Par Lab: Where we ended up

Krste Asanovic, Ras Bodik, 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 End-of-Project Party
May 30, 2013
Par Lab Timeline

- Initial Meetings
- "Berkeley View" Techreport
- Win UPCRC Competition
- UPCRC Phase-I
- UPCRC Phase-II
- You are here!
Easy to write correct programs that run efficiently on manycore
Dominant Application Platforms

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

Controversial in 2005, "Obvious" in 2013
Original predictions, 2x cores every 2 years
- 256 cores by 2013

Reality was <2+ cores every 2 years
- 8-16 cores in 2013

But real growth was in SIMD performance
- Wider, more capable SIMD units on multicore
- GP-GPUs

Recent GPUs have up to 2,048 vector lanes!
Many of the parallel patterns are amenable to data-parallel execution

Despite limited memory capacity and cumbersome programming model, GPUs were surprisingly effective on wide range of apps

- Easier to get higher speedups than multicore
- Apps developers voted with their feet

Prediction: Better CPU SIMD extensions and integrated GPUs having to use same memory system will blur/narrow CPU/GPU difference
We architected our bare minimum requirements for accurate performance/energy counters

Bad news: In 2013, commercial processors still don’t meet our bare minimum

Good news: Energy counters have started appearing

Prediction: Counters should be given higher priority but will continue to be “unloved” parts of future architectures
RAMP Gold

- 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>

Download at: [https://sites.google.com/site/rampgold/](https://sites.google.com/site/rampgold/)
RAMP Gold design forms core of DIABLO “Datacenter-In-A-Box at LOW cost”

- Execution-driven model of entire 2,000-node datacenter including network switches

Now, generate FPGA emulations (FAME-0) of own RISC-V processors from Chisel code

Future, developing techniques for automatic generation of efficient FPGA models from RTL

- Chisel automatically generating higher FAME
- New DREAMER emulation architecture
RISC-V ISA

- A new clean-slate open-source ISA to support research and education
- Ports of Tessellation, Akaros, Linux OS, gcc, LLVM,..
- Multiple implementations including “Rocket” in-order research core, plus “Sodor” family of educational processors
- New vector-thread architecture designs
- FPGA emulations + tapeouts of real chips
- To be released soon at: http://www.riscv.org
Embed a hardware-description language in Scala, using Scala’s extension facilities

A hardware module is just a data structure in Scala

Different output routines can generate different types of output (C, FPGA-Verilog, ASIC-Verilog) from same hardware representation

Full power of Scala for writing hardware generators
  - Object-Oriented: Factory objects, traits, overloading etc
  - Functional: Higher-order funcs, anonymous funcs, currying
  - Compiles to JVM: Good performance, Java interoperability

Download from http://chisel.eecs.berkeley.edu
"Agile Hardware" Development

Multiple 28nm and 45nm GHz-class processor tapeouts
Pursued our original approach of very thin hypervisor layer managing partitions

Many ideas swirling in early days of project, concrete implementations on real x86 hardware and RAMP Gold helped provide focus

OS group split into cloud team that moved to AMPLab (Akaros) and client team in Par Lab (Tessellation)
Tessellation OS: Space-Time Partitioning + 2-Level Scheduling

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
Each process receives a vector of basic resources dedicated to it:
- e.g., fractions of cores, cache slices, memory pages, bandwidth

Allocate minimum for QoS requirements

Allocate remaining to meet some system-level objective:
- e.g., 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
“Harts”: Hardw are Threads
A Better Resource Abstraction

- Merged resource and computation abstraction.

