Avoiding Communication in Linear Algebra

Grey Ballard

UC Berkeley

University of Virginia
April 5, 2013
Let’s start with matrix multiplication

Suppose we want to compute

\[ A \cdot B = C \]

where \( A \) and \( B \) are \( n \times n \) matrices

\[
\sum_{k=1}^{n} a_{ik} b_{kj} = c_{ij}
\]
Two algorithms for matrix multiplication...

We can multiply matrices like this ("matrix-vector" algorithm):

\[
\begin{array}{c}
A \\
\end{array}
\begin{array}{c}
B \\
\end{array}
= 
\begin{array}{c}
C \\
\end{array}
\]

or like this ("blocked" algorithm):

\[
\begin{array}{c}
A \\
\end{array}
\begin{array}{c}
B \\
\end{array}
= 
\begin{array}{c}
C \\
\end{array}
\]

In both cases, we do \(2n^3 + O(n^2)\) flops
Same computation, different performance

![Graph showing performance (GFLOPS) vs. matrix dimension with two lines for Blocked Algorithm and Mat−Vec Algorithm. The Blocked Algorithm line starts lower and remains higher than the Mat−Vec Algorithm line throughout the range of matrix dimensions shown.](image-url)
We must consider communication

By *communication*, I mean
- moving data within memory hierarchy on a sequential computer
- moving data between processors on a parallel computer

For high-level analysis, we’ll use these simple memory models:
Runtime Model

Measure computation in terms of \# flops performed

Time per flop: $\gamma$

Measure communication in terms of \# words communicated

Time per word: $\beta$

Total running time of an algorithm (ignoring overlap):

$\gamma \cdot (\# \text{ flops}) + \beta \cdot (\# \text{ words})$

$\beta \gg \gamma$ as measured in time and energy, and the relative cost of communication is increasing
Why avoid communication

Annual Improvements in Time

<table>
<thead>
<tr>
<th>Flop rate</th>
<th>DRAM Bandwidth</th>
<th>Network Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>$\beta$</td>
<td>$\beta$</td>
</tr>
<tr>
<td>59% per year</td>
<td>23% per year</td>
<td>26% per year</td>
</tr>
</tbody>
</table>

Energy cost comparisons

Source: John Shalf
Let $M$ be the size of the fast memory.

The blocked algorithm uses a block size of $\sqrt{M/3}$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Computation</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mat-Vec Algorithm</td>
<td>$O(n^3)$</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>Blocked Algorithm</td>
<td>$O(n^3)$</td>
<td>$O\left(\frac{n^3}{\sqrt{M}}\right)$</td>
</tr>
</tbody>
</table>
Costs of matrix multiplication algorithms

Let $M$ be the size of the fast memory.
The blocked algorithm uses a block size of $\sqrt{M/3}$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Computation</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mat-Vec Algorithm</td>
<td>$O(n^3)$</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>Blocked Algorithm</td>
<td>$O(n^3)$</td>
<td>$O\left(\frac{n^3}{\sqrt{M}}\right)$</td>
</tr>
</tbody>
</table>

Can we do better than the blocked algorithm?
Costs of matrix multiplication algorithms

Let $M$ be the size of the fast memory.

The blocked algorithm uses a block size of $\sqrt{M/3}$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Computation</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mat-Vec Algorithm</td>
<td>$O(n^3)$</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>Blocked Algorithm</td>
<td>$O(n^3)$</td>
<td>$O\left(\frac{n^3}{\sqrt{M}}\right)$</td>
</tr>
</tbody>
</table>

Can we do better than the blocked algorithm?

No . . . and Yes
Avoiding Communication
- some communication is necessary: we can prove lower bounds
- theoretical analysis identifies sub-optimal algorithms and spurs algorithmic innovation
- minimizing communication leads to speedups in practice

Reducing Computation and Communication
- there exist theoretical “fast” matrix multiplication algorithms
- I want to make them practical
Prior Work: Classical Matrix Multiplication

- Assume $O(n^3)$ algorithm
- Sequential case with fast memory of size $M$
  - lower bound on words moved between fast/slow mem:
    $$\Omega \left( \frac{n^3}{\sqrt{M}} \right) \quad \text{[Hong & Kung 81]}$$
    - attained by blocked algorithm
- Parallel case with $P$ processors (local memory of size $M$)
  - lower bound on words communicated (along critical path):
    $$\Omega \left( \frac{n^3}{P\sqrt{M}} \right) \quad \text{[Toledo et al. 04]}$$
    - also attainable
Theorem (Ballard, Demmel, Holtz, Schwartz 11)

