SAND2008-2188P

Belos:
A Framework for Next-generation Iterative Linear Solvers

April 7th, 2008

Mike Heroux
Rich Lehoucq
Mike Parks
Heidi Thornquist (Lead)

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

Sandia National Laboratories

Outline

- Introduction
  - Motivation
  - Belos design
- What's in Trilinos 8.0
  - Framework implementations
  - Current linear solver iterations
  - Parameter list driven solver managers
  - Stratimikos interface
- Concluding Remarks
  - What's next for Belos …

Sandia National Laboratories

Introducing Belos

- Next-generation linear solver library, written in templated C++.
- Provide a generic framework for developing algorithms for solving large-scale linear problems.
- Algorithm implementation is accomplished through the use of traits classes and abstract base classes:
  - ex.: **`MultiVecTraits`**, **`OperatorTraits`**
  - ex.: **`Iteration, SolverManager, StatusTest`**
- Includes block linear solvers:
  - Higher operator performance.

Sandia National Laboratories

Why Belos?

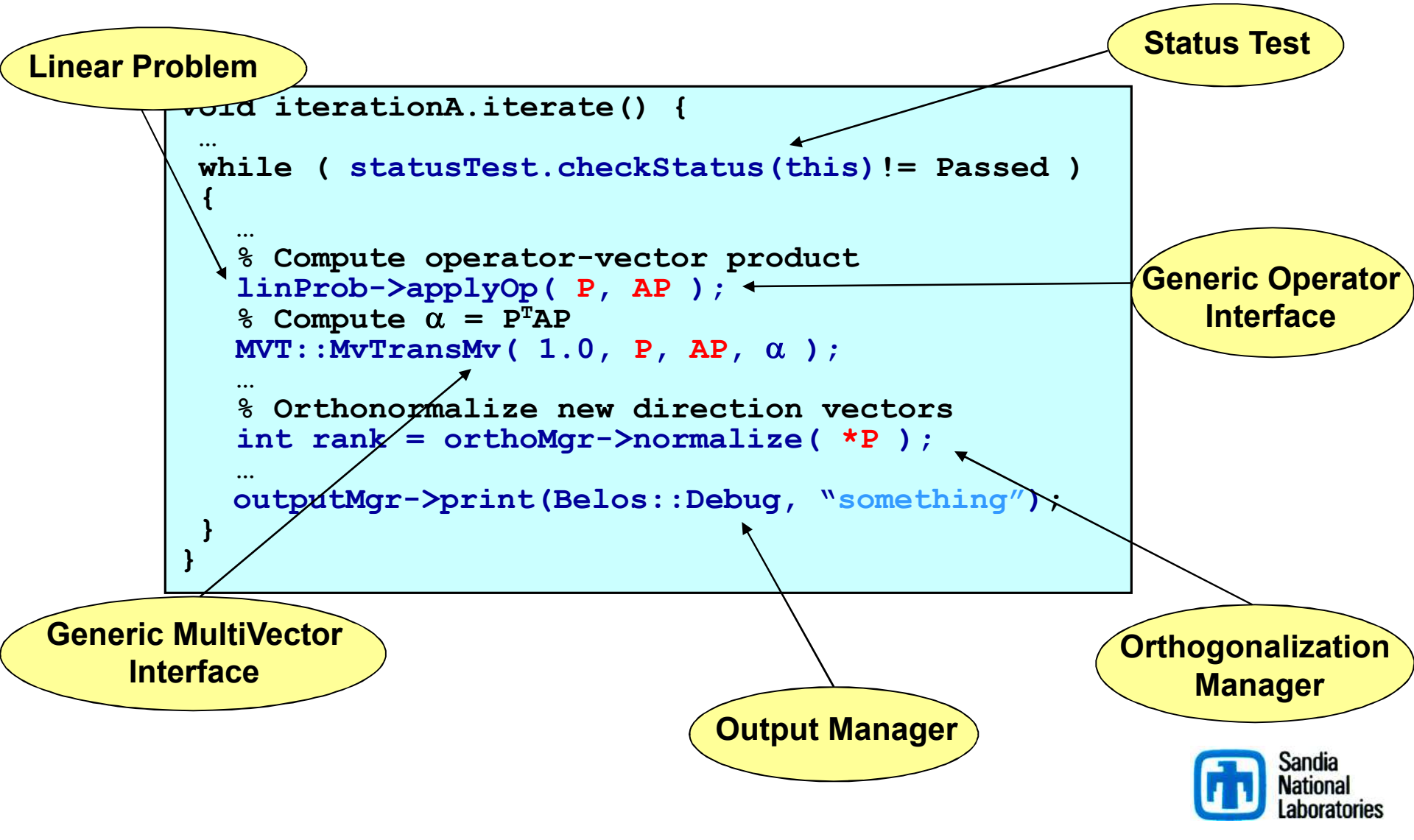
- AztecOO provides solvers for $Ax=b$
- What about solvers for:
  - Simultaneously solved systems w/ multiple-RHS: $AX = B$
  - Sequentially solved systems w/ multiple-RHS: $AX_i = B_i$, $i=1,\ldots,t$
  - Sequences of multiple-RHS systems: $A_iX_i = B_i$, $i=1,\ldots,t$
- Many advanced methods for these types of linear systems
  - Block methods: block GMRES [Vital], block CG/BICG [O'Leary]
  - "Seed" solvers: hybrid GMRES [Nachtigal, et al.]
  - Recycling solvers: recycled Krylov methods [Parks, et al.]
  - Restarting techniques, orthogonalization techiques, …
- Efficient R&D of advanced linear solution methods requires:
  - Interoperability
  - Extensibility
  - Reusability

