



Sparse BLAS-3 Reduction
to Banded Upper Triangular (Spar3Bnd)

Gary Howell, HPC/OIT
NC State University
gary_howell@ncsu.edu

Acknowledgements

James Demmel, Gene Golub, Franc Brglez

Supported by NIH Molecular Libraries Roadmap
for Medical Research, Grant 1 P20 Hg003900-01

Multi-Core, Lots of RAM

- Flops are cheap (multi-core). Transferring data from RAM to cache is expensive. Hence algorithms should use BLAS-3 (CPU speed limited) to reuse data in cache as opposed to BLAS-1 and BLAS-2 algorithm (limited by rate of reads and writes to RAM)
- RAM is cheap. Last year at NC State we only had 4 GBytes per blade. This year 16 GBytes. So we can afford to store more data than in the past.

Change in algorithm to better use Architecture?

Times change

Can we improve the algorithm for multi-core cache based architectures?

- Stability. To avoid the fake singular values associated with loss of basis rank, we need a linear independent basis. For accuracy, we need an orthogonal basis. **Use the RAM**
- $Ax, y'A$ are relatively slow computations on modern computers. $AX, Y'A$ are faster.
- Can use BLAS-3 operations with X, Y as opposed to BLAS-1 with x, y **Keep the cores busy by re-using data in cache.**

Lanczos bidiagonalization

minimizes storage and flops. It uses sparse matrix dense vector multiplications

$$y \leftarrow Ax, z^T \leftarrow w^T A$$

and some dense vector BLAS-1 operations

$$\alpha \leftarrow u^T v, c \leftarrow \beta u + d$$

It needs $O(nz) + O(n + m)$ total storage and the same number of flops per iteration. **Stability?**

Vectors too large for cache?

Computation of a few singular values

After k steps a bidiagonal $k \times k$ matrix A_k is produced. If at each step we compute the singular values $\rho_{ik}, 1 \dots k$ of A_k , then we get a triangular matrix of trial singular values.

$$\begin{array}{ccc} \rho_{1,k-1} & \rho_{1,k} & \rho_{1,k+1} \\ \rho_{2,k-1} & \rho_{2,k} & \rho_{2,k+1} \\ \vdots & \vdots & \vdots \\ \rho_{k-1,k-1} & \rho_{k-1,k} & \rho_{k-1,k+1} \\ 0 & \rho_{k,k} & \rho_{k,k+1} \\ 0 & 0 & \rho_{k+1,k+1} \end{array} \quad (1)$$

