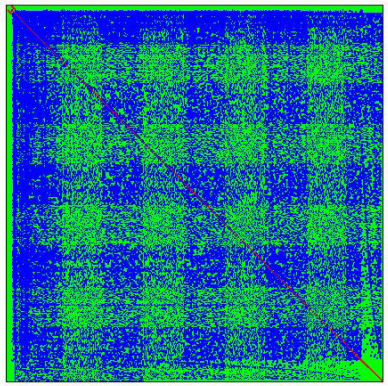
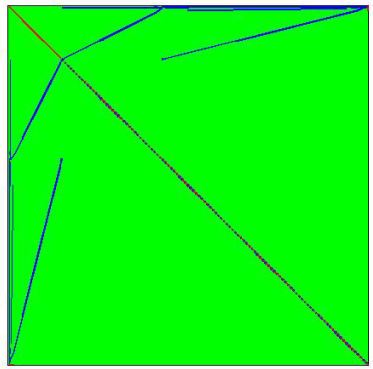
Enhanced Incomplete Factorizations Algorithms for Solving Circuit Modeling Problems





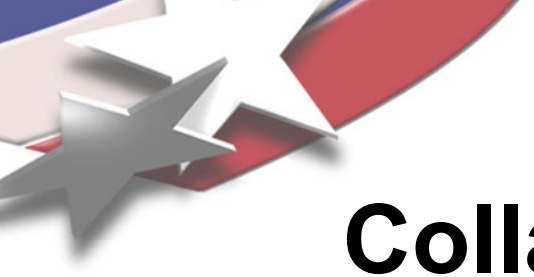
Michael A. Heroux

Numerical and Applied Mathematics Department

Sandia National Labs

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



Collaborators in This Work

- Trilinos Team:
  - David Day: Preconditioners and orderings.
  - Kevin Long: Abstract classes, aggregate preconditioners.
- Xyce Team:
  - Rob Hoekstra: Parallel data re-distribution.
  - Scott Hutchinson: Adaptive solver strategies.

Outline

Part 1: Adaptive Solver Strategies.

- **Solver Overview and Strategies.**
- **Preconditioners focus.**
- **Results.**

Part 2: TSF classes for aggregate preconditioners.

- **Overview.**
- **Components.**
- **Availability.**

Sandia National Laboratories

Trilinos Snapshot

About Trilinos[1]

- Sandia's multifaceted solver project.
- Encompasses efforts in these solver areas:
  - Linear (AztecOO, Komplex).
  - Preconditioners (ML, IFPACK).
  - Eigen (Anasazi).
  - Nonlinear and time dep (NOX).
- Petra matrix/vector classes:
  - All Trilinos solvers use Petra objects.
  - Solver components compatible, interoperable.
- TSF: Abstract Framework
  - Powerful aggregation capabilites.
  - Facilitates external component integration.

Trilinos Status

- Versions of Petra, AztecOO, Komplex, ML, IFPACK and NOX are available (under LGPL).
- AztecOO, Petra, IFPACK in:
  - Sierra, Xyce, Goma/Aria.
  - Others: TAO, Sundance,…
  - Available via ESI, FEI, CCA.
- ML: In ALEGRA, others.
- NOX: In MPSalsa, Xyce, FEAP, Salinas, Sierra, Goma/Aria.

- More information:

http://www.cs.sandia.gov/Trilinos

[1] Trilinos, pronounced tree-lee-nose, is a Greek word that, loosely translated, means a "string of pearls".

Sandia National Laboratories

Basic Solver Strategy

- **Domain decomposition:**
  - Overlapping additive Schwarz.
  - Problems with dense rows.
- **Scaling of entries.**
  - Typically row sum scaling.
  - Want to explore asymmetric row/col scaling.
- **Stabilized incomplete factorizations on subdomains.**
  - Primarily use ILUT on matrix *B*, where:
  - *B* is dual threshold *a priori* diagonal perturbation of original matrix *A.*
- **Non-restarted GMRES.**
  - Classical GS with refinement.
  - Scalable, very robust, potentially expensive.

