Communication Avoiding Successive Band Reduction

Nick Knight, Grey Ballard, James Demmel

UC Berkeley

SIAM PP12

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, and Samsung.
For high performance, we must reformulate existing algorithms in order to reduce data movement (i.e., avoid communication).

We want to tridiagonalize a symmetric band matrix:
- Application: dense symmetric eigenproblem
- Only want the eigenvalues (no eigenvectors)

Our improved band reduction algorithm:
- Moves asymptotically less data
- **Speeds up** against tuned libraries on a multicore platform, up to 2× serial, 6× parallel

With our band-reduction approach, two-step tridiagonalization of a dense matrix is **communication-optimal** for all problem sizes.
Motivation

By *communication* we mean

- moving data within memory hierarchy on a sequential computer
- moving data between processors on a parallel computer

Communication is expensive, so our goal is to minimize it

- in many cases we need new algorithms
- in many cases we can prove lower bounds and optimality
Direct vs Two-Step Tridiagonalization

Application: solving the dense symmetric eigenproblem via reduction to tridiagonal form (tridiagonalization)

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

**Direct:**

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

**Two-step:**

\[
\begin{pmatrix}
1 & 2 \\
A & B & T
\end{pmatrix}
\]
Direct vs Two-Step Tridiagonalization

Application: solving the dense symmetric eigenproblem via reduction to tridiagonal form (tridiagonalization)

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

**Direct:**

![Diagram of direct tridiagonalization process](image1)

**Two-step:**

![Diagram of two-step tridiagonalization process](image2)

![Graph showing MFLOPS vs n](image3)
Why is direct tridiagonalization slow?

Communication costs!

<table>
<thead>
<tr>
<th>Approach</th>
<th>Flops</th>
<th>Words Moved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>( \frac{4}{3} n^3 )</td>
<td>( O(n^3) )</td>
</tr>
<tr>
<td>Two-step</td>
<td>(1) ( \frac{4}{3} n^3 )</td>
<td>( O\left(\frac{n^3}{\sqrt{M}}\right) )</td>
</tr>
<tr>
<td></td>
<td>(2) ( O(n^2\sqrt{M}) )</td>
<td>( O(n^2\sqrt{M}) )</td>
</tr>
</tbody>
</table>

\( M = \) fast memory size

- Direct approach achieves \( O(1) \) data re-use
- Two-step approach moves fewer words than direct approach
  - using intermediate bandwidth \( b = \Theta(\sqrt{M}) \)
- Full-to-banded step (1) achieves \( O(\sqrt{M}) \) data re-use
  - this is optimal
- Band reduction step (2) achieves \( O(1) \) data re-use

Can we do better?
Band Reduction - previous work

1963 Rutishauser: Givens-based down diagonals and Householder-based
1968 Schwarz: Givens-based up columns
1975 Muraka-Horikoshi: improved R’s Householder-based algorithm
1984 Kaufman: vectorized S’s algorithm
1993 Lang: parallelized M-H’s algorithm (distributed-mem)
2000 Bischof-Lang-Sun: generalized everything but S’s algorithm
2009 Davis-Rajamanickam: Givens-based in blocks
2011 Luszczek-Ltaief-Dongarra: parallelized M-H’s algorithm (shared-mem)
2011 Haidar-Ltaief-Dongarra: combined L-L-D and D-R
  ● see A. Haidar’s talk in MS50 tomorrow
Band Reduction - previous work

1963 Rutishauser: Givens-based down diagonals and Householder-based
1968 Schwarz: Givens-based up columns
1975 Muraka-Horikoshi: improved R’s Householder-based algorithm
1984 Kaufman: vectorized S’s algorithm
1993 Lang: parallelized M-H’s algorithm (distributed-mem)
2000 Bischof-Lang-Sun: generalized everything but S’s algorithm
2009 Davis-Rajamanickam: Givens-based in blocks
2011 Luszczek-Ltaief-Dongarra: parallelized M-H’s algorithm (shared-mem)
2011 Haidar-Ltaief-Dongarra: combined L-L-D and D-R
  see A. Haidar’s talk in MS50 tomorrow
Successive Band Reduction (bulge-chasing)

\[
\begin{align*}
\text{constraint:} & \quad c + d \leq b \\
\text{b} & = \text{bandwidth} \\
\text{c} & = \text{columns} \\
\text{d} & = \text{diagonals}
\end{align*}
\]
How do we get data re-use?

