Communication-avoidance and Automatic performance tuning

Nick Knight
UC-Berkeley Parlab (Bebop group)
Students and Collaborators (Bebop)

- **Pls**
  - James Demmel
  - Kathy Yelick

- **Students**
  - Marghoob Mohiyuddin
  - Shoaib Kamil
  - Vasily Volkov
  - Andrew Gearheart
  - Razvan Carbunescu
  - Grey Ballard
  - Nick Knight
  - Erin Carson
  - Edgar Solomonik
  - Michael Anderson
  - Gilad Arnold

- **Collaborators**
  - Mark Hoemmen (Sandia)
  - Mike Heroux (Sandia)
  - Jack Dongarra (UTK)
  - Laura Grigori (INRIA)
  - Simplice Donfack (INRIA)
  - Amal Khabou (INRIA)
  - Oded Schwartz (TU-Berlin + UCB)
  - Olga Holtz (UCB)
  - Ming Gu (UCB)
  - Ioana Dumitriu (UW)
  - Julien Langou (UC-Denver)
  - Samuel Williams (LBNL)
  - Aydin Buluc (LBNL)
  - Kamesh Madduri (LBNL)
  - Jong-Ho Byun (UCB)
  - Armando Fox (UCB)
  - Ras Bodik (UCB)
  - Tony Keaveny (UCB)
Talk outline

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

- Automatic performance tuning
  - OSKI (Optimized Sparse Kernel Interface)
  - Future development
Communication =

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

Comm. lower bounds for direct linear algebra:

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

We develop:

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

Speedups on today’s, future hardware
Talk outline

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

- **Automatic performance tuning**
  - OSKI (Optimized Sparse Kernel Interface)
  - Future development
Heterogeneous comm. complexity

Heterogeneous memory model
(Different speeds, bandwidths, latencies)

How best to reduce communication costs on heterogeneous machine?

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

- New theory \(\rightarrow\) FASTER ALGORITHMS

(SEE POSTER)
Gearhart, Ballard
Applications:
- Image processing (M. Anderson’s talk yesterday)

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

New algorithm for successive band-reduction

- Minimizes bandwidth \textit{and} latency costs in serial
- Parallel SBR and implementations in development
2.5D algorithms I

2D processor array

1 copy of data

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

2.5D processor array

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

$1 \leq c \leq \frac{P}{c^{1/3}}$ copies of data, one per `plane' in the $k$ direction
Do you have extra memory available?

In direct linear algebra, we can use it to avoid communication by storing $1 \leq c \leq P^{1/3}$ redundant copies of data.

New theory….

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

→ Tune to find optimal duplication factor $c$

…. leads to faster algorithms

- New 2.5D matrix-matrix multiplication algorithm
  - Predict up to 5x speedups on future exascale hardware
  - using up to $c=100$ copies
- Also new 2.5D LU algorithm …
Direct linear algebra does $\Theta(n^\omega)$ flops.

Conventional algorithms: $\omega = 3$

Strassen-like algorithms: $\omega < 3$

eg: $\omega \approx 2.81$ (Strassen’s matmul)

New theory ...

- Strassen-like matmuls can communicate less than conventional matmuls.

  - # words moved decreases: $\Omega\left(\frac{n^3}{M^{1/2}}\right) = \Omega\left(M \left(\frac{n}{M^{1/2}}\right)^3\right) \rightarrow \Omega\left(M \left(\frac{n}{M^{1/2}}\right)^\omega\right)$

  - # messages $M$ times smaller

... leads to new algorithms

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

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

“CA-BiCG”

- Previously seen in poster, summer 2010 retreat
  - BiCG solves sparse, nonsymmetric systems of equations
  - CA-BiCG mathematically-equivalent formulation; takes multiple iterations with communication cost of one iteration of BiCG.

