A Robust and Efficient Parallel SVD Solver Based on Restarted Lanczos Bidiagonalization

Jose E. Roman

Universidad Politécnica de Valencia, Spain

(joint work with V. Hernandez and A. Tomas)

Harrachov 2007

UNIVERSIDAD POLITECNICA DE VALENCIA

grycap
Grupo de Redes y Computación de Altas Prestaciones

Outline

Overview of SLEPc

What Users Need

- Abstraction of mathematical objects: vectors and matrices
- Efficient linear solvers (direct or iterative)
- Easy programming interface
- Run-time flexibility, full control over the solution process
- Parallel computing, mostly transparent to the user

- State-of-the-art eigensolvers and SVD solvers
- Spectral transformations

What Users Need

Provided by PETSc

- Abstraction of mathematical objects: vectors and matrices
- Efficient linear solvers (direct or iterative)
- Easy programming interface
- Run-time flexibility, full control over the solution process
- Parallel computing, mostly transparent to the user

Provided by SLEPc

- State-of-the-art eigensolvers and SVD solvers
- Spectral transformations

Summary

PETSc: Portable, Extensible Toolkit for Scientific Computation

Software for the scalable (parallel) solution of algebraic systems arising from partial differential equation (PDE) simulations

- Developed at Argonne National Lab since 1991
- Usable from C, C++, Fortran77/90
- Focus on abstraction, portability, interoperability
- Extensive documentation and examples
- Freely available and supported through email

Current version: 2.3.3 (released May 2007)

`http://www.mcs.anl.gov/petsc`

Summary

SLEPc: Scalable Library for Eigenvalue Problem Computations

A *general* library for solving large-scale sparse eigenproblems on parallel computers

- For standard and generalized eigenproblems
- For real and complex arithmetic
- For Hermitian or non-Hermitian problems

Current version: 2.3.3 (released June 2007)

`http://www.grycap.upv.es/slepc`

PETSc/SLEPc Numerical Components

PETSc

Nonlinear Systems		
Line Search	Trust Region	Other

Time Steppers			
Euler	Backward Euler	Pseudo Time Step	Other

Krylov Subspace Methods							
GMRES	CG	CGS	Bi-CGStab	TFQMR	Richardson	Chebychev	Other

Preconditioners						
Additive Schwarz	Block Jacobi	Jacobi	ILU	ICC	LU	Other

Matrices				
Compressed Sparse Row	Block Compressed Sparse Row	Block Diagonal	Dense	Other

Vectors

Index Sets			
Indices	Block Indices	Stride	Other

PETSc/SLEPc Numerical Components

PETSc

Nonlinear Systems		
Line Search	Trust Region	Other

Time Steppers			
Euler	Backward Euler	Pseudo Time Step	Other

Krylov Subspace Methods							
GMRES	CG	CGS	Bi-CGStab	TFQMR	Richardson	Chebychev	Other

Preconditioners						
Additive Schwarz	Block Jacobi	Jacobi	ILU	ICC	LU	Other

Matrices				
Compressed Sparse Row	Block Compressed Sparse Row	Block Diagonal	Dense	Other

Vectors

Index Sets			
Indices	Block Indices	Stride	Other

SLEPc

SVD Solvers			
Cross Product	Cyclic Matrix	Lanczos	Thick Res. Lanczos

Eigensolvers			
Krylov-Schur	Arnoldi	Lanczos	Other

Spectral Transform			
Shift	Shift-and-invert	Cayley	Fold

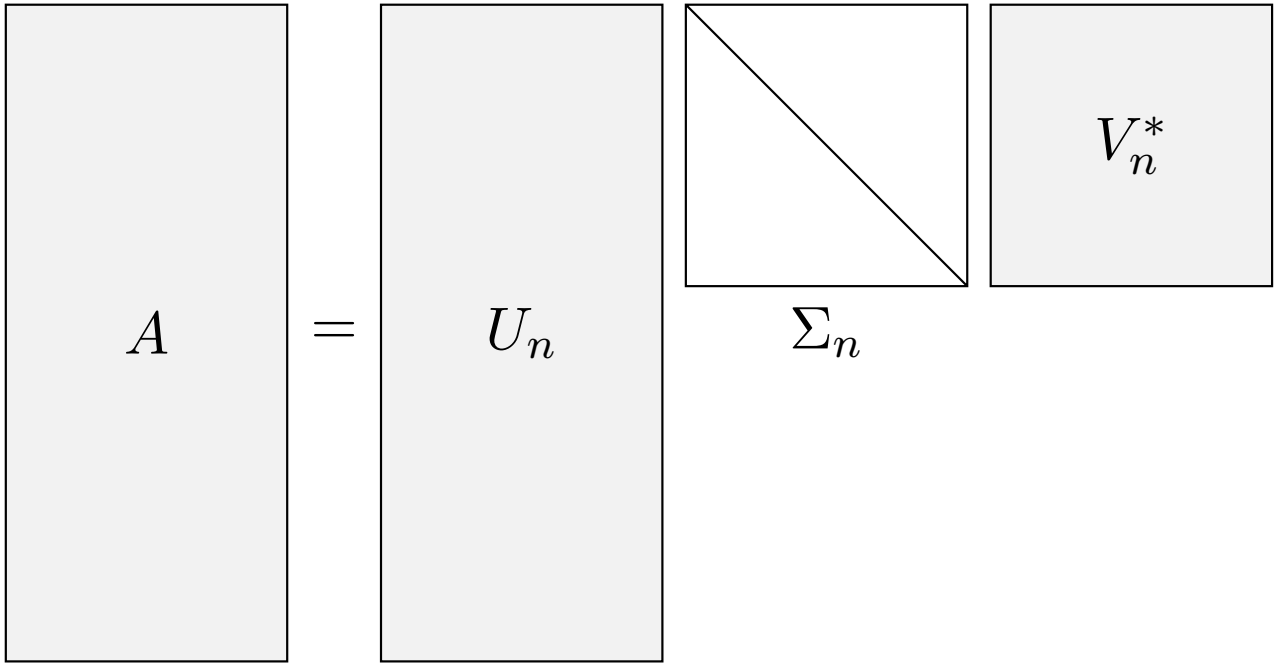
Singular Value Decomposition (SVD)