Interlacing singular values

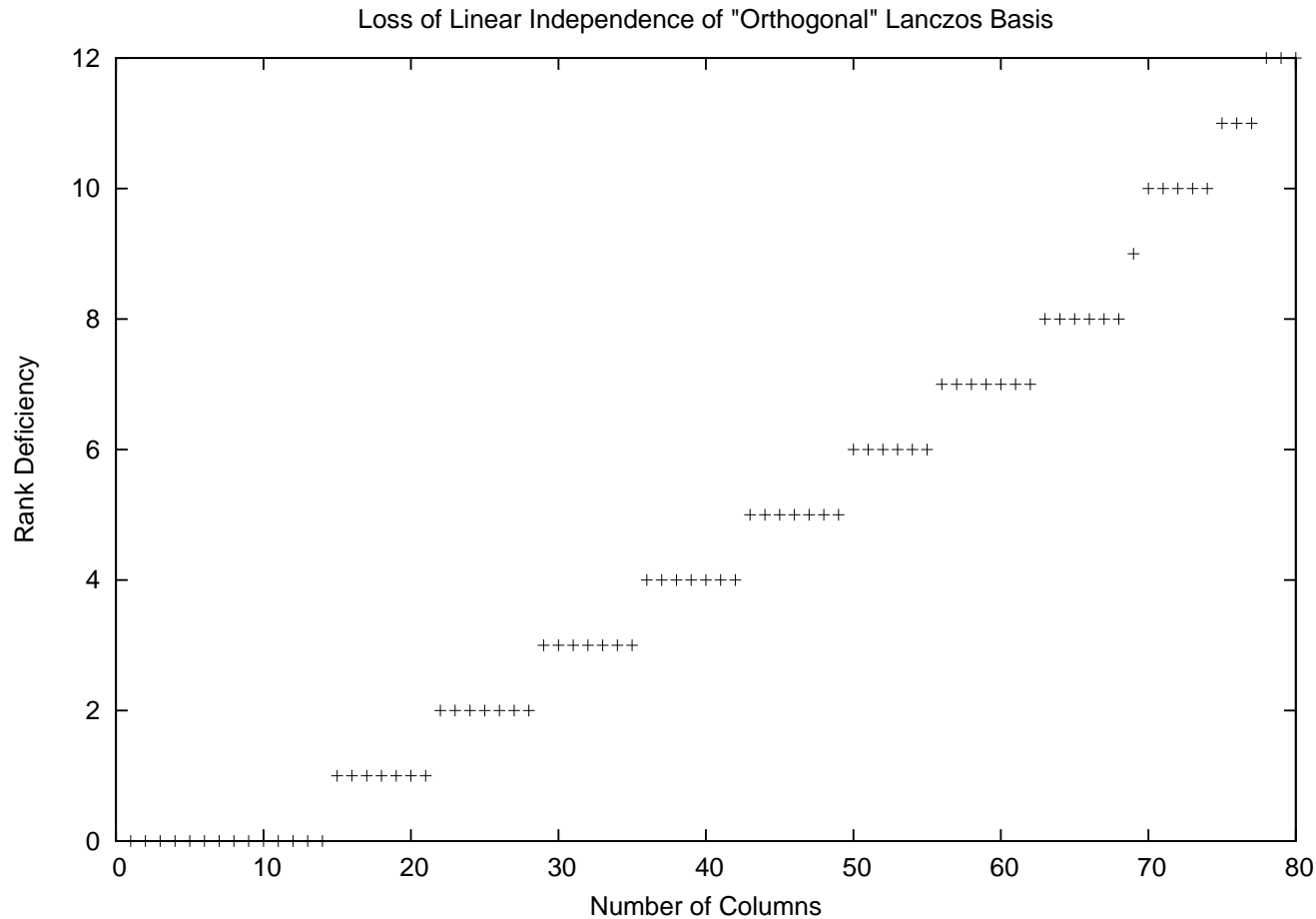
The computed singular values of A_k (column k) relate to those of A_{k+1} (column $k + 1$) by

$$\rho_{i,k} \leq \rho_{i,k+1}, \quad \rho_{i,k} \geq \rho_{i+1,k+1}$$

In exact arithmetic, as k goes to $M = \min(m, n)$, the singular values of A_k go to the singular values of A .

But in practice, we only run the Lanczos algorithm for $k \ll M$, and can recover at most a few singular values of A . **Problem: poor conditioning of Lanczos bases**

Loss of rank of Lanczos "basis"



Of the 50

.. Lanczos vectors produced after 50 steps, 45 are linearly independent (rank deficiency of 5).

Orthogonality for Stability

But for accurate singular values we need to preserve the orthogonality of bases. Some possibilities:

- Use Modified Gram Schmidt to orthogonalize, Simon and Zha, Berry, Larsen.
- Use LU decomposition (Sadok, Stephens)
- Use Householder transformations (ARPACK, Spar3Bnd) .

L_2 Condition Numbers of Bases

Sizes	10	20	30	40	50
Householder QR	1.0	1.0	1.0	1.0	1.0
L in LU	9.70	18.7	46.5	53.0	72.2
MGS QR	1.001	98.5	170.	122.5	3631

Here we factored Hilbert matrices of various sizes and compared condition numbers of Q for QR factorization or L for LU decomposition.

BLAS-3 Block Householder

By working with blocks X we can use BLAS-3 block Householder as opposed to x and BLAS-2 Householder (e.g. ARPACK).

