# A MATLAB interface for PRIMME for solving Eigenvalue and Singular Value problems

Andreas Stathopoulos and Lingfei Wu

Computer Science Department
College of William and Mary

Acknowledgment: National Science Foundation, DOE SciDAC

## The problems

$A$ large, sparse, Hermitian matrix:

Find *nev* eigenvalues and corresponding eigenvectors

$$\boxed{Ax_i = \lambda_i x_i}$$

$A$ large, sparse, $N \times M$ matrix

Find `nev` singular values and corresponding left and right singular vectors

$$\boxed{Av_i = \sigma_i u_i}$$

SVD problem solved as a Hermitian eigenvalue problem on

Normal equations $A^T A$ or $AA^T$,

Augmented matrix $[0\ A; A^T\ 0]$

## Available software

Lanczos-based, no preconditioning

- ARPACK (Sorensen, Lehoucq)
- TRLAN (Wu, Simon)
- Industrial strength Lanczos (Grimes, Lewis, Simon)

Preconditioned eigensolver packages with various methods

- ANASAZI (Baker, Thornquist, Lehoucq, Hetmaniuk)
- PRIMME (AS, J.M.)
- SLEPc (Roman et al.)

Specific method preconditioned eigensolver software

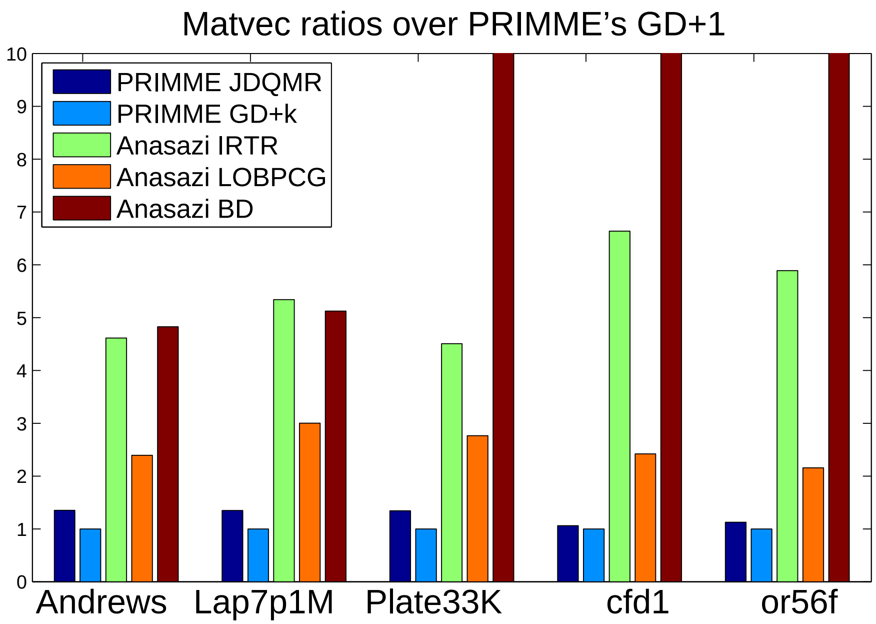
- BLOPEX (Knyazev)
- JADAMILU (Bollhoefer, Notay)

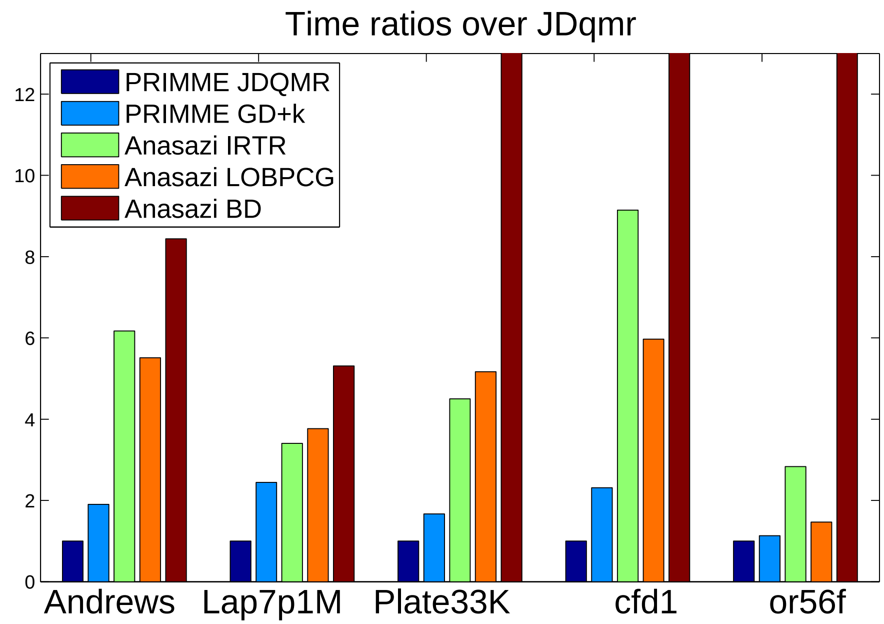
PRIMME: PReconditioned Iterative MultiMethod Eigensolver

- Near optimality through GD+k and JDQMR methods
- Over 12 methods accessible through PRIMME.
- Dynamic choice between best methods
- Block versions of methods
- Interior eigenvalues too
- Full set of defaults for non expert users
- Full customizability for expert users
- Parallel, high performance implementation
- C and Fortran interfaces, real and complex
- Accessible also in SLEPc