$$A = U\Sigma V^*$$

where

- A is an $m \times n$ rectangular matrix
- $U = [u_1, u_2, \ldots, u_m]$ is a $m \times m$ unitary matrix
- $V = [v_1, v_2, \ldots, v_n]$ is a $n \times n$ unitary matrix
- Σ is a $m \times n$ diagonal matrix with entries $\Sigma_{ii} = \sigma_i$
- $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$
- If A is real, U and V are real and orthogonal
- Each (σ_i, u_i, v_i) is called a singular triplet

Thin SVD



In SLEPc we compute a *partial* SVD, that is, only a subset of the singular triplets

SVD Solvers Based on EPS

Cross product matrix

$$A^*Ax = \lambda x \qquad AA^*y = \lambda y$$

The eigenvalues are $\lambda_i = \sigma_i^2$ and the eigenvectors $x_i = v_i$ or $y_i = u_i$

SVD Solvers Based on EPS

Cross product matrix

$$A^*Ax = \lambda x \qquad AA^*y = \lambda y$$

The eigenvalues are $\lambda_i = \sigma_i^2$ and the eigenvectors $x_i = v_i$ or $y_i = u_i$

Cyclic matrix

$$H(A)x = \lambda x \qquad H(A) = \begin{bmatrix} 0 & A \\ A^* & 0 \end{bmatrix}$$

The eigenvalues are $\pm\sigma_i$ and the eigenvectors $\frac{1}{\sqrt{2}} \begin{bmatrix} \pm u_i \\ v_i \end{bmatrix}$

Basic Usage

Usual steps for solving an SVD problem with SLEPc:

1. Create an SVD object
2. Define the SVD problem
3. (Optionally) Specify options for the solution
4. Run the SVD solver
5. Retrieve the computed solution
6. Destroy the SVD object

All these operations are done via a generic interface, common to all the SVD solvers

Simple Example

```
SVD         svd;     /*  singular value solver context  */
Mat         A;       /*  matrix                         */
Vec         u, v;    /*  singular vectors               */
PetscReal   sigma;   /*  singular value                 */
```

Simple Example

```
SVD       svd;     /*  singular value solver context  */
Mat       A;       /*  matrix                         */
Vec       u, v;    /*  singular vectors               */
PetscReal sigma;   /*  singular value                 */

SVDCreate(PETSC_COMM_WORLD, &svd);
SVDSetOperator(svd, A);
SVDSetFromOptions(svd);
```

Simple Example

```
SVD         svd;     /*  singular value solver context  */
Mat         A;       /*  matrix                         */
Vec         u, v;    /*  singular vectors               */
PetscReal   sigma;   /*  singular value                 */

SVDCreate(PETSC_COMM_WORLD, &svd);
SVDSetOperator(svd, A);
SVDSetFromOptions(svd);

SVDSolve(svd);
```

Simple Example

```
SVD       svd;     /*  singular value solver context  */
Mat       A;       /*  matrix                         */
Vec       u, v;    /*  singular vectors               */
PetscReal sigma;   /*  singular value                 */

SVDCreate(PETSC_COMM_WORLD, &svd);
SVDSetOperator(svd, A);
SVDSetFromOptions(svd);

SVDSolve(svd);

SVDGetConverged(svd, &nconv);
for (i=0; i<nconv; i++) {
  SVDGetSingularTriplet(svd, i, &sigma, u, v);
}
```

Simple Example

```
SVD       svd;     /*  singular value solver context  */
Mat       A;       /*  matrix                         */
Vec       u, v;    /*  singular vectors               */
PetscReal sigma;   /*  singular value                 */

SVDCreate(PETSC_COMM_WORLD, &svd);
SVDSetOperator(svd, A);
SVDSetFromOptions(svd);

SVDSolve(svd);

SVDGetConverged(svd, &nconv);
for (i=0; i<nconv; i++) {
  SVDGetSingularTriplet(svd, i, &sigma, u, v);
}

SVDDestroy(svd);
```

