UC Berkeley Par Lab Overview

David Patterson
Par Lab’s original research “bets”

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

*Above all, no preconceived big idea – see what works driven by application needs*
Easy to write correct programs that run efficiently on manycore

Par Lab Research Overview

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 and 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
Dominant Application Platforms

- Data Center or Cloud ("Server")
- Laptop/Handheld ("Mobile Client")
- Both together ("Server+Client")
  - New ParLab-RADLab collaborations
- Par Lab focuses on mobile clients
  - But many technologies apply to data center
Music and Hearing Application
(David Wessel)

- Musicians have an insatiable appetite for computation + real-time demands
  - More channels, instruments, more processing, more interaction!
  - Latency must be low (5 ms)
  - Must be reliable (No clicks!)

1. Music Enhancer
   - Enhanced sound delivery systems for home sound systems using large microphone and speaker arrays
   - Laptop/Handheld recreate 3D sound over ear buds

2. Hearing Augmenter
   - Handheld as accelerator for hearing aid

3. Novel Instrument User Interface
   - New composition and performance systems beyond keyboards
   - Input device for Laptop/Handheld

Berkeley Center for New Music and Audio Technology (CNMAT) created a compact loudspeaker array: 10-inch-diameter icosahedron incorporating 120 tweeters.
Health Application: Stroke Treatment  
(Tony Keaveny)

- Stroke treatment time-critical, need supercomputer performance in hospital
- Goal: First true 3D Fluid-Solid Interaction analysis of Circle of Willis
- Based on existing codes for distributed clusters
Content-Based Image Retrieval
(Kurt Keutzer)

- Query by example
- 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

Image Database

1000's of images

Relevance Feedback

Similarity Metric

Candidate Results

Final Result
Meeting Diarist

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

- Use cortically-inspired many-stream spatio-temporal features to tolerate noise
Parallel Browser (Ras Bodik)

Goal: Desktop quality browsing on handhelds
- Enabled by 4G networks, better output devices

Bottlenecks to parallelize
- Parsing, Rendering, Scripting

Slashdot (CSS Selectors)
Compelling Apps in a Few Years

- **Name Whisperer**
  - Built from Content Based Image Retrieval
  - Like Presidential Aid

- Handheld scans face of approaching person

- Matches image database

- Whispers name in ear, along with how you know him
Our initial survey of many applications brought out common recurring patterns:

“Dwarfs” -> Motifs

- Computational patterns
- Structural patterns

Insight: Successful codes have a comprehensible software architecture:
- Patterns give human language in which to describe architecture
• How do compelling apps relate to 12 motifs?

<table>
<thead>
<tr>
<th>Motif Name</th>
<th>Embed</th>
<th>SPEC</th>
<th>DB</th>
<th>Games</th>
<th>ML</th>
<th>CAD</th>
<th>HPC</th>
<th>Health</th>
<th>Image</th>
<th>Speech</th>
<th>Music</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Finite State Mach.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2  Circuits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3  Graph Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4  Structured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5  Dense Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6  Sparse Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7  Spectral (FFT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8  Dynamic Prog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9  Particle Methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 Backtrack/ B&amp;B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 Graphical Models</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 Unstructured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Architecting Parallel Software

Decompose Tasks/Data
Order tasks
Identify Data Sharing and Access

Identify the Software Structure

• Pipe-and-Filter
• Agent-and-Repository
• Event-based
• Bulk Synchronous
• MapReduce
• Layered Systems
• Arbitrary Task Graphs

Identify the Key Computations

• Graph Algorithms
• Dynamic programming
• Dense/Spare Linear Algebra
• (Un)Structured Grids
• Graphical Models
• Finite State Machines
• Backtrack Branch-and-Bound
• N-Body Methods
• Circuits
• Spectral Methods
### People, Patterns, and Frameworks

<table>
<thead>
<tr>
<th>People</th>
<th>Patterns</th>
<th>Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Developer</td>
<td>Uses application design patterns (e.g. feature extraction) to architect the application</td>
<td>Uses application frameworks (e.g. CBIR) to implement the application</td>
</tr>
<tr>
<td>Application-Framework Developer</td>
<td>Uses programming design patterns (e.g. Map/Reduce) to architect the application framework</td>
<td>Uses programming frameworks (e.g. MapReduce) to implement the application framework</td>
</tr>
</tbody>
</table>
The hope is for Domain Experts to create parallel code with little or no understanding of parallel programming. Leave hardcore “bare metal” efficiency-layer programming to the parallel programming experts.
Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore

- 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 and Services

Hypervisor

Multicore/GPGPU

ParLab Manycore/RAMP

Correctness

Static Verification

Type Systems

Directed Testing

Dynamic Checking

Debugging with Replay

Applications

Productivity Layer

Efficiency Layer

OS Arch.
Par Lab is Multi-Lingual

Applications require ability to compose parallel code written in many languages and several different parallel programming models
- Let application writer choose language/model best suited to task
- High-level productivity code and low-level efficiency code
- Old legacy code plus shiny new code

Correctness through all means possible
- Static verification, annotations, directed testing, dynamic checking
- Framework-specific constraints on non-determinism
- Programmer-specified semantic determinism
- Require common spec between languages for static checker

Common linking format at low level (Lithe) not intermediate compiler form
- Support hand-tuned code and future languages & parallel models
Why Consider New Languages?

