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.

Talk Summary

- 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)

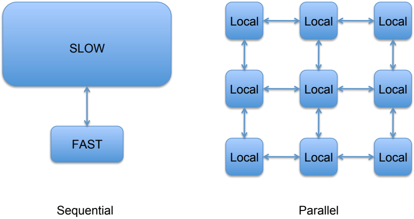
- Our improved **band reduction** algorithm
  - Moves asymptotically less data
  - **Speeds up** against tuned libraries on a multicore platform, up to $2\times$ serial, $6\times$ parallel

- 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

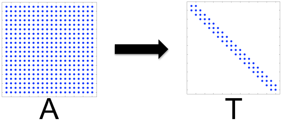
- in many cases we need new algorithms
- in many cases we can prove lower bounds and optimality

Direct vs Two-Step Tridiagonalization

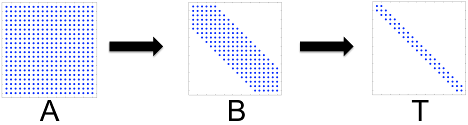
Application: solving the dense symmetric eigenproblem via reduction to tridiagonal form (tridiagonalization)

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

Direct:



Two-step:



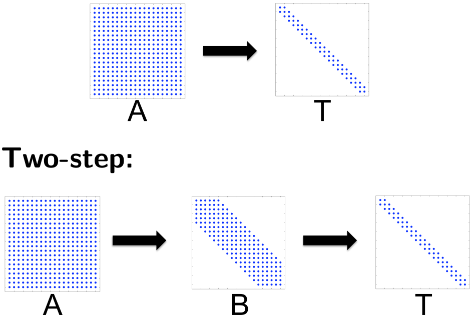
Direct vs Two-Step Tridiagonalization

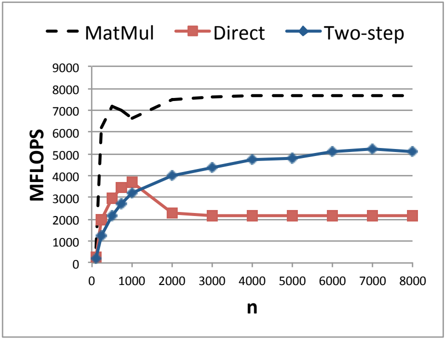
Application: solving the dense symmetric eigenproblem via reduction to tridiagonal form (tridiagonalization)

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

Direct:

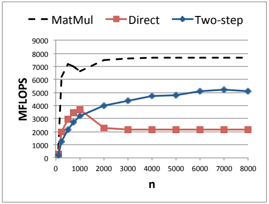
Two-step:





Why is direct tridiagonalization slow?

Communication costs!



Approach		Flops	Words Moved
Direct		$\frac{4}{3}n^3$	$O(n^3)$
Two-step	(1)	$\frac{4}{3}n^3$	$O(\frac{n^3}{\sqrt{M}})$
	(2)	$O(n^2\sqrt{M})$	$O(n^2\sqrt{M})$

$M =$ 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 ←

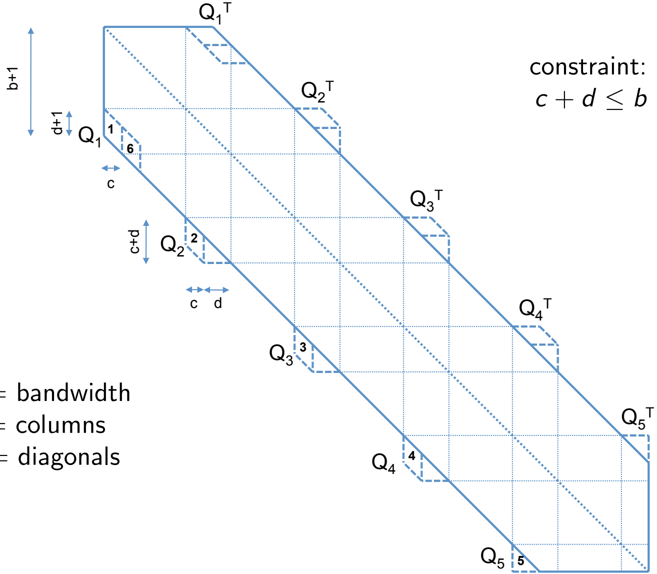
2009 Davis-Rajamanickam: Givens-based in blocks

