Communication-avoidance
and
Automatic performance tuning

Nick Knight
UC-Berkeley Parlab (Bebop group)

Students and Collaborators (Bebop)

- PIs
  - James Demmel
  - Kathy Yelick
- Students
  - Marghoob Mohiyuddin
  - Shoaib Kamil
  - Vasily Volkov
  - Andrew Gearheart
  - Razvan Carbunescu
  - Grey Ballard
  - Nick Knight
  - Erin Carson
  - Edgar Solomonik
  - Michael Anderson
  - Gilad Arnold
- Collaborators
  - Mark Hoemmen (Sandia)
  - Mike Heroux (Sandia)
  - Jack Dongarra (UTK)
  - Laura Grigori (INRIA)
  - Simplice Donfack (INRIA)
  - Amal Khabou (INRIA)
  - Oded Schwartz (TU-Berlin + UCB)
  - Olga Holtz (UCB)
  - Ming Gu (UCB)
  - Ioana Dumitriu (UW)
  - Julien Langou (UC-Denver)
  - Samuel Williams (LBNL)
  - Aydin Buluc (LBNL)
  - Kamesh Madduri (LBNL)
  - Jong-Ho Byun (UCB)
  - Armando Fox (UCB)
  - Ras Bodik (UCB)
  - Tony Keaveny (UCB)

Talk outline

Developing efficient algorithms

- Strategy: avoiding communication
- Dense linear algebra
  - Heterogeneous comm. complexity
  - Eigenvalue problems
  - 2.5D algorithms
  - Fast matmul comm. complexity
  - CA-pivoting (ask me about it)
- Sparse linear algebra
  - CA-Krylov methods
  - New matrix powers kernel

Automatic performance tuning

- OSKI (Optimized Sparse Kernel Interface)
- Future development

Avoiding communication

Communication =

- Data movement and synchronization
- The dominant performance bottleneck in numerical codes

Comm. lower bounds for direct linear algebra:

- #words moved = Ω(#flops / $M^{1/2}$)
- #messages sent = Ω(#flops / $M^{3/2}$)
- Standard algorithms do *not* attain these bounds!

M = fast/local memory size

We develop:

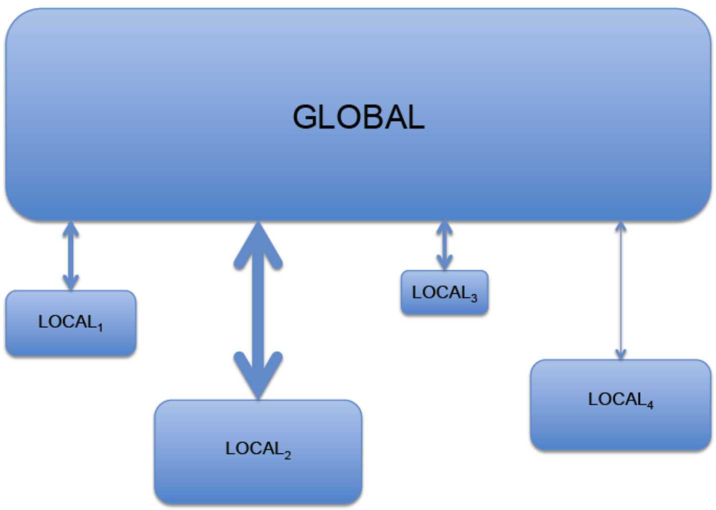
- *Direct* algorithms that attain these bounds (communication-optimal)
- *Iterative* algorithms that solve problems with optimal communication.

Speedups on today's, future hardware

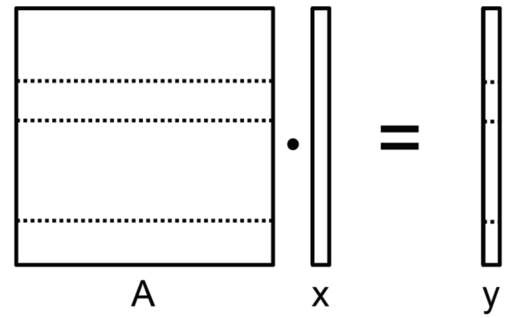
Talk outline

- Developing efficient algorithms
  - Strategy: avoiding communication
  - Dense linear algebra
    - Heterogeneous comm. complexity
    - Eigenvalue problems
    - 2.5D algorithms
    - Fast matmul comm. complexity
    - CA-pivoting (ask me about it)
  - Sparse linear algebra
    - CA-Krylov methods
    - New matrix powers kernel
- Automatic performance tuning
  - OSKI (Optimized Sparse Kernel Interface)
  - Future development

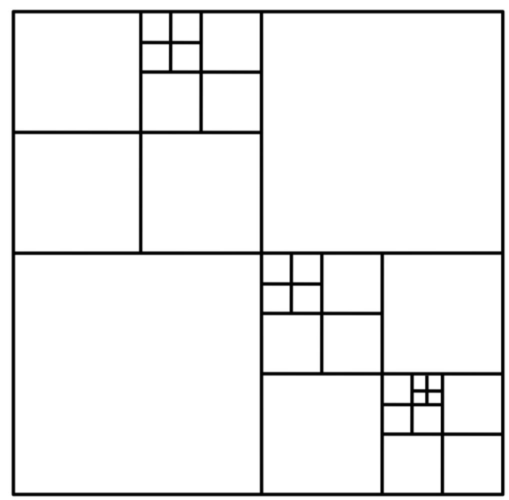
Heterogeneous comm. complexity



Heterogeneous memory model
(Different speeds, bandwidths, latencies)



(dense) matrix-vector multiply



(dense) matrix-matrix multiply

How best to reduce communication costs on heterogeneous machine?

