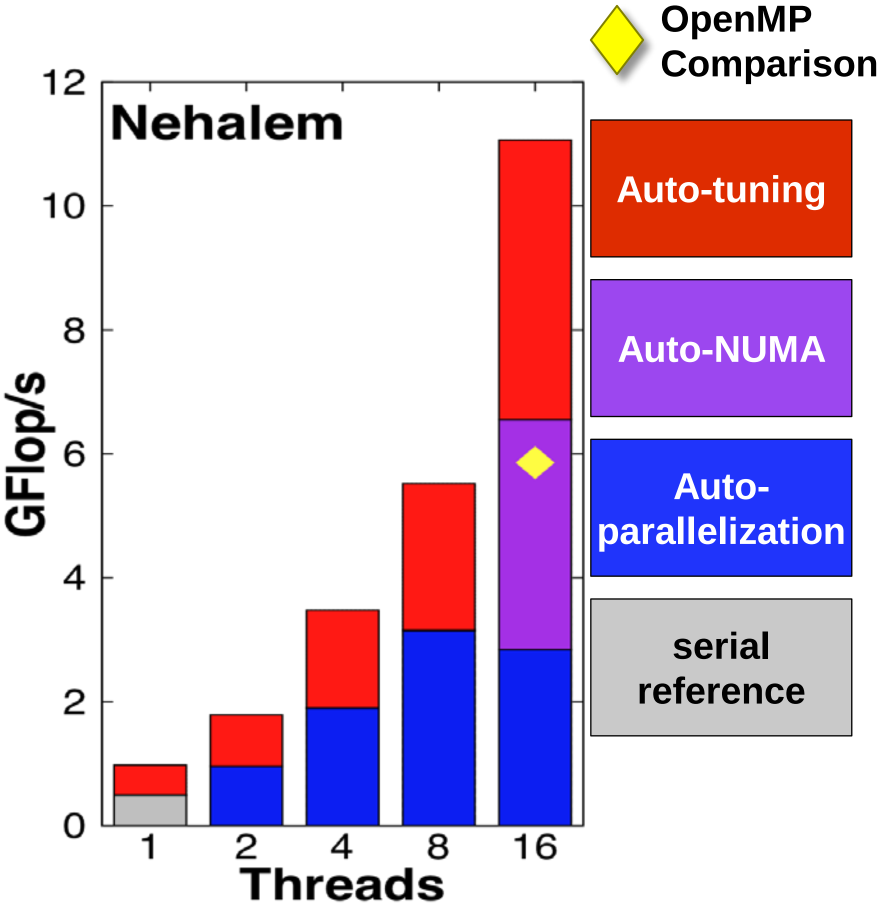
□ Auto-tuners approach: program *generates* optimized code and data structures for a “motif” (~kernel) mapped to some instance of a family of architectures (e.g., x86 multicore)

□ Use empirical measurement to select best performing

□ ParLab autotuners for stencils, sparse matrices, particle/mesh

□ ML to reduce search space?

□ (Note: Good for Heterogeneity?)



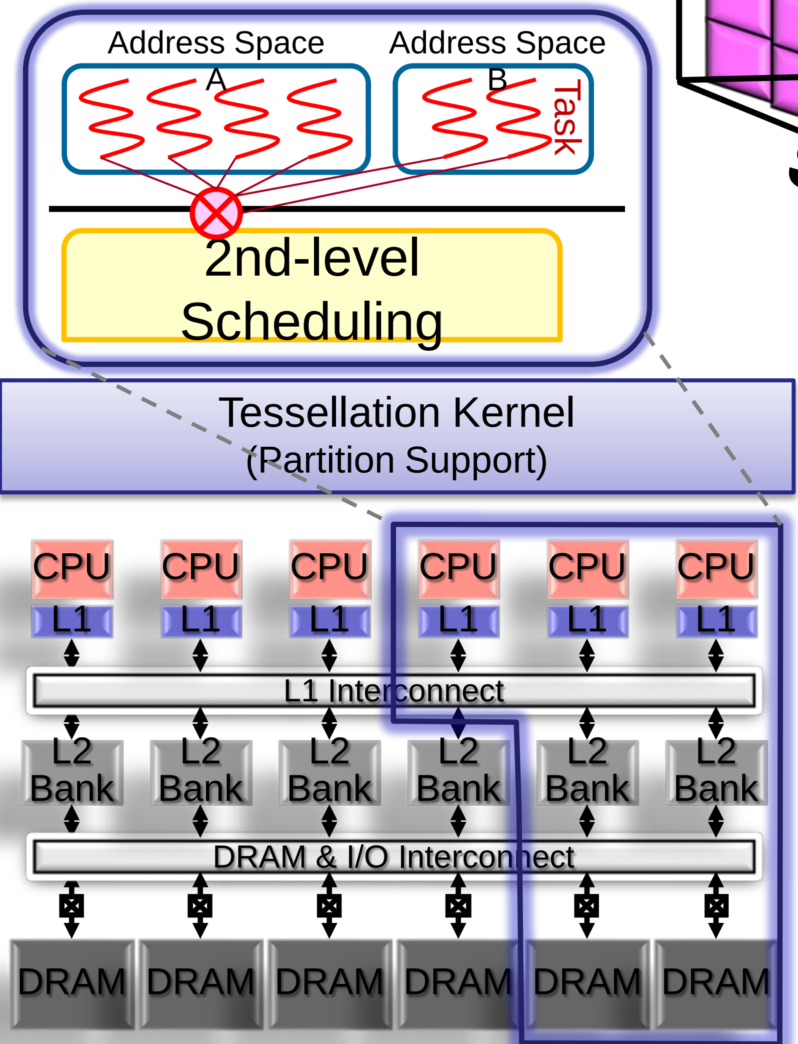
SEJITS: "Selective, Embedded, Just-In Time Specialization" (Fox)

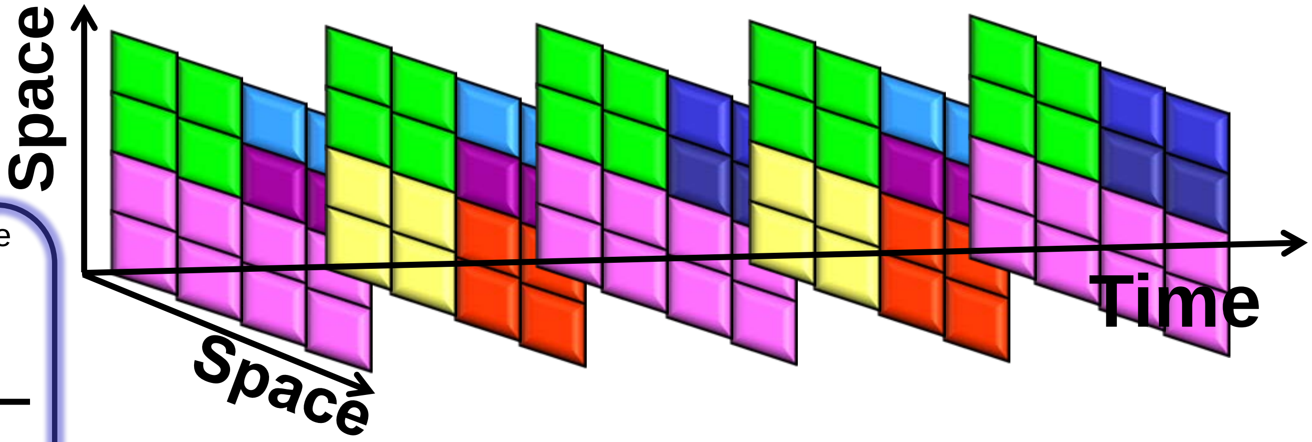
- SEJITS bridges productivity and efficiency layers through specializers embedded in modern high-level productivity language (Python, Ruby)
  - Embedded "specializers" use language facilities to map high-level pattern to efficient low-level code (at run time, install time, or development time)
  - Specializers can incorporate/package autotuners

Two ParLab SEJITS projects:

- **Copperhead**: Data-parallel subset of Python targeting GPUs
- **Asp**: "Asp is SEJITS in Python" general specializer framework
  - Provide functionality common across different specializers
- (Note: SEJITS helpful for Heterogeneity too?)

