

Towards Provably Optimal Parallel Systems

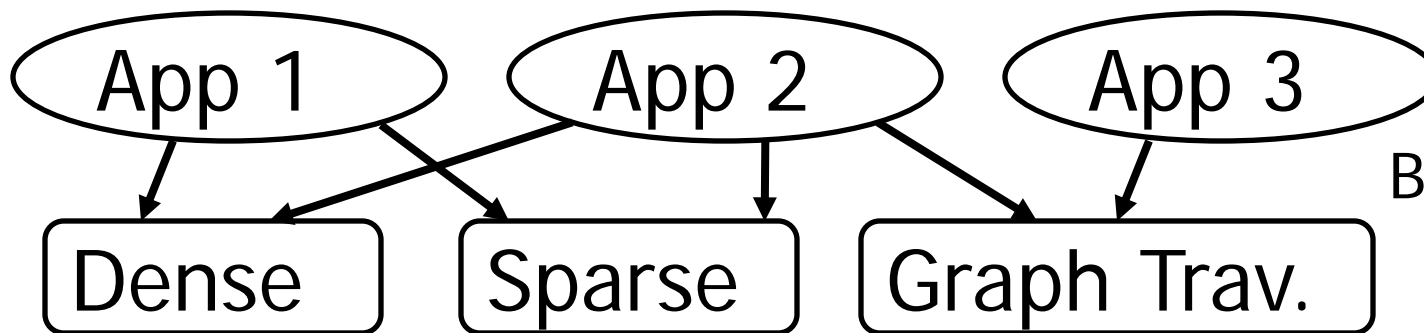
Krste Asanovic
UC Berkeley

MATEO-2012, Barcelona
June 29, 2012

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




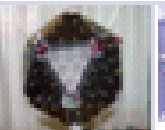
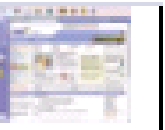
- ❖ Even if $\ll 1\%$ of programmers understand it
- ❖ Humanity's capabilities limited by biggest machines running most efficient code.
- ❖ Natural user interfaces are performance limited.
- ❖ More efficient primitives support more sloppy code above (e.g., Lua game scripting, or mashups built on top of warehouse-scale computer services)
- ❖ Efficient systems \Rightarrow Parallel systems