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)

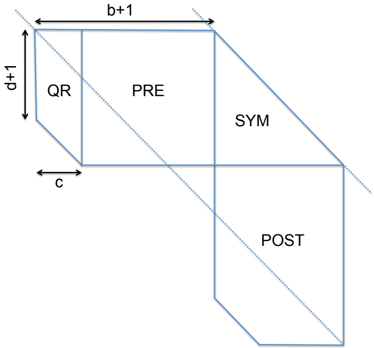


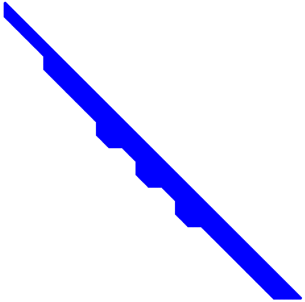
constraint:
$c + d \leq b$

b = bandwidth
c = columns
d = diagonals

How do we get data re-use?

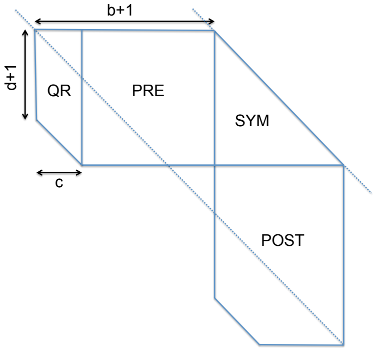
1. Increase number of columns in parallelogram (c)
   - permits blocking Householder updates: $O(c)$ re-use
   - constraint $c + d \leq b \implies$ trade-off between re-use and progress
2. Chase multiple bulges at a time (ω)
   - apply several updates to band while it's in cache: $O(\omega)$ re-use
   - bulges cannot overlap, need working set to fit in cache

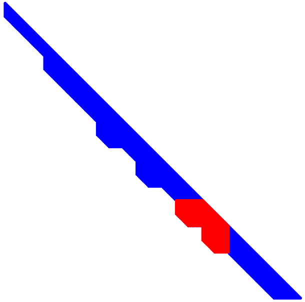




How do we get data re-use?

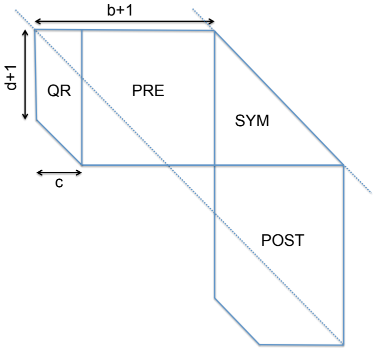
1. Increase number of columns in parallelogram (c)
   - permits blocking Householder updates: $O(c)$ re-use
   - constraint $c + d \leq b \implies$ trade-off between re-use and progress
2. Chase multiple bulges at a time (ω)
   - apply several updates to band while it's in cache: $O(\omega)$ re-use
   - bulges cannot overlap, need working set to fit in cache

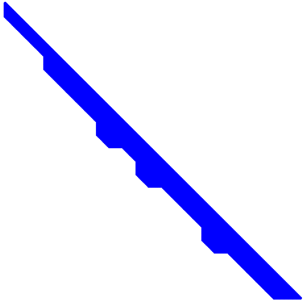




How do we get data re-use?