Tessellation OS: Space-Time Partitioning + 2-Level Scheduling (Kubiatowicz)





1st level: OS determines coarse-grain allocation of resources to jobs over space and time

2nd level: Application schedules component tasks onto available “harts” (hardware thread contexts) using Lithe

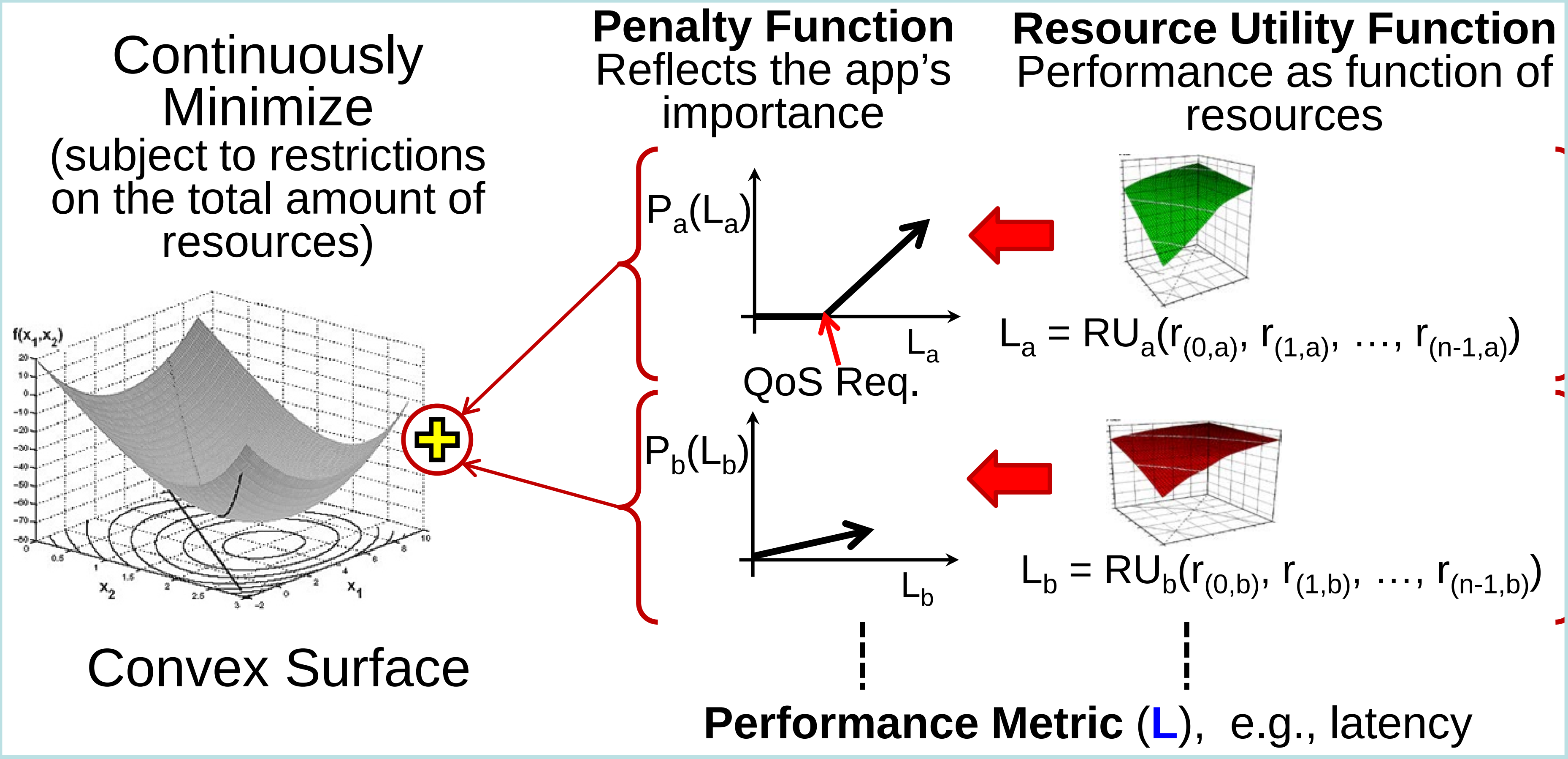
Adaptive Resource Management

- Resource allocation is about adapt/model/observe loop
- Pacora: using convex optimization as an instance to adapt to changing circumstances
- Each process receives a **vector of basic resources** dedicated to it
  - fractions of cores, cache slices, memory pages, BW
- Allocate minimum for QoS requirements
- Allocate remaining to meet system-level objective
  - best performance, lowest energy, best user experience

EECS
Electrical Engineering and Computer Sciences

Resource Management using Convex Optimization (Bird, Smith)

BERKELEY PAR LAB



Continuously Minimize (subject to restrictions on the total amount of resources)

Convex Surface

Penalty Function
Reflects the app's importance

Resource Utility Function
Performance as function of resources

$P_a(L_a)$

L_a

QoS Req.

$L_a = RU_a(r_{(0,a)}, r_{(1,a)}, \ldots, r_{(n-1,a)})$

$P_b(L_b)$

L_b

$L_b = RU_b(r_{(0,b)}, r_{(1,b)}, \ldots, r_{(n-1,b)})$

Performance Metric (L), e.g., latency

- (Note: Dynamic Resource Management Optimization needed for Heterogeneity too)

Chisel: Hardware Design Language (Asanović, Bachrach)

- Chisel (Constructing Hardware in a Scala Embedded Language) under active development
  - Generate C simulator + FPGA emulation + ASIC synthesis from one RTL description
  - Supports higher-level libraries
- Chisel compiles C-simulation of RTL RISC-V processor design in 12 seconds, runs at 4.5MHz on 3.2GHz Nehalem
  - FPGA tools take >1 hour to map same design, runs at 33MHz on FPGA.
- (Note: Helps for Heterogeneous HW too)

Theme: Specialized HW requires Specialized SW

- Patterns specialize general-purpose programming by giving programming constructs that are specialized for the 12 patterns
- Programmer composes functionality at high-level using productivity language
- Specializers are tools that specialize the generic compiler for each of the 12 patterns
  - A stovepipe specializes the general-purpose language+compiler combination into a pattern+specializer combination
- System composes resource usage using 2-level scheduling: Tessellation OS + Lithe at user-level

Theme: Specialized HW requires Specialized SW

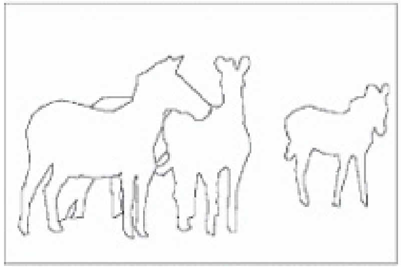
High-Level Description	Output	Name of Tool
Software in Our Pattern Language (OPL)	Software Architecture using Structural Patterns in ASP/Copperhead	ASP/Copperhead Compiler (DSLs embedded in Python)
Hardware in Berkeley Hardware Pattern Language (BHPL)	C++ simulator, FPGA bits, Synthesizable Verilog	Chisel Compiler (DSL embedded in Scala)
MUD/Ale programs	Parallel Layout Engine	MUD/Ale compiler

Berkeley Bet: Pattern-specific high-level programs can be automatically and dynamically specialized to pattern-specific hardware

Par Lab Apps

- **What are the compelling future workloads?**
  - o Need apps of future vs. legacy to drive agenda
  - o Improve research even if not the real killer apps
- **Music**: 3D Enhancer, Hearing Aid, Novel UI
- **Parallel Browser**: Layout, Scripting Language
- **Computer Vision**: Segment-Based Object Recognition, Poselet-Based Human Detection
- **Health**: MRI Reconstruction, Stroke Simulation
- **Speech**: Automatic Meeting Diary

Vision Acceleration (Kurt Keutzer)