Run-Time Examples

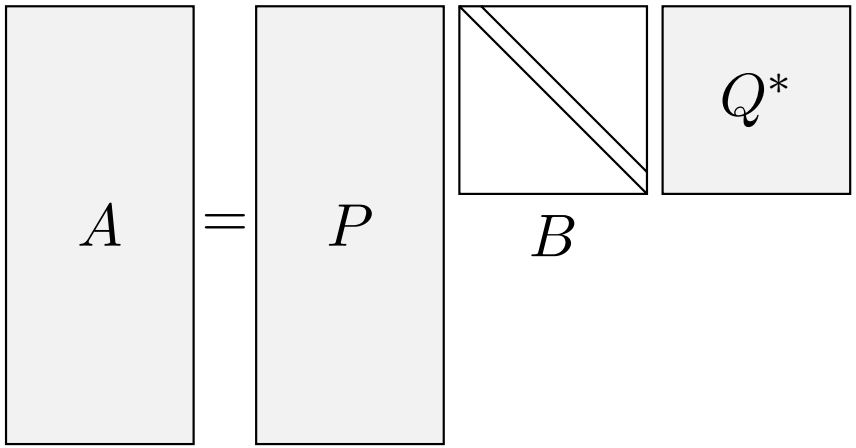
```
% program -svd_type lanczos -svd_tol 1e-12 -svd_max_it 200

% program -svd_type trlanczos -svd_nsv 4

% program -svd_type cross -svd_eps_type krylovschur
          -svd_ncv 30 -svd_smallest
          -svd_monitor_draw

% program -svd_type cyclic -svd_eps_type arpack
          -svd_st_type sinvert -svd_st_shift 1

% mpirun -np 16 program ...
```

Restarted Lanczos Bidiagonalization

Bidiagonalization

Compute the SVD in two stages [Golub and Kahan, 1965]:

1. $A = PBQ^*$

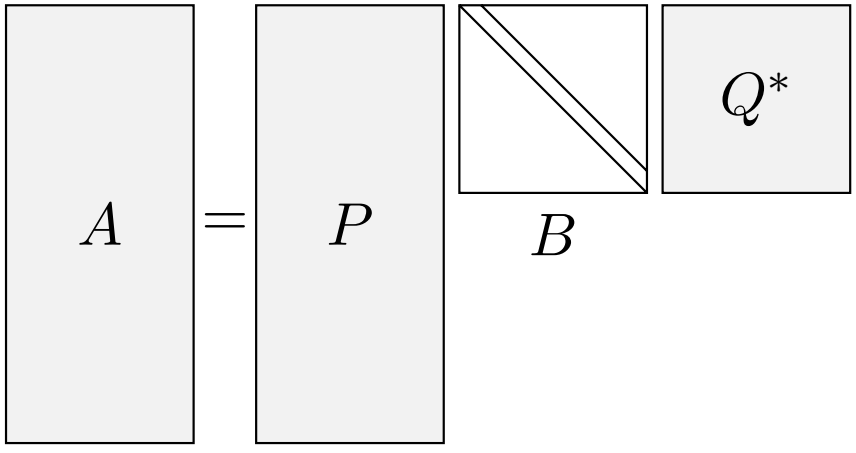


2. $B = X\Sigma Y^*$, with $U = PX$ and $V = QY$

Bidiagonalization

Compute the SVD in two stages [Golub and Kahan, 1965]:

1. $A = PBQ^*$



2. $B = X\Sigma Y^*$, with $U = PX$ and $V = QY$

Lanczos bidiagonalization computes part of the info: P_k, B_k, Q_k
$\rightarrow$ Ritz approximations: $\tilde{\sigma}_i, \quad \tilde{u}_i = P_k x_i, \quad \tilde{v}_i = Q_k y_i$

SLEPc

Lanczos Bidiagonalization

Equating the first k columns

$$AQ_k = P_k B_k$$
$$A^* P_k = Q_k B_k^* + \beta_k q_{k+1} e_k^*$$

$$B_k = \begin{bmatrix} \alpha_1 & \beta_1 & & & & \\ & \alpha_2 & \beta_2 & & & \\ & & \alpha_3 & \beta_3 & & \\ & & & \ddots & \ddots & \\ & & & & \alpha_{k-1} & \beta_{k-1} \\ & & & & & \alpha_k \end{bmatrix}$$

$$\alpha_j = p_j^* A q_j$$
$$\beta_j = p_j^* A q_{j+1}$$

Double recursion: $\alpha_j p_j = A q_j - \beta_{j-1} p_{j-1}, \quad \beta_j q_{j+1} = A^* p_j - \alpha_j q_j$

Golub-Kahan-Lanczos Bidiagonalization

Golub-Kahan-Lanczos algorithm

Choose a unit-norm vector q_1
Set $\beta_0 = 0$
For $j = 1, 2, \ldots, k$
$p_j = Aq_j - \beta_{j-1}p_{j-1}$
$\alpha_j = \|p_j\|_2$
$p_j = p_j/\alpha_j$
$q_{j+1} = A^*p_j - \alpha_j q_j$
$\beta_j = \|q_{j+1}\|_2$
$q_{j+1} = q_{j+1}/\beta_j$
end

Loss of orthogonality is an issue

Algorithm with Orthogonalization

Lanczos bidiagonalization with orthogonalization

Choose a unit-norm vector q_1

For $j = 1, 2, \ldots, k$

$p_j = Aq_j$

Orthogonalize p_j with respect to P_{j-1}

$\alpha_j = \|p_j\|_2$

$p_j = p_j/\alpha_j$

$q_{j+1} = A^* p_j$

Orthogonalize q_{j+1} with respect to Q_j

$\beta_j = \|q_{j+1}\|_2$

$q_{j+1} = q_{j+1}/\beta_j$

end

One-Sided Orthogonalization

Orthogonalizing right vectors is enough [Simon and Zha, 2000]

