ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

Parallel Public-domain Numerical Libraries

CSCS-USI Summer School, 19.07.2013

Dr. William Sawyer

Overview

Overview

- Introduction and library overview

Overview

- Introduction and library overview
  - Typical numerical and semi-numerical problems

Overview

- Introduction and library overview
  - Typical numerical and semi-numerical problems
  - Survey of parallel libraries

Overview

- Introduction and library overview
  - Typical numerical and semi-numerical problems
  - Survey of parallel libraries
- Case study 1: linear algebra

Overview

- Introduction and library overview
  - Typical numerical and semi-numerical problems
  - Survey of parallel libraries
- Case study 1: linear algebra
- Case study 2: PDE solver

But wait: first my own homework: What is a Cray “Dragonfly” network interconnect?

But wait: first my own homework: What is a Cray “Dragonfly” network interconnect?

Fully connected network topology:

But wait: first my own homework: What is a Cray “Dragonfly” network interconnect?

Fully connected network topology:

Torus network topology:

But wait: first my own homework: What is a Cray “Dragonfly” network interconnect?

Fully connected network topology:

Torus network topology:

Fat-tree network topology:

But wait: first my own homework: What is a Cray “Dragonfly” network interconnect?

Fully connected network topology:

Torus network topology:

Fat-tree network topology:

Some Interconnect considerations

Some Interconnect considerations

- Fully-connected networks are optimal, but not scalable: number of nodes must be small

Some Interconnect considerations

- Fully-connected networks are optimal, but not scalable: number of nodes must be small
- Torus is economical, but leads to network contention

Some Interconnect considerations

- Fully-connected networks are optimal, but not scalable: number of nodes must be small
- Torus is economical, but leads to network contention
- Fat tree is gives good bandwidth, but is not modular -- difficult to extend the system

Some Interconnect considerations

- Fully-connected networks are optimal, but not scalable: number of nodes must be small
- Torus is economical, but leads to network contention
- Fat tree is gives good bandwidth, but is not modular -- difficult to extend the system

A company like Cray needs to support customers with big and small systems.

Some Interconnect considerations

- Fully-connected networks are optimal, but not scalable: number of nodes must be small
- Torus is economical, but leads to network contention
- Fat tree is gives good bandwidth, but is not modular -- difficult to extend the system

A company like Cray needs to support customers with big and small systems.

Big systems are built modularly from small ones

Some Interconnect considerations

- Fully-connected networks are optimal, but not scalable: number of nodes must be small
- Torus is economical, but leads to network contention
- Fat tree is gives good bandwidth, but is not modular -- difficult to extend the system

A company like Cray needs to support customers with big and small systems.

Big systems are built modularly from small ones

Technology should be affordable: electrical connections for short distances (< 3m)

Cray Dragonfly

Cray Dragonfly

- Use fat tree within one “group” (cabinet)

Cray Dragonfly

- Use fat tree within one “group” (cabinet)
- Electrical connections within fat tree

Cray Dragonfly

- Use fat tree within one “group” (cabinet)
- Electrical connections within fat tree

Cray Dragonfly

- Use fat tree within one “group” (cabinet)
- Electrical connections within fat tree
- Fully connected optical network between groups

Cray Dragonfly

- Use fat tree within one “group” (cabinet)
- Electrical connections within fat tree
- Fully connected optical network between groups

Poll

Poll

- What programming languages do you work with?

Poll

- What programming languages do you work with?
- What parallel programming paradigms?

Poll

- What programming languages do you work with?
- What parallel programming paradigms?
- What problems do you want to solve ?

Poll

- What programming languages do you work with?
- What parallel programming paradigms?
- What problems do you want to solve ?
- What algorithms do you want to use?

Poll

- What programming languages do you work with?
- What parallel programming paradigms?
- What problems do you want to solve ?
- What algorithms do you want to use?
- What libraries, if any, do you use?

Problems you might like to solve

Problems you might like to solve

- Partial differential equations

Problems you might like to solve

- Partial differential equations
- Dense systems of linear equations

Problems you might like to solve

- Partial differential equations
- Dense systems of linear equations
- Sparse systems of linear equations

Problems you might like to solve

- Partial differential equations
- Dense systems of linear equations
- Sparse systems of linear equations
- Preconditioning of large systems

Problems you might like to solve

- Partial differential equations
- Dense systems of linear equations
- Sparse systems of linear equations
- Preconditioning of large systems
- Eigenvalue / singular value decompositions of sparse/dense matrices

Problems you might like to solve

- Partial differential equations
- Dense systems of linear equations
- Sparse systems of linear equations
- Preconditioning of large systems
- Eigenvalue / singular value decompositions of sparse/dense matrices
- Partitioning large graphs

Problems you might like to solve

- Partial differential equations
- Dense systems of linear equations
- Sparse systems of linear equations
- Preconditioning of large systems
- Eigenvalue / singular value decompositions of sparse/dense matrices
- Partitioning large graphs
- Non-linear systems and optimization

Problems you might like to solve

- Partial differential equations
- Dense systems of linear equations
- Sparse systems of linear equations
- Preconditioning of large systems
- Eigenvalue / singular value decompositions of sparse/dense matrices
- Partitioning large graphs
- Non-linear systems and optimization
- ...

Assumptions and prerequisites

Assumptions and prerequisites

- Basic background in numerical analysis

Assumptions and prerequisites

- Basic background in numerical analysis
- C/C++ programming experience

Assumptions and prerequisites

- Basic background in numerical analysis
- C/C++ programming experience
- Some parallel programming experience

Assumptions and prerequisites

- Basic background in numerical analysis
- C/C++ programming experience
- Some parallel programming experience
- Basic UNIX user background

Objectives of this tutorial

Objectives of this tutorial

- Awareness of the available libraries

Objectives of this tutorial

- Awareness of the available libraries
- Realization that it is not necessary to "recreate the wheel"

Objectives of this tutorial

- Awareness of the available libraries
- Realization that it is not necessary to "recreate the wheel"
- Bonus: some hands-on experiences

Parallel libraries for dense linear algebra

Parallel libraries for dense linear algebra

- *ScaLAPACK:* Fortran Scalable Linear Algebra PACKage (LAPACK) http://www.netlib.org/scalapack

Parallel libraries for dense linear algebra

- *ScaLAPACK:* Fortran Scalable Linear Algebra PACKage (LAPACK) http://www.netlib.org/scalapack
- *PLAPACK:* object-oriented Parallel Linear Algebra PACKage http://userweb.cs.utexas.edu/users/plapack

Parallel libraries for dense linear algebra

- *ScaLAPACK:* Fortran Scalable Linear Algebra PACKage (LAPACK) http://www.netlib.org/scalapack
- *PLAPACK:* object-oriented Parallel Linear Algebra PACKage http://userweb.cs.utexas.edu/users/plapack
- *PLASMA:* Parallel Linear Algebra for Scalable Multi-core Architectures http://icl.cs.utk.edu/plasma

Parallel libraries for dense linear algebra

- *ScaLAPACK:* Fortran Scalable Linear Algebra PACKage (LAPACK) http://www.netlib.org/scalapack
- *PLAPACK:* object-oriented Parallel Linear Algebra PACKage http://userweb.cs.utexas.edu/users/plapack
- *PLASMA:* Parallel Linear Algebra for Scalable Multi-core Architectures http://icl.cs.utk.edu/plasma
- *MAGMA:* Matrix Algebra on GPU and Multicore Architectures http://icl.cs.utk.edu/magma

Parallel libraries for dense linear algebra

- *ScaLAPACK:* Fortran Scalable Linear Algebra PACKage (LAPACK) http://www.netlib.org/scalapack
- *PLAPACK:* object-oriented Parallel Linear Algebra PACKage http://userweb.cs.utexas.edu/users/plapack
- *PLASMA:* Parallel Linear Algebra for Scalable Multi-core Architectures http://icl.cs.utk.edu/plasma
- *MAGMA:* Matrix Algebra on GPU and Multicore Architectures http://icl.cs.utk.edu/magma
- Soon: *D-PLASMA*, *D-MAGMA* for distributed memory platforms

Typical dense linear algebra operations

Typical dense linear algebra operations

- Cholesky factorization:

Typical dense linear algebra operations

- Cholesky factorization: $A = A^T = LL^T$ $\quad i < j \Rightarrow L_{i,j} = 0$
- QR factorization:

Typical dense linear algebra operations

- Cholesky factorization: $A = A^T = LL^T \qquad i < j \Rightarrow L_{i,j} = 0$
- QR factorization: $A = QR \qquad Q^T Q = I \qquad i > j \Rightarrow R_{i,j} = 0$
- LU factorization:

Typical dense linear algebra operations

- Cholesky factorization: $A = A^T = LL^T \qquad i < j \Rightarrow L_{i,j} = 0$
- QR factorization: $A = QR \qquad Q^T Q = I \qquad i > j \Rightarrow R_{i,j} = 0$
- LU factorization: $A = P^T LU \qquad P^T P = I$
- Forward/back-substitution:

Typical dense linear algebra operations

- Cholesky factorization: $A = A^T = LL^T \qquad i < j \Rightarrow L_{i,j} = 0$
- QR factorization: $A = QR \qquad Q^T Q = I \qquad i > j \Rightarrow R_{i,j} = 0$
- LU factorization: $A = P^T LU \qquad P^T P = I$
- Forward/back-substitution: $Ax = y \Rightarrow LUx = y \Rightarrow w = L^{-1}y \Rightarrow x = R^{-1}w$
- Eigenvalue decomposition:

Typical dense linear algebra operations

- Cholesky factorization: $A = A^T = LL^T \qquad i < j \Rightarrow L_{i,j} = 0$
- QR factorization: $A = QR \qquad Q^T Q = I \qquad i > j \Rightarrow R_{i,j} = 0$
- LU factorization: $A = P^T LU \qquad P^T P = I$
- Forward/back-substitution: $Ax = y \Rightarrow LUx = y \Rightarrow w = L^{-1}y \Rightarrow x = R^{-1}w$
- Eigenvalue decomposition: $Ax = \lambda x \Rightarrow A = QDQ^T$
- Generalized eigen-problem:

Typical dense linear algebra operations

- Cholesky factorization: $A = A^T = LL^T \qquad i < j \Rightarrow L_{i,j} = 0$
- QR factorization: $A = QR \qquad Q^T Q = I \qquad i > j \Rightarrow R_{i,j} = 0$
- LU factorization: $A = P^T LU \qquad P^T P = I$
- Forward/back-substitution: $Ax = y \Rightarrow LUx = y \Rightarrow w = L^{-1}y \Rightarrow x = R^{-1}w$
- Eigenvalue decomposition: $Ax = \lambda x \Rightarrow A = QDQ^T$
- Generalized eigen-problem: $Ax = \lambda Bx$
- Singular value decomposition:

Typical dense linear algebra operations

- Cholesky factorization: $A = A^T = LL^T \qquad i < j \Rightarrow L_{i,j} = 0$
- QR factorization: $A = QR \qquad Q^T Q = I \qquad i > j \Rightarrow R_{i,j} = 0$
- LU factorization: $A = P^T LU \qquad P^T P = I$
- Forward/back-substitution: $Ax = y \Rightarrow LUx = y \Rightarrow w = L^{-1}y \Rightarrow x = R^{-1}w$
- Eigenvalue decomposition: $Ax = \lambda x \Rightarrow A = QDQ^T$
- Generalized eigen-problem: $Ax = \lambda Bx$
- Singular value decomposition: $A = U\Sigma V^T \quad U^T U = I \quad V^T V = I$

ScaLAPACK: dense linear algebra

ScaLAPACK: dense linear algebra

- Scalable Linear Algebra PACKage

ScaLAPACK: dense linear algebra

- Scalable Linear Algebra PACKage
- Solves dense linear systems and computes eigen / singular values of dense matrices

ScaLAPACK: dense linear algebra

- Scalable Linear Algebra PACKage
- Solves dense linear systems and computes eigen / singular values of dense matrices
- Developing teams: UT Knoxville, UC Berkeley, ORNL, Rice U.,UCLA, UIUC, etc.

ScaLAPACK: dense linear algebra

- Scalable Linear Algebra PACKage
- Solves dense linear systems and computes eigen / singular values of dense matrices
- Developing teams: UT Knoxville, UC Berkeley, ORNL, Rice U.,UCLA, UIUC, etc.
- Supported in Commercial Packages

ScaLAPACK: dense linear algebra

- Scalable Linear Algebra PACKage
- Solves dense linear systems and computes eigen / singular values of dense matrices
- Developing teams: UT Knoxville, UC Berkeley, ORNL, Rice U.,UCLA, UIUC, etc.
- Supported in Commercial Packages
  - NAG Parallel Library, IBM PESSL, CRAY Scilib

ScaLAPACK: dense linear algebra

- Scalable Linear Algebra PACKage
- Solves dense linear systems and computes eigen / singular values of dense matrices
- Developing teams: UT Knoxville, UC Berkeley, ORNL, Rice U.,UCLA, UIUC, etc.
- Supported in Commercial Packages
  - NAG Parallel Library, IBM PESSL, CRAY Scilib
  - VNI IMSL, Fujitsu, HP/Convex, Hitachi, NEC

PLAPACK vs. ScaLAPACK

PLAPACK vs. ScaLAPACK

PLASMA Performance (Cholesky, QR, LU)

PLASMA Performance (Cholesky, QR, LU)

(a) DPOTRF - Intel64 - 16 cores (b) DGEQRF - Intel64 - 16 cores (c) DGETRF - Intel64 - 16 cores

(d) DPOTRF - Power6 - 32 cores (e) DGEQRF - Power6 - 32 cores (f) DGETRF - Power6 - 32 cores

Figure 1: Performance comparison on a large number of cores (Gflop/s).

MAGMA (GPU) Performance (QR, LU)

MAGMA on GTX280 vs. Xeon quad core Left: QR decomp. SP/DP Right: LU decomp. SP/DP

MAGMA (GPU) Performance (QR, LU)

MAGMA on GTX280 vs. Xeon quad core Left: QR decomp. SP/DP Right: LU decomp. SP/DP

ADVERTISEMENT: *GPU libraries course*

ADVERTISEMENT: *GPU libraries course*

- FoMICS Autumn School: *Sep. 14-15, 2013* (Sat. - Sun.)

ADVERTISEMENT: *GPU libraries course*

- FoMICS Autumn School: *Sep. 14-15, 2013* (Sat. - Sun.)
- Focus: GPU-enabled libraries, such as MAGMA, CUBLAS, CUSparse, CUFFT, Paralution, others

ADVERTISEMENT: *GPU libraries course*

- FoMICS Autumn School: *Sep. 14-15, 2013* (Sat. - Sun.)
- Focus: GPU-enabled libraries, such as MAGMA, CUBLAS, CUSparse, CUFFT, Paralution, others
- Brief introduction to Cuda and OpenCL

ADVERTISEMENT: *GPU libraries course*

- FoMICS Autumn School: *Sep. 14-15, 2013* (Sat. - Sun.)
- Focus: GPU-enabled libraries, such as MAGMA, CUBLAS, CUSparse, CUFFT, Paralution, others
- Brief introduction to Cuda and OpenCL
- Graduate student price: CHF 50

ADVERTISEMENT: *GPU libraries course*

- FoMICS Autumn School: *Sep. 14-15, 2013* (Sat. - Sun.)
- Focus: GPU-enabled libraries, such as MAGMA, CUBLAS, CUSparse, CUFFT, Paralution, others
- Brief introduction to Cuda and OpenCL
- Graduate student price: CHF 50
- http://icsweb.inf.unisi.ch/cms/index.php/component/content/article/12-news/97-autumnschool.html

ScaLAPACK Technical Information

ScaLAPACK Technical Information

- ScaLAPACK web page

ScaLAPACK Technical Information

- ScaLAPACK web page
  - http://www.netlib.org/scalapack

ScaLAPACK Technical Information

- ScaLAPACK web page
  - http://www.netlib.org/scalapack
- ScaLAPACK User's Guide

ScaLAPACK Technical Information

- ScaLAPACK web page
  - http://www.netlib.org/scalapack
- ScaLAPACK User's Guide
- Excellent introductory site

ScaLAPACK Technical Information

- ScaLAPACK web page
  - http://www.netlib.org/scalapack
- ScaLAPACK User's Guide
- Excellent introductory site
  - http://acts.nersc.gov/scalapack

ScaLAPACK Technical Information

- ScaLAPACK web page
  - http://www.netlib.org/scalapack
- ScaLAPACK User's Guide
- Excellent introductory site
  - http://acts.nersc.gov/scalapack
  - http://acts.nersc.gov/scalapack/hands-on

ScaLAPACK Technical Information

- ScaLAPACK web page
  - http://www.netlib.org/scalapack
- ScaLAPACK User's Guide
- Excellent introductory site
  - http://acts.nersc.gov/scalapack
  - http://acts.nersc.gov/scalapack/hands-on
- LAPACK Working Notes

ScaLAPACK Technical Information

- ScaLAPACK web page
  - http://www.netlib.org/scalapack
- ScaLAPACK User's Guide
- Excellent introductory site
  - http://acts.nersc.gov/scalapack
  - http://acts.nersc.gov/scalapack/hands-on
- LAPACK Working Notes
  - http://www.netlib.org/lapack/lawn

ScaLAPACK: *software hierarchy*

ScaLAPACK: *software hierarchy*

Clarity,modularity, performance and portability. Atlas can be used here for automatic tuning.

ScaLAPACK: *software hierarchy*

Linear systems, least squares, singular value decomposition, eigenvalues.

Clarity,modularity, performance and portability. Atlas can be used here for automatic tuning.

