The Past, Present and *Future* of High Performance Numerical Linear Algebra: *Minimizing Communication*

Jim Demmel

EECS & Math Departments

UC Berkeley
The Past, the Present ...

- LAPACK – sequential/shared memory parallel dense linear algebra library
The Past, the Present ...

- LAPACK contributors
The Past, the Present ...

- **LAPACK** – seq/shared mem dense linear algebra
- **ScaLAPACK** – distributed memory parallel dense linear algebra library
  - Jack Dongarra, Susan Blackford, Jaeyoung Choi, Andy Cleary, Ed D’Azevedo, Inderjit Dhillon, Sven Hammarling, Greg Henry, Antoine Petitet, Ken Stanley, David Walker, Clint Whaley, and many other contributors
The Past, the Present ...

- LAPACK – seq/shared mem dense linear algebra
- ScaLAPACK – dist mem dense linear algebra
- SuperLU – sparse direct solver for $Ax=b$
  – Xiaoye Sherry Li
The Past, the Present ...

• LAPACK – seq/shared mem dense linear algebra
• ScaLAPACK – dist mem dense linear algebra
• SuperLU – sparse direct solver for Ax=b
• Autotuning
  – PhiPAC for matrix multiplication
    • Jeff Bilmes, Krste Asanovic, Rich Vuduc,
      Sriram Iyer, CheeWhye Chin, Dominic Lam
  
  – OSKI for sparse-matrix-vector multiplication (SpMV)
    • Rich Vuduc, Kathy Yelick, Rajesh Nishtala,
      Ben Lee, Shoaib Kamil, Jen Hsu
The Past, the Present ...

- LAPACK – seq/shared mem dense linear algebra
- ScaLAPACK – dist mem dense linear algebra
- SuperLU – sparse direct solver for Ax=b
- Autotuning – matmul and SpMV
- Prometheus – parallel unstructured FE solver
  - Mark Adams
  - Gordon Bell Prize, 2004
... The Future

• 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

- Time_per_flop << 1/ bandwidth << latency

- Gaps growing exponentially with time (FOSC, 2004)

<table>
<thead>
<tr>
<th>Annual improvements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time_per_flop</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>Network</td>
</tr>
<tr>
<td>DRAM</td>
</tr>
</tbody>
</table>

- 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
Direct linear algebra: Prior Work on Matmul

- Assume \( n^3 \) algorithm for \( C = A \times B \) (i.e. not Strassen-like)
- Sequential case, with fast memory of size \( M \)
  - Lower bound on \# words moved to/from slow memory
    \[ = \Omega \left( \frac{n^3}{M^{1/2}} \right) \] [Hong, Kung, 81]
  - Attained using “blocked” or cache-oblivious algorithms

- Parallel case on \( P \) processors:
  - Assume load balanced, one copy each of \( A, B, C \)
  - Lower bound on \# words communicated
    \[ = \Omega \left( \frac{n^2}{P^{1/2}} \right) \] [Irony, Tiskin, Toledo, 04]
  - Attained by Cannon’s Algorithm
Lower bound for all “direct” linear algebra

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

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

$$
\#\text{messages\_sent (per processor)} = \Omega(\#\text{flops (per processor)} / M^{3/2})
$$

• Parallel case: assume either load or memory balanced

• Holds for
  – 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 (eg Cholesky)
### One-sided Factorizations (LU, QR), so far

<table>
<thead>
<tr>
<th>Classical Approach</th>
<th>Blocked Approach (LAPACK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>for i=1 to n</td>
<td>for i=1 to n/b</td>
</tr>
<tr>
<td>update column i</td>
<td>update block i of b columns</td>
</tr>
<tr>
<td>update trailing matrix</td>
<td>update trailing matrix</td>
</tr>
<tr>
<td>#words Moved = O(n^3)</td>
<td>#words moved = O(n^3/M^{1/3})</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Recursive Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>func factor(A)</code></td>
</tr>
<tr>
<td>if A has 1 column, update it</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>factor(left half of A)</td>
</tr>
<tr>
<td>update right half of A</td>
</tr>
<tr>
<td>factor(right half of A)</td>
</tr>
<tr>
<td>#words moved = O(n^3/M^{1/2})</td>
</tr>
</tbody>
</table>

