COALA: **C**ommunication **O**ptimal **A**lgorithms for **L**inear **A**lgebra

Jim Demmel
EECS & Math Depts.
UC Berkeley

Laura Grigori
INRIA Saclay
Ile de France

Collaborators and Supporters

- Collaborators at Berkeley (campus and LBL)
  - Michael Anderson , Grey Ballard, Jong-Ho Byun, Erin Carson , Ming Gu , Olga Holtz, Nick Knight, Marghoob Mohiyuddin , Hong Diep Nguyen, Oded Schwartz, Edgar Solomonik , Vasily Volkov, Sam Williams, Kathy Yelick, other members of BEBOP, ParLab and CACHE projects
- Collaborators at INRIA
  - Marc Baboulin, Simplice Donfack, Amal Khabou, Long Qu, Mikolaj Szydlarski, Alok Gupta, Sylvain Peyronnet
- Other Collaborators
  - Jack Dongarra (UTK), Ioana Dumitriu (U. Wash), Mark Hoemmen (Sandia NL), Julien Langou (U Colo Denver), Michelle Strout (Colo SU), Hua Xiang (Wuhan)
  - Other members of EASI, MAGMA, PLASMA, TOPS projects
- Supporters
  - INRIA, NSF, DOE, UC Discovery
  - Intel, Microsoft, Mathworks, National Instruments, NEC, Nokia, NVIDIA, Samsung, Sun

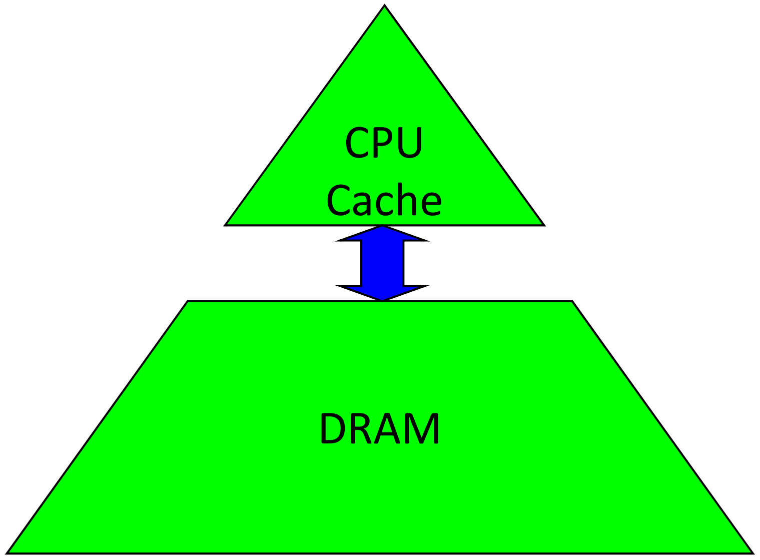
Outline

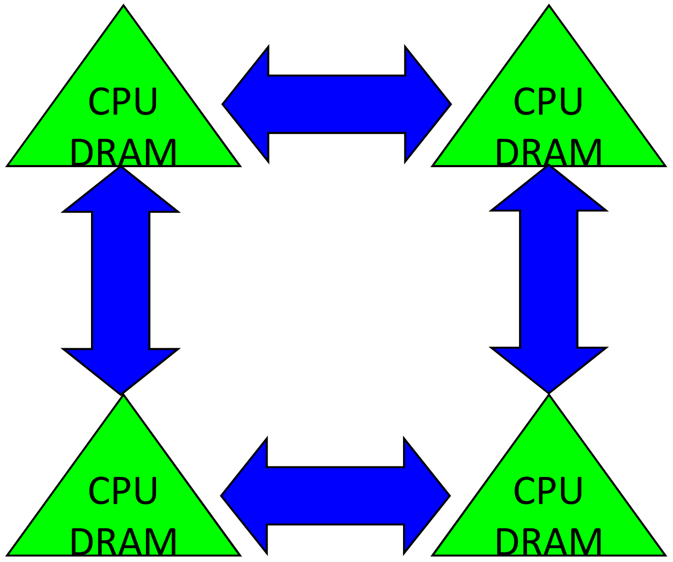
- Why we need to "avoid communication," i.e. avoid moving data
- "Direct" Linear Algebra
  - Lower bounds on communication for linear algebra problems like $Ax=b$, least squares, $Ax = \lambda x$, SVD, etc
  - New algorithms that attain these lower bounds
    - *Not* in libraries like Sca/LAPACK (yet!)
    - Large speed-ups possible
- "Iterative" Linear Algebra
  - Ditto for Krylov Subspace Methods

Why avoid communication? (1/2)

Algorithms have two costs:

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





Why avoid communication? (2/2)

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

$\left.\begin{array}{l}\text{\# words moved / bandwidth}\\ \text{\# messages * latency}\end{array}\right\}$ 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 $\longleftrightarrow$ L2 $\longleftrightarrow$ DRAM $\longleftrightarrow$ network, etc
  - Not just *hiding* communication (speedup $\leq$ 2x)
  - Arbitrary speedups possible

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, JD)
“Tall-Skinny” QR (Hoemmen, Langou, LG, JD)

Lower bound for all "direct" linear algebra

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

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

#messages_sent (per processor) = Ω(#flops (per processor) / $M^{3/2}$)

- Holds for anything that "smells like" 3 nested loops
  - BLAS, LU, QR, eig, SVD, tensor contractions, …
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, eg A^k)
  - Dense and sparse matrices (where #flops $\ll$ n^3)
  - Sequential and parallel algorithms
  - Some graph-theoretic algorithms (eg Floyd-Warshall)

Can we attain these lower bounds?