Motivations for This Approach

- Heritage of writing solvers for CFD apps.
  - Want to leverage that investment.
- Direct sparse solvers are a problem:
  - Serial solvers are OK (not great).
  - Distributed memory solvers are problematic:
    - Third party codes are not well-supported (realize this point is contentious).
    - Little incentive to develop in-house expertise.
  - Most issues are related to software reliability, extensibility and maintainability.
- Machinery is just coming into place to implement KKT preconditioners.

Other Ideas Considered and Considering

- Textbook ILU and related preconditioners.
  - Failed.
- Sub-optimal iterative methods (e.g., Bi-CGSTAB).
  - Unreliable convergence.
- Column Reordering (Duff-Koster):
  - Used to find better pivots for ILUT.
  - Worked some times, but not as well as current approach.
- KKT preconditioners.
- Serial or moderately parallel direct solvers, even when running on many processors.

Preconditioned Krylov Solvers: Sources of Robustness Difficulties

- In our experience iterative solvers fail for two primary reasons:
  - Preconditioner is too weak:
    - Iterative methods stagnates or diverges.
    - Note: For large parallel computers this can often be fixed by increasing the fill of the factors. (But larger fill can lead to second type of problem…)
  - Preconditioner is too ill-conditioned:
    - We often find that tough problems need large-fill ILU factors.
    - However, these factors are often extremely ill-conditioned.

Ill-conditioned Factors

- Ill-conditioned factors come from two major sources:
  - Source 1: Well-conditioned, poorly ordered matrices.
    - These matrices are not poorly conditioned.
    - May have zero diagonal entries.
    - Problem: Straight-forward incomplete factorization implementations suffer.
      - Factorization fails (zero pivot).
      - Factors are poorly conditioned.
    - Remedy: Can usually be fixed with proper matrix reordering, either *a priori* or dynamic.
    - Used Duff-Koster here with some improvement.

Ill-conditioned Factors (cont)

- Source 2: Truly ill-conditioned problems.
  - Regardless of ordering or scaling, factorization of user matrix is ill-conditioned.
  - Observations (general):
    - Large-fill incomplete factorizations of ill-conditioned matrices will almost invariably be ill-conditioned.
    - In this case, the more accurate the factorization is the more ill-conditioned it is.
  - Remedy:
    - First we need to detect the problem.
      - At the completion of the factorization, compute a condition number estimate.

Estimating Preconditioner Accuracy

- Applying an ILU preconditioner involves solving a lower/upper triangular system $LUy = x$:

  Solve $Lz = x$.
  Solve $Uy = z$.

- The accuracy of this calculation is a function of:
  - Machine Precision = $2^{-53} \approx 10^{-16}$.
  - Condition number of the operator ($M = LU$ in this case).
  - *Cond(M)* approaching 10^{16} means useless result.

$$\frac{\|\hat{x} - x\|}{\|x\|} \approx \text{cond}(M) \quad \delta_{\text{mach}}$$

Sandia National Laboratories

Lower Bound on Inf-norm Condition Number

$$
\begin{aligned}
cond(M)_\infty &= \| M \|_\infty \| M^{-1} \|_\infty \\
&\geq \| M^{-1} \|_\infty \\
&= \max_{\|x\|_\infty = 1} \| M^{-1} x \|_\infty \\
&\geq \| M^{-1} \hat{x} \|_\infty, \qquad \hat{x} = (1, 1, \ldots, 1)^T \\
&= \| (LU)^{-1} \hat{x} \|_\infty
\end{aligned}
$$

- **Idea (not new):**
  - **Form vector *x* of all 1's.**
  - **Call *standard solve routine* to find y s.t. *LUy = x.***
  - **Find max abs value of *y* values.**
- **Result: A reasonable approximation for *cond(LU)* writing almost no extra software.**

Sandia National Laboratories

Remedies for Ill-conditioned Factors

- **First: Detect problem:**
  - condest() method computes lower bound on condition number.
  - condest() = 0 means unusable factorization: NaNs present.
  - condest() > threshold then declare factorization “bad”.

