The Parallel Computing Laboratory

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

Microsoft Faculty Summit
Bellevue, WA
July 17, 2012
Transition to Multicore
Par Lab Inception: Needed a Fresh Approach to Parallelism

- Berkeley researchers from many backgrounds meeting since Feb. 2005 to discuss parallelism
  - Krste Asanovic, 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
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.*
Co-located Collaborative Center Approach

- 60+ students, 8+ faculty in one shared space
- Faculty in open space, not in offices
- Off-site retreat every 6 months with ~60 outside visitors (industry sponsors, and other invited experts)
Par Lab Timeline

- **Initial Meetings**
- "Berkeley View" Techreport
- Win UPCRC Competition
- UPCRC Phase-I
- UPCRC Phase-II
- End of Project Party!

You are here
Big Ideas from Par Lab

- Patterns for parallel programming
- Communication-avoiding algorithms
- Specializers: Pattern-specific compilers
- Effective composition of parallel modules
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

Content-Based Image Retrieval (Kurt Keutzer)

Application area expanded to many areas of computer machine vision.

- Built around Key Characteristics of personal databases
  - Very large number of pictures (>5K)
  - Non-labeled images
  - Many pictures of few people
  - Complex pictures including people, events, places, and objects
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)

Readable 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)
Browser Development Stack

- HTML
- CSS
- Multicore parser
  - tree
  - style template
- Multicore selector matcher
- Multicore cascade
  - tree decorated with style constraints
- Layout engine
  - Multicore fast tree library
    - layout visitor
    - scene graph
- OpenGL Qt Renderer
- Widget definition
- MUD language
- Grammar specification
- ALE synthesizer
- Incrementalizer
New user interfaces with pressure-sensitive multi-touch gestural interfaces

Programmable virtual instrument and audio processing

120-channel speaker array

Gestures

Audio
Music Software Structure

Pressure-sensitive multitouch array

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

Audio Processing

Output

End-to-end Deadline
• Laptops/ Handhelds at meeting coordinate to create speaker identified, partially transcribed text diary of meeting
Winner ACM Multimedia Grand Challenge 2009
- find best punchlines in Seinfeld episodes

Speedup progress in Par Lab:
2006 0.3x realtime, original code
2008 1.5x realtime, optimized serial code
2010 14.3x realtime, multicore CPU+GPU
2011 250x realtime, pure GPU, from Python code, changed the field!

Interactive GUI
<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</th>
<th>Example Languages</th>
<th>Example Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haskell, Scala, #</td>
<td>Python/Ruby/Lua, Scala, Go</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>C/C++/FORTRAN, assembler</td>
<td>Java/C#</td>
<td>Provides hardware primitives and OS services</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
Motifs common across applications

App 1: Dense
App 2: Sparse
App 3: Graph Trav.

Berkeley View Motifs ("Dwarfs")
How do compelling apps relate to 13 motifs?

<table>
<thead>
<tr>
<th>Motif</th>
<th>Popularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Finite State Mach.</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
<tr>
<td>2 Circuits</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
<tr>
<td>3 Graph Algorithms</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
<tr>
<td>4 Structured Grid</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
<tr>
<td>5 Dense Matrix</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
<tr>
<td>6 Sparse Matrix</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
<tr>
<td>7 Spectral (FFT)</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
<tr>
<td>8 Dynamic Prog</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
<tr>
<td>9 Particle Methods</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
<tr>
<td>10 Backtrack/ B&amp;B</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
<tr>
<td>11 Graphical Models</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
<tr>
<td>12 Unstructured Grid</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
<tr>
<td>13 Monte Carlo</td>
<td>![Chart showing popularity of motifs]</td>
</tr>
</tbody>
</table>
Structural Patterns
Pipe-and-Filter
Event-Based/Implicit Invocation
Puppeteer

Computational Patterns
Graphs
Dense-Linear-Algebra
Sparse-Linear-Algebra
Unstructured-Grids
Structured-Grids
Monte-Carlo

Structural Patterns
Refine Towards Implementation

A = M x V

Concurrent Algorithm Strategies
Task-Parallelism
Divide and Conquer

Implementation Strategy Patterns
SPMD
Data-Par/index-space
Fork/Join
Actors
Task-Queue
Partitioned Graph
Distributed-Array
Program structure

Parallel Execution Patterns
MIMD
SIMD
Thread-Pool
Task-Graph
Transactions

Concurrency Foundation constructs (not expressed as patterns)
Thread creation/destruction
Message-Passing
Point-To-Point-Sync. (mutual exclusion)
Process creation/destruction
Collective-Comm.
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”*

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

1. Arithmetic (FLOPS)

2. Communication: moving data between
   - levels of a memory hierarchy (sequential case)
   - processors over a network (parallel case).
Cost of communication $\gg$ cost of arithmetic
- True for cost = time, or cost = energy per operation
- Cost gap growing over time