- None of these approaches minimizes #messages
- Need another idea
TSQR: QR of a Tall, Skinny matrix

\[
W = \begin{pmatrix}
W_0 \\
W_1 \\
W_2 \\
W_3
\end{pmatrix}
\]

\[
\begin{pmatrix}
R_{00} \\
R_{10} \\
R_{20} \\
R_{30}
\end{pmatrix} = \begin{pmatrix}
Q_{01} & R_{01} \\
Q_{11} & R_{11}
\end{pmatrix}
\]

\[
\begin{pmatrix}
R_{01} \\
R_{11}
\end{pmatrix} = \begin{pmatrix}
Q_{02} & R_{02}
\end{pmatrix}
\]
TSQR: QR of a Tall, Skinny matrix

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

\[
\begin{pmatrix}
R_{00} \\
R_{10} \\
R_{20} \\
R_{30}
\end{pmatrix} = \begin{pmatrix}
Q_{01} & R_{01} \\
Q_{11} & R_{11}
\end{pmatrix} = \begin{pmatrix}
Q_{01} \\
Q_{11}
\end{pmatrix} \cdot \begin{pmatrix}
R_{01} \\
R_{11}
\end{pmatrix}
\]

\[
\begin{pmatrix}
R_{01} \\
R_{11}
\end{pmatrix} = \begin{pmatrix}
Q_{02} & R_{02}
\end{pmatrix}
\]
Minimizing Communication in TSQR

Parallel: \[ W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow R_{00} \rightarrow R_{10} \rightarrow R_{01} \rightarrow R_{02} \]

Sequential: \[ W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow R_{00} \rightarrow R_{01} \rightarrow R_{02} \rightarrow R_{03} \]

Dual Core: \[ W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow R_{00} \rightarrow R_{11} \rightarrow R_{01} \rightarrow R_{02} \rightarrow R_{11} \rightarrow R_{03} \]

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)
  – Grid – 4x on 4 cities (Dongarra et al)
  – Cloud – early result – up and running using Mesos

• 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

• What about QR for a general matrix?

Data from Grey Ballard, Mark Hoemmen, Laura Grigori, Julien Langou, Jack Dongarra, Michael Anderson
CAQR on a GPU (Fermi C2050) (1/2)

# rows = 8192

Performance vs. Matrix Width (SGEQRF + SORGQR)

GFLOPS/Sec.

Number of Columns:

- MKL (8 cores)
- BLAS2 (C2050)
- CULA 2.1 (C2050)
- CAQR (C2050)
CAQR on a GPU (Fermi C2050) (2/2)

#rows = 8192

Biggest speedup over MAGMA QR is 13x for 1M x 192
Partial History of dense *sequential* QR algorithms attaining communication lower bounds

- Algorithms shown minimizing # Messages assume (recursive) block layout
- *Many* references (see reports), only some shown, plus ours
  - Oldest reference may or may not include analysis
- Cache-oblivious are *underlined*, Green are ours, ? is unknown/future work
- $b =$ block size, $M =$ fast memory size, $n =$ dimension

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>2 Levels of Memory</th>
<th>Multiple Levels of Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Words Moved and # Messages</td>
<td>#Words Moved and # Messages</td>
</tr>
<tr>
<td>QR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CALU: Using similar idea for TSLU as TSQR: Use reduction tree, to do “Tournament Pivoting”

\[
W_{\text{nx}b} = \begin{pmatrix}
W_1 \\
W_2 \\
W_3 \\
W_4
\end{pmatrix} = \begin{pmatrix}
P_1 \cdot L_1 \cdot U_1 \\
P_2 \cdot L_2 \cdot U_2 \\
P_3 \cdot L_3 \cdot U_3 \\
P_4 \cdot L_4 \cdot U_4
\end{pmatrix}
\]