- BiCG vs. GMRES
  - GMRES: faster convergence and more stable, in theory and in practice
  - BiCG: small, constant-size workspace, less work per iteration
  - Stabilized variants (eg, BiCGStab) used in practice

- New communication-avoiding algorithms:
  - BiCG (2-term), CGS, BiCGStab, BiCGStab(l)
• Reduce communication by factor of s
• CA-BiCGStab follows convergence of standard BiCGStab, even with s=10
  • Hard problem! BiCGStab fails to converge in n its.
    • Preconditioning needed

(SEE POSTER)
Knight, Carson
Communication-avoiding generalized minimum residual method
“CA-GMRES”

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

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

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

(SEE POSTER)
Carson, Knight
Talk outline

Developing efficient algorithms

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

Automatic performance tuning

- OSKI (Optimized Sparse Kernel Interface)
- OSKI development
OSKI
(Optimized Sparse Kernel Interface)

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

Optimized Sparse Kernel Interface (OSKI):
Autotuned Sparse Matrix-Vector Multiplication (SpMV)

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

Efficient sparse codes are difficult to write
(SEE POSTER) Arnold, Bodik
OSKI (Optimized Sparse Kernel Interface)

How OSKI tunes:

Library Install-Time (offline)  →  Application Run-Time

1. Build for Target Arch.
2. Benchmark

Generated code variants
Benchmark data

1. Evaluate Models
2. Select Data Struct. & Code

Matrix
Workload from program monitoring
History
Heuristic models

To user: Matrix handle for kernel calls

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

Algorithm design space for next (p)OSKI release:

- Index compression
- Array padding
- Software prefetching
- Software pipelining
- Loop unrolling (depths)
- SpMM (multiple source vectors)
- Variable block splitting
- Switch-to-dense (SpTS)
- Cache interleaving ($A^T A$)
- Sparse tiling ($A^k x$)
- Symmetric storage for multicore
- SIMD intrinsics
- Data decomposition (shared/dist)
- NUMA awareness
- Hiding latency

- Reordering (RCM, TSP, …)
- TLB blocking
- Cache-blocking heuristics

Storage formats:
- CSB (compressed sparse block)
- Vector-style (manycore/GPU)
- DCSR (delta-coded CSR)
- RPCSR (row-pattern CSR)
- PBR (pattern-based repr.)
- RSDF (row-segmented diagonal fmt.)
Requested functionality (from HPC world):

- Change non-zero pattern of the matrix
  - Matrix may change or be perturbed during computation
- Assemble a matrix from (possibly overlapping) fragments
  - Common in finite element methods
- Perform variable block splitting
  - $A = A_1 + A_2$ where $A_1$ and $A_2$ have different natural block sizes

Our (proposed) solutions:

- List_of_matrices: allows a matrix to be expressed as a sum of matrices
  \( A = A_1 + \ldots + A_n \)
  - Easily allows for assembly from fragments and variable splitting
  - Pattern update: represent the changed entry as the addition of another matrix
- Merge() method: Merges the list $A_1 + \ldots + A_n$ into a single matrix
  - User can decide when to merge matrices, or…
  - In the future, merging may also be a tuning decision made by OSKI
Questions?
Summary of Performance Optimizations

- **Optimizations for SpMV**
  - **Register blocking (RB):** up to 4x over CSR
  - **Variable block splitting:** 2.1x over CSR, 1.8x over RB
  - **Diagonals:** 2x over CSR
  - **Reordering** to create dense structure + splitting: 2x over CSR
  - **Symmetry:** 2.8x over CSR, 2.6x over RB
  - **Cache blocking:** 2.8x over CSR
  - **Multiple vectors (SpMM):** 7x over CSR
  - And combinations…

- **Sparse triangular solve**
  - Hybrid sparse/dense data structure: 1.8x over CSR

- **Higher-level kernels**
  - **A·Aᵀ·x, Aᵀ·A·x:** 4x over CSR, 1.8x over RB
  - **A²·x:** 2x over CSR, 1.5x over RB
  - **[A·x, A²·x, A³·x, .., Aᵏ·x]**
Algorithms have two costs:
1. Arithmetic (flops)
2. Communication: moving data between
   • levels of a memory hierarchy (sequential)
   • processors (parallel)
     • messages (distributed mem)
     • cache-coherency (shared mem)
     • data transfers (bus-based)
Running time of an algorithm is sum of 3 terms:
- \# flops * time_per_flop
- \# words moved / bandwidth
- \# messages * latency

\[ \text{running time} = \text{flops} \times \text{time per flop} + \frac{\# \text{words moved}}{\text{bandwidth}} + \# \text{messages} \times \text{latency} \]

