SANDIA REPORT

SAND2007-xxxx
Unlimited Release
Printed August 2007

DRAFT: Meros User's Guide [a]

Victoria E. Howle, Robert Shuttleworth, Ray Tuminaro

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

[a] For **Meros**™ Version 2.0 in **Trilinos**™ Release 8.0

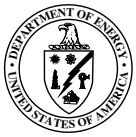
Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online



SAND2007-xxxx
Unlimited Release
Printed August 2007

DRAFT: Meros User's Guide [†]

Victoria E. Howle [‡] Robert Shuttleworth [§] Ray Tuminaro [¶]

Abstract

meros abstract

[†] For **Meros**[TM] Version 2.0 in **Trilinos**[TM] Release 8.0

[‡] Sandia National Laboratories, PO Box 969, MS 9159 Livermore, CA 94551, vehowle@sandia.gov.

[§] Applied Mathematics and Scientific Computing Program and Center for Scientific Computation and Mathematical Modeling, University of Maryland, College Park, MD 20742. rshuttle@math.umd.edu

[¶] Sandia National Laboratories, PO Box 969, MS 9159, Livermore, CA 94551, rstumin@sandia.gov.

Contents

Appendix

1 Getting Started

1.1 Downloading Meros and Trilinos

Get a local copy of Trilinos[1].

1.2 Licensing

1.3 Configuration, Compilation, and Installation

Using the instructions here, your build of trilinos will have the following libraries: aztecoo, epetra, epetraext, ML, newpackage, nox teuchos, and thyra.

- `cd` into the trilinos directory.
- Make a build directory, e.g., `mkdir LINUX`.
- `cd LINUX`.
- Configure trilinos:
  1. If you do not want to use MPI:

     ```
     { ../configure --enable-teuchos --enable-amesos}
     ```

  2. To use MPI:

     ```
     { ../configure  --enable-teuchos \\
       --enable-amesos --with-mpi-compilers=/usr/local/mpich/bin}
     ```

     where your path to the mpi-compilers is specified.
- Build trilinos: `make`.
- If your build finished without errors, you should see the directory `Trilinos/LINUX/packages/`, with subdirectories below that for each individual library. Meros's subdirectory, `meros`, should contain files `config.log`, `config.status`, `Makefile`, and `Makefile.export`, and directories `src` and `examples`. Directory `src` contains object files and `libmeros.a`. Directory `examples` contains executables with extension `.exe`, symbolic links to the corresponding source code, and object files. Directory `test` is intended primarily for developers and can be ignored.
- Look in `Trilinos/LINUX/packages/meros/examples` for examples of how to use Meros. File `Trilinos/packages/meros` suggests how to use the examples.

Sample configure script for Meros:

```
BUILD_DIR=‘pwd‘

../configure \
--enable-mpi \
```

[1] Please refer to the web page HTTP://SOFTWARE.SANDIA.GOV/TRILINOS on how to obtain a copy of trilinos.

```
--with-mpi-compilers="/usr/local/mpich-1.2.7_gcc3.4.3/bin" \
CXXFLAGS="-g -O0 -Wreturn-type" \
--with-libs="-lexpat" \
--disable-default-packages \
--enable-teuchos --enable-teuchos-extended \
--enable-teuchos-complex --enable-teuchos-abc --enable-teuchos-expat \
--enable-thyra \
--enable-thyra-examples \
--enable-epetra \
--enable-epetra-thyra \
--enable-epetraext \
--enable-epetraext-thyra \
--enable-ml \
--enable-ml-thyra \
--enable-aztecoo \
--enable-aztecoo-thyra \
--enable-ifpack \
--enable-ifpack-thyra \
--enable-nox \
--enable-stratimikos \
--enable-meros \
--enable-meros-examples \
--with-gnumake \
--with-install="/usr/bin/install -c -p" \
--prefix=$BUILD_DIR
```

2 Methods

2.1 Overview

Meros is a segregated preconditioning package within Trilinos [1]. Meros provides scalable block preconditioning for problems that couple simultaneous solution variables such as Navier-Strokes problems.