- Do conventional dense algorithms as implemented in LAPACK and ScaLAPACK attain these bounds?
  - Mostly not
- If not, are there other algorithms that do?
  - Yes, for dense linear algebra
- Only a few sparse algorithms so far
  - Cholesky on matrices with good separators
  - [David, Peyronnet, LG, JD]

TSQR: QR of a Tall, Skinny matrix

$$W = \begin{pmatrix} W_0 \\ \hline W_1 \\ \hline W_2 \\ \hline W_3 \end{pmatrix}$$

$$\begin{pmatrix} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{pmatrix} = \begin{pmatrix} Q_{01}\ R_{01} \\ \hline Q_{11}\ R_{11} \end{pmatrix}$$

$$\begin{pmatrix} R_{01} \\ \hline R_{11} \end{pmatrix} = \begin{pmatrix} Q_{02}\ R_{02} \end{pmatrix}$$

TSQR: QR of a Tall, Skinny matrix

$$W = \begin{pmatrix} W_0 \\ \hline W_1 \\ \hline W_2 \\ \hline W_3 \end{pmatrix} = \begin{pmatrix} Q_{00}\,R_{00} \\ \hline Q_{10}\,R_{10} \\ \hline Q_{20}\,R_{20} \\ \hline Q_{30}\,R_{30} \end{pmatrix} = \begin{pmatrix} Q_{00} & & & \\ \hline & Q_{10} & & \\ \hline & & Q_{20} & \\ \hline & & & Q_{30} \end{pmatrix} \cdot \begin{pmatrix} R_{00} \\ \hline R_{10} \\ \hline R_{20} \\ \hline R_{30} \end{pmatrix}$$

$$\begin{pmatrix} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{pmatrix} = \begin{pmatrix} Q_{01}\,R_{01} \\ \hline Q_{11}\,R_{11} \end{pmatrix} = \begin{pmatrix} Q_{01} & \\ \hline & Q_{11} \end{pmatrix} \cdot \begin{pmatrix} R_{01} \\ \hline R_{11} \end{pmatrix}$$

$$\begin{pmatrix} R_{01} \\ \hline R_{11} \end{pmatrix} = \begin{pmatrix} Q_{02}\,R_{02} \end{pmatrix}$$

Minimizing Communication in TSQR

Parallel:

$$\begin{tikzcd}
 & W_0 \arrow[r] & R_{00} \arrow[r] & R_{01} \arrow[dr] & \\
W = & W_1 \arrow[r] & R_{10} \arrow[ur] & & R_{02} \\
 & W_2 \arrow[r] & R_{20} \arrow[r] & R_{11} \arrow[ur] & \\
 & W_3 \arrow[r] & R_{30} \arrow[ur] & &
\end{tikzcd}$$

Sequential:

$$\begin{tikzcd}
 & W_0 \arrow[r] & R_{00} \arrow[dr] & & & \\
W = & W_1 \arrow[rr] & & R_{01} \arrow[dr] & & \\
 & W_2 \arrow[rrr] & & & R_{02} \arrow[dr] & \\
 & W_3 \arrow[rrrr] & & & & R_{03}
\end{tikzcd}$$

Dual Core:

$$\begin{tikzcd}
 & W_0 \arrow[r] & R_{00} \arrow[dr] & & & \\
W = & W_1 \arrow[r] & R_{01} \arrow[r] & R_{01} \arrow[dr] & & \\
 & W_2 \arrow[rr] & & R_{11} \arrow[r] & R_{02} \arrow[dr] & \\
 & W_3 \arrow[rrr] & & & R_{11} \arrow[r] & R_{03}
\end{tikzcd}$$

Multicore / Multisocket / Multirack / Multisite / Out-of-core: ?

Can choose reduction tree dynamically

TSQR Performance Results

- Parallel
  - Intel Clovertown
    - Up to **8x** speedup (8 core, dual socket, 10M x 10)
  - Pentium III cluster, Dolphin Interconnect, MPICH
    - Up to **6.7x** speedup (16 procs, 100K x 200)
  - BlueGene/L
    - Up to **4x** speedup (32 procs, 1M x 50)
  - Tesla C 2050 / Fermi
    - Up to **13x** (110,592 x 100)
  - Grid – **4x** on 4 cities (Dongarra et al)
  - Cloud – early result – up and running
- Sequential
  - "Infinite speedup" for out-of-Core on PowerPC laptop
    - As little as 2x slowdown vs (predicted) infinite DRAM
    - LAPACK with virtual memory never finished

Generalize to LU: How to Pivot?

Block Parallel Pivoting:

$$\begin{tikzcd}
 & W_0 \arrow[r] & U_{00} \arrow[dr] & & \\
W = & W_1 \arrow[r] & U_{10} \arrow[r] & U_{01} \arrow[dr] & \\
 & W_2 \arrow[r] & U_{20} \arrow[r] & U_{11} \arrow[r] & U_{02} \\
 & W_3 \arrow[r] & U_{30} \arrow[ur] & &
\end{tikzcd}$$

Block Pairwise Pivoting:

$$\begin{tikzcd}
 & W_0 \arrow[r] & U_{00} \arrow[dr] & & & \\
W = & W_1 \arrow[rr] & & U_{01} \arrow[dr] & & \\
 & W_2 \arrow[rrr] & & & U_{02} \arrow[dr] & \\
 & W_3 \arrow[rrrr] & & & & U_{03}
\end{tikzcd}$$