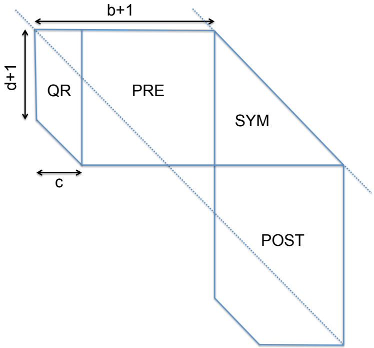
1. Increase number of columns in parallelogram (c)
   - permits blocking Householder updates: $O(c)$ re-use
   - constraint $c + d \leq b \implies$ trade-off between re-use and progress
2. Chase multiple bulges at a time (ω)
   - apply several updates to band while it's in cache: $O(\omega)$ re-use
   - bulges cannot overlap, need working set to fit in cache

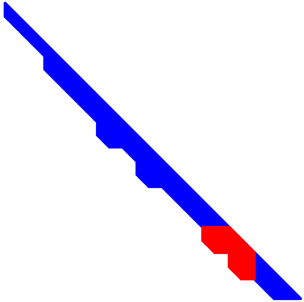




How do we get data re-use?

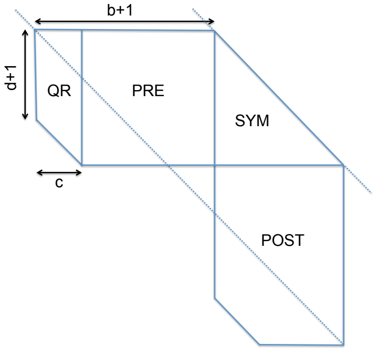
1. Increase number of columns in parallelogram (c)
   - permits blocking Householder updates: $O(c)$ re-use
   - constraint $c + d \leq b \implies$ trade-off between re-use and progress
2. Chase multiple bulges at a time (ω)
   - apply several updates to band while it's in cache: $O(\omega)$ re-use
   - bulges cannot overlap, need working set to fit in cache

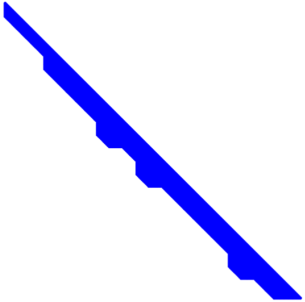




How do we get data re-use?

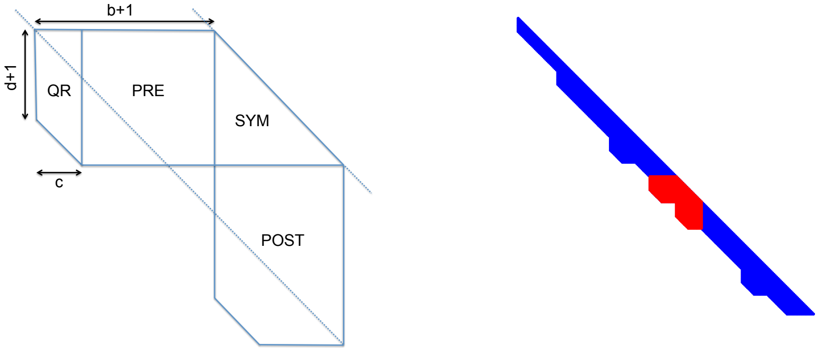
1. Increase number of columns in parallelogram (c)
   - permits blocking Householder updates: $O(c)$ re-use
   - constraint $c + d \leq b \implies$ trade-off between re-use and progress
2. Chase multiple bulges at a time (ω)
   - apply several updates to band while it's in cache: $O(\omega)$ re-use
   - bulges cannot overlap, need working set to fit in cache





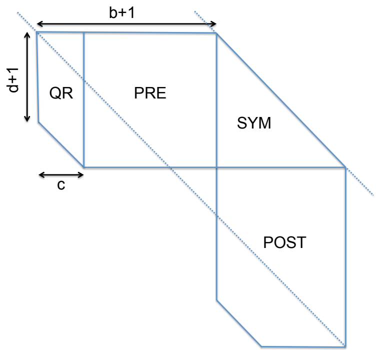
How do we get data re-use?