- New theoretical results:
  - Model algorithm costs as linear program
  - Solution gives you optimal partition of flops
- New theory → FASTER ALGORITHMS

(SEE POSTER)
Gearhart, Ballard

Eigenvalue problems and SVD

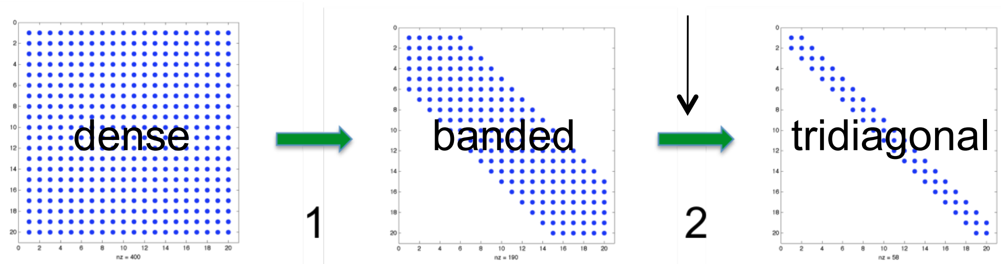
Applications:

- Image processing (M. Anderson's talk yesterday)

Previously:

- Presented three new algorithms at Winter 2010 retreat
- Solve eigenvalue and singular value problems with less communication

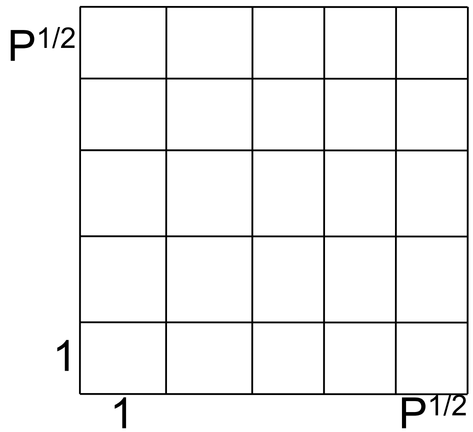
New algorithm for successive band-reduction



- Minimizes bandwidth *and* latency costs in serial
- Parallel SBR and implementations in development

2.5D algorithms I

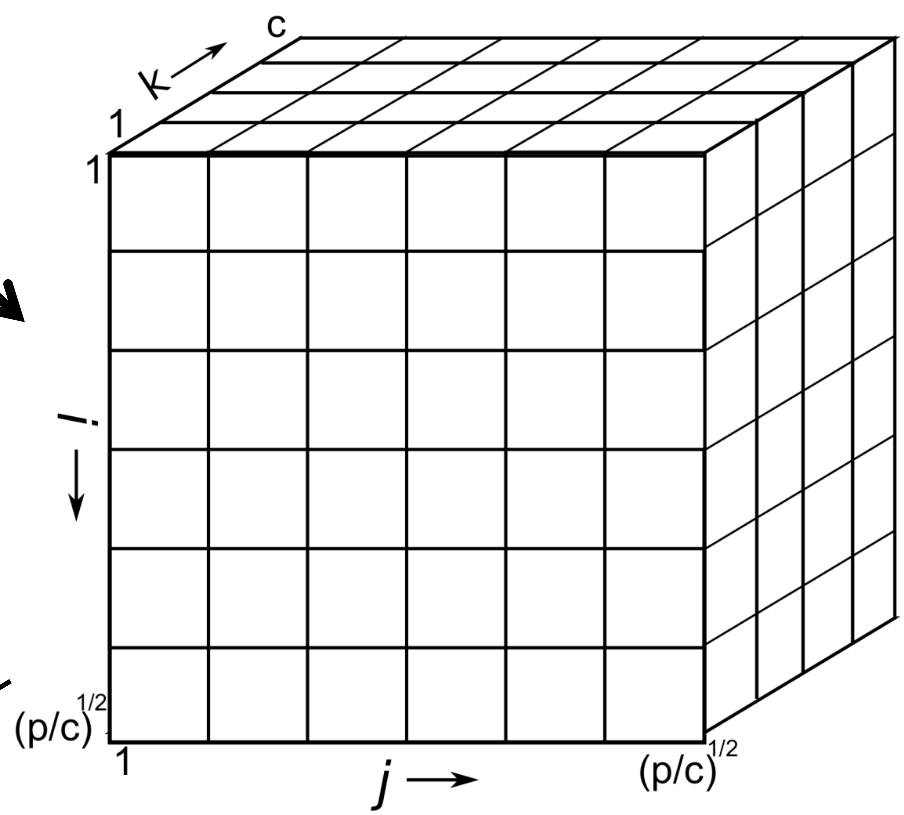
2D processor array



1 copy of data

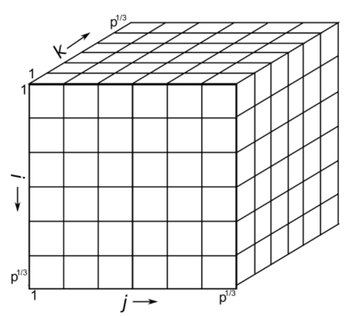
2.5D processor array

dimensions: $(P/c)^{1/2} \times (P/c)^{1/2} \times c$



$1 \le c \le P^{1/3}$ copies of data,
one per `plane' in the *k* direction

Special case: $c = P^{1/3}$ copies
Cube of processors: $P^{1/3} \times P^{1/3} \times P^{1/3}$



2.5D algorithms II

Do you have extra memory available?