Belos Design

Design Goal: Create a linear solver library that:

1. Provides an abstract interface to an operator-vector products, scaling, and preconditioning.
2. Allows the user to enlist any linear algebra package for the elementary vector space operations essential to the algorithm. (Epetra, PETSc, Thyra, etc.)
3. Allows the user to define convergence of any algorithm (a.k.a. status testing).
4. Allows the user to determine the verbosity level, formatting, and processor for the output.
5. Allows for easier R&D of new solvers through “managers” using “iterations” as the basic kernels.

Generic Iteration Kernel



Sandia National Laboratories

Generic Solver Manager

```
Belos::ReturnType solverMgrA.solve() {
  …
  <initialize iteration>
  …
  void iterationA.iterate() {
   …
   while ( statusTest.checkStatus(this)!= Passed )
   {
     …
     % Compute operator-vector product
     linProb->applyOp( P, AP );
     % Compute α = P^T AP
     MVT::MvTransMv( 1.0, P, AP, α );
     …
     % Orthonormalize new direction vectors
     int rank = orthoMgr->normalize( *P );
     …
     outputMgr->print(Belos::Debug, "something");
   }
  }
  …
  <restart, deflate converged solutions>
  …
}
```

Solution Strategy

Generic Iteration Kernel

Sandia National Laboratories

Belos Classes

- MultiVecTraits and OperatorTraits
  - Traits classes for interfacing linear algebra
- LinearProblem Class
  - Describes the problem and stores the answer
- OrthoManager Class
  - Provide basic interface for orthogonalization
- StatusTest Class
  - Control testing of convergence, etc.
- OutputManager Class
  - Control verbosity and printing in a MP scenario
- Iteration Class
  - Provide basic linear solver iteration interface.
- SolverManager Class
  - Parameter list driven strategy object describing behavior of solver

What's in Trilinos 8.0

Sandia National Laboratories

Linear Algebra Interface

- **`Belos::MultiVecTraits<ST,MV>`**
  - Interface to define the linear algebra required by most iterative linear solvers:
    - creational methods
    - dot products, norms
    - update methods
    - initialize / randomize
  - Implementations:
    - **`MultiVecTraits<double,Epetra_MultiVector>`**
    - **`MultiVecTraits<ST,Thyra::MultiVectorBase<ST> >`**
- **`Belos::OperatorTraits<ST,MV,OP>`**
  - Interface to enable the application of an operator to a multivector.
  - Implementations:
    - **`OperatorTraits<double,Epetra_MultiVector,Epetra_Operator>`**
    - **`OperatorTraits<ST,Thyra::MultiVectorBase<ST>,Thyra::LinearOpBase<ST> >`**