One-Sided Lanczos bidiagonalization

Choose a unit-norm vector q_1
Set $\beta_0 = 0$
For $j = 1, 2, \ldots, k$
$p_j = Aq_j - \beta_{j-1}p_{j-1}$
$\alpha_j = \|p_j\|_2$
$p_j = p_j/\alpha_j$
$q_{j+1} = A^*p_j$
Orthogonalize q_{j+1} with respect to Q_j
$\beta_j = \|q_{j+1}\|_2$
$q_{j+1} = q_{j+1}/\beta_j$
end

Restarted Bidiagonalization

Required k can be arbitrarily large (slow convergence, many singular triplets)

Problems: storage and computational effort

Solution: restart the computation when a certain k is reached

Restarted Bidiagonalization

Required k can be arbitrarily large (slow convergence, many singular triplets)

Problems: storage and computational effort

Solution: restart the computation when a certain k is reached

---

Explicit restart: re-run with a "better" q_1 (e.g. use Ritz vector associated to the first value)

Thick restart: a better alternative that avoids to explicitly compute a new initial vector

Idea: keep ℓ-dimensional subspace with relevant spectral information [Baglama and Reichel, 2005]

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$
\begin{aligned}
A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\
A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1}
\end{aligned}
$$

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$\begin{aligned} A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\ A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1} \end{aligned}$$

$$\tilde{Q}_{\ell+1} = [\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_\ell, q_{k+1}]$$

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$\begin{aligned} A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\ A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1} \end{aligned}$$

$\tilde{Q}_{\ell+1} = [\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_\ell, q_{k+1}]$ $\quad \tilde{v}_i = Q_k y_i$ right Ritz vectors

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$\begin{aligned} A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\ A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1} \end{aligned}$$

$\tilde{Q}_{\ell+1} = [\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_\ell, q_{k+1}]$ $\quad \tilde{v}_i = Q_k y_i$ right Ritz vectors

Residual of full decomposition

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$\begin{aligned} A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\ A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1} \end{aligned}$$

$\tilde{Q}_{\ell+1} = [\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_\ell, q_{k+1}]$

$\tilde{P}_{\ell+1} = [\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_\ell, \tilde{p}_{\ell+1}]$

$\tilde{v}_i = Q_k y_i$ right Ritz vectors

Residual of full decomposition

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$\begin{aligned} A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\ A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1} \end{aligned}$$

$\tilde{Q}_{\ell+1} = [\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_\ell, q_{k+1}]$ $\quad \tilde{v}_i = Q_k y_i$ right Ritz vectors

Residual of full decomposition

$\tilde{P}_{\ell+1} = [\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_\ell, \tilde{p}_{\ell+1}]$ $\quad \tilde{u}_i = P_k x_i$ left Ritz vectors

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$\begin{aligned} A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\ A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1} \end{aligned}$$

$\tilde{Q}_{\ell+1} = [\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_\ell, q_{k+1}]$ $\quad \tilde{v}_i = Q_k y_i$ right Ritz vectors

Residual of full decomposition

$\tilde{P}_{\ell+1} = [\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_\ell, \tilde{p}_{\ell+1}]$ $\quad \tilde{u}_i = P_k x_i$ left Ritz vectors

New left initial vector

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$\begin{aligned} A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\ A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1} \end{aligned}$$

$\tilde{Q}_{\ell+1} = [\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_\ell, q_{k+1}]$ $\quad \tilde{v}_i = Q_k y_i$ right Ritz vectors

Residual of full decomposition

$\tilde{P}_{\ell+1} = [\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_\ell, \tilde{p}_{\ell+1}]$ $\quad \tilde{u}_i = P_k x_i$ left Ritz vectors

New left initial vector

$$\tilde{p}_{\ell+1} = f/\|f\|, \quad f = Aq_{k+1} - \textstyle\sum_{i=1}^{\ell} \tilde{\rho}_i \tilde{u}_i, \quad \tilde{\alpha}_{\ell+1} = \|f\|$$

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$\begin{aligned} A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\ A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1} \end{aligned}$$

$\tilde{Q}_{\ell+1} = [\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_\ell, q_{k+1}]$ $\quad \tilde{v}_i = Q_k y_i$ right Ritz vectors

Residual of full decomposition

$\tilde{P}_{\ell+1} = [\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_\ell, \tilde{p}_{\ell+1}]$ $\quad \tilde{u}_i = P_k x_i$ left Ritz vectors

New left initial vector

$$\tilde{p}_{\ell+1} = f/\|f\|, \quad f = Aq_{k+1} - \textstyle\sum_{i=1}^{\ell} \tilde{\rho}_i \tilde{u}_i, \quad \tilde{\alpha}_{\ell+1} = \|f\|$$

$$\tilde{q}_{\ell+2} = g/\|g\|, \quad g = A^*\tilde{p}_{\ell+1} - \tilde{\alpha}_{\ell+1} q_{k+1}, \quad \tilde{\beta}_{\ell+1} = \|g\|$$

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$
\begin{aligned}
A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\
A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1}
\end{aligned}
$$