In direct linear algebra, we can

use it to avoid communication

by storing $1 \leq c \leq P^{1/3}$ redundant copies of data.

New theory….

- Potential 2.5D communication savings
  - #words moved, by factor of $c^{1/2}$
  - #messages sent, by factor of $c^{3/2}$
  - minimized in 3D case ($c = P^{1/3}$)

→ Tune to find optimal duplication factor c

Flanders ExaScience Lab (Intel Labs Europe) sending visitor this spring.

…. leads to faster algorithms

- New 2.5D matrix-matrix multiplication algorithm
  - Predict up to 5x speedups on future exascale hardware
  - using up to *c*=100 copies
- Also new 2.5D LU algorithm …

(SEE POSTER)
Solomonik

EECS Electrical Engineering and Computer Sciences

Fast matmul comm. complexity

BERKELEY PAR LAB

Direct linear algebra does $\Theta(n^\omega)$ flops.

Conventional algorithms: $\omega = 3$
Strassen-like algorithms: $\omega < 3$
eg: $\omega \approx 2.81$ (Strassen's matmul)

New theory ...

- Strassen-like matmuls can communicate less than conventional matmuls.
- # words moved decreases: $\Omega\left(\frac{n^3}{M^{1/2}}\right) = \Omega\left(M\left(\frac{n}{M^{1/2}}\right)^3\right) \longrightarrow \Omega\left(M\left(\frac{n}{M^{1/2}}\right)^\omega\right)$
- # messages M times smaller

… leads to new algorithms

- Result: Strassen-like matmul attains these lower bounds in serial
- Can we attain these bounds in parallel?
  - We believe we can… (current work)
- Rest of sequential direct linear algebra (LU, QR, …) can attain the same bounds.
  - We believe it can in parallel too… (current work)

Talk outline

- Developing efficient algorithms
  - Strategy: avoiding communication
  - Dense linear algebra
    - Heterogeneous comm. complexity
    - Eigenvalue problems
    - 2.5D algorithms
    - Fast matmul comm. complexity
    - CA-pivoting (ask me about it)
  - Sparse linear algebra
    - CA-Krylov methods
    - New matrix powers kernel
- Automatic performance tuning
  - OSKI (Optimized Sparse Kernel Interface)
  - Future development

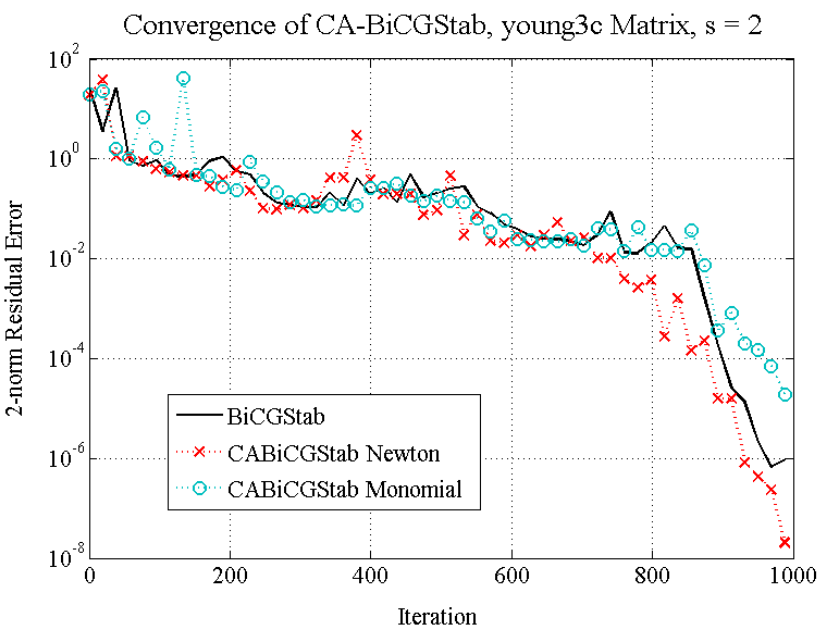
CA-Krylov methods I

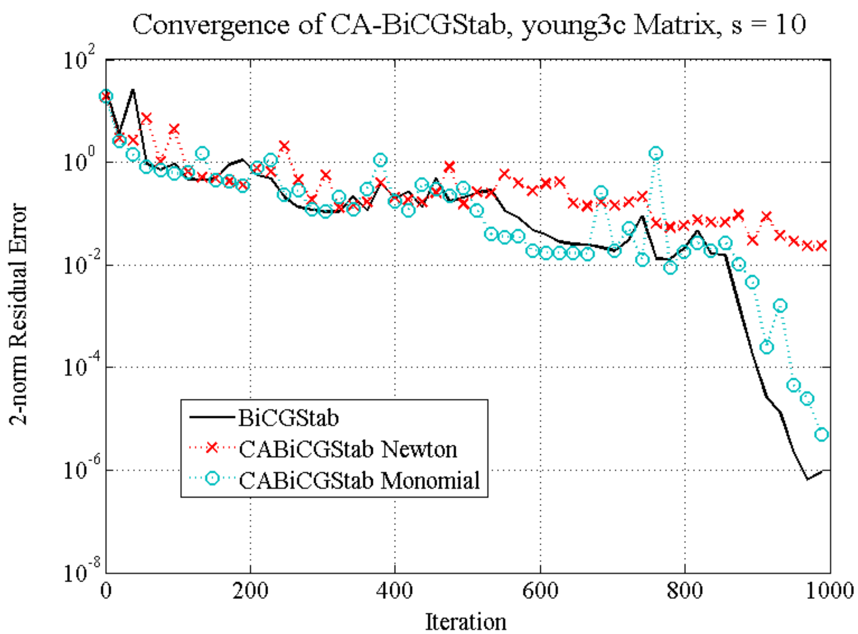
Communication-avoiding biconjugate gradient method