- **Next: Given bad factorization, attempt to make better:**
  - Define a modification of the original matrix that is better conditioned.
    - Modified matrix is “nearby” original matrix. Compute and use factorization of modified matrix.
  - Block matrix partitioning can help (not used for circuit problems).
    - We have block diagonal pivots.
    - Invert the blocks, thus use pivoting within the block.
    - Use SVD on block diagonal pivots.

A priori point diagonal perturbations

- Idea: Compute ILU factor of a matrix *B* that is "nearby" original matrix *A*, but better conditioned. (Generalization of Manteuffel shift)
- Sets up a continuum of preconditioners between accurate but poorly conditioned ILU factor and Jacobi scaling.
- *B* differs from *A* only on diagonal:

$$b_{ii} = \mathrm{sgn}(a_{ii})\alpha + (1+\rho)a_{ii}$$

- Forces:

$$|b_{ii}| \geq \alpha$$
$$|b_{ii}| \geq (1+\rho)|a_{ii}|$$

Choosing thresholds

- Now issue becomes: What values to pick for thresholds.
  - Too small: ILU factors are garbage.
  - Too large: Perturbed factors are poor approximations to original matrix.
- Answer: It's still an art.
- Partial solution: Semi-automated adaptive procedures.
- Ongoing work….

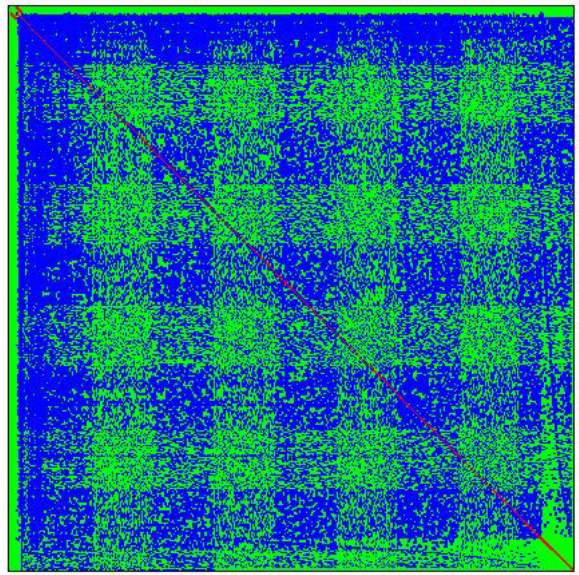
AdaptiveIterate() Method

```
AdaptiveIterate(int MaxIters,
                int MaxSolveAttempts,
                double Tolerance)

// Master "while" loop
while (!normalStop &&
       NumSolveAttempts<MaxSolveAttempts) {

  //Find preconditioner with condest < threshold

  while ( curTrial<NumTrials &&
          condest>=condestThreshold) {

    SetNextThresholdPair(curTrial);
    ConstructPreconditioner(condest); curTrial++;

  }

// Try solver now

Iterate(MaxIters, Tolerance);
NumSolveAttempts++;

If (!(oldResid>newResid)) X_ = Xold;

// Recompute prec if Krylov subspc ill-conditioned

while ( curTrial<NumTrials &&
        (illConditioned || breakdown)) {

   SetNextThresholdPair(curTrial);
   ConstructPreconditioner(condest); curTrial++;
   Iterate(MaxIters, Tolerance);
   NumSolveAttempts++;
   if (!(oldResid>newResid)) X_ = Xold;

  }

// Check for other failure exit states, fix if possible

if (lossOfOrthogonality)
      Try to solve one more time only…
   else if (maxits)  Try more robust preconditioner
      if (fill<curMaxFill)
        double amount of fill, reset curTrial = 0;
      else if (drop tolerance >0)
         set drop tolerance = 0.0; curTrial = 0;
      else if (kspace<curMaxKspace)
         double kspace and try again

} // End of Master “while” loop
```