Download: `www.cs.wm.edu/~andreas`

# PRIMME shown robust and efficient



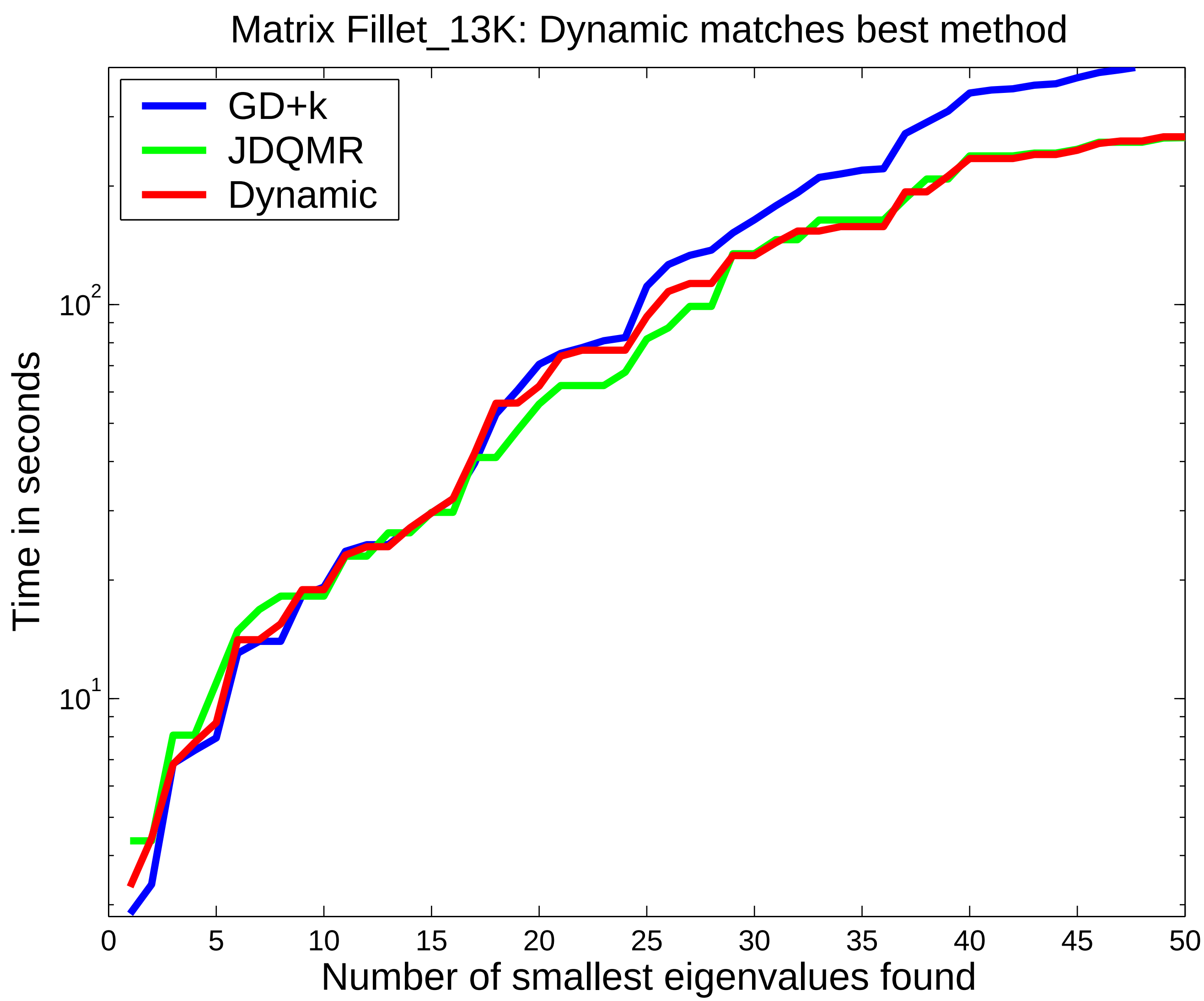




Typically:

GD+1 smallest number of matrix-vector ops

JDMQR lowest time (if matrix sparse enough)

# Dynamic method chooses between the fastest two algorithms



## PRIMME multi-layer interface – End user

```
#include "primme.h"

primme_params primme;
primme_Initialize(&primme);

primme.n = n;
primme.numEvals = 20;

primme.matrixMatvec          = MV(x,y,k)
primme.applyPreconditioner = PR(x,y,k)

ierr = dprimme(evals, evecs, rnorms, &primme);
```

Usually achieves full potential of the method

# PRIMME multi-layer interface – Advanced user

```
#include "primme.h"
primme_params primme;

primme.
   outputFile          = stdout              iseed                             = -1
   printLevel          = 5                   restarting.scheme                 = primme_thick
   numEvals            = 10                  restarting.maxPrevRetain          = 1
   aNorm               = 1.0                 correction.precondition           = 1
   eps                 = 1.0e-12             correction.robustShifts           = 1
   maxBasisSize        = 15                  correction.maxInnerIterations = -1
   minRestartSize      = 7                   correction.relTolBase             = 1.5
   maxBlockSize        = 1                   correction.convTest    = adaptive_ETolerance
   maxOuterIterations = 10000                correction.projectors.LeftQ   = 1
   maxMatvecs          = 300000              correction.projectors.LeftX   = 1
   target              = primme_smallest     correction.projectors.RightQ = 0
   numTargetShifts     = 0                   correction.projectors.SkewQ   = 0
   targetShifts        = 1.0 2.0             correction.projectors.RightX = 1
   locking             = 1                   correction.projectors.SkewX   = 1
   initSize            = 0                   matrixMatvec                      = MV(x,y,k)
   numOrthoConst       = 0;                  applyPreconditioner               = PR(x,y,k)

ierr = dprimme(evals, evecs, rnorms, &primme);
```

## PRIMME in MATLAB

Benefits to PRIMME users

- Optimized Sparse, Block Matrix-Vector (MV) function
- Optimized libraries for Sparse matrix inversion and ILU preconditioing
- Optimized BLAS/LAPACK libraries
- Ease of use/development
- Easier to build and experiment on a SVD solver

Benefits to MATLAB users

- Availability of a preconditioned eigensolver
- Availability of a preconditioned singular value solver
- As robust and easy to use as `eigs()` but much faster