1. Increase number of columns in parallelogram \((c)\)
   - permits blocking Householder updates: \(O(c)\) re-use
   - constraint \(c + d \leq b \implies \) trade-off between re-use and progress

2. Chase multiple bulges at a time \((\omega)\)
   - apply several updates to band while it’s in cache: \(O(\omega)\) re-use
   - bulges cannot overlap, need working set to fit in cache

Nick Knight
Communication Avoiding Successive Band Reduction
8
How do we get data re-use?

1. Increase number of columns in parallelogram \((c)\)
   - permits blocking Householder updates: \(O(c)\) re-use
   - constraint \(c + d \leq b \implies\) trade-off between re-use and progress

2. Chase multiple bulges at a time \((\omega)\)
   - apply several updates to band while it’s in cache: \(O(\omega)\) re-use
   - bulges cannot overlap, need working set to fit in cache
How do we get data re-use?

1. Increase number of columns in parallelogram \((c)\)
   - permits blocking Householder updates: \(O(c)\) re-use
   - constraint \(c + d \leq b \implies\) trade-off between re-use and progress

2. Chase multiple bulges at a time \((\omega)\)
   - apply several updates to band while it’s in cache: \(O(\omega)\) re-use
   - bulges cannot overlap, need working set to fit in cache
How do we get data re-use?

1. Increase number of columns in parallelogram \((c)\)
   - permits blocking Householder updates: \(O(c)\) re-use
   - constraint \(c + d \leq b\) \(\implies\) trade-off between re-use and progress

2. Chase multiple bulges at a time \((\omega)\)
   - apply several updates to band while it’s in cache: \(O(\omega)\) re-use
   - bulges cannot overlap, need working set to fit in cache
How do we get data re-use?

1. Increase number of columns in parallelogram \((c)\)
   - permits blocking Householder updates: \(O(c)\) re-use
   - constraint \(c + d \leq b\) \(\implies\) trade-off between re-use and progress

2. Chase multiple bulges at a time \((\omega)\)
   - apply several updates to band while it’s in cache: \(O(\omega)\) re-use
   - bulges cannot overlap, need working set to fit in cache
How do we get data re-use?

1. Increase number of columns in parallelogram \((c)\)
   - permits blocking Householder updates: \(O(c)\) re-use
   - constraint \(c + d \leq b \iff \) trade-off between re-use and progress

2. Chase multiple bulges at a time \((\omega)\)
   - apply several updates to band while it’s in cache: \(O(\omega)\) re-use
   - bulges cannot overlap, need working set to fit in cache
How do we get data re-use?

1. Increase number of columns in parallelogram ($c$)
   - permits blocking Householder updates: $O(c)$ re-use
   - constraint $c + d \leq b \implies$ trade-off between re-use and progress

2. Chase multiple bulges at a time ($\omega$)
   - apply several updates to band while it’s in cache: $O(\omega)$ re-use
   - bulges cannot overlap, need working set to fit in cache
How do we get data re-use?

1. Increase number of columns in parallelogram ($c$)
   - permits blocking Householder updates: $O(c)$ re-use
   - constraint $c + d \leq b \implies$ trade-off between re-use and progress

2. Chase multiple bulges at a time ($\omega$)
   - apply several updates to band while it’s in cache: $O(\omega)$ re-use
   - bulges cannot overlap, need working set to fit in cache
How do we get data re-use?