If a computation “smells” like 3 nested loops, it must communicate

\[
\# \text{ words} = \Omega \left( \frac{\# \text{ flops}}{\sqrt{\text{memory size}}} \right)
\]

This result applies to
- dense or sparse problems
- sequential or parallel computers

This work was recognized with the SIAM Linear Algebra Prize, given to the best paper from the years 2009-2011
Extensions to the rest of linear algebra

Theorem (Ballard, Demmel, Holtz, Schwartz 11)

If a computation “smells” like 3 nested loops, it must communicate

\[ \# \text{words} = \Omega \left( \frac{\# \text{flops}}{\sqrt{\text{memory size}}} \right) \]

What smells like 3 nested loops?
- the rest of BLAS 3 (e.g. matrix multiplication, triangular solve)
- Cholesky, LU, $LDL^T$, $LTL^T$ decompositions
- QR decomposition
- eigenvalue and SVD reductions
- sequences of algorithms (e.g. repeated matrix squaring)
- graph algorithms (e.g. all pairs shortest paths)

This work was recognized with the SIAM Linear Algebra Prize, given to the best paper from the years 2009-2011
Main Idea of Lower Bound Proof

Crux of proof based on geometric inequality [Loomis & Whitney 49]

Volume of box

\[ V = xyz = \sqrt{xz} \cdot \sqrt{yz} \cdot \sqrt{xy} \]

Volume of a 3D set

\[ V \leq \sqrt{\text{area}(A \text{ shadow})} \cdot \sqrt{\text{area}(B \text{ shadow})} \cdot \sqrt{\text{area}(C \text{ shadow})} \]

Given limited set of data, how much useful computation can be done?
Extensions to the rest of linear algebra

Theorem (Ballard, Demmel, Holtz, Schwartz 11)

If a computation “smells” like 3 nested loops, it must communicate

\[ \# \text{ words} = \Omega \left( \frac{\# \text{ flops}}{\sqrt{\text{memory size}}} \right) \]

What smells like 3 nested loops?

- the rest of BLAS 3 (e.g. matrix multiplication, triangular solve)
- Cholesky, LU, \(LDL^T\), \(LTL^T\) decompositions
- QR decomposition
- eigenvalue and SVD reductions
- sequences of algorithms (e.g. repeated matrix squaring)
- graph algorithms (e.g. all pairs shortest paths)

This work was recognized with the SIAM Linear Algebra Prize, given to the best paper from the years 2009-2011
Optimal Algorithms - Sequential $O(n^3)$ Linear Algebra

<table>
<thead>
<tr>
<th>Computation</th>
<th>Optimal Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS 3</td>
<td>blocked algorithms [Gustavson 97]</td>
</tr>
<tr>
<td>Cholesky</td>
<td>LAPACK [Ahmed &amp; Pingali 00] [BDHS10]</td>
</tr>
<tr>
<td>Symmetric Indefinite</td>
<td>LAPACK (rarely) [BDD+12a]</td>
</tr>
<tr>
<td>LU</td>
<td>LAPACK (rarely) [Toledo 97]* [Grigori et al. 11]</td>
</tr>
<tr>
<td>QR</td>
<td>LAPACK (rarely) [Frens &amp; Wise 03] [Elmo Roth &amp; Gustavson 98]* [Hoemmen et al. 12]*</td>
</tr>
<tr>
<td>Eig, SVD</td>
<td>[BDK12a], [BDD12b]</td>
</tr>
</tbody>
</table>

Parallel Case

SLOW

FAST
Suppose we want to solve $Ax = b$ where $A$

- is symmetric (save half the storage and flops)
- but indefinite (need to permute rows/cols for numerical stability)

We generally want to compute a factorization

$$PAP^T = LTL^T$$

$P$ is a permutation, $L$ is triangular, and $T$ is symmetric and “simpler”
Reducing communication improves performance

Performance of symmetric indefinite linear system solvers

Implemented within PLASMA library [BBD+13]
This work will receive a Best Paper Award at IPDPS ’13
Example: Compute Eigenvalues of Band Matrix

Suppose we want to solve $Ax = \lambda x$ where $A$

- is symmetric (save half the storage and flops)
- has band structure (exploit sparsity – ignore zeros)