ScaLAPACK: *software hierarchy*

ScaLAPACK: *software hierarchy*

ScaLAPACK: *software hierarchy*

ScaLAPACK: 1D block-cyclic distribution

ScaLAPACK: 1D block-cyclic distribution

ScaLAPACK: 2D block-cyclic distribution

ScaLAPACK: 2D block-cyclic distribution

5x5 matrix partitioned in 2x2 blocks

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix}$$

2x2 process grid point of view

ScaLAPACK: 2D block-cyclic distribution

5x5 matrix partitioned in 2x2 blocks

$$\left(\begin{array}{cc|cc|c} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ \hline a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{array}\right)$$

2x2 process grid point of view

ScaLAPACK: 2D block-cyclic distribution

5x5 matrix partitioned in 2x2 blocks

$$\left(\begin{array}{cc|cc|c} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ \hline a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{array}\right)$$

2x2 process grid point of view

http://acts.nersc.gov/scalapack/hands-on/datadist.html

BLACS: communication substructure

BLACS: communication substructure

- Basic Linear Algebra Comm. Subroutines

BLACS: communication substructure

- Basic Linear Algebra Comm. Subroutines
- describes the underlying (virtual) parallel platform

BLACS: communication substructure

- Basic Linear Algebra Comm. Subroutines
- describes the underlying (virtual) parallel platform
- processes embedded in 2D topology

BLACS: communication substructure

- Basic Linear Algebra Comm. Subroutines
- describes the underlying (virtual) parallel platform
- processes embedded in 2D topology
- provides point-to-point, collective communication and support functionality

BLACS: communication substructure

- Basic Linear Algebra Comm. Subroutines
- describes the underlying (virtual) parallel platform
- processes embedded in 2D topology
- provides point-to-point, collective communication and support functionality
- collective communication over a row, column or over all processes in the 2D grid

BLACS: controls a virtual process grid

BLACS: controls a virtual process grid

BLACS: controls a virtual process grid

General information:

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

Get default system context (communicator):

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

Get default system context (communicator):

```
CALL BLACS_GET(0,0,ICTXT)
```

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

Get default system context (communicator):

```
CALL BLACS_GET(0,0,ICTXT)
```

Initialize the grid:

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

Get default system context (communicator):

```
CALL BLACS_GET(0,0,ICTXT)
```

Initialize the grid:

```
NPROW = 3; NPCOL = 4
```

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

Get default system context (communicator):

```
CALL BLACS_GET(0,0,ICTXT)
```

Initialize the grid:

```
NPROW = 3; NPCOL = 4
CALL BLACS_GRIDINIT(ICTXT,"Row",NPROW,NPCOL)
```

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

Get default system context (communicator):

```
CALL BLACS_GET(0,0,ICTXT)
```

Initialize the grid:

```
NPROW = 3; NPCOL = 4
CALL BLACS_GRIDINIT(ICTXT,"Row",NPROW,NPCOL)
CALL BLACS_GRIDGET(ICTXT,NPROW,NPCOL,MYROW,MYCOL)
```

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

Get default system context (communicator):

```
CALL BLACS_GET(0,0,ICTXT)
```

Initialize the grid:

```
NPROW = 3; NPCOL = 4
CALL BLACS_GRIDINIT(ICTXT,"Row",NPROW,NPCOL)
CALL BLACS_GRIDGET(ICTXT,NPROW,NPCOL,MYROW,MYCOL)
```

If in the grid, perform the computation:

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

Get default system context (communicator):

```
CALL BLACS_GET(0,0,ICTXT)
```

Initialize the grid:

```
NPROW = 3; NPCOL = 4
CALL BLACS_GRIDINIT(ICTXT,"Row",NPROW,NPCOL)
CALL BLACS_GRIDGET(ICTXT,NPROW,NPCOL,MYROW,MYCOL)
```

If in the grid, perform the computation:

```
IF (MYROW /= -1) THEN
```

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

Get default system context (communicator):

```
CALL BLACS_GET(0,0,ICTXT)
```

Initialize the grid:

```
NPROW = 3; NPCOL = 4
CALL BLACS_GRIDINIT(ICTXT,"Row",NPROW,NPCOL)
CALL BLACS_GRIDGET(ICTXT,NPROW,NPCOL,MYROW,MYCOL)
```

If in the grid, perform the computation:

```
IF (MYROW /= -1) THEN
   …
```

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

Get default system context (communicator):

```
CALL BLACS_GET(0,0,ICTXT)
```

Initialize the grid:

```
NPROW = 3; NPCOL = 4
CALL BLACS_GRIDINIT(ICTXT,"Row",NPROW,NPCOL)
CALL BLACS_GRIDGET(ICTXT,NPROW,NPCOL,MYROW,MYCOL)
```

If in the grid, perform the computation:

```
IF (MYROW /= -1) THEN
   …
   CALL BLACS_GRIDEXIT(ICTXT)
```

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

Get default system context (communicator):

```
CALL BLACS_GET(0,0,ICTXT)
```

Initialize the grid:

```
NPROW = 3; NPCOL = 4
CALL BLACS_GRIDINIT(ICTXT,"Row",NPROW,NPCOL)
CALL BLACS_GRIDGET(ICTXT,NPROW,NPCOL,MYROW,MYCOL)
```

If in the grid, perform the computation:

```
IF (MYROW /= -1) THEN
   …
   CALL BLACS_GRIDEXIT(ICTXT)
ENDIF
```

BLACS: controls a virtual process grid

General information:

```
CALL BLACS_PINFO(iam,nprocs)
```

Get default system context (communicator):

```
CALL BLACS_GET(0,0,ICTXT)
```

Initialize the grid:

```
NPROW = 3; NPCOL = 4
CALL BLACS_GRIDINIT(ICTXT,"Row",NPROW,NPCOL)
CALL BLACS_GRIDGET(ICTXT,NPROW,NPCOL,MYROW,MYCOL)
```

If in the grid, perform the computation:

```
IF (MYROW /= -1) THEN
   …
   CALL BLACS_GRIDEXIT(ICTXT)
ENDIF
CALL BLACS_EXIT(0)
```

BLACS: communication routines

BLACS: communication routines

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix, TR: Trapezoidal matrix.

BLACS: communication routines

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix, TR: Trapezoidal matrix.

SCOPE	TOP
'Row' 'Column' 'All'	' ' (default) 'Increasing Ring' '1-tree' ...

BLACS: communication routines

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix, TR: Trapezoidal matrix.

SCOPE	TOP
'Row' 'Column' 'All'	' ' (default) 'Increasing Ring' '1-tree' ...

Send/receive submatrix from one process to another:

BLACS: communication routines

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix, TR: Trapezoidal matrix.

SCOPE	TOP
'Row' 'Column' 'All'	' ' (default) 'Increasing Ring' '1-tree' ...

Send/receive submatrix from one process to another:

```
_xxSD2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RDEST,CDEST)
```

BLACS: communication routines

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix, TR: Trapezoidal matrix.

SCOPE	TOP
'Row' 'Column' 'All'	' ' (default) 'Increasing Ring' '1-tree' ...

Send/receive submatrix from one process to another:

```
_xxSD2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RDEST,CDEST)
_xxRV2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

BLACS: communication routines

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix, TR: Trapezoidal matrix.

SCOPE	TOP
'Row' 'Column' 'All'	' ' (default) 'Increasing Ring' '1-tree' ...

Send/receive submatrix from one process to another:

```
_xxSD2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RDEST,CDEST)
_xxRV2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

Broadcast (and explicitly receive) submatrix:

BLACS: communication routines

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix, TR: Trapezoidal matrix.

SCOPE	TOP
'Row' 'Column' 'All'	' ' (default) 'Increasing Ring' '1-tree' ...

Send/receive submatrix from one process to another:

```
_xxSD2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RDEST,CDEST)
_xxRV2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

Broadcast (and explicitly receive) submatrix:

```
_xxBS2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA)
```

BLACS: communication routines

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix, TR: Trapezoidal matrix.

SCOPE	TOP
'Row' 'Column' 'All'	' ' (default) 'Increasing Ring' '1-tree' ...

Send/receive submatrix from one process to another:

```
_xxSD2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RDEST,CDEST)
_xxRV2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

Broadcast (and explicitly receive) submatrix:

```
_xxBS2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA)
_xxBR2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

BLACS: communication routines

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix, TR: Trapezoidal matrix.

SCOPE	TOP
'Row' 'Column' 'All'	' ' (default) 'Increasing Ring' '1-tree' ...

Send/receive submatrix from one process to another:

```
_xxSD2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RDEST,CDEST)
_xxRV2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

Broadcast (and explicitly receive) submatrix:

```
_xxBS2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA)
_xxBR2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

Global combine operations, element-wise sum, |max|, |min|:

BLACS: communication routines

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix, TR: Trapezoidal matrix.

SCOPE	TOP
'Row' 'Column' 'All'	' ' (default) 'Increasing Ring' '1-tree' ...

Send/receive submatrix from one process to another:

```
_xxSD2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RDEST,CDEST)
_xxRV2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

Broadcast (and explicitly receive) submatrix:

```
_xxBS2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA)
_xxBR2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

Global combine operations, element-wise sum, |max|, |min|:

```
_GSUM2D(ICTXT,SCOPE,TOP,M,N,A,LDA,RDEST,CDEST)
```

BLACS: communication routines

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix, TR: Trapezoidal matrix.

SCOPE	TOP
'Row' 'Column' 'All'	' ' (default) 'Increasing Ring' '1-tree' ...

Send/receive submatrix from one process to another:

```
_xxSD2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RDEST,CDEST)
_xxRV2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

Broadcast (and explicitly receive) submatrix:

```
_xxBS2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA)
_xxBR2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

Global combine operations, element-wise sum, |max|, |min|:

```
_GSUM2D(ICTXT,SCOPE,TOP,M,N,A,LDA,RDEST,CDEST)
_GAMX2D(ICTXT,SCOPE,TOP,M,N,A,LDA,RA,CA,RCFLAG,RDEST,CDEST)
```

BLACS: communication routines

_ (Data type)	xx (Matrix type)
I: Integer, S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: General rectangular matrix, TR: Trapezoidal matrix.

SCOPE	TOP
'Row' 'Column' 'All'	' ' (default) 'Increasing Ring' '1-tree' ...

Send/receive submatrix from one process to another:

```
_xxSD2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RDEST,CDEST)
_xxRV2D(ICTXT,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

Broadcast (and explicitly receive) submatrix:

```
_xxBS2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA)
_xxBR2D(ICTXT,SCOPE,TOP,[UPLO,DIAG],M,N,A,LDA,RSRC,CSRC)
```

Global combine operations, element-wise sum, |max|, |min|:

```
_GSUM2D(ICTXT,SCOPE,TOP,M,N,A,LDA,RDEST,CDEST)
_GAMX2D(ICTXT,SCOPE,TOP,M,N,A,LDA,RA,CA,RCFLAG,RDEST,CDEST)
_GAMN2D(ICTXT,SCOPE,TOP,M,N,A,LDA,RA,CA,RCFLAG,RDEST,CDEST)
```

PBLAS: parallel basic LA subroutines

PBLAS: parallel basic LA subroutines

- Level 1 PBLAS Vector-Vector operations
- Level 2 PBLAS Matrix-Vector operations
- Level 3 PBLAS Matrix-Matrix operations

PBLAS: operations

PBLAS: operations

_ (Data type)	xx (Matrix type)
S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: All matrix operands are General rectangular, HE: One of the matrix operands is HErmitian, SY: One of the matrix operands is SYmmetric, TR: One of the matrix operands is TRiangular.

PBLAS: operations

_ (Data type)	xx (Matrix type)
S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: All matrix operands are General rectangular, HE: One of the matrix operands is HErmitian, SY: One of the matrix operands is SYmmetric, TR: One of the matrix operands is TRiangular.

YYY	(Operation)
MM	Matrix-matrix product;
MV	Matrix-vector product;
R	Rank-1 update of a matrix;
R2	Rank-2 update of a matrix;
RK	Rank-k update of a symmetric /Hermitian matrix;
R2K	Rank-2k update of a symmetric /Hermitian matrix;
SM	Solves a system of linear eqns. for a matrix of rhs;
SV	Solves a system of linear eqns. for a rhs vector.

PBLAS: operations

_ (Data type)	xx (Matrix type)
S: Real, D: Double Precision, C: Complex, Z: Double Complex.	GE: All matrix operands are General rectangular, HE: One of the matrix operands is HErmitian, SY: One of the matrix operands is SYmmetric, TR: One of the matrix operands is TRiangular.

YYY	(Operation)
MM	Matrix-matrix product;
MV	Matrix-vector product;
R	Rank-1 update of a matrix;
R2	Rank-2 update of a matrix;
RK	Rank-k update of a symmetric /Hermitian matrix;
R2K	Rank-2k update of a symmetric /Hermitian matrix;
SM	Solves a system of linear eqns. for a matrix of rhs;
SV	Solves a system of linear eqns. for a rhs vector.

- Naming scheme follows BLAS conventions: `P_XXYYY`
- No vector rotations, banded matrix routines
- Matrix transposition: `P_TRANx`
- Level 1 routines have meaningful names: `PSCOPY, PSDOT, ..`

PBLAS: global view of data

PBLAS: global view of data

PBLAS provides global view of matrix, allowing global addressing

BLAS: `CALL DGEXX(M, N, A(IA,JA), LDA,…)`

PBLAS: `CALL PDGEXX(M, N, A, IA, JA, DESCA, …)`

PBLAS: global view of data

PBLAS provides global view of matrix, allowing global addressing

BLAS: `CALL DGEXX(M, N, A(IA,JA), LDA,…)`

PBLAS: `CALL PDGEXX(M, N, A, IA, JA, DESCA, …)`

PBLAS: storage descriptors

PBLAS: storage descriptors

- An M x N matrix is block-partitioned into MB x NB blocks and distributed with 2D block-cyclic scheme

PBLAS: storage descriptors

- An M x N matrix is block-partitioned into MB x NB blocks and distributed with 2D block-cyclic scheme
- Locally, the scattered columns stored contiguously (Fortran “Column Major” ordering)

PBLAS: storage descriptors

- An M x N matrix is block-partitioned into MB x NB blocks and distributed with 2D block-cyclic scheme
- Locally, the scattered columns stored contiguously (Fortran “Column Major” ordering)
- Descriptor DESC_: 9-integer array describing the matrix layout

PBLAS: storage descriptors

- An M x N matrix is block-partitioned into MB x NB blocks and distributed with 2D block-cyclic scheme
- Locally, the scattered columns stored contiguously (Fortran “Column Major” ordering)
- Descriptor DESC_: 9-integer array describing the matrix layout
  - Type and context: `DTYPE_, CTXT_`

PBLAS: storage descriptors

- An M x N matrix is block-partitioned into MB x NB blocks and distributed with 2D block-cyclic scheme
- Locally, the scattered columns stored contiguously (Fortran “Column Major” ordering)
- Descriptor DESC_: 9-integer array describing the matrix layout
  - Type and context: `DTYPE_, CTXT_`
  - Matrix dimensions: `M_, N_, MB_, NB_`

PBLAS: storage descriptors

- An M x N matrix is block-partitioned into MB x NB blocks and distributed with 2D block-cyclic scheme
- Locally, the scattered columns stored contiguously (Fortran “Column Major” ordering)
- Descriptor DESC_: 9-integer array describing the matrix layout
  - Type and context: `DTYPE_, CTXT_`
  - Matrix dimensions: `M_, N_, MB_, NB_`
  - Coordinates owning first matrix entry: `RSRC_, CSRC_`

PBLAS: storage descriptors

- An M x N matrix is block-partitioned into MB x NB blocks and distributed with 2D block-cyclic scheme
- Locally, the scattered columns stored contiguously (Fortran “Column Major” ordering)
- Descriptor DESC_: 9-integer array describing the matrix layout
  - Type and context: `DTYPE_, CTXT_`
  - Matrix dimensions: `M_, N_, MB_, NB_`
  - Coordinates owning first matrix entry: `RSRC_, CSRC_`
  - Leading local dimension: `LLD_`

ScaLAPACK: Interfaces

ScaLAPACK: Interfaces

ScaLAPACK: Interfaces

Data Type	real	double	cmplx	dble cmplx
X	S	D	C	Z

ScaLAPACK: Interfaces

PXYYZZZ

- Computation Performed
- Matrix Type
- Data Types

Data Type	real	double	cmplx	dble cmplx
X	S	D	C	Z

DB General Band (diagonally dominant-like)
DT general Tridiagonal (Diagonally dominant-like)
GB General Band
GE GEneral matrices (e.g., unsymmetric, rectangular, etc.)
GG General matrices, Generalized problem
HE Complex Hermitian
OR Orthogonal Real
PB Positive definite Banded (symmetric or Hermitian)
PO Positive definite (symmetric or Hermitian)
PT Positive definite Tridiagonal (symmetric or Hermitian)
ST Symmetric Tridiagonal Real
SY SYmmetric
TR TRiangular (possibly quasi-triangular)
TZ TrapeZoidal
UN UNitary complex

ScaLAPACK: Interfaces

PXYYZZZ

- Computation Performed
- Matrix Type
- Data Types

Data Type	real	double	cmplx	dble cmplx
X	S	D	C	Z

DB General Band (diagonally dominant-like)
DT general Tridiagonal (Diagonally dominant-like)
GB General Band
GE GEneral matrices (e.g., unsymmetric, rectangular, etc.)
GG General matrices, Generalized problem
HE Complex Hermitian
OR Orthogonal Real
PB Positive definite Banded (symmetric or Hermitian)
PO Positive definite (symmetric or Hermitian)
PT Positive definite Tridiagonal (symmetric or Hermitian)
ST Symmetric Tridiagonal Real
SY SYmmetric
TR TRiangular (possibly quasi-triangular)
TZ TrapeZoidal
UN UNitary complex

SL Linear Equations
SV LU Solver
SVD Singular Value
EV Eigenvalue

Example: Singular Value Decomposition

Example: Singular Value Decomposition

```
      SUBROUTINE PSGESVD(JOBU,JOBVT,M,N,A,IA,JA,DESCA,S,U,IU,JU,DESCU,
     +                   VT,IVT,JVT,DESCVT,WORK,LWORK,INFO)

* .. Scalar Arguments ..
      CHARACTER JOBU,JOBVT
      INTEGER IA,INFO,IU,IVT,JA,JU,JVT,LWORK,M,N

* .. Array Arguments ..
      INTEGER DESCA(*),DESCU(*),DESCVT(*)
      REAL A(*),U(*),VT(*),WORK(*)
```

Example: Singular Value Decomposition

```
      SUBROUTINE PSGESVD(JOBU,JOBVT,M,N,A,IA,JA,DESCA,S,U,IU,JU,DESCU,
     +                   VT,IVT,JVT,DESCVT,WORK,LWORK,INFO)

* .. Scalar Arguments ..
      CHARACTER JOBU,JOBVT
      INTEGER IA,INFO,IU,IVT,JA,JU,JVT,LWORK,M,N

* .. Array Arguments ..
      INTEGER DESCA(*),DESCU(*),DESCVT(*)
      REAL A(*),U(*),VT(*),WORK(*)
      REAL S(*)
```

Case Study 1: tridiagonal solve

Case Study 1: tridiagonal solve

Objectives:

Case Study 1: tridiagonal solve

Objectives:

- Get acquainted with ScaLAPACK with simple code

Case Study 1: tridiagonal solve

Objectives:

- Get acquainted with ScaLAPACK with simple code
- Solve tridiagonal system of equations

Case Study 1: tridiagonal solve

Objectives:

- Get acquainted with ScaLAPACK with simple code
- Solve tridiagonal system of equations $\mathrm{Ax} = \mathrm{b}$

Case Study 1: tridiagonal solve

Objectives:

- Get acquainted with ScaLAPACK with simple code
- Solve tridiagonal system of equations $\mathrm{Ax} = \mathrm{b}$

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \vdots \\ \vdots & 0 & a_{43} & a_{44} & a_{44} & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\ 0 & \vdots & \vdots & a_{n-1} & a_{n-1} & a_{n-1,n} \\ 0 & \cdots & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}$$

Case Study 1: tridiagonal solve

Objectives:

- Get acquainted with ScaLAPACK with simple code
- Solve tridiagonal system of equations $\mathrm{Ax} = \mathrm{b}$

Tasks:

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \vdots \\ \vdots & 0 & a_{43} & a_{44} & a_{44} & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\ 0 & \vdots & \vdots & a_{n-1} & a_{n-1} & a_{n-1,n} \\ 0 & \cdots & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}$$

Case Study 1: tridiagonal solve

Objectives:

- Get acquainted with ScaLAPACK with simple code
- Solve tridiagonal system of equations $\mathrm{Ax} = \mathrm{b}$

Tasks:

  – get default system context

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \vdots \\ \vdots & 0 & a_{43} & a_{44} & a_{44} & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\ 0 & \vdots & \vdots & a_{n-1} & a_{n-1} & a_{n-1,n} \\ 0 & \cdots & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}$$

Case Study 1: tridiagonal solve

Objectives:

- Get acquainted with ScaLAPACK with simple code
- Solve tridiagonal system of equations $\mathrm{Ax} = \mathrm{b}$

Tasks:

- get default system context
- initialize process grid with BLACS

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \vdots \\ \vdots & 0 & a_{43} & a_{44} & a_{44} & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\ 0 & \vdots & \vdots & a_{n-1} & a_{n-1} & a_{n-1,n} \\ 0 & \cdots & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}$$

Case Study 1: tridiagonal solve

Objectives:

- Get acquainted with ScaLAPACK with simple code
- Solve tridiagonal system of equations $\mathrm{Ax} = \mathrm{b}$

Tasks:

- get default system context
- initialize process grid with BLACS
- solve system (two routines)

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \vdots \\ \vdots & 0 & a_{43} & a_{44} & a_{44} & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\ 0 & \vdots & \vdots & a_{n-1} & a_{n-1} & a_{n-1,n} \\ 0 & \cdots & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}$$

Case Study 1: tridiagonal solve

Objectives:

- Get acquainted with ScaLAPACK with simple code
- Solve tridiagonal system of equations $\mathrm{Ax} = \mathrm{b}$

Tasks:

- get default system context
- initialize process grid with BLACS
- solve system (two routines)
- exit process grid

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \vdots \\ \vdots & 0 & a_{43} & a_{44} & a_{44} & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\ 0 & \vdots & \vdots & a_{n-1} & a_{n-1} & a_{n-1,n} \\ 0 & \cdots & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}$$

Tridiagonal solve: assignment

Tridiagonal solve: assignment

- Code:

Tridiagonal solve: assignment

- Code:
  - `git clone https://github.com/fomics/SummerSchool2013.git`

Tridiagonal solve: assignment

- Code:
  - `git clone https://github.com/fomics/SummerSchool2013.git`
  - `cd Libraries/ScaLAPACK/TridiagC`

Tridiagonal solve: assignment

- Code:
  - `git clone https://github.com/fomics/SummerSchool2013.git`
  - `cd Libraries/ScaLAPACK/TridiagC`
  - README contains instructions; read this first

Tridiagonal solve: assignment

- Code:
  - `git clone` https://github.com/fomics/SummerSchool2013.git
  - `cd Libraries/ScaLAPACK/TridiagC`
  - README contains instructions; read this first
  - standalone main program `pddttrdrv.c`

Tridiagonal solve: assignment

- Code:
  - `git clone` https://github.com/fomics/SummerSchool2013.git
  - `cd Libraries/ScaLAPACK/TridiagC`
  - README contains instructions; read this first
  - standalone main program `pddttrdrv.c`
  - Makefile -- This time no Cmake or Scons!

