

The Trilinos Solver Framework

Kevin Long (SNL/CA)

Mike Heroux (SNL/Minnesota)

Trilinos Project Overview

- PI: Mike Heroux
- <http://www.cs.sandia.gov/Trilinos/>

EPetra
Core matrix-vector

AztecOO
Aztec wrappers

IFPack
Incomplete factorizations

NOX
Nonlinear solvers

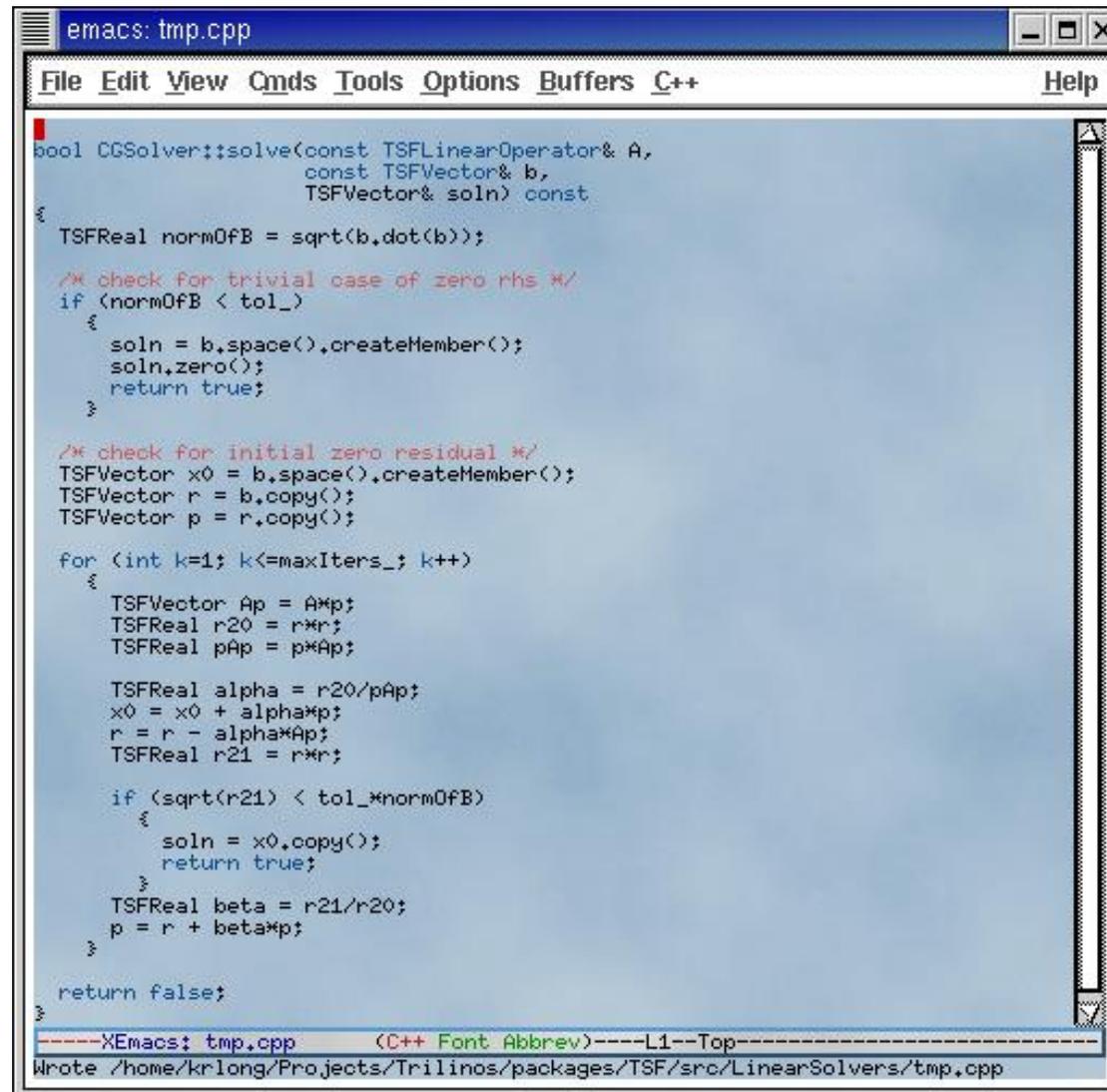
ML
Algebraic Multilevel

TSF
High-level interface

TSF

- Interface for representation-independent solvers
 - Abstract interfaces for vectors and operators
 - Composable operators
 - Block operators
- Design goal
 - Matlab-like simplicity, running on a supercomputer
- High-level conveniences
 - Minimal-overhead operator overloading
 - Transparent memory management
- Many design ideas derived from Gockenbach and Symes' Hilbert Class Library