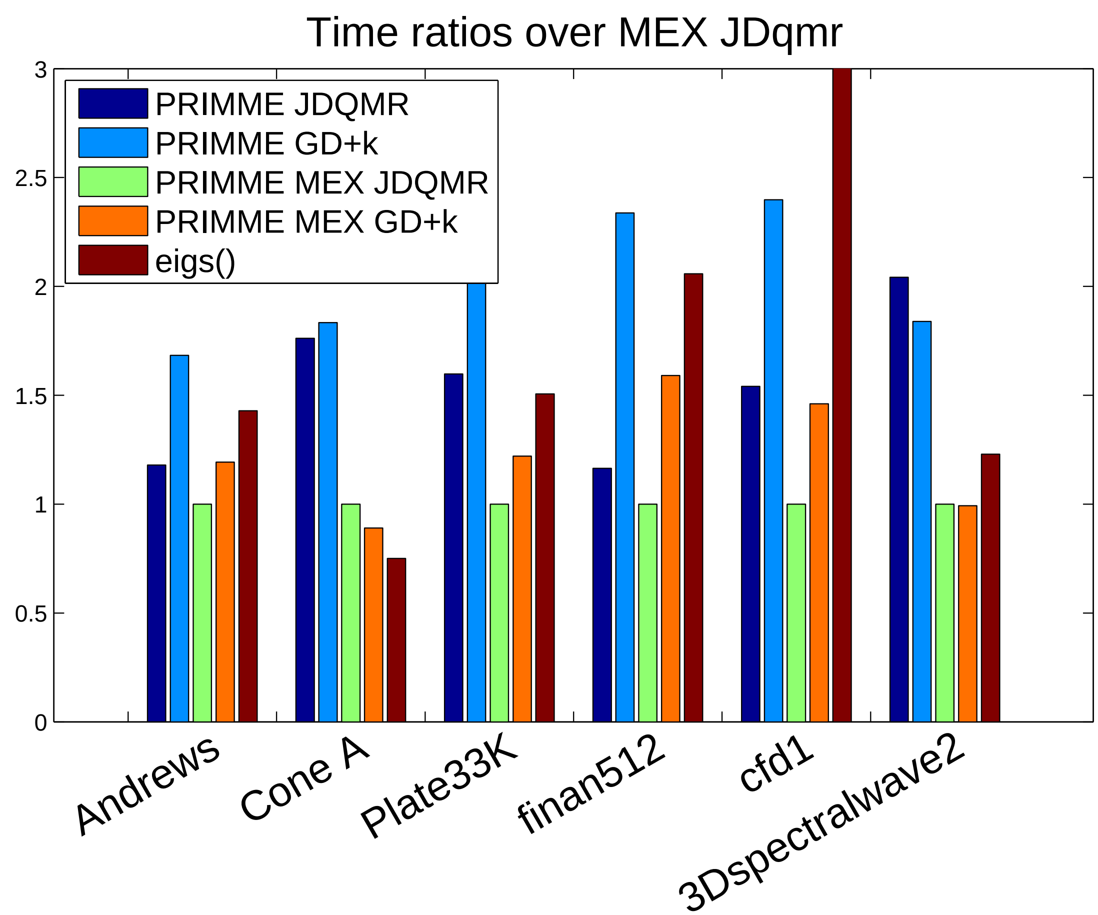
## Interface similar to `eigs()`

```
[evals] =
[evecs, evals] =
[evecs, evals, resnorms] =
[evecs, evals, resnorms, primmeStats] =
  primme_eigs(A)
  primme_eigs(A, numEvals)
  primme_eigs(A, numEvals, target)
  primme_eigs(A, numEvals, target, opts)
  primme_eigs(A, numEvals, target, opts, eigsMethod)
  primme_eigs(A, numEvals, target, opts, eigsMethod, P)
  primme_eigs(A, numEvals, target, opts, eigsMethod, P1,P2)
  primme_eigs(A, numEvals, target, opts, eigsMethod, Pfun)
  primme_eigs(Afun, dim,...)
```

We allow `Afun` even for smallest magnitude eigenvalues

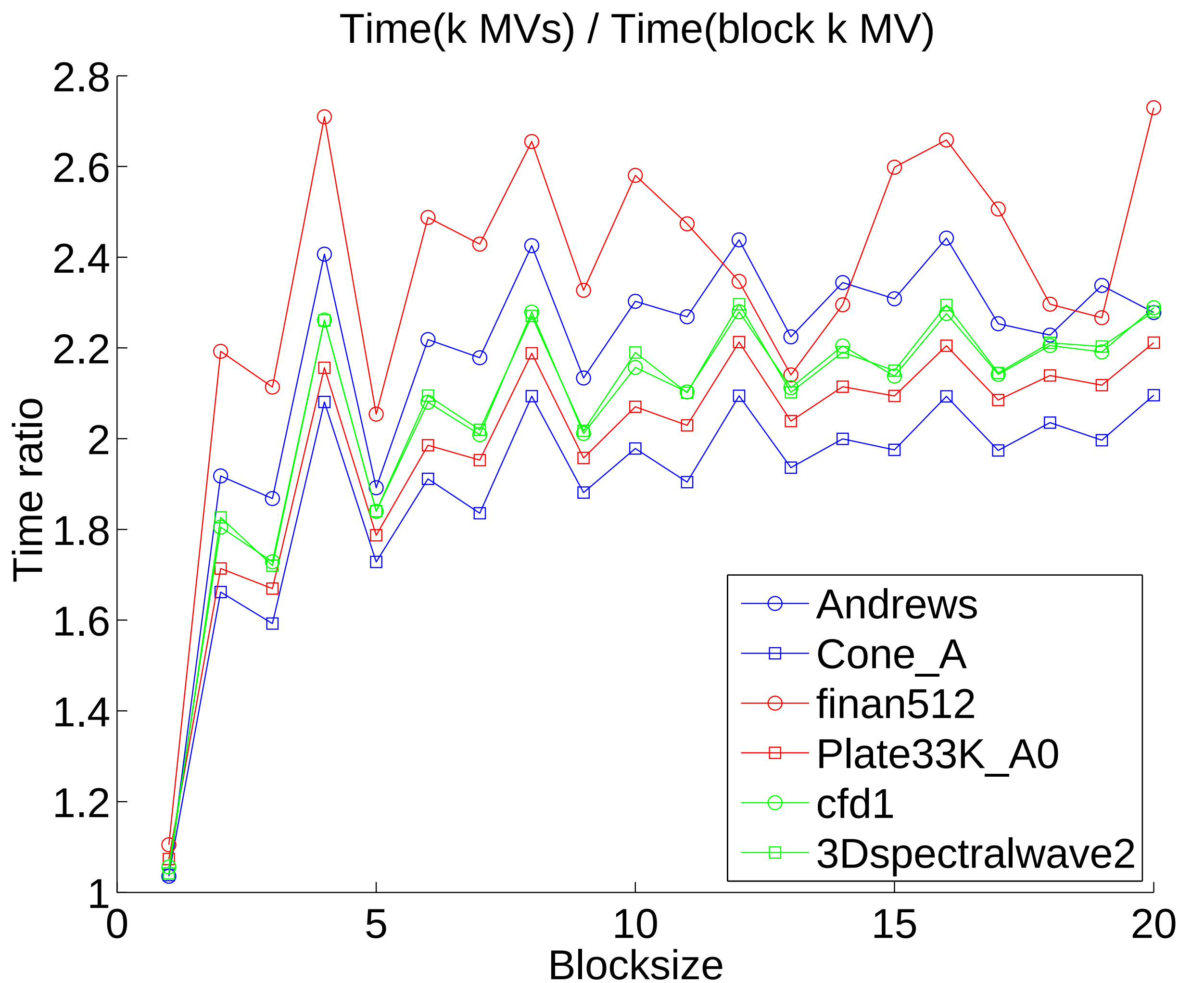
## PRIMME vs PRIMME MEX vs `eigs()` — No preconditioner