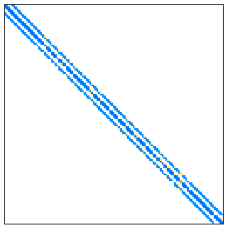
"CA-BiCG"

- Previously seen in poster, summer 2010 retreat
  - BiCG solves sparse, nonsymmetric systems of equations
  - CA-BiCG mathematically-equivalent formulation; takes multiple iterations with communication cost of one iteration of BiCG.
- BiCG vs. GMRES
  - GMRES: faster convergence and more stable, in theory and in practice
  - BiCG: small, constant-size workspace, less work per iteration
  - Stabilized variants (eg, BiCGStab) used in practice
- New communication-avoiding algorithms:
  - BiCG (2-term), CGS, BiCGStab, BiCGStab(*l*)

CA-Krylov methods II







young3c
N=841
nnz=3988
κ=1.15e4

- Reduce communication by factor of s
- CA-BiCGStab follows convergence of standard BiCGStab, even with s=10
- Hard problem! BiCGStab fails to converge in n its.
  - Preconditioning needed

(SEE POSTER)
Knight, Carson

CA-Krylov methods III

Communication-avoiding generalized minimum residual method

"CA-GMRES"

- Previously seen in poster, winter 2010 retreat
- Success story: Parlab → DOE-funded
  - CA-GMRES expected to appear in Spring 2011 Trilinos release
  - Contains tall-skinny QR (TSQR) based on Parlab work
    - Intel TBB + MPI
  - Ongoing work:
    - Incorporate M. Anderson's GPU TSQR (SEE POSTER)
    - New CA-GMRES variants: Flexible GMRES, Recycling GMRES
    - Fault tolerance

New matrix powers kernel

- Previously (Winter 2010):
  - Matrix powers kernel: key to avoiding synchronization in Krylov subspace methods (BiCG, GMRES, etc)
  - $[A, s, x] \rightarrow [x, Ax, A^2x, \ldots, A^sx]$
  - Analyzes system A at runtime

New algorithmic variants required by new CA Krylov methods:

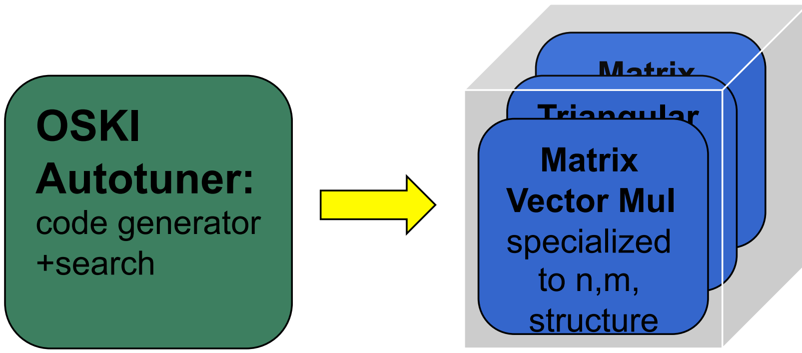
- Both A and A^T
  - $[A, s, x] \rightarrow [[x, Ax, A^2x, \ldots, A^sx], [x, A^Tx, (A^T)^2x, \ldots, (A^T)^sx]]$
  - Multiple source vectors
  - $[A, s, X] \rightarrow [X, AX, A^2X, \ldots, A^sX]$
- Hypergraph partitioning (new communication model)
  - Beats current graph partitioning approach for structured, nonsymmetric matrices. (Up to 80% fewer words moved)
- Extends to (nonlinear) Health App

(SEE POSTER)
Carson, Knight

Talk outline

- Developing efficient algorithms
  - Strategy: avoiding communication
  - Dense linear algebra
    - Heterogeneous comm. complexity
    - Eigenvalue problems
    - 2.5D algorithms
    - Fast matmul comm. complexity
    - CA-pivoting (ask me about it)
  - Sparse linear algebra
    - CA-Krylov methods
    - New matrix powers kernel
- Automatic performance tuning
  - OSKI (Optimized Sparse Kernel Interface)
  - OSKI development

OSKI (Optimized Sparse Kernel Interface)



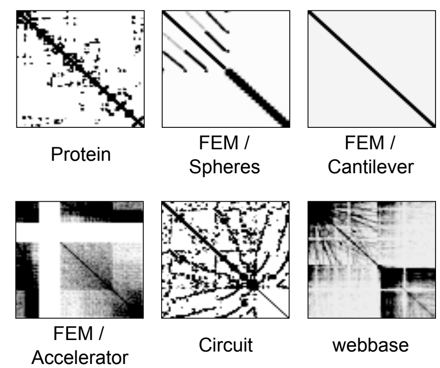
OSKI Library

- Functional portability
  - Python interface (via SEJITS)
  - C code underneath
- Performance portability
  - search/tune at install time

Optimized Sparse Kernel Interface (OSKI):

Autotuned Sparse Matrix-Vector Multiplication (SpMV)