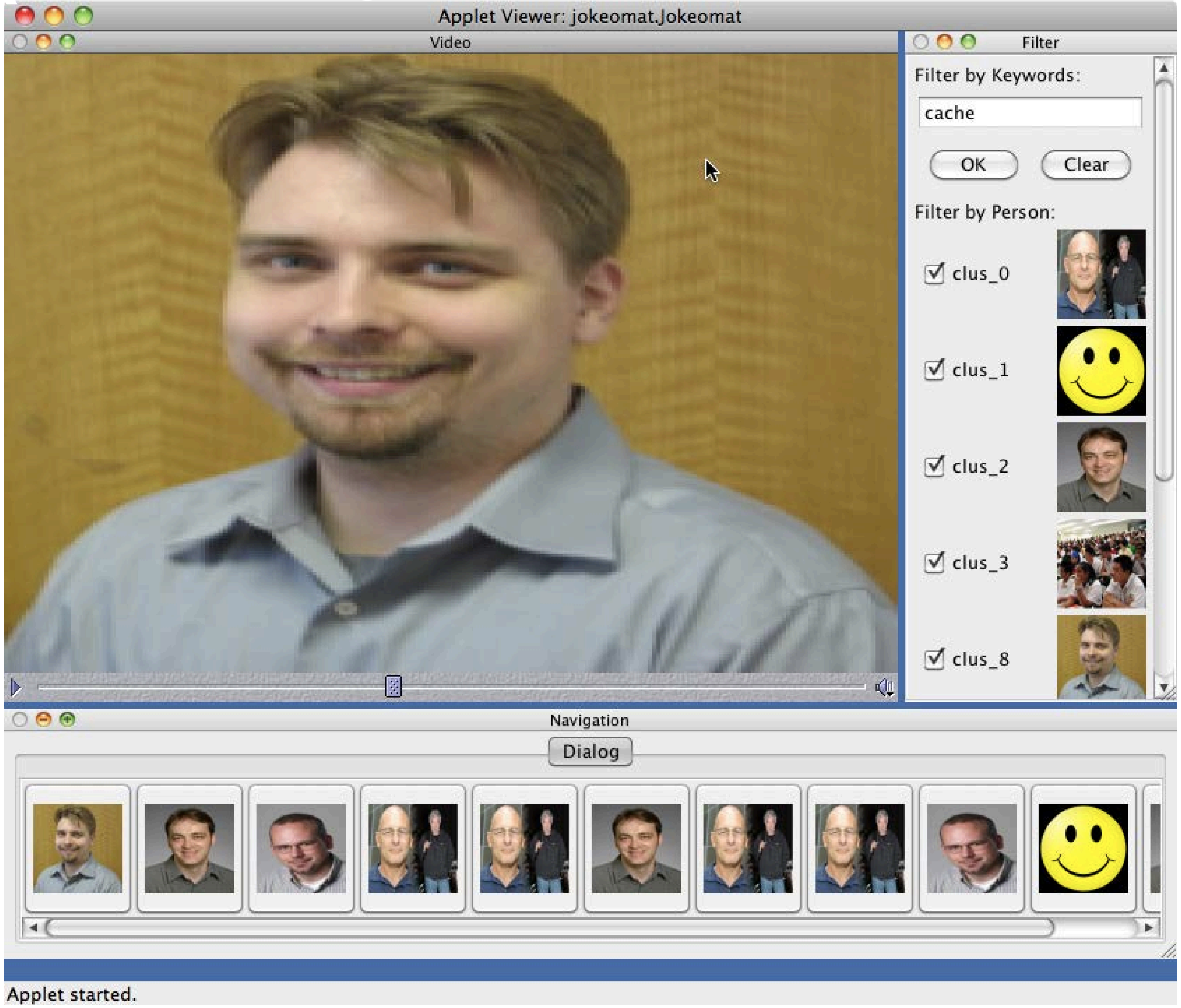
- Parallelizing Computer Vision (image segmentation)
- Problem: Malik's highest quality algorithm was 5.5 minutes / image on new PC
- Good SW architecture+talk within Par Lab on to use new algorithms, data structures
  - Current result: 1.8 seconds / image on manycore
- ~ 150X speedup
  - Factor of 10 quantitative change is a qualitative change
- Enabled propagation of best in class algorithm

Fast Pediatric MRI (Kurt Keutzer)

- 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 2010 for further clinical study

Speech: Meeting Diarist (Nelson Morgan, Gerald Friedland, ICSI/UCB)



Won ACM Multimedia Grand Challenge 2009

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

Parallelization of Diarization

- Five versions (so far):

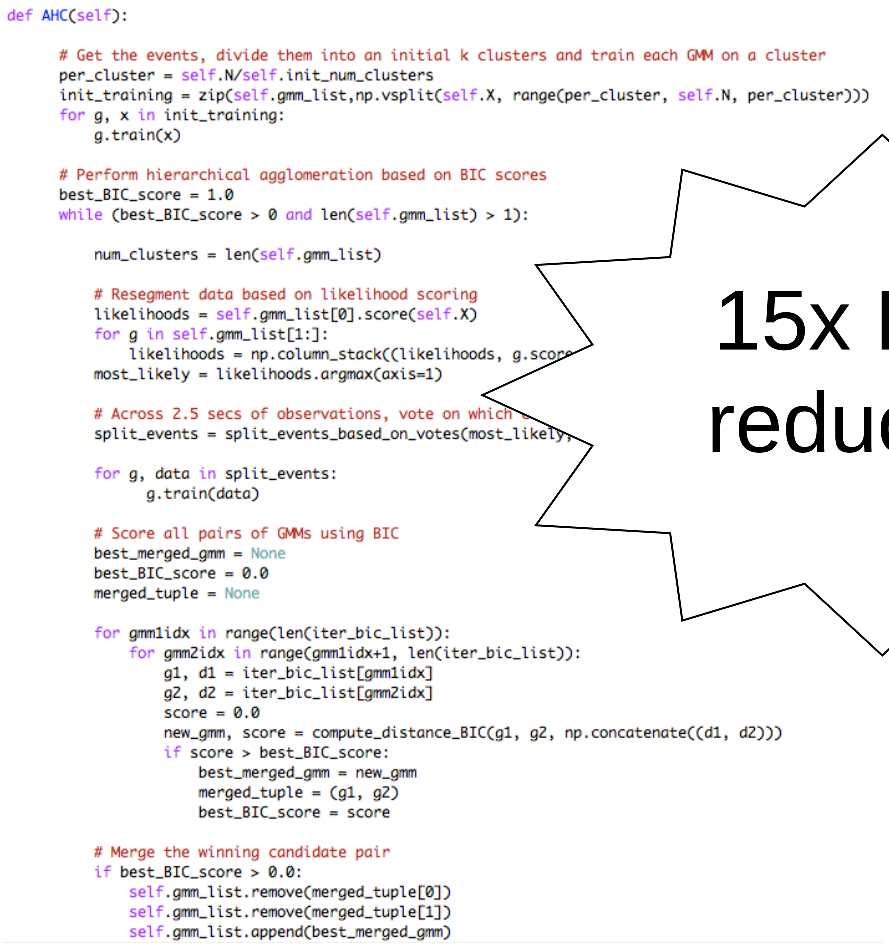
1. Initial code (2006): 0.333 x realtime
   (i.e., 1 hour audio = 3 hours processing)
2. Serially optimized (2008): 1.5 x realtime
3. Parlab retreat summer 2010: Multicore+GPU parallelization: 14.3 x realtime
4. Parlab retreat winter 2011: GPU-only parallelization 250 x realtime
   (i.e., 1 hour audio = 14.4 sec processing)
   - -> Offline = online!
5. Parlab retreat June 2011: SEJITized! [1]

[1] H. Cook, E. Gonina, S. Kamil, G. Friedland, D. Patterson, A. Fox. CUDA-level Performance with Python-level Productivity for Gaussian Mixture Model Applications. USENIX HotPar Workshop, 2011.

Speaker Diarization in Python

Python: 45 LOC

C



```
def AHC(self):

    # Get the events, divide them into an initial k clusters and train each GMM on a cluster
    per_cluster = self.N/self.init_num_clusters
    init_training = zip(self.gmm_list,np.vsplit(self.X, range(per_cluster, self.N, per_cluster)))
    for g, x in init_training:
        g.train(x)

    # Perform hierarchical agglomeration based on BIC scores
    best_BIC_score = 1.0
    while (best_BIC_score > 0 and len(self.gmm_list) > 1):

        num_clusters = len(self.gmm_list)

        # Resegment data based on likelihood scoring
        likelihoods = self.gmm_list[0].score(self.X)
        for g in self.gmm_list[1:]:
            likelihoods = np.column_stack((likelihoods, g.score
        most_likely = likelihoods.argmax(axis=1)

        # Across 2.5 secs of observations, vote on which c
        split_events = split_events_based_on_votes(most_likely,

        for g, data in split_events:
              g.train(data)

        # Score all pairs of GMMs using BIC
        best_merged_gmm = None
        best_BIC_score = 0.0
        merged_tuple = None

        for gmm1idx in range(len(iter_bic_list)):
            for gmm2idx in range(gmm1idx+1, len(iter_bic_list)):
                g1, d1 = iter_bic_list[gmm1idx]
                g2, d2 = iter_bic_list[gmm2idx]
                score = 0.0
                new_gmm, score = compute_distance_BIC(g1, g2, np.concatenate((d1, d2)))
                if score > best_BIC_score:
                    best_merged_gmm = new_gmm
                    merged_tuple = (g1, g2)
                    best_BIC_score = score

        # Merge the winning candidate pair
        if best_BIC_score > 0.0:
            self.gmm_list.remove(merged_tuple[0])
            self.gmm_list.remove(merged_tuple[1])
            self.gmm_list.append(best_merged_gmm)
```

15x LOC reduction

.....