- ❖ 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

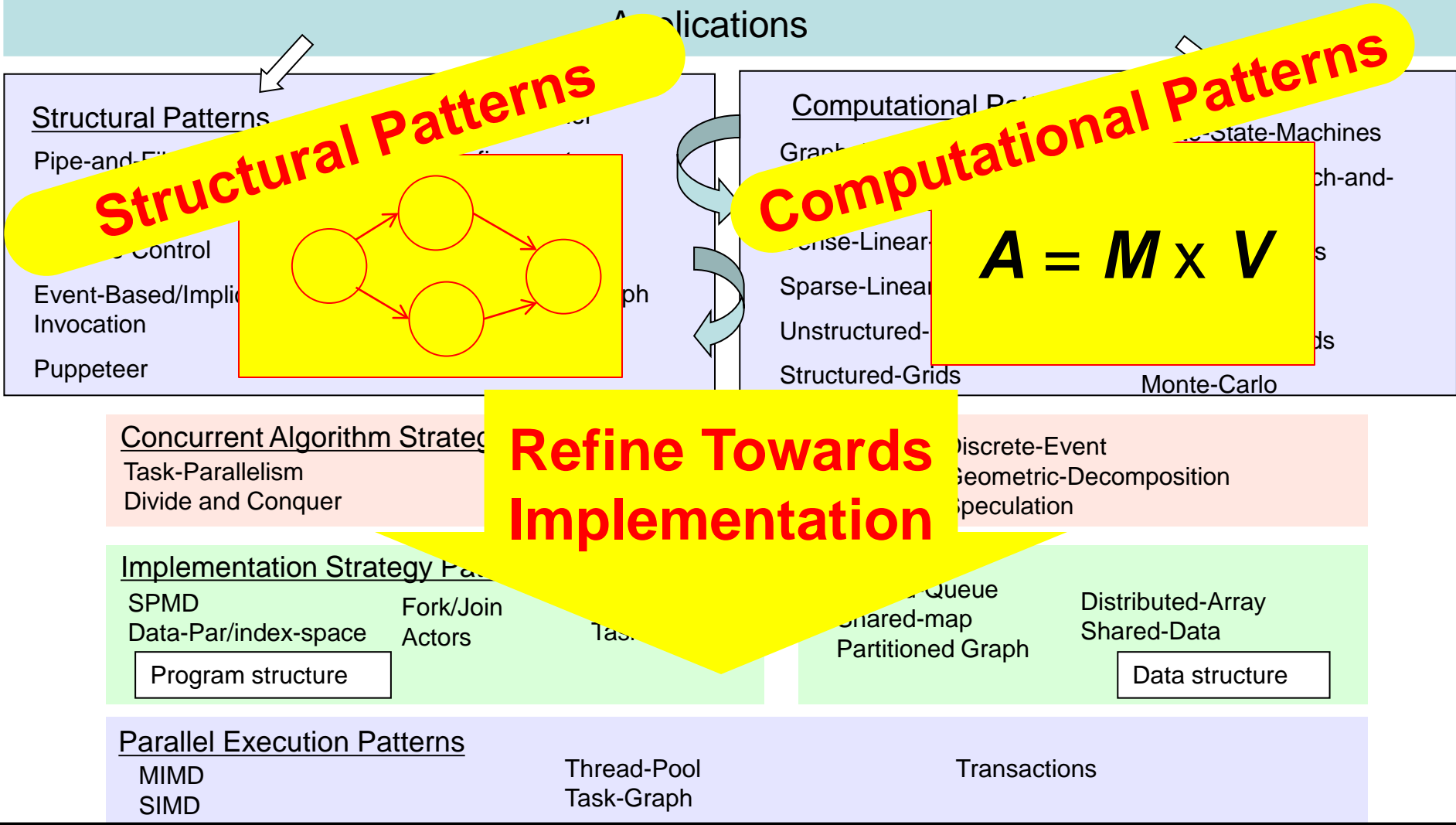


Berkeley View
Motifs
("Dwarfs")

How do compelling apps relate to 13 motifs?

	Embed	SPEC	DB	Games	ML	CAD	HPC	Health 	Image 	Speech 	Music 	Browser 
1 Finite State Mach.	Red	Red	Red	Yellow	Yellow	Yellow	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue
2 Circuits	Red	Light Blue	Light Green	Light Blue	Light Green	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue
3 Graph Algorithms	Red	Yellow	Light Blue	Yellow	Red	Red	Light Blue	Red	Light Blue	Red	Light Green	Light Green
4 Structured Grid	Red	Red	Light Blue	Yellow	Light Blue	Red	Light Blue	Light Blue	Red	Light Blue	Light Blue	Light Blue
5 Dense Matrix	Red	Red	Yellow	Red	Red	Red	Light Blue	Light Blue	Red	Red	Red	Light Blue
6 Sparse Matrix	Yellow	Yellow	Light Blue	Red	Red	Red	Light Blue	Red	Light Blue	Light Blue	Red	Light Blue
7 Spectral (FFT)	Yellow	Light Blue	Light Blue	Yellow	Yellow	Yellow	Red	Light Blue	Light Green	Red	Red	Red
8 Dynamic Prog	Yellow	Light Blue	Red	Light Blue	Red	Red	Light Blue	Light Blue	Light Blue	Yellow	Light Blue	Light Blue
9 Particle Methods	Light Blue	Yellow	Light Blue	Yellow	Light Blue	Red	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue
10 Backtrack/ B&B	Light Blue	Light Blue	Yellow	Light Blue	Red	Red	Light Blue	Light Blue	Light Blue	Light Blue	Yellow	Light Blue
11 Graphical Models	Light Blue	Light Blue	Yellow	Light Blue	Red	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue	Red	Light Blue
12 Unstructured Grid	Light Blue	Light Blue	Light Blue	Yellow	Yellow	Yellow	Red	Red	Light Blue	Light Blue	Red	Light Blue
13 Monte Carlo	Light Blue	Light Green	Light Blue	Red	Yellow	Yellow	Red	Red	Yellow	Light Blue	Yellow	Light Blue

"Our" Pattern Language (OPL-2010) (Kurt Keutzer, Tim Mattson)