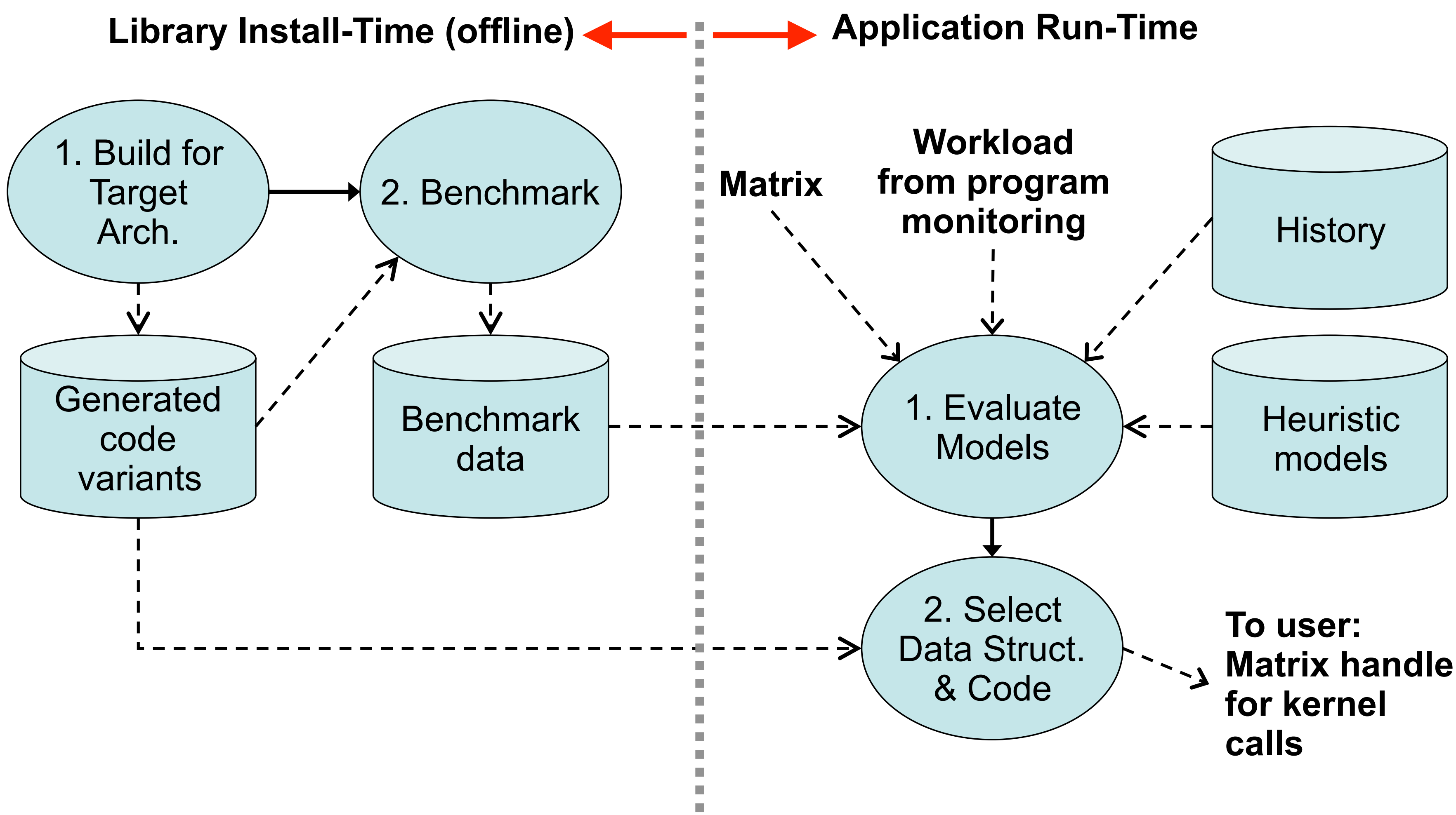
- Huge algorithm design space
- Performance = f(structure,dimension)
  - vs. dense matrix-vector mult: Perf = f(dimension)
  - Runtime tuning necessary



Efficient sparse codes are difficult to write
(SEE POSTER) Arnold, Bodik

OSKI (Optimized Sparse Kernel Interface)

How OSKI tunes:



Extensibility: Advanced users may write & dynamically add “Code variants” and “Heuristic models” to system.

EECS
Electrical Engineering and Computer Sciences

OSKI development I

BERKELEY PAR LAB

Algorithm design space for next (p)OSKI release:

- Index compression
- Array padding
- Software prefetching
- Software pipelining
- Loop unrolling (depths)
- SpMM (multiple source vectors)
- Variable block splitting
- Switch-to-dense (SpTS)
- Cache interleaving (A^TA)
- Sparse tiling (A^kx)
- Symmetric storage for multicore
- SIMD intrinsics
- Data decomposition (shared/dist)
- NUMA awareness
- Hiding latency
- Reordering (RCM, TSP, …)
- TLB blocking
- Cache-blocking heuristics
- Storage formats:
  - CSB (compressed sparse block)
  - Vector-style (manycore/GPU)
  - *DCSR (delta-coded CSR)*
  - *RPCSR (row-pattern CSR)*
  - *PBR (pattern-based repr.)*
  - *RSDF (row-segmented diagonal fmt.)*

EECS
Electrical Engineering and Computer Sciences

OSKI development II

BERKELEY PAR LAB

- Requested functionality (from HPC world):
  - Change non-zero pattern of the matrix
    - Matrix may change or be perturbed during computation
  - Assemble a matrix from (possibly overlapping) fragments
    - Common in finite element methods
  - Perform variable block splitting
    - $A = A_1 + A_2$ where A_1 and A_2 have different natural block sizes
- Our (proposed) solutions:
  - List_of_matrices: allows a matrix to be expressed as a sum of matrices ($A = A_1 + ... + A_n$)
    - Easily allows for assembly from fragments and variable splitting
    - Pattern update: represent the changed entry as the addition of another matrix
  - Merge() method: Merges the list $A_1 + ... + A_n$ into a single matrix
    - User can decide when to merge matrices, or…
    - In the future, merging may also be a tuning decision made by OSKI

Questions?

