Communication-Avoiding Algorithms for Linear Algebra and Beyond

Jim Demmel
EECS & Math Departments
UC Berkeley
Why avoid communication?

• Communication = moving data
  – Between levels of memory hierarchy, between processors over network, ...

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

• Time_per_flop \ll 1/bandwidth \ll latency
  • Gaps growing exponentially with time

• Avoid communication to save time
Why avoid communication?

• Communication = moving data
  – Between levels of memory hierarchy, between processors over network, ...

• 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

• Avoid communication to save time

• Same story for energy:
  Avoid communication to save energy
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.”

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.”

Avoiding Communication in Direct Linear Algebra

• Communication lower bounds for “3 nested loops”
  – Matmul, BLAS, LU, QR, eig, svd, APSP, whole programs, ...
  – Dense and sparse matrices
  – Sequential, parallel and heterogeneous architectures

• Not attained by standard algorithms
  – Big speedups in theory and practice (eg 12x for matmul on BG/P)

• Some open problems
  – More practical optimal eigenvalue algorithms
  – Optimal algorithms for sparse matrices
  – Extensions to more Strassen-like algorithms
  – Use all available memory to further reduce communication: “2.5D algorithms”
  – How do latency and bandwidth costs tradeoff?
  – Which network topologies are needed to attain bounds?
  – Are there cache/network/hierarchy oblivious algorithms?
Perfect Strong Scaling – in Time and Energy

- If we can run on P processors, how much time/energy needed on cP processors?
  - Use all additional processors and additional memory

- $T_{MM}(cP) = \frac{n^3}{(cP)} \left[ \gamma_T + \frac{\beta_T}{M^{1/2}} + \frac{\alpha_T}{(mM^{1/2})} \right] = \frac{T_{MM}(P)}{c}$
- $E_{MM}(cP) = cP \left\{ \frac{n^3}{(cP)} \left[ \gamma_E + \frac{\beta_E}{M^{1/2}} + \frac{\alpha_E}{(mM^{1/2})} \right] + \delta_{EM}T(cP) + \varepsilon_T(cP) \right\} = E_{MM}(P)$
**Perfect Strong Scaling – in Time and Energy**

- If we can run on $P$ processors, how much time/energy needed on $cP$ processors?
  - Use all additional processors *and* additional memory

- $T_{\text{MM}}(cP) = \frac{n^3}{(cP)} \left[ \gamma_T + \beta_T/M^{1/2} + \alpha_T/(mM^{1/2}) \right] = T_{\text{MM}}(P)/c$
- $E_{\text{MM}}(cP) = cP \left\{ \frac{n^3}{(cP)} \left[ \gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2}) \right] + \delta_E M T(cP) + \varepsilon_E T(cP) \right\} = E_{\text{MM}}(P)$

- Perfect strong scaling extends to N-body, Strassen, ...
- We can use these models to answer many questions, including:

  - What is the minimum energy $E$ required for a computation?
  - Given a maximum runtime $T$, what is the minimum energy $E$ needed to achieve it?
  - Given a maximum energy $E$, what is the minimum runtime $T$ that we can attain?
  - Can we minimize the average power $P = E/T$?
  - Given an algorithm, problem size, bounds on power $P$ and time $T$, what architectural parameters are needed to attain these bounds?
    - How can we attain an exaflop for 20 Mwatts?
Beyond Linear algebra

• Communication lower bounds extend to any program that
  – Has iterations indexed by \((i_1, i_2, i_3, \ldots)\)
  – Accesses arrays \(A(i_1, i_3, 3i_2 - 5i_6 + 4i_7), B(pnt(i_2 + i_3)), \ldots\)
  – Eg: Linear algebra, n-body, data-base join, ...

• Builds on Christ/Tao/Carberry/Bennett (2008)

• Open questions
  – How can we write down lower bounds?
    • Need to avoid Hilbert’s 10th problem over \(\mathbb{Q}\)
  – When can we attain lower bounds?
    • Conjecture: Always, dependencies permitting
  – Generalizations of direct linear algebra questions:
    • “2.5D” algorithms? Network topologies? Latency vs bandwidth?
  – Can this be automated by compilers?
Avoiding Communication in Iterative Linear Algebra

• k steps of a Krylov method for $Ax=b$ or $Ax=\lambda x$ costs $k$ Sparse-matrix-vector-multiplies (SpMV), $k$ dot products, etc.

• New algorithms – assume A “well-partitioned”
  – Reduces sequential bandwidth cost of $k$ steps from $O(k)$ to $O(1)$ – optimal
  – Reduces parallel latency cost of $k$ steps from $O(k \log p)$ to $O(\log p)$ – optimal
  – Big speedups in theory and practice

• Open problems
  – Continue reorganizing other Krylov methods
  – Generate, recognize stable variants more easily
  – Accommodate more preconditioners, eg multigrid
  – Accommodate other types of sparse matrices
Other open problems

• How can we automatically use the least precision needed to get the right answer?
  – Saves time, energy, space
  – Precimonious – tool under development

• How much does it cost to guarantee reproducibility from run to run?
  – Important for reproducible science, simulating rare events, debugging, ...
  – New algorithms for summation, generalizations
Collaborators and Supporters

- Austin Benson, Maryam Dehnavi, Mark Hoemmen, Shoaib Kamil, Marghoob Mohiyuddin
- Abhinav Bhatelte, Aydin Buluc, Michael Christ, Ioana Dumitriu, Armando Fox, David Gleich, Ming Gu, Jeff Hammond, Mike Heroux, Olga Holtz, Kurt Keutzer, Julien Langou, Devin Matthews, Tom Scanlon, Michelle Strout, Sam Williams, Hua Xiang
- Jack Dongarra, Dulceneia Becker, Ichitaro Yamazaki
- Sivan Toledo, Alex Druinsky, Inon Peled
- Laura Grigori, Sebastien Cayrols, Simplice Donfack, Mathias Jacquelin, Amal Khabou, Sophie Moufawad, Mikolaj Szydlarski
- Members of ParLab, ASPIRE, BEBOP, CACHE, EASI, FASTMath, MAGMA, PLASMA
- Thanks to DOE, NSF, UC Discovery, INRIA, Intel, Microsoft, Mathworks, National Instruments, NEC, Nokia, NVIDIA, Samsung, Oracle

