Communication-Avoiding Algorithms and Autotuning

Grey Ballard

BeBOP Group

May 30, 2013
### Where we fit in the Par Lab

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Embed</td>
<td>SPEC</td>
<td>DB</td>
<td>Games</td>
<td>ML</td>
<td>HPC</td>
<td>Health</td>
<td>Image</td>
<td>Speech</td>
<td>Music</td>
<td>Browser</td>
</tr>
</tbody>
</table>

Grey Ballard

BeBOP Group
Where we fit in the Par Lab

<table>
<thead>
<tr>
<th>Finite State Mach.</th>
<th>Embed</th>
<th>SPEC</th>
<th>DB</th>
<th>Games</th>
<th>ML</th>
<th>HPC</th>
<th>Health</th>
<th>Image</th>
<th>Speech</th>
<th>Music</th>
<th>Browser</th>
<th>CAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graph Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dense Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparse Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral (FFT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Prog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N-Body</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backtrack B&amp;B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphical Models</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unstructured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Algorithms have two kinds of costs

Measure computation in terms of \# flops performed

- Time per flop: $\gamma$

Measure communication in terms of \# words and \# messages communicated

- Time per word: $\beta$
- Time per message: $\alpha$

$\gamma \ll \beta \ll \alpha$ and the relative costs of communication are increasing
Algorithms have two kinds of costs

Measure computation in terms of \# flops performed

Time per flop: $\gamma$

Measure communication in terms of \# words and \# messages communicated

Time per word: $\beta$
Time per message: $\alpha$

Total running time of an algorithm (ignoring overlap):

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

$\gamma \ll \beta \ll \alpha$

and the relative costs of communication are increasing
Our goals are to

- Redesign algorithms to *avoid* communication
  - between all levels of memory hierarchy
    - L1 ↔ L2 ↔ DRAM ↔ network, etc.

- Attain lower bounds if possible
  - current algorithms often far from lower bounds
  - large speedups (and energy savings) possible

- Provide performance portability
  - using autotuning to map algorithms to various architectures

- Maintain productivity: deliver ideas to
  - libraries like (Sca)LAPACK and PLASMA/MAGMA and
  - frameworks like SEJITS
Communication lower bounds for linear algebra

Theorem

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

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

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

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

What smells like 3 nested loops? Most of dense and sparse linear algebra:

- Matrix multiplication, triangular solve, other “BLAS-3” computations
- Solving linear systems: Cholesky, LU, \( LDL^T \), \( LTL^T \) decompositions
- Solving least squares problems: QR decomposition
- Solving eigenvalue problems: eigenvalue and SVD reductions
- 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
Consider computing QR decomposition of $m \times n$ matrix with $m \gg n$

Standard Algorithm

accesses $O(mn)$ data $n$ times
Example comm-avoiding algorithm: tall-skinny QR

Consider computing QR decomposition of $m \times n$ matrix with $m \gg n$

Standard Algorithm

- accesses $O(mn)$ data $n$ times

Communication-Avoiding Algorithm

- accesses $O(mn)$ data $O(1)$ times
New algorithms and implementations

- QR decomposition
  - up to $8 \times$ speedup on multicore
  - see demo for GPU implementation

- Rectangular matrix multiplication
  - up to $7 \times$ speedup on multicore, $141 \times$ speedup on dist-mem

- Strassen's matrix multiplication algorithm
  - up to $3 \times$ speedup on dist-mem

Euro-Par 2011 Distinguished Paper

SPAA 2011 Best Paper and CACM Research Highlight
New algorithms and implementations

- QR decomposition
  - up to $8 \times$ speedup on multicore
  - see demo for GPU implementation

- “2.5D” square matrix multiplication and LU decomposition
  - up to $12 \times$ speedup for MM and $2 \times$ for LU on dist-mem
  - Euro-Par 2011 Distinguished Paper

- Rectangular matrix multiplication
  - up to $7 \times$ speedup on multicore, $141 \times$ speedup on dist-mem

- Strassen’s matrix multiplication algorithm
  - up to $3 \times$ speedup on dist-mem
  - SPAA 2011 Best Paper and CACM Research Highlight
New algorithms and implementations

- Solving symmetric indefinite linear systems
  - up to $3 \times$ speedup on multicore
  - IPDPS 2013 Best Paper (Algorithms Track)