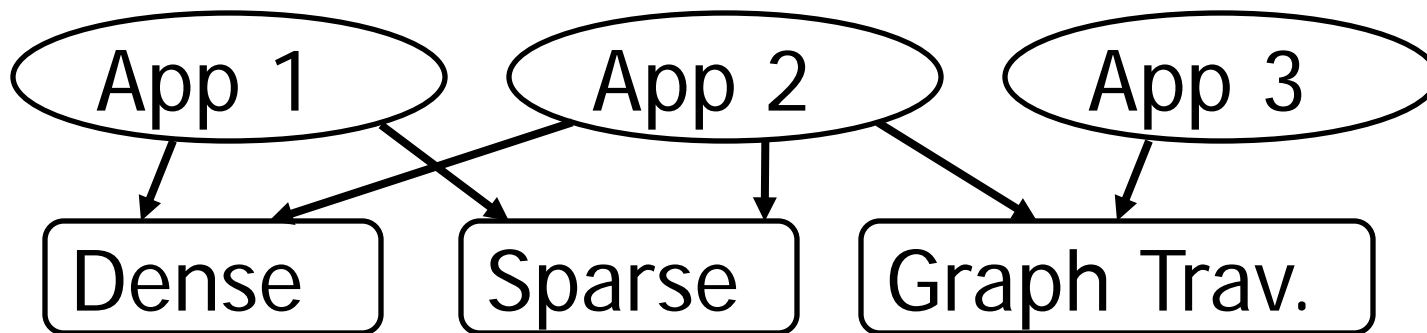


Concurrency Foundation constructs (not expressed as patterns)

Thread creation/destruction
Process creation/destruction

Message-Passing
Collective-Comm.

Point-To-Point-Sync. (mutual exclusion)
collective sync. (barrier)



