Towards Provably Optimal Parallel Systems

Krste Asanovic
UC Berkeley

MATEO-2012, Barcelona
June 29, 2012
Background

- Berkeley Par Lab ending (Wrap May 23, 2013).
- Talk today is early look at next project, ASPIRE
- Faculty participants:
  - Elad Alon
  - Krste Asanovic (PI)
  - Jim Demmel
  - Armando Fox
  - Kurt Keutzer
  - Borivoje Nikolic
  - David Patterson
  - Koushik Sen
  - John Wawrzynek

Circuits → Architecture → Algorithms → Programming Systems → Applications/Patterns → Circuits → Architecture → Programming Systems → Reconfigurable
Efficiency Matters

- Even if <<1% of programmers understand it

- Humanity’s capabilities limited by biggest machines running most efficient code.

- Natural user interfaces are performance limited.

- More efficient primitives support more sloppy code above (e.g., Lua game scripting, or mashups built on top of warehouse-scale computer services)

- Efficient systems => Parallel systems
Par Lab Software Stack Highlights

- Organize software around parallel patterns
  - Maximize reuse since patterns common across domains
- Communication-Avoiding Algorithms for patterns
- Implement each pattern with highly efficient specializers using SEJITS-based autotuners
- Programmer composes functionality at high-level using productivity language
- System composes resource usage using 2-level scheduling: 1) Tessellation OS at coarse-grain and 2) Lithe user-level scheduler at fine-grain
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?
“Our” Pattern Language (OPL-2010)  
(Kurt Keutzer, Tim Mattson)

Structural Patterns
Pipe-and-Filter  
Event-Based/Implicit Invocation  
Puppeteer

Computational Patterns
Graphs  
Finite-State-Machines  
Backtrack-Branch-and-Bound  
N-Body-Methods  
Circuits  
Spectral-Methods  
Monte-Carlo

A = M x V

Refine Towards Implementation

Concurrent Algorithm Strategy Patterns
Task-Parallelism  
Divide and Conquer

Implementation Strategy Patterns
SPMD  
Data-Par/index-space  
Fork/Join  
Actors  
Task-Queue  
Partitioned Graph  
Distributed-Array  
Shared-Data  
Thread-Pool

Parallel Execution Patterns
MIMD  
SIMD  
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

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.

Asp infrastructure supports specializer authors.
The Next Challenge

- Technology scaling slowing down/stopping
- No savior device technology on horizon
- Parallelism was one-time gain, using more, lower-performance cores for better energy efficiency
  - Simpler general-purpose microarchitectures
    - Limited by smallest sensible core
  - Lower Vdd/Frequency
    - Limited by Vdd/Vt scaling, errors
- Now what? Only option appears to be more specialized hardware running more optimized software.
What is the best we can do?
For a fixed target technology (e.g., 7nm):
Can we prove a bound?
Can we design an implementation to approach that bound?
  => Provably Optimal Implementations!

*ASPIRE: Algorithms and Specializers for Provably Optimal Implementations with Resiliency and Efficiency
Algorithm Costs

1. Arithmetic (FLOPS)
2. Communication: moving data between
   - levels of a memory hierarchy (sequential case)
   - processors over a network (parallel case).
Communication-Avoiding Algorithms
(Jim Demmel & BEBOP Group)

- Communication = moving data, between levels of memory or between processors over a network
- Cost of communication >> 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 (eg Sca/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
Ex: Lower bound for all “direct” linear algebra

• Let $M$ = “fast” memory size (per processor)

$$\# \text{words\_moved (per processor)} = \Omega(\# \text{flops (per processor)} / M^{1/2})$$

$$\# \text{messages\_sent} \geq \# \text{words\_moved} / \text{largest\_message\_size}$$

❖ Holds for
  - Matmul, BLAS, LU, QR, eig, SVD, tensor contractions, …
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, e.g. $A^k$)
  - Dense and sparse matrices (where $\# \text{flops} \ll n^3$)
  - Sequential and parallel algorithms
  - Some graph-theoretic algorithms (e.g. Floyd-Warshall)
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)
From C-A Algorithms to Provably Optimal Systems?

- 1) Prove lower bounds on communication for a computation
- 2) Develop algorithm that achieves lower bound on a system
- 3) Find that communication time/energy cost is >90% of resulting implementation
- 4) We know we’re within 10% of optimal

Supporting technique: Optimize cores so that they get sufficiently low energy to ignore
ESP Architecture: Ensembles of Specialized Processors

- General-purpose hardware, flexible but inefficient
- Fixed-function hardware, efficient but inflexible
- Insight: Patterns capture common operations across many applications, each with unique computation/communication structure
- Build an ensemble of specialized engines, each individually optimized for particular pattern but collectively covering application needs
- Bet: Will give us efficiency plus flexibility
  - Any given core can have a different mix of these depending on workload
Optimize compute and data movement per pattern

- Dense Engine: Provide sub-matrix load/store operations, support in-register reuse
- Structured Grid Engine: Supports in-register operand reuse across neighborhood
- Sparse Engine: Support load/store of various sparse data structures
- Graph Engine: Provide load/store of bitmap vertex representations, support many outstanding requests
ASPIRE Initial Stack Bet

Applications/Domains

- Object Recognition
- Scene Analysis

Computational and Structural Patterns

- Dense
  - C-A GEMM
- Sparse
  - C-A SpMV
- Graph
  - C-A BFS

Software

Specializers with SEJITS Implementations and Autotuning

- Glue Code
- Dense Code
- Sparse Code
- Graph Code
- ESP Code

Hardware

- ILP Engine
- Dense Engine
- Sparse Engine
- Graph Engine
- ESP Core

Hardware Cache Coherence

- Local Stores + DMA

Deep HW/SW Design-Space Exploration

Hardware Generators using Chisel HDL

- C++ Simulation
- FPGA Emulation
- FPGA Computer
- ASIC SoC

Validation/Verification

Implementation Technologies

ESP (Ensembles of Specialized Processors) Architecture

Communication-Avoiding Algorithms

Pipe&Filter

Map-Reduce
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.