Sandia National Laboratories

Linear Problem Interface

- Provides an interface between the basic iterations and the linear problem to be solved.
- Templated class **`Belos::LinearProblem<ST,MV,OP>`**
  - Allows preconditioning to be removed from the algorithm.
  - Behavior defined through traits mechanisms.
  - Methods:
    - **`setOperator(…) / getOperator()`**
    - **`setLHS(…) / getLHS()`**
    - **`setRHS(…) / getRHS()`**
    - **`setLeftPrec(…) / getLeftPrec() / isLeftPrec()`**
    - **`setRightPrec(…) / getRightPrec() / isRightPrec()`**
    - **`apply(…) / applyOp(…) / applyLeftPrec(…) / applyRightPrec(…)`**
    - **`setHermitian(…) / isHermitian()`**
    - **`setProblem(…)`**

Orthogonalization Manager

- Abstract interface to orthogonalization / orthonormalization routines for multivectors.
- Abstract base class **`Belos::[Mat]OrthoManager<ST,MV,OP>`**
  - `void innerProd(…) const;`
  - `void norm(…) const;`
  - `void project(…) const;`
  - `int normalize(…) const;`
  - `int projectAndNormalize(…) const;`
  - `magnitudeType orthogError(…) const;`
  - `magnitudeType orthonormError(…) const;`
- Implementations:
  - **`Belos::DGKSOrthoManager`**
  - **`Belos::ICGSOrthoManager`**
  - **`Belos::IMGSOrthoManager`**

Sandia National Laboratories

StatusTest Interface

- Informs linear solver iteration of change in state, as defined by user.
- Similar to NOX / AztecOO.
- Multiple criterion can be logically connected.
- Abstract base class **`Belos::StatusTest<ST,MV,OP>`**
  - `TestStatus checkStatus( Iteration<…>* iterate );`
  - `TestStatus getStatus() const;`
  - `void clearStatus();`
  - `void reset();`
  - `ostream& print( ostream& os, int indent = 0 ) const;`
- Implementations:
  - **`Belos::StatusTestMaxIters`**
  - **`Belos::StatusTestGenResNorm`**
  - **`Belos::StatusTestImpResNorm`**
  - **`Belos::StatusTestOutput`**
  - **`Belos::StatusTestCombo`**

Sandia National Laboratories

Output Manager Interface

- Templated class that enables control of the linear solver output.
  - Behavior defined through traits mechanism
- **`Belos::OutputManager<ST>`**
  - Get/Set Methods:
    - `void setVerbosity( int vbLevel );`
    - `int getVerbosity( );`
    - `ostream& stream( MsgType type );`
  - Query Methods:
    - `bool isVerbosity( MsgType type );`
  - Print Methods:
    - `void print( MsgType type, const string output );`
  - Message Types:
    - `Belos::Errors, Belos::Warnings, Belos::IterationDetails, Belos::OrthoDetails, Belos::FinalSummary, Belos::TimingDetails, Belos::StatusTestDetails, Belos::Debug`
- Default is lowest verbosity (`Belos::Errors`), output on one processor.

Sandia National Laboratories

Iteration Interface

- Provides an abstract interface to Belos basic iteration kernels.
- Abstract base class **`Belos::Iteration<ST,MV,OP>`**
  - `int getNumIters() const; void resetNumIters(int iter);`
  - `Teuchos::RCP<const MV> getNativeResiduals( … ) const;`
  - `Teuchos::RCP<const MV> getCurrentUpdate() const;`
  - `int getBlockSize() const; void setBlockSize(int blockSize);`
  - `const LinearProblem<ST,MV,OP>& getProblem() const;`
  - `void iterate(); void initialize();`
- Iterations require these classes:
  - **`Belos::LinearProblem`**
  - **`Belos::OutputManager`**
  - **`Belos::StatusTest`**
  - **`Belos::OrthoManager`**
- Implementations:
  - **`Belos::BlockGmresIter`**
  - **`Belos::BlockFGmresIter`**
  - **`Belos::PseudoBlockGmresIter`**
  - **`Belos::GCRODRIter`**
  - **`Belos::CGIter,`**
  - **`Belos::BlockCGIter`**

Sandia National Laboratories

Solver Manager

- Provides an abstract interface to Belos solver managers
  - solver strategies
- Abstract base class **`Belos::SolverManager`**
  - `void setProblem(…); void setParameters(…);`
  - `const Belos::LinearProblem<ST,MV,OP>& getProblem() const;`
  - `Teuchos::RCP<const Teuchos::ParameterList> getValidParameters() const;`
  - `Teuchos::RCP<const Teuchos::ParameterList> getCurrentParameters() const;`
  - `Belos::ReturnType solve();`
- Solvers are parameter list driven, take two arguments:
  - **`Belos::LinearProblem`**
  - **`Teuchos::ParameterList`** [validated]
- Implementations:
  - **`Belos::BlockGmresSolMgr`**
  - **`Belos::PseudoBlockGmresSolMgr`**
  - **`Belos::GCRODRSolMgr`**
  - **`Belos::BlockCGSolMgr`**