- Block Pairwise Pivoting used in PLASMA and FLAME
- Both can be much less numerically stable than partial pivoting
- Need a new idea…

CALU: Using similar idea for TSLU as TSQR: Use reduction tree, to do "Tournament Pivoting"

$$W^{nxb} = \begin{pmatrix} W_1 \\ \hline W_2 \\ \hline W_3 \\ \hline W_4 \end{pmatrix} = \begin{pmatrix} P_1 \cdot L_1 \cdot U_1 \\ \hline P_2 \cdot L_2 \cdot U_2 \\ \hline P_3 \cdot L_3 \cdot U_3 \\ \hline P_4 \cdot L_4 \cdot U_4 \end{pmatrix}$$

Choose b pivot rows of W_1, call them W_1'
Choose b pivot rows of W_2, call them W_2'
Choose b pivot rows of W_3, call them W_3'
Choose b pivot rows of W_4, call them W_4'

$$\begin{pmatrix} W_1' \\ W_2' \\ \hline W_3' \\ W_4' \end{pmatrix} = \begin{pmatrix} P_{12} \cdot L_{12} \cdot U_{12} \\ \hline P_{34} \cdot L_{34} \cdot U_{34} \end{pmatrix}$$

Choose b pivot rows, call them W_{12}'

Choose b pivot rows, call them W_{34}'

$$\begin{pmatrix} W_{12}' \\ W_{34}' \end{pmatrix} = P_{1234} \cdot L_{1234} \cdot U_{1234}$$

Choose b pivot rows

- Go back to W and use these b pivot rows (move them to top, do LU without pivoting)
- Repeat on each set of b columns to do CALU
- Provably stable [Xiang, LG, JD]

Performance vs ScaLAPACK

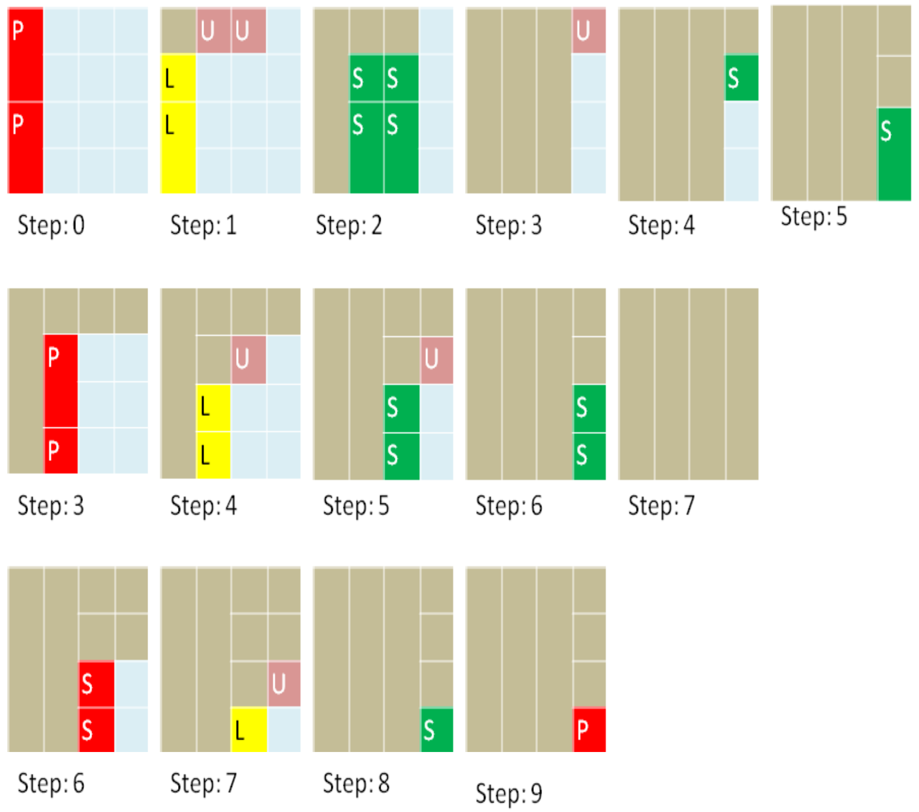
- Parallel TSLU (LU on tall-skinny matrix)
  - IBM Power 5
    - Up to **4.37x** faster (16 procs, 1M x 150)
  - Cray XT4
    - Up to **5.52x** faster (8 procs, 1M x 150)