Simple example: a representation-independent conjugate gradients solver



```
emacs: tmp.cpp
File Edit View Cmds Tools Options Buffers C++ Help

bool CGSolver::solve(const TSFLinearOperator& A,
                    const TSFVector& b,
                    TSFVector& soln) const
{
    TSFReal normOfB = sqrt(b.dot(b));

    /* check for trivial case of zero rhs */
    if (normOfB < tol_)
    {
        soln = b.space().createMember();
        soln.zero();
        return true;
    }

    /* check for initial zero residual */
    TSFVector x0 = b.space().createMember();
    TSFVector r = b.copy();
    TSFVector p = r.copy();

    for (int k=1; k<=maxIters_; k++)
    {
        TSFVector Ap = A*p;
        TSFReal r20 = r*r;
        TSFReal pAp = p*Ap;

        TSFReal alpha = r20/pAp;
        x0 = x0 + alpha*p;
        r = r - alpha*Ap;
        TSFReal r21 = r*r;

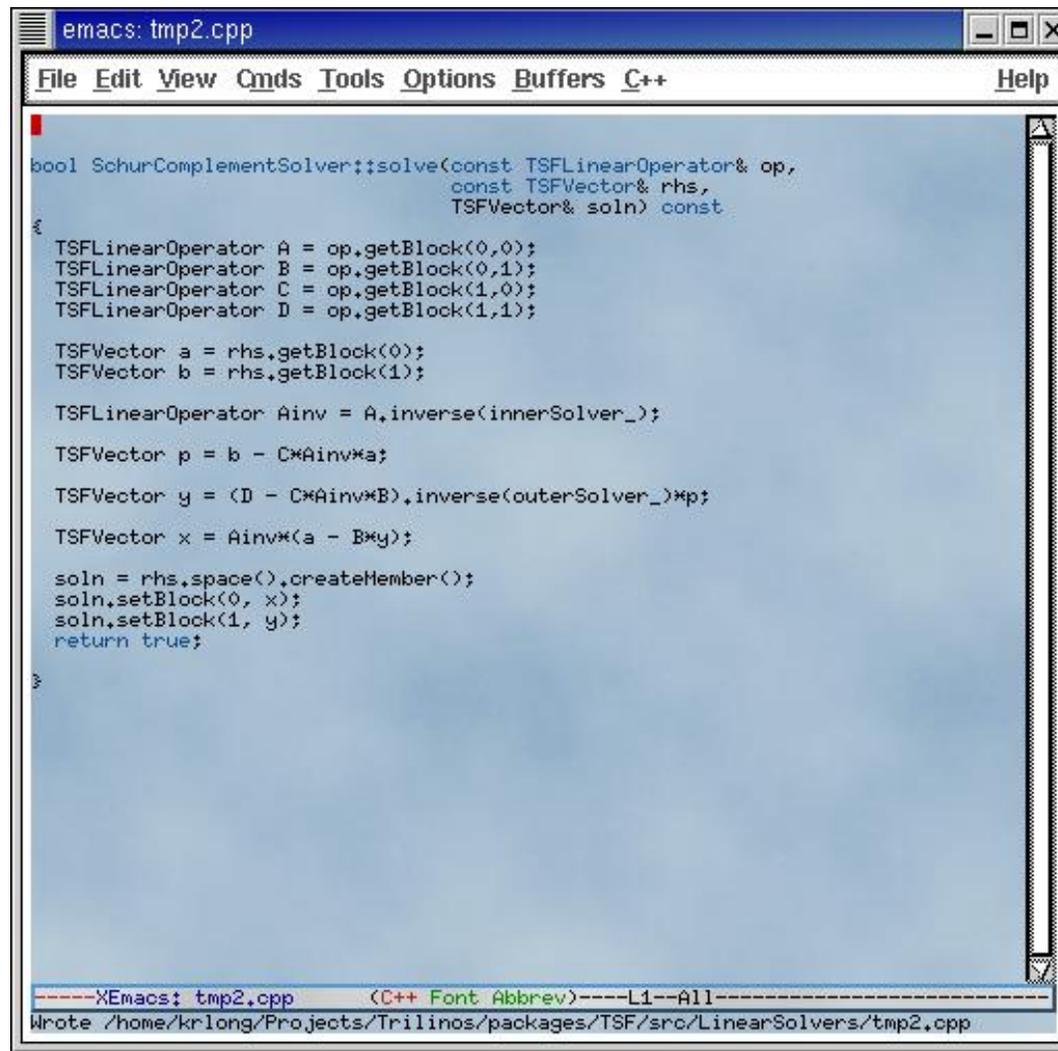
        if (sqrt(r21) < tol_*normOfB)
        {
            soln = x0.copy();
            return true;
        }

        TSFReal beta = r21/r20;
        p = r + beta*p;
    }

    return false;
}

-----XEmacs: tmp.cpp (C++ Font Abbrev)-----L1--Top-----
Write /home/krlong/Projects/Trilinos/packages/TSF/src/LinearSolvers/tmp.cpp
```