Extra Slides

Summary of Performance Optimizations

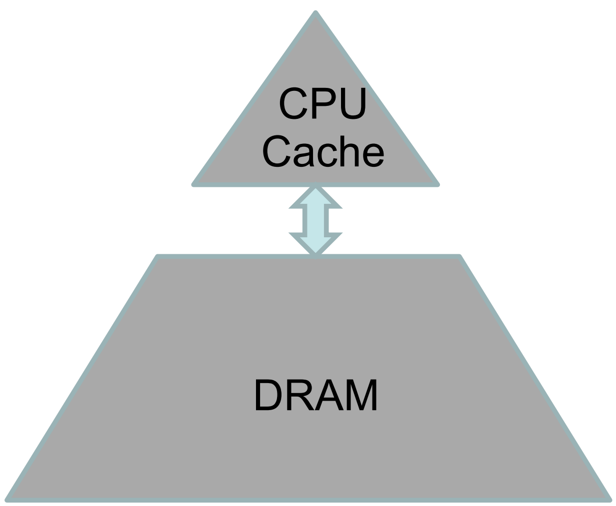
- Optimizations for SpMV
  - **Register blocking (RB)**: up to **4x** over CSR
  - **Variable block splitting**: **2.1x** over CSR, 1.8x over RB
  - **Diagonals**: **2x** over CSR
  - **Reordering** to create dense structure + **splitting**: **2x** over CSR
  - **Symmetry**: **2.8x** over CSR, 2.6x over RB
  - **Cache blocking**: **2.8x** over CSR
  - **Multiple vectors (SpMM)**: **7x** over CSR
  - And combinations…
- Sparse triangular solve
  - Hybrid sparse/dense data structure: **1.8x** over CSR
- Higher-level kernels
  - **$A \cdot A^T \cdot x$, $A^T \cdot A \cdot x$**: **4x** over CSR, 1.8x over RB
  - **$A^2 \cdot x$**: **2x** over CSR, 1.5x over RB
  - $[A \cdot x, A^2 \cdot x, A^3 \cdot x, .. , A^k \cdot x]$

Why Avoid Communication?

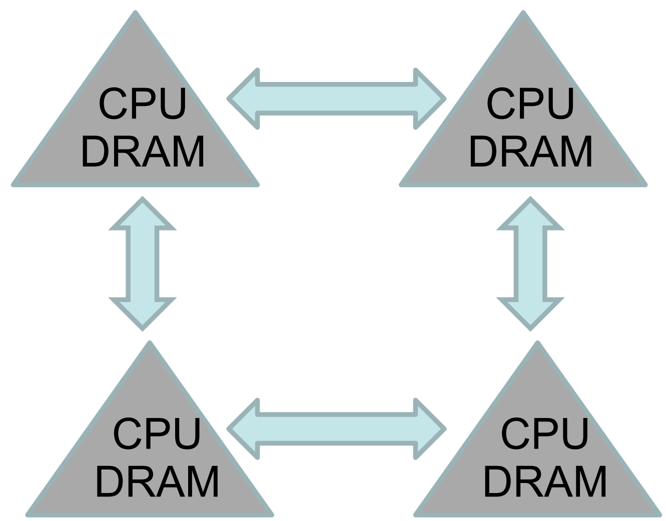
Algorithms have two costs:

1. Arithmetic (flops)
2. Communication: moving data between

- levels of a memory hierarchy (sequential)



- processors (parallel)
  - messages (distributed mem)



  - cache-coherency (shared mem)
  - data transfers (bus-based)

Why Avoid Communication?

- Running time of an algorithm is sum of 3 terms:
  - # flops * time_per_flop
  - # words moved / bandwidth ⎫
  - # messages * latency ⎭ communication

- Time_per_flop << 1/ bandwidth << latency
  - Gaps growing exponentially with time (FOSC, 2004)

Annual improvements			
Time_per_flop		Bandwidth	Latency
59%	Network	26%	15%
	DRAM	23%	5%

- Goal : reorganize linear algebra to *avoid* communication
  - Between all memory hierarchy levels
    - L1 ⟷ L2 ⟷ DRAM ⟷ network, etc
  - Not just *hiding* communication (speedup ≤ 2x)
  - Arbitrary speedups possible

25

Avoiding Communication

- Let M = "fast" memory size (per processor). Then,

$$\#\ \text{words moved (per processor)} = \Omega\left(\frac{\#\ \text{flops per processor}}{M^{1/2}}\right)$$

$$\#\ \text{messages sent (per processor)} = \Omega\left(\frac{\#\ \text{flops per processor}}{M^{3/2}}\right)$$

- Parallel case: assume either load- or memory- balanced
- Trivial lower bound: #words moved ≥ #inputs + #outputs

- Holds for:
  - BLAS, LU, QR, EVD/SVD, tensor contractions, …
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, e.g., A^k)
  - Sequential and parallel algorithms
  - Some graph theoretic algorithms (e.g., Floyd-Warshall)