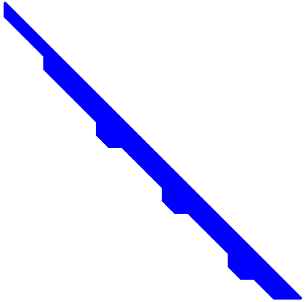
1. Increase number of columns in parallelogram (c)
   - permits blocking Householder updates: $O(c)$ re-use
   - constraint $c + d \leq b \implies$ trade-off between re-use and progress
2. Chase multiple bulges at a time (ω)
   - apply several updates to band while it's in cache: $O(\omega)$ re-use
   - bulges cannot overlap, need working set to fit in cache



How do we get data re-use?

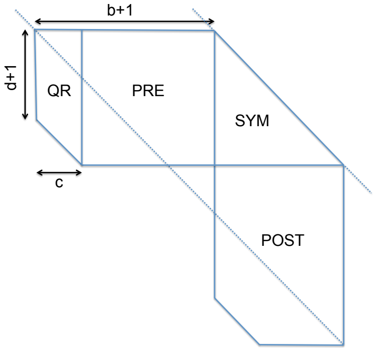
1. Increase number of columns in parallelogram (c)
   - permits blocking Householder updates: $O(c)$ re-use
   - constraint $c + d \leq b \implies$ trade-off between re-use and progress
2. Chase multiple bulges at a time (ω)
   - apply several updates to band while it's in cache: $O(\omega)$ re-use
   - bulges cannot overlap, need working set to fit in cache

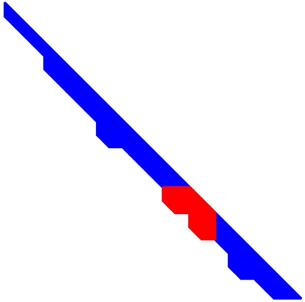




How do we get data re-use?

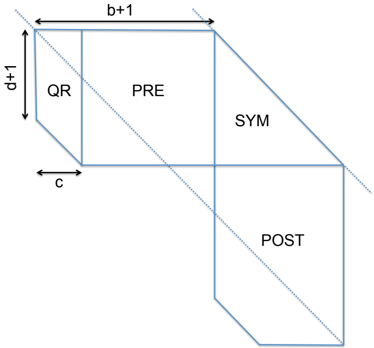
1. Increase number of columns in parallelogram (c)
   - permits blocking Householder updates: $O(c)$ re-use
   - constraint $c + d \leq b \implies$ trade-off between re-use and progress
2. Chase multiple bulges at a time (ω)
   - apply several updates to band while it's in cache: $O(\omega)$ re-use
   - bulges cannot overlap, need working set to fit in cache

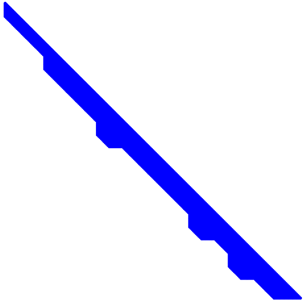




How do we get data re-use?

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 (ω)
   - apply several updates to band while it's in cache: $O(\omega)$ re-use
   - bulges cannot overlap, need working set to fit in cache





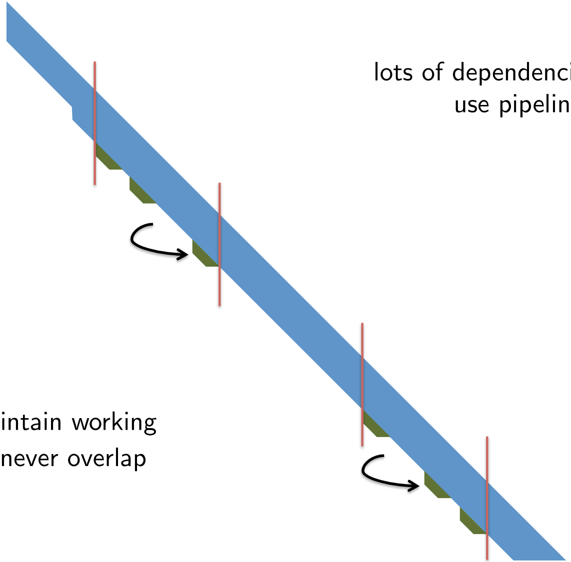
Data access patterns

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

Communication-Avoiding SBR - theory

Tradeoff: c and ω

- c - number of columns in each parallelogram
- ω - 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 ω and doubles at each sweep

