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
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

Performance/Energy
Efficiency relative to GPP

Application coverage

1000
100
10
1

1 10 100 1000

Fixed-function

General Purpose Proc.

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

You are here

Heterogeneity?
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 of 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
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”)!

<table>
<thead>
<tr>
<th>Domain-Level (No formal CS)</th>
<th>Example Languages</th>
<th>Example Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max/MSP, SQL, CSS/Flash/Silverlight, Matlab, Excel</td>
<td>Builds app with DSL and/or by customizing app frameworks (or apps)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Productivity-Level (Some CS course)</th>
<th>Example Languages</th>
<th>Example Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Python/Ruby/Java, Scala</td>
<td>Uses programming frameworks, writes application frameworks (or apps)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Efficiency-Level (MS in CS)</th>
<th>Example Languages</th>
<th>Example Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C/C+++/FORTRAN assembler</td>
<td>Provides hardware primitives and OS services</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hardware/ OS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

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
How do compelling apps relate to 12 motifs?

<table>
<thead>
<tr>
<th>Motif</th>
<th>Health</th>
<th>Image</th>
<th>Speech</th>
<th>Music</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite State Mach.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Circuits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dense Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparse Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral (FFT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Prog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Particle Methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backtrack/ B&amp;B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphical Models</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unstructured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
“Our” Pattern Language (OPL-2010)
(Kurt Keutzer, Tim Mattson)

Structural Patterns
- Pipe-and-Filter
- Agent-and-Repository
- Task-Parallelism
- Divide and Conquer

Computational Patterns
- Data Parallelism
- Task-Queue
- Distributed-Array

Refine Towards Implementation

Concurrent Algorithm Strategies
- Task-Parallelism
- Divide and Conquer

Implementation Strategy Patterns
- SPMD
- Data-Par/index-space
- Thread-Pool

Parallel Execution Patterns
- MIMD
- SPMD

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

Only a few types of hardware platform

- Multicore
- GPU
- “Cloud”
High-level pattern constrains space of reasonable low-level mappings

**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”*

- App 1
  - Dense
  - Multicore
- App 2
  - Sparse
  - GPU
- App 3
  - Graph Trav.
  - “Cloud”

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 a needle in a 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)

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

2\textsuperscript{nd} level: Application schedules component tasks onto available “harts” (hardware thread contexts) using Lithe
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
Continuously Minimize (subject to restrictions on the total amount of resources)

Penalty Function
Reflects the app’s importance

Resource Utility Function
Performance as function of resources

Convex Surface

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

- \( P_a(L_a) \)
- \( P_b(L_b) \)

\( L_a = RU_a(r(0,a), r(1,a), \ldots, r(n-1,a)) \)

\( L_b = RU_b(r(0,b), r(1,b), \ldots, r(n-1,b)) \)

(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)
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

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

**Berkeley Bet:** Pattern-specific high-level programs can be automatically and dynamically specialized to pattern-specific hardware
What are the compelling future workloads?
- Need apps of future vs. legacy to drive agenda
- 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
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)

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

Won ACM Multimedia Grand Challenge 2009
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!

Speaker Diarization in Python

Python: 45 LOC

```python
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)
        likelihoods = self.gmm_list[0].score(self.X)
        for g in self.gmm_list[1:]:
            likelihoods = np.concatenate((likelihoods, g.score(self.X)))
        most_likely = likelihoods.argmax(axis=1)

        # Across 2.5 secs of observations, vote on which
        split_events = split_events_based_on_voters(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(self.gmm_list)):
            for gmm2idx in range(gmm1idx+1, len(self.gmm_list)):
                g1, d1 = self.gmm_list[gmm1idx]
                g2, d2 = self.gmm_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
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
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

<table>
<thead>
<tr>
<th></th>
<th>Area (mm²)</th>
<th>Frames/Second</th>
<th>Joules/Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium-4 (720x480)</td>
<td>122</td>
<td>30</td>
<td>0.742</td>
</tr>
<tr>
<td>Pentium-4 (1280x720)</td>
<td>122</td>
<td>11</td>
<td>2.023</td>
</tr>
<tr>
<td>ASIC (1280x720)</td>
<td>8</td>
<td>30</td>
<td>0.004</td>
</tr>
</tbody>
</table>

- 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?
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*
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)
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
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”)

<table>
<thead>
<tr>
<th>Domain-Level (No formal CS)</th>
<th>Example Languages</th>
<th>Example Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max/MSP, SQL, CSS/Flash/Silverlight, Matlab, Excel</td>
<td>Builds app with DSL and/or by customizing app framework</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Productivity-Level (Some CS courses)</th>
<th>Example Languages</th>
<th>Example Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Python/Ruby/Lua</td>
<td>Uses programming frameworks, writes application frameworks (or apps)</td>
</tr>
<tr>
<td></td>
<td>Haskell/OCaml/F#</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scala</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Efficiency-Level (MS in CS)</th>
<th>Example Languages</th>
<th>Example Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C/C++/FORTRAN assembler</td>
<td>Uses hardware/OS primitives, builds programming frameworks (or apps)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hardware/ OS</th>
<th>Provides hardware primitives and OS services</th>
</tr>
</thead>
</table>
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")
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
- 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,
Easy to write correct programs that run efficiently on manycore
Transition to Multicore

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


Goal: To enable most programmers to be productive writing efficient, correct, portable SW for 100+ cores & scale as cores increase every 2 years (!)
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
New user interfaces with pressure-sensitive multi-touch gestural interfaces

Programmable virtual instrument and audio processing

120-channel speaker array
Music Software Structure

- GUI Service
- Front-end
- File Service
- Solid State Drive
- Oscillator Bank Plug-in
- Filter Plug-in
- Audio Processing & Synthesis Engine
- Network Service
- 120-Channel Spherical Speaker Array
- Pressure-sensitive multitouch array
- End-to-end Deadline
- Input → Audio Processing → Output

EECS
Electrical Engineering and Computer Sciences

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

- 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)

Readble Layouts

- 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)
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!

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th>Performance (MIPS)</th>
<th>Time per 64 core simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Simulator</td>
<td>$2,000</td>
<td>0.1 - 1</td>
<td>250 hours</td>
</tr>
<tr>
<td>RAMP Gold</td>
<td>$2,000 + $750</td>
<td>50 - 100</td>
<td>1 hour</td>
</tr>
</tbody>
</table>
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)*
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
- It's 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?
<table>
<thead>
<tr>
<th>Quest for Fundamental Understanding?</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consideration of use?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Pure Basic Research (Bohr)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use-inspired Basic Research (Pasteur)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attack CS Research by Helping Real App?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pure Applied Research (Edison)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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)