written with block and composed operators



```
emacs: tmp2.cpp
File Edit View Cmds Tools Options Buffers C++ Help

bool SchurComplementSolver::solve(const TSFLinearOperator& op,
                                  const TSFVector& rhs,
                                  TSFVector& soln) const
{
    TSFLinearOperator A = op.getBlock(0,0);
    TSFLinearOperator B = op.getBlock(0,1);
    TSFLinearOperator C = op.getBlock(1,0);
    TSFLinearOperator D = op.getBlock(1,1);

    TSFVector a = rhs.getBlock(0);
    TSFVector b = rhs.getBlock(1);

    TSFLinearOperator Ainv = A.inverse(innerSolver_);

    TSFVector p = b - C*Ainv*a;

    TSFVector y = (D - C*Ainv*B).inverse(outerSolver_)*p;

    TSFVector x = Ainv*(a - B*y);

    soln = rhs.space().createMember();
    soln.setBlock(0, x);
    soln.setBlock(1, y);
    return true;
}

-----XEmacs: tmp2.cpp (C++ Font Abbrev)-----L1--All-----
Write /home/krlong/Projects/Trilinos/packages/TSF/src/LinearSolvers/tmp2.cpp
```

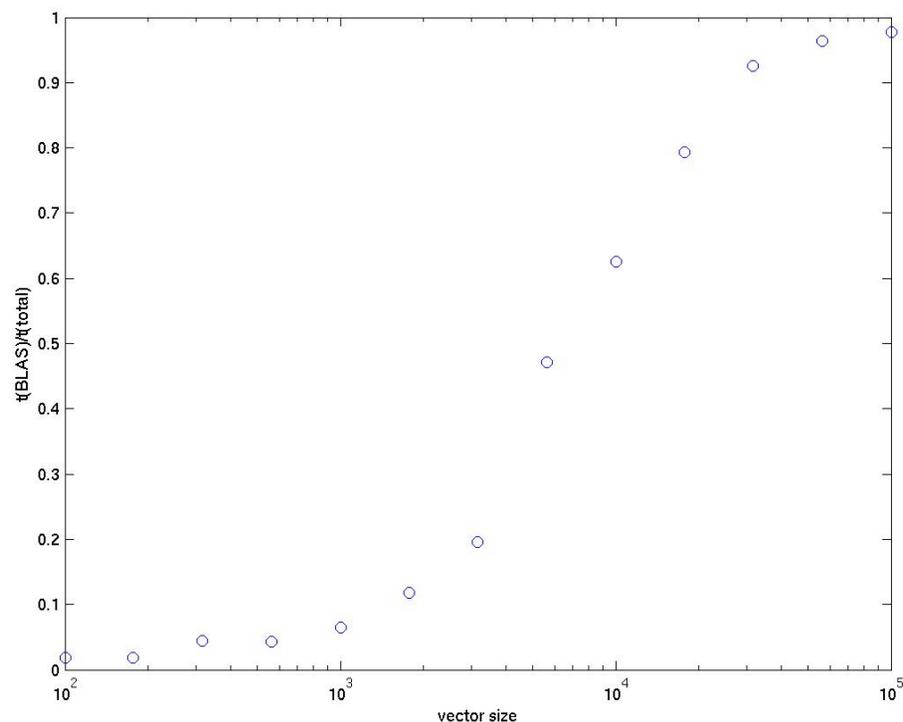
But is it fast?