Most of work is in runtime and libraries

Do we need a language? And a compiler?

- If higher level syntax is needed for productivity
  - We need a language
- If static analysis is needed to help with correctness
  - We need a compiler (front-end)
- If static optimizations are needed to get performance
  - We need a compiler (back-end)

Will prototype frameworks in conventional languages, but investigate how new languages or pattern-specific compilers can improve productivity, efficiency, and/or correctness
Selective Embedded Just-In-Time Specialization (SEJITS) for Productivity

Modern scripting languages (e.g., Python and Ruby) have powerful language features and are easy to use.

Idea: Dynamically generate source code in C within the context of a Python or Ruby interpreter, allowing app to be written using Python or Ruby abstractions but automatically generating, compiling C at runtime.

Like a JIT but:

- **Selective**: Targets a particular method and a particular language/platform (C+OpenMP on multicore or CUDA on GPU)
- **Embedded**: Make specialization machinery productive by implementing in Python or Ruby itself by exploiting key features: introspection, runtime dynamic linking, and foreign function interfaces with language-neutral data representation.
## Selective Embedded Just-In-Time Specialization for Productivity

**Case Study: Stencil Kernels on AMD Barcelona, 8 threads**
- Hand-coded in C/OpenMP: 2-4 days
- SEJITS in Ruby: 1-2 hours

**Time to run 3 stencil codes:**

<table>
<thead>
<tr>
<th>Hand-coded (seconds)</th>
<th>SEJITS from cache (seconds)</th>
<th>Extra JIT-time 1st time executed (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.74</td>
<td>0.74</td>
<td>0.25</td>
</tr>
<tr>
<td>0.72</td>
<td>0.70</td>
<td>0.27</td>
</tr>
<tr>
<td>1.26</td>
<td>1.26</td>
<td>0.27</td>
</tr>
</tbody>
</table>
Autotuning for Code Generation (Demmel, Yelick)

- Problem: generating optimal code like searching for needle in haystack
- Manycore is even more diverse
- New approach: “Auto-tuners”
  - 1st generate program variations of combinations of optimizations (blocking, prefetching, …) and data structures
  - Then compile and run to heuristically search for best code for that computer
- Examples: PHiPAC (BLAS), Atlas (BLAS), Spiral (DSP), FFT-W (FFT)
Tessellation OS allocates hardware resources (e.g., cores) at coarse-grain, and user software shares hardware threads co-operatively using Lithe ABI.

Lithe provides performance composability for multiple concurrent and nested parallel libraries.

Already supports linking of parallel OpenMP code with parallel TBB code, without changing legacy OpenMP/TBB code and without measurable overhead.
Tessellation: Space-Time Partitioning for Manycore Client OS

Wireless radio

QoS Allocations

De-scheduled Partitions

Memory

Network Driver

Video decoder

GUI

Browser

Windows VM

Filesystem

Media Player
Tessellation Kernel Structure

Application

Or

OS Service

Library OS

Or

Functionality

Custom

Scheduler

Partition

Management

Layer

Partition

Scheduler

Partition

Scheduler

Partition

Allocator

Partition

Resizing

Callback API

Partition

Resizing

Callback API

Hardware Partitioning Mechanisms

Interconnect Bandwidth

Message Passing

Cache

Physical Memory

CPUs

Performance Counters

Tessellation Kernel

Partition

Mechanism

Layer (Trusted)

Configure

HW-supported

Communication

Configure

HW-supported

Communication

Kernel being written in Ivy

a safe C dialect

Component

managed by

Runtime

Component

managed by

Runtime

Sched

Reqs.

Comm.

Reqs

Res.

Reqs.

Res.

Reqs.
Par Lab Architecture

Architect a long-lived horizontal software platform for independent software vendors (ISVs)

- ISVs won’t rewrite code for each chip or system
- Customer buys application from ISV 8 years from now, wants to run on machine bought 13 years from now (and see improvements)

...instead, one type of multi-paradigm core

Not multiple paradigms of core

- Fat Cores (InstLP)
- Thin Cores (ThreadLP)
- Weird Cores (DataLP)
- Weirder Cores (GateLP)

Not multiple paradigms of core
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

<table>
<thead>
<tr>
<th></th>
<th>Software Simulator</th>
<th>RAMP Gold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>$2,000</td>
<td>$2,000 + $750</td>
</tr>
<tr>
<td>Performance (MIPS)</td>
<td>0.1 - 1</td>
<td>50 - 100</td>
</tr>
<tr>
<td>Simulations per day</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>
Par Lab’s original research “bets”

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

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

To learn more: http://parlab.eecs.berekeley.edu
Par Lab Research Overview

Easy to write correct programs that run efficiently on manycore

- **Applications**
  - 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**