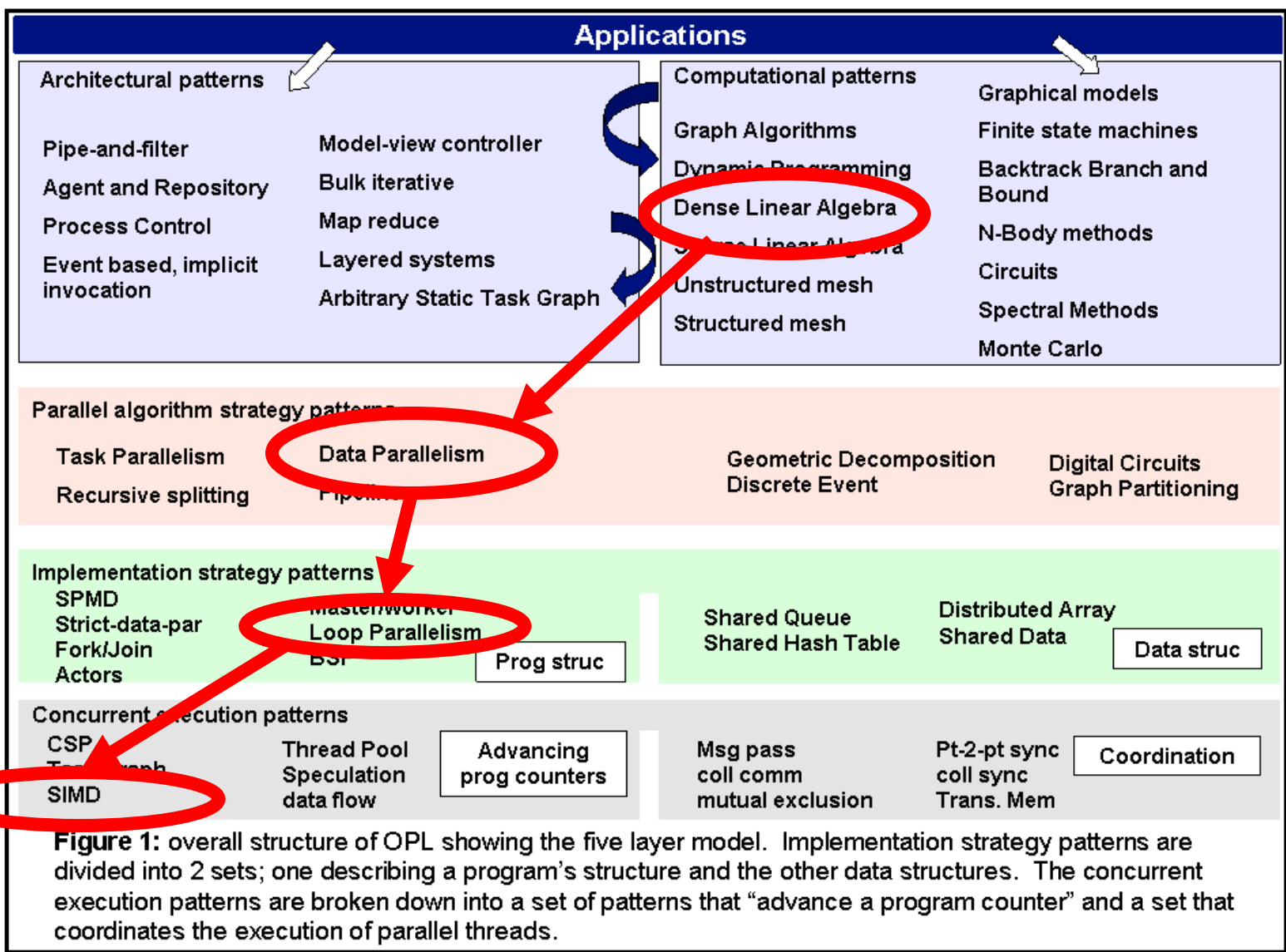
Only a few types of hardware platform

Multicore

GPU

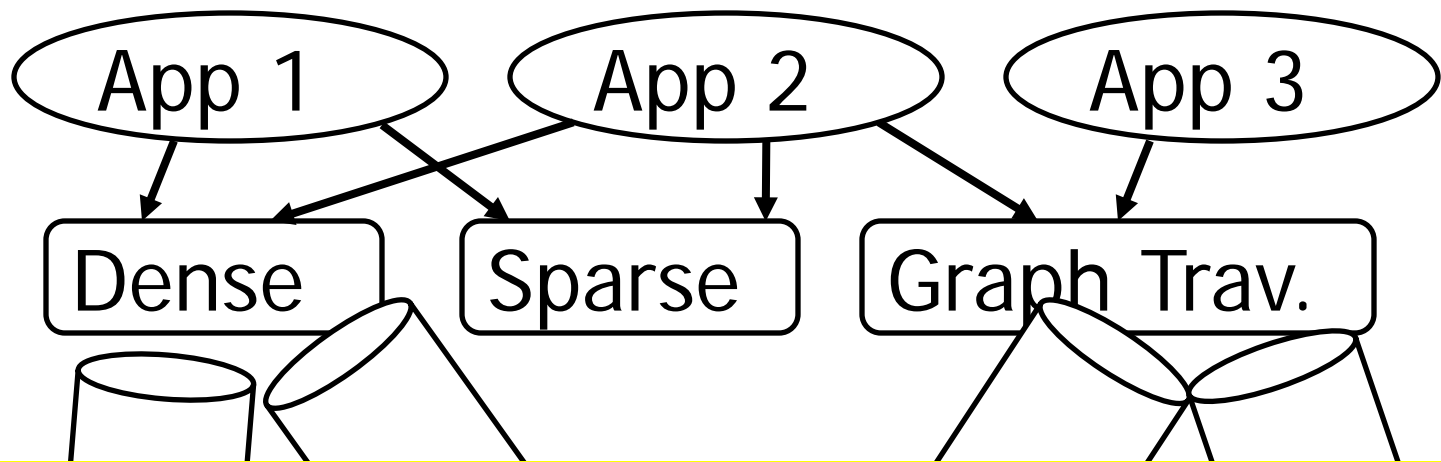
“Cloud”

High-level pattern constrains space of reasonable low-level mappings



Specializers: Pattern-specific and platform-specific compilers

aka. "Stovepipes"