1. Increase number of columns in parallelogram \( (c) \)
   - permits blocking Householder updates: \( O(c) \) re-use
   - constraint \( c + d \leq b \) \( \implies \) trade-off between re-use and progress

2. Chase multiple bulges at a time \( (\omega) \)
   - apply several updates to band while it’s in cache: \( O(\omega) \) re-use
   - bulges cannot overlap, need working set to fit in cache
Data access patterns

One bulge at a time

Four bulges at a time

$\omega = 4$: same amount of work, $4 \times$ fewer words moved
Shared-Memory Parallel Implementation

lots of dependencies:
use pipelining

threads maintain working sets which never overlap
Tradeoff: $c$ and $\omega$
- $c$ - number of columns in each parallelogram
- $\omega$ - number of bulges chased at a time

CA-SBR cuts remaining bandwidth in half at each sweep
- starts with big $c$ and decreases by half at each sweep
- starts with small $\omega$ and doubles at each sweep
Communication-Avoiding SBR - theory

Tradeoff: \( c \) and \( \omega \)
- \( c \) - number of columns in each parallelogram
- \( \omega \) - number of bulges chased at a time

CA-SBR cuts remaining bandwidth in half at each sweep
- starts with big \( c \) and decreases by half at each sweep
- starts with small \( \omega \) and doubles at each sweep

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Flops</th>
<th>Words Moved</th>
<th>Data Re-use</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</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(^\dagger)</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>

\(^*\)SBR framework with optimal parameter choices
\(^\dagger\)assuming \(1 \leq b \leq \sqrt{M}/3\)
Communication-Avoiding SBR - theory

Tradeoff: $c$ and $\omega$

- $c$ - number of columns in each parallelogram
- $\omega$ - number of bulges chased at a time

CA-SBR cuts remaining bandwidth in half at each sweep

- starts with big $c$ and decreases by half at each sweep
- starts with small $\omega$ and doubles at each sweep

<table>
<thead>
<tr>
<th>Alg.</th>
<th>Flops</th>
<th>Words Moved</th>
<th>Data Re-use</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</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>

*SBR framework with optimal parameter choices
†assuming $1 \leq b \leq \sqrt{M}/3$

- We have similar theoretical improvements in dist-mem parallel case
Main tuning parameters:

1. Number of sweeps and diagonals per sweep: \( \{d_i\} \)
   - satisfying \( \sum d_i = b \)

2. Parameters for \( i^{th} \) sweep
   a. number of columns in each parallelogram: \( c_i \)
      - satisfying \( c_i + d_i \leq b_i \)
   b. number of bulges chased at a time: \( \omega_i \)
   c. number of times bulge is chased in a row: \( \ell_i \)

3. Parameters for individual bulge chase
   a. algorithm choice (BLAS-1, BLAS-2, BLAS-3 varieties)
   b. inner blocking size for BLAS-3
Experimental Platform

- Intel Westmere-EX (Boxboro)
  - 4 sockets, 10 cores per socket, hyperthreading
  - 24MB L3 (shared) per socket, 256KB L2 (private) per core
  - MKL v.10.3, PLASMA v.2.4.1, ICC v.11.1
- Experiments run on single socket (up to 10 threads)
CA-SBR vs MKL (dsbtrd), sequential

Speedup

Bandwidth $b$

Matrix dimension $n$

<table>
<thead>
<tr>
<th>$n$</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>250</th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>0.9</td>
<td>0.9</td>
<td>1.1</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>8000</td>
<td>1.0</td>
<td>0.9</td>
<td>1.1</td>
<td>1.3</td>
<td>1.4</td>
<td>1.6</td>
</tr>
<tr>
<td>12000</td>
<td>1.0</td>
<td>0.9</td>
<td>1.2</td>
<td>1.5</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>16000</td>
<td>0.9</td>
<td>0.9</td>
<td>1.4</td>
<td>1.7</td>
<td>1.8</td>
<td>1.9</td>
</tr>
<tr>
<td>20000</td>
<td>1.0</td>
<td>1.1</td>
<td>1.5</td>
<td>1.8</td>
<td>1.9</td>
<td>2.0</td>
</tr>
<tr>
<td>24000</td>
<td>1.0</td>
<td>1.2</td>
<td>1.6</td>
<td>1.8</td>
<td>2.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>
CA-SBR (10 threads) vs CA-SBR (1 thread)