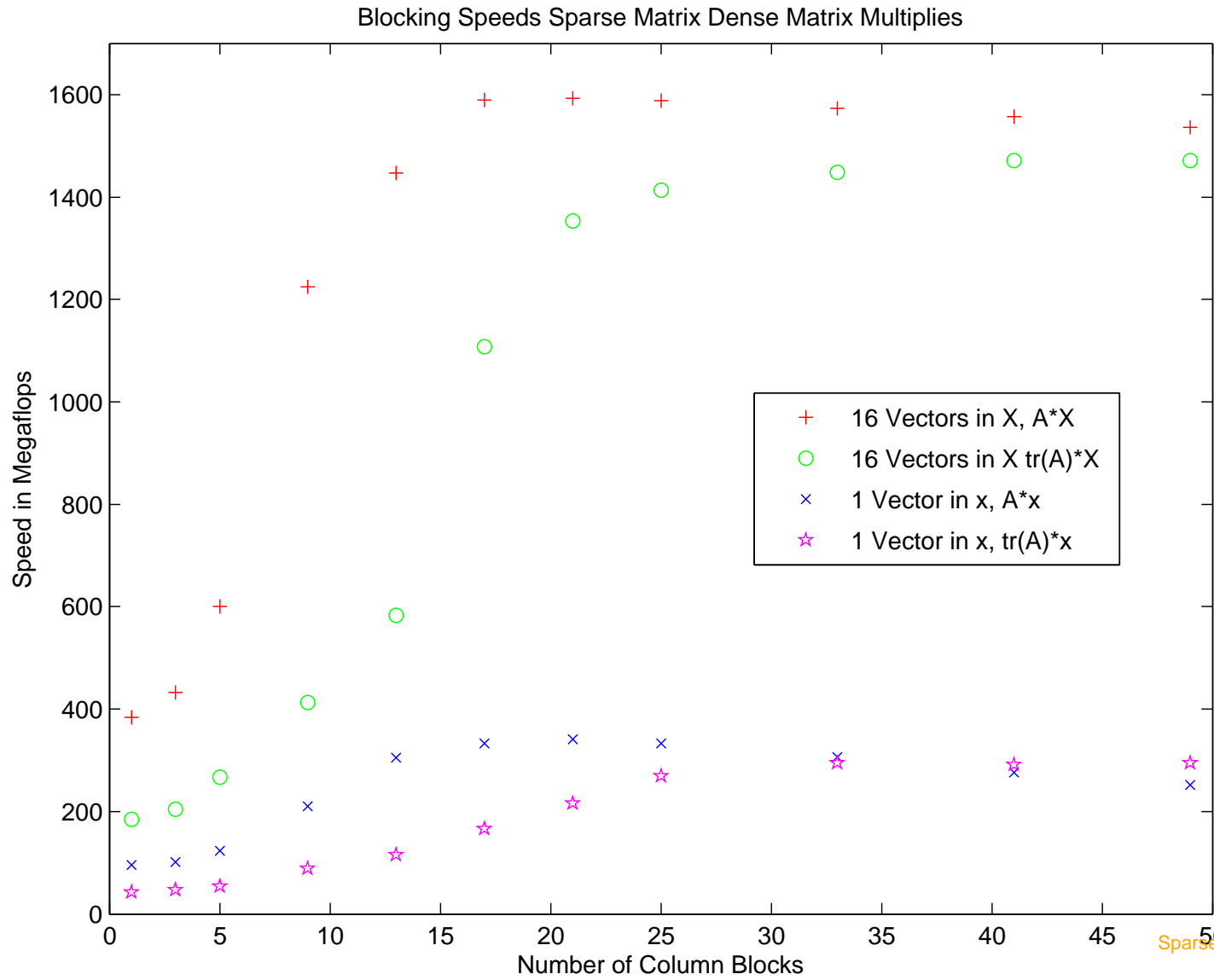
Replace Ax by AX

In the Lanczos algorithm, we bidiagonalize. If instead we reduce to an upper triangular matrix with bandwidth $K + 1$, then we can replace Ax by AX , $y^T A$ by $Y^T A$.

Then dragging A from RAM to cache allows $2K$ flops per element of A as opposed to 2 flops.

- When x or X is too big to fit in cache, can get a drastic hit due to cache misses.
- Solution here to put A in column blocks.
- Example. 200K square matrices, 200 randomly distributed nonzero entries per row. Dual core OpenMP.

(Sparse A)*(Dense X)-20X Seedup



Reducing to upper bandwidth $K + 1$

Sparse BLAS-3 Reduction 2 **Banded** Upper Triangular form (Spar3Bnd) Spar3Bnd adapts dense BLAS-3 algorithm for reduction to bandwidth $K + 1$ using Householder reductions of block size K .

- Dense implementations were by Chris Bischof, Bruno Lang, related work by Xioabai Sun, the current Fortran implementation is from "scratch".
- **The Trick** for a sparse algorithm, do not update the trailing part of the matrix.