Tridiagonal solve: assignment

- Code:
  - `git clone` https://github.com/fomics/SummerSchool2013.git
  - `cd Libraries/ScaLAPACK/TridiagC`
  - README contains instructions; read this first
  - standalone main program `pddttrdrv.c`
  - Makefile -- This time no Cmake or Scons!
- Read through program; code will not compile!

Tridiagonal solve: assignment

- Code:
  - `git clone` https://github.com/fomics/SummerSchool2013.git
  - `cd Libraries/ScaLAPACK/TridiagC`
  - README contains instructions; read this first
  - standalone main program `pddttrdrv.c`
  - Makefile -- This time no Cmake or Scons!
- Read through program; code will not compile!
- Modify code, replacing "***" as needed.

Tridiagonal solve: assignment

- Code:
  - `git clone` https://github.com/fomics/SummerSchool2013.git
  - `cd Libraries/ScaLAPACK/TridiagC`
  - README contains instructions; read this first
  - standalone main program `pddttrdrv.c`
  - Makefile -- This time no Cmake or Scons!
- Read through program; code will not compile!
- Modify code, replacing "***" as needed.
- Code hardwired to run on 2 MPI processes

Tridiagonal solve: assignment

- Code:
  - `git clone` https://github.com/fomics/SummerSchool2013.git
  - `cd Libraries/ScaLAPACK/TridiagC`
  - README contains instructions; read this first
  - standalone main program `pddttrdrv.c`
  - Makefile -- This time no Cmake or Scons!
- Read through program; code will not compile!
- Modify code, replacing "***" as needed.
- Code hardwired to run on 2 MPI processes
- Bonus: modify code to run on 3 MPI processes

Tridiagonal solve (C version)

Tridiagonal solve (C version)

```
/*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
Cblacs_get( ***, ***, *** );
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
Cblacs_get( ***, ***, *** );
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
 Cblacs_get( ***, ***, *** );

 /* ***   INITIALIZE THE VIRTUAL TOPOLOGY AS A 1D ARRAY OF SIZE 1 X NPE
*/
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
 Cblacs_get( ***, ***, *** );

 /* ***   INITIALIZE THE VIRTUAL TOPOLOGY AS A 1D ARRAY OF SIZE 1 X NPE
*/
 Cblacs_gridinit( ***, ***, ***, *** );
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
 Cblacs_get( ***, ***, *** );

 /* ***   INITIALIZE THE VIRTUAL TOPOLOGY AS A 1D ARRAY OF SIZE 1 X NPE
*/
 Cblacs_gridinit( ***, ***, ***, *** );
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
 Cblacs_get( ***, ***, *** );

 /* ***   INITIALIZE THE VIRTUAL TOPOLOGY AS A 1D ARRAY OF SIZE 1 X NPE
*/
 Cblacs_gridinit( ***, ***, ***, *** );

 /* *** FACTORIZE THE MATRIX  */
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
 Cblacs_get( ***, ***, *** );

 /* ***   INITIALIZE THE VIRTUAL TOPOLOGY AS A 1D ARRAY OF SIZE 1 X NPE
*/
 Cblacs_gridinit( ***, ***, ***, *** );

 /* *** FACTORIZE THE MATRIX  */
 pddttrs_( ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***,
***, ***, ***, *** );
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
 Cblacs_get( ***, ***, *** );

 /* ***   INITIALIZE THE VIRTUAL TOPOLOGY AS A 1D ARRAY OF SIZE 1 X NPE
*/
 Cblacs_gridinit( ***, ***, ***, *** );

 /* *** FACTORIZE THE MATRIX  */
 pddttrs_( ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***,
***, ***, ***, *** );
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
 Cblacs_get( ***, ***, *** );

 /* ***   INITIALIZE THE VIRTUAL TOPOLOGY AS A 1D ARRAY OF SIZE 1 X NPE
*/
 Cblacs_gridinit( ***, ***, ***, *** );

 /* *** FACTORIZE THE MATRIX  */
 pddttrs_( ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***,
***, ***, ***, *** );

 /* *** SOLVE THE SYSTEM USING THE ABOVE FACTORIZATION */
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
 Cblacs_get( ***, ***, *** );

 /* ***   INITIALIZE THE VIRTUAL TOPOLOGY AS A 1D ARRAY OF SIZE 1 X NPE
*/
 Cblacs_gridinit( ***, ***, ***, *** );

 /* *** FACTORIZE THE MATRIX  */
 pddttrs_( ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***,
***, ***, ***, *** );

 /* *** SOLVE THE SYSTEM USING THE ABOVE FACTORIZATION */
 pddttrs_( ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***,
***, ***, ***, *** );
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
 Cblacs_get( ***, ***, *** );

 /* ***   INITIALIZE THE VIRTUAL TOPOLOGY AS A 1D ARRAY OF SIZE 1 X NPE
*/
 Cblacs_gridinit( ***, ***, ***, *** );

 /* *** FACTORIZE THE MATRIX  */
 pddttrs_( ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***,
***, ***, ***, *** );

 /* *** SOLVE THE SYSTEM USING THE ABOVE FACTORIZATION */
 pddttrs_( ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***,
***, ***, ***, *** );
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
 Cblacs_get( ***, ***, *** );

 /* ***   INITIALIZE THE VIRTUAL TOPOLOGY AS A 1D ARRAY OF SIZE 1 X NPE
*/
 Cblacs_gridinit( ***, ***, ***, *** );

 /* *** FACTORIZE THE MATRIX  */
 pddttrs_( ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***,
***, ***, ***, *** );

 /* *** SOLVE THE SYSTEM USING THE ABOVE FACTORIZATION */
 pddttrs_( ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***,
***, ***, ***, *** );

 /*    *** EXIT THE GRID */
```

Tridiagonal solve (C version)

```
    /*   ***  RETRIEVE THE DEFAULT SYSTEM CONTEXT FROM BLACS */
 Cblacs_get( ***, ***, *** );

 /* ***   INITIALIZE THE VIRTUAL TOPOLOGY AS A 1D ARRAY OF SIZE 1 X NPE
*/
 Cblacs_gridinit( ***, ***, ***, *** );

 /* *** FACTORIZE THE MATRIX  */
 pddttrs_( ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***,
***, ***, ***, *** );

 /* *** SOLVE THE SYSTEM USING THE ABOVE FACTORIZATION */
 pddttrs_( ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***, ***,
***, ***, ***, *** );

 /*    *** EXIT THE GRID */
 Cblacs_gridexit( *** )
```

ScaLAPACK Summary

ScaLAPACK Summary

- ScaLAPACK is the defacto standard library for dense linear algebra

ScaLAPACK Summary

- ScaLAPACK is the defacto standard library for dense linear algebra
- Complete, high quality, widely available, free

ScaLAPACK Summary

- ScaLAPACK is the defacto standard library for dense linear algebra
- Complete, high quality, widely available, free
- But beware: it is old technology and will be supplanted in the next few years

ScaLAPACK Summary

- ScaLAPACK is the defacto standard library for dense linear algebra
- Complete, high quality, widely available, free
- But beware: it is old technology and will be supplanted in the next few years
- Consider the emerging technologies (D-PLASMA, D-MAGMA) as you develop code

Portable, Extensible Toolkit for Scientific Computation (PETSc)

Portable, Extensible Toolkit for Scientific Computation (PETSc)

Considerations:

Portable, Extensible Toolkit for Scientific Computation (PETSc)

Considerations:

- Developing parallel, non-trivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort.

Portable, Extensible Toolkit for Scientific Computation (PETSc)

Considerations:

- Developing parallel, non-trivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort.
- PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver nor a silver bullet.

What is PETSc

What is PETSc

- A freely available (and supported!) research code
  - Available via http://www.mcs.anl.gov/petsc
  - Free for everyone, including industrial users
  - Hyperlinked documentation and manual pages for all routines
  - Many tutorial-style examples
  - Support via email: petsc-maint@mcs.anl.gov
  - Current version: 3.3 (released Jun. 5, 2012)

What is PETSc

- A freely available (and supported!) research code
  - Available via http://www.mcs.anl.gov/petsc
  - Free for everyone, including industrial users
  - Hyperlinked documentation and manual pages for all routines
  - Many tutorial-style examples
  - Support via email: petsc-maint@mcs.anl.gov
  - Current version: 3.3 (released Jun. 5, 2012)
- Portable to any parallel system supporting MPI
  - Tightly coupled systems, e.g., Cray XK6, XE6
  - Loosely coupled systems, e.g., networks of workstations, PCs

What is PETSc

- A freely available (and supported!) research code
  - Available via http://www.mcs.anl.gov/petsc
  - Free for everyone, including industrial users
  - Hyperlinked documentation and manual pages for all routines
  - Many tutorial-style examples
  - Support via email: petsc-maint@mcs.anl.gov
  - Current version: 3.3 (released Jun. 5, 2012)
- Portable to any parallel system supporting MPI
  - Tightly coupled systems, e.g., Cray XK6, XE6
  - Loosely coupled systems, e.g., networks of workstations, PCs
- Effort started in 1991, funded by US DOE and NSF

PETSc Concepts

PETSc Concepts

- To specify the mathematics of the problem:

PETSc Concepts

- To specify the mathematics of the problem:
  - Programmer manipulates mathematical objects (sparse matrices, nonlinear equations), algorithmic objects (solvers) and discrete geometry (meshes)

PETSc Concepts

- To specify the mathematics of the problem:
  - Programmer manipulates mathematical objects (sparse matrices, nonlinear equations), algorithmic objects (solvers) and discrete geometry (meshes)
- To solve the problem:

PETSc Concepts

- To specify the mathematics of the problem:
  - Programmer manipulates mathematical objects (sparse matrices, nonlinear equations), algorithmic objects (solvers) and discrete geometry (meshes)
- To solve the problem:
  - Solvers: linear, nonlinear, and time stepping (ODE)

PETSc Concepts

- To specify the mathematics of the problem:
  - Programmer manipulates mathematical objects (sparse matrices, nonlinear equations), algorithmic objects (solvers) and discrete geometry (meshes)
- To solve the problem:
  - Solvers: linear, nonlinear, and time stepping (ODE)
- Parallel computing considerations:

PETSc Concepts

- To specify the mathematics of the problem:
  - Programmer manipulates mathematical objects (sparse matrices, nonlinear equations), algorithmic objects (solvers) and discrete geometry (meshes)
- To solve the problem:
  - Solvers: linear, nonlinear, and time stepping (ODE)
- Parallel computing considerations:
  - Parallel data layout, e.g., structured and unstructured meshes

Structure of PETSc

Structure of PETSc

PETSc Numerical Components

PETSc Numerical Components

Nonlinear Solvers

Newton-based Methods		Other
Line Search	Trust Region	

Time Steppers

Euler	Backward Euler	Pseudo Time Stepping	Other

Krylov Subspace Methods

GMRES	CG	CGS	Bi-CG-STAB	TFQMR	Richardson	Chebychev	Other

Preconditioners

Additive Schwartz	Block Jacobi	Jacobi	ILU	ICC	LU (Sequential only)	Others

Matrices

Compressed Sparse Row (AIJ)	Blocked Compressed Sparse Row (BAIJ)	Block Diagonal (BDIAG)	Dense	Matrix-free	Other

Distributed Arrays

Index Sets

Indices	Block Indices	Stride	Other

Vectors

Flow control for PDE solution

Flow control for PDE solution

PETSc Programming Model

PETSc Programming Model

- **Goals**
  - Portable, runs everywhere
  - Performance
  - Scalable parallelism

PETSc Programming Model

- **Goals**
  - Portable, runs everywhere
  - Performance
  - Scalable parallelism
- **Approach**
  - Distributed memory, “shared-nothing”
    - Access to data on remote machines through MPI
  - Can still exploit node parallelism on each node (e.g., SMP), with limitations (see PETSc home page)
  - Hide within parallel objects the details of the communication
  - User orchestrates communication at a higher abstract level than message passing

PETSc Collective Communication

PETSc Collective Communication

- MPI communicators (MPI_Comm) specify collectivity (processors involved in a computation)

PETSc Collective Communication

- MPI communicators (MPI_Comm) specify collectivity (processors involved in a computation)
- All PETSc creation routines for solver and data objects are collective with respect to a communicator, e.g.,
  - `VecCreate(MPI_Comm comm, int m, int M, Vec *x)`

PETSc Collective Communication

- MPI communicators (MPI_Comm) specify collectivity (processors involved in a computation)
- All PETSc creation routines for solver and data objects are collective with respect to a communicator, e.g.,
  - `VecCreate(MPI_Comm comm, int m, int M, Vec *x)`
- Some operations are collective, while others are not, e.g.,
  - collective: `VecNorm()`
  - not collective: `VecGetLocalSize()`

PETSc Collective Communication

- MPI communicators (MPI_Comm) specify collectivity (processors involved in a computation)
- All PETSc creation routines for solver and data objects are collective with respect to a communicator, e.g.,
  - `VecCreate(MPI_Comm comm, int m, int M, Vec *x)`
- Some operations are collective, while others are not, e.g.,
  - collective: `VecNorm()`
  - not collective: `VecGetLocalSize()`
- If a sequence of collective routines is used, they **must** be called in the same order on each processor

PETSc Data Objects

- Vectors (**Vec**)
  - focus: field data arising in nonlinear PDEs
- Matrices (**Mat**)
  - focus: linear operators arising in nonlinear PDEs (i.e., Jacobians)

beginner	• Object creation
beginner	• Object assembly
intermediate	• Setting options
intermediate	• Viewing
advanced	• User-defined customizations

PETSc vectors

- What are PETSc vectors?
  - Fundamental objects for storing field solutions, right-hand sides, etc.
  - Each process locally owns a subvector of contiguously numbered global indices
- Create vectors via
  - VecCreate(...,Vec *)
    - MPI_Comm - processors that share the vector
    - number of elements local to this processor
    - or total number of elements
  - VecSetType(Vec,VecType)
    - Where VecType is
      - VEC_SEQ, VEC_MPI, or VEC_SHARED

Parallel Vector (and Matrix) Assembly

Parallel Vector (and Matrix) Assembly

- Processors may generate any entries in vectors and matrices

Parallel Vector (and Matrix) Assembly

- Processors may generate any entries in vectors and matrices
- Entries need not be generated on the processor on which they ultimately will be stored

Parallel Vector (and Matrix) Assembly

- Processors may generate any entries in vectors and matrices
- Entries need not be generated on the processor on which they ultimately will be stored
- PETSc automatically moves data during the assembly process if necessary

PETSc Vector Assembly

PETSc Vector Assembly

- `VecSetValues(Vec,…)`
  - number of entries to insert/add
  - indices of entries
  - values to add
  - mode: `[INSERT_VALUES,ADD_VALUES]`

PETSc Vector Assembly

- `VecSetValues(Vec,…)`
  - number of entries to insert/add
  - indices of entries
  - values to add
  - mode: `[INSERT_VALUES,ADD_VALUES]`
- `VecAssemblyBegin(Vec)`

PETSc Vector Assembly

- `VecSetValues(Vec,…)`
  - number of entries to insert/add
  - indices of entries
  - values to add
  - mode: `[INSERT_VALUES,ADD_VALUES]`
- `VecAssemblyBegin(Vec)`
- `VecAssemblyEnd(Vec)`

PETSc Matrices

PETSc Matrices

- PETSc matrices are fundamental objects for storing linear operators (e.g., Jacobians)

PETSc Matrices

- PETSc matrices are fundamental objects for storing linear operators (e.g., Jacobians)
- Create matrices via
  - MatCreate(…,Mat *)
    - MPI_Comm - processors that share the matrix
    - number of local/global rows and columns
  - MatSetType(Mat, MatType), where MatType is one of
    - default sparse AIJ: MPIAIJ, SEQAIJ
    - block sparse AIJ (for multi-component PDEs): MPIAIJ, SEQAIJ
    - symmetric block sparse AIJ: MPISBAIJ, SAEQSBAIJ
    - block diagonal: MPIBDIAG, SEQBDIAG
    - dense: MPIDENSE, SEQDENSE
    - matrix-free

PETSc Matrix Assembly

PETSc Matrix Assembly

- MatSetValues(Mat,…)
  - number of rows to insert/add
  - indices of rows and columns
  - number of columns to insert/add
  - values to add
  - mode: [INSERT_VALUES,ADD_VALUES]

PETSc Matrix Assembly

- MatSetValues(Mat,…)
  - number of rows to insert/add
  - indices of rows and columns
  - number of columns to insert/add
  - values to add
  - mode: [INSERT_VALUES,ADD_VALUES]
- MatAssemblyBegin(Mat, MAT_FINAL_ASSEMBLY)

PETSc Matrix Assembly

- MatSetValues(Mat,…)
  - number of rows to insert/add
  - indices of rows and columns
  - number of columns to insert/add
  - values to add
  - mode: [INSERT_VALUES,ADD_VALUES]
- MatAssemblyBegin(Mat, MAT_FINAL_ASSEMBLY)
- MatAssemblyEnd(Mat, MAT_FINAL_ASSEMBLY)

PETSc Parallel Matrix Distribution

Each process locally owns a submatrix of contiguously numbered global rows.

proc 0

PETSc Parallel Matrix Distribution

Each process locally owns a submatrix of contiguously numbered global rows.

PETSc Parallel Matrix Distribution

Each process locally owns a submatrix of contiguously numbered global rows.

```
MatGetOwnershipRange(Mat A, int *rstart, int *rend)
```

- `rstart`: first locally owned row of global matrix
- `rend -1`: last locally owned row of global matrix

Linear Solvers

Goal: Support the solution of linear systems,

$$Ax=b,$$

particularly for sparse, parallel problems arising from PDE-based models.