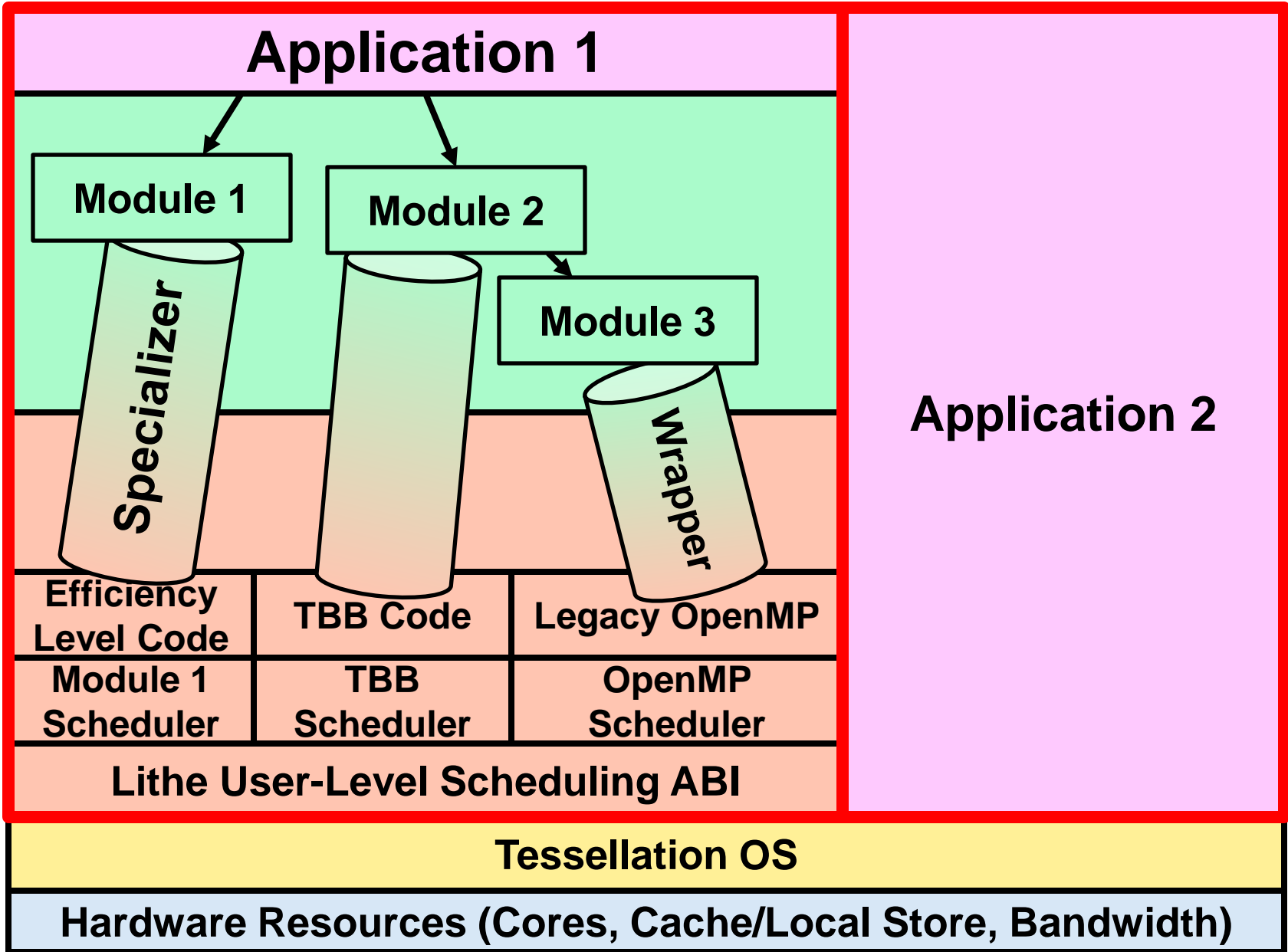
Allow maximum efficiency and expressibility in specializers by avoiding mandatory intermediary layers

Asp infrastructure supports specializer authors

Multicore

GPU

"Cloud"