We generally want to compute a factorization

$$A = QTQ^T$$

$Q$ is an orthogonal matrix and $T$ is symmetric tridiagonal
Successive Band Reduction (bulge-chasing)

constraint:
\[ c + d \leq b \]

\[ b = \text{bandwidth} \]
\[ c = \text{columns} \]
\[ d = \text{diagonals} \]
CASBR Data Access Pattern

One bulge at a time

Four bulges at a time
Implementation of Band Eigensolver (CASBR)

Speedup of sequential CASBR over Intel’s Math Kernel Library

Benchmarked on Intel 10-core Westmere-EX socket [BDK12a]
Implementation of Band Eigensolver (CASBR)

Speedup of parallel CASBR (10 threads) over PLASMA library

Benchmarked on Intel 10-core Westmere-EX socket [BDK12a]
Example Application: Video Background Subtraction

Idea: use Robust PCA algorithm [CLMW09] to subtract constant background from the action of a surveillance video

Given a matrix $M$ whose columns represent frames, compute

$$M = L + S$$

where $L$ is low-rank and $S$ is sparse
Example Application: Video Background Subtraction

Compute:

\[ M = L + S \]

where \( L \) is low-rank and \( S \) is sparse

The algorithm works iteratively, each iteration requires a singular value decomposition (SVD)

- \( M \) is \( 110,000 \times 100 \)

Communication-avoiding algorithm provided 3\( \times \) speedup over best GPU implementation [ABDK11]
Theory to Practice

Theoretical Lower Bounds

Algorithmic Innovation

Optimized Implementation

Improved Applications
Can we do better than the blocked algorithm?

Given the computation involved, it minimized communication...
Let’s go back to matrix multiplication

Can we do better than the blocked algorithm?

Given the computation involved, it minimized communication. . .

. . . but what if we change the computation?

It’s possible to reduce both computation and communication
Strassen’s Algorithm

Strassen showed how to use 7 multiplies instead of 8 for $2 \times 2$ multiplication

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

**Classical Algorithm**

\[
\begin{align*}
M_1 &= A_{11} \cdot B_{11} \\
M_2 &= A_{12} \cdot B_{21} \\
M_3 &= A_{11} \cdot B_{12} \\
M_4 &= A_{12} \cdot B_{22} \\
M_5 &= A_{21} \cdot B_{11} \\
M_6 &= A_{22} \cdot B_{21} \\
M_7 &= A_{21} \cdot B_{12} \\
M_8 &= A_{22} \cdot B_{22}
\end{align*}
\]

\[
\begin{align*}
C_{11} &= M_1 + M_2 \\
C_{12} &= M_3 + M_4 \\
C_{21} &= M_5 + M_6 \\
C_{22} &= M_7 + M_8
\end{align*}
\]

**Strassen’s Algorithm**

\[
\begin{align*}
M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\end{align*}
\]

\[
\begin{align*}
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6
\end{align*}
\]
Strassen’s Algorithm

Strassen showed how to use 7 multiplies instead of 8 for $2 \times 2$ multiplication.

\[
\begin{array}{cc}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{array}
= \begin{array}{cc}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{array}
\cdot \begin{array}{cc}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{array}
\]

Flop count recurrence:

\[
F(n) = 7 \cdot F(n/2) + \Theta(n^2)
\]

\[
F(n) = \Theta(n^{\log_2 7})
\]

\[
\log_2 7 \approx 2.81
\]

\[
M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})
\]

\[
M_2 = (A_{21} + A_{22}) \cdot B_{11}
\]

\[
M_3 = A_{11} \cdot (B_{12} - B_{22})
\]

\[
M_4 = A_{22} \cdot (B_{21} - B_{11})
\]

\[
M_5 = (A_{11} + A_{12}) \cdot B_{22}
\]

\[
M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})
\]

\[
M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\]

\[
C_{11} = M_1 + M_4 - M_5 + M_7
\]

\[
C_{12} = M_3 + M_5
\]

\[
C_{21} = M_2 + M_4
\]

\[
C_{22} = M_1 - M_2 + M_3 + M_6
\]
If you implement Strassen’s algorithm recursively on a sequential computer:

<table>
<thead>
<tr>
<th></th>
<th>Computation</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical (blocked)</td>
<td>$O(n^3)$</td>
<td>$O\left(\left(\frac{n}{\sqrt{M}}\right)^3 M\right)$</td>
</tr>
<tr>
<td>Strassen</td>
<td>$O(n^{\log_2 7})$</td>
<td>$O\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$</td>
</tr>
</tbody>
</table>
If you implement Strassen’s algorithm recursively on a sequential computer:

<table>
<thead>
<tr>
<th></th>
<th>Computation</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical (blocked)</td>
<td>$O(n^3)$</td>
<td>$O\left(\left(\frac{n}{\sqrt{M}}\right)^3 M\right)$</td>
</tr>
<tr>
<td>Strassen</td>
<td>$O(n^{\log_2 7})$</td>
<td>$O\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$</td>
</tr>
</tbody>
</table>

Can we reduce Strassen’s communication cost further?
Theorem (Ballard, Demmel, Holtz, Schwartz 12)

On a sequential machine, Strassen’s algorithm must communicate

\[ \# \text{ words} = \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} M \right) \]

and on a parallel machine, it must communicate

\[ \# \text{ words} = \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} \frac{M}{P} \right) \]
Theorem (Ballard, Demmel, Holtz, Schwartz 12)

On a sequential machine, Strassen’s algorithm must communicate

$$\# \text{ words} = \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} M \right)$$

and on a parallel machine, it must communicate

$$\# \text{ words} = \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} \frac{M}{P} \right)$$

This work

- received the SPAA Best Paper Award [BDHS11]
- appeared in the Journal of the ACM [BDHS12a]
- and has been invited to appear as a Research Highlight in the Communications of the ACM
This lower bound proves that the sequential recursive algorithm is communication-optimal

What about the parallel case?
This lower bound proves that the sequential recursive algorithm is communication-optimal.

What about the parallel case?

- Earlier attempts to parallelize Strassen had communication costs which exceeded the lower bound.
- We developed a new algorithm that is communication-optimal, called Communication-Avoiding Parallel Strassen (CAPS) [BDH+12].
Main idea of CAPS algorithm

At each level of recursion tree, choose either breadth-first or depth-first traversal of the recursion tree

**Breadth-First-Search (BFS)**

- Runs all 7 multiplies in parallel
- each uses \( P/7 \) processors
- Requires 7/4 as much extra memory
- Requires communication, but minimizes communication in subtrees

**Depth-First-Search (DFS)**

- Runs all 7 multiplies sequentially
- each uses all \( P \) processors
- Requires 1/4 as much extra memory
- Increases communication by factor of 7/4 in subtrees
Performance of CAPS on a large problem

Strong-scaling on a Cray XT4, \( n = 94,080 \)

![Graph showing performance comparison between different algorithms]

- Classical Peak
- New Algorithm
- Best Previous Strassen
- Best Classical

Effective Gflops / sec / processor vs. Number of Processors
Can we beat Strassen?

Strassen’s algorithm allows for less computation and communication than the classical $O(n^3)$ algorithm.

We have algorithms that attain its communication lower bounds and perform well on highly parallel machines.

Can we do any better?
Can an \( n \times n \) linear system of equations \( Ax = b \) be solved in \( O(n^{2+\varepsilon}) \) operations, where \( \varepsilon \) is arbitrarily small?

\[
\text{\ldots if solved affirmatively, [this] would change the world.}
\]

It is an article of faith for some of us that if \( O(n^{2+\varepsilon}) \) is ever achieved, the big idea that achieves it will correspond to an algorithm that is really practical.

-Nick Trefethen, 2012 SIAM President
How much computation will that save?

Percentage of Classical Computation

<table>
<thead>
<tr>
<th>Exponent</th>
<th>n=10^4</th>
<th>n=10^5</th>
<th>n=10^6</th>
<th>n=10^7</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.01%</td>
<td>0.1%</td>
<td>1%</td>
<td>10%</td>
</tr>
<tr>
<td>2.81</td>
<td>100%</td>
<td>10%</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>2.37</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>2+\varepsilon</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>
Can we beat Strassen?

Exponent of matrix multiplication over time

Unfortunately, these improvements are only theoretical because they involve approximations and have (possibly) large constants.
Can we beat Strassen?