- bebop.cs.berkeley.edu
Summary

Time to redesign all linear algebra, n-body, ... algorithms and software (and compilers)

Don’t Communic...
Backup slides
Why avoid communication? (1/3)

Algorithms have two costs (measured in time or energy):
1. Arithmetic (FLOPS)
2. Communication: moving data between
   – levels of a memory hierarchy (sequential case)
   – processors over a network (parallel case).
Why avoid communication? (2/3)

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

  \[
  \text{communication} = \{ \text{running time} \}
  \]

• Time_per_flop \ll 1/\text{bandwidth} \ll \text{latency}
  
  • Gaps growing exponentially with time [FOSC]

<table>
<thead>
<tr>
<th>Annual improvements</th>
<th>Time_per_flop</th>
<th>Bandwidth</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>59% Network</td>
<td>26%</td>
<td>15%</td>
</tr>
<tr>
<td></td>
<td>59% DRAM</td>
<td>23%</td>
<td>5%</td>
</tr>
</tbody>
</table>

• Avoid communication to save time
Why Minimize Communication? (3/3)

Minimize communication to save energy

Source: John Shalf, LBL
Goals

• Redesign algorithms to *avoid* communication
  • Between all memory hierarchy levels
    • L1 ↔ L2 ↔ DRAM ↔ network, etc
• Attain lower bounds if possible
  • Current algorithms often far from lower bounds
  • Large speedups and energy savings possible
Outline

• Survey state of the art of CA (Comm-Avoiding) algorithms
  – Review previous Matmul algorithms
  – CA $O(n^3)$ 2.5D Matmul
  – TSQR: Tall-Skinny QR
  – CA $O(n^3)$ 2.5D LU
  – CA Strassen Matmul

• Beyond linear algebra
  – Extending lower bounds to any algorithm with arrays
  – Communication-optimal N-body algorithm

• CA-Krylov methods
Outline

• Survey state of the art of CA (Comm-Avoiding) algorithms
  – Review previous Matmul algorithms
  – CA $O(n^3)$ 2.5D Matmul
  – TSQR: Tall-Skinny QR
  – CA $O(n^3)$ 2.5D LU
  – CA Strassen Matmul

• Beyond linear algebra
  – Extending lower bounds to any algorithm with arrays
  – Communication-optimal N-body algorithm

• CA-Krylov methods
Summary of CA Linear Algebra

- “Direct” Linear Algebra
  - Lower bounds on communication for linear algebra problems like $Ax=b$, least squares, $Ax = \lambda x$, SVD, etc
  - Mostly not attained by algorithms in standard libraries
  - New algorithms that attain these lower bounds
    - Being added to libraries: Sca/LAPACK, PLASMA, MAGMA
    - Large speed-ups possible
    - Autotuning to find optimal implementation

- Ditto for “Iterative” Linear Algebra
Lower bound for all “$n^3$-like” linear algebra

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

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

- Parallel case: assume either load or memory balanced

- Holds for
  - Matmul
Lower bound for all “n^3-like” linear algebra

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

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

\[ \#\text{messages\_sent} \geq \#\text{words\_moved} / \text{largest\_message\_size} \]

• Parallel case: assume either load or memory balanced

• Holds for
  - Matmul, 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 \( << n^3 \))
  - Sequential and parallel algorithms
  - Some graph-theoretic algorithms (eg Floyd-Warshall)
Lower bound for all “$n^3$-like” linear algebra

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

  \[
  \#\text{words}_\text{moved} (\text{per processor}) = \Omega(\#\text{flops} (\text{per processor}) / M^{1/2})
  \]

  \[
  \#\text{messages}_\text{sent} (\text{per processor}) = \Omega(\#\text{flops} (\text{per processor}) / M^{3/2})
  \]

- Parallel case: assume either load or memory balanced

- Holds for
  - Matmul, BLAS, LU, QR, eig, SVD, tensor contractions, ... 
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, eg $A^k$)

SIAM SIAG/Linear Algebra Prize, 2012
Ballard, D., Holtz, Schwartz
Can we attain these lower bounds?

• Do conventional dense algorithms as implemented in LAPACK and ScaLAPACK attain these bounds?
  – Often not

• If not, are there other algorithms that do?
  – Yes, for much of dense linear algebra
  – New algorithms, with new numerical properties, new ways to encode answers, new data structures
  – Not just loop transformations (need those too!)

• Only a few sparse algorithms so far

• Lots of work in progress
Outline

• Survey state of the art of CA (Comm-Avoiding) algorithms
  – Review previous Matmul algorithms
    – CA $O(n^3)$ 2.5D Matmul
    – TSQR: Tall-Skinny QR
    – CA $O(n^3)$ 2.5D LU
    – CA Strassen Matmul

• Beyond linear algebra
  – Extending lower bounds to any algorithm with arrays
  – Communication-optimal N-body algorithm

• CA-Krylov methods
Naïve Matrix Multiply

{implements $C = C + A*B$}

for $i = 1$ to $n$

for $j = 1$ to $n$

for $k = 1$ to $n$

$C(i,j) = C(i,j) + A(i,k) * B(k,j)$
Naïve Matrix Multiply

{implements $C = C + A*B$}
for $i = 1$ to $n$
{read row $i$ of $A$ into fast memory}
for $j = 1$ to $n$
{read $C(i,j)$ into fast memory}
{read column $j$ of $B$ into fast memory}
for $k = 1$ to $n$
\[ C(i,j) = C(i,j) + A(i,k) \times B(k,j) \]
{write $C(i,j)$ back to slow memory}
Naïve Matrix Multiply

\{ \text{implements } C = C + A \times B \}\n
\text{for } i = 1 \text{ to } n

\{ \text{read row } i \text{ of } A \text{ into fast memory} \}\n
\text{for } j = 1 \text{ to } n