Speedup

Matrix dimension n

Bandwidth b

8.8 8.1 9.4 9.2 8.5 8.4
9.2 8.8 9.2 8.9 8.2 8.3
8.9 9.3 9.2 8.6 8.0 7.8
9.0 9.8 8.9 7.9 7.4 7.4
8.7 9.2 8.1 6.8 5.9 6.0
8.2 6.7 5.6 4.4 3.6 3.6
8.8 9.2 9.4 9.2 8.9 8.1
6.7 9.4 9.2 8.9 8.6 7.9
5.6 4.4 3.6 3.6 4.4 5.6
4.4 3.6 3.6 4.4 5.6 6.7
3.6 3.6 4.4 5.6 6.7 8.2
2.5 3.6 3.6 4.4 5.6 6.7
1.5 2.5 3.6 4.4 5.6 6.7
0.5 1.5 2.5 3.6 4.4 5.6
0 0.5 1.5 2.5 3.6 4.4

Nick Knight Communication Avoiding Successive Band Reduction
CA-SBR vs PLASMA (pdsbrdt), 10 threads

Speedup

Matrix dimension n

Bandwidth b

Nick Knight
Communication Avoiding Successive Band Reduction

16
On the largest experimental problem \( n = 24000, \ b = 300 \), our serial CA-SBR implementation attained

- **2× speedup** vs. MKL dsbtrd \( (p = 1 \text{ thread}) \)
  - 36% of dgemm peak (50% counting actual flops).

- dsbtrd is a vectorized version of the Schwarz algorithm \( (O(1) \text{ reuse}) \).

- dsbtrd performance did not improve with \( p \) so we compared only serial implementations.

- MKL also provides an implementation of SBR (dsyrdb) but does not expose the band-to-tridiagonal routine, so we could not compare.
On the largest experimental problem $n = 24000$, $b = 300$, our multithreaded CA-SBR implementation attained

- **6× speedup** vs. PLASMA `pdsbrdt` ($p = 10$ threads)
  - 30% of `dgemm` peak (40% counting actual flops).

- In PLASMA v.2.4.1, `pdsbrdt` is a tiled, multithreaded, dynamically scheduled implementation of M-H algorithm ($O(1)$ reuse).

- We are collaborating with the PLASMA developers - they have improved their `pdsbrdt` scheduler since (current version is 2.4.5).

- Our CA-SBR implementation is not NUMA-aware so we restricted our tests to a single socket (10 cores).
Conclusions and Future Work

Theoretical Results
- Analysis of communication costs of existing algorithms
- CA-SBR reduces communication below lower bound for matmul
  - Is it optimal?

Practical Results
- Heuristic tuning leads to speedups, for both the band reduction problem and the dense eigenproblem
- Implementation exposes important tuning parameters
  - Automate tuning process

Extensions
- Handle eigenvector updates (results here are for eigenvalues only)
- Extend to bidiagonal reduction (SVD) case
- Distributed-memory parallel algorithm
Aggarwal, A., and Vitter, J. S.  
The input/output complexity of sorting and related problems.  

http://icl.cs.utk.edu/plasma/.

Ballard, G., Demmel, J., Holtz, O., and Schwartz, O.  
Minimizing communication in linear algebra.  

Bischof, C., Lang, B., and Sun, X.  
A framework for symmetric band reduction.  
References II

Bischof, C. H., Lang, B., and Sun, X.
Algorithm 807: The SBR Toolbox—software for successive band reduction.

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

Dongarra, J., Hammarling, S., and Sorensen, D.
Block reduction of matrices to condensed forms for eigenvalue computations.
Fuller, S. H., and Millett, L. I., Eds.  
*The Future of Computing Performance: Game Over or Next Level?*  

Haidar, A., Ltaief, H., and Dongarra, J.  
Parallel reduction to condensed forms for symmetric eigenvalue problems using aggregated fine-grained and memory-aware kernels.  

Howell, G., Demmel, J., Fulton, C., Hammarling, S., and Marmol, K.  
Cache efficient bidiagonalization using BLAS 2.5 operators.  

Kaufman, L.  
Banded eigenvalue solvers on vector machines.  
Kaufman, L.
Band reduction algorithms revisited.

Lang, B.
A parallel algorithm for reducing symmetric banded matrices to tridiagonal form.