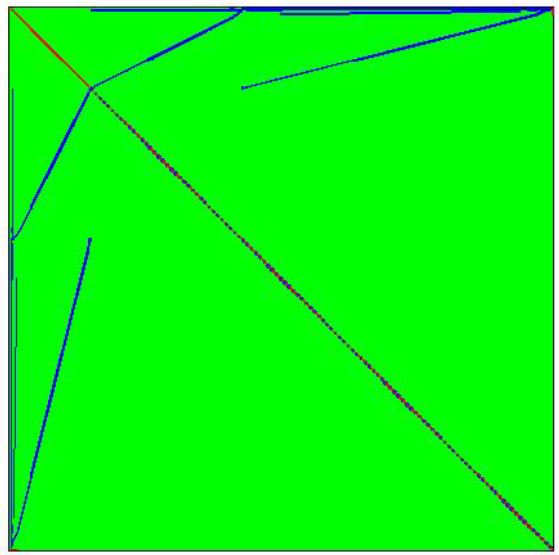
Sandia National Laboratories

Test Problems

- **RHP:**
  - **Eqs:25187**
  - **NNZ:258265**



- **Hutch3:**
  - **Eqs:32634**
  - **NNZ:153390**



Platform: Beowulf Cluster

Cluster specs:

- 8 nodes.
- Each node:
  - Uni-processor 1.4 GHz Athlons.
  - 1 GB DDR RAM.
- 100 Mbit/s network switch.
- Redhat Linux 7.1.
- GCC compilers.
- LAM MPI.

Sandia National Laboratories

Solvers

- **SuperLU:**
  - **Standard Serial Distribution.**
  - **Minimum Degree Reordering ($A^T + A$ pattern).**
- **AztecOO:**
  - **ILUT(r,d).**
    - **r – ratio of fill applied row-by-row. r = 1.0.**
    - **d – drop tolerance. d = 0.01.**
  - **Non-restarted GMRES, classical Gram-Schmidt with refinement.**
- **AztecOO with AdaptiveIterate Method:**
  - **Same as AztecOO but adapt to problems.**

Results (Time in secs)

SuperLU

Problem	Reordering Time	Factor Time	Solve Time	Total
RHP	5.2	8.7	0.18	14.4
Hutch3	8.1	0.39	0.06	8.6

AztecOO

Problem	Drop Tolerance	Iters (Res)	Fill	Time
RHP	10^{-2}	3 (10^{-12})	1.0	0.16
Hutch3	10^{-2}	2 (10^{-8})	1.0	0.12
Hutch3	0.0	NC (NaNs)	1.0	N/A

Adaptive Results:
Hutch3 (last case above)

Condest	Iters (Res)	Result	Action	Time
0.0 (NaNs in factor)	None	Condest not < Condest thresh	Set $\alpha = 10^{-12}$	
10^9	3 (10^{-8})	Converge		0.23

Note that matrix is singular (as is RHP):

$$\Delta x = x_{\text{superlu}} - x_{\text{aztecoo}}$$

$$|| \Delta x || = 10.4$$

$$|| A\Delta x || = 10^{-8}$$

Sandia National Laboratories

Parallel Adaptive Results (Hutch3: 2 and 8 PEs)

Max Condest	Iters (Res)	Result	Action	Time
inf	None	Condest not < Condest thresh	Set $\alpha = 10^{-12}$	
10^{12}	3 (10^{0})	Hessenberg Ill-conditioned	Set $\rho = 10^{-2}$	
10^{5}	11 (10^{-7})	Converged		0.55

Max Condest	Iters (Res)	Result	Action	Time
inf	None	Condest not < Condest thresh	Set $\alpha = 10^{-12}$	
10^{12}	1 (10^{0})	Hessenberg Ill-conditioned	Set $\rho = 10^{-2}$	
10^{2}	10 (10^{-7})	Converged		0.35

AdaptiveIterate Summary

- ILUT/GMRES can be effective for Xyce problems.
- Unpredictable behavior is chronic issue.
- Use of simple adaptive methods can minimize user intervention incidents.
- Issue becomes even more important in parallel.