- Solving the symmetric eigenproblem for band matrices
  - up to $6 \times$ speedup on multicore

- Krylov (iterative) methods for sparse linear systems
  - up to $4 \times$ speedup for GMRES on multicore
  - up to $3.5 \times$ speedup for BiCG-Stab within multigrid on dist-mem

- Sparse matrix-matrix multiplication
  - up to $11 \times$ speedup on dist-mem
New algorithms and implementations

- Sparse matrix kernels (pOSKI library)
  - up to $9 \times$ speedup for SpMV and $21 \times$ for multiple vectors on multicore

- Stencil operations on structured grids
  - up to $4 \times$ speedup on multicore

- Floyd-Warshall (all-pairs shortest paths)
  - up to $2 \times$ speedup on dist-mem

- Direct N-body calculations
  - up to $10 \times$ speedup on dist-mem
Can we improve dense matrix multiplication?

Here’s a strong-scaling plot, for fixed matrix dimension: $n = 94,080$

\[
\frac{2n^3}{P \cdot \text{time}} \rightarrow
\]

Effective GFLOPS per node

Machine peak (for classical algorithms)

ScaLAPACK (2D)

benchmarked on a Cray XT4
Can we improve dense matrix multiplication?

Here’s a strong-scaling plot, for fixed matrix dimension: $n = 94,080$

![Graph showing performance of matrix multiplication algorithms compared to machine peak.](image)

- ScaLAPACK (2D)
- 2.5D (classical)

Benchmarked on a Cray XT4
Can we improve dense matrix multiplication?

Here’s a strong-scaling plot, for fixed matrix dimension: $n = 94,080$

benchmarked on a Cray XT4

(Old) Strassen

2.5D (classical)

ScaLAPACK (2D)
Can we improve dense matrix multiplication?

Here’s a strong-scaling plot, for fixed matrix dimension: \( n = 94,080 \)

\[
\frac{2n^3}{P \cdot \text{time}} \rightarrow 50
\]

(Old) Strassen

New Strassen

2.5D (classical)

ScaLAPACK (2D)

Machine peak (for classical algorithms)

 benchmarks on a Cray XT4
Need autotuning to optimize performance

All of the previous performance numbers required low-level optimization and tuning

- automating the process improves performance portability

Prime example of effectiveness of autotuning: pOSKI

- sparse matrix-vector operations library for multicore
- performs two types of autotuning:
  - off-line architecture-specific tuning
  - on-line matrix-specific tuning
pOSKI: autotuning for sparse matrix-vector operations

Best speedups: \(9.3 \times\) over CSR, \(8.6 \times\) over OSKI, \(3.2 \times\) over MKL
Future Directions

Communication lower bounds extend to programs that access arrays
- much more general than just linear algebra!
- working to incorporate analysis into compilers

Models for running time extend to energy cost
- new theory for minimizing energy at algorithmic level
- crucial to achieve goals of exascale computing and ASPIRE project

Ultimate goal: develop communication-avoiding, autotuned algorithms for all computational patterns, minimizing both time and energy
Beboppers

Michael Anderson
Grey Ballard
Austin Benson
Aydın Buluç
Jong-Ho Byun
Razvan Carbunescu
Erin Carson
Kaushik Datta
Orianna DeMasi
Jim Demmel
Aditya Devarakonda
Michael Driscoll
David Eliahu
Andrew Gearhart
Evangelos Georganas
Mark Hoemmen
Shoaib Kamil
Nick Knight
Penporn Koanantakool
Richard Lin
Benjamin Lipshitz
Marghoob Mohiyuddin
Hong Diep Nguyen
Rajesh Nishtala
Cindy Rubio González
Oded Schwartz
Edgar Solomonik
Omer Spillinger
Brian Van Straalen
Vasily Volkov
Sam Williams
Kathy Yelick

bebop.cs.berkeley.edu
We applied a communication-avoiding algorithm to a video background subtraction application.

Solved using “Robust PCA”.

Iteratively take singular value decomposition of the tall-skinny video matrix.
- Tall-skinny SVD can be efficiently solved using QR factorization.

Background subtraction application visualization.
We demonstrate the following implementations:

- BLAS2 QR on NVIDIA GPU
- Communication-Avoiding QR on NVIDIA GPU
- MKL SVD on Intel Multicore