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 \(W_4'\)

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

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
Stability of CALU using Tournament Pivoting

• Thm: Tournament Pivoting (TP) as stable as Partial Pivoting (PP) in following sense: TP gets same results as PP applied to different input matrix whose entries are blocks taken from input A

• Worst case pivot growth factor on n x n matrix
  – Proven: $2^{nH-1}$ where $H = 1 + \text{height of reduction tree}$
  – Attained: $2^{n-1}$, same as PP

• There are examples where PP is exponentially less stable than TP, and vice-versa, so neither is always better than the other

• Extensive numerical testing confirms stability
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

\[ \log_2 \left( \frac{n^2}{p} \right) = \log_2 \left( \frac{\text{memory per proc}}{n} \right) \]
History of Spectral Divide-and-Conquer for Eigenvalue problems and the SVD

• Ideas go back to Bulgakov, Godunov, Malyshev [BG88, Mal89]
• Bai, D., Gu [BDG97]
  – Reduced to matmul, QR, generalized QR with pivoting (bug)
• D., Dumitriu, Holtz [DDH07]
  – Uses RURV (randomized URV), not pivoted QR (same bug)
• D., Grigori, Hoemmen, Langou [DGHL08]
  – Communication-avoiding QR (CAQR)
• New Communication-avoiding algorithm [BDD10]
  – Use generalized RURV for better rank-detection (fixes bug)
  – Uses communication-avoiding matmul and QR
  – Uses randomization in divide-and-conquer
  – Ballard, D., Dumitriu
Overview of Spectral Divide-and-Conquer

• Implicitly compute $(I + A^{-2^k})^{-1}$
  – Maps spectrum to 0 and 1 (as $k \to \infty$)
  – Converges to spectral projector of $A$ for $|\lambda| > 1$
  – Rank Revealing QR yields invariant subspace

\[
A_{\text{new}} = Q*AQ = \begin{pmatrix}
A_{11} & A_{12} \\
\varepsilon & A_{22}
\end{pmatrix}
\]

• Algorithm:

$$A_0 = A, \quad B_0 = I$$
Repeat

$$\begin{pmatrix} Q_{11} & Q_{12} \\ Q_{12} & Q_{22} \end{pmatrix} \cdot \begin{pmatrix} R_k \\ 0 \end{pmatrix} = \text{qr} \begin{pmatrix} B_k \\ A_k \end{pmatrix}$$

$$A_{k+1} = Q_{12} \ast \cdot A_k$$

$$B_{k+1} = Q_{22} \ast \cdot B_k$$
until $R_k$ converges

• Fact: $A_k^{-1} B_k = A^{-2^k}$
• So need RRQR of

$$(I + A^{-2^k})^{-1} = (A_k + B_k)^{-1} A_k$$
without forming $(A_k + B_k)^{-1} A_k$
Rank Revealing CA-Decompositions

• Randomized URV of $A$
  • $[V,R'] = \text{caqr}(\text{randn}(n,n))$ ... $V$ random orthogonal
  • $[U,R] = \text{caqr}(A\cdot V^T)$
  • Thm: $A = URV$ reveals the rank of $A$ with high probability

• Randomized URV of $(A_k + B_k)^{-1} A_k$
  • $[V,R'] = \text{caqr}(\text{randn}(n,n))$ ... $V$ random orthogonal
  • $[Q_1,R_1] = \text{caqr}(A_k\cdot V^T)$
  • $[R_2,U] = \text{carq}(Q_1^T\cdot (A_k + B_k))$
  • Thm: $(A_k + B_k)^{-1} A_k = U^T(R_2^{-1}R_1)V$ reveals rank with high probability
  • Only need $U$, not $R_2^{-1}R_1$
  • Applies to $M_1^{\pm 1} \cdot M_2^{\pm 1} \cdot M_3^{\pm 1} \cdot \cdots = URV$