Untouched sparse matrix

Partitioning before a row and block elimination

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \quad (2)$$

The only operations on the original sparse matrix are

- Extract blocks.
- Multiply the sparse matrix by dense matrices.

We can't actually form the updated $\hat{A} = A + UZ + WV$. It would be dense and hence exhaust RAM. Instead compute

$$\hat{A}X = AX + U(ZX) + W(VX)$$

On the extracted blocks of A

- Perform the sequence of block Householder eliminations which had already been made to eliminate A_{21}, A_{11}, A_{12} . For example,

$$\hat{A}_{23} \leftarrow A_{23} - U(2, :)Z(:, 3) - W(2, :)V(:, 3)$$

These are all BLAS-3 dense matrix dense matrix operations.

- Perform a QR (actually since this is the row block an LQ) factorization of \hat{A}_{23} . Also BLAS 3.

Append new blocks to U, V, W, Z

To get the new block of W, Z , multiply the currently produced blocks of dense vectors by the sparse A , e.g.

$$W(:, 3) \leftarrow V(3, :)A$$

with a similar multiplication by $U(:, 3)$

The down side here is the storage of the 4 dense matrices, U, W, Z, V . To eliminate l rows and columns requires $2l(m + n)$ storage.

Test 1 – Spar3Bnd is Stable

Spar3Bnd is a stable algorithm which in the sparse case is not (due to storage constraints) run to completion. On my 2 GByte RAM desktop we can test to size around $3K$ square against LAPACK.

LAPACK gives the same singular values for A and \hat{A} to high accuracy.

Practical for real problems ?

We are taking a standard dense algorithm and at the cost of storing multipliers applying it to the sparse case.

- Experiment, given a desktop with 2 GBytes RAM, can we handle reasonable sized sparse matrices?
- Take the largest sparse matrices from the UF Sparse collection.
- Download all the unsymmetric and rectangular matrices (more rows than columns) with more than 1M nonzero entries.

Description of Experiment

For each matrix, ran as many steps as 2 GBytes storage on my desktop permitted. The number of steps N was taken as

$$N = \max(4.5e7/m, 1200)$$

where m is the number of rows of the sparse matrix.

Spar3Bnd returns an $N \times N$ upper triangular matrix A_0 with bandwidth $K + 1$.

Description – 2

LAPACK dgesvd is used to compute singular values of the returned block. If the block size is K , the input matrices

- A_2, A_1, A_0 to dgesvd are square of size $N - 2K, N - K, N$.
- The singular values of A_2, A_1, A_0 interlace
- Claim observed convergence if relative difference less than $1.e-10$.

106 Sparse Matrices

17 were too large – $m > 450k$, segmentation fault in LAPACK or BLAS

Converged Singular Values	$> \sqrt{N}$	$5 - \sqrt{N}$	1 - 4	0
Matrices	38 Many	21 Some	12 A Few	18 None

Table 1: N was the number of steps of SparBnd3

38 matrices had many converged singular values.

Notes

- The problems with fewer converged singular values tended to be the larger ones (for which case we could perform relatively few steps before exhausting the 2 GBytes of RAM)
- Strictly diagonal rows and columns should be permuted to the top and left and those elimination steps should be skipped. A preconditioning step should select what rows and columns to eliminate.

Is BLAS-3 Worth It?

Basis	Lanczos	PROPACK	ARPACK	Spar3Bnd
Vecs	$O(1)$	$2N$	$2N$	$4N$
Basis	Loss of Rank	Loss of Orthog	Keeps Orthog	Keeps Orthog
BLAS	BLAS-1	BLAS-1	BLAS-2	BLAS-3

Need some experimentation comparing Spar3Bnd to other algorithms.

Parallelization of Spar3Bnd

- Distributed memory case. BLAS-3 operations are expected to work better to deal with issues of latency and bandwidth than do BLAS-2 operations. Similarly the sparse matrix, dense matrix operations AX and $Y^T A$ are likely to be preferable to Ax and $y^T A$.
- For shared memory (or multi-core processors), we get good parallelization with a few OpenMP calls in the AX and $Y^T A$ routines – and by linking to a multi-thread enabled BLAS. Already works.

Next Stuff

- In some other talk, I'll show how to use Spar3Bnd to find x to minimize $\|Ax - b\|$.
Modification of LSQR.
- Preconditioning. Permute diagonal entries to upper left and wisely choose which elements to eliminate.
- Some data mining applications.