Results – Specializer Overhead

- 15x reduction in lines of code (Python vs. C/Cuda)
- Python AHC code is within **1.25x** of pure C/CUDA implementation performance
  - C/CUDA – 250x realtime on GPU
  - SEJITized AHC – 200x realtime on GPU
- Time lost in:
  - Data copying overhead from CPU to GPU
  - Outer loop and GMM creation in Python
  - GMM scoring in Python
- Initial retarget to Cilk++ – ~ 100x realtime on Nehalem Multicore

Outline

- Why Heterogeneity?
- Quick Summary of Some Par Lab Advances
- Berkeley Hunch on Heterogeneity

Earlier Successful Examples: FPUs, Vector Units

- FPUs are specialized hardware
  - Only useful for floating-point code
  - Easy for programmers to use because already had programming model
  - Needed some tuning to use effectively
- Vector units are specialized hardware
  - Only useful for data-parallel code
  - Easy for programmers to use, already had loop nests in application code
  - Needed some tuning to use effectively, but had compiler feedback

The Opportunity

- Intel researchers picked 14 throughput oriented kernels to benchmark multicore vs. GPU
    - Lee et al “Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU,” ISCA June 2010.
- Collision Detection Application ran 15.2X faster on NVIDIA GPU vs. Intel Nehalem due to

1. GPU Gather-Scatter addressing
2. More GPU hardware for transcendental functions

The Opportunity

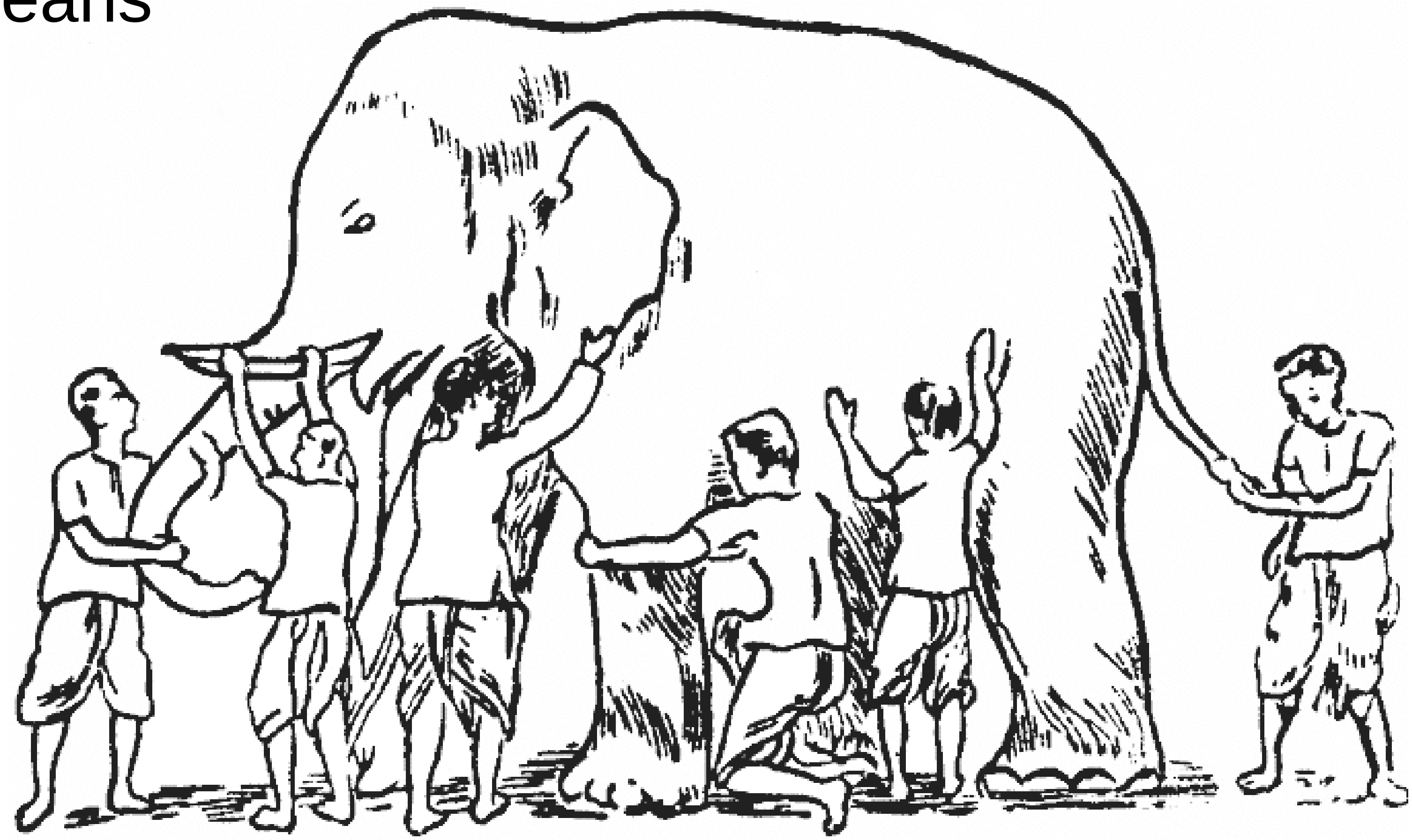
- Example of H.264 video decoder ***[Hameed et al, ISCA 2010]***
- Highly tuned software H.264 decoder vs. fixed-function ASIC
- Normalized to 130nm technology

	Area (mm^2)	Frames/Second	Joules/Frame
Pentium-4 (720x480)	122	30	0.742
Pentium-4 (1280x720)	122	11	2.023
ASIC (1280x720)	8	30	0.004

- 45X throughput/area advantage
  - (3x frame rate, 15x less area)
- 500X energy/task advantage

Heterogeneity?

- Much agreement that heterogeneity comes next
- But many different views on what heterogeneity means



Heterogeneity Research

- Large design space
- Lots of earlier work
  - many failures (e.g., NeXT DSP, IBM Cell, reconfigurable computing)
  - few successes (e.g., GP-GPU)
- Used in niche applications now, but looks inevitable for widespread hardware adoption
- How can software keep up?
- Much confusion in industry

- Sound familiar? => Berkeley View on …

Specialization >> Heterogeneity

- Do not need heterogeneity to benefit from specialization
- Heterogeneity is one way to deliver specialization
- Alternative approaches:
  - Homogeneous cores with wide variety of coprocessors/extended instruction sets
  - Homogeneous reconfigurable cores
- Can use all of the above in one system

- Research question: When does core heterogeneity make sense versus richer homogeneous cores?

Structure of Heterogeneity

- How are heterogeneous components arranged?
- Temporal heterogeneity
  - One core changes over time (voltage, frequency, runtime configurable)
- Spatial heterogeneity
  - Hetero. computers in datacenter (Niagara + Sandy Bridge)
  - Hetero. nodes in single address space (Cray XT6 nodes)
  - Hetero. nodes on one motherboard (CPU + discrete GPU)
  - Hetero. nodes on one chip (SoC CPU+DSP+GPU)
  - Hetero. coprocessors (Vector Units, Conservation Cores)
  - Hetero. functional units (AES instructions)

Berkeley Bet: Focus on problem on one die

Types of Specialization

Less specialized

- Same core design, different VF operating points
- Same core design, runtime configurable components
- Same ISA, different µarchitectures
- Variants of same ISA (subsets, different extensions)
- Completely different ISAs
- Programmable logic (no ISA)
- Fixed-function accelerators (no programming)

More specialized

Operating-Point Specialization

- One core operates at different Voltage/Frequency over time (temporal specialization)
- Multiple cores experience different Voltage/Frequency at same time (spatial specialization)
- Where to manage?
  - Purely in hardware power management unit (PMU)?
  - In OS?
  - With application help?

Berkeley Bet: Useful tool, can be used with any architecture to trade performance and energy/op, but benefit decreasing with shrinking transistors

Specialization through Runtime Configuration

One ISA, one microarch, but provide runtime configurable components

- Issue width
  - Reduce active issue width to match ILP
- Cache capacity
  - activate fewer ways if small working set
  - can also reduce number of sets
- Turn attached units on and off
  - Floating-point units
  - SIMD engines
  - Attached coprocessors
- Prefetchers, how aggressive, what patterns to prefetch
- Multithreading, number of active threads

Specialized µArchitectures

One ISA, different µarchitectures

- “Fat” out-of-order vs. “Thin” in-order
- Lightly threaded (1-2) vs. heavily threaded (4-128)
- Wide SIMD (256+bits) vs. Narrow SIMD (<= 64bits)
- Few pipestages (latency critical) vs. many pipestages (throughput-centric)
- Note: some ISAs better than others to get large dynamic range

ISA Specialization

- ISA extensions
  - E.g., crypto operations (+instructions)
- Slave units
  - E.g., vector units (+state, + instructions)
- Autonomous Coprocessors
  - E.g. conservation cores (+state, +instructions, + control)

Multiple Different ISAs

CPU vs. GPU vs. DSP vs. …

- Implies heterogeneous cores
- Probably different programming models
- Any technical reason this is needed (above µarch specialization or different ISA extensions) or just business/IP ?

Berkeley Bet: Where there is an ISA, can usually use same base ISA, but ISA not where action is

Programmable Logic

- FPGAs
- Programmable logic coprocessors
  - GARP, Stretch, Convey
- Successful at accelerating some kinds of compute in niche areas
- Programming productivity has been a challenge.

Fixed-Function Accelerators

- Avoid instruction stream overhead by building fixed-function hardware
  - E.g., crypto engine
- Not programmable, but maybe parameterizable
- Very high efficiency for one kernel
- Software accesses through API calls

Berkeley Bet: Important component of all future systems, but not a focus of our research effort

End of RISC?

- If have 10 specialized cores each aimed at 10% of workload, then ISAs likely to grow?