The initial focus of Meros is on preconditioners for the incompressible Navier-Stokes equations:

$$\begin{aligned} \eta \mathbf{u}_t - \nu \nabla^2 \mathbf{u} + (\mathbf{u} \cdot \text{grad})\,\mathbf{u} + \text{grad}\, p &= \mathbf{f} \\ -\text{div}\,\mathbf{u} &= 0 \end{aligned} \tag{1}$$

on $\Omega \subset \mathbb{R}^d$, $d = 2$ or 3. Here, $\mathbf{u}$ is the d-dimensional velocity field, which is assumed to satisfy suitable boundary conditions on $\partial\Omega$, p is the pressure, and ν is the kinematic viscosity, which is inversely proportional to the Reynolds number. The value $\eta = 0$ corresponds to the steady-state problem and $\eta = 1$ to the case of unsteady flow. Linearization and discretization of (1) by finite elements, finite differences or finite volumes leads to a sequence of linear systems of equations of the form

$$\begin{bmatrix} F & B^T \\ B & -C \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ p \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ g \end{bmatrix}. \tag{2}$$

These systems, which are the primary focus of the Meros preconditioners, must be solved at each step of a nonlinear (Picard or Newton) iteration, or at each time step. Here, B and B^T are matrices corresponding to discrete divergence and gradient operators, respectively and F operates on the discrete velocity space.

This guide describes the use of a block preconditioner within the Meros package. The block preconditioners can be used to solve block linear systems of the type in (2). We will denote the velocity and pressure degrees of freedom as v and p respectively.

The user must supply the four subblocks F, B^T, B, and C. If the matrix C is not provided, it is assumed to be the zero matrix. F is the convection-diffusion-like operator of size $v \times v$, B^T the pressure gradient of size $v \times p$, B the divergence operator of size $p \times v$, and C is a stabilization matrix of size $p \times p$. Depending on the discretization C might be the zero matrix. For *div-stable* discretizations, $C = 0$. For mixed approximation methods that do not uniformly satisfy a discrete inf-sup condition, the matrix C is a nonzero *stabilization operator*.

Block preconditioners a useful for a number of reasons:

- Want the scalability and mesh-independence of multigrid
- Difficult to apply multigrid to the whole system
- Segregate blocks and apply multigrid separately to subproblems
- Requires a good Schur complement approximation

Meros 1.0 includes the following classes of methods:

- Approximate Commutator Methods
    1. Pressure Convection-Diffusion (Fp) methods
    2. Least Squares Commutator (LSC) methods

- Pressure-Projection Methods
    1. SIMPLE - Semi Implicit Methods for Pressure Linked Equations
    2. SIMPLEC

This guide describes the use of a block preconditioner within the Meros package. The block preconditioners in Meros are designed for block linear systems of the type

$$A = \begin{bmatrix} F & B^T \\ B & C \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}$$

that usually arise from linearization and discretization of the incompressible Navier-Stokes equations. When using Meros, the sub matrices F, B, B^T, C and the vectors f and g are user supplied and u and p are vectors to be computed. Meros is intended to be used on large block sparse linear systems arising from partial differential equation (PDE) discretizations. While technically any block linear system can be considered, Meros should be used on linear systems that correspond to things that work well with multigrid methods (e.g. elliptic PDEs).

The methods in Meros are based on an LDU factorization of the saddlepoint system. The LDU factors of a saddle point system are given as follows:

$$\begin{bmatrix} A & B^T \\ B & C \end{bmatrix} = \begin{bmatrix} I & \\ BF^{-1} & I \end{bmatrix} \begin{bmatrix} F & \\ & -S \end{bmatrix} \begin{bmatrix} I & F^{-1}B^T \\ & I \end{bmatrix},$$

where S is the Schur complement $S = BF^{-1}B^T - C$.

2.2 Pressure Convection-Diffusion (PCD)

In this section we describe the factory for building a pressure convection-diffusion style block preconditioner. This class of preconditioners were originally proposed by Kay, Loghin, and Wathen [2] and Silvester, Elman, Kay, and Wathen [3].

Meros currently implements the PCD preconditioner, a.k.a. Fp preconditioner.