\{ \text{read } C(i,j) \text{ into fast memory} \}\n
\{ \text{read column } j \text{ of } B \text{ into fast memory} \}\n
\text{for } k = 1 \text{ to } n

\text{C}(i,j) = C(i,j) + A(i,k) \times B(k,j)

\{ \text{write } C(i,j) \text{ back to slow memory} \}\n
\text{... } n^2 \text{ writes altogether}

n^3 + 3n^2 \text{ reads/writes altogether – dominates } 2n^3 \text{ arithmetic}
Consider $A, B, C$ to be $n/b$-by-$n/b$ matrices of $b$-by-$b$ subblocks where $b$ is called the block size; assume $3$ $b$-by-$b$ blocks fit in fast memory

For $i = 1$ to $n/b$

For $j = 1$ to $n/b$

{read block $C(i,j)$ into fast memory}

For $k = 1$ to $n/b$

{read block $A(i,k)$ into fast memory}

{read block $B(k,j)$ into fast memory}

$C(i,j) = C(i,j) + A(i,k) \times B(k,j)$ \{do a matrix multiply on blocks\}

{write block $C(i,j)$ back to slow memory}
Consider A, B, C to be \( \frac{n}{b} \)-by-\( \frac{n}{b} \) matrices of \( b \)-by-\( b \) subblocks where \( b \) is called the block size; assume 3 \( b \)-by-\( b \) blocks fit in fast memory

for \( i = 1 \) to \( \frac{n}{b} \)

for \( j = 1 \) to \( \frac{n}{b} \)

\{read block \( C(i,j) \) into fast memory\} \( \ldots b^2 \times (\frac{n}{b})^2 = n^2 \) reads

for \( k = 1 \) to \( \frac{n}{b} \)

\{read block \( A(i,k) \) into fast memory\} \( \ldots b^2 \times (\frac{n}{b})^3 = n^3/b \) reads

\{read block \( B(k,j) \) into fast memory\} \( \ldots b^2 \times (\frac{n}{b})^3 = n^3/b \) reads

\( C(i,j) = C(i,j) + A(i,k) \times B(k,j) \) \{do a matrix multiply on blocks\}

\{write block \( C(i,j) \) back to slow memory\} \( \ldots b^2 \times (\frac{n}{b})^2 = n^2 \) writes

\( 2n^3/b + 2n^2 \) reads/writes \( \ll 2n^3 \) arithmetic \( - \) Faster!
Does blocked matmul attain lower bound?

- Recall: if 3 b-by-b blocks fit in fast memory of size $M$, then $\text{reads/writes} = 2n^3/b + 2n^2$
- Make $b$ as large as possible: $3b^2 \leq M$, so $\text{reads/writes} \geq 3^{1/2}n^3/M^{1/2} + 2n^2$
- Attains lower bound $= \Omega (\text{flops} / M^{1/2} )$

- But what if we don’t know $M$?
- Or if there are multiple levels of fast memory?
- How do we write the algorithm?
Recursive Matrix Multiplication (RMM) (1/2)

- For simplicity: square matrices with $n = 2^m$
- $C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = A \cdot B = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$

  $= \begin{pmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{pmatrix}$

- True when each $A_{ij}$ etc 1x1 or $n/2 \times n/2$

```
func C = RMM (A, B, n)
if n = 1, C = A * B, else
    \{ C_{11} = RMM (A_{11}, B_{11}, n/2) + RMM (A_{12}, B_{21}, n/2) \\
    C_{12} = RMM (A_{11}, B_{12}, n/2) + RMM (A_{12}, B_{22}, n/2) \\
    C_{21} = RMM (A_{21}, B_{11}, n/2) + RMM (A_{22}, B_{21}, n/2) \\
    C_{22} = RMM (A_{21}, B_{12}, n/2) + RMM (A_{22}, B_{22}, n/2) \}
return
```
Recursive Matrix Multiplication (RMM) (2/2)

```python
func C = RMM (A, B, n)
    if n=1, C = A * B, else
    {  C_{11} = RMM (A_{11}, B_{11}, n/2) + RMM (A_{12}, B_{21}, n/2)
        C_{12} = RMM (A_{11}, B_{12}, n/2) + RMM (A_{12}, B_{22}, n/2)
        C_{21} = RMM (A_{21}, B_{11}, n/2) + RMM (A_{22}, B_{21}, n/2)
        C_{22} = RMM (A_{21}, B_{12}, n/2) + RMM (A_{22}, B_{22}, n/2)  }
    return
```

A(n) = # arithmetic operations in RMM( . , . , n)
      = 8 \cdot A(n/2) + 4(n/2)^2  \text{ if } n > 1, \text{ else } 1
      = 2n^3  \text{ … same operations as usual, in different order}

W(n) = # words moved between fast, slow memory by RMM( . , . , n)
      = 8 \cdot W(n/2) + 12(n/2)^2  \text{ if } 3n^2 > M, \text{ else } 3n^2
      = O( n^3 / M^{1/2} + n^2 )  \text{ … same as blocked matmul}

“Cache oblivious”, works for memory hierarchies, but not panacea
How hard is hand-tuning matmul, anyway?

- Results of 22 student teams trying to tune matrix-multiply, in CS267 Spr09
- Students given “blocked” code to start with (7x faster than naïve)
- Still hard to get close to vendor tuned performance (ACML) (another 6x)
- For more discussion, see www.cs.berkeley.edu/~volkov/cs267.sp09/hw1/results/
How hard is hand-tuning matmul, anyway?
SUMMA— n x n matmul on P^{1/2} x P^{1/2} grid (nearly) optimal using minimum memory M=O(n^2/P)

For k=0 to n/b-1  
  ... b = block size = #cols in A(i,k) = #rows in B(k,j)  
  for all i = 1 to P^{1/2}  
    owner of A(i,k) broadcasts it to whole processor row (using binary tree)  
  for all j = 1 to P^{1/2}  
    owner of B(k,j) broadcasts it to whole processor column (using bin. tree)  
  Receive A(i,k) into Acol  
  Receive B(k,j) into Brow  
  C_myproc = C_myproc + Acol * Brow
Summary of dense \textit{parallel} algorithms attaining communication lower bounds