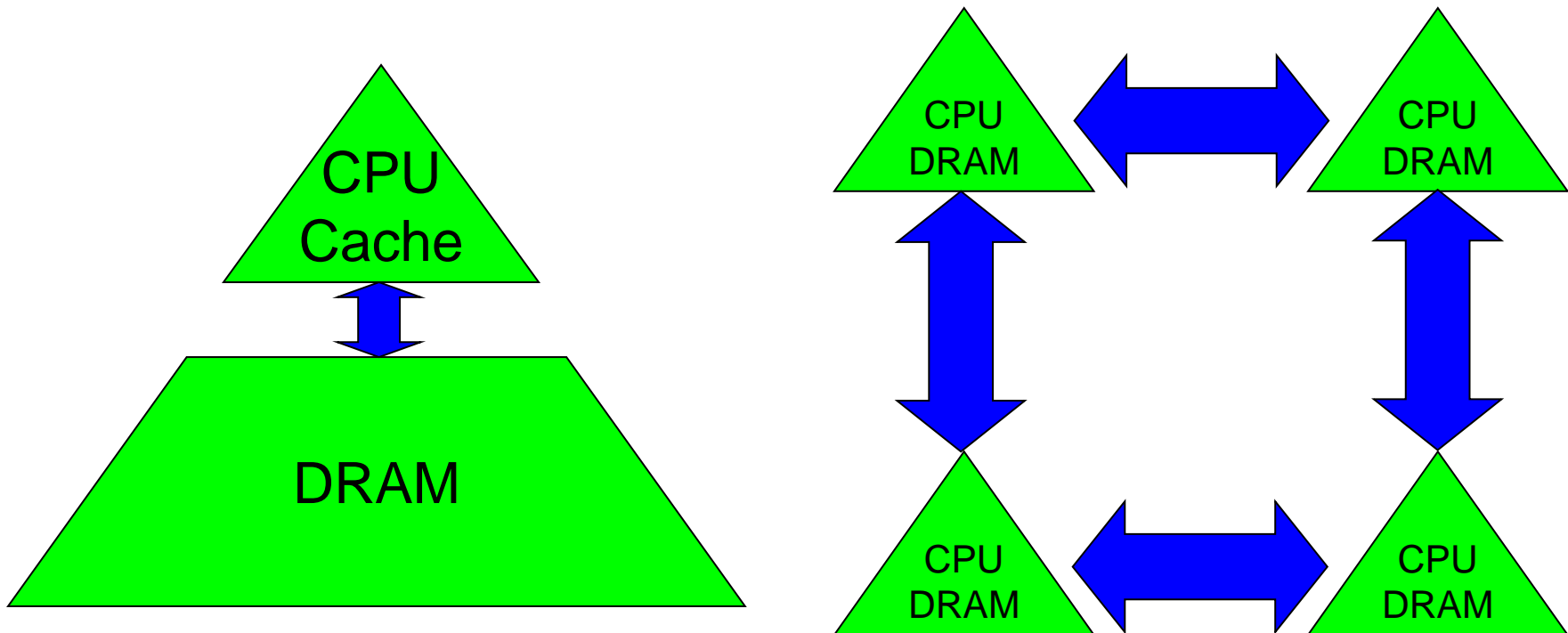


- ❖ Technology scaling slowing down/stopping
- ❖ No savior device technology on horizon
- ❖ Parallelism was one-time gain, using more, lower-performance cores for better energy efficiency
 - Simpler general-purpose microarchitectures
 - Limited by smallest sensible core
 - Lower V_{dd} /Frequency
 - Limited by V_{dd}/V_t 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

1. Arithmetic (FLOPS)
2. Communication: moving data between
 - levels of a memory hierarchy (sequential case)
 - processors over a network (parallel case).



- ❖ Communication = moving data, between levels of memory or between processors over a network
- ❖ 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 (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

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

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

$$\#messages_sent \geq \#words_moved / largest_message_size$$

❖ 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 $\#flops \ll n^3$)
- Sequential and parallel algorithms
- Some graph-theoretic algorithms (e.g. Floyd-Warshall)



- ❖ 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 $10M \times 10$
 - Up to **6.7x** on 16-proc. Pentium III cluster, for $100K \times 200$
 - Up to **13x** on Tesla C2050 / Fermi, for $110k \times 100$
 - 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.

- ❖ 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 ...

President Obama cites Communication-Avoiding Algorithms in the FY 2012 Department of Energy Budget Request to Congress:

“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.”

FY 2010 Congressional Budget, Volume 4, FY2010 Accomplishments, Advanced Scientific Computing Research (ASCR), pages 65-67.