The LDU factors of a saddle point system are given as follows:

$$\begin{bmatrix} A & B^T \\ B & C \end{bmatrix} = \begin{bmatrix} I & \\ BF^{-1} & I \end{bmatrix} \begin{bmatrix} F & \\ & -S \end{bmatrix} \begin{bmatrix} I & F^{-1}B^T \\ & I \end{bmatrix}, \tag{3}$$

where S is the Schur complement $S = BF^{-1}B^T - C$. A pressure convection-diffusion style preconditioner is then given by

$$P^{-1} = \begin{bmatrix} F & B^T \\ & -\tilde{S} \end{bmatrix}^{-1} = \begin{bmatrix} F^{-1} & \\ & I \end{bmatrix} \begin{bmatrix} I & -B^T \\ & I \end{bmatrix} \begin{bmatrix} I & \\ & -\tilde{S}^{-1} \end{bmatrix} \tag{4}$$

where for $\tilde{S}$ is an approximation to the Schur complement S.

To apply the above preconditioner, we need a linear solver on the (0,0) block and an approximation to the inverse of the Schur complement.

To build a concrete preconditioner object, we will also need a 2x2 block Thyra matrix or the 4 separate blocks as either Thyra or Epetra matrices. If Thyra, assumes each block is a Thyra EpetraMatrix.

2.3 Least Squares Commutator (LSC)

Factory for building least squares commutator style block preconditioner.

Note that the LSC preconditioner assumes that we are using a stable discretization an a uniform mesh.

The LDU factors of a saddle point system are given as follows:

$$\begin{bmatrix} A & B^T \\ B & C \end{bmatrix} = \begin{bmatrix} I & \\ BF^{-1} & I \end{bmatrix} \begin{bmatrix} F & \\ & -S \end{bmatrix} \begin{bmatrix} I & F^{-1}B^T \\ & I \end{bmatrix}, \tag{5}$$

where S is the Schur complement $S = BF^{-1}B^T - C$. A pressure convection-diffusion style preconditioner is then given by

$$P^{-1} = \begin{bmatrix} F & B^T \\ & -\tilde{S} \end{bmatrix}^{-1} = \begin{bmatrix} F^{-1} & \\ & I \end{bmatrix} \begin{bmatrix} I & -B^T \\ & I \end{bmatrix} \begin{bmatrix} I & \\ & -\tilde{S}^{-1} \end{bmatrix} \tag{6}$$

where for $\tilde{S}$ is an approximation to the Schur complement S.

To apply the above preconditioner, we need a linear solver on the (0,0) block and an approximation to the inverse of the Schur complement.

To build a concrete preconditioner object, we will also need a 2x2 block Thyra matrix or the 4 separate blocks as either Thyra or Epetra matrices. If Thyra, assumes each block is a Thyra EpetraMatrix.

2.4 SIMPLE

3 Examples

In this section, we go through several examples of using Meros as a preconditioner.