- More accurate resource abstraction.
- Let apps provide own computation abstractions
Lithe: “Liquid Thread Environment”

- Lithe is an ABI to allow application components to co-operatively share hardware threads.
- Each component is free to map computational to hardware threads in any way they see fit
  - No mandatory thread or task abstractions
- Components request but cannot demand harts, and must yield harts when blocked or finished with task
### 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>Efficiency-Level (MS in CS)</td>
<td>C/C++/FORTRAN assembler</td>
<td>Provides hardware primitives and OS services</td>
</tr>
<tr>
<td>Productivity-Level (Some CS)</td>
<td>Python/Ruby/Lua Scala</td>
<td>Uses programming frameworks (or apps)</td>
</tr>
<tr>
<td>Hardware/OS</td>
<td>Java/C#</td>
<td>Uses hardware/OS primitives, builds programming frameworks (or apps)</td>
</tr>
<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>

**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 bet 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
Computational Patterns
Common Across Applications

App 1  App 2  App 3
Dense  Sparse  Graph Trav.

Berkeley View
“Dwarfs”
How do compelling apps relate to 13 dwarfs?

<table>
<thead>
<tr>
<th>Apps</th>
<th>Computation</th>
<th>Embed</th>
<th>SPEC</th>
<th>DB</th>
<th>Games</th>
<th>ML</th>
<th>HPC</th>
<th>CAD</th>
<th>Health</th>
<th>Image</th>
<th>Speech</th>
<th>Music</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Algorithms</td>
<td></td>
<td>Red</td>
<td>Red</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Blue</td>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphical Models</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Yellow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backtrack / B&amp;B</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Yellow</td>
<td>Red</td>
<td></td>
<td>Blue</td>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finite State Mach.</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Yellow</td>
<td>Red</td>
<td></td>
<td>Blue</td>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Circuits</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Yellow</td>
<td>Red</td>
<td></td>
<td>Blue</td>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Prog.</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Yellow</td>
<td>Red</td>
<td></td>
<td>Blue</td>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unstructured Grid</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Yellow</td>
<td>Red</td>
<td></td>
<td>Blue</td>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structured Grid</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Yellow</td>
<td>Red</td>
<td></td>
<td>Blue</td>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dense Matrix</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Yellow</td>
<td>Red</td>
<td></td>
<td>Blue</td>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparse Matrix</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Yellow</td>
<td>Red</td>
<td></td>
<td>Blue</td>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral (FFT)</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Yellow</td>
<td>Red</td>
<td></td>
<td>Blue</td>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monte Carlo</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Yellow</td>
<td>Red</td>
<td></td>
<td>Blue</td>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N-Body</td>
<td></td>
<td></td>
<td>Blue</td>
<td>Yellow</td>
<td>Red</td>
<td></td>
<td>Blue</td>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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”*

允许通过避免强制的中间层来允许在特殊化中实现最大效率和可表达性。
SEJITS: “Selective, Embedded, Just-In Time Specialization”

- 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 or package autotuners

Two ParLab SEJITS projects:

- **Copperhead**: Data-parallel subset of Python, development continuing at NVIDIA
- **Asp**: “Asp is SEJITS in Python” general specializer framework
  - Provide functionality common across different specializers
SEJITS In A Nutshell

Program

- Non-DSL Code
- Code in DSL A
- Code in DSL B

Compile Phase

- Data
- Code in DSL A
- DSL Codegen

Execute Phase

- External Compiler
- Dynamic Link Library

Result

Data → Interpreter
Supporting QoS inside Apps

Application

Module 1
Specializer
Efficiency
Level Code
TBB Code
Module 1 Scheduler
Lithe

Module 2
Best-Effort Cell

Real-Time Cell
Module 3
Real-Time Scheduler

Tessellation OS

Hardware Resources (Cores, Cache/Local Store, Bandwidth)
Communication-Avoiding Algorithms

- Past algorithms: FLOPs expense, Moves cheap
- New theory: proves lower bounds on data movement; both serial (memory hierarchy) and parallel data movement
- New practice: codes achieve lower bound and speedups
- Widely applicable: all linear algebra, Health app…

Idea #1: read a piece of a sparse matrix (= graph) into fast memory and take multiple steps of higher-level algorithm

Idea #2: replicate data (including left-hand side arrays, as in C in C=A*B) and compute partial results, reduce later
A few examples of speedups