Trilinos Solver Framework (TSF)

- **Epetra, AztecOO, Ifpack, ML, etc. PETSc, SuperLU, Hypre, HSL, ScaLapack**

  Lots of good solver components available

- **TSF is an abstract class hierarchy:**
  - **Provides uniform API to solvers.**
  - **Allows integration of many solvers via implementation of abstract classes.**
  - **Provides compositional classes.**
- **Similar to HCL.**

Sample TSF Class Category: TSFLinearOperator

Constructors

TSFLinearOperator ()
empty ctor constructs a null linear operator.

TSFLinearOperator (TSFLinearOperatorBase *ptr)
create a TSFLinearOperator from a pointer to a subtype.

Introspection methods

const TSFVectorSpace& domain () const
return domain space.

const TSFVectorSpace& range () const
return range space.

bool isZeroOperator () const
am I a zero operator?

Block structure information

int numBlockRows () const
get the number of block rows.

int numBlockCols () const
get the number of block columns.

TSFLinearOperator getBlock (int i, int j) const
get the (i,j)-th submatrix of a block operator.

TSFLinearOperator& setBlock (int i, int j, const TSFLinearOperator &sub)
set the (i,j)-th submatrix of a block

Application of forward, inverse, adjoint,and adjoint inverse operators

void apply (const TSFVector &arg, TSFVector &out) const
apply the operator to a vector.

void applyInverse (const TSFVector &arg, TSFVector &out) const
apply the operator's inverse to a vector.

void applyInverse (const TSFLinearSolver &solver, const TSFVector &arg, TSFVector &out) const
apply the operator's inverse to a vector, using the specified solver.

void applyAdjoint (const TSFVector &arg, TSFVector &out) const
apply the operator's adjoint to a vector.

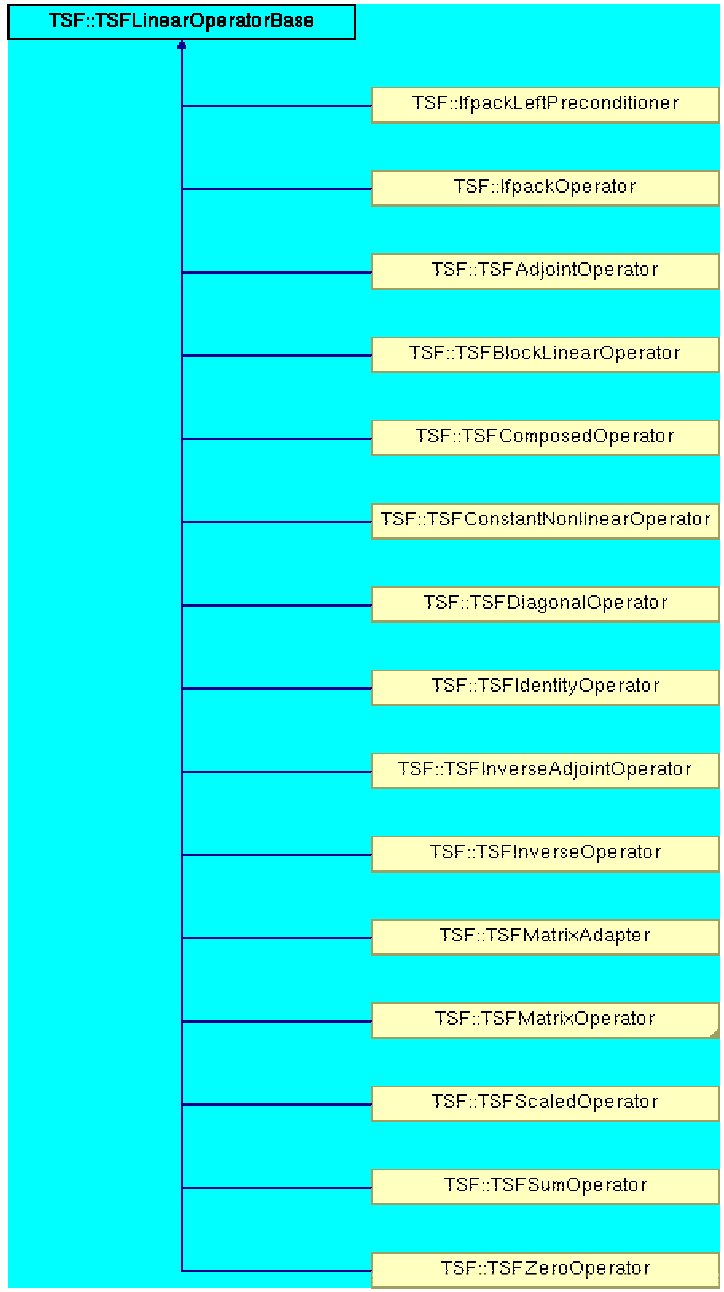
void applyInverseAdjoint (const TSFVector &arg, TSFVector &out) const
apply the operator's inverse to a vector.