• QR with Tournament Pivoting
  – Emulate column pivoting by using a “tournament” to pick next group of $b$ columns
  – Only applies to single matrix, not $(A_k + B_k)^{-1} A_k$
Splitting the spectrum in different places

• Splitting the spectrum along unit circle not enough
• Nonsymmetric case
  – Moebius transformation $z \to (\alpha z + \beta)/(\gamma z + \delta)$ maps unit circle to any line or circle in complex plane
  – Apply splitting algorithm to $(A_0, B_0) = (\alpha A + \beta I, \gamma A + \delta I)$ instead of $(A, I)$, to split along any desired line or circle
• Symmetric case - easier
  – Just shift $A$ to $A - \sigma I$
  – SVD analogous
Randomized Spectral Divide-and-Conquer

- Pick random place to divide spectrum
- Two progress metrics
  - Spectrum splits, with $f \cdot n$ and $(1-f) \cdot n$ evals per part, with $0 < f < 1$
  - Region splits, with fractions $f$ and $1-f$ of area in each
  - Progress guaranteed with high probability; always stable
- SymEig and SVD: random point near center of spectrum
- NonsymEig: random line/circle near “center” of spectrum
Successive Band Reduction

• Works just for eig($A=A^T$) or SVD
  – Bischof, Lang Sun

• Step 1: Reduce Dense $\rightarrow$ Narrow Band
  – If bandwidth $\approx M^{1/2}$, can minimize communication

• Step 2: Reduce Narrow Band $\rightarrow$ Tridiagonal
  – Trickier...
Successive Band Reduction
Successive Band Reduction
Successive Band Reduction
Successive Band Reduction
Successive Band Reduction
Successive Band Reduction
Successive Band Reduction
Successive Band Reduction

\[ b+1 \]

\[ d+1 \]

\[ d+c \]

\[ Q_1^T \]

\[ Q_2^T \]

\[ Q_3^T \]
Successive Band Reduction
Successive Band Reduction
Successive Band Reduction

To minimize communication (when \( n > M^{1/2} \)) after reducing to bandwidth \( b = M^{1/2} \):

if \( n > M \)

Use \( d = b - 1 \) and \( c = 1 \)

else if \( M^{1/2} < n \leq M \)

Pass 1: Use \( d = b - \frac{M}{4n} \) and \( c = \frac{M}{4n} \)

Pass 2: Use \( d = \frac{M}{4n} - 1 \) and \( c = 1 \)
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:
  \[ \#\text{words}_\text{moved} = \Omega\left( \frac{n^2}{P^{1/2}} \right) \]  and  \[ \#\text{messages} = \Omega\left( P^{1/2} \right) \]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Reference</th>
<th>Factor exceeding lower bound for #words_moved</th>
<th>Factor exceeding lower bound for #messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix multiply</td>
<td>[Cannon, 69]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cholesky</td>
<td>ScaLAPACK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LU</td>
<td>[GDX08]</td>
<td>[GDX08]</td>
<td>[GDX08]</td>
</tr>
<tr>
<td>QR</td>
<td>[DCHL08]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sym Eig, SVD</td>
<td>[BDD10]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonsym Eig</td>
<td>[BDD10]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:
  \[
  \#\text{words}_\text{moved} = \Omega\left( \frac{n^2}{P^{1/2}} \right) \quad \text{and} \quad \#\text{messages} = \Omega\left( P^{1/2} \right)
  \]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Reference</th>
<th>Factor exceeding lower bound for #\text{words}_\text{moved}</th>
<th>Factor exceeding lower bound for #\text{messages}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix multiply</td>
<td>[Cannon, 69]</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Cholesky</td>
<td>ScaLAPACK</td>
<td>(\log P)</td>
<td>(\log P)</td>
</tr>
<tr>
<td>LU</td>
<td>[GDX08], ScaLAPACK</td>
<td>(\log P)</td>
<td>(\log P) (\left( \frac{N}{P^{1/2}} \right) \cdot \log P)</td>
</tr>
<tr>
<td>QR</td>
<td>[DCHL08], ScaLAPACK</td>
<td>(\log P)</td>
<td>(\log^3 P) (\left( \frac{N}{P^{1/2}} \right) \cdot \log P)</td>
</tr>
<tr>
<td>Sym Eig, SVD</td>
<td>[BDD10], ScaLAPACK</td>
<td>(\log P)</td>
<td>(\log^3 P) (\frac{N}{P^{1/2}})</td>
</tr>
<tr>
<td>Nonsym Eig</td>
<td>[BDD10], ScaLAPACK</td>
<td>(\log P) (P^{1/2} \cdot \log P)</td>
<td>(\log^3 P) (\frac{N \cdot \log P}{P^{1/2}})</td>
</tr>
</tbody>
</table>