- Five smallest eigenvalues — eigs(Afun) to avoid inversion
- PRIMME compiled -O3 including SparseMatVec,BLAS,LAPACK
- PRIMME MEX uses MATLAB's SparseMatVec, BLAS, LAPACK




- MATLAB Library benefits
- PRIMME MEX faster than `eigs()`

## Performance of MATLAB's Block Sparse Matvec

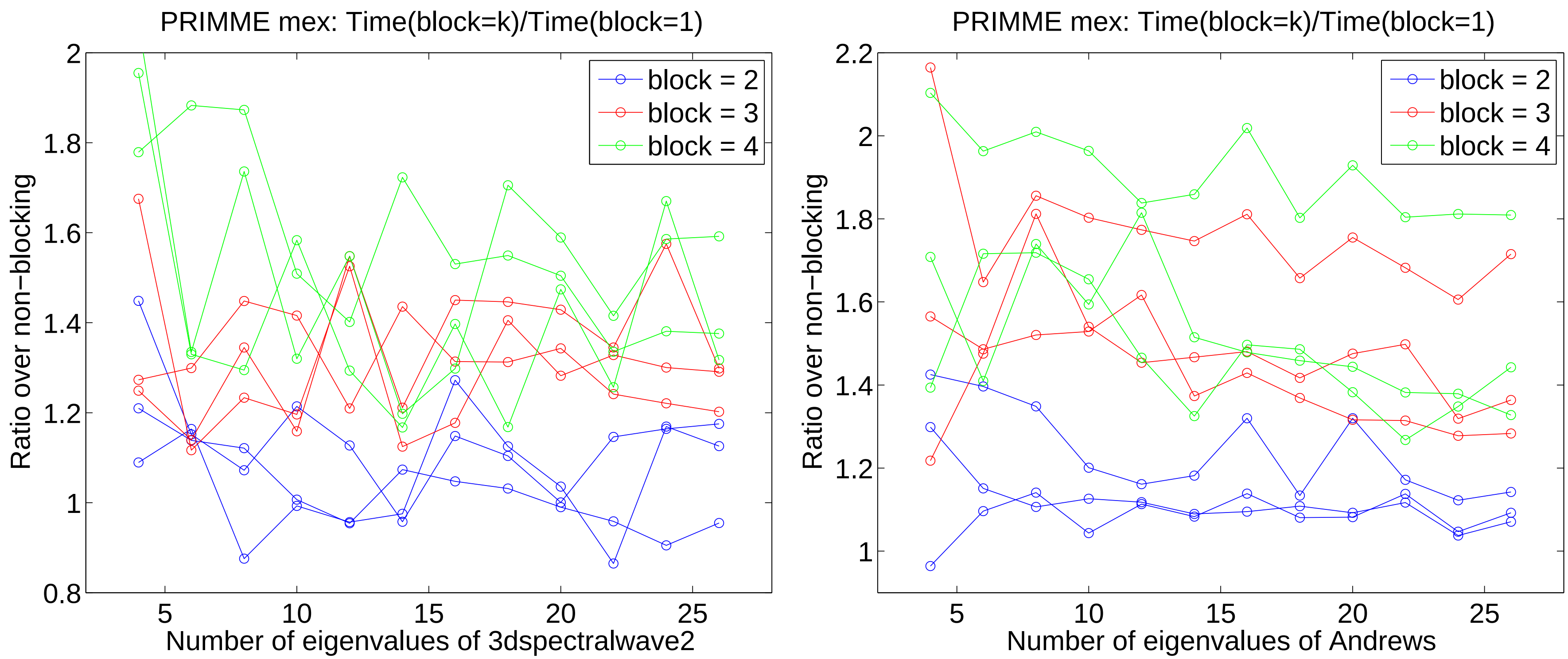



Block size of 2 biggest gain. Larger block no much better.

# PRIMME MEX with block size of 2

In Block Krylov methods # Matvecs increase almost linearly with block size

Using the MATLAB routines, block=2 offers better robustness with only small additional expense



## Using PRIMME for SVD

- Allow access to full functionality of PRIMME
- Allow choice of normal equations (ATA) or augmented matrix $B$ (OAAO)
- Allow for various preconditioning techniques:
  - $P \approx (A^T A)^{-1}$ or $P \approx B^{-1} = \begin{bmatrix} 0 & A^{-1} \\ A^{-T} & 0 \end{bmatrix}$ directly
  - $P \approx A^{-1}$ explicitly or through ILU: $P = U^{-1} L^{-1} \approx A^{-1}$. Use as:

$$PP^T \approx (A^T A)^{-1} \text{ or as } \begin{bmatrix} 0 & P \\ P^T & 0 \end{bmatrix} \approx B^{-1}$$

    (matrix products not formed)
  - `Pfun` user provided function for implementing any of the above $P$

# Using PRIMME for SVD

```
[S] =
[U, S, V] =
[U, S, V, norms, primmeout] =
primme_svds(A)
primme_svds(A, numSvs)
primme_svds(A, numSvs, target)
primme_svds(A, numSvs, target, opts)
primme_svds(A, numSvs, target, opts, eigsMethod)
primme_svds(A, numSvs, target, opts, eigsMethod, svdsMethod)
primme_svds(A, numSvs, target, opts, eigsMethod, svdsMethod, P)
primme_svds(A, numSvs, target, opts, eigsMethod, svdsMethod, P1,P2)
primme_svds(A, numSvs, target, opts, eigsMethod, svdsMethod, Pfun)
primme_svds(Afun, M, N,...)
```

## What threshold to converge to?

Let $(u_i, \sigma_i, v_i)$ approximate singular triplet. Define:

$$\begin{aligned} r_v &= \|Av_i - \sigma_i u_i\| \\ r_u &= \|A^T u_i - \sigma_i v_i\| \\ r_{ATA} &= \|A^T A v_i - \sigma_i^2 v_i\| \\ r_B &= \|B \begin{bmatrix} v \\ u \end{bmatrix} - \sigma_i \begin{bmatrix} v \\ u \end{bmatrix} \| \end{aligned}$$