Specialized Memory and Interconnect too

- Coherence protocols
- Software-managed memory
- Synchronization primitives
- On-the-fly compression/decompression
- Easier to make configurable, since switching and translation/virtualization already part of the design

Berkeley Bet: At least as important as specialized cores

Software Challenges

- Can the benefit of hardware specialization be widely obtained for third-party application developers (ISVs)?
- Can *most* programmers leverage specialized hardware - portably, productively, efficiently, and correctly?
- And have their software automatically take advantage of advances in specialized hardware?

Berkeley Bet: Pattern-specific high-level programs can be automatically and dynamically specialized to pattern-specific hardware

Reasons for Hope, Building on Par Lab

- Pattern-based view of software architecture provides basis for structuring heterogeneous software stack
- Programmers already calling out patterns in their code to use pattern-specific optimizing specializers
- Match specialized hardware to patterns already called out in programmers code
- Which programmers affected by heterogeneity?

Types of Programming (or "types of programmer")

	Example Languages	Example Activities
Domain-Level (No formal CS)	Max/MSP, SQL, CSS/Flash/Silverlight, Matlab, Excel	Builds app with DSL and/or by customizing app framework
Productivity-Level (Some CS courses)	Python/Ruby/Lua Haskell/OCamL/F# Scala	Uses programming frameworks, writes application frameworks (or apps)
Efficiency-Level (MS in CS)	Java/C# C/C++/FORTRAN assembler	Uses hardware/OS primitives, builds programming frameworks (or apps)
Hardware/OS		Provides hardware primitives and OS services

Idea: Pattern-Specific VMs

- For porting SW, can provide *pattern-specific virtual machines (PSVMs)* to hide hardware differences
  - For each pattern, define new abstract ISA that encodes operations and data access patterns
  - Family of VMs designed together as a coherent whole
  - E.g., for DLP, encode loops with independent iterations
  - E.g., for circuits, encode bit-level dataflow graph
- Each HW platform provides JITs/autotuning to map to available accelerator
  - Can map to GPP if no accelerator available, or if instance of pattern doesn’t fit on accelerator

Berkeley Bet: Innovate at pattern level, not at binary ISA

Thought Experiment

- If Intel had defined a data-parallel VM plus effective JIT, maybe could have avoided:
  - MMX
  - SSE +2,3,4
  - AVX
  - LNI

- Already used by GPU vendors to hide hardware ISA changes (“PTX”)

Legacy Code and Hetero

- Look for events that indicate translate x86 binary from running on general purpose “Productivity Cores” to run on specialized “Efficiency Cores”
  - Execute Transcendental instructions
  - Execute SSE instructions
  - Reads CPUID to decide which version to run
  - Instruction Level Parallelism counters too high
  - Memory counters indicate bottleneck
  - …

Research Questions

- How much benefit is available across our workloads?
  - Some codes constrained by memory traffic or low parallelism
- Are there new programmable architectures that capture a significant part of space not already covered?
- Managing hardware design cost and support software development cost (per-accelerator JIT)?

Summary

- Par Lab Theme: Specialized HW needs Specialized SW
- Power forced Uniprocessor => Multicore, soon Homogeneous to Heterogeneous Multicore
  - Must make ~invisible to most programmers
- Multicore Advances help Hurtle to Heterogeneity?
  - Pattern based innovations: SW architecture
  - Communication-Avoiding Algorithms
  - Dynamic Selective Embedded JIT Specialization & Autotuning
  - OS dynamic resource allocation optimization
  - Chisel high-level hardware description

Questions? (FYI: Par Lab References)

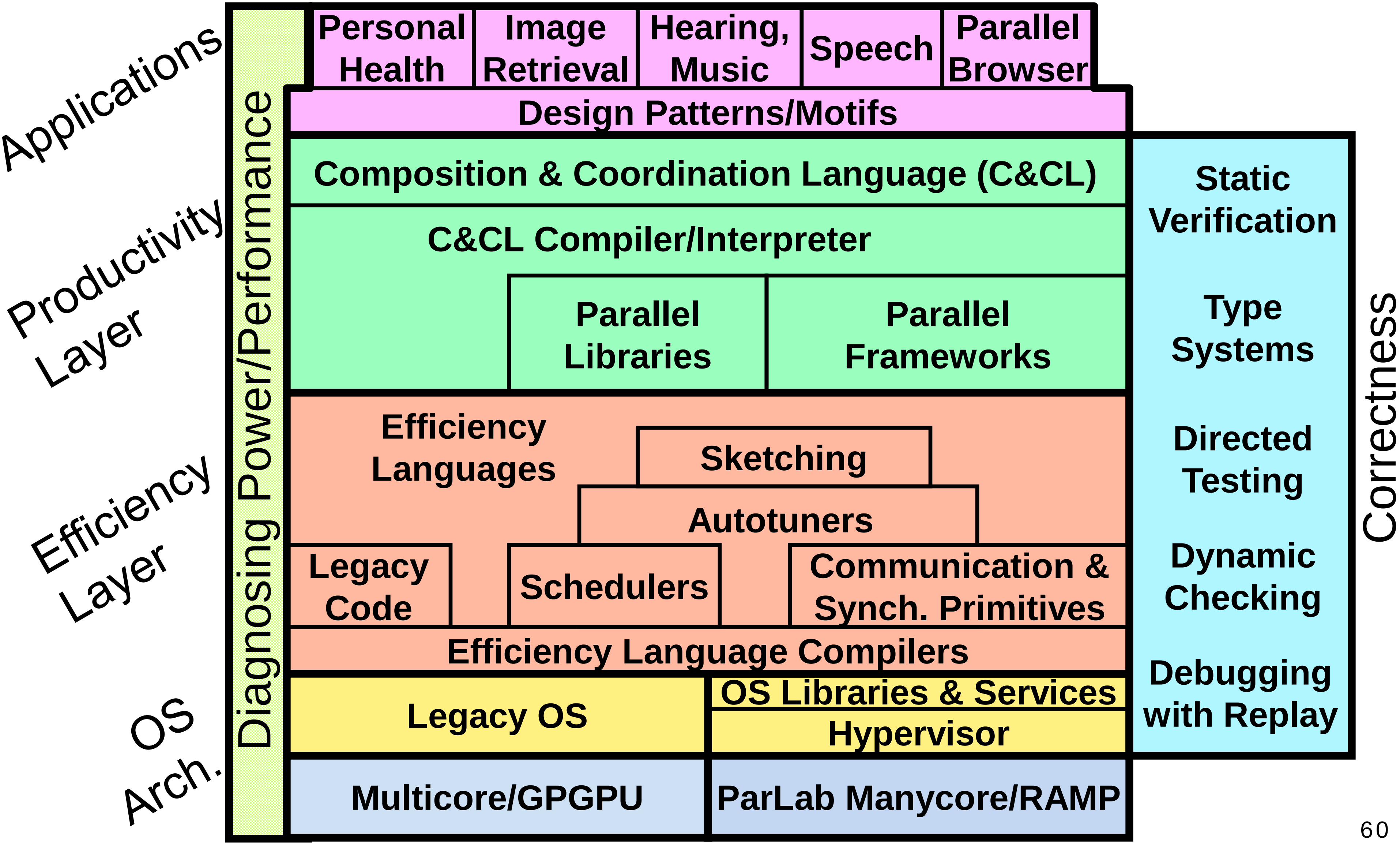
- See parlab.eecs.berkeley.edu/publications
- Asanović, K., R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, K. Yelick., "A View of the Parallel Computing Landscape," *Communications of the ACM*, vol. 52, no. 10, October 2009.
- Bird, S., B.Smith, PACORA: Performance-Aware Convex Optimization for Resource Allocation
- In the 3rd USENIX Workshop on Hot Topics in Parallelism (HotPar), May 2011.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 (at the 18th Int'l Conf. on Parallel Architectures and Compilation Techniques)*, Raleigh, North Carolina, November 2009.
- Tan, Z., A. Waterman, S. Bird, H. Cook, K. Asanović, and D. Patterson, "A Case for FAME: FPGA Architecture Model Execution," ISCA, 2010.

EECS
Electrical Engineering and
Computer Sciences

Par Lab Research Overview

BERKELEY PAR LAB

Easy to write correct programs that run efficiently on manycore



Applications

Productivity Layer

Efficiency Layer

OS Arch.