- Assume nxn matrices on P processors
- Minimum Memory per processor = $M = O(n^2 / P)$
- Recall lower bounds:
  \begin{align*}
  \#\text{words}_\text{moved} &= \Omega\left( \frac{n^3}{P} / M^{1/2} \right) = \Omega\left( \frac{n^2}{P^{1/2}} \right) \\
  \#\text{messages} &= \Omega\left( \frac{n^3}{P} / M^{3/2} \right) = \Omega\left( P^{1/2} \right)
  \end{align*}
- Does ScALAPACK attain these bounds?
  - For $\#\text{words}_\text{moved}$: mostly, except nonsym. Eigenproblem
  - For $\#\text{messages}$: asymptotically worse, except Cholesky
- New algorithms attain all bounds, up to polylog(P) factors
  - Cholesky, LU, QR, Sym. and Nonsym eigenproblems, SVD

Can we do Better?
Can we do better?

• Aren’t we already optimal?
• Why assume $M = O(n^2/p)$, i.e. minimal?
  – Lower bound still true if more memory
  – Can we attain it?
• Special case: “3D Matmul”
  – Uses $M = O(n^2/p^{2/3})$
  – Dekel, Nassimi, Sahni [81], Bernsten [89],
     Agarwal, Chandra, Snir [90], Johnson [93],
     Agarwal, Balle, Gustavson, Joshi, Palkar [95]
  – Processors arranged in $p^{1/3} \times p^{1/3} \times p^{1/3}$ grid
  – Processor $(i,j,k)$ performs $C(i,j) = C(i,j) + A(i,k) \times B(k,j)$,
     where each submatrix is $n/p^{1/3} \times n/p^{1/3}$
• Not always that much memory available…
Outline

• Survey state of the art of CA (Comm-Avoiding) algorithms
  – Review previous Matmul algorithms
  – CA $O(n^3)$ 2.5D Matmul
  – TSQR: Tall-Skinny QR
  – CA $O(n^3)$ 2.5D LU
  – CA Strassen Matmul

• Beyond linear algebra
  – Extending lower bounds to any algorithm with arrays
  – Communication-optimal N-body algorithm

• CA-Krylov methods
2.5D Matrix Multiplication

- Assume can fit $cn^2/P$ data per processor, $c > 1$
- Processors form $(P/c)^{1/2} \times (P/c)^{1/2} \times c$ grid

Example: $P = 32, c = 2$
2.5D Matrix Multiplication

- Assume can fit $cn^2/P$ data per processor, $c > 1$
- Processors form $(P/c)^{1/2} \times (P/c)^{1/2} \times c$ grid

Initially $P(i,j,0)$ owns $A(i,j)$ and $B(i,j)$ each of size $n(c/P)^{1/2} \times n(c/P)^{1/2}$

1. $P(i,j,0)$ broadcasts $A(i,j)$ and $B(i,j)$ to $P(i,j,k)$
2. Processors at level $k$ perform $1/c$-th of SUMMA, i.e. $1/c$-th of $\sum_m A(i,m) \times B(m,j)$
3. Sum-reduce partial sums $\sum_m A(i,m) \times B(m,j)$ along $k$-axis so $P(i,j,0)$ owns $C(i,j)$
2.5D Matmul on BG/P, 16K nodes / 64K cores

Matrix multiplication on 16,384 nodes of BG/P

Using $c=16$ matrix copies

- 2D MM
- 2.5D MM

- 12X faster
- 2.7X faster

Percentage of machine peak

$n$ vs. $n$
2.5D Matmul on BG/P, 16K nodes / 64K cores

c = 16 copies

Matrix multiplication on 16,384 nodes of BG/P

95% reduction in comm

Distinguished Paper Award, EuroPar’11 (Solomonik, D.)
SC’11 paper by Solomonik, B hatele, D.
Perfect Strong Scaling – in Time and Energy (1/2)

- Every time you add a processor, you should use its memory $M$ too
- Start with minimal number of procs: $PM = 3n^2$
- Increase $P$ by a factor of $c \Rightarrow$ total memory increases by a factor of $c$
- Notation for timing model:
  - $\gamma_T, \beta_T, \alpha_T =$ secs per flop, per word_moved, per message of size $m$
- $T(cP) = n^3/(cP) \left[ \gamma_T + \beta_T/M^{1/2} + \alpha_T/(mM^{1/2}) \right]$
  = $T(P)/c$
- Notation for energy model:
  - $\gamma_E, \beta_E, \alpha_E =$ joules for same operations
  - $\delta_E =$ joules per word of memory used per sec
  - $\varepsilon_E =$ joules per sec for leakage, etc.
- $E(cP) = cP \left\{ n^3/(cP) \left[ \gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2}) \right] + \delta_E MT(cP) + \varepsilon_E T(cP) \right\}$
  = $E(P)$
Perfect Strong Scaling — in Time and Energy (2/2)

- \( T(cP) = \frac{n^3}{(cP)} \left[ \gamma_T + \frac{\beta_T}{M^{1/2}} + \frac{\alpha_T}{(mM^{1/2})} \right] = \frac{T(P)}{c} \)
- \( E(cP) = cP \left\{ \frac{n^3}{(cP)} \left[ \gamma_E + \frac{\beta_E}{M^{1/2}} + \frac{\alpha_E}{(mM^{1/2})} \right] + \delta_E MT(cP) + \epsilon_E T(cP) \right\} = E(P) \)

- Perfect scaling extends to N-body, Strassen, ...
- We can use these models to answer many questions, including:
  - What is the minimum energy required for a computation?
  - Given a maximum allowed runtime \( T \), what is the minimum energy \( E \) needed to achieve it?
  - Given a maximum energy budget \( E \), what is the minimum runtime \( T \) that we can attain?
  - The ratio \( P = \frac{E}{T} \) gives us the average power required to run the algorithm. Can we minimize the average power consumed?
  - Given an algorithm, problem size, number of processors and target energy efficiency (GFLOPS/W), can we determine a set of architectural parameters to describe a conforming computer architecture?
Outline