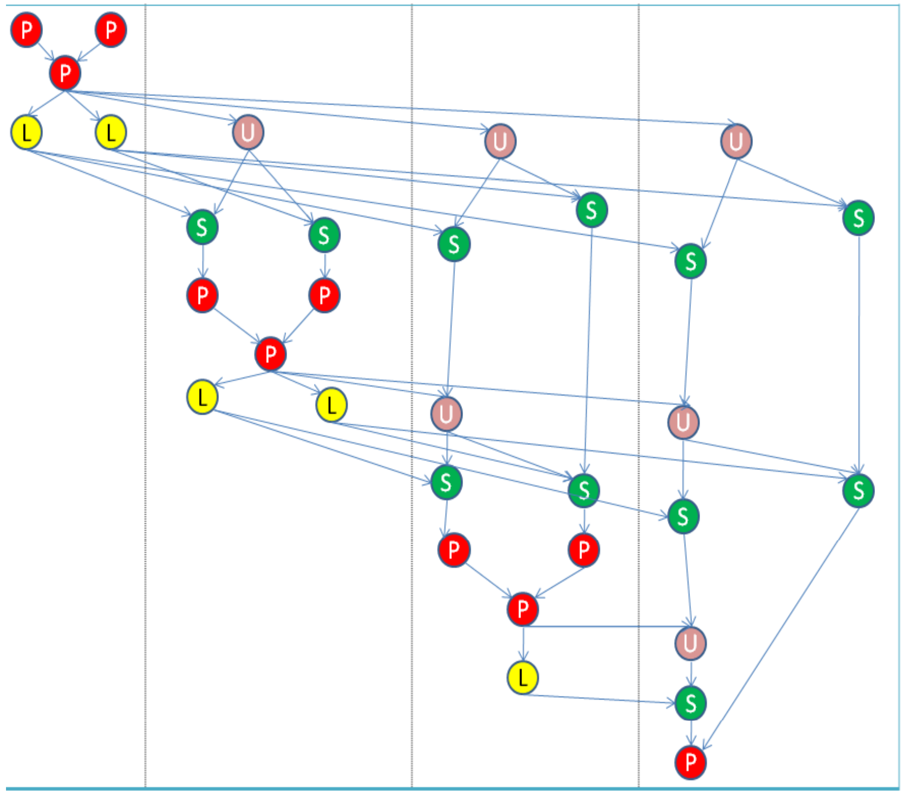
- Parallel CALU (LU on general matrices)
  - Intel Xeon (two socket, quad core)
    - Up to **2.3x** faster (8 cores, 10^6 x 500)
  - IBM Power 5
    - Up to **2.29x** faster (64 procs, 1000 x 1000)
  - Cray XT4
    - Up to **1.81x** faster (64 procs, 1000 x 1000)

- Details in SC08 (LG, JD, Xiang), IPDPS'10 (S. Donfack, LG)

CA(LU/QR) on multicore architectures

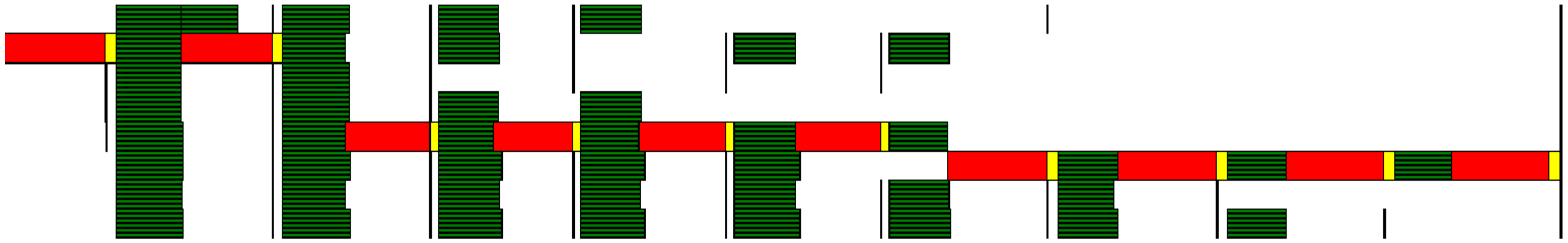
- The matrix is partitioned in blocks of size Tr x b.
- The computation of each block is associated with a task.
- The task dependency graph is scheduled using a dynamic scheduler.



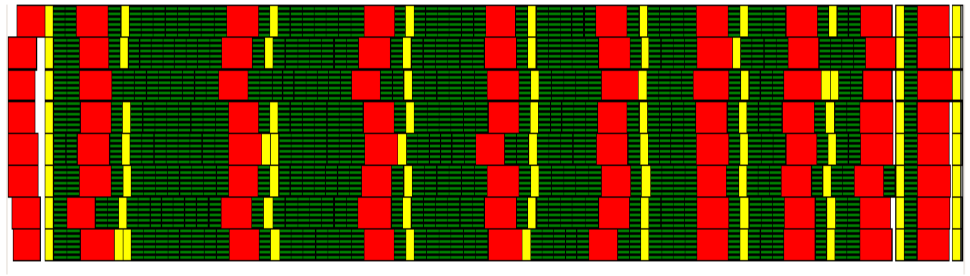


CA(LU/QR) on multicore architectures (contd)

The panel factorization stays on the critical path, but it is much faster. Exemple of execution on Intel 8 cores machine for a matrix of size 10^5 x 1000, with block size b = 100.