Exponent of matrix multiplication over time

Unfortunately, these improvements are only theoretical because they
- involve approximations
- are existence proofs
- have (possibly) large constants
Solving the base case...

\[ 2 \times 2 \times 2 \]

\[
\begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{bmatrix}
\begin{bmatrix}
b_{11} & b_{12} \\
b_{21} & b_{22}
\end{bmatrix} =
\begin{bmatrix}
c_{11} & c_{12} \\
c_{21} & c_{22}
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th>multiplies</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>flop count</td>
<td>( \mathcal{O}(n^{2.58}) )</td>
<td>( \mathcal{O}(n^{2.81}) )</td>
<td>( \mathcal{O}(n^3) )</td>
</tr>
</tbody>
</table>
Solving the base case...

\[ 2 \times 2 \times 2 \]

\[
\begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{bmatrix}
\begin{bmatrix}
b_{11} & b_{12} \\
b_{21} & b_{22}
\end{bmatrix}
= 
\begin{bmatrix}
c_{11} & c_{12} \\
c_{21} & c_{22}
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th>multiplies</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>flop count</td>
<td>(O(n^{2.58}))</td>
<td>(O(n^{2.81}))</td>
<td>(O(n^3))</td>
</tr>
</tbody>
</table>
Solving the base case...

\[ 2 \times 2 \times 2 \]

\[
\begin{bmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{bmatrix}
\begin{bmatrix}
b_{11} & b_{12} \\
b_{21} & b_{22}
\end{bmatrix}
= 
\begin{bmatrix}
c_{11} & c_{12} \\
c_{21} & c_{22}
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th>multiplies</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>flop count</td>
<td>(O(n^{2.58}))</td>
<td>(O(n^{2.81}))</td>
<td>(O(n^3))</td>
</tr>
</tbody>
</table>

\[ 3 \times 3 \times 3 \]

\[
\begin{bmatrix}
a_{11} & a_{12} & a_{13} \\
a_{21} & a_{22} & a_{23} \\
a_{31} & a_{32} & a_{33}
\end{bmatrix}
\begin{bmatrix}
b_{11} & b_{12} & b_{13} \\
b_{21} & b_{22} & b_{23} \\
b_{31} & b_{32} & b_{33}
\end{bmatrix}
= 
\begin{bmatrix}
c_{11} & c_{12} & c_{13} \\
c_{21} & c_{22} & c_{23} \\
c_{31} & c_{32} & c_{33}
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th>multiplies</th>
<th>19</th>
<th>21</th>
<th>23</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>flop count</td>
<td>(O(n^{2.68}))</td>
<td>(O(n^{2.77}))</td>
<td>(O(n^{2.85}))</td>
<td>(O(n^3))</td>
</tr>
</tbody>
</table>
Finding a better base case corresponds to computing a low-rank decomposition of a particular 3D tensor

\[ \text{Cube} = \text{Line} + \cdots + \text{Line} \]
Beating Strassen

Finding a better base case corresponds to computing a low-rank decomposition of a particular 3D tensor

Unfortunately, this is a nonlinear integer optimization problem
- it’s NP-complete (in general), but need to solve it only once
- I used this method to re-discover Strassen

Could use (numerical) low-rank tensor approximation algorithms
- very efficient, but no guarantees
Beating Strassen

Finding a better base case corresponds to computing a low-rank decomposition of a particular 3D tensor

\[
\text{Cube} = \begin{array}{c}
\text{Bar 1} \\
\text{Bar 2} \quad + \quad \ldots \quad + \\
\text{Bar N}
\end{array}
\]

If we find it, we can make it practical!

- same parallelization as Strassen, but with less computation \textit{and} communication
Collaborators

- Michael Anderson (UC Berkeley)
- Aydin Buluc (LBNL)
- James Demmel (UC Berkeley)
- Alex Druinsky (Tel-Aviv U)
- Ioana Dumitriu (U Washington)
- Andrew Gearhart (UC Berkeley)
- Laura Grigori (INRIA)
- Olga Holtz (UC Berkeley/TU Berlin)
- Mathias Jacquelin (INRIA)
- Nicholas Knight (UC Berkeley)
- Kurt Keutzer (UC Berkeley)
- Tamara Kolda (Sandia NL)
- Benjamin Lipshitz (UC Berkeley)
- Inon Peled (Tel-Aviv U)
- Todd Plantenga (Sandia NL)
- Oded Schwartz (UC Berkeley)
- Edgar Solomonik (UC Berkeley)
- Sivan Toledo (Tel-Aviv U)
- Ichitaro Yamazaki (UT Knoxville)
Thank You!

www.eecs.berkeley.edu/~ballard
http://bebop.cs.berkeley.edu
Other Ongoing and Future Projects