Lang, B.
Efficient eigenvalue and singular value computations on shared memory machines.
Ltaief, H., Luszczek, P., and Dongarra, J.
High performance bidiagonal reduction using tile algorithms on homogeneous multicore architectures.

Luszczek, P., Ltaief, H., and Dongarra, J.
Two-stage tridiagonal reduction for dense symmetric matrices using tile algorithms on multicore architectures.

Murata, K., and Horikoshi, K.
A new method for the tridiagonalization of the symmetric band matrix.
Rajamanickam, S.

Rutishauser, H.
On Jacobi rotation patterns.

Schwarz, H.
Algorithm 183: Reduction of a symmetric bandmatrix to triple diagonal form.
Comm. ACM 6, 6 (June 1963), 315–316.

Schwarz, H.
Tridiagonalization of a symmetric band matrix.
Anatomy of a bulge-chase

QR: create zeros
PRE: $A \leftarrow Q^T A$
SYM: $A \leftarrow Q^T AQ$
POST: $A \leftarrow AQ$
CA-SBR sequential performance \((p = 1)\)

<table>
<thead>
<tr>
<th>(n / b)</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>250</th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>24000</td>
<td>1.78</td>
<td>1.85</td>
<td>2.25</td>
<td>2.55</td>
<td>2.78</td>
<td>2.93</td>
</tr>
<tr>
<td>20000</td>
<td>1.77</td>
<td>1.86</td>
<td>2.27</td>
<td>2.56</td>
<td>2.80</td>
<td>2.94</td>
</tr>
<tr>
<td>16000</td>
<td>1.77</td>
<td>1.87</td>
<td>2.27</td>
<td>2.57</td>
<td>2.80</td>
<td>2.95</td>
</tr>
<tr>
<td>12000</td>
<td>1.78</td>
<td>1.87</td>
<td>2.27</td>
<td>2.58</td>
<td>2.81</td>
<td>2.95</td>
</tr>
<tr>
<td>8000</td>
<td>1.80</td>
<td>1.85</td>
<td>2.27</td>
<td>2.59</td>
<td>2.80</td>
<td>2.96</td>
</tr>
<tr>
<td>4000</td>
<td>1.63</td>
<td>1.87</td>
<td>2.28</td>
<td>2.58</td>
<td>2.82</td>
<td>2.88</td>
</tr>
</tbody>
</table>

**Table:** Performance of sequential CA-SBR in GFLOPS. Each row corresponds to a matrix dimension, and each column corresponds to a matrix bandwidth. Effective flop rates are shown—actual performance may be up to 50% higher.
CA-SBR parallel performance \((p = 10)\)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>24000</td>
<td>15.59</td>
<td>14.92</td>
<td>21.17</td>
<td>23.43</td>
<td>23.48</td>
<td>24.79</td>
</tr>
<tr>
<td>20000</td>
<td>16.29</td>
<td>16.47</td>
<td>20.81</td>
<td>22.78</td>
<td>22.89</td>
<td>24.56</td>
</tr>
<tr>
<td>16000</td>
<td>15.80</td>
<td>17.32</td>
<td>20.81</td>
<td>22.02</td>
<td>22.34</td>
<td>23.08</td>
</tr>
<tr>
<td>12000</td>
<td>16.06</td>
<td>18.29</td>
<td>20.19</td>
<td>20.28</td>
<td>20.76</td>
<td>21.74</td>
</tr>
<tr>
<td>8000</td>
<td>15.64</td>
<td>17.14</td>
<td>18.39</td>
<td>17.62</td>
<td>16.56</td>
<td>17.80</td>
</tr>
<tr>
<td>4000</td>
<td>13.36</td>
<td>12.56</td>
<td>12.82</td>
<td>11.48</td>
<td>10.26</td>
<td>10.44</td>
</tr>
</tbody>
</table>

| n / b  |  50 |  100 |  150 |  200 |  250 |  300 |

Table: Performance of parallel CA-SBR in GFLOPS. Each row corresponds to a matrix dimension, and each column corresponds to a matrix bandwidth. Effective flop rates are shown—actual performance may be up to 50% higher.