EECS
Electrical Engineering and Computer Sciences

2.5D algorithms

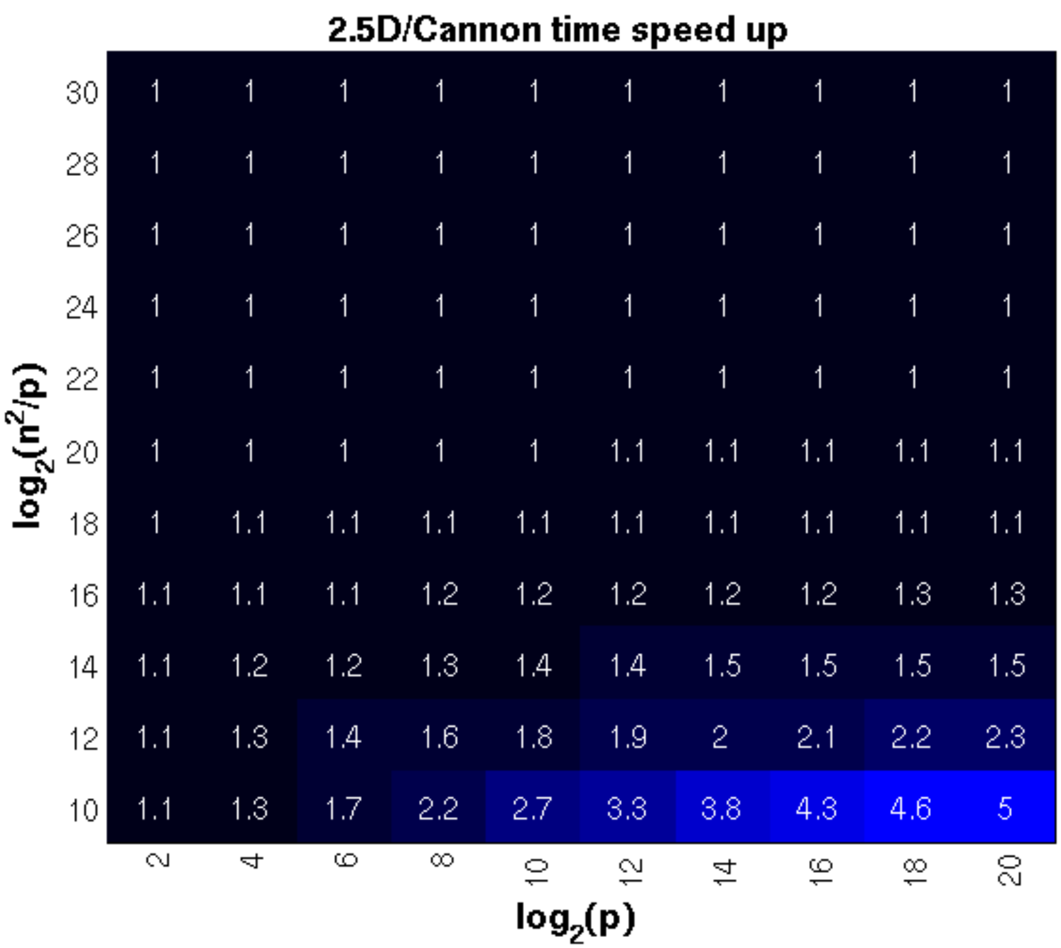
BERKELEY PAR LAB

- P processors: # words moved = $\Omega\left(\frac{n^3/P}{M^{1/2}}\right)$ (intuition: make M bigger!)
- 2D algorithms: distribute matrices (2D arrays) across $\sqrt{P} \times \sqrt{P}$ (logical) 2D grid of processors
  - If one copy of data, $M = \frac{n^2}{P}$
  - What if you have extra memory?
- 3D algorithms: distribute matrices across $P^{1/3} \times P^{1/3} \times P^{1/3}$ processor cube,
  - $P^{1/3}$ duplicate copies of data (M increased by a factor of $P^{1/3}$)
  - This decreases lower bounds for:
    - # words moved by a factor of $P^{1/6}$
    - # messages sent by a factor of $P^{1/2}$
- 2.5D algorithms:
  - $1 \leq c \leq P^{1/3}$ copies of data: smooth transition between 2D to 3D bounds.
  - Flexibility

SEE POSTER

2.5D algorithms

Predicted exascale speedups



Model Parameters:

- 1×10^{18} flops/s (`exa-')
- 24 PB total memory
- 2^{20} nodes → 24 GB/node
- 100 GB/s interconnect bandwidth (overestimate?)
- 100 ns network latency