If $\|v_i\| = 1$, $\|u_i\| = \|Av_i/\sigma_i\| = 1$, then $r_v = 0$ and

$$r_u = \frac{r_{ATA}}{\sigma_i} = r_B \sqrt{2}$$

Want to guarantee $r_u < \delta \|A\|$, so converge each method to

$$\begin{aligned} r_{ATA} &< \sigma_i \, \delta \, \|A\| \\ r_B &< \sqrt{2} \, \delta \, \|A\| \end{aligned}$$

## Which SVD method?

### Convergence speed issue

| | |
|---|---|
| ATA | very fast for largest SVs (squared gaps)<br>slow for smallest but still much faster than OAAO |
| OAAO | slower for largest eigenvalues<br>extremely slow and not robust for smallest (interior) SVs |

### Accuracy issue

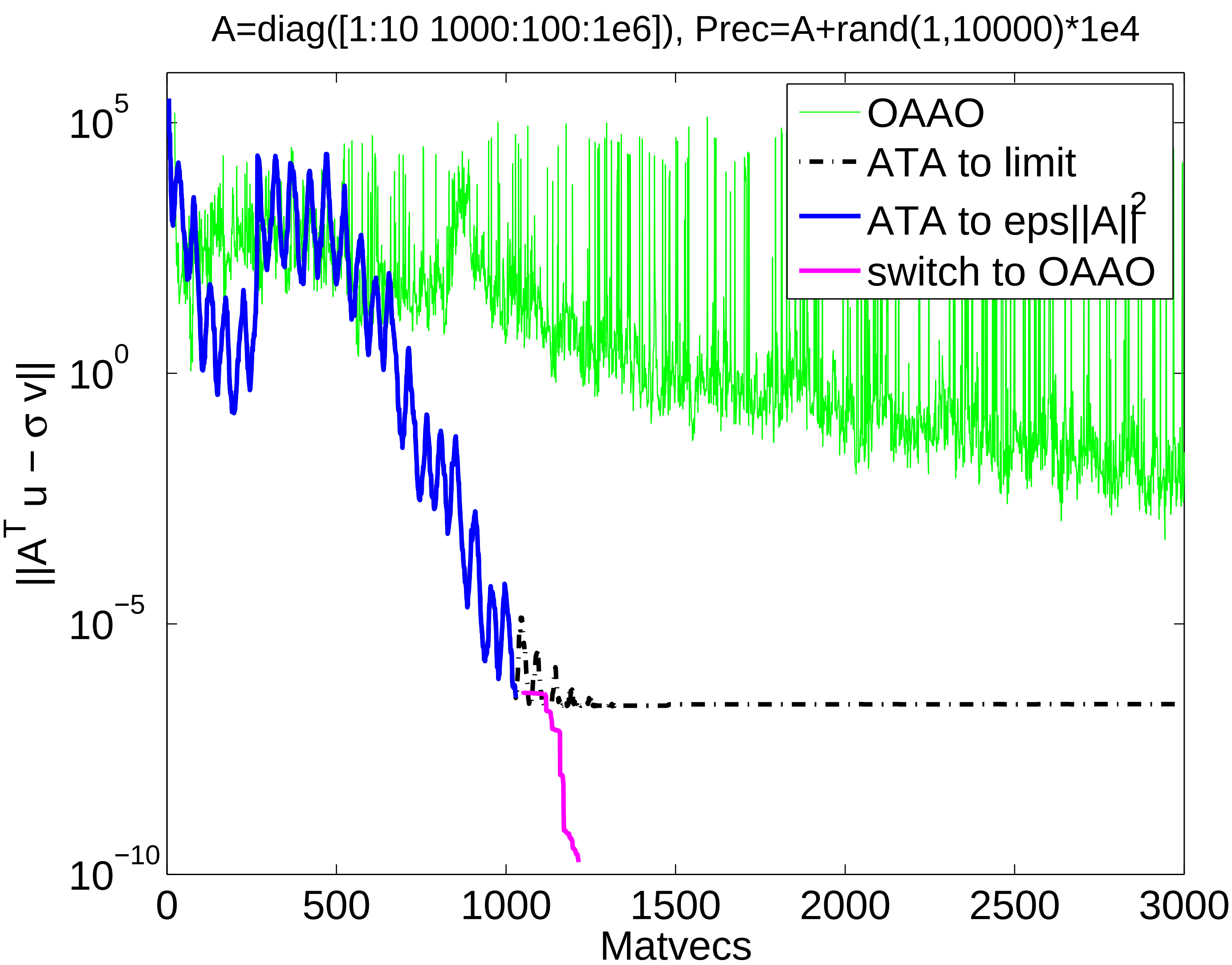
| | |
|---|---|
| OAAO | can converge up to $\|A\|\varepsilon_{mach}$ |
| ATA | can only converge up to $\|A\|^2\varepsilon_{mach}$<br>$\Rightarrow$ for small $\sigma_i$ cannot reach needed $\sigma_i\delta\|A\|$, if $\delta < \varepsilon_{mach}\|A\|/\sigma_i$ |