Diagnosing Power/Performance

Personal Health

Image Retrieval

Hearing, Music

Speech

Parallel Browser

Design Patterns/Motifs

Composition & Coordination Language (C&CL)

C&CL Compiler/Interpreter

Parallel Libraries

Parallel Frameworks

Efficiency Languages

Sketching

Autotuners

Legacy Code

Schedulers

Communication & Synch. Primitives

Efficiency Language Compilers

Legacy OS

OS Libraries & Services

Hypervisor

Multicore/GPGPU

ParLab Manycore/RAMP

Static Verification

Type Systems

Directed Testing

Dynamic Checking

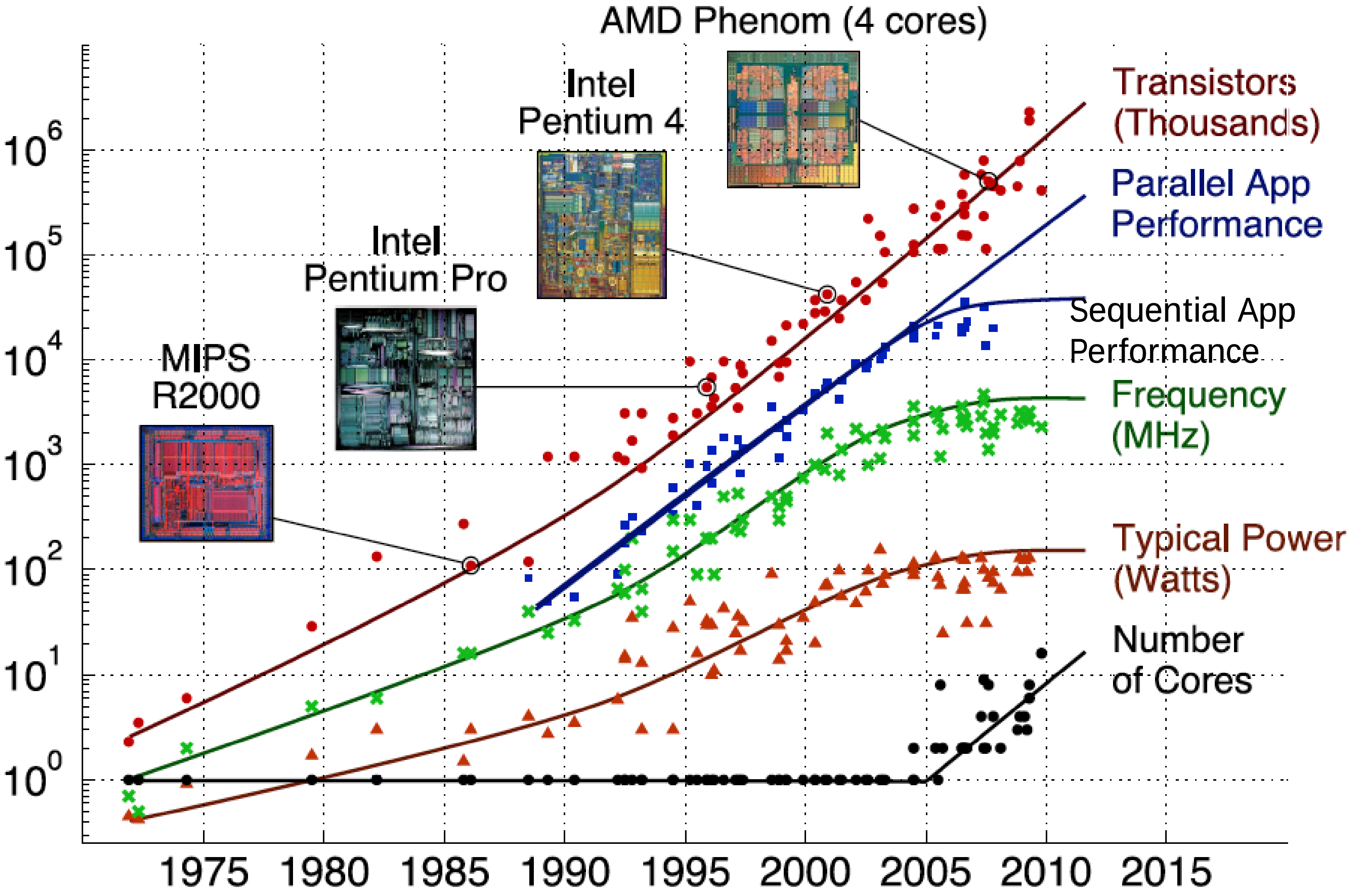
Debugging with Replay

Correctness

EECS
Electrical Engineering and
Computer Sciences

Transition to Multicore

BERKELEY PAR LAB



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

Needed a Fresh Approach to Parallelism

- Berkeley researchers from many backgrounds meeting since Feb. 2005 to discuss parallelism
  - Krste Asanović, Eric Brewer, Ras Bodik, Jim Demmel, Kurt Keutzer, John Kubiatowicz, Dave Patterson, Koushik Sen, 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")
- Goal: To enable most programmers to be productive writing efficient, correct, portable SW for 100+ cores & scale as cores increase every 2 years (!)

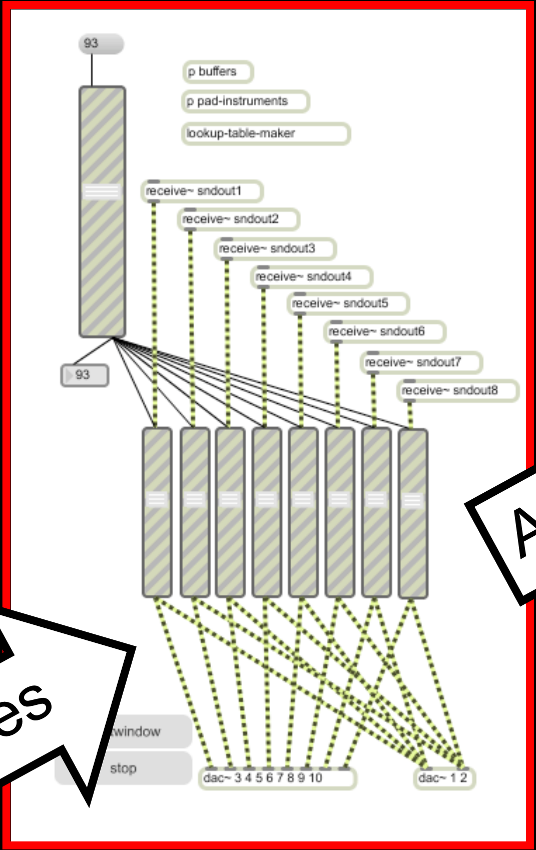
Traditional Parallel Research Project

- Past parallel projects often dominated by hardware architecture:
  - *This is the one true way to build computers, software must adapt to this breakthrough!*
  - E.g., ILLIAC IV, Thinking Machines CM-2, Transputer, Kendall Square KSR-1, Silicon Graphics Origin 2000 …
- Or sometimes by programming language:
  - *This is the one true way to write programs, hardware must adapt to this breakthrough!*
  - E.g., Id, Backus Functional Language FP, Occam, Linda, HPF, Chapel, X10, Fortress …
- Applications usually an afterthought

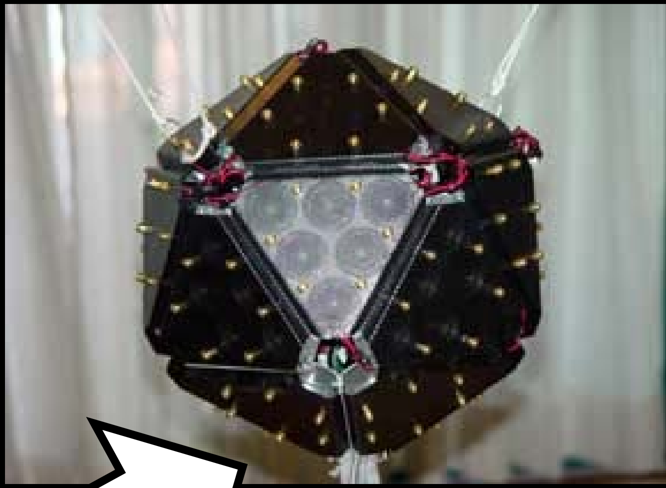
Music Application (David Wessel, CNMAT@UCB)