2.5D algorithms

2.5D Best choice of c for efficiency

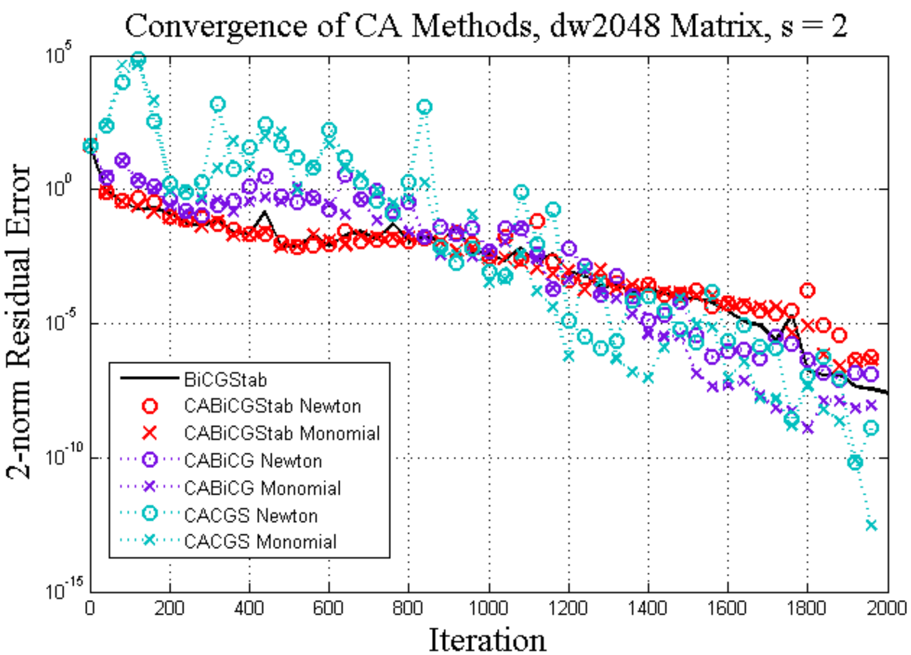
$\log_2(n^2/p)$ \ $\log_2(p)$	2	4	6	8	10	12	14	16	18	20
30	1	1	1	1	1	1	1	1	1	1
28	2	3	4	4	4	4	4	4	4	4
26	2	3	4	6	10	16	15	16	15	16
24	2	3	4	6	10	16	25	40	64	57
22	2	3	4	6	10	16	25	40	64	102
20	2	3	4	6	10	16	25	40	64	102
18	2	3	4	6	10	16	25	40	64	102
16	2	3	4	6	10	16	25	40	64	102
14	2	3	4	6	10	16	25	40	64	102
12	2	3	4	6	10	16	25	40	64	102
10	2	3	4	6	10	16	25	40	64	102

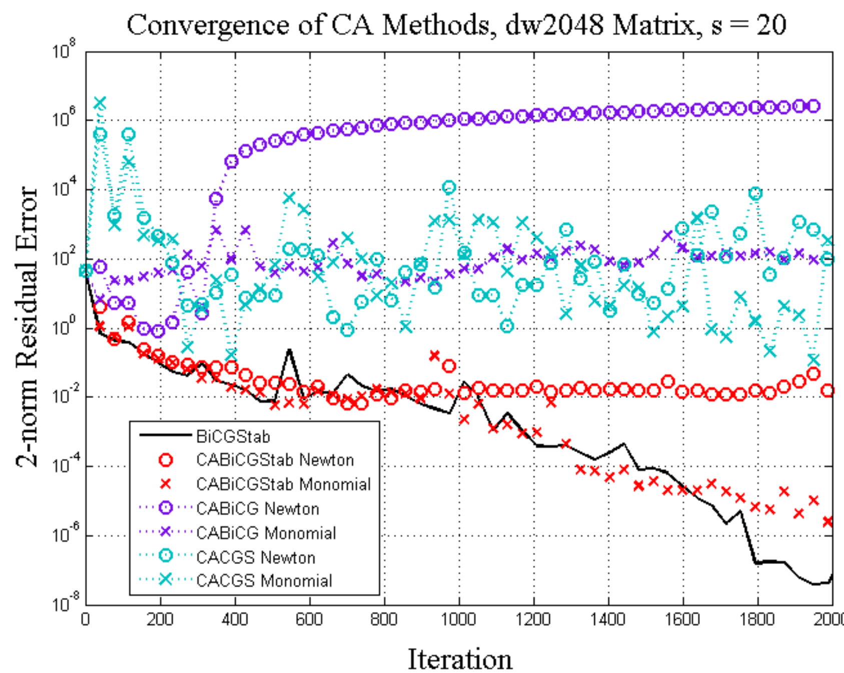
Model Parameters:

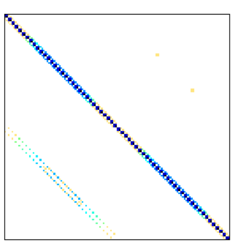
- 1×10^{18} flops/s (`exa-')
- 24 PB total memory
- 2^{20} nodes → 24 GB/node
- 100 GB/s interconnect bandwidth (overestimate?)
- 100 ns network latency

CA-Krylov methods II

CA-BiCGStab convergence







Name	n	NNZ	Pattern Symmetry	Value Symmetry	Condition Number	Application
dw2048	2048	10114	No	No	5.3015e3	Electromagnetics Problem (H. Dong, 1993)

EECS Electrical Engineering and Computer Sciences

Fast matmul comm. complexity

BERKELEY PAR LAB

- M = fast memory size
- Conventional matrix-matrix multiplication (matmul)
  - # flops = $2n^3$
  - # words moved = $\Omega\left(\frac{n^3}{M^{1/2}}\right) = \Omega\left(M\left(\frac{n}{M^{1/2}}\right)^3\right)$, attainable
  - # messages = $\Omega\left(\frac{n^3}{M^{3/2}}\right) = \Omega\left(\left(\frac{n}{M^{1/2}}\right)^3\right)$, attainable
- Strassen's matmul
  - # flops = $\Theta(n^\omega)$ where $\omega = \log_2(7) \approx 2.81$
  - # words moved = $\Omega\left(M\left(\frac{n}{M^{1/2}}\right)^\omega\right)$, attainable too
  - # messages = $\Omega\left(\left(\frac{n}{M^{1/2}}\right)^\omega\right)$, attainable too
- Applies to all other Fast matmul algorithms we know
  - How broadly does it apply?
    - We know rest of linear algebra can be done in $O(n^\omega)$ flops and attain these #words moved and #messages, in serial.
    - If these are valid lower bounds, then they are tight