Communication-Avoiding SBR - theory

Tradeoff: c and ω

- c - number of columns in each parallelogram
- ω - 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 ω and doubles at each sweep

Alg.	Flops	Words Moved	Data Re-use
S	$4n^2b$	$O(n^2b)$	$O(1)$
M-H	$6n^2b$	$O(n^2b)$	$O(1)$
B-L-S*	$5n^2b$	$O(n^2 \log b)$	$O\left(\frac{b}{\log b}\right)$
CA-SBR†	$5n^2b$	$O\left(\frac{n^2b^2}{M}\right)$	$O\left(\frac{M}{b}\right)$

*SBR framework with optimal parameter choices

†assuming $1 \leq b \leq \sqrt{M}/3$

Communication-Avoiding SBR - theory

Tradeoff: c and ω

- c - number of columns in each parallelogram
- ω - 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 ω and doubles at each sweep

Alg.	Flops	Words Moved	Data Re-use
S	$4n^2b$	$O(n^2b)$	$O(1)$
M-H	$6n^2b$	$O(n^2b)$	$O(1)$
B-L-S*	$5n^2b$	$O(n^2\log b)$	$O\left(\frac{b}{\log b}\right)$
CA-SBR†	$5n^2b$	$O\left(\frac{n^2b^2}{M}\right)$	$O\left(\frac{M}{b}\right)$

*SBR framework with optimal parameter choices

†assuming $1 \le b \le \sqrt{M}/3$

- We have similar theoretical improvements in dist-mem parallel case

Search Space for Autotuning

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: ω_i

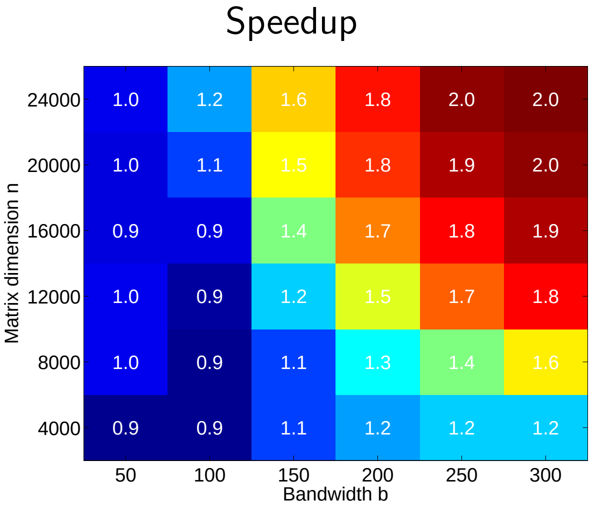
   c. number of times bulge is chased in a row: ℓ_i
3. Parameters for individual bulge chase
   a. algorithm choice (BLAS-1, BLAS-2, BLAS-3 varieties)

   b. inner blocking size for BLAS-3

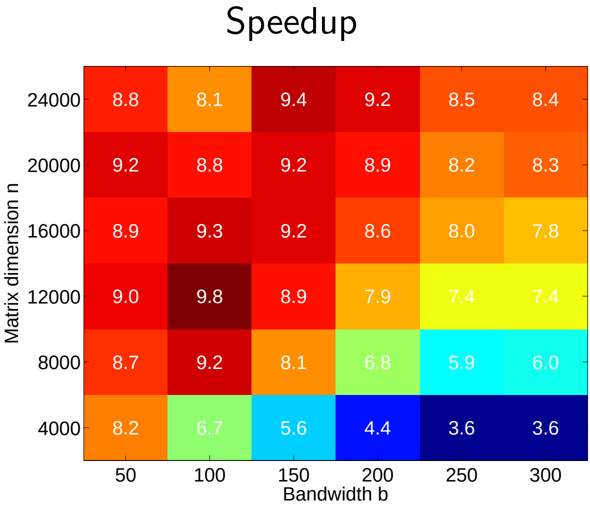
Experimental Platform

- 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)