- ⋮ The TSF approach involves considerable overhead
 - » Handle classes involve smart pointer dereferencing overhead
 - » Inheritance-based design is costlier than template-based generic programming
 - » Operator overloading

- ⋮ However...
 - » Overhead is constant as problem size increases
 - » Operator overloading can be done carefully to avoid large temporaries

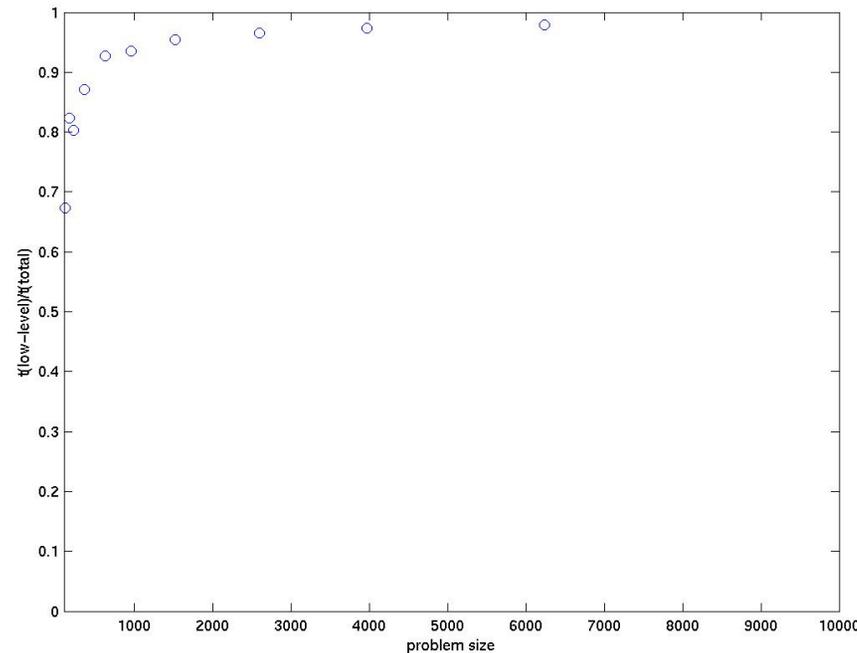
Operator overloading can be made fast

- Overloaded TSFVector operations are deferred
 - Alternative to expression templates
 - Temporaries are unavoidable with overloaded C++ operators
 - Result of each binary operation is a deferred linear combination object
 - Carry out actual calculations only when necessary



The overhead incurred by using TSF is negligible in a linear solve

- Figure shows low-level Epetra and BLAS times relative to total TSF solve time.
- Poisson solved with TSF BICGSTAB and ILU(1)
- TSF and operator overloading overhead is $< 5\%$ for problems larger than $N \sim 1500$



A short TSF tutorial

- About the code
- Creating vectors and matrices
- Filling a matrix with values
- Building complicated linear operators
- Preconditioners and Preconditioner Factories
- Linear solvers
- Nonlinear operators
- Nonlinear solvers

General code style

- ; TSF is in C++
- ; User-level objects such as TSFVector, TSFLinearSolver are handle classes
 - » At the user level, vectors are created from the createMember() method of vector space
 - » Create objects such as solvers from a concrete type as follows:
 - TSFLinearSolver solver = new BICGSTABSolver(...):
- ; Concrete implementations, e.g, PetraVector, derived from extensible base classes