User provides:

- *A (matrix or operator)*
- *b (right-hand side)*
- *u (initial guess)*

Hestenes/Stiefel, 1952: Conjugate Gradient

Hestenes/Stiefel, 1952: Conjugate Gradient

Hestenes/Stiefel, 1952: Conjugate Gradient

$$k = 0; \quad x_0 = 0; \quad r_0 = 0$$

$$\textit{while} \quad r_k \neq 0 \quad \{$$

$$k = k + 1$$

$$\textit{if}\,(k = 0) \Rightarrow p_1 = r_0$$

$$\textit{if}\,(k > 0) \Rightarrow \beta_k = r_{k-1}^T r_{k-1} / r_{k-2}^T r_{k-2}; \quad p_k = r_{k-1} + \beta_k p_{k-1}$$

$$\alpha_k = r_{k-1}^T r_{k-1} / p_k^T A p_k$$

$$x_k = x_{k-1} + \alpha_k p_k$$

$$r_k = r_{k-1} - \alpha_k A p_k$$

$$\}$$

Some Krylov subspace methods

Some Krylov subspace methods

- Symmetric matrices (requires)

Some Krylov subspace methods

- Symmetric matrices (requires Ax)
  - CG (Conjugate Gradient): convergence ‘assured’

Some Krylov subspace methods

- Symmetric matrices (requires Ax)
  - CG (Conjugate Gradient): convergence ‘assured’
- Non-symmetric matrices (require)

Some Krylov subspace methods

- Symmetric matrices (requires Ax)
  - CG (Conjugate Gradient): convergence ‘assured’
- Non-symmetric matrices (require $Ax \quad A^T x$)
  - BCG (Bi-CG): may not converge

Some Krylov subspace methods

- Symmetric matrices (requires Ax)
  - CG (Conjugate Gradient): convergence 'assured'
- Non-symmetric matrices (require $Ax \quad A^T x$)
  - BCG (Bi-CG): may not converge
  - QMR (Quasi-minimal residual): may not converge

Some Krylov subspace methods

- Symmetric matrices (requires Ax)
  - CG (Conjugate Gradient): convergence ‘assured’
- Non-symmetric matrices (require Ax $A^T x$)
  - BCG (Bi-CG): may not converge
  - QMR (Quasi-minimal residual): may not converge
- Non-symmetric matrices (require)

Some Krylov subspace methods

- Symmetric matrices (requires Ax)
  - CG (Conjugate Gradient): convergence ‘assured’
- Non-symmetric matrices (require Ax $A^T x$)
  - BCG (Bi-CG): may not converge
  - QMR (Quasi-minimal residual): may not converge
- Non-symmetric matrices (require Ax)
  - GMRES (Generalized Minimal Residuals): ‘converges’

Some Krylov subspace methods

- Symmetric matrices (requires Ax)
  - CG (Conjugate Gradient): convergence ‘assured’
- Non-symmetric matrices (require Ax $A^T x$)
  - BCG (Bi-CG): may not converge
  - QMR (Quasi-minimal residual): may not converge
- Non-symmetric matrices (require Ax)
  - GMRES (Generalized Minimal Residuals): ‘converges’
  - BiCGstab (Bi-CG stabilized): may not converge

Some Krylov subspace methods

- Symmetric matrices (requires Ax)
  - CG (Conjugate Gradient): convergence ‘assured’
- Non-symmetric matrices (require Ax $A^T x$)
  - BCG (Bi-CG): may not converge
  - QMR (Quasi-minimal residual): may not converge
- Non-symmetric matrices (require Ax)
  - GMRES (Generalized Minimal Residuals): ‘converges’
  - BiCGstab (Bi-CG stabilized): may not converge
  - CGS (CG squared): may not converge

Some Krylov subspace methods

- Symmetric matrices (requires Ax)
  - CG (Conjugate Gradient): convergence 'assured'
- Non-symmetric matrices (require Ax $A^T x$)
  - BCG (Bi-CG): may not converge
  - QMR (Quasi-minimal residual): may not converge
- Non-symmetric matrices (require Ax)
  - GMRES (Generalized Minimal Residuals): 'converges'
  - BiCGstab (Bi-CG stabilized): may not converge
  - CGS (CG squared): may not converge
  - TFQMR (Transpose-free QMR): may not converge

Preconditioners: KSM alone insufficient!

Preconditioners: KSM alone insufficient!

- CG method initially ignored due to slow convergence

Preconditioners: KSM alone insufficient!

- CG method initially ignored due to slow convergence
  - Theoretical convergence after 2*n steps, but n is huge

Preconditioners: KSM alone insufficient!

- CG method initially ignored due to slow convergence
  - Theoretical convergence after 2*n steps, but n is huge
  - Convergence rate related to ratio largest/smallest eigenvalue

Preconditioners: KSM alone insufficient!

- CG method initially ignored due to slow convergence
  - Theoretical convergence after 2*n steps, but n is huge
  - Convergence rate related to ratio largest/smallest eigenvalue
- Easier problem: preconditioner

$$M \approx A$$

$$Ax = b \Rightarrow M^{-1}Ax = M^{-1}b$$

Preconditioners: KSM alone insufficient!

- CG method initially ignored due to slow convergence
  - Theoretical convergence after 2*n steps, but n is huge
  - Convergence rate related to ratio largest/smallest eigenvalue
- Easier problem: preconditioner $M \approx A$ $Ax = b \Rightarrow M^{-1}Ax = M^{-1}b$
  - Find an approximation for A with $M^{-1}x$ 'easily' calculated

Preconditioners: KSM alone insufficient!

- CG method initially ignored due to slow convergence
  - Theoretical convergence after 2*n steps, but n is huge
  - Convergence rate related to ratio largest/smallest eigenvalue
- Easier problem: preconditioner $M \approx A$ $Ax = b \Rightarrow M^{-1}Ax = M^{-1}b$
  - Find an approximation for A with $M^{-1}x$ 'easily' calculated
  - Possibilities:

Preconditioners: KSM alone insufficient!

- CG method initially ignored due to slow convergence
  - Theoretical convergence after 2*n steps, but n is huge
  - Convergence rate related to ratio largest/smallest eigenvalue
- Easier problem: preconditioner $M \approx A$ $Ax = b \Rightarrow M^{-1}Ax = M^{-1}b$
  - Find an approximation for A with $M^{-1}x$ 'easily' calculated
  - Possibilities:
    - Approximate inverse known through physical description

Preconditioners: KSM alone insufficient!

- CG method initially ignored due to slow convergence
  - Theoretical convergence after 2*n steps, but n is huge
  - Convergence rate related to ratio largest/smallest eigenvalue
- Easier problem: preconditioner $M \approx A$ $Ax = b \Rightarrow M^{-1}Ax = M^{-1}b$
  - Find an approximation for A with $M^{-1}x$ 'easily' calculated
  - Possibilities:
    - Approximate inverse known through physical description
    - Incomplete LU decomposition

Preconditioners: KSM alone insufficient!

- CG method initially ignored due to slow convergence
  - Theoretical convergence after 2*n steps, but n is huge
  - Convergence rate related to ratio largest/smallest eigenvalue
- Easier problem: preconditioner $M \approx A$ $Ax = b \Rightarrow M^{-1}Ax = M^{-1}b$
  - Find an approximation for A with $M^{-1}x$ 'easily' calculated
  - Possibilities:
    - Approximate inverse known through physical description
    - Incomplete LU decomposition
    - Sparse approximative inverse (assume inverse also sparse)

Preconditioners: KSM alone insufficient!

- CG method initially ignored due to slow convergence
  - Theoretical convergence after 2*n steps, but n is huge
  - Convergence rate related to ratio largest/smallest eigenvalue
- Easier problem: preconditioner $M \approx A$ $Ax = b \Rightarrow M^{-1}Ax = M^{-1}b$
  - Find an approximation for A with $M^{-1}x$ 'easily' calculated
  - Possibilities:
    - Approximate inverse known through physical description
    - Incomplete LU decomposition
    - Sparse approximative inverse (assume inverse also sparse)
    - Multilevel (multigrid) preconditioners

Preconditioners: KSM alone insufficient!

- CG method initially ignored due to slow convergence
  - Theoretical convergence after 2*n steps, but n is huge
  - Convergence rate related to ratio largest/smallest eigenvalue
- Easier problem: preconditioner $M \approx A$ $Ax = b \Rightarrow M^{-1}Ax = M^{-1}b$
  - Find an approximation for A with $M^{-1}x$ 'easily' calculated
  - Possibilities:
    - Approximate inverse known through physical description
    - Incomplete LU decomposition
    - Sparse approximative inverse (assume inverse also sparse)
    - Multilevel (multigrid) preconditioners
    - More…

PETSc Linear Solvers (subset)

PETSc Linear Solvers (subset)

Krylov Methods (KSP)

PETSc Linear Solvers (subset)

Krylov Methods (KSP)

- Conjugate Gradient
- GMRES
- CG-Squared
- Bi-CG-stab
- Transpose-free QMR
- etc.

PETSc Linear Solvers (subset)

Krylov Methods (KSP)

- Conjugate Gradient
- GMRES
- CG-Squared
- Bi-CG-stab
- Transpose-free QMR
- etc.

Preconditioners (PC)

PETSc Linear Solvers (subset)

Krylov Methods (KSP)

- Conjugate Gradient
- GMRES
- CG-Squared
- Bi-CG-stab
- Transpose-free QMR
- etc.

Preconditioners (PC)

- Block Jacobi
- Overlapping Additive Schwarz
- ICC, ILU via BlockSolve95
- ILU(k), LU (sequential only)
- etc.

PETSc Krylov Subspace Solvers

PETSc Krylov Subspace Solvers

Only KSP solvers discussed here

PETSc Krylov Subspace Solvers

Only KSP solvers discussed here

- Create a KSP solver:

```
ierr = KSPCreate(PETSC_COMM_WORLD,&ksp)
```

PETSc Krylov Subspace Solvers

Only KSP solvers discussed here

- Create a KSP solver:

```
ierr = KSPCreate(PETSC_COMM_WORLD,&ksp)
```

- Use matrix to define an operator

```
ierr = KSPSetOperators(ksp,A,A,XXXX)
```

PETSc Krylov Subspace Solvers

Only KSP solvers discussed here

- Create a KSP solver:

```
ierr = KSPCreate(PETSC_COMM_WORLD,&ksp)
```

- Use matrix to define an operator

```
ierr = KSPSetOperators(ksp,A,A,XXXX)
```

- Set the solver from the command-line options

```
ierr = KSPSetFromOptions(ksp);
```

PETSc Krylov Subspace Solvers

Only KSP solvers discussed here

- Create a KSP solver:

```
ierr = KSPCreate(PETSC_COMM_WORLD,&ksp)
```

- Use matrix to define an operator

```
ierr = KSPSetOperators(ksp,A,A,XXXX)
```

- Set the solver from the command-line options

```
ierr = KSPSetFromOptions(ksp);
```

- Set the solver from the command-line options

```
ierr = KSPSolve(ksp,b,u);
```

PETSc Profiling and Logging

Profiling:

- beginner — Integrated profiling using –log_summary
- intermediate — User-defined events
- intermediate — Profiling by stages of an application

Performance Tuning:

- intermediate — Matrix optimizations
- intermediate — Application optimizations
- advanced — Algorithmic tuning

PETSc Profiling

PETSc Profiling

- Integrated monitoring of
  - time
  - floating-point performance
  - memory usage
  - communication

PETSc Profiling

- Integrated monitoring of
  - time
  - floating-point performance
  - memory usage
  - communication
- All PETSc events are logged if compiled with `-DPETSC_LOG` (default); can also profile application code segments

PETSc Profiling

- Integrated monitoring of
  - time
  - floating-point performance
  - memory usage
  - communication
- All PETSc events are logged if compiled with `-DPETSC_LOG` (default); can also profile application code segments
- Print summary data with option: `-log_summary`

PETSc Profiling

- Integrated monitoring of
  - time
  - floating-point performance
  - memory usage
  - communication
- All PETSc events are logged if compiled with `-DPETSC_LOG` (default); can also profile application code segments
- Print summary data with option: `-log_summary`
- Print information from PETSc routines: `-log_info`

PETSc Profiling

- Integrated monitoring of
  - time
  - floating-point performance
  - memory usage
  - communication
- All PETSc events are logged if compiled with `-DPETSC_LOG` (default); can also profile application code segments
- Print summary data with option: `-log_summary`
- Print information from PETSc routines: `-log_info`
- Print the trace of the functions called: `-log_trace`

PETSc User-defined Event

PETSc User-defined Event

```
int USER_EVENT;
```

PETSc User-defined Event

```
int USER_EVENT;
int user_event_flops
```

PETSc User-defined Event

```
int USER_EVENT;
int user_event_flops
PetscLogEventRegister(&USER_EVENT,"User event
    name", "eventColor");
```

PETSc User-defined Event

```
int USER_EVENT;
int user_event_flops
PetscLogEventRegister(&USER_EVENT,"User event
    name", "eventColor");
PetscLogEventBegin(USER_EVENT,0,0,0,0);
```

PETSc User-defined Event

```
int USER_EVENT;
int user_event_flops
PetscLogEventRegister(&USER_EVENT,"User event
   name", "eventColor");
PetscLogEventBegin(USER_EVENT,0,0,0,0);
[ code to monitor]
```

PETSc User-defined Event

```
int USER_EVENT;
int user_event_flops
PetscLogEventRegister(&USER_EVENT,"User event
   name", "eventColor");
PetscLogEventBegin(USER_EVENT,0,0,0,0);
[ code to monitor]
PetscLogFlops(user_evnet_flops);
```

PETSc User-defined Event

```
int USER_EVENT;
int user_event_flops
PetscLogEventRegister(&USER_EVENT,"User event
   name", "eventColor");
PetscLogEventBegin(USER_EVENT,0,0,0,0);
[ code to monitor]
PetscLogFlops(user_evnet_flops);
PetscLogEvent End(USER_EVENT,0,0,0,0);
```

Case Study 2: elliptic PDE solver

Case Study 2: elliptic PDE solver

Objectives:

Case Study 2: elliptic PDE solver

Objectives:

- Get acquainted with PETSc

Case Study 2: elliptic PDE solver

Objectives:

- Get acquainted with PETSc
- Peek into parallel performance

Case Study 2: elliptic PDE solver

Objectives:

- Get acquainted with PETSc
- Peek into parallel performance

Tasks:

Case Study 2: elliptic PDE solver

Objectives:

- Get acquainted with PETSc
- Peek into parallel performance

Tasks:

- Create/define/assemble an FE stiffness matrix

Case Study 2: elliptic PDE solver

Objectives:

- Get acquainted with PETSc
- Peek into parallel performance

Tasks:

- Create/define/assemble an FE stiffness matrix
- Create/define/assemble right-hand side (stemming from the Dirichlet boundary condition)

Case Study 2: elliptic PDE solver

Objectives:

- Get acquainted with PETSc
- Peek into parallel performance

Tasks:

- Create/define/assemble an FE stiffness matrix
- Create/define/assemble right-hand side (stemming from the Dirichlet boundary condition)
- Solve system

Case Study 2: elliptic PDE solver

Objectives:

- Get acquainted with PETSc
- Peek into parallel performance

Tasks:

- Create/define/assemble an FE stiffness matrix
- Create/define/assemble right-hand side (stemming from the Dirichlet boundary condition)
- Solve system
- Evaluate performance

Case Study 2: assignment

Case Study 2: assignment

- Code:
  - `git clone https://github.com/fomics/SummerSchool2013.git`
  - `cd Libraries/PETSc`
  - README -- read this first
  - Main program: `FEsolver.c`
  - Makefile
  - http://www.mcs.anl.gov/petsc/petsc-as/documentation/index.html

Case Study 2: assignment

- Code:
  - `git clone` https://github.com/fomics/SummerSchool2013.git
  - `cd Libraries/PETSc`
  - README -- read this first
  - Main program: `FEsolver.c`
  - Makefile
  - http://www.mcs.anl.gov/petsc/petsc-as/documentation/index.html

- Read through `FEsolver.c`; Code will not compile!

Case Study 2: assignment

- Code:
  - `git clone` https://github.com/fomics/SummerSchool2013.git
  - `cd Libraries/PETSc`
  - README -- read this first
  - Main program: `FEsolver.c`
  - Makefile
  - http://www.mcs.anl.gov/petsc/petsc-as/documentation/index.html

- Read through `FEsolver.c`; Code will not compile!
- Modify `FEsolver.c` (replacing "***" as needed)

Case Study 2: assignment

- Code:
  - `git clone` https://github.com/fomics/SummerSchool2013.git
  - `cd Libraries/PETSc`
  - README -- read this first
  - Main program: `FEsolver.c`
  - Makefile
  - http://www.mcs.anl.gov/petsc/petsc-as/documentation/index.html

- Read through `FEsolver.c`; Code will not compile!
- Modify `FEsolver.c` (replacing "***" as needed)
- Experiment with command line arguments – vary problem size, and number of processes; evaluate performance.