• Survey state of the art of CA (Comm-Avoiding) algorithms
  – Review previous Matmul algorithms
  – CA $O(n^3)$ 2.5D Matmul
  – TSQR: Tall-Skinny QR
  – CA $O(n^3)$ 2.5D LU
  – CA Strassen Matmul
• Beyond linear algebra
  – Extending lower bounds to any algorithm with arrays
  – Communication-optimal N-body algorithm
• CA-Krylov methods
TSQR: QR of a Tall, Skinny matrix

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

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

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

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

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

\[ \begin{pmatrix} R_{01} \\ R_{11} \end{pmatrix} = \begin{pmatrix} Q_{02} & R_{02} \end{pmatrix} \]

Output = \{ Q_{00}, Q_{10}, Q_{20}, Q_{30}, Q_{01}, Q_{11}, Q_{02}, R_{02} \}
TSQR: An Architecture-Dependent Algorithm

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: \( W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow R_{00} \rightarrow R_{01} \rightarrow R_{01} \rightarrow R_{02} \rightarrow R_{03} \)

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)
  - Tesla C 2050 / Fermi
    - Up to $13x$ (110,592 x 100)
- Grid – $4x$ on 4 cities vs 1 city (Dongarra, Langou et al)
- Cloud – **1.6x slower than accessing data twice** (Gleich and Benson)

- **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

- SVD costs about the same
- Joint work with Grigori, Hoemmen, Langou, Anderson, Ballard, Keutzer, others
Outline

• Survey state of the art of CA (Comm-Avoiding) algorithms
  – Review previous Matmul algorithms
  – CA $O(n^3)$ 2.5D Matmul
  – TSQR: Tall-Skinny QR
  – CA $O(n^3)$ 2.5D LU
  – CA Strassen Matmul

• Beyond linear algebra
  – Extending lower bounds to any algorithm with arrays
  – Communication-optimal N-body algorithm

• CA-Krylov methods
Back to LU: Using similar idea for TSLU as TSQR: Use reduction tree, to do “Tournament Pivoting”

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

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
  - Extra work, but lower order term
- Thm: As numerically stable as Partial Pivoting on a larger matrix
Exascale Machine Parameters
Source: DOE Exascale Workshop

- $2^{20} \approx 1,000,000$ 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 Gaussian Elimination: 2D CA-LU vs ScaLAPACK-LU

\[
\log_2 \left( \frac{n^2}{p} \right) = \log_2 \left( \frac{\text{memory\ per\ proc}}{p} \right)
\]
2.5D vs 2D LU
With and Without Pivoting

LU on 16,384 nodes of BG/P (n=131,072)

Time (sec)

communication
idle
compute

NO-pivot 2D
NO-pivot 2.5D
CA-pivot 2D
CA-pivot 2.5D

2X faster
Ongoing Work

• Lots more work on
  – Algorithms:
    • BLAS, LDL$^T$, QR with pivoting, other pivoting schemes, eigenproblems, ...
    • All-pairs-shortest-path, ...
    • Both 2D (c=1) and 2.5D (c>1)
    • But only bandwidth may decrease with c>1, not latency
  – Platforms:
    • Multicore, cluster, GPU, cloud, heterogeneous, low-energy, ...
  – Software:
    • Integration into Sca/LAPACK, PLASMA, MAGMA,...

• Integration into applications (on IBM BG/Q)
  – CTF (with ANL): symmetric tensor contractions
Outline

• Survey state of the art of CA (Comm-Avoiding) algorithms
  – Review previous Matmul algorithms
    – CA $O(n^3)$ 2.5D Matmul
    – TSQR: Tall-Skinny QR
    – CA $O(n^3)$ 2.5D LU
    – CA Strassen Matmul

• Beyond linear algebra
  – Extending lower bounds to any algorithm with arrays
  – Communication-optimal N-body algorithm

• CA-Krylov methods
Communication Lower Bounds for Strassen-like matmul algorithms

<table>
<thead>
<tr>
<th></th>
<th>Classical (O(n^3)) matmul:</th>
<th>Strassen’s (O(n^{\lg 7})) matmul:</th>
<th>Strassen-like (O(n^\omega)) matmul:</th>
</tr>
</thead>
<tbody>
<tr>
<td>#wordsMoved</td>
<td>(\Omega(M(n/M^{1/2})^3/P))</td>
<td>(\Omega(M(n/M^{1/2})^{\lg 7}/P))</td>
<td>(\Omega(M(n/M^{1/2})^\omega/P))</td>
</tr>
</tbody>
</table>

- **Proof:** graph expansion (different from classical matmul)
  - Strassen-like: DAG must be “regular” and connected
- Extends up to \(M = n^2 / p^{2/\omega}\)
- **Best Paper Prize (SPAA’11), Ballard, D., Holtz, Schwartz,** also in JACM
- Is the lower bound attainable?
Communication Avoiding Parallel Strassen (CAPS)

**BFS**
- A • B
- Runs all 7 multiplies in parallel
- Each on P/7 processors
- Needs 7/4 as much memory

**DFS**
- A • B
- Runs all 7 multiplies sequentially
- Each on all P processors
- Needs 1/4 as much memory

**CAPS**

If EnoughMemory and P \( \geq 7 \)
then BFS step
else DFS step
end if

Best way to interleave BFS and DFS is a tuning parameter
Performance Benchmarking, Strong Scaling Plot
Franklin (Cray XT4) n = 94080

Speedups: 24%-184%
(over previous Strassen-based algorithms)

Invited to appear as Research Highlight in CACM
Outline

• Survey state of the art of CA (Comm-Avoiding) algorithms
  – Review previous Matmul algorithms
  – CA $O(n^3)$ 2.5D Matmul
  – TSQR: Tall-Skinny QR
  – CA $O(n^3)$ 2.5D LU
  – CA Strassen Matmul

• Beyond linear algebra
  – Extending lower bounds to any algorithm with arrays
  – Communication-optimal N-body algorithm