3.1 PCD Example

3.2 LSC Example

```
// @HEADER
// ************************************************************************
//
//              Meros: Segregated Preconditioning Package
//                 Copyright (2004) Sandia Corporation
//
// Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
// license for use of this work by or on behalf of the U.S. Government.
//
// This library is free software; you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as
// published by the Free Software Foundation; either version 2.1 of the
// License, or (at your option) any later version.
//
// This library is distributed in the hope that it will be useful, but
// WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
// Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public
// License along with this library; if not, write to the Free Software
// Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
// USA
// Questions? Contact Michael A. Heroux (maherou@sandia.gov)
//
// ************************************************************************
// @HEADER

// saddle_lsc.cpp

// Example program that reads in Epetra matrices from files, creates a
// Meros Least Squares Commutator (LSC) preconditioner and does a
// solve.

#include "Teuchos_ConfigDefs.hpp"
#include "Teuchos_MPISession.hpp"
#include "Teuchos_GlobalMPISession.hpp"
#include "Teuchos_DefaultComm.hpp"
#include "Teuchos_ParameterList.hpp"
#include "Teuchos_ParameterXMLFileReader.hpp"
```

```
#include "Teuchos_RefCountPtr.hpp"

#include "Thyra_SolveSupportTypes.hpp"
#include "Thyra_LinearOpBase.hpp"
#include "Thyra_LinearOpBaseDecl.hpp"
#include "Thyra_VectorDecl.hpp"
#include "Thyra_VectorImpl.hpp"
#include "Thyra_VectorSpaceImpl.hpp"
#include "Thyra_LinearOperatorDecl.hpp"
#include "Thyra_LinearOperatorImpl.hpp"
#include "Thyra_SpmdVectorBase.hpp"
#include "Thyra_DefaultZeroLinearOp.hpp"
#include "Thyra_DefaultBlockedLinearOpDecl.hpp"
#include "Thyra_LinearOpWithSolveFactoryHelpers.hpp"
#include "Thyra_PreconditionerFactoryHelpers.hpp"
#include "Thyra_DefaultInverseLinearOpDecl.hpp"
#include "Thyra_DefaultInverseLinearOp.hpp"
#include "Thyra_PreconditionerFactoryBase.hpp"
#include "Thyra_DefaultPreconditionerDecl.hpp"
#include "Thyra_DefaultPreconditioner.hpp"
#include "Thyra_SingleRhsLinearOpWithSolveBase.hpp"
#include "Thyra_AztecOOLinearOpWithSolveFactory.hpp"

#ifdef HAVE_MPI
#include "Epetra_MpiComm.h"
#include "mpi.h"
#else
#include "Epetra_SerialComm.h"
#endif

#include "Epetra_Comm.h"
#include "Epetra_Map.h"
#include "Epetra_CrsMatrix.h"
#include "Epetra_Vector.h"

#include "EpetraExt_CrsMatrixIn.h"
#include "EpetraExt_VectorIn.h"

#include "Thyra_EpetraLinearOp.hpp"
#include "Thyra_EpetraThyraWrappers.hpp"

#include "AztecOO.h"
#include "Thyra_AztecOOLinearOpWithSolveFactory.hpp"
#include "Thyra_AztecOOLinearOpWithSolve.hpp"

#include "Meros_ConfigDefs.h"
#include "Meros_Version.h"
#include "Meros_LSCPreconditionerFactory.h"
#include "Meros_LSCOperatorSource.h"
#include "Meros_AztecSolveStrategy.hpp"
#include "Meros_InverseOperator.hpp"
```

```
#include "Meros_ZeroOperator.hpp"
#include "Meros_IdentityOperator.hpp"
#include "Meros_LinearSolver.hpp"

using namespace Teuchos;
using namespace EpetraExt;
using namespace Thyra;
using namespace Meros;


int main(int argc, char *argv[])
{
  GlobalMPISession mpiSession(&argc, &argv);

  // DEBUG 0 = no extra tests or printing
  // DEBUG 1 = print out basic diagnostics as we go.
  //           prints outer saddle system iterations, but not inner solves
  // DEBUG > 1 = test usage of some of the operators before proceeding
  //           prints inner and outer iterations
  int DEBUG = 1;

  // Get stream that can print to just root or all streams!
  Teuchos::RefCountPtr<Teuchos::FancyOStream>
    out = Teuchos::VerboseObjectBase::getDefaultOStream();

  //  Epetra_Comm* Comm;
#ifdef HAVE_MPI