Case Study 2: assignment

- Code:
  - `git clone` https://github.com/fomics/SummerSchool2013.git
  - `cd Libraries/PETSc`
  - README -- read this first
  - Main program: `FEsolver.c`
  - Makefile
  - http://www.mcs.anl.gov/petsc/petsc-as/documentation/index.html

- Read through `FEsolver.c`; Code will not compile!
- Modify `FEsolver.c` (replacing "***" as needed)
- Experiment with command line arguments – vary problem size, and number of processes; evaluate performance.
- Bonus: use PETSc events to understand the setup overhead

PETSc Summary

PETSc Summary

- PETSc library is one of the victors of 20 years of development on PDE/ODE solvers

PETSc Summary

- PETSc library is one of the victors of 20 years of development on PDE/ODE solvers
- Extensive selection of solvers

PETSc Summary

- PETSc library is one of the victors of 20 years of development on PDE/ODE solvers
- Extensive selection of solvers
- High quality, good support, free

PETSc Summary

- PETSc library is one of the victors of 20 years of development on PDE/ODE solvers
- Extensive selection of solvers
- High quality, good support, free
- However: much more effective for new code

PETSc Summary

- PETSc library is one of the victors of 20 years of development on PDE/ODE solvers
- Extensive selection of solvers
- High quality, good support, free
- However: much more effective for new code
- Saddled by design choices, i.e., not thread-safe

PETSc Summary

- PETSc library is one of the victors of 20 years of development on PDE/ODE solvers
- Extensive selection of solvers
- High quality, good support, free
- However: much more effective for new code
- Saddled by design choices, i.e., not thread-safe
- Monolithic: one package tries to solve all, though there are adaptors to other libraries

Trilinos: a ‘pearl necklace’ of packages

Trilinos: a ‘pearl necklace’ of packages

Trilinos: a ‘pearl necklace’ of packages

Object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems

Trilinos: a ‘pearl necklace’ of packages

Object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems

Trilinos: a ‘pearl necklace’ of packages

Object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems

- http://trilinos.sandia.gov

Trilinos: a ‘pearl necklace’ of packages

Object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems

- http://trilinos.sandia.gov
- http://code.google.com/p/trilinos/wiki/TrilinosHandsOnTutorial

Trilinos: problems covered

Trilinos: problems covered

- Construction and use of sparse and dense matrices, graphs and vectors

Trilinos: problems covered

- Construction and use of sparse and dense matrices, graphs and vectors
- Iterative and direct solution of linear systems

Trilinos: problems covered

- Construction and use of sparse and dense matrices, graphs and vectors
- Iterative and direct solution of linear systems
- Parallel multilevel and algebraic preconditioning

Trilinos: problems covered

- Construction and use of sparse and dense matrices, graphs and vectors
- Iterative and direct solution of linear systems
- Parallel multilevel and algebraic preconditioning
- Solution of non-linear, eigenvalue and time-dependent problems

Trilinos: problems covered

- Construction and use of sparse and dense matrices, graphs and vectors
- Iterative and direct solution of linear systems
- Parallel multilevel and algebraic preconditioning
- Solution of non-linear, eigenvalue and time-dependent problems
- Constrained optimization problems

Trilinos: problems covered

- Construction and use of sparse and dense matrices, graphs and vectors
- Iterative and direct solution of linear systems
- Parallel multilevel and algebraic preconditioning
- Solution of non-linear, eigenvalue and time-dependent problems
- Constrained optimization problems
- Partitioning and load balancing of distributed data structures

Trilinos: problems covered

- Construction and use of sparse and dense matrices, graphs and vectors
- Iterative and direct solution of linear systems
- Parallel multilevel and algebraic preconditioning
- Solution of non-linear, eigenvalue and time-dependent problems
- Constrained optimization problems
- Partitioning and load balancing of distributed data structures
- Automatic differentiation

Trilinos: problems covered

- Construction and use of sparse and dense matrices, graphs and vectors
- Iterative and direct solution of linear systems
- Parallel multilevel and algebraic preconditioning
- Solution of non-linear, eigenvalue and time-dependent problems
- Constrained optimization problems
- Partitioning and load balancing of distributed data structures
- Automatic differentiation
- PDE discretizations

Trilinos: problems covered

- Construction and use of sparse and dense matrices, graphs and vectors
- Iterative and direct solution of linear systems
- Parallel multilevel and algebraic preconditioning
- Solution of non-linear, eigenvalue and time-dependent problems
- Constrained optimization problems
- Partitioning and load balancing of distributed data structures
- Automatic differentiation
- PDE discretizations
- More …

Trilinos: parallel packages

Trilinos: parallel packages

- Basic linear algebra: *Epetra/EpetraExt (C++), Tpetra (C++ templates)*

Trilinos: parallel packages

- Basic linear algebra: *Epetra/EpetraExt (C++), Tpetra (C++ templates)*
- Preconditioners: *AztecOO, Ifpack/Tifpack, ML, Meros*

Trilinos: parallel packages

- Basic linear algebra: *Epetra/EpetraExt (C++), Tpetra (C++ templates)*
- Preconditioners: *AztecOO, Ifpack/Tifpack, ML, Meros*
- Iterative linear solvers: *AztecOO, Belos*

Trilinos: parallel packages

- Basic linear algebra: *Epetra/EpetraExt (C++), Tpetra (C++ templates)*
- Preconditioners: *AztecOO, Ifpack/Tifpack, ML, Meros*
- Iterative linear solvers: *AztecOO, Belos*
- Direct linear solvers: *Amesos (SuperLU, UMFPACK, MUMPS, ScaLAPACK, …)*

Trilinos: parallel packages

- Basic linear algebra: *Epetra/EpetraExt (C++), Tpetra (C++ templates)*
- Preconditioners: *AztecOO, Ifpack/Tifpack, ML, Meros*
- Iterative linear solvers: *AztecOO, Belos*
- Direct linear solvers: *Amesos (SuperLU, UMFPACK, MUMPS, ScaLAPACK, …)*
- Non-linear / optimization solvers: *NOX, MOOCHO*

Trilinos: parallel packages

- Basic linear algebra: *Epetra/EpetraExt (C++), Tpetra (C++ templates)*
- Preconditioners: *AztecOO, Ifpack/Tifpack, ML, Meros*
- Iterative linear solvers: *AztecOO, Belos*
- Direct linear solvers: *Amesos (SuperLU, UMFPACK, MUMPS, ScaLAPACK, …)*
- Non-linear / optimization solvers: *NOX, MOOCHO*
- Eigensolvers: *Anasazi*

Trilinos: parallel packages

- Basic linear algebra: *Epetra/EpetraExt (C++), Tpetra (C++ templates)*
- Preconditioners: *AztecOO, Ifpack/Tifpack, ML, Meros*
- Iterative linear solvers: *AztecOO, Belos*
- Direct linear solvers: *Amesos (SuperLU, UMFPACK, MUMPS, ScaLAPACK, …)*
- Non-linear / optimization solvers: *NOX, MOOCHO*
- Eigensolvers: *Anasazi*
- Mesh generation / adaptivity: *Mesquite, PAMGEN*

Trilinos: parallel packages

- Basic linear algebra: *Epetra/EpetraExt (C++), Tpetra (C++ templates)*
- Preconditioners: *AztecOO, Ifpack/Tifpack, ML, Meros*
- Iterative linear solvers: *AztecOO, Belos*
- Direct linear solvers: *Amesos (SuperLU, UMFPACK, MUMPS, ScaLAPACK, …)*
- Non-linear / optimization solvers: *NOX, MOOCHO*
- Eigensolvers: *Anasazi*
- Mesh generation / adaptivity: *Mesquite, PAMGEN*
- Domain decomposition: *Claps*

Trilinos: parallel packages

- Basic linear algebra: *Epetra/EpetraExt (C++), Tpetra (C++ templates)*
- Preconditioners: *AztecOO, Ifpack/Tifpack, ML, Meros*
- Iterative linear solvers: *AztecOO, Belos*
- Direct linear solvers: *Amesos (SuperLU, UMFPACK, MUMPS, ScaLAPACK, …)*
- Non-linear / optimization solvers: *NOX, MOOCHO*
- Eigensolvers: *Anasazi*
- Mesh generation / adaptivity: *Mesquite, PAMGEN*
- Domain decomposition: *Claps*
- Partitioning / load balance: *Isorropia, Zoltan*

Trilinos solver overview

Trilinos solver overview

Optimization **Unconstrained:** **Constrained:**	Find $u \in \Re^n$ that minimizes $g(u)$ Find $x \in \Re^m$ and $u \in \Re^n$ that minimizes $g(x,u)$ s.t. $f(x,u)=0$	**Sensitivities (Automatic Differentiation: Sacado)**	**MOOCHO**
Bifurcation Analysis	Given nonlinear operator $F(x,u) \in \Re^{n+m}$ For $F(x,u)=0$ find space $u \in U \ni \frac{\partial F}{\partial x}$		**LOCA**
Transient Problems **DAEs/ODEs:**	Solve $f(\dot{x}(t),x(t),t)=0$ $t \in [0,T], x(0)=x_0, \dot{x}(0)=x_0'$ for $x(t) \in \Re^n, t \in [0,T]$		**Rythmos**
Nonlinear Problems	Given nonlinear operator $F(x) \in \Re^m \rightarrow \Re$ Solve $F(x)=0$ $x \in \Re^n$		**NOX**
Linear Problems **Linear Equations:** **Eigen Problems:**	Given Linear Ops (Matrices) $A,B \in \Re^{m\times n}$ Solve $Ax=b$ for $x \in \Re^n$ Solve $A\nu=\lambda B\nu$ for (all) $\nu \in \Re^n$, $\lambda \in$		**AztecOO** **Belos** **Ifpack, ML, etc...** **Anasazi**
Distributed Linear Algebra **Matrix/Graph Equations:** **Vector Problems:**	Compute $y=Ax; A=A(G); A \in \Re^{m\times n}, G \in \Im^{m\times n}$ Compute $y=\alpha x+\beta w; \alpha=\langle x,y\rangle; x,y \in \Re^n$		**Epetra** **Tpetra**

Trilinos / PETSc Interoperability

Trilinos / PETSc Interoperability

- `Epetra_PETScAIJMatrix` class

Trilinos / PETSc Interoperability

- `Epetra_PETScAIJMatrix` class
- Allows PETSc to call Trilinos preconditioners

Trilinos / PETSc Interoperability

- `Epetra_PETScAIJMatrix` class
- Allows PETSc to call Trilinos preconditioners
- ML (Multilevel precond.) can call PETSc solvers as smoothers

Trilinos Summary

Trilinos Summary

- Non-monolithic set of packages, some interoperating tightly, some loosely, some not at all

Trilinos Summary

- Non-monolithic set of packages, some interoperating tightly, some loosely, some not at all
- Large development team, free software

Trilinos Summary

- Non-monolithic set of packages, some interoperating tightly, some loosely, some not at all
- Large development team, free software
- Technically advanced, latest solvers, following emerging technologies (e.g. GPUs)

Trilinos Summary

- Non-monolithic set of packages, some interoperating tightly, some loosely, some not at all
- Large development team, free software
- Technically advanced, latest solvers, following emerging technologies (e.g. GPUs)
- Solvers are opaque, hard to see internals

Trilinos Summary

- Non-monolithic set of packages, some interoperating tightly, some loosely, some not at all
- Large development team, free software
- Technically advanced, latest solvers, following emerging technologies (e.g. GPUs)
- Solvers are opaque, hard to see internals
- Trilinos bugs can be hard to deal with

Libraries for Sparse Linear Algebra

- ScaLAPACK limited to $m \times n$ matrices with $m,n = O(10^4)$
- Sparse matrices typically contain at least 90% zeros
- Number of non-zero (nz) elements, large: $nz = O(10^7)$

Libraries for Sparse Linear Algebra

- ScaLAPACK limited to $m \times n$ matrices with $m,n = O(10^4)$
- Sparse matrices typically contain at least 90% zeros
- Number of non-zero (nz) elements, large: $nz = O(10^7)$

Libraries for Sparse Linear Algebra

- ScaLAPACK limited to $m \times n$ matrices with $m,n = O(10^4)$
- Sparse matrices typically contain at least 90% zeros
- Number of non-zero (nz) elements, large: $nz = O(10^7)$

Libraries for Sparse Linear Algebra

- ScaLAPACK limited to $m \times n$ matrices with $m,n = O(10^4)$
- Sparse matrices typically contain at least 90% zeros
- Number of non-zero (nz) elements, large: $nz = O(10^7)$

Libraries for Sparse Linear Algebra

- ScaLAPACK limited to $m \times n$ matrices with $m,n = O(10^4)$
- Sparse matrices typically contain at least 90% zeros
- Number of non-zero (nz) elements, large: $nz = O(10^7)$

Linear System Solution: $Ax = b$

Linear System Solution: $Ax = b$

Two basic techniques:

- Direct methods, i.e. factorize matrix
  - good for multiple right hand sides
  - tend to be more robust
- Iterative methods
  - good if matrix known via operators
  - possibilities for approximate solutions

Linear System Solution: $Ax = b$

Two basic techniques:

- Direct methods, i.e. factorize matrix
  - good for multiple right hand sides $AX = B$
  - tend to be more robust
- Iterative methods
  - good if matrix known via operators
  - possibilities for approximate solutions

Linear System Solution: $Ax = b$

Two basic techniques:

- Direct methods, i.e. factorize matrix
  - good for multiple right hand sides $AX = B$
  - tend to be more robust
- Iterative methods
  - good if matrix known via operators $Ax \quad A^T x$
  - possibilities for approximate solutions

LU decomposition

LU decomposition

- Formulations:

LU decomposition

- Formulations:
- Permutation Matrices, P, Q:

LU decomposition

- Formulations: $A = LU$
- Permutation Matrices, P, Q:

LU decomposition

- Formulations: $A = LU \qquad A = P^T LU$
- Permutation Matrices, P, Q:

LU decomposition

- Formulations: $A = LU$ $\quad A = P^T LU$ $\quad A = P^T LUQ^T$
- Permutation Matrices, P, Q:

LU decomposition

- Formulations: $A = LU$ $A = P^T LU$ $A = P^T LUQ^T$
- Permutation Matrices, P, Q:
- Fill-in:

LU decomposition

- Formulations: $A = LU$ $\quad A = P^T LU$ $\quad A = P^T LUQ^T$
- Permutation Matrices, P, Q:
- Fill-in:

$$P = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

LU decomposition

- Formulations: $A = LU$ $\quad A = P^T LU$ $\quad A = P^T LUQ^T$
- Permutation Matrices, P, Q:
- Fill-in:

$$P = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Parallel direct linear system solvers

Parallel direct linear system solvers

- SuperLU_Dist (Super-nodal LU decomposition, C)
  - http://acts.nersc.gov/superlu/
- MUMPS (Multifrontal Massively Parallel, F90)
  - http://graal.ens-lyon.fr/MUMPS/
- PSPASES (Par. SPArse Symm. dirEct, C)
  - http://www-users.cs.umn.edu/~mjoshi/pspases/
- PaStiX (Parallel Sparse matriX package, C)
  - http://pastix.gforge.inria.fr
- PARDISO (SMP), SPOOLES, PSPIKE, others…

Performance comparison

Performance comparison

- Each technique has strengths/weaknesses
  - Gould, Scott, RAL-TR-2005-005
- MUMPS and SuperLU possibly the two most used – comparison inconclusive

Performance comparison

- Each technique has strengths/weaknesses
  - Gould, Scott, RAL-TR-2005-005
- MUMPS and SuperLU possibly the two most used – comparison inconclusive

Matrix	Ordering	Solver	Number of processors						
			1	4	8	16	32	64	128
bbmat	AMD	MUMPS	-	44.8	23.6	15.7	12.6	10.1	9.5
		SuperLU	-	64.7	36.6	21.3	12.8	9.2	7.2
	ND(metis)	MUMPS	-	32.1	10.8	12.3	10.4	9.1	7.8
		SuperLU	-	132.9	72.5	39.8	23.5	15.6	11.1
ecl32	AMD	MUMPS	-	53.1	31.3	20.7	14.7	13.5	12.9
		SuperLU	-	106.8	56.7	31.2	18.3	12.3	8.2
	ND(metis)	MUMPS	-	23.9	13.4	9.7	6.6	5.6	5.4
		SuperLU	-	48.5	26.6	15.7	9.6	7.6	5.6

Direct solvers: PETSc interfaces

Direct solvers: PETSc interfaces

- MUMPS

Direct solvers: PETSc interfaces

- MUMPS
- SuperLU_Dist

Direct solvers: PETSc interfaces

- MUMPS
- SuperLU_Dist
- PaStiX

Direct solvers: PETSc interfaces

- MUMPS
- SuperLU_Dist
- PaStiX
- SPOOLES

Direct solvers: PETSc interfaces

- MUMPS
- SuperLU_Dist
- PaStiX
- SPOOLES
- UMFPACK (serial)

Iterative linear system solvers

Iterative linear system solvers

- Older methods:

Iterative linear system solvers

- Older methods:
  - Jacobi iteration

Iterative linear system solvers

- Older methods:
  - Jacobi iteration
  - Gauss/Seidel

Iterative linear system solvers

- Older methods:
  - Jacobi iteration
  - Gauss/Seidel
  - SOR/SSOR: Successive Over Relaxation

Iterative linear system solvers

- Older methods:
  - Jacobi iteration
  - Gauss/Seidel
  - SOR/SSOR: Successive Over Relaxation
  - Method of steepest descent

Iterative linear system solvers

- Older methods:
  - Jacobi iteration
  - Gauss/Seidel
  - SOR/SSOR: Successive Over Relaxation
  - Method of steepest descent
- Newer methods:

Iterative linear system solvers

- Older methods:
  - Jacobi iteration
  - Gauss/Seidel
  - SOR/SSOR: Successive Over Relaxation
  - Method of steepest descent
- Newer methods:
  - Krylov subspace methods, e.g.,

Iterative linear system solvers

- Older methods:
  - Jacobi iteration
  - Gauss/Seidel
  - SOR/SSOR: Successive Over Relaxation
  - Method of steepest descent
- Newer methods:
  - Krylov subspace methods, e.g.,
  - Conjugate Gradients (CG)

Iterative linear system solvers

- Older methods:
  - Jacobi iteration
  - Gauss/Seidel
  - SOR/SSOR: Successive Over Relaxation
  - Method of steepest descent
- Newer methods:
  - Krylov subspace methods, e.g.,
  - Conjugate Gradients (CG)

Parallel libraries with KSM

Parallel libraries with KSM

- PETSc: all of the above and more

Parallel libraries with KSM

- PETSc: all of the above and more
- AztecOO: Trilinos package, numerous methods
  - Object oriented, C++, requires Epetra / EpetraExt
  - http://trilinos.sandia.gov/packages/aztecoo/
- BELOS: Trilinos package, reaching maturity
  - Templated operators and vectors as opaque objects
  - http://trilinos.sandia.gov/packages/belos/
- pARMS (parallel Algebraic Recursive Multilevel Solvers)
  - Currently only FGMRES supported
  - http://www-users.cs.umn.edu/~saad/software/pARMS/
- Hypre (iterative solvers and scalable preconditioners)
  - http://acts.nersc.gov/hypre/

Parallel preconditioner libraries

Parallel preconditioner libraries

- SPAI/MSPAI (Modified Sparse Approximative Inverse)

Parallel preconditioner libraries

- SPAI/MSPAI (Modified Sparse Approximative Inverse)
  - Numerical technique is inherently parallel

Parallel preconditioner libraries

- SPAI/MSPAI (Modified Sparse Approximative Inverse)
  - Numerical technique is inherently parallel
  - http://www5.in.tum.de/wiki/index.php/MSPAI

Parallel preconditioner libraries

- SPAI/MSPAI (Modified Sparse Approximative Inverse)
  - Numerical technique is inherently parallel
  - http://www5.in.tum.de/wiki/index.php/MSPAI
- Trilinos

Parallel preconditioner libraries

- SPAI/MSPAI (Modified Sparse Approximative Inverse)
  - Numerical technique is inherently parallel
  - http://www5.in.tum.de/wiki/index.php/MSPAI
- Trilinos
  - Various (AztecOO), Multilevel (ML), Incomplete Factor ([T]IFPACK)

Parallel preconditioner libraries

- SPAI/MSPAI (Modified Sparse Approximative Inverse)
  - Numerical technique is inherently parallel
  - http://www5.in.tum.de/wiki/index.php/MSPAI
- Trilinos
  - Various (AztecOO), Multilevel (ML), Incomplete Factor ([T]IFPACK)
  - http://trilinos.sandia.gov/packages/

Parallel preconditioner libraries

- SPAI/MSPAI (Modified Sparse Approximative Inverse)
  - Numerical technique is inherently parallel
  - http://www5.in.tum.de/wiki/index.php/MSPAI
- Trilinos
  - Various (AztecOO), Multilevel (ML), Incomplete Factor ([T]IFPACK)
  - http://trilinos.sandia.gov/packages/
- Hypre (high-perf preconditioners)

Parallel preconditioner libraries

- SPAI/MSPAI (Modified Sparse Approximative Inverse)
  - Numerical technique is inherently parallel
  - http://www5.in.tum.de/wiki/index.php/MSPAI
- Trilinos
  - Various (AztecOO), Multilevel (ML), Incomplete Factor ([T]IFPACK)
  - http://trilinos.sandia.gov/packages/
- Hypre (high-perf preconditioners)
  - High level interface for grid-based problems

Parallel preconditioner libraries

- SPAI/MSPAI (Modified Sparse Approximative Inverse)
  - Numerical technique is inherently parallel
  - http://www5.in.tum.de/wiki/index.php/MSPAI
- Trilinos
  - Various (AztecOO), Multilevel (ML), Incomplete Factor ([T]IFPACK)
  - http://trilinos.sandia.gov/packages/
- Hypre (high-perf preconditioners)
  - High level interface for grid-based problems
  - ILU (Euclid/pilut), multigrid (PFMG/BoomerAMG), sparse (parasails)

Parallel preconditioner libraries

- SPAI/MSPAI (Modified Sparse Approximative Inverse)
  - Numerical technique is inherently parallel
  - http://www5.in.tum.de/wiki/index.php/MSPAI
- Trilinos
  - Various (AztecOO), Multilevel (ML), Incomplete Factor ([T]IFPACK)
  - http://trilinos.sandia.gov/packages/
- Hypre (high-perf preconditioners)
  - High level interface for grid-based problems
  - ILU (Euclid/pilut), multigrid (PFMG/BoomerAMG), sparse (parasails)
  - https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html