New user interfaces with pressure-sensitive multi-touch gestural interfaces

Gestures



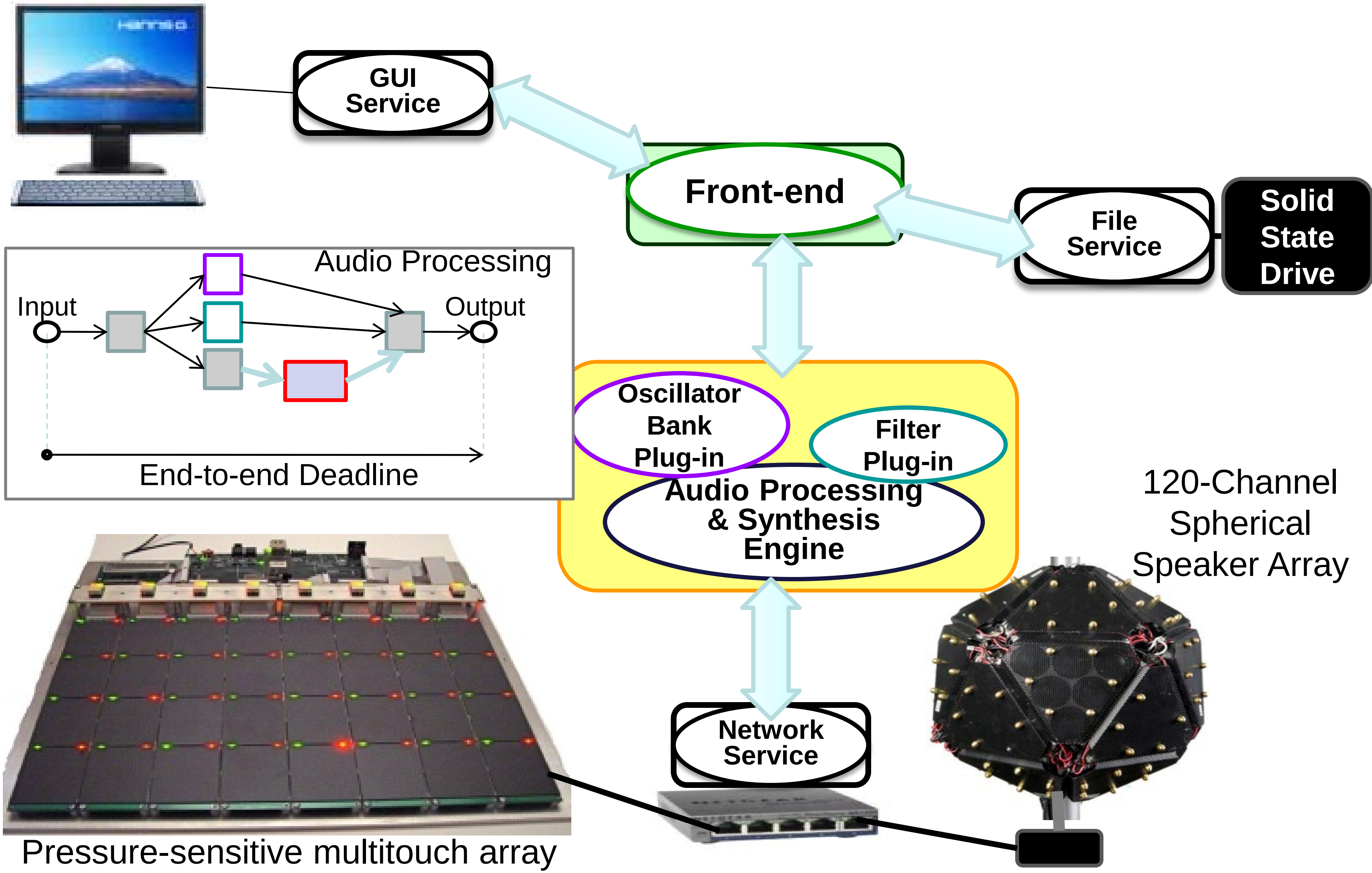
Audio



120-channel speaker array

Programmable virtual instrument and audio processing

Music Software Structure



GUI Service

Front-end

File Service

Solid State Drive

Audio Processing

Input

Output

End-to-end Deadline

Oscillator Bank Plug-in

Filter Plug-in

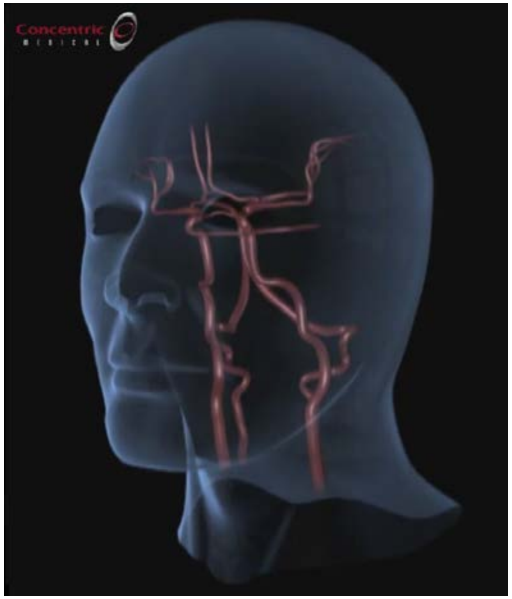
Audio Processing & Synthesis Engine

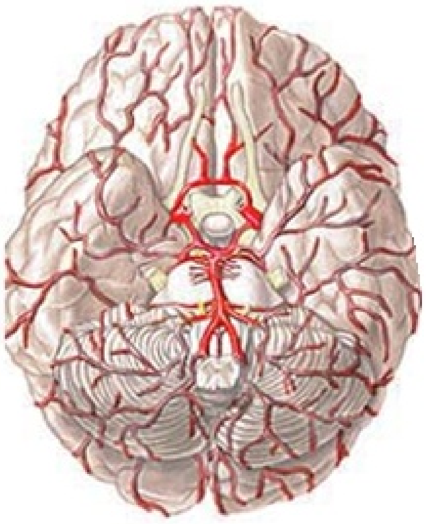
Network Service

120-Channel Spherical Speaker Array

Pressure-sensitive multitouch array

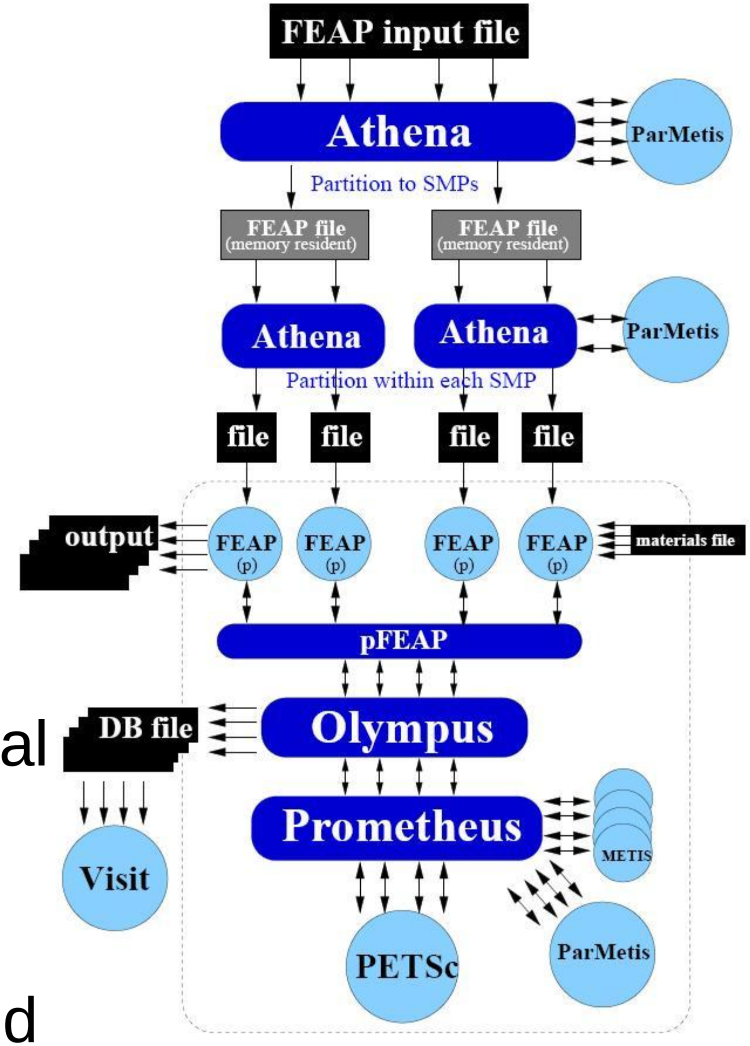
Health Application: Stroke Treatment (Tony Keaveny, ME@UCB)





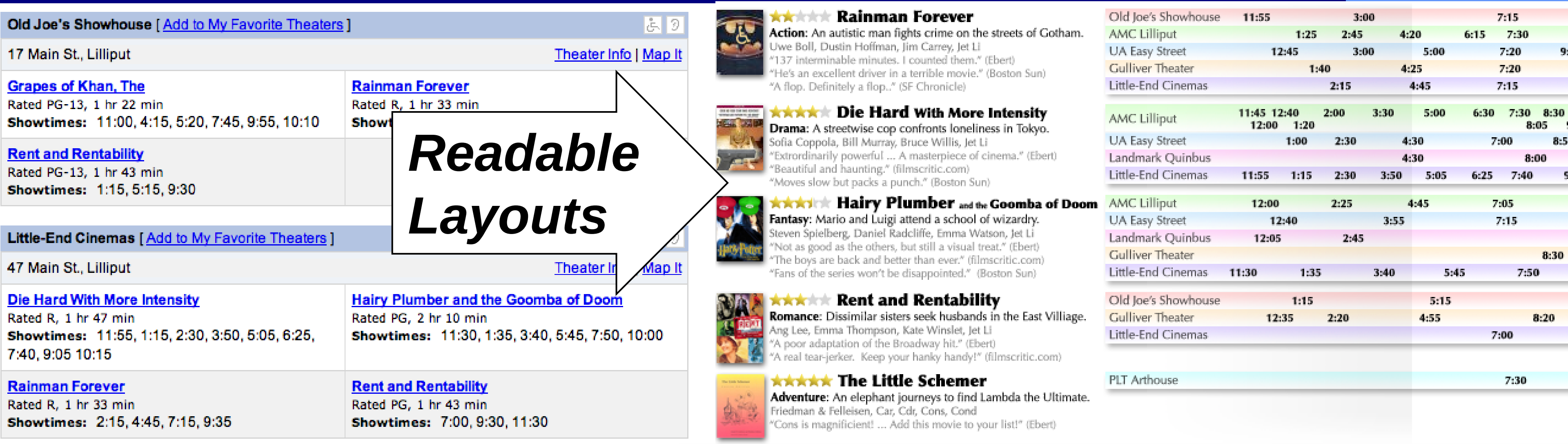
Bottom view of brain

©ADAM, Inc.