Sandia National Laboratories

Solver Manager Implementations

- **`Belos::BlockGmresSolMgr`**
  - Performs regular and flexible block GMRES
  - Block size can be greater than, or less than, the number of right-hand sides
  - Adaptive block size; no deflation
- **`Belos::PseudoBlockGmresSolMgr`**
  - Single-vector simultaneous GMRES solves (block size)
  - Iteration performs block application of operator-vector products
  - Easier deflation strategy; “Deflation Quorum”
- **`Belos::GCRODRSolMgr`**
  - Single-vector Krylov subspace recycling solver written by Mike Parks
  - Validates iteration kernel design by using **`Belos::BlockGmresIter`**
- **`Belos::BlockCGSolMgr`**
  - Performs single-vector and block CG
  - Block size can be greater than, or less than, the number of right-hand sides
  - Adaptive block size, deflation

Examples

Sandia National Laboratories

Block CG Example

(belos/example/BlockCG/BlockCGEpetraExFile.cpp)

```
#include "BelosConfigDefs.hpp"
#include "BelosLinearProblem.hpp"
#include "BelosEpetraAdapter.hpp"
#include "BelosBlockCGSolMgr.hpp"
…
int main(int argc, char *argv[]) {
  typedef Epetra_MultiVector MV;
  typedef Epetra_Operator OP;
  …
  // Create AX=B
  Teuchos::RCP<Epetra_Operator> A;
  Teuchos::RCP<Epetra_MultiVector> X;
  Teuchos::RCP<Epetra_MultiVector> B;
  …
  // Create parameter list
  Teuchos::ParameterList belosList;
  belosList.set( "Block Size", blocksize );
  belosList.set( "Maximum Iterations", maxiters );
  belosList.set( "Convergence Tolerance", tol );
  if (verbose) {
    belosList.set( "Verbosity", Belos::Errors +
     Belos::Warnings + Belos::TimingDetails +
     Belos::FinalSummary + Belos::StatusTestDetails
     );
    belosList.set( "Output Frequency", frequency );
  }
  else
    belosList.set( "Verbosity", Belos::Errors +
     Belos::Warnings );
```

```
// Construct an unpreconditioned linear problem
Belos::LinearProblem<double,MV,OP> problem( A, X, B );
problem.setHermitian();
bool set = problem.setProblem();  // VERY IMPORTANT!
if (set == false) {
  return -1;
}

// Create an iterative solver manager
Belos::BlockCGSolMgr<double,MV,OP> newSolver(
   rcp(&problem,false), rcp(&belosList,false) );

// Perform solve
Belos::ReturnType ret = newSolver.solve();

if ( ret!=Belos::Converged ) {
  // Belos did not converge!
  return -1;
}
```

Sandia National Laboratories

Preconditioned Block CG Example

(belos/example/BlockCG/BlockPrecCGEpetraExFile.cpp)

```
#include "BelosConfigDefs.hpp"
#include "BelosLinearProblem.hpp"
#include "BelosEpetraAdapter.hpp"
#include "BelosBlockCGSolMgr.hpp"
…
int main(int argc, char *argv[]) {
  typedef Epetra_MultiVector MV;
  typedef Epetra_Operator OP;
  …
  // Create AX=B
  Teuchos::RCP<OP> A;
  Teuchos::RCP<MV> X;
  Teuchos::RCP<MV> B;
  …
  // Create parameter list
  Teuchos::ParameterList belosList;
  belosList.set( "Block Size", blocksize );
  belosList.set( "Maximum Iterations", maxiters );
  belosList.set( "Convergence Tolerance", tol );
  if (verbose) {
    belosList.set( "Verbosity", Belos::Errors +
     Belos::Warnings + Belos::TimingDetails +
     Belos::FinalSummary + Belos::StatusTestDetails
     );
    belosList.set( "Output Frequency", frequency );
  }
  else
    belosList.set( "Verbosity", Belos::Errors +
     Belos::Warnings );
```

```
// Construct an SPD preconditioner M
// Anything that supports OP interface can go here!
Teuchos::RCP<OP> M;
…
// Construct a preconditioned linear problem
Belos::LinearProblem<double,MV,OP> problem( A, X, B );
problem.setHermitian();
if (leftprec)
  problem.setLeftPrec( M );
else
  problem.setRightPrec( M );
bool set = problem.setProblem();  // VERY IMPORTANT!
if (set == false) {
  return -1;
}

// Create an iterative solver manager
Belos::BlockCGSolMgr<double,MV,OP> newSolver(
   rcp(&problem,false), rcp(&belosList,false) );

// Perform solve
Belos::ReturnType ret = newSolver.solve();

if ( ret!=Belos::Converged ) {
  // Belos did not converge!
  return -1;
}
```