PETSc preconditioners

PETSc preconditioners

- Internal

PETSc preconditioners

- Internal
  - Jacobi, SOR

PETSc preconditioners

- Internal
  - Jacobi, SOR
  - ILU(k), ICC(k) (sequential only)

PETSc preconditioners

- Internal
  - Jacobi, SOR
  - ILU(k), ICC(k) (sequential only)
- External

PETSc preconditioners

- Internal
  - Jacobi, SOR
  - ILU(k), ICC(k) (sequential only)
- External
  - Hypre

PETSc preconditioners

- Internal
  - Jacobi, SOR
  - ILU(k), ICC(k) (sequential only)
- External
  - Hypre
  - Prometheus (http://www.columbia.edu/~ma2325/prom_intro.html)

PETSc preconditioners

- Internal
  - Jacobi, SOR
  - ILU(k), ICC(k) (sequential only)
- External
  - Hypre
  - Prometheus (http://www.columbia.edu/~ma2325/prom_intro.html)
  - SPAI

PETSc preconditioners

- Internal
  - Jacobi, SOR
  - ILU(k), ICC(k) (sequential only)
- External
  - Hypre
  - Prometheus (http://www.columbia.edu/~ma2325/prom_intro.html)
  - SPAI
  - ML

Eigen- and Singular values/vectors

Eigen- and Singular values/vectors

- Formulations:

Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real:

Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real:
  - Symmetric, real:

Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real:
  - Symmetric, real:

Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real:
  - Symmetric, real:
  - Non-symmetric, non-defective:

Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real:
  - Symmetric, real: $Ax = \lambda x \Rightarrow Q^T AQ = diag(\lambda_1,..,\lambda_n)$
  - Non-symmetric, non-defective:

Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real:
  - Symmetric, real: $Ax = \lambda x \Rightarrow Q^T AQ = diag(\lambda_1,..,\lambda_n)$
  - Non-symmetric, non-defective:
  - Generalized, symm:

Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real:
  - Symmetric, real: $Ax = \lambda x \Rightarrow Q^T AQ = diag(\lambda_1,..,\lambda_n)$
  - Non-symmetric, non-defective: $Ax = \lambda x \Rightarrow X^{-1}AX = diag(\lambda_1,..,\lambda_n)$
  - Generalized, symm:

Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real:
  - Symmetric, real: $Ax = \lambda x \Rightarrow Q^T AQ = diag(\lambda_1,..,\lambda_n)$
  - Non-symmetric, non-defective: $Ax = \lambda x \Rightarrow X^{-1}AX = diag(\lambda_1,..,\lambda_n)$
  - Generalized, symm:
  - Singular values:

Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real:
  - Symmetric, real: $Ax = \lambda x \Rightarrow Q^T AQ = diag(\lambda_1,..,\lambda_n)$
  - Non-symmetric, non-defective: $Ax = \lambda x \Rightarrow X^{-1}AX = diag(\lambda_1,..,\lambda_n)$
  - Generalized, symm: $Ax = \lambda Bx \Rightarrow Q^T AQ = diag(a_1,..,a_n) \quad Q^T BQ = diag(b_1,..,b_n)$
  - Singular values:

Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real:
  - Symmetric, real: $Ax = \lambda x \Rightarrow Q^T AQ = diag(\lambda_1,..,\lambda_n)$
  - Non-symmetric, non-defective: $Ax = \lambda x \Rightarrow X^{-1}AX = diag(\lambda_1,..,\lambda_n)$
  - Generalized, symm: $Ax = \lambda Bx \Rightarrow Q^T AQ = diag(a_1,..,a_n) \quad Q^T BQ = diag(b_1,..,b_n)$
  - Singular values:
- Interpretation: eigenvectors of A when multiplied by A are parallel to themselves

Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real:
  - Symmetric, real: $Ax = \lambda x \Rightarrow Q^T AQ = diag(\lambda_1,..,\lambda_n)$
  - Non-symmetric, non-defective: $Ax = \lambda x \Rightarrow X^{-1}AX = diag(\lambda_1,..,\lambda_n)$
  - Generalized, symm: $Ax = \lambda Bx \Rightarrow Q^T AQ = diag(a_1,..,a_n) \quad Q^T BQ = diag(b_1,..,b_n)$
  - Singular values: $U^T AV = diag(\sigma_1,..,\sigma_n)$
- Interpretation: eigenvectors of A when multiplied by A are parallel to themselves

Eigen- and Singular values/vectors

- Formulations:
  - Non-symmetric, real:
  - Symmetric, real: $Ax = \lambda x \Rightarrow Q^T AQ = diag(\lambda_1,..,\lambda_n)$
  - Non-symmetric, non-defective: $Ax = \lambda x \Rightarrow X^{-1}AX = diag(\lambda_1,..,\lambda_n)$
  - Generalized, symm: $Ax = \lambda Bx \Rightarrow Q^T AQ = diag(a_1,..,a_n) \quad Q^T BQ = diag(b_1,..,b_n)$
  - Singular values: $U^T AV = diag(\sigma_1,..,\sigma_n)$
- Interpretation: eigenvectors of A when multiplied by A are parallel to themselves
- Applications in numerous fields

Eigen- and singular value solvers, dense matrices

Eigen- and singular value solvers, dense matrices

- ScaLAPACK

Eigen- and singular value solvers, dense matrices

- ScaLAPACK
  - PSSYTRD /PDSYTRD: Reduction of a symmetric matrix to tridiagonal form

Eigen- and singular value solvers, dense matrices

- ScaLAPACK
  - PSSYTRD /PDSYTRD: Reduction of a symmetric matrix to tridiagonal form
  - PSGEBRD /PDGEBRD: Reduction of a rectangular matrix to bidiagonal form to compute a singular value decomposition:

Eigen- and singular value solvers, dense matrices

- ScaLAPACK
  - PSSYTRD /PDSYTRD: Reduction of a symmetric matrix to tridiagonal form
  - PSGEBRD /PDGEBRD: Reduction of a rectangular matrix to bidiagonal form to compute a singular value decomposition:
  - PSGEHRD /PDGEHRD: Reduction of a nonsymmetric matrix to Hessenberg form to solve a nonsymmetric eigenvalue problem

Eigen- and singular value solvers, dense matrices

- ScaLAPACK
  - PSSYTRD /PDSYTRD: Reduction of a symmetric matrix to tridiagonal form
  - PSGEBRD /PDGEBRD: Reduction of a rectangular matrix to bidiagonal form to compute a singular value decomposition:
  - PSGEHRD /PDGEHRD: Reduction of a nonsymmetric matrix to Hessenberg form to solve a nonsymmetric eigenvalue problem
- MAGMA / PLASMA

Eigen- and singular value solvers, dense matrices

- ScaLAPACK
  - PSSYTRD /PDSYTRD: Reduction of a symmetric matrix to tridiagonal form
  - PSGEBRD /PDGEBRD: Reduction of a rectangular matrix to bidiagonal form to compute a singular value decomposition:
  - PSGEHRD /PDGEHRD: Reduction of a nonsymmetric matrix to Hessenberg form to solve a nonsymmetric eigenvalue problem
- MAGMA / PLASMA
  - Under development! Please let us know your needs

Eigenvalues/vectors of large, sparse matrices

Eigenvalues/vectors of large, sparse matrices

- Techniques based on Lanczos iteration (symm. A)

Eigenvalues/vectors of large, sparse matrices

- Techniques based on Lanczos iteration (symm. A)

Eigenvalues/vectors of large, sparse matrices

- Techniques based on Lanczos iteration (symm. A)

$$r_0 = q_1; \quad \beta_0 = 1; \quad q_0 = 0; \quad j = 0$$

$$\textit{while} \quad \beta_j \neq 0 \quad \{$$

$$q_{j+1} = r_j / \beta_j; \quad j = j + 1; \quad \alpha_j = q_j^T A q_j$$

$$r_j = (A - \alpha_j I) q_j - \beta_{j-1} q_{j-1}; \quad \beta_j = \left\| r_j \right\|_2$$

$$\}$$

Eigenvalues/vectors of large, sparse matrices

- Techniques based on Lanczos iteration (symm. A)

$$r_0 = q_1; \quad \beta_0 = 1; \quad q_0 = 0; \quad j = 0$$

$$\textit{while} \quad \beta_j \neq 0 \quad \{$$

$$q_{j+1} = r_j / \beta_j; \quad j = j + 1; \quad \alpha_j = q_j^T A q_j$$

$$r_j = (A - \alpha_j I) q_j - \beta_{j-1} q_{j-1}; \quad \beta_j = \left\| r_j \right\|_2$$

$$\}$$

$$q_j$$

Eigenvalues/vectors of large, sparse matrices

- Techniques based on Lanczos iteration (symm. A)

$$r_0 = q_1; \quad \beta_0 = 1; \quad q_0 = 0; \quad j = 0$$

$$\textit{while} \quad \beta_j \neq 0 \quad \{$$

$$q_{j+1} = r_j / \beta_j; \quad j = j + 1; \quad \alpha_j = q_j^T A q_j$$

$$r_j = (A - \alpha_j I) q_j - \beta_{j-1} q_{j-1}; \quad \beta_j = \left\| r_j \right\|_2$$

$$\}$$

q_j

α_j β_j

Eigenvalues/vectors of large, sparse matrices

- Techniques based on Lanczos iteration (symm. A)

$$r_0 = q_1; \quad \beta_0 = 1; \quad q_0 = 0; \quad j = 0$$

$$\textit{while} \quad \beta_j \neq 0 \quad \{$$

$$q_{j+1} = r_j / \beta_j; \quad j = j + 1; \quad \alpha_j = q_j^T A q_j$$

$$r_j = (A - \alpha_j I) q_j - \beta_{j-1} q_{j-1}; \quad \beta_j = \left\| r_j \right\|_2$$

$$\}$$

q_j

α_j β_j

Eigenvalues/vectors of large, sparse matrices

- Techniques based on Lanczos iteration (symm. A)

$$r_0 = q_1; \quad \beta_0 = 1; \quad q_0 = 0; \quad j = 0$$

$$\textit{while} \quad \beta_j \neq 0 \quad \{$$

$$q_{j+1} = r_j / \beta_j; \quad j = j + 1; \quad \alpha_j = q_j^T A q_j$$

$$r_j = (A - \alpha_j I) q_j - \beta_{j-1} q_{j-1}; \quad \beta_j = \left\| r_j \right\|_2$$

$$\}$$

q_j

α_j β_j

Eigenvalues/vectors of large, sparse matrices

- Techniques based on Lanczos iteration (symm. A)

$$r_0 = q_1; \quad \beta_0 = 1; \quad q_0 = 0; \quad j = 0$$

$$\textit{while} \quad \beta_j \neq 0 \quad \{$$

$$q_{j+1} = r_j / \beta_j; \quad j = j + 1; \quad \alpha_j = q_j^T A q_j$$

$$r_j = (A - \alpha_j I) q_j - \beta_{j-1} q_{j-1}; \quad \beta_j = \left\| r_j \right\|_2$$

$$\}$$

- Lanczos vectors: q_j

α_j β_j

Eigenvalues/vectors of large, sparse matrices

- Techniques based on Lanczos iteration (symm. A)

$$r_0 = q_1; \quad \beta_0 = 1; \quad q_0 = 0; \quad j = 0$$

$$while \quad \beta_j \neq 0 \quad \{$$

$$q_{j+1} = r_j / \beta_j; \quad j = j+1; \quad \alpha_j = q_j^T A q_j$$

$$r_j = (A - \alpha_j I) q_j - \beta_{j-1} q_{j-1}; \quad \beta_j = \left\| r_j \right\|_2$$

$$\}$$

- Lanczos vectors: q_j
- Form tridiagonal matrix T: diagonal α_j subdiagonal β_j

Eigenvalues/vectors of large, sparse matrices

- Techniques based on Lanczos iteration (symm. A)

$$r_0 = q_1; \quad \beta_0 = 1; \quad q_0 = 0; \quad j = 0$$

$$\textit{while} \quad \beta_j \neq 0 \quad \{$$

$$q_{j+1} = r_j / \beta_j; \quad j = j+1; \quad \alpha_j = q_j^T A q_j$$

$$r_j = (A - \alpha_j I) q_j - \beta_{j-1} q_{j-1}; \quad \beta_j = \left\| r_j \right\|_2$$

$$\}$$

- Lanczos vectors: q_j
- Form tridiagonal matrix T: diagonal α_j subdiagonal β_j
- Diagonalization of T is stable iterative procedure

Nonsymmetric matrices: Arnoldi method, 1951

Nonsymmetric matrices: Arnoldi method, 1951

- Similar to Lanczos but result is upper Hessenberg

Nonsymmetric matrices: Arnoldi method, 1951

- Similar to Lanczos but result is upper Hessenberg
- Stable iterative procedure to transform to upper triangular form; Ritz values on diagonal

Nonsymmetric matrices: Arnoldi method, 1951

- Similar to Lanczos but result is upper Hessenberg
- Stable iterative procedure to transform to upper triangular form; Ritz values on diagonal

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{pmatrix}$$

Nonsymmetric matrices: Arnoldi method, 1951

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{pmatrix}$$

- Similar to Lanczos but result is upper Hessenberg
- Stable iterative procedure to transform to upper triangular form; Ritz values on diagonal
- Issues: loss of orthogonality in , poor choice of

Nonsymmetric matrices: Arnoldi method, 1951

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{pmatrix}$$

- Similar to Lanczos but result is upper Hessenberg
- Stable iterative procedure to transform to upper triangular form; Ritz values on diagonal
- Issues: loss of orthogonality in q_j , poor choice of q_1
- ARPACK (Arnoldi Package, 1992), F77

Nonsymmetric matrices: Arnoldi method, 1951

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{pmatrix}$$

- Similar to Lanczos but result is upper Hessenberg
- Stable iterative procedure to transform to upper triangular form; Ritz values on diagonal
- Issues: loss of orthogonality in q_j , poor choice of q_1
- ARPACK (Arnoldi Package, 1992), F77
  - Implicitly restarted algorithm, generalized eigenvalue problem

Nonsymmetric matrices: Arnoldi method, 1951

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{pmatrix}$$

- Similar to Lanczos but result is upper Hessenberg
- Stable iterative procedure to transform to upper triangular form; Ritz values on diagonal
- Issues: loss of orthogonality in q_j , poor choice of q_1
- ARPACK (Arnoldi Package, 1992), F77
  - Implicitly restarted algorithm, generalized eigenvalue problem
  - Calculates a few eigenvalues and corresponding eigenvectors

Nonsymmetric matrices: Arnoldi method, 1951

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{pmatrix}$$

- Similar to Lanczos but result is upper Hessenberg
- Stable iterative procedure to transform to upper triangular form; Ritz values on diagonal
- Issues: loss of orthogonality in q_j , poor choice of q_1
- ARPACK (Arnoldi Package, 1992), F77
  - Implicitly restarted algorithm, generalized eigenvalue problem
  - Calculates a few eigenvalues and corresponding eigenvectors
  - Call-back mechanism to request Ax operation, preconditioner

Nonsymmetric matrices: Arnoldi method, 1951

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{pmatrix}$$

- Similar to Lanczos but result is upper Hessenberg
- Stable iterative procedure to transform to upper triangular form; Ritz values on diagonal
- Issues: loss of orthogonality in q_j , poor choice of q_1
- ARPACK (Arnoldi Package, 1992), F77
  - Implicitly restarted algorithm, generalized eigenvalue problem
  - Calculates a few eigenvalues and corresponding eigenvectors
  - Call-back mechanism to request Ax operation, preconditioner
  - P_ARPACK (1996), ARPACK++ (1998)

Other eigen-solver libraries, nonsymmetric matrices

Other eigen-solver libraries, nonsymmetric matrices

- IETL (Iterative Eigensolver Template Library):

Other eigen-solver libraries, nonsymmetric matrices

- IETL (Iterative Eigensolver Template Library):
  - C++ with templates; four methods for hermitian matrices

Other eigen-solver libraries, nonsymmetric matrices

- IETL (Iterative Eigensolver Template Library):
  - C++ with templates; four methods for hermitian matrices
  - Sequential only! http://www.comp-phys.org/software/ietl/

Other eigen-solver libraries, nonsymmetric matrices

- IETL (Iterative Eigensolver Template Library):
  - C++ with templates; four methods for hermitian matrices
  - Sequential only! http://www.comp-phys.org/software/ietl/
- PRIMME (PReconditioned Iterative MultiMethod Eigensolve):

Other eigen-solver libraries, nonsymmetric matrices

- IETL (Iterative Eigensolver Template Library):
  - C++ with templates; four methods for hermitian matrices
  - Sequential only! http://www.comp-phys.org/software/ietl/
- PRIMME (PReconditioned Iterative MultiMethod Eigensolve):
  - Parallel version for double precision but not complex double

Other eigen-solver libraries, nonsymmetric matrices

- IETL (Iterative Eigensolver Template Library):
  - C++ with templates; four methods for hermitian matrices
  - Sequential only! http://www.comp-phys.org/software/ietl/
- PRIMME (PReconditioned Iterative MultiMethod Eigensolve):
  - Parallel version for double precision but not complex double
  - http://www.cs.wm.edu/~andreas/software/

Other eigen-solver libraries, nonsymmetric matrices

- IETL (Iterative Eigensolver Template Library):
  - C++ with templates; four methods for hermitian matrices
  - Sequential only! http://www.comp-phys.org/software/ietl/
- PRIMME (PReconditioned Iterative MultiMethod Eigensolve):
  - Parallel version for double precision but not complex double
  - http://www.cs.wm.edu/~andreas/software/
- SLEPc (Scalable Library for Eigenvalue Problem computations):

Other eigen-solver libraries, nonsymmetric matrices

- IETL (Iterative Eigensolver Template Library):
  - C++ with templates; four methods for hermitian matrices
  - Sequential only! http://www.comp-phys.org/software/ietl/
- PRIMME (PReconditioned Iterative MultiMethod Eigensolve):
  - Parallel version for double precision but not complex double
  - http://www.cs.wm.edu/~andreas/software/
- SLEPc (Scalable Library for Eigenvalue Problem computations):
  - (non-)hermitian matrices; generalized EVP; partial SVD

Other eigen-solver libraries, nonsymmetric matrices

- IETL (Iterative Eigensolver Template Library):
  - C++ with templates; four methods for hermitian matrices
  - Sequential only! http://www.comp-phys.org/software/ietl/
- PRIMME (PReconditioned Iterative MultiMethod Eigensolve):
  - Parallel version for double precision but not complex double
  - http://www.cs.wm.edu/~andreas/software/
- SLEPc (Scalable Library for Eigenvalue Problem computations):
  - (non-)hermitian matrices; generalized EVP; partial SVD
  - Extension to PETSc

Other eigen-solver libraries, nonsymmetric matrices

- IETL (Iterative Eigensolver Template Library):
  - C++ with templates; four methods for hermitian matrices
  - Sequential only! http://www.comp-phys.org/software/ietl/
- PRIMME (PReconditioned Iterative MultiMethod Eigensolve):
  - Parallel version for double precision but not complex double
  - http://www.cs.wm.edu/~andreas/software/
- SLEPc (Scalable Library for Eigenvalue Problem computations):
  - (non-)hermitian matrices; generalized EVP; partial SVD
  - Extension to PETSc
  - http://www.grycap.upv.es/slepc/