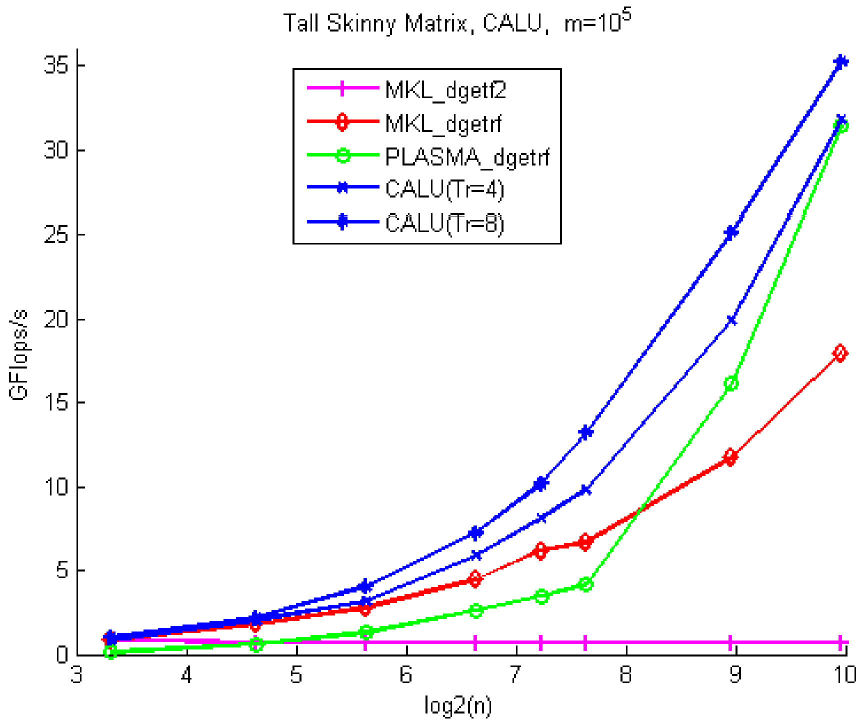


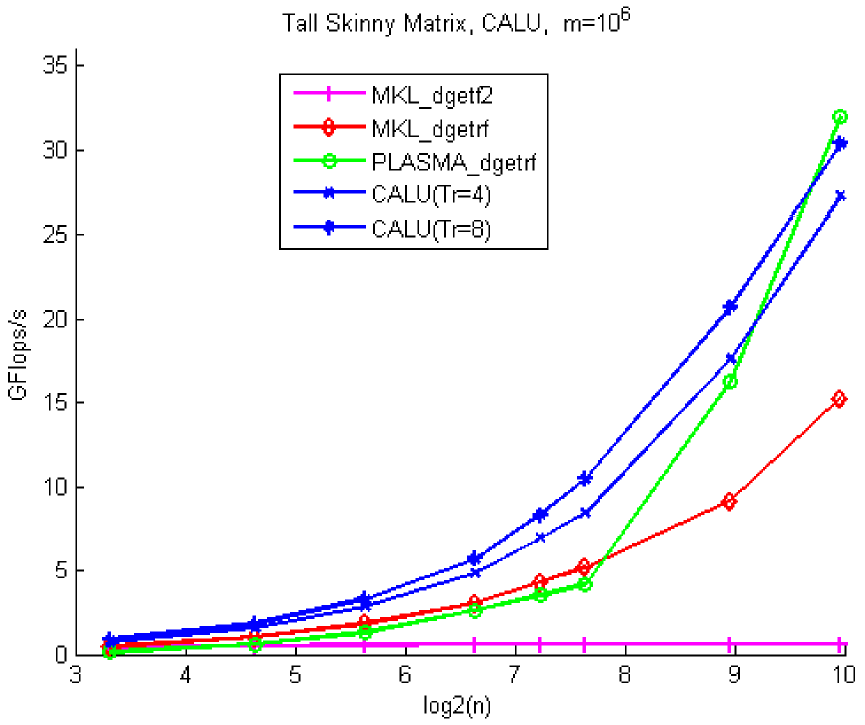
T=0 CALU: one thread computes the panel factorizaton



T=0 CALU: 8 threads compute the panel factorizaton

Performance of CALU on multicore architectures





• Results obtained on a two socket, quad core machine based on Intel Xeon EMT64 processor, and on a four socket, quad core machine based on AMD Opteron processor.

• Matrices of size m= 10^5 and n varies from 10 to 1000.

Courtesy of S. Donfack

Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors, memory per processor = $O(n^2 / P)$
- ScaLAPACK assumes best block size b chosen
- *Many* references (see reports), Green are ours
- Recall lower bounds:

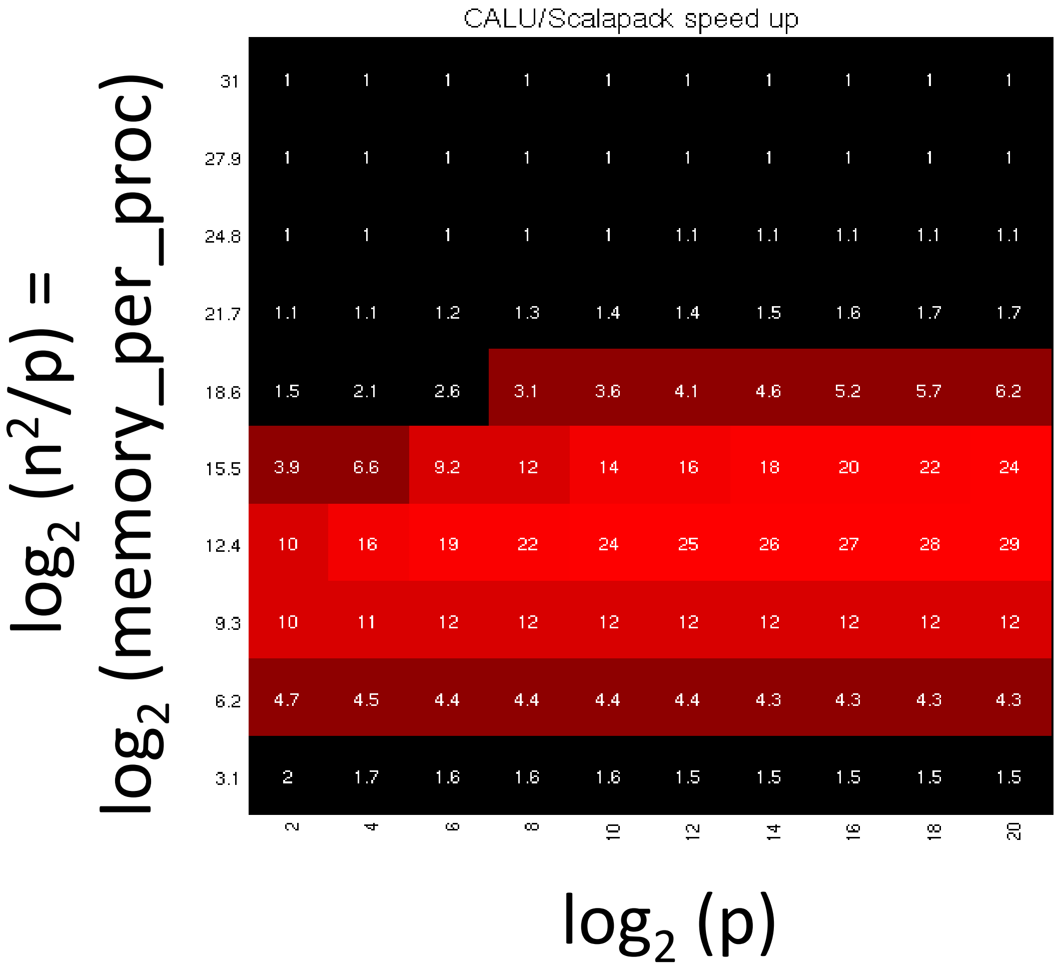
  #words_moved = $\Omega(n^2 / P^{1/2})$ and #messages = $\Omega(P^{1/2})$