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$\begin{aligned} A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\ A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1} \end{aligned}$$

$$\tilde{B}_{\ell+1} = \begin{bmatrix} \tilde{\sigma}_1 & & & & \tilde{\rho}_1 \\ & \tilde{\sigma}_2 & & & \tilde{\rho}_2 \\ & & \ddots & & \vdots \\ & & & \tilde{\sigma}_\ell & \tilde{\rho}_\ell \\ & & & & \tilde{\alpha}_{\ell+1} \end{bmatrix}$$

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$\begin{aligned} A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\ A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1} \end{aligned}$$

$$\tilde{B}_{\ell+1} = \begin{bmatrix} \tilde{\sigma}_1 & & & \tilde{\rho}_1 \\ & \tilde{\sigma}_2 & & \tilde{\rho}_2 \\ & & \ddots & \vdots \\ & & \tilde{\sigma}_\ell & \tilde{\rho}_\ell \\ & & & \tilde{\alpha}_{\ell+1} \end{bmatrix}$$

$\tilde{\sigma}_i$ are Ritz values

Thick-Restart Lanczos Bidiagonalization

Compact Lanczos Bidiagonalization of order $\ell+1$:

$$\begin{aligned} A\tilde{Q}_{\ell+1} &= \tilde{P}_{\ell+1}\tilde{B}_{\ell+1} \\ A^*\tilde{P}_{\ell+1} &= \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1} \end{aligned}$$

$$\tilde{B}_{\ell+1} = \begin{bmatrix} \tilde{\sigma}_1 & & & \tilde{\rho}_1 \\ & \tilde{\sigma}_2 & & \tilde{\rho}_2 \\ & & \ddots & \vdots \\ & & \tilde{\sigma}_\ell & \tilde{\rho}_\ell \\ & & & \tilde{\alpha}_{\ell+1} \end{bmatrix}$$

$\tilde{\sigma}_i$ are Ritz values

$$\tilde{p}_{\ell+1} = f/\|f\|, \quad f = Aq_{k+1} - \sum_{i=1}^{\ell} \tilde{\rho}_i \tilde{u}_i, \quad \tilde{\alpha}_{\ell+1} = \|f\|$$

Thick-Restart Lanczos Bidiagonalization

The decomposition can be extended to order k and the Lanczos relations are maintained

$$\tilde{B}_k = \begin{bmatrix} \tilde{\sigma}_1 & & & & \tilde{\rho}_1 & & & \\ & \tilde{\sigma}_2 & & & \tilde{\rho}_2 & & & \\ & & \ddots & & \vdots & & & \\ & & & \tilde{\sigma}_\ell & \tilde{\rho}_\ell & & & \\ & & & & \tilde{\alpha}_{\ell+1} & \beta_{\ell+1} & & \\ & & & & & \ddots & \ddots & \\ & & & & & & \alpha_{k-1} & \beta_{k-1} \\ & & & & & & & \alpha_k \end{bmatrix}$$

Thick-Restart Lanczos Bidiagonalization

Thick-restarted Lanczos bidiagonalization

Input: Matrix A, initial unit-norm vector q_1,
and number of steps k

Output: $\ell \leq k$ Ritz triplets

1. Build an initial Lanczos bidiagonalization of order k
2. Compute Ritz approximations of the singular triplets
3. Truncate to a Lanczos bidiagonalization of order ℓ
4. Extend to a Lanczos bidiagonalization of order k
5. If not satisfied, go to step 2

SLEPc

Thick-Restart Lanczos Bidiagonalization

One-Sided Lanczos bidiagonalization – restarted

$$p_{\ell+1} = Aq_{\ell+1} - B_{1:\ell,\ell+1}P_\ell$$
$$\alpha_{\ell+1} = \|p_{\ell+1}\|_2,\ p_{\ell+1} = p_{\ell+1}/\alpha_{\ell+1}$$
For $j = \ell+1, \ell+2, \ldots, k$
  $q_{j+1} = A^* p_j$
  Orthogonalize q_{j+1} with respect to Q_j
  $\beta_j = \|q_{j+1}\|_2$
  $q_{j+1} = q_{j+1}/\beta_j$
  If $j < k$
    $p_{j+1} = Aq_{j+1} - \beta_j p_j$
    $\alpha_{j+1} = \|p_{j+1}\|_2$
    $p_{j+1} = p_{j+1}/\alpha_{j+1}$
  end
end

Enhancements for Parallel Efficiency

In general, eigensolvers require high-quality orthogonality for numerical robustness

- Classical Gram-Schmidt with selective refinement (DGKS criterion)

In parallel computations, the number of synchronizations should be reduced to a minimum [Hernandez *et al.*, 2007]