Other eigen-solver libraries, nonsymmetric matrices

- IETL (Iterative Eigensolver Template Library):
  - C++ with templates; four methods for hermitian matrices
  - Sequential only! http://www.comp-phys.org/software/ietl/
- PRIMME (PReconditioned Iterative MultiMethod Eigensolve):
  - Parallel version for double precision but not complex double
  - http://www.cs.wm.edu/~andreas/software/
- SLEPc (Scalable Library for Eigenvalue Problem computations):
  - (non-)hermitian matrices; generalized EVP; partial SVD
  - Extension to PETSc
  - http://www.grycap.upv.es/slepc/
- Anasazi: http://trilinos.sandia.gov/packages/anasazi/

Graph partitioning / re-partitioning

Graph partitioning / re-partitioning

- Given n vertices joined by e edges (directed or undirected), find the partition into p groups which nearly minimizes a metric relating to the cut edges

Graph partitioning / re-partitioning

- Given n vertices joined by e edges (directed or undirected), find the partition into p groups which nearly minimizes a metric relating to the cut edges
- Application areas:

Graph partitioning / re-partitioning

- Given n vertices joined by e edges (directed or undirected), find the partition into p groups which nearly minimizes a metric relating to the cut edges
- Application areas:
  - Partition mesh on p processes

Graph partitioning / re-partitioning

- Given n vertices joined by e edges (directed or undirected), find the partition into p groups which nearly minimizes a metric relating to the cut edges
- Application areas:
  - Partition mesh on p processes
  - Fill reducing orderings for sparse matrix decompositions

Graph partitioning / re-partitioning

- Given n vertices joined by e edges (directed or undirected), find the partition into p groups which nearly minimizes a metric relating to the cut edges
- Application areas:
  - Partition mesh on p processes
  - Fill reducing orderings for sparse matrix decompositions
  - Some optimization problems can be formulated as a graph

Graph partitioning / re-partitioning

- Given n vertices joined by e edges (directed or undirected), find the partition into p groups which nearly minimizes a metric relating to the cut edges
- Application areas:
  - Partition mesh on p processes
  - Fill reducing orderings for sparse matrix decompositions
  - Some optimization problems can be formulated as a graph

Graph partitioning facets

Dynamic Load Balancing

Graph Coloring

Matrix Ordering

Unstructured Communication

Distributed Data Directories

A	B	C	D	E	F	G	H	I
0	1	0	2	1	0	1	2	1

Parallel graph (re-)partitioning libraries

Parallel graph (re-)partitioning libraries

- METIS / ParMETIS:
  - http://glaros.dtc.umn.edu/gkhome/metis
- Jostle / PJostle (Networks):
  - http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/
- Zoltan
  - http://www.cs.sandia.gov/Zoltan/
- SCOTCH / PT-SCOTCH
  - http://www.labri.fr/perso/pelegrin/scotch/

Non-linear and other problems

Non-linear and other problems

- Non-linear ordinary differential equations

Non-linear and other problems

- Non-linear ordinary differential equations
- Optimization problems

Non-linear and other problems

- Non-linear ordinary differential equations
- Optimization problems
- Differential-algebraic equations

Non-linear and other problems

- Non-linear ordinary differential equations
- Optimization problems
- Differential-algebraic equations
- Non-linear algebraic systems

MOOCHO (Trilinos): Optimization

MOOCHO (Trilinos): Optimization

- Multi-functional Object-Oriented arCHitecture for Optimization

MOOCHO (Trilinos): Optimization

- Multi-functional Object-Oriented arCHitecture for Optimization
- Minimize subject to

MOOCHO (Trilinos): Optimization

- Multi-functional Object-Oriented arCHitecture for Optimization
- Minimize $f(x_D, x_I)$ subject to $c(x_D, x_I) = 0$ $x_D^{lower} \le x_D \le x_D^{upper}$ $x_I^{lower} \le x_I \le x_I^{upper}$
- A priori partitioned into dependent state variables and independent optimization variables

MOOCHO (Trilinos): Optimization

- Multi-functional Object-Oriented arCHitecture for Optimization
- Minimize $f(x_D, x_I)$ subject to $c(x_D, x_I) = 0 \quad x_D^{lower} \le x_D \le x_D^{upper} \quad x_I^{lower} \le x_I \le x_I^{upper}$
- A priori partitioned into dependent state variables x_D and independent optimization variables x_I
- Can utilize parallel direct solvers (Amesos), preconditioners (Ifpack,ML), Krylov methods (AztecOO, Belos)

MOOCHO (Trilinos): Optimization

- Multi-functional Object-Oriented arCHitecture for Optimization
- Minimize $f(x_D, x_I)$ subject to $c(x_D, x_I) = 0 \quad x_D^{lower} \le x_D \le x_D^{upper} \quad x_I^{lower} \le x_I \le x_I^{upper}$
- A priori partitioned into dependent state variables x_D and independent optimization variables x_I
- Can utilize parallel direct solvers (Amesos), preconditioners (Ifpack,ML), Krylov methods (AztecOO, Belos)
- http://trilinos.sandia.gov/packages/moocho/

NOX (Trilinos): non-linear equations

NOX (Trilinos): non-linear equations

- Solve with and

NOX (Trilinos): non-linear equations

- Solve $F(x) = 0$ with $F(x) = \begin{pmatrix} f_1(x_1, \cdots, x_n) \\ \vdots \\ f_n(x_1, \cdots, x_n) \end{pmatrix}$ and $J_{i,j} = \frac{\partial F_i}{\partial x_j}(x)$
- User supplies:

NOX (Trilinos): non-linear equations

- Solve $F(x) = 0$ with $F(x) = \begin{pmatrix} f_1(x_1, \cdots, x_n) \\ \vdots \\ f_n(x_1, \cdots, x_n) \end{pmatrix}$ and $J_{i,j} = \frac{\partial F_i}{\partial x_j}(x)$
- User supplies:
  - Function F(x) evaluation

NOX (Trilinos): non-linear equations

- Solve $F(x) = 0$ with $F(x) = \begin{pmatrix} f_1(x_1, \cdots, x_n) \\ \vdots \\ f_n(x_1, \cdots, x_n) \end{pmatrix}$ and $J_{i,j} = \frac{\partial F_i}{\partial x_j}(x)$
- User supplies:
  - Function F(x) evaluation
  - Optional: Jacobian evaluation, preconditioner

NOX (Trilinos): non-linear equations

- Solve $F(x)=0$ with $F(x)=\begin{pmatrix} f_1(x_1,\cdots,x_n) \\ \vdots \\ f_n(x_1,\cdots,x_n) \end{pmatrix}$ and $J_{i,j}=\frac{\partial F_i}{\partial x_j}(x)$
- User supplies:
  - Function F(x) evaluation
  - Optional: Jacobian evaluation, preconditioner
- With good guess, convergence quadratic

NOX (Trilinos): non-linear equations

- Solve $F(x)=0$ with $F(x)=\begin{pmatrix} f_1(x_1,\cdots,x_n) \\ \vdots \\ f_n(x_1,\cdots,x_n) \end{pmatrix}$ and $J_{i,j}=\frac{\partial F_i}{\partial x_j}(x)$
- User supplies:
  - Function F(x) evaluation
  - Optional: Jacobian evaluation, preconditioner
- With good guess, convergence quadratic
- Heuristics used to improve first guess

NOX (Trilinos): non-linear equations

- Solve $F(x)=0$ with $F(x)=\begin{pmatrix} f_1(x_1,\cdots,x_n) \\ \vdots \\ f_n(x_1,\cdots,x_n) \end{pmatrix}$ and $J_{i,j}=\frac{\partial F_i}{\partial x_j}(x)$
- User supplies:
  - Function F(x) evaluation
  - Optional: Jacobian evaluation, preconditioner
- With good guess, convergence quadratic
- Heuristics used to improve first guess
- PETSc interface available

NOX (Trilinos): non-linear equations

- Solve $F(x)=0$ with $F(x)=\begin{pmatrix} f_1(x_1,\cdots,x_n) \\ \vdots \\ f_n(x_1,\cdots,x_n) \end{pmatrix}$ and $J_{i,j}=\frac{\partial F_i}{\partial x_j}(x)$
- User supplies:
  - Function F(x) evaluation
  - Optional: Jacobian evaluation, preconditioner
- With good guess, convergence quadratic
- Heuristics used to improve first guess
- PETSc interface available
- http://trilinos.sandia.gov/packages/nox/

SUNDIALS: parallel ODE solvers

SUNDIALS: parallel ODE solvers

- Contains various stiff ordinary differential equation solvers for initial value problems

SUNDIALS: parallel ODE solvers

- Contains various stiff ordinary differential equation solvers for initial value problems
  - CVODE

SUNDIALS: parallel ODE solvers

- Contains various stiff ordinary differential equation solvers for initial value problems
  - CVODE $y' = f(t,y)$
  - CVODES

SUNDIALS: parallel ODE solvers

- Contains various stiff ordinary differential equation solvers for initial value problems
  - CVODE $y' = f(t,y)$
  - CVODES $y' = f(t,y,p)$
  - IDA

SUNDIALS: parallel ODE solvers

- Contains various stiff ordinary differential equation solvers for initial value problems
  - CVODE $y' = f(t,y)$
  - CVODES $y' = f(t,y,p)$
  - IDA $F(t,y,y') = 0$
  - IDAS

SUNDIALS: parallel ODE solvers

- Contains various stiff ordinary differential equation solvers for initial value problems
  - CVODE $y' = f(t,y)$
  - CVODES $y' = f(t,y,p)$
  - IDA $F(t,y,y') = 0$
  - IDAS $F(t,y,y',p) = 0$
  - KINSOL: nonlinear algebraic systems (Newton-Krylov)

SUNDIALS: parallel ODE solvers

- Contains various stiff ordinary differential equation solvers for initial value problems
  - CVODE $y' = f(t,y)$
  - CVODES $y' = f(t,y,p)$
  - IDA $F(t,y,y') = 0$
  - IDAS $F(t,y,y',p) = 0$
  - KINSOL: nonlinear algebraic systems (Newton-Krylov)
- Sequential and (reduced) parallel functionality

SUNDIALS: parallel ODE solvers

- Contains various stiff ordinary differential equation solvers for initial value problems
  - CVODE $y' = f(t,y)$
  - CVODES $y' = f(t,y,p)$
  - IDA $F(t,y,y') = 0$
  - IDAS $F(t,y,y',p) = 0$
  - KINSOL: nonlinear algebraic systems (Newton-Krylov)
- Sequential and (reduced) parallel functionality
- https://computation.llnl.gov/casc/sundials/

Epetra: linear algebra primitives

Epetra: linear algebra primitives

Epetra contents:

Epetra: linear algebra primitives

Epetra contents:

- *Serial* and *distributed matrix* and *(multi-)vector* objects (various types)

Epetra: linear algebra primitives

Epetra contents:

- *Serial* and *distributed matrix* and *(multi-)vector* objects (various types)
- Operator objects to avoid explicit matrix construction

Epetra: linear algebra primitives

Epetra contents:

- *Serial* and *distributed matrix* and *(multi-)vector* objects (various types)
- Operator objects to avoid explicit matrix construction
- Map object of global to local indices for distributed case

Epetra: linear algebra primitives

Epetra contents:

- *Serial* and *distributed matrix* and *(multi-)vector* objects (various types)
- Operator objects to avoid explicit matrix construction
- Map object of global to local indices for distributed case
- Communicator object

Epetra: linear algebra primitives

Epetra contents:

- *Serial* and *distributed matrix* and *(multi-)vector* objects (various types)
- Operator objects to avoid explicit matrix construction
- Map object of global to local indices for distributed case
- Communicator object
- Import/Export objects for off-core communication

Epetra: linear algebra primitives

Epetra contents:

- *Serial* and *distributed matrix* and *(multi-)vector* objects (various types)
- Operator objects to avoid explicit matrix construction
- Map object of global to local indices for distributed case
- Communicator object
- Import/Export objects for off-core communication
- Time, Flop and other utilities

Epetra: serial and distributed objects

Epetra: serial and distributed objects

- Serial vectors are generally short, dense, and not partitioned; each process manages its own

Epetra: serial and distributed objects

- Serial vectors are generally short, dense, and not partitioned; each process manages its own
- Serial matrices are small, dense, and process local

Epetra: serial and distributed objects

- Serial vectors are generally short, dense, and not partitioned; each process manages its own
- Serial matrices are small, dense, and process local
- `Epetra_LAPACK` class: thin layer on top of LAPACK

Epetra: serial and distributed objects

- Serial vectors are generally short, dense, and not partitioned; each process manages its own
- Serial matrices are small, dense, and process local
- `Epetra_LAPACK` class: thin layer on top of LAPACK
- Distributed vectors and matrices require a `Epetra_map` defines the global-to-local mapping

Epetra: serial and distributed objects

- Serial vectors are generally short, dense, and not partitioned; each process manages its own
- Serial matrices are small, dense, and process local
- `Epetra_LAPACK` class: thin layer on top of LAPACK
- Distributed vectors and matrices require a `Epetra_map` defines the global-to-local mapping
- Multiple ways to define maps; some opaque (best balance algorithm), others ‘hands on’

Epetra: serial and distributed objects

- Serial vectors are generally short, dense, and not partitioned; each process manages its own
- Serial matrices are small, dense, and process local
- `Epetra_LAPACK` class: thin layer on top of LAPACK
- Distributed vectors and matrices require a `Epetra_map` defines the global-to-local mapping
- Multiple ways to define maps; some opaque (best balance algorithm), others 'hands on'
- `Epetra_Import` and `Epetra_Export` transfer data between two maps

Epetra: distributed matrices

Epetra: distributed matrices

- Virtual class: `Epetra_RowMatrix`; derived classes:

Epetra: distributed matrices

- Virtual class: `Epetra_RowMatrix`; derived classes:
  - `Epetra_CrsMatrix`: compressed row storage

Epetra: distributed matrices

- Virtual class: `Epetra_RowMatrix`; derived classes:
  - `Epetra_CrsMatrix`: compressed row storage
  - `Epetra_VbrMatrix`: matrices with block structure, may have differently sized blocks

Epetra: distributed matrices

- Virtual class: `Epetra_RowMatrix`; derived classes:
  - `Epetra_CrsMatrix`: compressed row storage
  - `Epetra_VbrMatrix`: matrices with block structure, may have differently sized blocks
  - `Epetra_FECrsMatrix` / `Epetra_FEVbrMatrix`: matrices arising from finite element discretizations

Epetra: distributed matrices

- Virtual class: `Epetra_RowMatrix`; derived classes:
  - `Epetra_CrsMatrix`: compressed row storage
  - `Epetra_VbrMatrix`: matrices with block structure, may have differently sized blocks
  - `Epetra_FECrsMatrix` / `Epetra_FEVbrMatrix`: matrices arising from finite element discretizations
- Strategies to define a sparse matrix

Epetra: distributed matrices

- Virtual class: `Epetra_RowMatrix`; derived classes:
  - `Epetra_CrsMatrix`: compressed row storage
  - `Epetra_VbrMatrix`: matrices with block structure, may have differently sized blocks
  - `Epetra_FECrsMatrix` / `Epetra_FEVbrMatrix`: matrices arising from finite element discretizations
- Strategies to define a sparse matrix
  - *Suggest* number of non-zeros per row

Epetra: distributed matrices

- Virtual class: `Epetra_RowMatrix`; derived classes:
  - `Epetra_CrsMatrix`: compressed row storage
  - `Epetra_VbrMatrix`: matrices with block structure, may have differently sized blocks
  - `Epetra_FECrsMatrix` / `Epetra_FEVbrMatrix`: matrices arising from finite element discretizations
- Strategies to define a sparse matrix
  - *Suggest* number of non-zeros per row
  - Specify the global indices on each process

Epetra: defining a sparse matrix

```
int NumGlobalElements=50
Epetra_Map Map(NumGlobal;Elements,0,Comm);
int NumMyElements = Map.NumMyElements();
int * MyGlobalElements = Map.MyGlobalElements();
int * NumNz = new int[NumMyElements];
for( int i=0 ; i<NumMyElements ; i++ )
  if( MyGlobalElements[i]==0 || MyGlobalElements[i] == NumGlobalElements-1)
    NumNz[i] = 2;
  else
    NumNz[i] = 3;

Epetra_CrsMatrix A(Copy,Map,NumNz)
// At this point there are two nonzeros in first and last row
// Otherwise three nonzeros per row
```

Epetra: filling in matrix structure and values

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
for( int i=0 ; i<NumMyElements; ++I ){
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
for( int i=0 ; i<NumMyElements; ++I ){
  if (MyGlobalElements[i]==0) { Indices[0] = 1; NumEntries = 1; }
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
for( int i=0 ; i<NumMyElements; ++I ){
  if (MyGlobalElements[i]==0) { Indices[0] = 1; NumEntries = 1; }
  else if (MyGlobalElements[i] == NumGlobalElements-1) {
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
for( int i=0 ; i<NumMyElements; ++I ){
  if (MyGlobalElements[i]==0) { Indices[0] = 1; NumEntries = 1; }
  else if (MyGlobalElements[i] == NumGlobalElements-1) {
    Indices[0] = NumGlobalElements-2; NumEntries = 1; }
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
for( int i=0 ; i<NumMyElements; ++I ){
  if (MyGlobalElements[i]==0) { Indices[0] = 1; NumEntries = 1; }
  else if (MyGlobalElements[i] == NumGlobalElements-1) {
    Indices[0] = NumGlobalElements-2; NumEntries = 1; }
  else {
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
for( int i=0 ; i<NumMyElements; ++I ){
  if (MyGlobalElements[i]==0) { Indices[0] = 1; NumEntries = 1; }
  else if (MyGlobalElements[i] == NumGlobalElements-1) {
    Indices[0] = NumGlobalElements-2; NumEntries = 1; }
  else {
    Indices[0] = MyGlobalElements[i]-1;
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
for( int i=0 ; i<NumMyElements; ++I ){
  if (MyGlobalElements[i]==0) { Indices[0] = 1; NumEntries = 1; }
  else if (MyGlobalElements[i] == NumGlobalElements-1) {
    Indices[0] = NumGlobalElements-2; NumEntries = 1; }
  else {
    Indices[0] = MyGlobalElements[i]-1;
    Indices[1]=MyGlobalElements[i}+1; NumEntries = 2; }
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
for( int i=0 ; i<NumMyElements; ++I ){
  if (MyGlobalElements[i]==0) { Indices[0] = 1; NumEntries = 1; }
  else if (MyGlobalElements[i] == NumGlobalElements-1) {
    Indices[0] = NumGlobalElements-2; NumEntries = 1; }
  else {
    Indices[0] = MyGlobalElements[i]-1;
    Indices[1]=MyGlobalElements[i}+1; NumEntries = 2; }
  }
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
for( int i=0 ; i<NumMyElements; ++I ){
  if (MyGlobalElements[i]==0) { Indices[0] = 1; NumEntries = 1; }
  else if (MyGlobalElements[i] == NumGlobalElements-1) {
    Indices[0] = NumGlobalElements-2; NumEntries = 1; }
  else {
    Indices[0] = MyGlobalElements[i]-1;
    Indices[1]=MyGlobalElements[i}+1; NumEntries = 2; }
  }
  A.InsertGlobalValues(MyGlobalElements[i], NumEntries, Values, Indices);
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
for( int i=0 ; i<NumMyElements; ++I ){
  if (MyGlobalElements[i]==0) { Indices[0] = 1; NumEntries = 1; }
  else if (MyGlobalElements[i] == NumGlobalElements-1) {
    Indices[0] = NumGlobalElements-2; NumEntries = 1; }
  else {
    Indices[0] = MyGlobalElements[i]-1;
    Indices[1]=MyGlobalElements[i}+1; NumEntries = 2; }
  }
  A.InsertGlobalValues(MyGlobalElements[i], NumEntries, Values, Indices);
  A.InsertGlobalValues(MyGlobalElements[i], 1, &two, MyGlobalElements+1);
```

Epetra: filling in matrix structure and values