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?
- Johnson’s Matmul Algorithm on \( P^{1/3} \times P^{1/3} \times P^{1/3} \) processor grid
  - Broadcast A in j direction (\( P^{1/3} \) redundant copies)
  - Broadcast B in i direction (ditto)
  - Local multiplies
  - Reduce (sum) in k direction

- Communication volume
  = \( O((n/P^{1/3})^2 \log(P)) \) - optimal
- Number of messages
  = \( O(\log(P)) \) – optimal

- Extends to \( 1 \leq c \leq P^{1/3} \) copies
- What about LU, etc?

\[
\text{Cube representing C face}
\]

\[
\text{“A face”}
\]

\[
\text{“B face”}
\]
Predicted Exascale
2.5D Matmul - Speedup over Cannon

\[ \log_2 \left( \frac{n^2}{p} \right) = \log_2 \left( \frac{\text{memory\_per\_proc}}{p} \right) \]
Exascale 2.5D Matmul – optimal # copies

\[
\log_2 \left( \frac{n^2}{p} \right) = \log_2 \left( \frac{\text{memory}_\text{per_proc}}{p} \right)
\]
Strassen-like Algorithms

- $M =$ fast memory size
- Conventional matmul
  - $\#\text{flops} = 2n^3$
  - $\#\text{words}_\text{moved} = \Omega(n^3/M^{1/2}) = \Omega(M(n/M^{1/2})^3)$, attainable
- Strassen matmul
  - $\#\text{flops} = \Theta(n^\omega)$ where $\omega = \log_2(7) \approx 2.81$
  - $\#\text{words}_\text{moved} = \Omega(M(n/M^{1/2})^\omega)$, attainable too
- Applies to other sufficiently “Strassen-like“ algorithms
  - How broadly does it apply?
- We know rest of linear algebra can be done in $O(n^\omega)$ flops
  - Also with $\#\text{words}_\text{moved} = \Omega(M(n/M^{1/2})^\omega)$ (sequential)
  - Lower bounds?
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”
  – Heterogeneous architectures
  – Some sparse algorithms
  – Autotuning. . .
Avoiding Communication in Iterative Linear Algebra

- \( k \)-steps of iterative solver for sparse \( Ax=b \) or \( Ax=\lambda x \)
  - 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
Communication Avoiding Kernels:
The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)

\[
\begin{align*}
A^3x & \cdot \cdot \cdot \\
A^2x & \cdot \cdot \cdot \\
A\cdot x & \cdot \cdot \cdot \\
x & \cdot \cdot \cdot \\
1 & 2 & 3 & 4 & \ldots & \ldots & 32
\end{align*}
\]

- Example: \(A\) tridiagonal, \(n=32\), \(k=3\)
- Works for any “well-partitioned” \(A\)
Communication Avoiding Kernels:  
The Matrix Powers Kernel : \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)

- Example: A tridiagonal, \(n=32, k=3\)
Communication Avoiding Kernels:

The Matrix Powers Kernel: \([Ax, A^2x, ..., A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, ..., A^kx]\)

**Example:** A tridiagonal, \(n=32\), \(k=3\)
Communication Avoiding Kernels:
The Matrix Powers Kernel : \([Ax, A^2x, ..., A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, ..., A^kx]\)