- Estimation of the norm
- Delayed normalization

Enhancements for Parallel Efficiency

For $j = \ell+1, \ell+2, \ldots, k$

$q_{j+1} = A^* p_j$

$c = Q_j^* q_{j+1}$

$\rho = \|q_{j+1}\|_2$

$\alpha_j = \|p_j\|_2$

$p_j = p_j/\alpha_j$

$q_{j+1} = q_{j+1}/\alpha_j$

$c = c/\alpha_j$

$\rho = \rho/\alpha_j$

$q_{j+1} = q_{j+1} - Q_j c$

$\beta_j = \sqrt{\rho^2 - \sum_{i=1}^{j} c_j^2}$

If $\beta_j < \eta\rho$

$c = Q_j^* q_{j+1}$

$\rho = \|q_{j+1}\|_2$

$q_{j+1} = q_{j+1} - Q_j c$

$\beta_j = \sqrt{\rho^2 - \sum_{i=1}^{j} c_j^2}$

end

$q_{j+1} = q_{j+1}/\beta_j$

If $j < k$

$p_{j+1} = A q_{j+1} - \beta_j p_j$

end

end

Enhancements for Parallel Efficiency

For $j = \ell+1, \ell+2, \ldots, k$

$q_{j+1} = A^* p_j$

$c = Q_j^* q_{j+1}$

$\rho = \|q_{j+1}\|_2$

$\alpha_j = \|p_j\|_2$

$p_j = p_j/\alpha_j$

$q_{j+1} = q_{j+1}/\alpha_j$

$c = c/\alpha_j$

$\rho = \rho/\alpha_j$

$q_{j+1} = q_{j+1} - Q_j c$

$\beta_j = \sqrt{\rho^2 - \sum_{i=1}^{j} c_j^2}$

DGKS criterion

If $\beta_j < \eta\rho$

$c = Q_j^* q_{j+1}$

$\rho = \|q_{j+1}\|_2$

$q_{j+1} = q_{j+1} - Q_j c$

$\beta_j = \sqrt{\rho^2 - \sum_{i=1}^{j} c_j^2}$

end

$q_{j+1} = q_{j+1}/\beta_j$

If $j < k$

$p_{j+1} = A q_{j+1} - \beta_j p_j$

end

end

Enhancements for Parallel Efficiency

For $j = \ell+1, \ell+2, \ldots, k$

$q_{j+1} = A^* p_j$

$c = Q_j^* q_{j+1}$

$\rho = \|q_{j+1}\|_2$

$\alpha_j = \|p_j\|_2$

$p_j = p_j/\alpha_j$

$q_{j+1} = q_{j+1}/\alpha_j$

$c = c/\alpha_j$

$\rho = \rho/\alpha_j$

$q_{j+1} = q_{j+1} - Q_j c$

$\beta_j = \sqrt{\rho^2 - \sum_{i=1}^{j} c_j^2}$

Estimation of the norm

If $\beta_j < \eta\rho$

$c = Q_j^* q_{j+1}$

$\rho = \|q_{j+1}\|_2$

$q_{j+1} = q_{j+1} - Q_j c$

$\beta_j = \sqrt{\rho^2 - \sum_{i=1}^{j} c_j^2}$

end

$q_{j+1} = q_{j+1}/\beta_j$

If $j < k$

$p_{j+1} = A q_{j+1} - \beta_j p_j$

end

end

SLEPc

Enhancements for Parallel Efficiency

For $j = \ell+1, \ell+2, \ldots, k$

$q_{j+1} = A^* p_j$

$c = Q_j^* q_{j+1}$

$\rho = \|q_{j+1}\|_2$

$\alpha_j = \|p_j\|_2$

$p_j = p_j/\alpha_j$

$q_{j+1} = q_{j+1}/\alpha_j$

$c = c/\alpha_j$

$\rho = \rho/\alpha_j$

$q_{j+1} = q_{j+1} - Q_j c$

$\beta_j = \sqrt{\rho^2 - \sum_{i=1}^{j} c_j^2}$

Delayed normalization

If $\beta_j < \eta\rho$

$c = Q_j^* q_{j+1}$

$\rho = \|q_{j+1}\|_2$

$q_{j+1} = q_{j+1} - Q_j c$

$\beta_j = \sqrt{\rho^2 - \sum_{i=1}^{j} c_j^2}$

end

$q_{j+1} = q_{j+1}/\beta_j$

If $j < k$

$p_{j+1} = A q_{j+1} - \beta_j p_j$

end

end

Evaluation

Numerical Robustness Evaluation

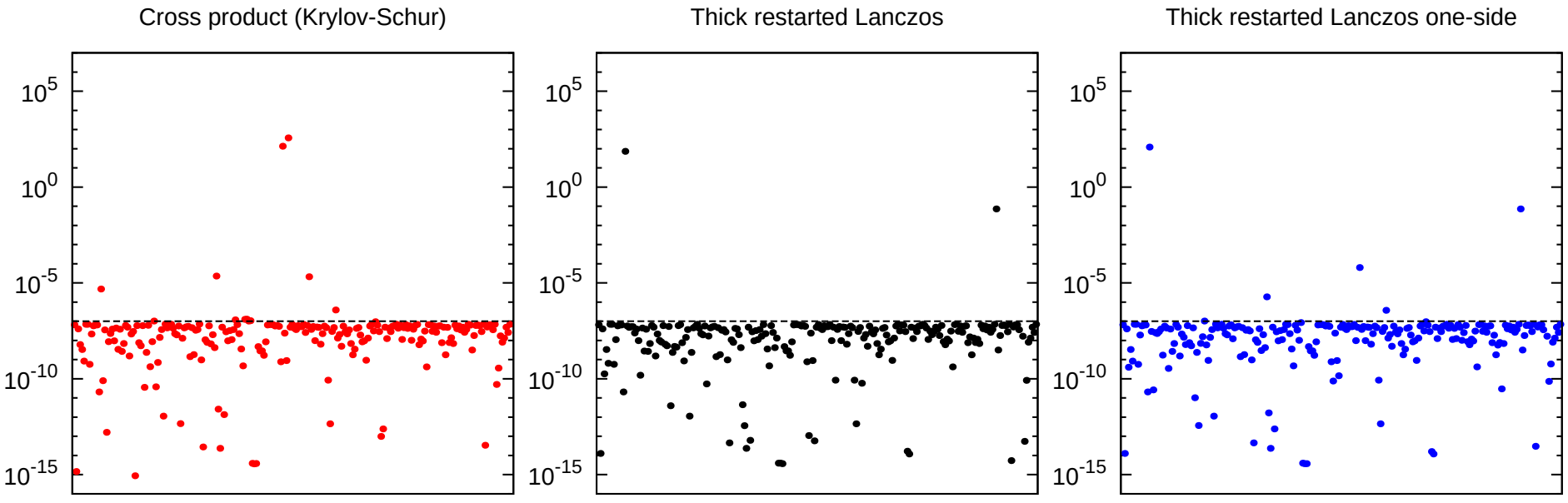
Compute the 10 largest singular values of 232 matrices available at MatrixMarket site