Time_per_flop \ll 1/\text{bandwidth} \ll \text{latency}
- Gaps growing exponentially with time (FOSC, 2004)

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

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

Why Avoid Communication?
Avoiding Communication

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

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

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

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

• Holds for:
  • BLAS, LU, QR, EVD/SVD, tensor contractions, …
  • Some whole programs (sequences of these operations, no matter how individual ops are interleaved, e.g., A^k)
  • Sequential and parallel algorithms
  • Some graph theoretic algorithms (e.g., Floyd-Warshall)
2.5D algorithms

- P processors: \(\# \text{ words moved} = \Omega\left(\frac{n^3/P}{M^{1/2}}\right)\) (intuition: make M bigger!)

- 2D algorithms: distribute matrices (2D arrays) across \(\sqrt{P} \times \sqrt{P}\) (logical) 2D grid of processors
  - If one copy of data, \(M = \frac{n^2}{P}\)

- What if you have extra memory?

- 3D algorithms: distribute matrices across \(P^{1/3} \times P^{1/3} \times P^{1/3}\) processor cube,
  - \(P^{1/3}\) duplicate copies of data (M increased by a factor of \(P^{1/3}\))
  - This decreases lower bounds for:
    - \# words moved by a factor of \(P^{1/6}\)
    - \# messages sent by a factor of \(P^{1/2}\)

- 2.5D algorithms:
  - \(1 \leq c \leq P^{1/3}\) copies of data: smooth transition between 2D to 3D bounds.
  - Flexibility

SEE POSTER
2.5D algorithms

Predicted exascale speedups

<table>
<thead>
<tr>
<th>log₂(n²/p)</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
<th>20</th>
<th>22</th>
<th>24</th>
<th>26</th>
<th>28</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Model Parameters:
- $1 \times 10^{18}$ flops/s (`exa-`)
- 24 PB total memory
- $2^{20}$ nodes $\rightarrow$ 24 GB/node
- 100 GB/s interconnect bandwidth (overestimate?)
- 100 ns network latency
2.5D algorithms

2.5D Best choice of c for efficiency

<table>
<thead>
<tr>
<th>log₂(p)</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Model Parameters:
- $1 \times 10^{18}$ flops/s (\textit{exa-})
- 24 PB total memory
- $2^{20}$ nodes $\rightarrow$ 24 GB/node
- 100 GB/s interconnect bandwidth (overestimate?)
- 100 ns network latency
## CA-BiCGStab convergence

### Convergence of CA Methods, dw2048 Matrix, \( s = 2 \)

<table>
<thead>
<tr>
<th>Name</th>
<th>( n )</th>
<th>NNZ</th>
<th>Pattern Symmetry</th>
<th>Value Symmetry</th>
<th>Condition Number</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>dw2048</td>
<td>2048</td>
<td>10114</td>
<td>No</td>
<td>No</td>
<td>5.3015e3</td>
<td>Electromagnetics Problem (H. Dong, 1993)</td>
</tr>
</tbody>
</table>
• M = fast memory size
• Conventional matrix-matrix multiplication (matmul)
  • # flops = $2n^3$
  • # words moved = $\Omega \left( \frac{n^3}{M^{1/2}} \right) = \Omega \left( M \left( \frac{n}{M^{1/2}} \right)^3 \right)$, attainable
  • # messages = $\Omega \left( \frac{n^3}{M^{3/2}} \right) = \Omega \left( \left( \frac{n}{M^{1/2}} \right)^3 \right)$, attainable
• Strassen’s matmul
  • # flops = $\Theta(n^\omega)$ where $\omega = \log_2(7) \approx 2.81$
  • # words moved = $\Omega \left( M \left( \frac{n}{M^{1/2}} \right)^\omega \right)$, attainable too
  • # messages = $\Omega \left( \left( \frac{n}{M^{1/2}} \right)^\omega \right)$, attainable too
• Applies to all other Fast matmul algorithms we know
• How broadly does it apply?
  • We know rest of linear algebra can be done in $O(n^\omega)$ flops and attain these #words moved and #messages, in serial.
  • If these are valid lower bounds, then they are tight