• CA-Krylov methods
Recall optimal sequential Matmul

• Naïve code
  for i=1:n, for j=1:n, for k=1:n, C(i,j) += A(i,k) * B(k,j)

• “Blocked” code
  for i1 = 1:b:n, for j1 = 1:b:n, for k1 = 1:b:n
     for i2 = 0:b-1, for j2 = 0:b-1, for k2 = 0:b-1
        i = i1+i2, j = j1+j2, k = k1+k2
        C(i,j) += A(i,k) * B(k,j)

• Thm: Picking $b = M^{1/2}$ attains lower bound:
  $\#\text{words}\_\text{moved} = \Omega(n^3/M^{1/2})$
• Where does $1/2$ come from?
New Thm applied to Matmul

• for i=1:n, for j=1:n, for k=1:n, C(i,j) += A(i,k)*B(k,j)
• Record array indices in matrix Δ

\[
\begin{pmatrix}
i & j & k \\
1 & 0 & 1 \\
0 & 1 & 1 \\
1 & 1 & 0
\end{pmatrix}
\]

• Solve LP for \( x = [x_i, x_j, x_k]^T \): max \( 1^T x \) s.t. \( \Delta x \leq 1 \)
  – Result: \( x = [1/2, 1/2, 1/2]^T \), \( 1^T x = 3/2 = s_{HBL} \)
• Thm: \#words_moved = \( \Omega(n^3/M^{S_{HBL}-1}) = \Omega(n^3/M^{1/2}) \)
  Attained by block sizes \( M^{x_i}, M^{x_j}, M^{x_k} = M^{1/2}, M^{1/2}, M^{1/2} \)
New Thm applied to Direct N-Body

• for i=1:n, for j=1:n, F(i) += force(P(i), P(j))

• Record array indices in matrix Δ

\[
\Delta = \begin{pmatrix}
1 & 0 \\
1 & 0 \\
0 & 1 \\
\end{pmatrix}
\]

• Solve LP for \( x = [xi,xj]^T \): max \( 1^T x \) s.t. \( \Delta x \leq 1 \)
  
  — Result: \( x = [1,1], 1^T x = 2 = s_{HBL} \)

• Thm: \#words Moved = \( \Omega(n^2/M^{S_{HBL}-1}) = \Omega(n^2/M^1) \)

  Attained by block sizes \( M^{xi}, M^{xj} = M^1, M^1 \)
N-Body Speedups on IBM-BG/P (Intrepid)
8K cores, 32K particles

K. Yelick, E. Georganas, M. Driscoll, P. Koanantakool, E. Solomonik
New Thm applied to Random Code

- for \(i1=1:n\), for \(i2=1:n\), ..., for \(i6=1:n\)
  
  \[ A1(i1,i3,i6) += \text{func1}(A2(i1,i2,i4),A3(i2,i3,i5),A4(i3,i4,i6)) \]
  
  \[ A5(i2,i6) += \text{func2}(A6(i1,i4,i5),A3(i3,i4,i6)) \]

- Record array indices in matrix \(\Delta\)

\[
\begin{pmatrix}
1 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 \\
\end{pmatrix}
\]

- Solve LP for \(x = [x1,\ldots,x7]^T\): \(\max 1^T x \) s.t. \(\Delta x \leq 1\)
  
  - Result: \(x = [2/7,3/7,1/7,2/7,3/7,4/7]\), \(1^T x = 15/7 = S_{\text{HBL}}\)

- Thm: \#words\_moved = \(\Omega(n^6/M_{\text{HBL}}^{\text{SHBL}-1}) = \Omega(n^6/M^{8/7})\)

Attained by block sizes \(M^{2/7}, M^{3/7}, M^{1/7}, M^{2/7}, M^{3/7}, M^{4/7}\)
Where do lower and matching upper bounds on communication come from? (1/3)

• Originally for $C = A*B$ by Irony/Tiskin/Toledo (2004)

• Proof idea
  – Suppose we can bound $\#\text{useful\_operations} \leq G$ doable with data in fast memory of size $M$
  – So to do $F = \#\text{total\_operations}$, need to fill fast memory $F/G$ times, and so $\#\text{words\_moved} \geq MF/G$

• Hard part: finding $G$

• Attaining lower bound
  – Need to “block” all operations to perform $\sim G$ operations on every chunk of $M$ words of data
Proof of communication lower bound (2/3)

"C face"

Cube representing
\( \text{C}(1,1) \equiv \text{A}(1,3) \cdot \text{B}(3,1) \)

"A face"

• If we have at most \( M \) “A squares”, \( M \) “B squares”, and \( M \) “C squares”, how many cubes \( G \) can we have? 69
Proof of communication lower bound (3/3)

\[
G = \# \text{ cubes in black box with side lengths } x, y \text{ and } z = \text{Volume of black box} = x \cdot y \cdot z = (x z \cdot z y \cdot y x)^{1/2} = (\#A\square s \cdot \#B\square s \cdot \#C\square s)^{1/2} \leq M^{3/2}
\]

(i,k) is in “A shadow” if (i,j,k) in 3D set
(j,k) is in “B shadow” if (i,j,k) in 3D set
(i,j) is in “C shadow” if (i,j,k) in 3D set

Thm (Loomis & Whitney, 1949)
\[
G = \# \text{ cubes in 3D set} = \text{Volume of 3D set} \leq \text{area}(A \text{ shadow}) \cdot \text{area}(B \text{ shadow}) \cdot \text{area}(C \text{ shadow})^{1/2} \leq M^{3/2}
\]
Approach to generalizing lower bounds

- Matmul
  for \(i=1:n,\) for \(j=1:n,\) for \(k=1:n,\)
  \[C(i,j) += A(i,k) * B(k,j)\]
  \[\Rightarrow\]
  for \((i,j,k)\) in \(S = \text{subset of } Z^3\)
  Access locations indexed by \((i,j), (i,k), (k,j)\)