  Epetra_MpiComm Comm(MPI_COMM_WORLD);
  const int myRank = Comm.MyPID();
  const int numProcs = Comm.NumProc();
#else
  Epetra_SerialComm Comm;
  const int myRank = 0;
  const int numProcs = 1;
#endif

  if(DEBUG > 0)
    {
      if (myRank == 0)
{
  cout << "Proc " << myRank << ": "
       << "Number of processors = "
       << numProcs << endl;
  cout << "Proc " << myRank << ": "
       << Meros::Meros_Version() << endl;
}
    }

  try
    {
      /* ------ Read in epetra matrices and rhs ------------------- */
```

```
// Using Q1 matrix example; see Meros/examples/data/q1
// (2,2 block (C block) is zero in this example)

// Make necessary Epetra maps.
// Need a velocity space map and a pressure space map.
const Epetra_Map* velocityMap = new Epetra_Map(578, 0, Comm);
const Epetra_Map* pressureMap = new Epetra_Map(192, 0, Comm);


// Read matrix and vector blocks into Epetra_Crs matrices
Epetra_CrsMatrix* FMatrix(0);
char * filename = "../../../../../packages/meros/example/data/q1/Aq1.mm";
MatrixMarketFileToCrsMatrix(filename,
                            *velocityMap,
                            *velocityMap, *velocityMap,
                            FMatrix);

// cerr << "If you get Epetra ERROR -1 here, the file was not found"
//  << "Need to make symbolic link to data directory"
//  << endl;

Epetra_CrsMatrix* BtMatrix(0);
filename = "../../../../../packages/meros/example/data/q1/Btq1.mm";
MatrixMarketFileToCrsMatrix(filename,
                            *velocityMap,
                            *velocityMap, *pressureMap,
                            BtMatrix);

Epetra_CrsMatrix* BMatrix(0);
filename = "../../../../../packages/meros/example/data/q1/Bq1.mm";
MatrixMarketFileToCrsMatrix(filename,
                            *pressureMap,
                            *pressureMap, *velocityMap,
                            BMatrix);


Epetra_Vector* rhsq1_vel(0);
filename = "../../../../../packages/meros/example/data/q1/rhsq1_vel.mm";
MatrixMarketFileToVector(filename,
                         *velocityMap,
                         rhsq1_vel);

Epetra_Vector* rhsq1_press(0);
filename = "../../../../../packages/meros/example/data/q1/rhsq1_press.mm";
MatrixMarketFileToVector(filename,
                         *pressureMap,
                         rhsq1_press);

FMatrix->FillComplete();
BtMatrix->FillComplete(*pressureMap, *velocityMap);
BMatrix->FillComplete(*velocityMap, *pressureMap);
```

```
// Wrap Epetra operators into Thyra operators. To do this, we
// first wrap as Thyra core operators, then convert to the
// handle layer LinearOperators.

RefCountPtr<LinearOpBase<double> >
  tmpF = rcp(new EpetraLinearOp(rcp(FMatrix,false)));
const LinearOperator<double> F = tmpF;

RefCountPtr<LinearOpBase<double> >
  tmpBt = rcp(new EpetraLinearOp(rcp(BtMatrix,false)));
const LinearOperator<double> Bt = tmpBt;

RefCountPtr<LinearOpBase<double> >
  tmpB = rcp(new EpetraLinearOp(rcp(BMatrix,false)));
const LinearOperator<double> B = tmpB;


// Wrap Epetra vectors into Thyra vectors similarly by first
// wrapping to the Thyra core vector layer, then converting to
// the handle layer

RefCountPtr<const Thyra::VectorSpaceBase<double> > epetra_vs_press
  = Thyra::create_VectorSpace(rcp(pressureMap,false));
RefCountPtr<const Thyra::VectorSpaceBase<double> > epetra_vs_vel
  = Thyra::create_VectorSpace(rcp(velocityMap,false));


RefCountPtr<VectorBase<double> > rhs1
  = create_Vector(rcp(rhsq1_press, false), epetra_vs_press);
RefCountPtr<VectorBase<double> > rhs2
  = create_Vector(rcp(rhsq1_vel, false), epetra_vs_vel);

// Convert the vectors to handled vectors
RefCountPtr<VectorBase<double> > tmp1 = rhs1;
const Vector<double> rhs_press = tmp1;

RefCountPtr<VectorBase<double> > tmp2 = rhs2;
const Vector<double> rhs_vel = tmp2;


if(DEBUG > 1){
  // Test the wrapped operators
  cerr << "F matrix: description  " << F.description() << endl;