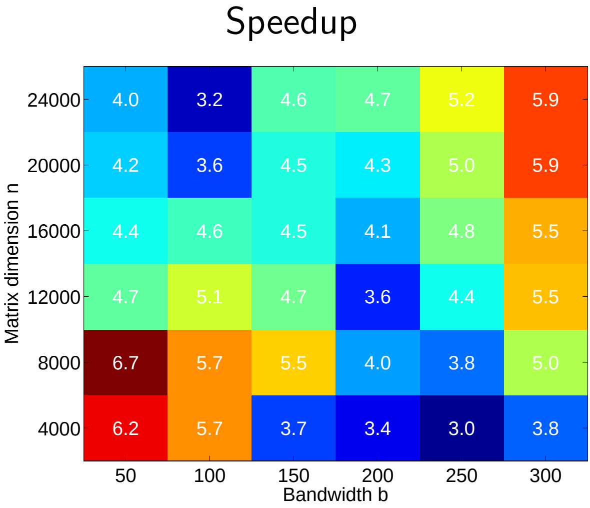
CA-SBR vs MKL (dsbtrd), sequential



CA-SBR (10 threads) vs CA-SBR (1 thread)



CA-SBR vs PLASMA (pdsbrdt), 10 threads



Best serial speedups on Boxboro

On the largest experimental problem $n = 24000$, $b = 300$, our serial CA-SBR implementation attained

- **2× speedup** vs. MKL `dsbtrd` ($p = 1$ thread)
  - 36% of `dgemm` peak (50% counting actual flops).
- `dsbtrd` is a vectorized version of the S algorithm ($O(1)$ 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.

Best parallel speedups on Boxboro

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

Thank you!

Nick Knight, Grey Ballard, James Demmel
{knight,ballard,demmel}@cs.berkeley.edu

References I

Aggarwal, A., and Vitter, J. S.
The input/output complexity of sorting and related problems.
Comm. ACM 31, 9 (1988), 1116–1127.

Agullo, E., Dongarra, J., Hadri, B., Kurzak, J., Langou, J., Langou, J., Ltaief, H., Luszczek, P., and YarKhan, A.
PLASMA users' guide, 2009.
http://icl.cs.utk.edu/plasma/.

Ballard, G., Demmel, J., Holtz, O., and Schwartz, O.
Minimizing communication in linear algebra.
SIAM Journal on Matrix Analysis and Applications 32, 3 (2011), 866-901.

Bischof, C., Lang, B., and Sun, X.
A framework for symmetric band reduction.
ACM Trans. Math. Soft. 26, 4 (2000), 581–601.

References II

Bischof, C. H., Lang, B., and Sun, X.
Algorithm 807: The SBR Toolbox—software for successive band reduction.
ACM Trans. Math. Soft. 26, 4 (2000), 602–616.

Demmel, J., Grigori, L., Hoemmen, M., and Langou, J.
Communication-optimal parallel and sequential QR and LU factorizations.
SIAM J. Sci. Comput. (2011). To appear.

Dongarra, J., Hammarling, S., and Sorensen, D.
Block reduction of matrices to condensed forms for eigenvalue computations.
Journal of Computational and Applied Mathematics 27 (1989).

References III

Fuller, S. H., and Millett, L. I., Eds.
The Future of Computing Performance: Game Over or Next Level?
The National Academies Press, Washington, D.C., 2011.

Haidar, A., Ltaief, H., and Dongarra, J.
Parallel reduction to condensed forms for symmetric eigenvalue problems using aggregated fine-grained and memory-aware kernels.
Proceedings of the ACM/IEEE Conference on Supercomputing (2011).

Howell, G., Demmel, J., Fulton, C., Hammarling, S., and Marmol, K.
Cache efficient bidiagonalization using BLAS 2.5 operators.
ACM Trans. Math. Softw. 34, 3 (2008), 14:1-14:33.