- General case
  for \(i_1=1:n,\) for \(i_2 = i_1:m,\) ... for \(i_k = i_3:i_4\)
  \[C(i_1+2*i_3-i_7) = \text{func}(A(i_2+3*i_4,i_1,i_2,i_1+i_2,...)B(\text{pnt}(3*i_4)),...)\]
  \[D(\text{something else}) = \text{func}(\text{something else}), \ldots\]

  \[\Rightarrow\]
  for \((i_1,i_2,...,i_k)\) in \(S = \text{subset of } Z^k\)
  Access locations indexed by group homomorphisms, eg
  \[\phi_C(i_1,i_2,...,i_k) = (i_1+2*i_3-i_7)\]
  \[\phi_A(i_1,i_2,...,i_k) = (i_2+3*i_4,i_1,i_2,i_1+i_2,...), \ldots\]

- Can we bound \#loop_iterations \((= |S|)\)
  given bounds on \#points in its images, i.e. bounds on \(|\phi_C(S)|, |\phi_A(S)|, \ldots| ?\]
General Communication Bound

- Given $S$ subset of $\mathbb{Z}^k$, group homomorphisms $\phi_1, \phi_2, \ldots$, bound $|S|$ in terms of $|\phi_1(S)|$, $|\phi_2(S)|$, $\ldots$, $|\phi_m(S)|$
- Def: Hölder-Brascamp-Lieb LP (HBL-LP) for $s_1, \ldots, s_m$:
  for all subgroups $H < \mathbb{Z}^k$, $\text{rank}(H) \leq \sum_j s_j \cdot \text{rank}(\phi_j(H))$
- Thm (Christ/Tao/Carbery/Bennett): Given $s_1, \ldots, s_m$
  $$|S| \leq \prod_j |\phi_j(S)|^{s_j}$$
- Thm: Given a program with array refs given by $\phi_j$, choose $s_j$ to minimize $s_{\text{HBL}} = \sum_j s_j$ subject to HBL-LP. Then
  $$\#\text{words\_moved} = \Omega (\#\text{iterations}/M^{s_{\text{HBL}}^{-1}})$$
Is this bound attainable (1/2)?

• But first: Can we write it down?
• Thm: (bad news) HBL-LP reduces to Hilbert’s 10th problem over Q (conjectured to be undecidable)
• Thm: (good news) Another LP with same solution is decidable (but expensive, so far)
• Thm: (better news) Easy to write down LP explicitly in many cases of interest (eg all $\phi_j = \{\text{subset of indices}\})
• Thm: (good news) Easy to approximate, i.e. get upper or lower bounds on $s_{HBL}$
Is this bound attainable (2/2)?

• Depends on loop dependencies
• Best case: none, or reductions (matmul)
• Thm: When all \( \phi_j = \{\text{subset of indices}\} \), dual of HBL-LP gives optimal tile sizes:

  \[
  \text{HBL-LP: } \text{minimize } 1^T s \quad \text{s.t. } s^T \Delta \geq 1^T \\
  \text{Dual-HBL-LP: } \text{maximize } 1^T x \quad \text{s.t. } \Delta^* x \leq 1
  \]

Then for sequential algorithm, tile \( i_j \) by \( M^{x_j} \)

• Ex: Matmul: \( s = [1/2, 1/2, 1/2]^T = x \)
• Extends to unimodular transforms of indices
Ongoing Work

• Accelerate decision procedure for lower bounds
  – Ex: At most 3 arrays, or 4 loop nests
• Have yet to find a case where we cannot attain lower bound – can we prove this?
• Extend “perfect scaling” results for time and energy by using extra memory
  – “n.5D algorithms”
• Incorporate into compilers
Outline

• Survey state of the art of CA (Comm-Avoiding) algorithms
  – Review previous Matmul algorithms
  – CA $O(n^3)$ 2.5D Matmul
  – TSQR: Tall-Skinny QR
  – CA $O(n^3)$ 2.5D LU
  – CA Strassen Matmul

• Beyond linear algebra
  – Extending lower bounds to any algorithm with arrays
  – Communication-optimal N-body algorithm

• CA-Krylov methods
Avoiding Communication in Iterative Linear Algebra

- k-steps of iterative solver for sparse $Ax=b$ or $Ax=\lambda x$
  - Does $k$ SpMV's 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
  - Challenges: Poor partitioning, Preconditioning, Num. Stability
Communication Avoiding Kernels:
The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

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

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

• Example: A tridiagonal, \(n=32\), \(k=3\)

• Works for any “well-partitioned” \(A\)
Communication Avoiding Kernels:
The Matrix Powers Kernel: \([Ax, A^2x, ..., A^kx]\)

- 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: \([Ax, A^2x, ..., A^kx]\)

- 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 : $[Ax, A^2x, ..., A^kx]$

- 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: \([Ax, A^2x, ..., A^kx]\)

- 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: \([Ax, A^2x, \ldots, A^kx]\)

- 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 \([Ax, A^2x, \ldots, A^kx]\)
- Parallel Algorithm

- Example: A tridiagonal, \(n=32\), \(k=3\)
- Each processor communicates once with neighbors
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: \([Ax, A^2x, \ldots, A^kx]\)

Same idea works for general sparse matrices

Simple block-row partitioning ➔ (hyper)graph partitioning

Top-to-bottom processing ➔ Traveling Salesman Problem
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 $k$
  
  $w = A \cdot v(i-1)$ ... *SpMV*
  
  MGS($w$, $v(0), \ldots, v(i-1)$)
  
  update $v(i)$, $H$
  
  endfor
  
  solve LSQ problem with $H$

  **Communication-avoiding GMRES**
  
  $W = [ v, Av, A^2 v, \ldots, A^k v ]$
  
  $[Q,R] = \text{TSQR}(W)$
  
  ... *“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!**
“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]
Compute $r_0 = b - Ax_0$. Choose $r_0^*$ arbitrary.
Set $p_0 = r_0$, $q_{-1} = 0_{N \times 1}$.
For $k = 0, 1, \ldots$, until convergence, Do

$$P = [p_{sk}, A p_{sk}, \ldots, A^s p_{sk}]$$
$$Q = [q_{sk-1}, A q_{sk-1}, \ldots, A^s q_{sk-1}]$$
$$R = [r_{sk}, A r_{sk}, \ldots, A^s r_{sk}]$$