- Example: A tridiagonal, \(n=32\), \(k=3\)
Communication Avoiding Kernels:
The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)

- Example: A tridiagonal, \(n=32\), \(k=3\)
Communication Avoiding Kernels: The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)

- Example: A tridiagonal, \(n=32\), \(k=3\)
Communication Avoiding Kernels:
The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)
- Sequential Algorithm
- Example: A tridiagonal, \(n=32, k=3\)
Communication Avoiding Kernels:
The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)
- Sequential Algorithm

Example: \(A\) tridiagonal, \(n=32\), \(k=3\)
Communication Avoiding Kernels:
The Matrix Powers Kernel: \([Ax, A^2x, ..., A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, ..., A^kx]\)

- Sequential Algorithm

- Example: A tridiagonal, \(n=32, k=3\)
Communication Avoiding Kernels:
The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)

- **Sequential Algorithm**

- **Example:** \(A\) tridiagonal, \(n=32\), \(k=3\)
Communication Avoiding Kernels:
The Matrix Powers Kernel: \([Ax, A^2x, ..., A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, ..., A^kx]\)

- Parallel Algorithm

- Example: A tridiagonal, \(n=32, k=3\)
- Each processor works on (overlapping) trapezoid
Communication Avoiding Kernels:
The Matrix Powers Kernel: $[Ax, A^2x, ..., A^kx]$

Remotely Dependent Entries for $[x, Ax, A^2x, A^3x]$, A irregular, multiple processors
Minimizing Communication of GMRES to solve $Ax=b$

- GMRES: find $x$ in $\text{span}\{b, Ab, \ldots, A^k b\}$ minimizing $||Ax-b||_2$

**Standard GMRES**

for $i=1$ to $k$

- $w = A \cdot v(i-1)$ \hspace{1cm} $\ldots$ SpMV
- MGS($w, v(0), \ldots, v(i-1)$)
- update $v(i)$, $H$

endfor

solve LSQ problem with $H$

**Communication-avoiding GMRES**

$$W = [v, Av, A^2v, \ldots, A^k v]$$

$$[Q, R] = \text{TSQR}(W)$$

$\ldots$ “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!
“Monomial” basis $[Ax,\ldots,A^kx]$ fails to converge

Different polynomial basis $[p_1(A)x,\ldots,p_k(A)x]$ does converge
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)

\[ \log_2 \left( \frac{n^2}{p} \right) = \log_2 \left( \frac{\text{memory}_{\text{per}}}{\text{proc}} \right) \]
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

• Papers
  – bebop.cs.berkeley.edu
  – www.netlib.org/lapack/lawns

• 1-week-short course – slides and video
  – www.ba.cnr.it/ISSNLA2010
Collaborators and Supporters

• Collaborators
  – Michael Anderson (UCB), Grey Ballard (UCB), Erin Carson (UCB), Jack Dongarra (UTK), Ioana Dumitriu (U. Wash), Laura Grigori (INRIA), Ming Gu (UCB), Mark Hoemmen (Sandia NL), Olga Holtz (UCB & TU Berlin), Nick Knight (UCB), Julien Langou, (U Colo. Denver), Marghoob Mohiyuddin (UCB), Oded Schwartz (TU Berlin), Edgar Solomonik (UCB), Michelle Strout (Colo. SU), Vasily Volkov (UCB), Sam Williams (LBNL), Hua Xiang (INRIA), Kathy Yelick (UCB & LBNL)
  – Other members of the ParLab, BEBOP, CACHE, EASI, MAGMA, PLASMA, TOPS projects

• Supporters
  – NSF, DOE, UC Discovery
  – Intel, Microsoft, Mathworks, National Instruments, NEC, Nokia, NVIDIA, Samsung, Sun
We’re hiring!

• Seeking software engineers to help develop the next versions of LAPACK and ScaLAPACK
• Locations: UC Berkeley and UC Denver
• Please send applications to julie@cs.utk.edu
Summary

Time to redesign all linear algebra algorithms and software

Don’t Communic...