  cerr << "F domain: " << F.domain().dim()
       << ",  F range: " << F.range().dim()
       << endl;
  cerr << "B matrix: description  " << B.description() << endl;
  cerr << "B domain: " << B.domain().dim()
       << ",  B range: " << B.range().dim()
       << endl;
```

```
        cerr << "Bt matrix: description  " << Bt.description() << endl;
        cerr << "Bt domain: " << Bt.domain().dim()
             << ",  Bt range: " << Bt.range().dim()
             << endl;

        Vector<double> testvec1 = Bt * rhs_press;
        cerr << "Bt * rhs_press = " << norm2(testvec1) <<endl;

cerr << "domain size of B " << B.domain().dim() << endl;
cerr << "range size of B " << B.range().dim() << endl;
cerr << "size of testvec1 " << dim(testvec1) << endl;

        Vector<double> testvec2 = B * testvec1;
        cerr << "B * Bt * rhs_press = " << norm2(testvec2) <<endl;

        Vector<double> testvec3 = F * rhs_vel;
        cerr << "F * rhs_vel = " << norm2(testvec3) <<endl;

        Vector<double> testvec4 = B * F * Bt * rhs_press;
        cerr << "BFBt * rhs_press = " << norm2(testvec4) <<endl;

      }



      // Get Thyra velocity and pressure spaces from the operators.
      VectorSpace<double> velocitySpace = F.domain();
      VectorSpace<double> pressureSpace = Bt.domain();

      if(DEBUG > 0){
        cerr << "P" << myRank
             << ": vel space dim = " << velocitySpace.dim() << endl;
        cerr << "P" << myRank
             << ": press space dim = " << pressureSpace.dim() << endl;
      }

      // Make a zero operator on the small (pressure) space since the
      // 2,2 block (C) is zero in this example.
      // RefCountPtr<Thyra::LinearOpBase<double> > tmpZ =
      // rcp(new DefaultZeroLinearOp<double>(tmpBt->domain(),
      //                                     tmpBt->domain()));
      // const LinearOperator<double> Z = tmpZ;
      const LinearOperator<double> Z;

      // Build the block saddle operator with F, Bt, B, and Z.  This
      // operator will be blockOp = [F Bt; B zero] (in Matlab
      // notation).
      LinearOperator<double> blockOp = block2x2(F, Bt, B, Z);

      if(DEBUG > 1)
        {
```

```
    // Test getting subblock components out of the block operator.
    LinearOperator<double> testF1 = blockOp.getBlock(0,0);
    LinearOperator<double> testBt1 = blockOp.getBlock(0,1);
    LinearOperator<double> testB1 = blockOp.getBlock(1,0);
    cerr << "Checking blocks after extracting them out the block op"
         << endl;
    cerr << "testF domain: " << testF1.domain().dim() << endl;
    cerr << "testF range: " << testF1.range().dim() << endl;
    cerr << "testBt domain: " << testBt1.domain().dim() << endl;
    cerr << "testBt range: " << testBt1.range().dim() << endl;
    cerr << "testB domain: " << testB1.domain().dim() << endl;
    cerr << "testB range: " << testB1.range().dim() << endl;

    Vector<double> testvec1 = testBt1 * rhs_press;
    cerr << "Bt * rhs_press = " << norm2(testvec1) <<endl;

    Vector<double> testvec2 = testB1 * testvec1;
    cerr << "B * Bt * rhs_press = " << norm2(testvec2) <<endl;

    Vector<double> testvec3 = testF1 * rhs_vel;
    cerr << "F * rhs_vel = " << norm2(testvec3) <<endl;

    Vector<double> testvec4 = testB1 * testF1 * testBt1 * rhs_press;
    cerr << "BFBt * rhs_press = " << norm2(testvec4) <<endl;
  }

// Get the domain and range product spaces from the block operator.
VectorSpace<double> domain = blockOp.domain();
VectorSpace<double> range = blockOp.range();

// Build a product vector for the rhs and set the velocity and
// pressure components of the rhs with the vectors we previously
// read in from files and wrapped in Thyra.
Vector<double> rhs = range.createMember();
rhs.setBlock(0,rhs_vel);
rhs.setBlock(1,rhs_press);

// Build a solution vector and initialize it to zero.
Vector<double> solnblockvec = domain.createMember();