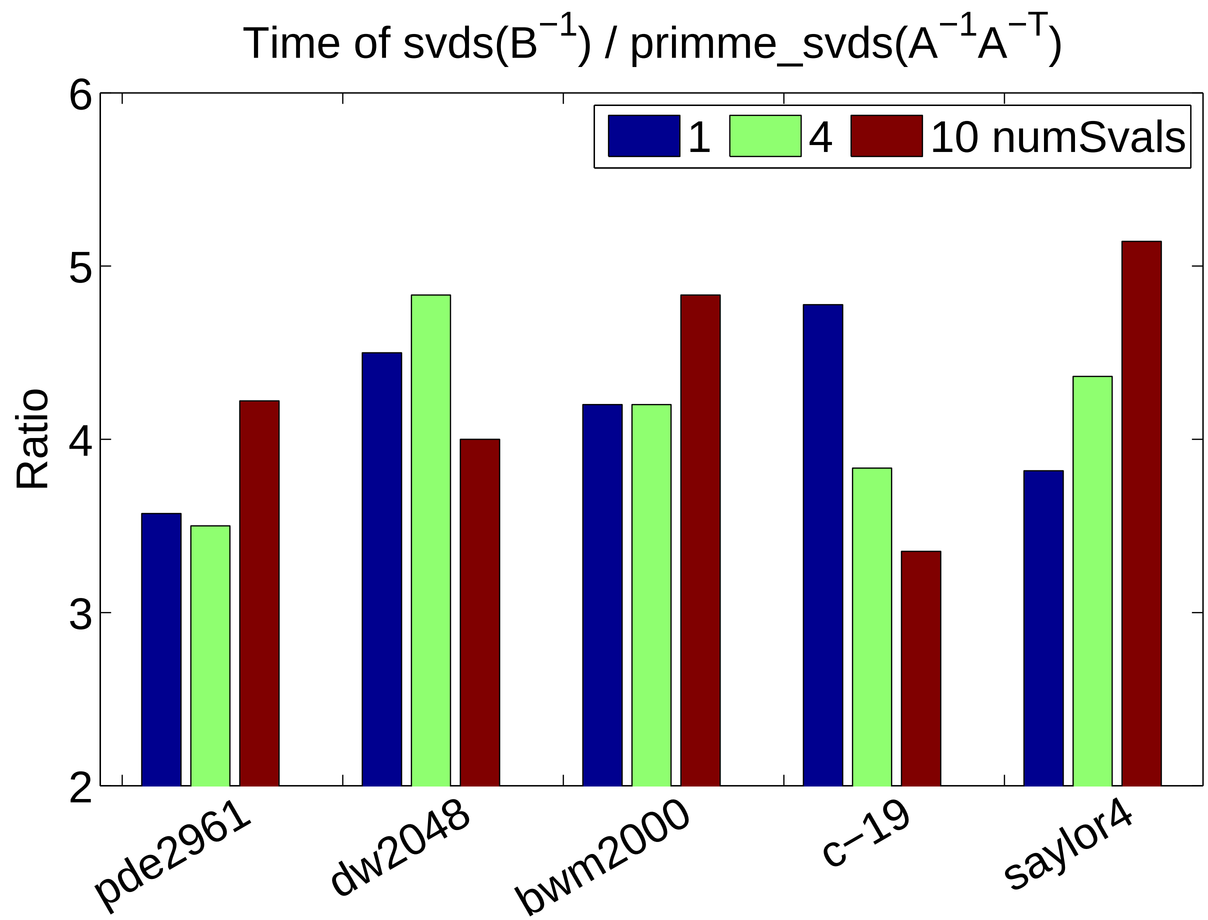
### Our PRIMME SVDS solution:

Use ATA to residual threshold $\max\left(\sigma_i\delta\|A\|, \|A\|^2\varepsilon_{mach}\right)$

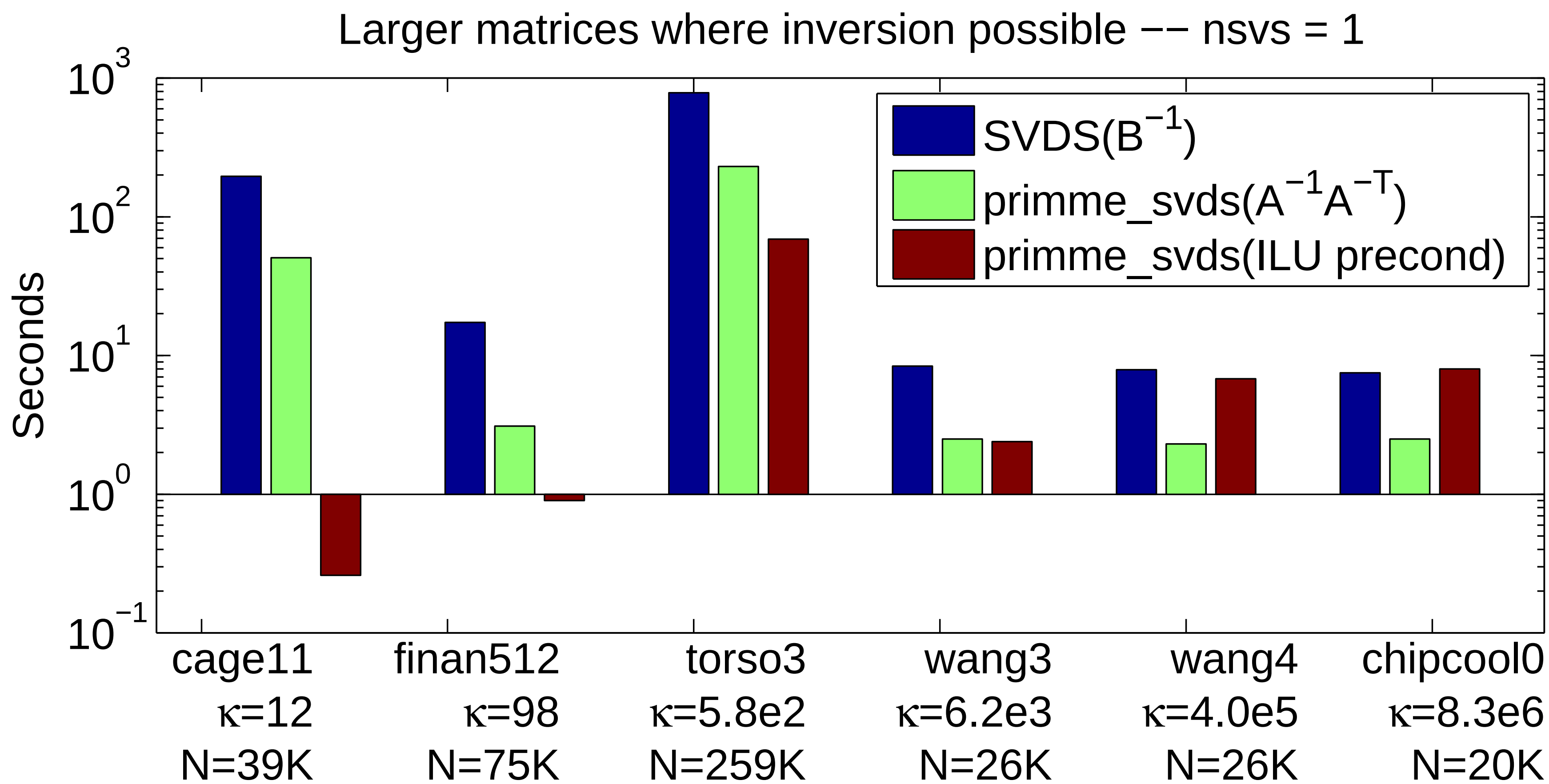
Use OAAO to improve the ATA approximations to the required threshold