- **Matrix multiplication**
  - Up to 12x on IBM 64K-core BG/P for n=8K; **95% less communication**
- **QR decomposition** (used in least squares, data mining, …)
  - Up to 8x on 8-core dual-socket Intel Clovertown, for 10M x 10
  - Up to 6.7x on 16-proc. Pentium III cluster, for 100K x 200
  - Up to 13x on Tesla C2050 / Fermi, for 110k x 100
  - “infinite speedup” for out-of-core on PowerPC laptop
    - LAPACK thrashed virtual memory, didn’t finish
- **Eigenvalues of band symmetric matrices**
  - Up to 17x on Intel Gainestown, 8 core, vs MKL 10.0
- **Iterative sparse linear equations solvers (GMRES)**
  - Up to 4.3x on Intel Clovertown, 8 core
- **N-body** (direct particle interactions with cutoff distance)
  - Up to 10x on Cray XT-4 (Hopper), 24K particles on 6K procs.

Next: automatically xform code; new “HBL” theory just out!
Autotuning: Computers, Rather than People Tune Code

- Autotuners are code generators plus search
- Avoids two unsolved compiler problems: dependence analysis and accurate performance models
- New: particles, stencils, graphs, … and manylane/core optimizations
- New roofline model to aid in performance understanding

Work by Williams, Oliker, Shalf, Madduri, Kamil, Im, Ethier,…
Some Results

- Active Testing: for finding non-deterministic bugs such as data races, deadlocks, atomicity violations
  - Open-source [BSD-licensed] CalFuzzer for Java
  - Thrille for C/Pthreads and UPC

- Specification and assertion framework for parallelism correctness
  - Introduced Bridge Predicates
  - Nondeterministic Sequential Specification to separate parallel correctness from functional correctness

- Concurrit: A testing framework for concurrent programs
  - JUnit or xUnit for concurrent programs
  - Applied to Chrome and Firefox browsers

- Concolic testing: for automated test input generation
  - Java
  - Javascript (ongoing)
Parallel Browser

- **2007 Vision**: desktop-quality browsing on mobiles.
- **Now**: yes, but only 200 web pages / battery charge.
- **2007 Vision**: browser as an app platform.
- **Now**: Google Glass is a browser app.
Parallel Browser

Parallelism improves responsiveness, energy use.

- 2007: parallel browser controversial
- 2013: Mozilla Servo & Google Blink browsers

Some of our results:

- First scalable parser (in Qualcomm browser)
- Synthesizer of parallel layout engines
- Parallel layout retrofitted to Safari via WebCL
- Collaboration with Mozilla on parallel Servo
Synthesis: search a huge space for a program that is semantically correct and high-performance

- alternative to classical AST-rewrite compilers
- search implemented as constraint solving

Some of our synthesizers:

- FTL: parallel layout engines
- Programming for ULP spatial manycore
- SQL query programming by demonstration
Music Application

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

Programmable virtual instrument and audio processing

120-channel speaker array
More Applications: “Beating down our doors!”

- 5 Original Apps: Parallel Browser (Ras Bodik), Music (David Wessel), Speech (Nelson Morgan), Health (Tony Keavney), Image Retrieval (Kurt Keutzer)

- New external application collaborators:
  - Pediatric MRI (Michael Lustig, Shreyas Vassanwala @Stanford)
  - Multimedia and Speech (Dorothea Kolossa @TU Berlin)
  - Computer Vision (Jitendra Malik, Thomas Brox)
  - Computational Finance (Matthew Dixon @UCD)
  - Natural Language Translation (Dan Klein)
  - Programming multitouch interfaces (Maneesh Agrawala)
  - Protein Docking (Henry Gabb, Intel)
Pediatric MRI

Typical exam ~ 1 hour
Motion blurs the images
Scanner is a small loud tunnel
Difficult for children to stay still!