Solver settings:

- restarting with a maximum of 30 basis vectors
- stopping criteria with a tolerance of 10^{-7}

Maximum Relative Residual

$$\xi = \frac{\sqrt{\|Av - \sigma u\|_2^2 + \|A^T u - \sigma v\|_2^2}}{\sigma\sqrt{\|u\|_2^2 + \|v\|_2^2}}$$



Parallel Performance

Compute the 10 largest singular values of two matrices, restarting with a maximum of 30 basis vectors

Speed-up

Calculated as the ratio of elapsed time with p processors to elapsed time of the fastest algorithm with one processor

$$S_p = \frac{T_p}{T_1}$$

Computer used

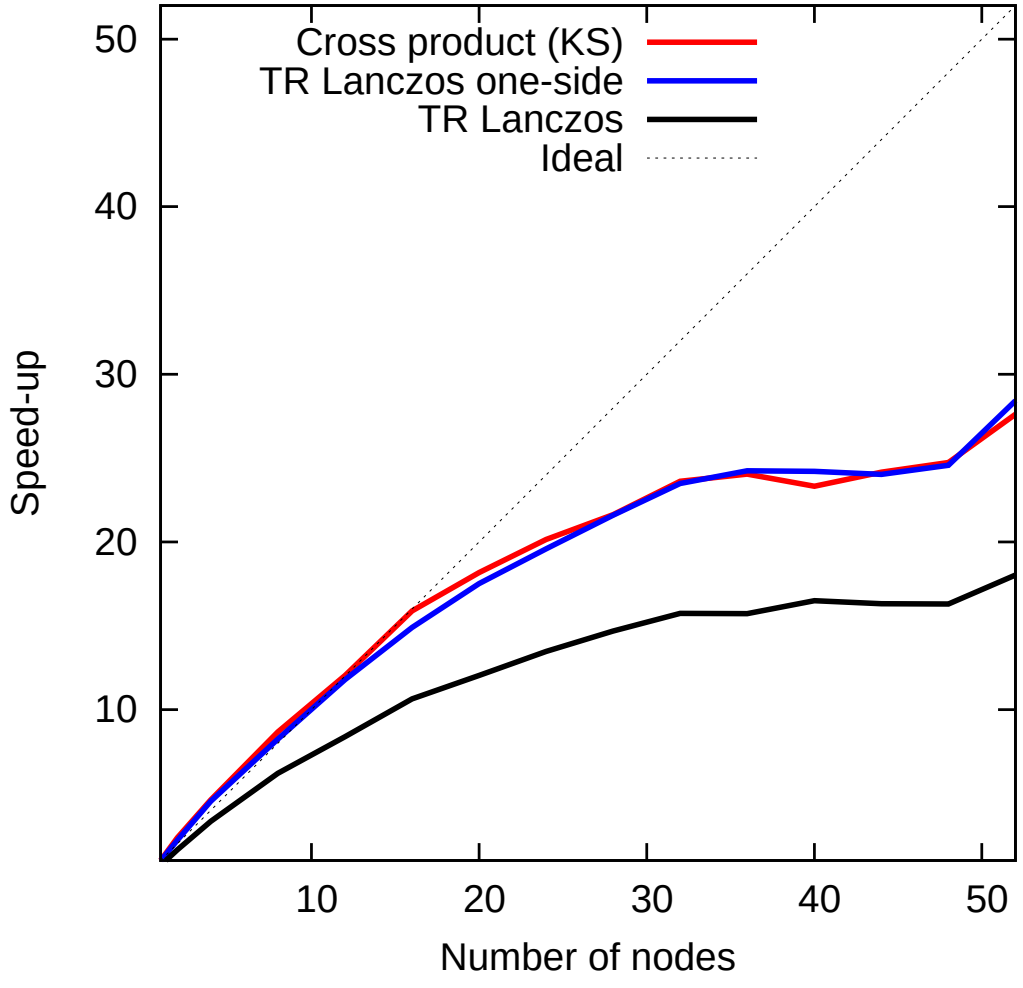
- Cluster of 55 Pentium Xeon biprocessors with SCI interconnect

Only one processor per node was used

Performance in Xeon cluster (1)