void applyInverseAdjoint (const TSFLinearSolver &solver, const TSFVector &arg, TSFVector &out) const
apply the operator's inverse to a vector, using the specified solver.

Sandia National Laboratories

LinearOperator and LinearOperatorBase

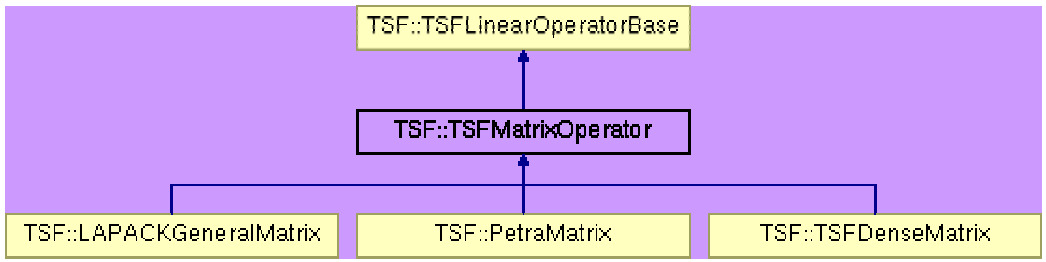
- TSFLinearOperator:
  - Has smart pointer to TSFLinearOperatorBase class.
  - Concrete class, but all methods implemented via identical methods in base class implementation.
- Advantages:
  - Memory management transparent.
  - Can use Matlab-like notation.
- Disadvantage:
  - Must manually maintain mirror set of methods.



Some LinearOperator Implementations

- **Concrete Implementations:**

  TSFPetraMatrix (const TSFVectorSpace &domain, const TSFVectorSpace &range)



- **Aggregate Implementations:**

  TSFComposedOperator (const TSFLinearOperator &left, const TSFLinearOperator &right)

  TSFInverseOperator (const TSFLinearOperator &op, const TSFLinearSolver &solver=TSFLinearSolver())

  TSFSumOperator (const TSFLinearOperator &left, const TSFLinearOperator &right, bool subtraction=false)

Sandia National Laboratories

Other Primary TSF Classes

- All Primary TSF classes follow similar model:
  - User class with smart pointer attribute.
  - Shadow base class.
  - One or more implementations of base class.
- Primary classes:
  - Vector Space.
  - Linear Operator.
  - Vector/MultiVector.
  - Linear Problem.

Aggregate Operator Construction

- **TSF facilitates implicit (and explicit) construction of operators:**
  - **Partitioned (block):** $A = \begin{bmatrix} F & B \\ C & 0 \end{bmatrix}$
  - **Composite:** $A = B \circ C$
  - **Sum:** $A = B + C$
  - **Inverse:** $A^{-1} = A \,\&\, solver(A)$
  - **Others: Zero, Identity, Transpose, …**
  - **Recursively.**