- ➢ Stroke treatment time-critical, need supercomputer performance in hospital
- ➢ Goal: 1.5D Fluid-Solid Interaction analysis of Circle of Willis (3D vessel geometry + 1D blood flow).
- ➢ Based on existing codes for distributed clusters

Parallel Browser
(Ras Bodik)



Old Joe's Showhouse [Add to My Favorite Theaters]

17 Main St., Lilliput — Theater Info | Map It

Grapes of Khan, The Rated PG-13, 1 hr 22 min **Showtimes:** 11:00, 4:15, 5:20, 7:45, 9:55, 10:10	Rainman Forever Rated R, 1 hr 33 min **Show**
Rent and Rentability Rated PG-13, 1 hr 43 min **Showtimes:** 1:15, 5:15, 9:30	

Little-End Cinemas [Add to My Favorite Theaters]

47 Main St., Lilliput — Theater In | Map It

Die Hard With More Intensity Rated R, 1 hr 47 min **Showtimes:** 11:55, 1:15, 2:30, 3:50, 5:05, 6:25, 7:40, 9:05 10:15	Hairy Plumber and the Goomba of Doom Rated PG, 2 hr 10 min **Showtimes:** 11:30, 1:35, 3:40, 5:45, 7:50, 10:00
Rainman Forever Rated R, 1 hr 33 min **Showtimes:** 2:15, 4:45, 7:15, 9:35	Rent and Rentability Rated PG, 1 hr 43 min **Showtimes:** 7:00, 9:30, 11:30

Readable Layouts

★★☆☆☆ Rainman Forever
Action: An autistic man fights crime on the streets of Gotham.
Uwe Boll, Dustin Hoffman, Jim Carrey, Jet Li
"137 interminable minutes. I counted them." (Ebert)
"He's an excellent driver in a terrible movie." (Boston Sun)
"A flop. Definitely a flop.." (SF Chronicle)

Old Joe's Showhouse 11:55 3:00 7:15
AMC Lilliput 1:25 2:45 4:20 6:15 7:30
UA Easy Street 12:45 3:00 5:00 7:20 9:
Gulliver Theater 1:40 4:25 7:20
Little-End Cinemas 2:15 4:45 7:15

★★★★☆ Die Hard With More Intensity
Drama: A streetwise cop confronts loneliness in Tokyo.
Sofia Coppola, Bill Murray, Bruce Willis, Jet Li
"Extrordinarily powerful ... A masterpiece of cinema." (Ebert)
"Beautiful and haunting." (filmscritic.com)
"Moves slow but packs a punch." (Boston Sun)

AMC Lilliput 11:45 12:40 12:00 1:20 2:00 3:30 5:00 6:30 7:30 8:30 8:05
UA Easy Street 1:00 2:30 4:30 7:00 8:5
Landmark Quinbus 4:30 8:00
Little-End Cinemas 11:55 1:15 2:30 3:50 5:05 6:25 7:40 9

★★★½☆ Hairy Plumber and the Goomba of Doom
Fantasy: Mario and Luigi attend a school of wizardry.
Steven Spielberg, Daniel Radcliffe, Emma Watson, Jet Li
"Not as good as the others, but still a visual treat." (Ebert)
"The boys are back and better than ever." (filmscritic.com)
"Fans of the series won't be disappointed." (Boston Sun)

AMC Lilliput 12:00 2:25 4:45 7:05
UA Easy Street 12:40 3:55 7:15
Landmark Quinbus 12:05 2:45
Gulliver Theater 8:30
Little-End Cinemas 11:30 1:35 3:40 5:45 7:50

★★★☆☆ Rent and Rentability
Romance: Dissimilar sisters seek husbands in the East Villiage.
Ang Lee, Emma Thompson, Kate Winslet, Jet Li
"A poor adaptation of the Broadway hit." (Ebert)
"A real tear-jerker. Keep your hanky handy!" (filmscritic.com)

Old Joe's Showhouse 1:15 5:15
Gulliver Theater 12:35 2:20 4:55 8:20
Little-End Cinemas 7:00

★★★★★ The Little Schemer
Adventure: An elephant journeys to find Lambda the Ultimate.
Friedman & Felleisen, Car, Cdr, Cons, Cond
"Cons is magnificient! ... Add this movie to your list!" (Ebert)

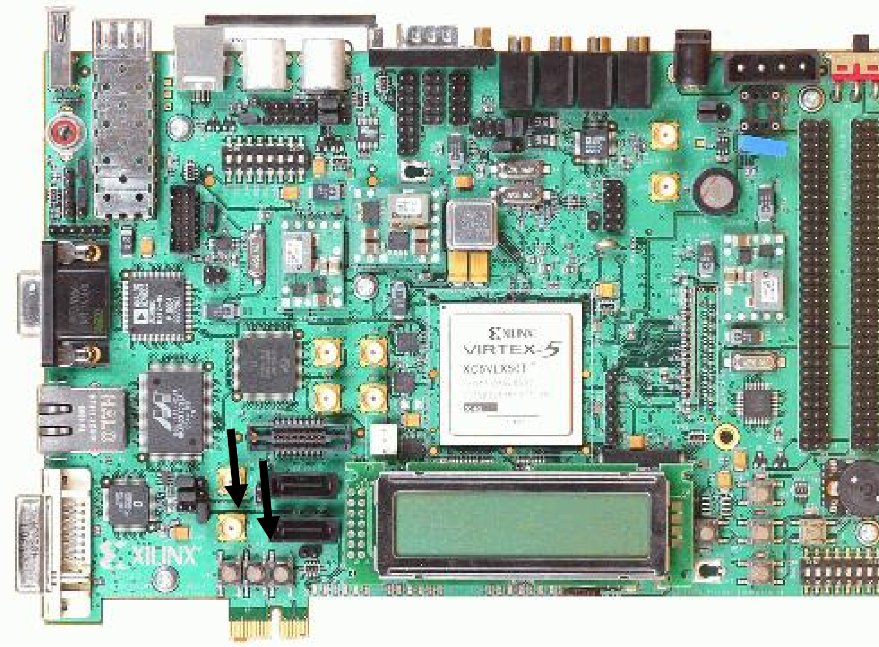
PLT Arthouse 7:30

- Original goal: Desktop-quality browsing on handhelds (Enabled by 4G networks, better output devices)
- Now: Better development environment for new mobile-client applications, merging characteristics of browsers and frameworks (Silverlight, Qt, Android)

Augmented Reality

RAMP Gold (Asanović, Patterson)

- 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 our OS and Par Lab stack!



	Cost	Performance (MIPS)	Time per 64 core simulation
Software Simulator	$2,000	0.1 - 1	250 hours
RAMP Gold	$2,000 + $750	50 - 100	1 hour

Heterogeneity from Manufacturing and Wear

- Heterogeneity from process variations at manufacturing and subsequent wearout
  - Replicating same core design, results in different energy and performance characteristics (max frequency, energy/op @Vdd/Vt setting) (spatial process heterogeneity)
  - One core will drift (usually get worse) over time as part wears out (temporal process heterogeneity)

- Heterogeneity is the *problem* here, not a *solution*
- *(Par Lab is NOT going to work on this)*

Computer Science & Apps

- Career so far: Done 9 (overlapping) 5-year projects

  X-tree
  Reduced Instruction Set Computer (RISC)
  Smalltalk on a RISC (SOAR)
  Symbolic Processing Using RISCs (SPUR)
  Redundant Array of Inexpensive Disks (RAID)
  Network of Workstations (NOW)
  Intelligent RAM (IRAM)
  Recovery Oriented Computing (ROC)
  Reliable Adaptive Distributed systems (RAD Lab)

- 10th project (Par Lab) is 1st project with real apps people
  - Its been great – ask what problem is vs. pretend to know
  - So new Algorithms Machines People (AMP) Lab does too
- Why? 1st 50 years of CS Research solve our own problems? Now CS is ready to help others?

Big Data and Pasteur's Quadrant

Research is inspired by:

Consideration of use?

		No	Yes
Quest for Fundamental Understanding?	Yes	Pure Basic Research (Bohr)	**Use-inspired Basic Research (Pasteur)** **Attack CS Research by Helping Real App?**
	No		Pure Applied Research (Edison)

Adapted from *Pasteur's Quadrant: Basic Science and Technological Innovation*, Donald E. Stokes 1997
(This slide from "Engineering Education and the Challenges of the 21st Century," Charles Vest, 9/22/09)