Sandia National Laboratories

Preconditioned Block GMRES Example

(belos/example/BlockGmres/BlockPrecGmresEpetraExFile.cpp)

```
#include "BelosConfigDefs.hpp"
#include "BelosLinearProblem.hpp"
#include "BelosEpetraAdapter.hpp"
#include "BelosBlockGmresSolMgr.hpp"
…
int main(int argc, char *argv[]) {
  typedef Epetra_MultiVector MV;
  typedef Epetra_Operator OP;
  …
  // Create AX=B
  Teuchos::RCP<OP> A;
  Teuchos::RCP<MV> X;
  Teuchos::RCP<MV> B;
  …
  // Create parameter list
  Teuchos::ParameterList belosList;
  belosList.set( "Num Blocks", maxsubspace );
  belosList.set( "Block Size", blocksize );
  belosList.set( "Maximum Iterations", maxiters );
  belosList.set( "Maximum Restarts", maxrestarts );
  belosList.set( "Convergence Tolerance", tol );
  if (verbose) {
    belosList.set( "Verbosity", Belos::Errors +
     Belos::Warnings + Belos::TimingDetails +
     Belos::FinalSummary + Belos::StatusTestDetails
     );
    belosList.set( "Output Frequency", frequency );
  }
  else
    belosList.set( "Verbosity", Belos::Errors +
     Belos::Warnings );
```

```
  // Construct a preconditioner M
  // Anything that supports OP interface can go here!
  Teuchos::RCP<OP> M;
  …
  // Construct a preconditioned linear problem
  Belos::LinearProblem<double,MV,OP> problem( A, X, B );
  if (leftprec)
    problem.setLeftPrec( M );
  else
    problem.setRightPrec( M );
  bool set = problem.setProblem();  // VERY IMPORTANT!
  if (set == false) {
    return -1;
  }

  // Create an iterative solver manager
  Belos::BlockGmresSolMgr<double,MV,OP> newSolver(
     rcp(&problem,false), rcp(&belosList,false) );

  // Perform solve
  Belos::ReturnType ret = newSolver.solve();

  if ( ret!=Belos::Converged ) {
    // Belos did not converge!
    if ( newSolver.isLOADetected() ) {
      // Convergence was impeded by a loss of accuracy
    }
    return -1;
  }
```

Sandia National Laboratories

Stratimikos-Belos Interface

- Stratimikos:
  - Thyra-based linear solver and preconditioner strategy package
    - `MultiVecTraits<ST,Thyra::MultiVectorBase<ST> >`
    - `OperatorTraits<ST,Thyra::MultiVectorBase<ST>,Thyra::LinearOpBase<ST> >`
  - Belos-Thyra-Tpetra interface

- Stratimikos-Belos Interface
  - Intent: access current Belos solver managers using a factory interface
  - Implements Thyra::LinearOpWithSolveFactory / Thyra::LinearOpWithSolve
  - Stratimikos interface uses valid parameter list generated from Belos
  - Check out:

    `stratimikos/adapters/belos/example/LOWSFactory/[Epetra/Tpetra]`

Sandia National Laboratories

What's next for Belos?

- Belos provides a powerful framework for developing robust linear solvers, but there's more work to do …
- Future Development:
  - More performance improvements
  - More advanced solution methods
- Check out the Trilinos Tutorial:

  http://trilinos.sandia.gov/Trilinos8.0Tutorial.pdf

- See Belos website for more info:

  http://trilinos.sandia.gov/packages/belos

Sandia National Laboratories