Core interface components

; Vectors

- » TSFVectorTypeBase
- » TSFVectorSpaceBase
- » TSFVectorBase

; Operators

- » TSFLinearOperatorBase
- » TSFMatrixOperator
- » TSFNonlinearOperatorBase

; Preconditioners

- » TSFPreconditionerFactoryBase

; Solvers

- ; TSFLinearSolverBase
- ; TSFNonlinearSolverBase

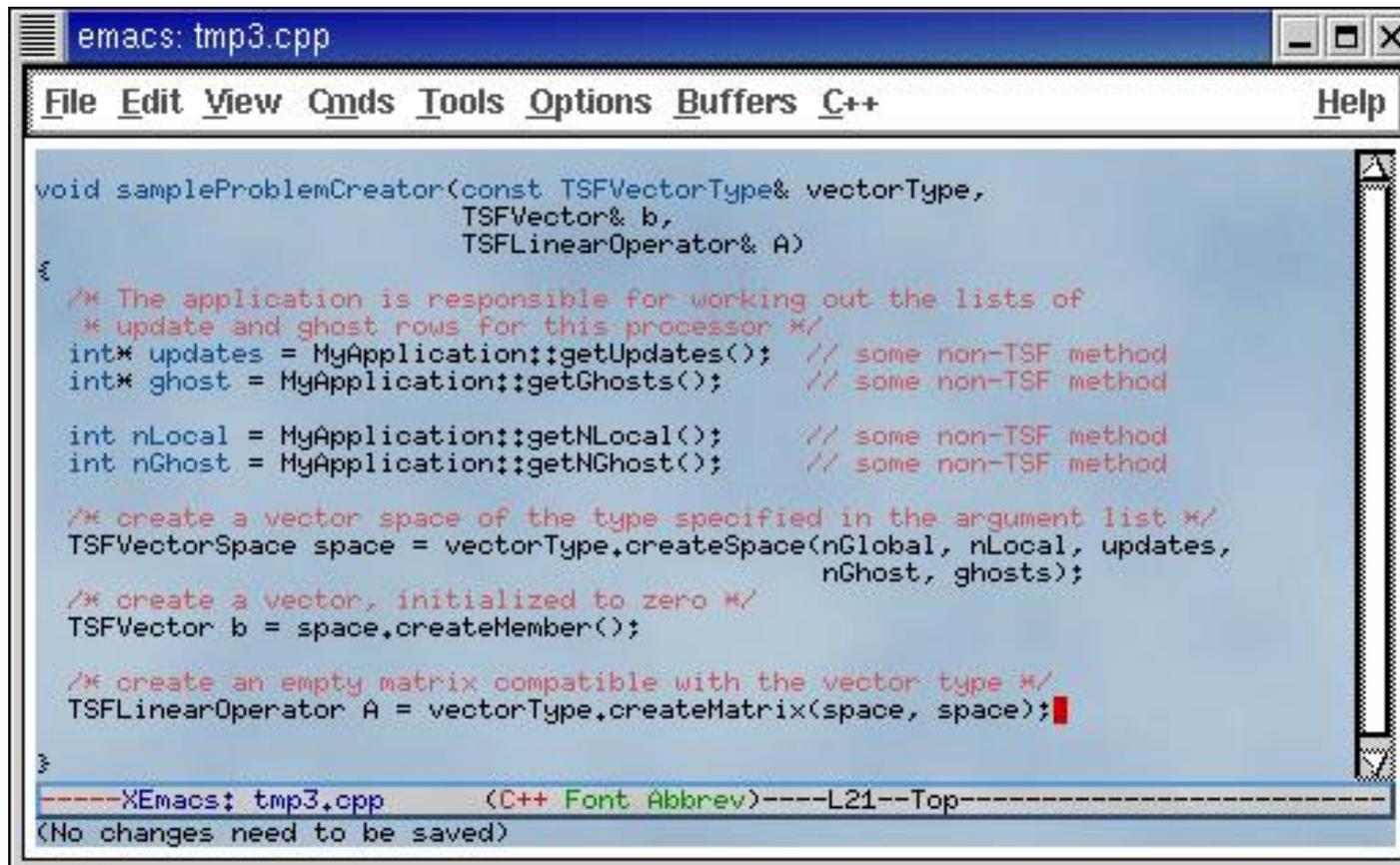
; I/O

- » TSFMatrixReaderBase
- » TSFMatrixWriterBase

» Utilities

- » TSFErrorHandlerBase
- » TSFWriterBase

C++ TSFVectorType and TSFVectorSpace objects to create vectors and matrices of a specified type



```
emacs: tmp3.cpp
File Edit View Cmds Tools Options Buffers C++ Help

void sampleProblemCreator(const TSFVectorType& vectorType,
                          TSFVector& b,
                          TSFLinearOperator& A)
{
    /* The application is responsible for working out the lists of
     * update and ghost rows for this processor */
    int* updates = MyApplication::getUpdates(); // some non-TSF method
    int* ghost = MyApplication::getGhosts(); // some non-TSF method

    int nLocal = MyApplication::getNLocal(); // some non-TSF method
    int nGhost = MyApplication::getNGhost(); // some non-TSF method

    /* create a vector space of the type specified in the argument list */
    TSFVectorSpace space = vectorType.createSpace(nGlobal, nLocal, updates,
                                                  nGhost, ghosts);

    /* create a vector, initialized to zero */
    TSFVector b = space.createMember();

    /* create an empty matrix compatible with the vector type */
    TSFLinearOperator A = vectorType.createMatrix(space, space);
}

-----XEmacs: tmp3.cpp (C++ Font Abbrev)-----L21--Top-----
(No changes need to be saved)
```

TSF provides an interface for configuring and loading a matrix

- ; First build the matrix as shown previously
- ; Next, configure the matrix
 - » `setColumnSize()`
 - » `freezeStructure()`
- ; Load the values
 - » `setRowStructure()` and `addToRow()` for each row
- ; Finalize the matrix, distributing ghost rows
 - » `freezeValues()`

Adjoint, inverse, and adjoint-inverse operators can be created at a high level

- TSFAdjointOperator applies the adjoint of another operator

```
TSFLinearOperator adj = new TSFAdjointOperator(A);  
    or more simply,  
TSFLinearOperator Aadj = A.adjoint();
```

- TSFInverseOperator applies the inverse of another operator using a specified solver. The inverse is never formed.

```
TSFLinearSolver solver = new BICGSTABSolver(...);  
TSFLinearOperator Ainv = A.inverse(solver);