//Compute the $1 \times (3s + 3)$ Gram vector.
$$g = (r_0^*)^T [P, Q, R]$$

//Compute the $(3s + 3) \times (3s + 3)$ Gram matrix
$$G = \begin{bmatrix} P^T \\ Q^T \\ R^T \end{bmatrix} \begin{bmatrix} P & Q & R \end{bmatrix}$$

For $\ell = 0$ to $s$,
$$b_{sk}^\ell = \begin{bmatrix} B_1 (:, \ell)^T, 0_{s+1}^T, 0_{s+1}^T \end{bmatrix}^T$$
$$c_{sk-1}^\ell = \begin{bmatrix} 0_{s+1}^T, B_2 (:, \ell)^T, 0_{s+1}^T \end{bmatrix}^T$$
$$d_{sk}^\ell = \begin{bmatrix} 0_{s+1}^T, 0_{s+1}^T, B_3 (:, \ell)^T \end{bmatrix}^T$$

1. Compute $r_0 := b - Ax_0; r_0^*$ arbitrary;
2. $p_0 := r_0$.
3. For $j = 0, 1, \ldots$, until convergence Do:
   4. $\alpha_j := (r_j, r_0^*) / (Ap_j, r_0^*)$
   5. $s_j := r_j - \alpha_j Ap_j$
   6. $\omega_j := As_j / (As_j, As_j)$
   7. $x_{j+1} := x_j + \alpha_j p_j + \omega_j s_j$
   8. $r_{j+1} := s_j - \omega_j As_j$
   9. $\beta_j := \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\omega_j}$
   10. $p_{j+1} := r_{j+1} + \beta_j (p_j - \omega_j Ap_j)$
   11. EndDo

CA-BiCGStab

For $j = 0$ to $\left\lfloor \frac{s}{2} \right\rfloor - 1$, Do
$$\alpha_{sk+j} = \frac{<g, d_{sk+j}^0>}{<g, b_{sk+j}^0>}$$
$$q_{sk+j} = r_{sk+j} - \alpha_{sk+j}[P, Q, R]b_{sk+j}^1$$
For $\ell = 0$ to $s - 2j + 1$, Do
$$c_{sk+j}^\ell = d_{sk+j}^\ell - \alpha_{sk+j}b_{sk+j}^{\ell+1}$$
//such that $[P, Q, R]c_{sk+j}^\ell = A^\ell q_{sk+j}$
$$\omega_{sk+j} = \frac{<c_{sk+j}^{\ell+1}, G c_{sk+j+1}^0>}{<c_{sk+j+1}^0, G c_{sk+j+1}^0>}$$
$$x_{sk+j+1} = x_{sk+j} + \alpha_{sk+j} P s_{sk+j} + \omega_{sk+j} q_{sk+j}$$
$$r_{sk+j+1} = q_{sk+j} - \omega_{sk+j}[P, Q, R]c_{sk+j+1}^1$$
For $\ell = 0$ to $s - 2j$, Do
$$d_{sk+j+1}^\ell = c_{sk+j+1}^\ell - \omega_{sk+j} c_{sk+j+1}^{\ell+1}$$
//such that $[P, Q, R]d_{sk+j+1}^\ell = A^\ell r_{sk+j+1}$
$$\beta_{sk+j} = \frac{<g, d_{sk+j+1}^0>}{<g, b_{sk+j}^0>} \times \frac{\alpha_j}{\omega_j}$$
$$p_{sk+j+1} = r_{sk+j+1} + \beta_{sk+j} p_{sk+j} - \beta_{sk+j} \omega_{sk+j} b_{sk+j}^1$$
For $\ell = 0$ to $s - 2j$, Do
$$b_{sk+j+1}^\ell = d_{sk+j+1}^\ell + \beta_{sk+j} b_{sk+j}^{\ell+1} - \beta_{sk+j} \omega_{sk+j} b_{sk+j}^{\ell+1}$$
//such that $[P, Q, R]b_{sk+j+1}^\ell = A^\ell p_{sk+j+1}$.
EndDo
EndDo
CA-BICGSTAB Convergence, $s = 32$

- **Exact Residual (2-norm)**

- **Iteration**

- **Lines and Legends**:
  - Monomial
  - Newton
  - Chebyshev
  - Naive
With Residual Replacement (RR) a la Van der Vorst and Ye

CA-BICGSTAB Convergence, $s = 32$

<table>
<thead>
<tr>
<th>Replacement Its.</th>
<th>Naive</th>
<th>Monomial</th>
<th>Newton</th>
<th>Chebyshev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>74 (1)</td>
<td>[7, 15, 24, 31, ..., 92, 97, 103] (17)</td>
<td>[67, 98] (2)</td>
<td>68 (1)</td>
</tr>
</tbody>
</table>
Summary of Iterative Linear Algebra

• New lower bounds, optimal algorithms, big speedups in theory and practice

• Lots of other progress, open problems
  – Many different algorithms reorganized
    • More underway, more to be done
  – Need to recognize stable variants more easily
  – Preconditioning
    • Hierarchically Semiseparable Matrices
  – Autotuning and synthesis
    • Different kinds of “sparse matrices”
Why avoid communication?

• Communication = moving data
  – Between levels of memory hierarchy, between processors over network, ...

• 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

• Avoid communication to save time
Avoiding Communication in Iterative Linear Algebra

- **k-steps of iterative solver for sparse $Ax=b$ or $Ax=\lambda x$**
  - Does $k$ SpMV 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
  - Challenges: Poor partitioning, Preconditioning, Num. Stability
For more details

• Bebop.cs.berkeley.edu
• CS267 – Berkeley’s Parallel Computing Course
  – Live broadcast in Spring 2013
    • [www.cs.berkeley.edu/~demmel](http://www.cs.berkeley.edu/~demmel)
    • All slides, video available
  – Prerecorded version broadcast in Spring 2013
    • [www.xsede.org](http://www.xsede.org)
    • Free supercomputer accounts to do homework
    • Free autograding of homework