- implementing these algorithms in communication-bound settings
  - e.g., SVD of a tall-skinny matrix on a Hadoop cluster

- extending these algorithmic ideas to sparse matrices
  - e.g., sparse matrix-matrix multiplication

- using Strassen to do the rest of linear algebra in parallel

- trading off local memory and communication in parallel QR decomposition
## Memory-Independent Lower Bounds

<table>
<thead>
<tr>
<th></th>
<th>Classical</th>
<th>Strassen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory-dependent</td>
<td>$\Omega \left( \frac{n^3}{P\sqrt{M}} \right)$</td>
<td>$\Omega \left( \frac{n^\omega}{PM^{\omega/2} - 1} \right)$</td>
</tr>
<tr>
<td>lower bound</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory-independent</td>
<td>$\Omega \left( \frac{n^2}{P^{2/3}} \right)$</td>
<td>$\Omega \left( \frac{n^2}{P^{2/\omega}} \right)$</td>
</tr>
<tr>
<td>lower bound</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Perfect strong</td>
<td>$P = O \left( \frac{n^3}{M^{3/2}} \right)$</td>
<td>$P = O \left( \frac{n^\omega}{M^{\omega/2}} \right)$</td>
</tr>
<tr>
<td>scaling range</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attaining algorithm</td>
<td>[SD11]</td>
<td>[BDH+12]</td>
</tr>
</tbody>
</table>
Overview of Divide & Conquer Algorithm for Nonsymmetric Eigenproblem

One step of divide and conquer:

1. Compute \( \left( I + (A^{-1})^{2^k} \right)^{-1} \) implicitly
   - maps eigenvalues of \( A \) to 0 and 1 (roughly)
2. Compute rank-revealing decomposition to find invariant subspace
3. Output block-triangular matrix

\[
A_{\text{new}} = U^* A U = \begin{bmatrix}
A_{11} & A_{12} \\
\varepsilon & A_{22}
\end{bmatrix}
\]

- block sizes chosen to minimize norm of \( \varepsilon \)
- eigenvalues of \( A_{11} \) all lie outside unit circle, eigenvalues of \( A_{22} \) lie inside unit circle, subproblems solved recursively
- stable, but progress guaranteed only with high probability
Reduction Example: LU

It’s easy to reduce matrix multiplication to LU:

\[
T \equiv \begin{bmatrix}
I & 0 & -B \\
A & I & 0 \\
0 & 0 & I
\end{bmatrix} = \begin{bmatrix}
I & 0 & -B \\
A & I & 0 \\
0 & 0 & I
\end{bmatrix} \begin{bmatrix}
I & 0 & -B \\
I & A \cdot B & I
\end{bmatrix} \equiv L \cdot U
\]

- LU factorization can be used to perform matrix multiplication
- Communication lower bound for matrix multiplication applies to LU

Reduction to Cholesky is a little trickier, but same idea [BDHS10]
### Algorithms - Parallel $O(n^3)$ Linear Algebra

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Reference</th>
<th>Factor exceeding lower bound for # words</th>
<th>Factor exceeding lower bound for # messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Multiply</td>
<td>[Can69]</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cholesky</td>
<td>ScaLAPACK</td>
<td>$\log P$</td>
<td>$\log P$</td>
</tr>
<tr>
<td>Symmetric Indefinite</td>
<td>[BDD$^+$12a]</td>
<td>proposed work $\log P$</td>
<td>proposed work $(N/P^{1/2})\log P$</td>
</tr>
<tr>
<td></td>
<td>ScaLAPACK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LU</td>
<td>[GDX11] ScaLAPACK</td>
<td>$\log P$</td>
<td>$\log P$</td>
</tr>
<tr>
<td>QR</td>
<td>[DGHL12] ScaLAPACK</td>
<td>$\log P$</td>
<td>$\log P$</td>
</tr>
<tr>
<td>SymEig, SVD</td>
<td>[BDK12a] ScaLAPACK</td>
<td>proposed work $\log P$</td>
<td>proposed work $N/P^{1/2}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NonsymEig</td>
<td>[BDD12b] ScaLAPACK</td>
<td>$\log P$</td>
<td>$\log^3 P$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$P^{1/2}\log P$</td>
<td>$N\log P$</td>
</tr>
</tbody>
</table>

*This table assumes that one copy of the data is distributed evenly across processors.