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix multiply	[Cannon, 69]		
Cholesky	ScaLAPACK		
LU	[GDX08] ScaLAPACK		
QR	[DGHL08] ScaLAPACK		
Sym Eig, SVD	[BDD10] ScaLAPACK		
Nonsym Eig	[BDD10] ScaLAPACK		

Exascale Machine Parameters

- $2^{30} \approx 1{,}000{,}000$ nodes
- 1024 cores/node   (a billion cores!)
- 100 GB/sec interconnect bandwidth
- 400 GB/sec DRAM bandwidth
- 1 microsec interconnect latency
- 50 nanosec memory latency
- 32 Petabytes of memory
- 1/2 GB total L1 on a node

Exascale predicted speedups for CA-LU vs ScaLAPACK-LU



	2	4	6	8	10	12	14	16	18	20
31	1	1	1	1	1	1	1	1	1	1
27.9	1	1	1	1	1	1	1	1	1	1
24.8	1	1	1	1	1	1.1	1.1	1.1	1.1	1.1
21.7	1.1	1.1	1.2	1.3	1.4	1.4	1.5	1.6	1.7	1.7
18.6	1.5	2.1	2.6	3.1	3.6	4.1	4.6	5.2	5.7	6.2
15.5	3.9	6.6	9.2	12	14	16	18	20	22	24
12.4	10	16	19	22	24	25	26	27	28	29
9.3	10	11	12	12	12	12	12	12	12	12
6.2	4.7	4.5	4.4	4.4	4.4	4.4	4.3	4.3	4.3	4.3
3.1	2	1.7	1.6	1.6	1.6	1.5	1.5	1.5	1.5	1.5

Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors, memory per processor = $O(n^2 / P)$
- ScaLAPACK assumes best block size b chosen
- *Many* references (see reports), Green are ours
- Recall lower bounds:

  #words_moved = $\Omega(n^2 / P^{1/2})$ and #messages = $\Omega(P^{1/2})$

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix multiply	[Cannon, 69]	1	1
Cholesky	ScaLAPACK	log P	log P
LU	[GDX08] ScaLAPACK	log P log P	log P $(N / P^{1/2}) \cdot \log P$
QR	[DGHL08] ScaLAPACK	log P log P	$\log^3 P$ $(N / P^{1/2}) \cdot \log P$
Sym Eig, SVD	[BDD10] ScaLAPACK	log P log P	$\log^3 P$ $N / P^{1/2}$
Nonsym Eig	[BDD10] ScaLAPACK	log P $P^{1/2} \cdot \log P$	$\log^3 P$ $N \cdot \log P$

Can we do better?

Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors, **memory per processor = $O(n^2 / P)$? Why?**
- ScaLAPACK assumes best block size b chosen
- *Many* references (see reports), Green are ours
- Recall lower bounds:

  #words_moved = $\Omega(n^2 / P^{1/2})$ and #messages = $\Omega(P^{1/2})$

Algorithm	Reference	Factor exceeding lower bound for #words_moved	Factor exceeding lower bound for #messages
Matrix multiply	[Cannon, 69]	1	1
Cholesky	ScaLAPACK	log P	log P
LU	[GDX08] ScaLAPACK	log P log P	log P $(N / P^{1/2}) \cdot \log P$
QR	[DGHL08] ScaLAPACK	log P log P	$\log^3 P$ $(N / P^{1/2}) \cdot \log P$
Sym Eig, SVD	[BDD10] ScaLAPACK	log P log P	$\log^3 P$ $N / P^{1/2}$
Nonsym Eig	[BDD10] ScaLAPACK	log P $P^{1/2} \cdot \log P$	$\log^3 P$ $N \cdot \log P$

Can we do better?

Beating #words_moved = $\Omega(n^2/P^{1/2})$

- #words_moved = $\Omega((n^3/P)/\text{local\_mem}^{1/2})$
- If one copy of data, local_mem = n^2/P
- Can we use more memory to communicate less?
- "3D" Matmul Algorithm on $P^{1/3} \times P^{1/3} \times P^{1/3}$ processor grid
  - $P^{1/3}$ redundant copies of A and B
  - Reduces communication volume to $O((n^2/P^{2/3}) \log(P))$
    - optimal for $P^{1/3}$ copies
  - Reduces number of messages to $O(\log(P))$ – also optimal
- "2.5D" Algorithms
  - Extends to $1 \le c \le P^{1/3}$ copies on $(P/c)^{1/2} \times (P/c)^{1/2} \times c$ grid
  - Reduces communication volume of Matmul, LU, by $c^{1/2}$
  - Reduces comm 92% on 64K proc BG-P, LU&MM speedup 2.6x

Summary of Direct Linear Algebra

- New lower bounds, optimal algorithms, big speedups in theory and practice
- Lots of other progress, open problems
  - New ways to “pivot”
  - Extensions to Strassen-like algorithms
  - Heterogeneous architectures
  - Some sparse algorithms
  - Autotuning. . .