```

- TSFInverseAdjointOperator applies the inverse adjoint

```
TSFLinearSolver solver = new BICGSTABSolver(...);  
TSFLinearOperator AinvAdj = A.inverseAdjoint(solver);
```

Overloaded operators are used to form efficient compound operators

- ; Operator composition with overloaded $*$, e.g. $A*B*C$
- ; Operator addition with overloaded $+$ and $-$, e.g. $A-B$
- ; Scalar multiplication with overloaded $*$, e.g. $a*A$

- ; All of these are implemented without forming the explicit matrix product or sum

Linear solvers in TSF

- class TSFLinearSolver
- Iterative solvers are simple to code in TSF using overloaded vector and matrix operations.
 - Currently, we have implemented BICGSTAB, CG, GMRES, and an interface to AztecOO
 - Flexible Krylov methods (FCG, FGMRES) implemented by V. Howle, used for fault-tolerant linear algebra
 - Block Krylov methods being implemented by M. Heroux and T. Barth
- Direct solvers can be implemented for matrix operators only
- Block solvers, e.g. Schur complement, block backsolve can be implemented matrix-free

Preconditioning

- TSFPreconditionerFactory builds a TSFPreconditioner for a given operator
- TSFPreconditioner has left() and right() methods to access operators.

Preconditioning

```
emacs: BICGSTABSolver.cpp
File Edit Apps Options Buffers Tools C++ Help
bool BICGSTABSolver::solve(const TSFLinearOperator& op,
                          const TSFVector& b,
                          TSFVector& soln) const
{
    TSFPreconditioner p = preconditionerFactory_.createPreconditioner(op);
    if (p.isIdentity())
    {
        return solveUnpreconditioned(op, b, soln);
    }
    else if (lp.hasRight())
    {
        TSFLinearOperator A = p.left()*op;
        TSFVector newRHS = b.space().createMember();
        p.left().apply(b, newRHS);
        return solveUnpreconditioned(A, newRHS, soln);
    }
    else if (lp.hasLeft())
    {
        TSFLinearOperator A = op * p.right();
        TSFVector intermediateSoln;
        bool success = solveUnpreconditioned(A, b, intermediateSoln);
        if (success) p.right().apply(intermediateSoln, soln);
        return success;
    }
    else
    {
        TSFLinearOperator A = p.left() * op * p.right();
        TSFVector newRHS;
        p.left().apply(b, newRHS);
        TSFVector intermediateSoln;
        bool success = solveUnpreconditioned(A, newRHS, intermediateSoln);
        if (success) p.right().apply(intermediateSoln, soln);
        return success;
    }
}
-----XEmacs: BICGSTABSolver.cpp (C++ Font)-----L37--18%
```