Aggregate Construction of Other Classes

- **Block Vectors/MultiVectors:** $x = \begin{bmatrix} y \\ z \end{bmatrix}$
- **Implicit Vectors/Multivectors:** $x = Ay$
- **Multi-tolerance stopping criteria:** $\left(\| r \| < \alpha_1 \right) \& \left(\frac{\| r \|}{\| r_0 \|} < \alpha_2 \right)$

Block Preconditioner Tools

- **Many systems are of the form:** $\overbrace{\begin{bmatrix} F & B \\ C & 0 \end{bmatrix}}^{A} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$

- **Note that:** $\begin{bmatrix} F & B \\ C & 0 \end{bmatrix} = \begin{bmatrix} I & 0 \\ CF^{-1} & I \end{bmatrix} \begin{bmatrix} F & B \\ 0 & S \end{bmatrix}, \quad S = -CF^{-1}B$

- **Precondition by:** $\begin{bmatrix} P_F & B \\ 0 & P_S \end{bmatrix}^{-1} = \overbrace{\begin{bmatrix} P_F^{-1} & 0 \\ 0 & I \end{bmatrix}}^{P_1} \overbrace{\begin{bmatrix} I & -B \\ 0 & I \end{bmatrix}}^{P_2} \overbrace{\begin{bmatrix} I & 0 \\ 0 & P_S^{-1} \end{bmatrix}}^{P_3}$

  where

  - P_F is your favorite preconditioner for F.
  - P_S is something special.

Sandia National Laboratories

Forming $\overbrace{\begin{bmatrix} P_F^{-1} & 0 \\ 0 & I \end{bmatrix}}^{P_1}\overbrace{\begin{bmatrix} I & -B \\ 0 & I \end{bmatrix}}^{P_2}\overbrace{\begin{bmatrix} I & 0 \\ 0 & P_S^{-1} \end{bmatrix}}^{P_3}$ for $\overbrace{\begin{bmatrix} F & B \\ C & 0 \end{bmatrix}}^{A}$ using TSF

```
KKTRightPreconditionerFactory::createPreconditioner(const TSFLinearOperator& A) {

    TSFLinearOperator F = A.getBlock(0,0);    // Pointer to upper left block of A
    TSFPreconditionerFactory pf = new ILUKPreconditionerFactory(1); // IFPACK parallel ILU(1)
    TSFPreconditioner fp = pf.createPreconditioner(F);
    TSFLinearOperator Finv = fp.left();

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

    TSFLinearOperator I00 = new TSFIdentityOperator(F.domain());
    TSFLinearOperator I11= new TSFIdentityOperator(Bt.domain());
    TSFLinearOperator Sinv = new TSFSomethingSpecial(C,A,B); // Schur compliment approximation

    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, -B);      P2.setBlock(1, 1, I11);

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

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

Sandia National Laboratories

Conclusions

- Direct solvers are essential but problematic:
  - No internal expertise (hard to justify).
  - How to develop/support special features in 3rd party codes?
  - Distributed solvers are (so far) unsatisfactory.
- Careful use of preconditioned Krylov methods can be useful:
  - Textbook varieties will fail.
  - Dealing with ill-conditioned factors is essential.
  - Good parallel robust GMRES is important.
  - Some kind of adaptive strategy seems essential.
- Aggregate preconditioners are next area of study:
  - Tools are in place.
  - Hope to learn about possible formulations at this meeting.

Trilinos and the Outside World

- **ESI (Equation Solver Interface):**
  de facto standard solver interface.
  - Epetra and AztecOO provided the first ESI-compliant implementation (thanks to Alan Williams).
- **TAO (Toolkit for Advanced Optimization):**
  Argonne optimization package.
  - Epetra/AztecOO are being used (via ESI interface) for TAO solver services, along with PETSc implementation of ESI.
- **CCA (Common Component Architecture):**
  Community effort to develop scientific SW components.
  - Epetra/AztecOO is a CCA solver component.
- **Public Release of Trilinos/Epetra:**
  - We will release Trilinos/Epetra this summer.
  - Using LGPL for licensing.
  - ML, IFPACK , AztecOO, Komplex, Anasazi , NOX will be (or are) going through the same release process.