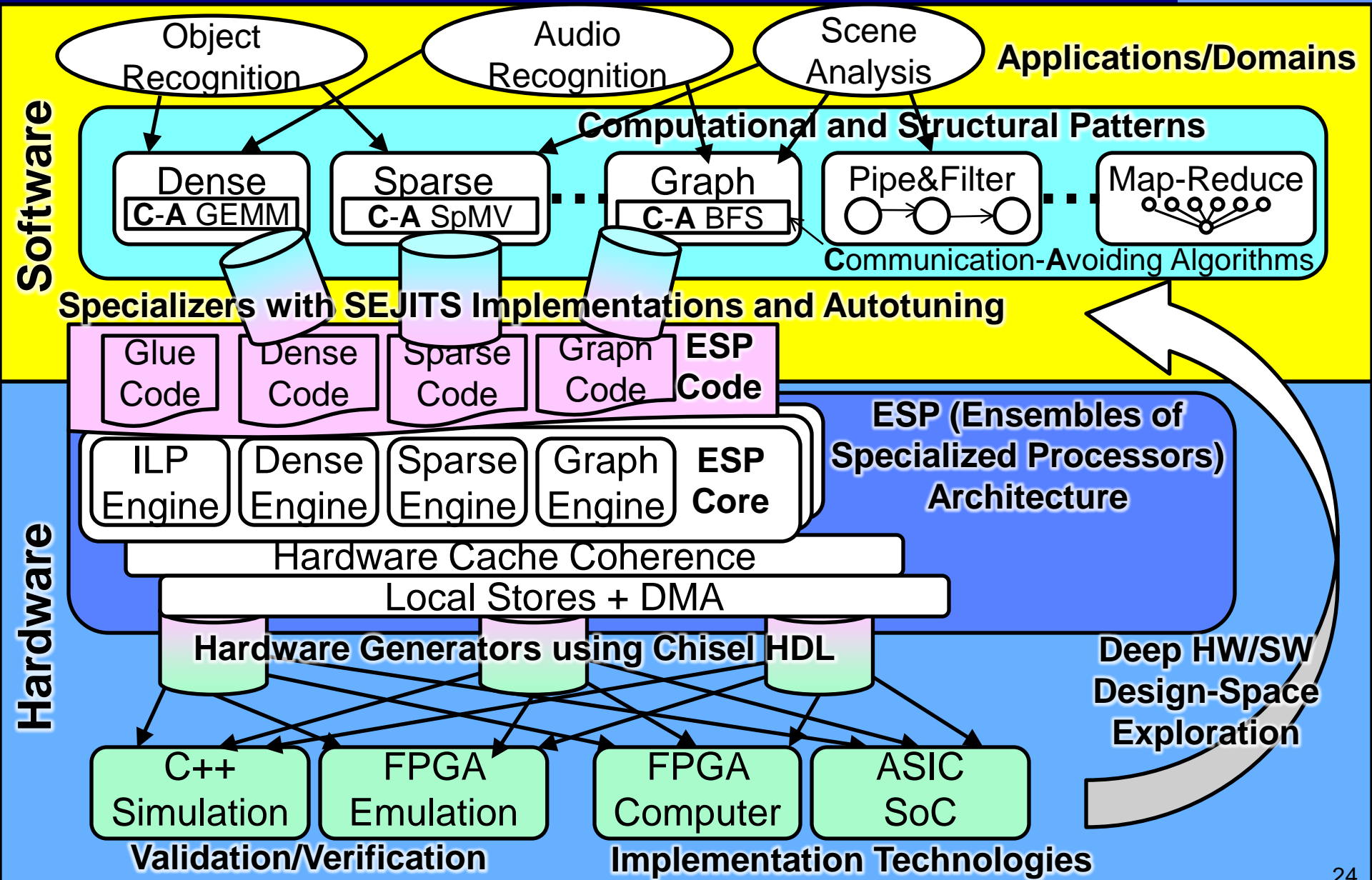
CA-GMRES (Hoemmen, Mohiyuddin, Yelick, Demmel)
“Tall-Skinny” QR (Grigori, Hoemmen, Langou, Demmel)

- ❖ 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

- ❖ 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



- ❖ 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.