# Accuracy limit and dynamic switching



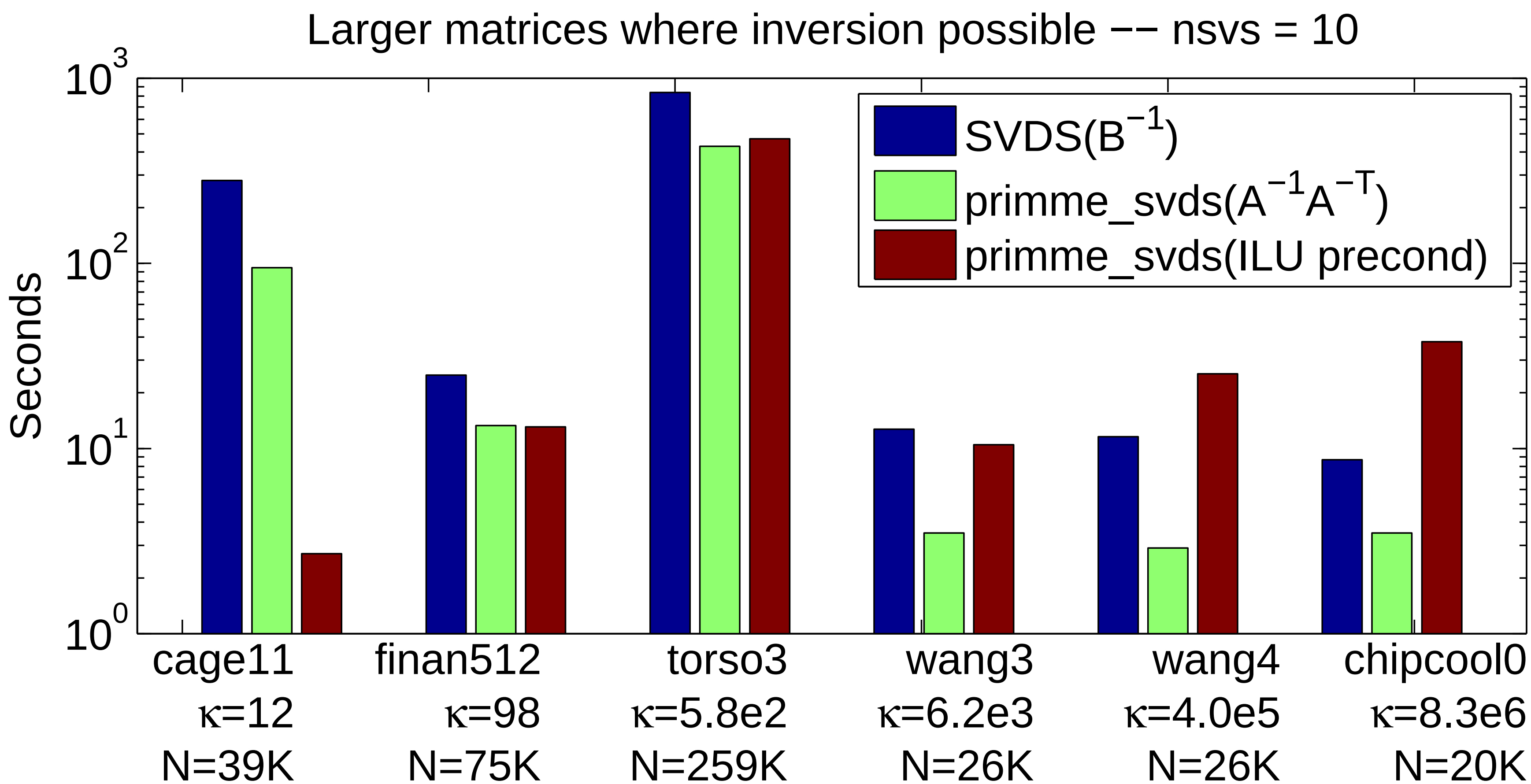



When $A = LU$ or $A \approx LU$, store $L^T, U^T$ for faster memory access.
Still less memory than svds inverting $B$.




Shift-invert speedup factor 4

When preconditioning effective speedup 1000.




Preconditioned benefits decrease when more values needed

Shift-invert still speedup 2-4

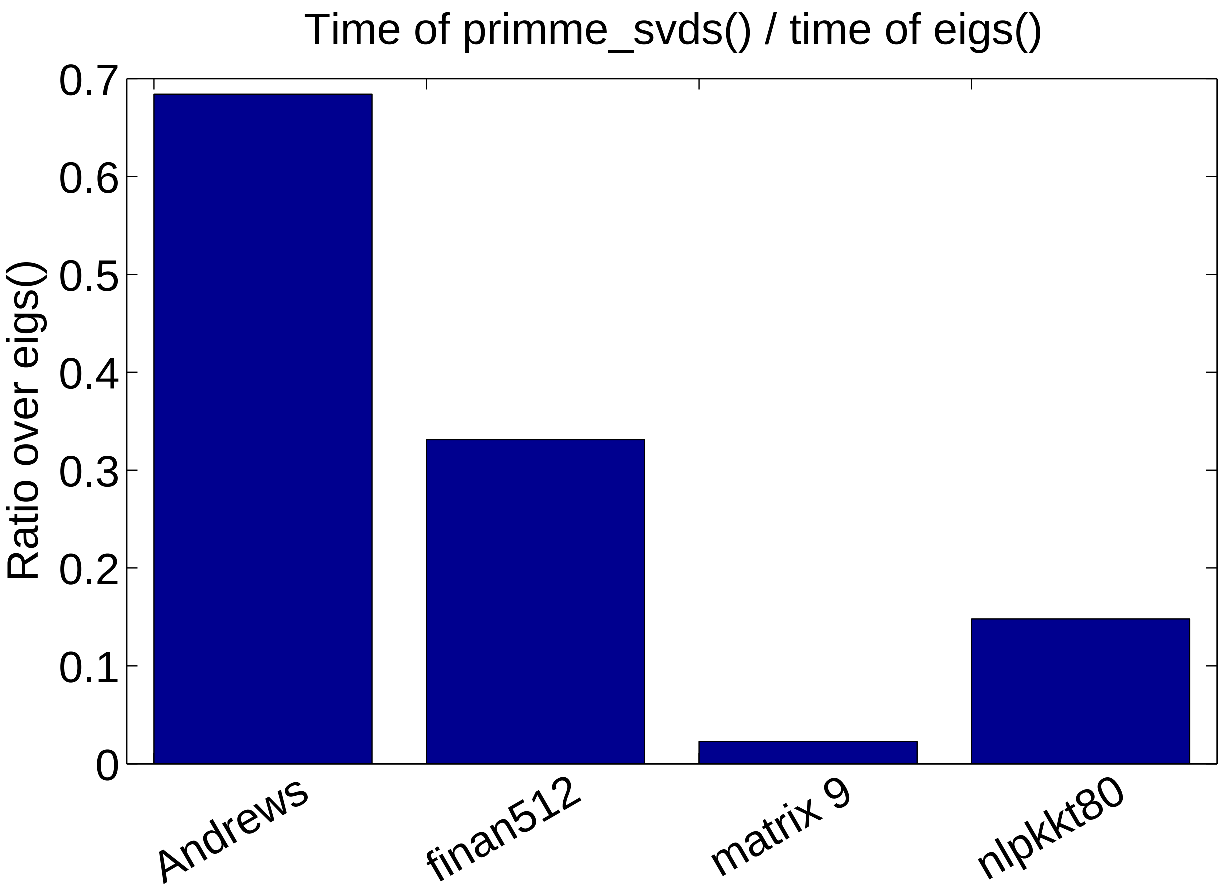
Factorization not possible, PRIMME the only alternative