AF23560 matrix	
Order	23,560
Non-zeros	484,256
Sparsity	0.0087 %

Largest matrix from Matrix Market NEP collection

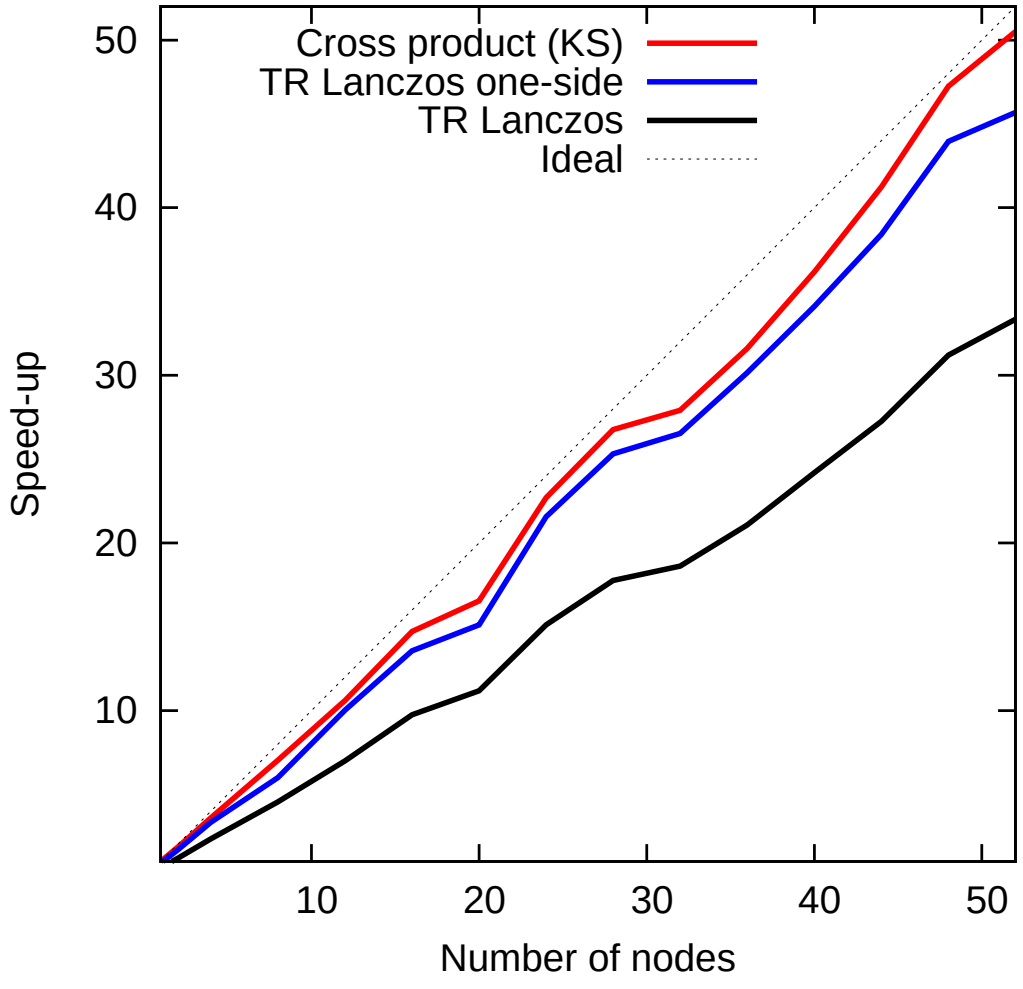


Performance in Xeon cluster (2)

PRE2 matrix

Order	659,033
Non-zeros	5,834,044
Sparsity	0.0013 %

Non-symmetric matrix from University of Florida sparse matrix collection



Conclusion

Two specific SVD solvers in SLEPc

- `lanczos`: explicit-restart Lanczos bidiagonalization
- `trlanczos`: thick-restart Lanczos bidiagonalization

One-sided orthogonalization available in both cases

Conclusion

Two specific SVD solvers in SLEPc

- `lanczos`: explicit-restart Lanczos bidiagonalization
- `trlanczos`: thick-restart Lanczos bidiagonalization

One-sided orthogonalization available in both cases

Performance:

- As efficient as Krylov-Schur on cross-product matrix A^*A
- Slightly more robust numerically
- Presumably more accurate in small singular values

However, small singular values are difficult to converge: need to implement harmonic or refined projection

References

- G. H. Golub and W. Kahan (1965).
  Calculating the Singular Values and Pseudo-Inverse of a Matrix.
  J. Soc. Indust. Appl. Math. Ser. B Numer. Anal., 2:205–224.
- J. Baglama and L. Reichel (2005).
  Augmented Implicitly Restarted Lanczos Bidiagonalization Methods.
  SIAM J. Sci. Comput., 27(1):19–42.
- H. D. Simon and H. Zha (2000).
  Low-Rank Matrix Approximation Using the Lanczos Bidiagonalization Process with Applications.
  SIAM J. Sci. Comput., 21(6):2257–2274.
- V. Hernandez, J. E. Roman, and A. Tomas (2007).
  Parallel Arnoldi Eigensolvers with Enhanced Scalability via Global ...
  Parallel Comput., 33(7–8):521–540.

Thanks!

http://www.grycap.upv.es/slepc

slepc-maint@grycap.upv.es