Traditional Solution: Anesthesia

Compressed Sensing reduces each scan to 15 seconds
But takes too long (hours) to reconstruct image
Compressed Sensing for Pediatric MRI

- Image reconstruction from 1-2 hours down to < 1 min
- In use in clinical trials
Today’s Demos

Applications
- Personal Health
  - Contour Detection, Optical Flow
  - Music Synthesis, PyCASP
  - Meeting Diarizer
  - Parallel Browser

Patterns
- SEJITS

Productivity Layer
- Efficiency Layer
- Legacy Code
  - PULSE
  - Lithe

Efficiency Language Compilers
- Tessellation, PACORA
- RISC-V Vector Processor

OS
- Legacy OS
- Multicore/GPGPU

Architecture
- Correctness
Demo: The Parallel Meeting Diarist

Gerald Friedland
International Computer Science Institute (ICSI)

Work together with primarily:
Katya Gonina
David Sheffield
Adam Janin
Brian Hill
Jike Chong
Nelson Morgan
Kurt Keutzer
Ganapati S.
Mishali N.
Components of the Meeting Diarist

Created a fully integrated, very fast meeting diarist SEJITized that is currently tech-transfered to Intel.
### Diarizer Results and Impacts

<table>
<thead>
<tr>
<th></th>
<th>Before Par Lab</th>
<th>After Par Lab</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;who spoke when&quot;</td>
<td>~10k LOC</td>
<td>~100 LOC</td>
</tr>
<tr>
<td></td>
<td>0.3 x RT</td>
<td>250 x RT</td>
</tr>
<tr>
<td>&quot;what was said&quot;</td>
<td>~100k LOC</td>
<td>~5k LOC</td>
</tr>
<tr>
<td></td>
<td>0.1 x RT</td>
<td>1 x RT</td>
</tr>
</tbody>
</table>

Created a fully integrated, very fast meeting diarist using SEJITs:

- Online=Offline processing for “who spoke when”
- Diarization used for BIG DATA video processing
- Tech-transfer to Intel
Par Lab Publications Tally

- 193 Conference Papers
- 28 Journal Papers
- 94 Technical Reports
- 12+ Best Paper Awards

- Berkeley View TR, >1,300 citations
- Par Lab papers, >8,000 citations
Par Lab Educational Impact

- Par Lab summer bootcamps
  - 2009-2012: >1300 attendees including >300 from industry
- CS61C Reworked intro architecture class with parallelism
  - 480 students in Fall 2012 semester
  - Revised 5th edition of undergrad text (used at 400 universities)
- CS152 Undergrad Architecture using Chisel processors
- CS164 Students design and own DSLs, implement a browser
- CS194 Undergrad parallel patterns class
  - 3rd offering, co-taught with Tim Mattson, Intel
  - 3 posters here from successful undergrad projects
- CS250 Graduate VLSI Design using Chisel/Agile Hardware
- CS267 Graduate parallel computing class
  - Added material on dwarfs, patterns, autotuning, apps
  - Homeworks ported to .net with Microsoft
  - NSF-funded MOOC XSEDE launched spring 2013
Par Lab Students

PhD students
- 91 total PhD participants,
- 23 of which graduated by 2013

MS Students
- 35 total MS participants,
- 13 of which graduated by 2013

Post-Docs
- 5 current
Par Lab Book

- 18 chapters
  - Overview + 1-2 research papers
- ≈ 600 pages
- Expected by June 30
  - Amazon Ebook $0.99
- Print book signup page
Multiple Follow-On Projects Underway

- OS and Music work continuing in new SWARM Lab, programming the “swarm” of environmental devices
  
  [Url: http://swarmlab.eecs.berkeley.edu]

- Software synthesis, Correctness -> Chaperone, ExCape NSF Expedition

- ASPIRE: patterns, communication-avoiding algorithms, SEJITS, RISC-V, specialized architectures, Chisel, Agile Hardware
  
  [Url: http://aspire.eecs.berkeley.edu]
Thanks to our sponsors!

- Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227).
- Additional support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, Samsung, and Oracle/Sun.