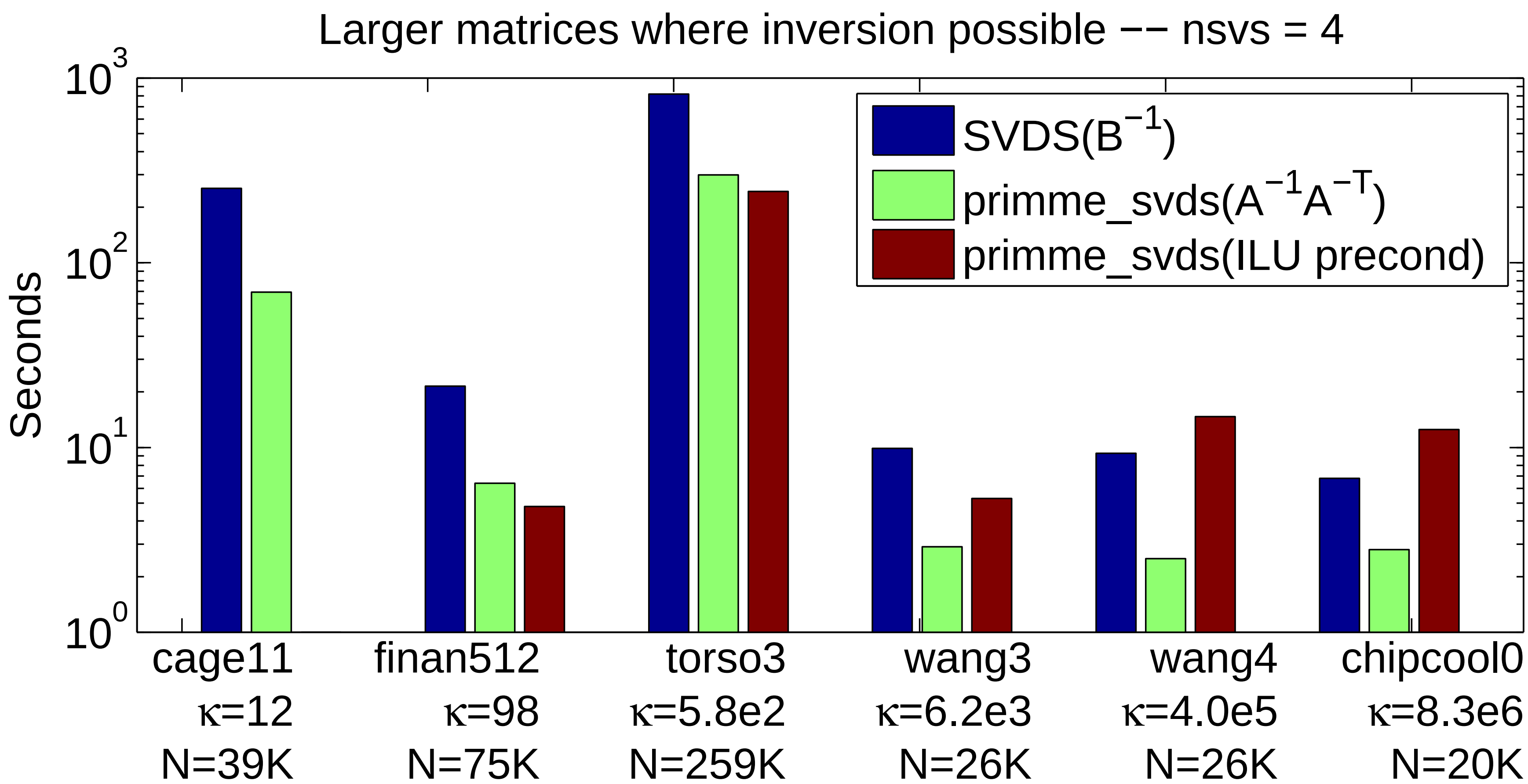
| matrix: | cage14 | | dielFilterV2real | | G3_circuit | |
|---|---|---|---|---|---|---|
| $N$ | 1,505,785 | | 1,157,456 | | 1,585,478 | |
| $\kappa(A)$ | 12 | | 6.0E7 | | 2.2E7 | |
| | MV | Sec | MV | Sec | MV | Sec |
| ilu(A) | – | 1.3 | – | 5.8 | – | 0.2 |
| primme_svds | 128 | 61.2 | 3494 | 4198 | 101595 | 33077 |

## Conclusions

- MATLAB interface to PRIMME
- MATLAB's optimized libraries and sparse Matvec improve PRIMME
- PRIMME svds() more flexible and efficient than svds()
- Dynamic switching of ATA and OAAO methods in primme_svds()

## No preconditioning — 10 largest svs




Larger matrices where inversion possible −− nsvs = 4
Seconds
$10^0$
$10^1$
$10^2$
$10^3$
SVDS($B^{-1}$)
primme_svds($A^{-1}A^{-T}$)
primme_svds(ILU precond)
cage11 κ=12 N=39K
finan512 κ=98 N=75K
torso3 κ=5.8e2 N=259K
wang3 κ=6.2e3 N=26K
wang4 κ=4.0e5 N=26K
chipcool0 κ=8.3e6 N=20K