Red = not optimal
Symmetric Eigenproblem and SVD via SBR

We’re solving the symmetric eigenproblem via reduction to tridiagonal form

- Conventional approach (e.g. LAPACK) is direct tridiagonalization
- Two-phase approach reduces first to band, then band to tridiagonal

**Direct:**

\[
\begin{pmatrix}
1 & 2 \\
1 & 2
\end{pmatrix}
\]

**Two-step:**

\[
\begin{pmatrix}
1 & 2 \\
1 & 2
\end{pmatrix}
\]

- first phase can be done efficiently
- second phase is trickier, requires successive band reduction (SBR) [BLS00]

\[
\begin{pmatrix}
1 & 2 \\
1 & 2
\end{pmatrix}
\]

- involves “bulge-chasing”
- we’ve improved it to reduce communication [BDK12b]
Communication-Avoiding SBR - theory

<table>
<thead>
<tr>
<th>Method</th>
<th>Flops</th>
<th>Words Moved</th>
<th>Data Re-use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schwarz</td>
<td>$4n^2b$</td>
<td>$O(n^2b)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>M-H</td>
<td>$6n^2b$</td>
<td>$O(n^2b)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>B-L-S*</td>
<td>$5n^2b$</td>
<td>$O(n^2 \log b)$</td>
<td>$O\left(\frac{b}{\log b}\right)$</td>
</tr>
<tr>
<td>CA-SBR†</td>
<td>$5n^2b$</td>
<td>$O\left(\frac{n^2b^2}{M}\right)$</td>
<td>$O\left(\frac{M}{b}\right)$</td>
</tr>
</tbody>
</table>

*with optimal parameter choices
†assuming $1 \leq b \leq \sqrt{M}/3$
Symmetric Indefinite Factorization

We’re solving $Ax = b$ where $A = A^T$ but $A$ is indefinite

- Standard approach is to compute $PAP^T = LDL^T$
  - $L$ is lower triangular and $D$ is block diagonal (1 $\times$ 1 and 2 $\times$ 2 blocks)
  - requires complicated pivoting, harder to do tournament pivoting

- Alternative approach is to compute $PAP^T = LTL^T$ [Aas71]
  - $L$ is lower triangular and $T$ is tridiagonal
  - pivoting is more like LU (nonsymmetric case)
Performance of CAPS on large problems

Strong-scaling on Intrepid (IBM BG/P), \( n = 65,856 \).
Performance of CAPS on large problems

Strong-scaling on Intrepid (IBM BG/P), $n = 65,856$. 

![Graph showing performance comparison]
Comparison of the parallel models with the algorithms in strong scaling of matrix dimension $n = 65,856$ on Intrepid.
J. O. Aasen.
On the reduction of a symmetric matrix to tridiagonal form.
10.1007/BF01931804.

Communication-avoiding QR decomposition for GPUs.

Implementing a blocked Aasen’s algorithm with a dynamic scheduler on multicore architectures, 2013.
To appear.

Communication avoiding symmetric indefinite factorization, 2012.
In preparation.

In preparation.

Communication-optimal parallel algorithm for Strassen’s matrix multiplication.
Communication-optimal parallel and sequential Cholesky decomposition.

Graph expansion and communication costs of fast matrix multiplication: regular submission.

Graph expansion and communication costs of fast matrix multiplication.

Sequential communication bounds for fast linear algebra.
Technical Report UCB/EECS-2012-36, EECS Department, University of California, Berkeley, Mar 2012.

G. Ballard, J. Demmel, and N. Knight.
Avoiding communication in the symmetric eigenproblem and SVD, 2012.
In preparation.

G. Ballard, J. Demmel, and N. Knight.
Communication avoiding successive band reduction.

C. Bischof, B. Lang, and X. Sun.
A framework for symmetric band reduction.
L. Cannon.
A cellular computer to implement the Kalman filter algorithm.

Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright.
Robust principal component analysis?

J. Demmel, L. Grigori, M. Hoemmen, and J. Langou.
Communication-optimal parallel and sequential QR and LU factorizations.

L. Grigori, J. Demmel, and H. Xiang.
CALU: A communication optimal LU factorization algorithm.

E. Solomonik and J. Demmel.
Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms.
Avoiding Communication in Linear Algebra

Grey Ballard

Thank You!

www.eecs.berkeley.edu/~ballard
http://bebop.cs.berkeley.edu