Goals
- Identify lower bounds on communication required by widely used algorithms
  - Many widely used libraries (e.g., Scy/LAPACK) communicate asymptotically more than necessary
- Design new algorithms that attain lower bounds
  - Possible for dense and sparse linear algebra, n-body, ...
  - Big speedups and energy savings possible
A few examples of speedups

- **Matrix multiplication**
  - Up to **12x** on IBM BG/P for n=8K on 64K cores; **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
  - Up to **4x** on Grid of 4 cities (Dongarra, Langou et al)
  - “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 (up to **1.9x** sequential)

- **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.
Recent Prizes for CA Work

- SIAM Linear Algebra Prize 2012, for best paper in previous 3 years, deriving lower bounds
- SPAA’11 Best Paper Award, for Strassen lower bounds
- EuroPar’11 Distinguished Paper Award, for asymptotically faster “2.5D” matmul and LU
- Citation in 2012 DOE Budget Request …
“New Algorithm Improves Performance and Accuracy on Extreme-Scale Computing Systems. On modern computer architectures, communication between processors takes longer than the performance of a floating point arithmetic operation by a given processor. ASCR researchers have developed a new method, derived from commonly used linear algebra methods, to minimize communications between processors and the memory hierarchy, by reformulating the communication patterns specified within the algorithm. This method has been implemented in the TRILINOS framework, a highly-regarded suite of software, which provides functionality for researchers around the world to solve large scale, complex multi-physics problems.”


CA-GMRES (Hoemmen, Mohiyuddin, Yelick, Demmel)
“Tall-Skinny” QR (Grigori, Hoemmen, Langou, Demmel)
Graph Algorithms
(Scott Beamer)

- New algorithm for Breadth-First Search
- Highest single-node performance in November 2011, Graph500, using Intel Xeon E7-8870 (Mirasol)

- #15: BlueGene 2048 cores  6.93 GTEPS
- #16: Jaguar  1024 cores  6.26 GTEPS
- #17: Mirasol  40 cores  5.12 GTEPS
- #18: Blacklight  512 cores  4.45 GTEPS
- #19: Todi  176 TESLA GPUs  3.05 GTEPS
- #20: Convey  4 FPGAs  1.76 GTEPS
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 (e.g., images), sparse matrices, particle/mesh, collectives (e.g., “reduce”), ...
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, development continuing at NVIDIA
- **Asp**: “Asp is SEJITS in Python” general specializer framework
  - Provide functionality common across different specializers
SEJITS Overview

Selective

Activity app

.f

.h

JIT

.C

c/c/ld

$.

Specialization

Specializer

Embedded

PLL Interp

OS/HW
Asp: Who Does What?

App author (PLL)
Application
Kernel
Kernel call & Input data
Results

Specializer author (ELL)
Specializer
Domain-Specific Transforms
Target AST

Asp team
Asp core
Python AST
Utilities
Asp Module

3rd party libraries
Compiled libraries

SEJITS version of meeting diarizer, 250x realtime from 50 lines of Python
Composition

- All applications built as a hierarchy of modules, not just one kernel

Structural patterns describe the common forms of composing sub-computations:
  E.g., task graph, pipelines, agent/repository
Effective Parallel Composition

- **Data format/layout**: Must translate between data formats or layouts expected by different components
- **Synchronization**: Must correctly synchronize data passing between or shared by multiple components
- **Resource management**: Must share hardware resources to execute components in parallel
Efficient Parallel Composition of Libraries is Hard

Gaming App Example

Libraries compete unproductively for resources!
"Harts": **Hardware Threads**

A Better Resource Abstraction

- **Merged** resource and computation abstraction.
- More accurate resource abstraction.
- Let apps provide own computation abstractions.

Virtualized Threads

Hardware Partitions

(HW Thread Contexts)
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
**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
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

**Penalty Function**
Reflects the app’s importance

**Resource Utility Function**
Performance as function of resources

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

- Allocate minimum for QoS requirements
- Allocate remaining to meet some system-level objective
  - e.g., best performance, lowest energy, best user experience
Par Lab Stack Summary

- Organize software around parallel patterns
  - Maximize reuse since patterns common across domains
- Each pattern implemented with efficient algorithms packaged as SEJITS specializers using autotuners
- Programmer composes functionality at high-level using productivity language
- System composes resource usage at low-level using 2-level scheduling
  - Tessellation OS at coarse-grain
  - Lithe user-level scheduler ABI at fine-grain
Future Architectures?

- What about GPUs versus CPUs?
- These architectures are closely related, and converging.
- Both have multiple multithreaded cores each with many SIMD lanes
  - original vision was “manycore” – more accurate to say future is “manylane”.
- Most of our techniques can be applied to both
Focus on supporting application and OS needs:

- Hardware partitioning support
- Performance counters
- High-performance FPGA-based simulators

New architecture ideas:

- New data-parallel execution engines
- Hardware+software managed memory hierarchy
- Specialized accelerators (e.g., graph machines)

Extensive development of VLSI flow to allow real layout of various data-parallel accelerators

- Accurate cycle time, area, energy
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>
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.
Questions?