Avoiding Communication in Iterative Linear Algebra

- k-steps of iterative solver for sparse Ax=b or Ax=λx
  - Does k SpMVs with A and starting vector
  - Many such “Krylov Subspace Methods”
    - Conjugate Gradients (CG), GMRES, Lanczos, Arnoldi, …
- Goal: minimize communication
  - Assume matrix “well-partitioned”
  - Serial implementation
    - Conventional: O(k) moves of data from slow to fast memory
    - **New**: **O(1) moves of data – optimal**
  - Parallel implementation on p processors
    - Conventional: O(k log p) messages (k SpMV calls, dot prods)
    - **New: O(log p) messages - optimal**
- Lots of speed up possible (modeled and measured)
  - Price: some redundant computation

Minimizing Communication of GMRES to solve Ax=b

- GMRES: find x in span{b,Ab,…,A^kb} minimizing || Ax-b ||$_2$

Standard GMRES

```
for i=1 to k
   w = A · v(i-1)   … *SpMV*
   MGS(w, v(0),…,v(i-1))
   update v(i), H
endfor
solve LSQ problem with H
```

Communication-avoiding GMRES

```
W = [ v, Av, A²v, … , A^k v ]
[Q,R] = TSQR(W)
      … "Tall Skinny QR"
build H from R
solve LSQ problem with H
```

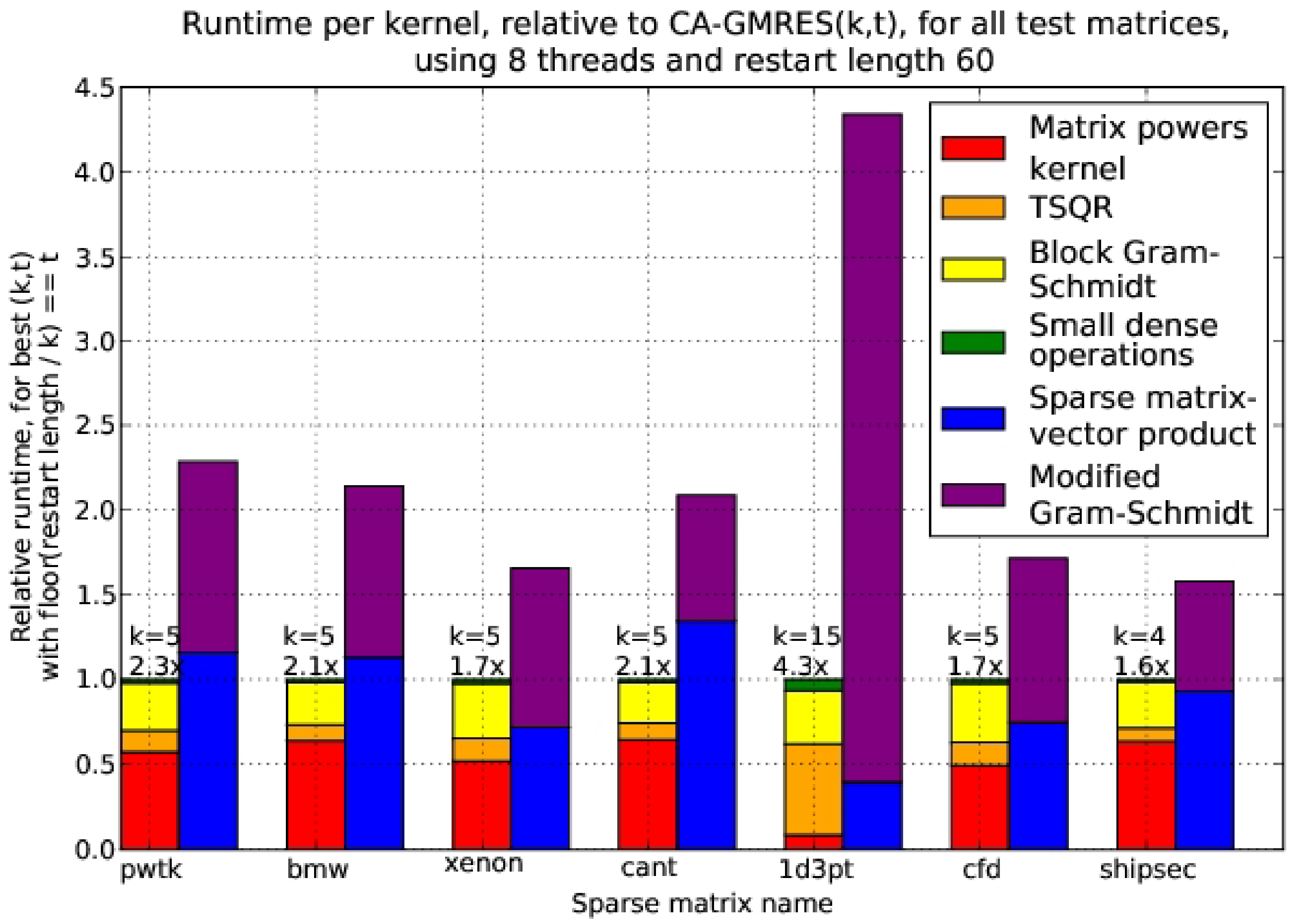
Sequential case: #words moved decreases by a factor of k
Parallel case: #messages decreases by a factor of k