Kaufman, L.
Banded eigenvalue solvers on vector machines.
ACM Trans. Math. Softw. 10 (1984), 73–86.

References IV

KAUFMAN, L.
Band reduction algorithms revisited.
ACM Trans. Math. Softw. 26 (December 2000), 551–567.

LANG, B.
A parallel algorithm for reducing symmetric banded matrices to tridiagonal form.
SIAM J. Sci. Comput. 14, 6 (1993), 1320–1338.

LANG, B.
Efficient eigenvalue and singular value computations on shared memory machines.
Par. Comp. 25, 7 (1999), 845 – 860.

References V

LTAIEF, H., LUSZCZEK, P., AND DONGARRA, J.
High performance bidiagonal reduction using tile algorithms on homogeneous multicore architectures.
Tech. Rep. 247, LAPACK Working Note, May 2011.
Submitted to ACM TOMS.

LUSZCZEK, P., LTAIEF, H., AND DONGARRA, J.
Two-stage tridiagonal reduction for dense symmetric matrices using tile algorithms on multicore architectures.
In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium* (2011).

MURATA, K., AND HORIKOSHI, K.
A new method for the tridiagonalization of the symmetric band matrix.
Information Processing in Japan 15 (1975), 108–112.

References VI

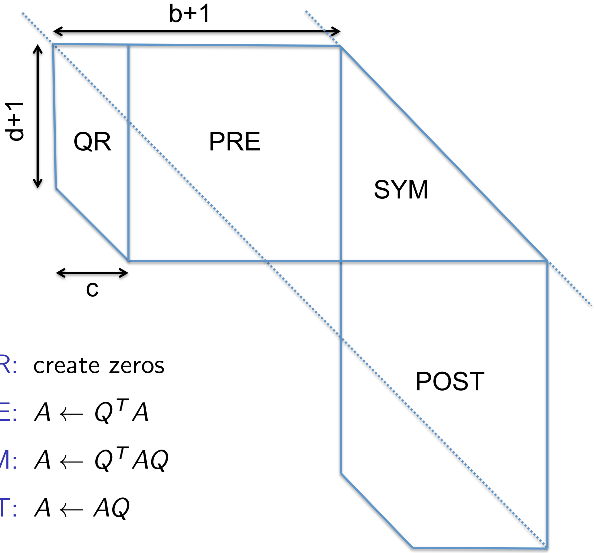
Rajamanickam, S.
Efficient Algorithms for Sparse Singular Value Decomposition.
PhD thesis, University of Florida, 2009.

Rutishauser, H.
On Jacobi rotation patterns.
In *Proceedings of Symposia in Applied Mathematics* (1963), vol. 15, pp. 219–239.

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.
Numerische Mathematik 12 (1968), 231–241.

Anatomy of a bulge-chase



QR: create zeros

PRE: $A \leftarrow Q^T A$

SYM: $A \leftarrow Q^T A Q$

POST: $A \leftarrow AQ$

CA-SBR sequential performance ($p = 1$)

24000	1.78	1.85	2.25	2.55	2.78	2.93
20000	1.77	1.86	2.27	2.56	2.80	2.94
16000	1.77	1.87	2.27	2.57	2.80	2.95
12000	1.78	1.87	2.27	2.58	2.81	2.95
8000	1.80	1.85	2.27	2.59	2.80	2.96
4000	1.63	1.87	2.28	2.58	2.82	2.88
n / b	**50**	**100**	**150**	**200**	**250**	**300**

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 ($p = 10$)

24000	15.59	14.92	21.17	23.43	23.48	24.79
20000	16.29	16.47	20.81	22.78	22.89	24.56
16000	15.80	17.32	20.81	22.02	22.34	23.08
12000	16.06	18.29	20.19	20.28	20.76	21.74
8000	15.64	17.14	18.39	17.62	16.56	17.80
4000	13.36	12.56	12.82	11.48	10.26	10.44
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.