```
double * Values = new double[2];
Values[0] = -1.0; Values[1] = -1.0;
int * Indices = new int[2];
double two = 2.0;
int NumEntries;
for( int i=0 ; i<NumMyElements; ++I ){
  if (MyGlobalElements[i]==0) { Indices[0] = 1; NumEntries = 1; }
  else if (MyGlobalElements[i] == NumGlobalElements-1) {
    Indices[0] = NumGlobalElements-2; NumEntries = 1; }
  else {
    Indices[0] = MyGlobalElements[i]-1;
    Indices[1]=MyGlobalElements[i}+1; NumEntries = 2; }
  }
  A.InsertGlobalValues(MyGlobalElements[i], NumEntries, Values, Indices);
  A.InsertGlobalValues(MyGlobalElements[i], 1, &two, MyGlobalElements+1);
}
```

Epetra: defining a matrix-vector product operator

- Many applications never construct matrix
- Instead define an operator with Epetra_operator

```
class TriDiagonalOperator : public Epetra_operator{
public:  // constructors, methods
private:
  Epetra_Map Map_;
  double diag_minus_one_;  // sub-diagonal value
  double diag_;            // values on diagonal
  double diag_plus_one_;   // super-diagonal value
}
```

Epetra_operator: applying mat-vec product

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
    // first row
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
    // first row
    Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
    // first row
    Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];
    for( int i=1 ; i<Length-1 ; ++I ){  // intermediate rows
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
    // first row
    Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];
    for( int i=1 ; i<Length-1 ; ++I ){  // intermediate rows
      Y[vec][i] = diag_ * X[vec][i] + diag_plus_one_ * X[vec][i+1] +
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
    // first row
    Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];
    for( int i=1 ; i<Length-1 ; ++I ){  // intermediate rows
      Y[vec][i] = diag_ * X[vec][i] + diag_plus_one_ * X[vec][i+1] +
                  diag_minus_one_ * X[vec][i-1];
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
    // first row
    Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];
    for( int i=1 ; i<Length-1 ; ++I ){  // intermediate rows
      Y[vec][i] = diag_ * X[vec][i] + diag_plus_one_ * X[vec][i+1] +
                  diag_minus_one_ * X[vec][i-1];
    }
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
    // first row
    Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];
    for( int i=1 ; i<Length-1 ; ++I ){  // intermediate rows
      Y[vec][i] = diag_ * X[vec][i] + diag_plus_one_ * X[vec][i+1] +
                  diag_minus_one_ * X[vec][i-1];
    }
    // final_row
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
    // first row
    Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];
    for( int i=1 ; i<Length-1 ; ++I ){  // intermediate rows
      Y[vec][i] = diag_ * X[vec][i] + diag_plus_one_ * X[vec][i+1] +
                  diag_minus_one_ * X[vec][i-1];
    }
    // final_row
    Y[vec][Length-1] = diag_*X[vec][Length-1] +
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
    // first row
    Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];
    for( int i=1 ; i<Length-1 ; ++I ){  // intermediate rows
      Y[vec][i] = diag_ * X[vec][i] + diag_plus_one_ * X[vec][i+1] +
                  diag_minus_one_ * X[vec][i-1];
    }
    // final_row
    Y[vec][Length-1] = diag_*X[vec][Length-1] +
                       diag_minus_one_*X[vec][Length-2];
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
    // first row
    Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];
    for( int i=1 ; i<Length-1 ; ++I ){  // intermediate rows
      Y[vec][i] = diag_ * X[vec][i] + diag_plus_one_ * X[vec][i+1] +
                  diag_minus_one_ * X[vec][i-1];
    }
    // final_row
    Y[vec][Length-1] = diag_*X[vec][Length-1] +
                       diag_minus_one_*X[vec][Length-2];
  }
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
    // first row
    Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];
    for( int i=1 ; i<Length-1 ; ++I ){  // intermediate rows
      Y[vec][i] = diag_ * X[vec][i] + diag_plus_one_ * X[vec][i+1] +
                  diag_minus_one_ * X[vec][i-1];
    }
    // final_row
    Y[vec][Length-1] = diag_*X[vec][Length-1] +
                       diag_minus_one_*X[vec][Length-2];
  }
  return true;
```

Epetra_operator: applying mat-vec product

```
int Apply( const Epetra_Multivector & X, Epetra_Multivector & Y) const {
  int Length = X.MyLength();
  for( in vec=0 ; vec<X.NumVectors() ; ++vec ){
    if( Length == 1 ) { Y[vec][0] = diag_ * X[vec][0]; break; }
    // first row
    Y[vec][0] = diag_ * X[vec][0] + diag_plus_one_ * X[vec][1];
    for( int i=1 ; i<Length-1 ; ++I ){  // intermediate rows
      Y[vec][i] = diag_ * X[vec][i] + diag_plus_one_ * X[vec][i+1] +
                  diag_minus_one_ * X[vec][i-1];
    }
    // final_row
    Y[vec][Length-1] = diag_*X[vec][Length-1] +
                       diag_minus_one_*X[vec][Length-2];
  }
  return true;
}
```

Back to eigensolvers

Back to eigensolvers

- ARPACK (F77) is a popular eigen-solver library, with several novel features when released (1992):
  - Reverse communication mechanism allowing user to define mat-vec product, preconditioner, etc.
  - The implicitly-restarted Arnoldi algorithm, efficiently finding a small set of eigenvalues around a given value
- Some limitations:
  - Dependent on one underlying implementation of LA primitives; reverse communication is high maintenance
  - Interfaces were not abstract, revealing underlying complexity
  - Limited to one, albeit successful, method

Anasazi: design objectives

Anasazi: design objectives

- Opaque objects: hide low level complexity

Anasazi: design objectives

- Opaque objects: hide low level complexity
- Flexibility to allow for various linear algebra primitives; ease of incorporation with other frameworks

Anasazi: design objectives

- Opaque objects: hide low level complexity
- Flexibility to allow for various linear algebra primitives; ease of incorporation with other frameworks
- Provide a small set of ‘turn-key’ eigen-solvers for large (sparse) matrices

Anasazi: design objectives

- Opaque objects: hide low level complexity
- Flexibility to allow for various linear algebra primitives; ease of incorporation with other frameworks
- Provide a small set of ‘turn-key’ eigen-solvers for large (sparse) matrices
- Provide a ‘workbench’ framework in which new methods can be implemented

Anasazi: eigen-solver framework

Anasazi: eigen-solver framework

- Utilizes abstract interfaces for operators and multi-vectors

Anasazi: eigen-solver framework

- Utilizes abstract interfaces for operators and multi-vectors
- Specifies what operations the multi-vectors and operators must support

Anasazi: eigen-solver framework

- Utilizes abstract interfaces for operators and multi-vectors
- Specifies what operations the multi-vectors and operators must support
- Access to unterlying object with Anasazi::MultiVecTraits and Anasazi::OperatorTraits

Anasazi: eigen-solver framework

- Utilizes abstract interfaces for operators and multi-vectors
- Specifies what operations the multi-vectors and operators must support
- Access to unterlying object with Anasazi::MultiVecTraits and Anasazi::OperatorTraits
- MultiVecTraits: e.g., MvAddMv, MvDot, MvNorm, …

Anasazi: eigen-solver framework

- Utilizes abstract interfaces for operators and multi-vectors
- Specifies what operations the multi-vectors and operators must support
- Access to unterlying object with Anasazi::MultiVecTraits and Anasazi::OperatorTraits
- MultiVecTraits: e.g., MvAddMv, MvDot, MvNorm, …
- OperatorTraits method:

Anasazi: eigen-solver framework

- Utilizes abstract interfaces for operators and multi-vectors
- Specifies what operations the multi-vectors and operators must support
- Access to unterlying object with Anasazi::MultiVecTraits and Anasazi::OperatorTraits
- MultiVecTraits: e.g., MvAddMv, MvDot, MvNorm, …
- OperatorTraits method:

```
OperatorTraits<ScalarType,MV,OP>::Apply(const OP &Op,
```

Anasazi: eigen-solver framework

- Utilizes abstract interfaces for operators and multi-vectors
- Specifies what operations the multi-vectors and operators must support
- Access to unterlying object with Anasazi::MultiVecTraits and Anasazi::OperatorTraits
- MultiVecTraits: e.g., MvAddMv, MvDot, MvNorm, …
- OperatorTraits method:

```
OperatorTraits<ScalarType,MV,OP>::Apply(const OP &Op,
                                        const MV &x, MV &y)
```

Anasazi: classes for $Ax = \lambda Bx$

Anasazi: classes for $Ax = \lambda Bx$

- `Anasazi::Eigenproblem`

Anasazi: classes for $Ax = \lambda Bx$

- `Anasazi::Eigenproblem`
  - Contains components of eigen-problem

Anasazi: classes for $Ax = \lambda Bx$

- `Anasazi::Eigenproblem`
  - Contains components of eigen-problem
  - setOperator, SetA, SetB, setPrec, setInitVec

Anasazi: classes for $Ax = \lambda Bx$

- `Anasazi::Eigenproblem`
  - Contains components of eigen-problem
  - setOperator, SetA, SetB, setPrec, setInitVec
- `Anasazi::Eigensolution`

Anasazi: classes for $Ax = \lambda Bx$

- `Anasazi::Eigenproblem`
  - Contains components of eigen-problem
  - setOperator, SetA, SetB, setPrec, setInitVec
- `Anasazi::Eigensolution`
  - Manages the solution of the eigen-problem

Anasazi: classes for $Ax = \lambda Bx$

- `Anasazi::Eigenproblem`
  - Contains components of eigen-problem
  - setOperator, SetA, SetB, setPrec, setInitVec
- `Anasazi::Eigensolution`
  - Manages the solution of the eigen-problem
- `Anasazi::Eigensolver`

Anasazi: classes for $Ax = \lambda Bx$

- `Anasazi::Eigenproblem`
  - Contains components of eigen-problem
  - setOperator, SetA, SetB, setPrec, setInitVec
- `Anasazi::Eigensolution`
  - Manages the solution of the eigen-problem
- `Anasazi::Eigensolver`
  - Defines interface which must be met by any solver

Anasazi: classes for $Ax = \lambda Bx$

- `Anasazi::Eigenproblem`
  - Contains components of eigen-problem
  - setOperator, SetA, SetB, setPrec, setInitVec
- `Anasazi::Eigensolution`
  - Manages the solution of the eigen-problem
- `Anasazi::Eigensolver`
  - Defines interface which must be met by any solver
  - Currently implemented solvers: BlockDavidson, BlockKrylovSchur, LOBPCG

Anasazi: classes for $Ax = \lambda Bx$

- `Anasazi::Eigenproblem`
  - Contains components of eigen-problem
  - setOperator, SetA, SetB, setPrec, setInitVec
- `Anasazi::Eigensolution`
  - Manages the solution of the eigen-problem
- `Anasazi::Eigensolver`
  - Defines interface which must be met by any solver
  - Currently implemented solvers: BlockDavidson, BlockKrylovSchur, LOBPCG
- `Anasazi::SolverManager`

Anasazi: classes for $Ax = \lambda Bx$

- `Anasazi::Eigenproblem`
  - Contains components of eigen-problem
  - setOperator, SetA, SetB, setPrec, setInitVec
- `Anasazi::Eigensolution`
  - Manages the solution of the eigen-problem
- `Anasazi::Eigensolver`
  - Defines interface which must be met by any solver
  - Currently implemented solvers: BlockDavidson, BlockKrylovSchur, LOBPCG
- `Anasazi::SolverManager`
  - ‘Turn-key’ class to use existing eigen-solvers

Anasazi: other classes

Anasazi: other classes

- `Anasazi::StatusTest`
  - Responsible for stopping the solver iteration
- `Anasazi::SortManager`
  - Sorts the eigenvalues and eigenvectors
  - Increasing/decreasing magnitude, real/imaginary part
- `Anasazi::OrthoManager`
  - Provides methods to perform (re-)orthogonalization
- `Anasazi::OutputManager`
  - Controls verbosity of output

Parallel Libraries Summary

- The take home message is:

  Don't recreate the wheel

- A vast number of high-quality, well-maintained public domain parallel libraries available for numerical and semi-numerical problems
- Some large community efforts are: ScaLAPACK (MAGMA/PLASMA), PETSc, Trilinos

Acknowledgments

Acknowledgments

- Osni Marques and Tony Drummond (LBL): ScaLAPACK material

Acknowledgments

- Osni Marques and Tony Drummond (LBL): ScaLAPACK material
- Susan Blackford: ScaLAPACK tutorial

Acknowledgments

- Osni Marques and Tony Drummond (LBL): ScaLAPACK material
- Susan Blackford: ScaLAPACK tutorial
- Chris Hastings: ScaLAPACK examples

Acknowledgments

- Osni Marques and Tony Drummond (LBL): ScaLAPACK material
- Susan Blackford: ScaLAPACK tutorial
- Chris Hastings: ScaLAPACK examples
- Rich Lehoucq, et al: Anasazi overview

Acknowledgments

- Osni Marques and Tony Drummond (LBL): ScaLAPACK material
- Susan Blackford: ScaLAPACK tutorial
- Chris Hastings: ScaLAPACK examples
- Rich Lehoucq, et al: Anasazi overview
- The PETSc team: most PETSc slides

Acknowledgments

- Osni Marques and Tony Drummond (LBL): ScaLAPACK material
- Susan Blackford: ScaLAPACK tutorial
- Chris Hastings: ScaLAPACK examples
- Rich Lehoucq, et al: Anasazi overview
- The PETSc team: most PETSc slides
- Trilinos tutorial

Acknowledgments

- Osni Marques and Tony Drummond (LBL): ScaLAPACK material
- Susan Blackford: ScaLAPACK tutorial
- Chris Hastings: ScaLAPACK examples
- Rich Lehoucq, et al: Anasazi overview
- The PETSc team: most PETSc slides
- Trilinos tutorial
- Neil Stringfellow: summer school organization

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction to Parallel Programming : Parallel Debugging Tools

CSCS summer school, 6-8 August 2012

ssh -Y coursexx@ela.cscs.ch
Slides : /project/csstaff/PP_school/Slides2012/debug

Do you have X running (xclock) ?
Doodle : http://is.gd/gJa8wP

OPT++: Object-Oriented Nonlinear Optimization Library

OPT++: Object-Oriented Nonlinear Optimization Library

- Minimization problem:

OPT++: Object-Oriented Nonlinear Optimization Library

- Minimization problem:
- Methods:

OPT++: Object-Oriented Nonlinear Optimization Library

- Minimization problem:

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \ldots, p, \\ & g_i(x) \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

- Methods:

OPT++: Object-Oriented Nonlinear Optimization Library

- Minimization problem:
- Methods:
  - various Newton methods;

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \ldots, p, \\ & g_i(x) \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

OPT++: Object-Oriented Nonlinear Optimization Library

- Minimization problem:
- Methods:
  - various Newton methods;
  - a parallel Newton method;

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \ldots, p, \\ & g_i(x) \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

OPT++: Object-Oriented Nonlinear Optimization Library

- Minimization problem:

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \ldots, p, \\ & g_i(x) \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

- Methods:
  - various Newton methods;
  - a parallel Newton method;
  - a nonlinear conjugate gradient method

OPT++: Object-Oriented Nonlinear Optimization Library

- Minimization problem:

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \ldots, p, \\ & g_i(x) \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

- Methods:
  - various Newton methods;
  - a parallel Newton method;
  - a nonlinear conjugate gradient method
  - a parallel direct search method

OPT++: Object-Oriented Nonlinear Optimization Library

- Minimization problem:

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \ldots, p, \\ & g_i(x) \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

- Methods:
  - various Newton methods;
  - a parallel Newton method;
  - a nonlinear conjugate gradient method
  - a parallel direct search method
  - a nonlinear interior point method.

OPT++: Object-Oriented Nonlinear Optimization Library

- Minimization problem:

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \ldots, p, \\ & g_i(x) \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

- Methods:
  - various Newton methods;
  - a parallel Newton method;
  - a nonlinear conjugate gradient method
  - a parallel direct search method
  - a nonlinear interior point method.
- http://crd.lbl.gov/~meza/projects/opt++/

OPT++: Object-Oriented Nonlinear Optimization Library

- Minimization problem:

$$\begin{aligned} \min_{x \in R^n} \quad & f(x) \\ \text{subject to} \quad & h_i(x) = 0, \quad i = 1, \ldots, p, \\ & g_i(x) \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

- Methods:
  - various Newton methods;
  - a parallel Newton method;
  - a nonlinear conjugate gradient method
  - a parallel direct search method
  - a nonlinear interior point method.
- http://crd.lbl.gov/~meza/projects/opt++/
- Parallel OPT++ only tested on Intel Linux, MPICH1.2.2.3

Teuchos: utility classes

Teuchos: utility classes

- Teuchos::ScalarTraits -- extension for arbitrary precisions

Teuchos: utility classes

- Teuchos::ScalarTraits -- extension for arbitrary precisions
- Teuchos::SerialDenseMatrix: templated version of Epetra_SerialDenseMatrix

Teuchos: utility classes

- Teuchos::ScalarTraits -- extension for arbitrary precisions
- Teuchos::SerialDenseMatrix: templated version of Epetra_SerialDenseMatrix
- Teuchos::BLAS -- templated wrappers for BLAS

Teuchos: utility classes

- Teuchos::ScalarTraits -- extension for arbitrary precisions
- Teuchos::SerialDenseMatrix: templated version of Epetra_SerialDenseMatrix
- Teuchos::BLAS -- templated wrappers for BLAS
- Teuchos::LAPACK -- templated wrappers for LAPACK

Teuchos: utility classes

- Teuchos::ScalarTraits -- extension for arbitrary precisions
- Teuchos::SerialDenseMatrix: templated version of Epetra_SerialDenseMatrix
- Teuchos::BLAS -- templated wrappers for BLAS
- Teuchos::LAPACK -- templated wrappers for LAPACK
- Teuchos::ParameterList: container to group parameters

Teuchos: utility classes

- Teuchos::ScalarTraits -- extension for arbitrary precisions
- Teuchos::SerialDenseMatrix: templated version of Epetra_SerialDenseMatrix
- Teuchos::BLAS -- templated wrappers for BLAS
- Teuchos::LAPACK -- templated wrappers for LAPACK
- Teuchos::ParameterList: container to group parameters
- Teuchos::RCP: smart reference-counted pointer class with garbage collection

Teuchos: utility classes

- Teuchos::ScalarTraits -- extension for arbitrary precisions
- Teuchos::SerialDenseMatrix: templated version of Epetra_SerialDenseMatrix
- Teuchos::BLAS -- templated wrappers for BLAS
- Teuchos::LAPACK -- templated wrappers for LAPACK
- Teuchos::ParameterList: container to group parameters
- Teuchos::RCP: smart reference-counted pointer class with garbage collection
- Others…

Epetra: pictorial summary

Epetra: pictorial summary

Epetra: pictorial summary

Epetra: pictorial summary

Anasazi: the Epetra adapter

- Anasazi can use any linear algebra functionality satisfying the multi-vector and operator traits
- Implementing the MV and operator classes is tedious
- Epetra/EpetraExt provide the needed classes, e.g.,

```
#include “AnasaziEpetraAdapter.hpp
typedef Epetra_MultiVector MV;
typedef Epetra_Operator OP;
// Multi-vectors of type MV
Teuchos::RefCountPtr<MM> X = Teuchos::rcp( new MV(…) );
// Operators can be any subclass of OP
Teuchos::RefCountPtr<OP> A = Teuchos::rcp( new
Epetra_CrsMatrix(…) );
```