- Oops – W from power method, precision lost!
- Need different polynomials than A^k for stability

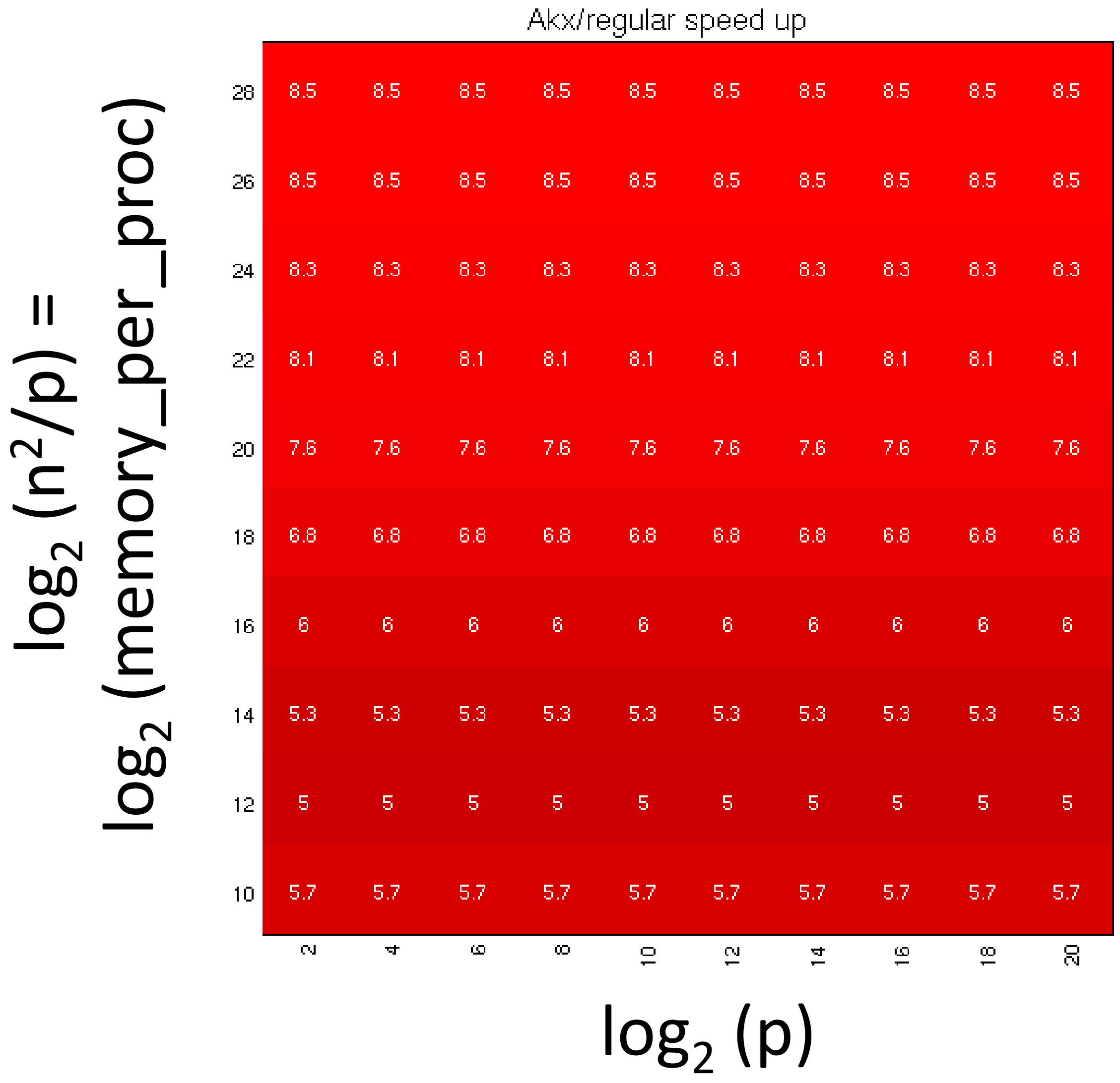
Speed ups of GMRES on 8-core Intel Clovertown

Requires Co-tuning Kernels

[MHDY09]



Exascale predicted speedups for Matrix Powers Kernel over SpMV for 2D Poisson (5 point stencil)



Summary of Iterative Linear Algebra

- New Lower bounds, optimal algorithms, big speedups in theory and practice
- Lots of other progress, open problems
  - GMRES, CG, BiCGStab, Arnoldi, Lanczos reorganized
  - Other Krylov methods?
  - Recognizing stable variants more easily?
  - Avoiding communication with preconditioning harder
    - “Hierarchically semi-separable” preconditioners work
  - Autotuning

For further information

- www.cs.berkeley.edu/~demmel
- www-rocq.inria.fr/who/Laura.Grigori
- Papers
  - bebop.cs.berkeley.edu
  - www-rocq.inria.fr/who/Laura.Grigori/COALA2010/coala.html
  - www.netlib.org/lapack/lawns
- 1-week-short course – slides and video
  - www.ba.cnr.it/ISSNLA2010
- Google “parallel computing course”

Summary

Time to redesign all linear algebra algorithms and software

And eventually all of applied mathematics…

Don’t Communic…