Example: creating a Kay-Loghin preconditioner

```
emacs: tmp4.cpp
File Edit View Cmds Tools Options Buffers C++ Help

TSFPreconditioner KayLoghinPreconditionerFactory
::createPreconditioner(const TSFLinearOperator& A) const
{
    TSFLinearOperator F = A.getBlock(0,0);

    TSFLinearOperator Mp = mpProb_.getOperator();
    TSFLinearOperator Ap = apProb_.getOperator();
    TSFLinearOperator Fp = fpProb_.getOperator();

    /* solve the F block and the auxiliary system Ap approximately */
    TSFLinearOperator Finv = F.inverse(fSolver_);
    TSFLinearOperator ApInv = Ap.inverse(apSolver_);

    TSFLinearOperator Xinv = MpInv*Fp*ApInv;

    TSFLinearOperator Bt = A.getBlock(0,1);
    TSFLinearOperator C = A.getBlock(1,0);

    TSFLinearOperator I00 = new TSFIdentityOperator(F.domain());
    TSFLinearOperator I11= new TSFIdentityOperator(Bt.domain());

    TSFLinearOperator P1 = new TSFBlockLinearOperator(A.domain(), A.range());
    TSFLinearOperator P2 = new TSFBlockLinearOperator(A.domain(), A.range());
    TSFLinearOperator P3 = new TSFBlockLinearOperator(A.domain(), A.range());

    P1.setBlock(0, 0, Finv);
    P1.setBlock(1, 1, I11);

    P2.setBlock(0, 0, I00);
    P2.setBlock(0, 1, Bt);
    P2.setBlock(1, 1, -I11);

    P3.setBlock(0, 0, I00);
    P3.setBlock(1, 1, Xinv);

    return new GenericRightPreconditioner(P1*P2*P3);
}

-----XEmacs: tmp4.cpp (C++ Font Abbrev)-----L19--All-----
```

Current status

; Concrete types

- » Epetra and LAPACK vectors and matrices

; Matrix-free types

- » block, sum, product, diagonal, zero, and identity operators

; Preconditioners

- » IFPack
- » User-defined block preconditioners

; Linear solvers

- » BICGSTAB, GMRES, CG, FCG, FGMRES
- » AztecOO
- » LAPACK direct
- » Schur complement, block triangular

; Nonlinear operators

- » general nonlinear operator interface
- » Composed operators

; Nonlinear solvers

- » Newton with line search
- » Picard

; Utilities

- » Controllable error handling and diagnostic reporting
- » reader/writer interfaces
 - » Matlab and Matrix Market readers and writers

Where is TSF being used?

- ; Sundance
 - » a high-level PDE simulation and optimization package (KL)
- ; Split/O3D
 - » a SQP optimization code (Paul Boggs)
- ; rSQP++
 - » a rSQP optimization code (Ross Bartlett)

Plans

; Development

- » Broader selection of components
 - AMG preconditioning
 - More solvers
- » Documentation

; Research

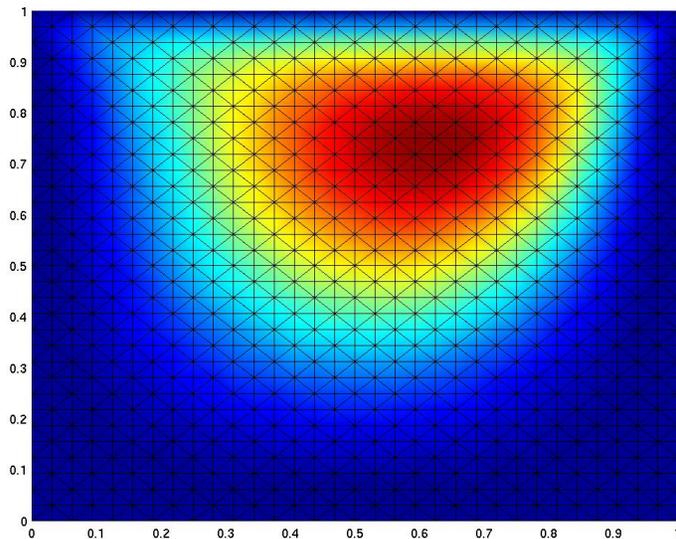
- » Block Krylov methods (Heroux & Barth)
- » Physics-based preconditioners (Howle, Heroux, KL, Tuminaro)
- » Fault tolerance (Howle, Hough)
- » PDE-constrained optimization (KL, Boggs, van Bloemen Waanders, Bartlett)
- » Preconditioners for inequality constraints (Boggs, Howle, Tuminaro)

Acknowledgements

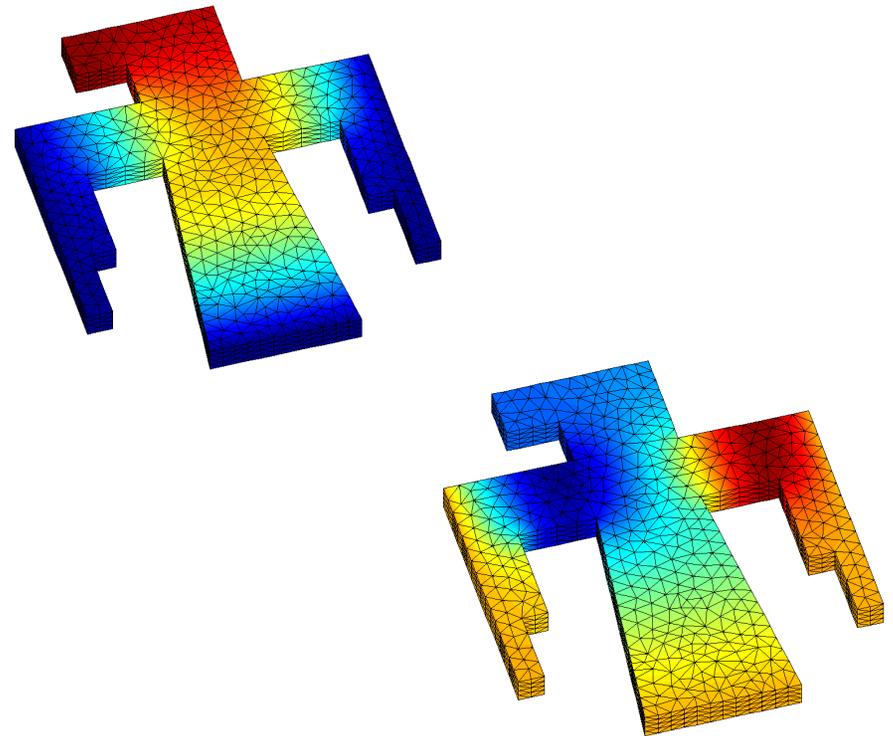
- ; Helpful design criticism and alpha testing from
 - » Paul Boggs, Ross Bartlett, Vicki Howle
- ; Some examples and tests from
 - » Mike Boldt
- ; Some solvers and preconditioners from
 - » Vicki Howle, George Biros
- ; EPetra vectors and matrices from
 - » Epetra/Trilinos development team

Some pretty pictures from Sundance/TSF

Contours of vorticity in a lid-driven cavity, computed using the Kay-Loghin block preconditioner on a Taylor-Hood discretization of Navier-Stokes



Temperature and sensitivity to Peclet number computed with a convection-diffusion model on the Sandia Thunderbird



Formulation and solution of a source inversion problem

- Simulation written by Omar Ghattas – 120 lines of Sundance code
- Advection-diffusion of a contaminant released from a Gaussian source
- Concentration is measured at 16 discrete sensor locations
- Adjoint formulation of inversion problem for distributed source field
- Ill-posed – smoothed with Tikhonov regularization
- Non-symmetric block reordering of adjoint and state variables