zeroOut(solnblockvec);


// We now have Thyra block versions of the saddle point system,
// rhs, and solution vector. Next we build the Meros LSC
// preconditioner.


/* -------- Build the Meros preconditioner factory ---------*/

// Build a Least Squares Commutator (LSC) block preconditioner
// with Meros
```

```
//
// | inv(F) 0 | | I  -Bt | | I        |
// | 0      I | |     I  | |   -inv(X)|
//
// where inv(X) = inv(B*Bt) * B * F * Bt * inv(B*Bt)
// (velocity mass matrix is the identity in this example)
//
// We'll do this in 4 steps:
// 1) Build an AztecOO ParameterList for inv(F) solve
// 2) Build an AztecOO ParameterList for inv(B*Bt) solve
//    The Schur complement approximation inverse requires solves
//    on the composed operator B*Bt.
// 3) Make an LSCOperatorSource with blockOp  (and Qu if needed)
// 4) Build the LSC block preconditioner factory


// 1) Build an AztecOO ParameterList for inv(F) solve
//    This one corresponds to (unpreconditioned) GMRES

RefCountPtr<ParameterList>
  aztecFParams = rcp(new ParameterList("aztecOOFSolverFactory"));

RefCountPtr<LinearOpWithSolveFactoryBase<double> > aztecFLowsFactory;

if(DEBUG> 1)
  {
    // Print out valid parameters and the existing default params.
    aztecFLowsFactory = rcp(new AztecOOLinearOpWithSolveFactory());
    cerr << "\naztecFLowsFactory.getValidParameters():\n" << endl;
    aztecFLowsFactory->getValidParameters()->print(cerr, 0, true, false);
    cerr << "\nPrinting initial parameters. " << endl;
    aztecFLowsFactory->setParameterList(aztecFParams);
    aztecFLowsFactory->getParameterList()->print(cerr, 0, true, false);
  }


// forward solve settings
aztecFParams->sublist("Forward Solve").set("Max Iterations", 100);
aztecFParams->sublist("Forward Solve").set("Tolerance", 10e-8);
// aztecOO solver settings
aztecFParams->sublist("Forward Solve")
  .sublist("AztecOO Settings").set("Aztec Solver", "GMRES");
aztecFParams->sublist("Forward Solve")
  .sublist("AztecOO Settings").set("Aztec Preconditioner", "none");
aztecFParams->sublist("Forward Solve")
  .sublist("AztecOO Settings").set("Size of Krylov Subspace", 100);


if(DEBUG > 1)
  {
    // turn on AztecOO output
```

```
    aztecFParams->sublist("Forward Solve")
      .sublist("AztecOO Settings").set("Output Frequency", 10);
  }

if(DEBUG > 1)
  {
    // Print out the parameters we just set
    aztecFLowsFactory->setParameterList(aztecFParams);
    aztecFLowsFactory->getParameterList()->print(cerr, 0, true, false);
  }


// 2) Build an AztecOO ParameterList for inv(Ap) solve
//    This one corresponds to unpreconditioned CG.

RefCountPtr<ParameterList>
  aztecBBtParams = rcp(new ParameterList("aztecOOBBtSolverFactory"));

// forward solve settings
aztecBBtParams->sublist("Forward Solve").set("Max Iterations", 100);
aztecBBtParams->sublist("Forward Solve").set("Tolerance", 10e-8);
// aztecOO solver settings
aztecBBtParams->sublist("Forward Solve")
  .sublist("AztecOO Settings").set("Aztec Solver", "CG");
aztecBBtParams->sublist("Forward Solve")
  .sublist("AztecOO Settings").set("Aztec Preconditioner", "none");


if(DEBUG > 1)
  {
    // turn on AztecOO output
    aztecBBtParams->sublist("Forward Solve")
      .sublist("AztecOO Settings").set("Output Frequency", 10);
  }


if(DEBUG > 1)
  {
    // Print out the parameters we just set
    RefCountPtr<LinearOpWithSolveFactoryBase<double> >
      aztecBBtLowsFactory = rcp(new AztecOOLinearOpWithSolveFactory());
    aztecBBtLowsFactory->setParameterList(aztecBBtParams);
    aztecBBtLowsFactory->getParameterList()->print(cerr, 0, true, false);
  }


// 3) Make an LSCOperatorSource that contains  blockOp
//    The velocity mass matrix Qu is the identity in this example.
RefCountPtr<const LinearOpSourceBase<double> > myLSCopSrcRcp
  = rcp(new LSCOperatorSource(blockOp));
```

```
      if(DEBUG > 1)
        {
          // Test getting subblock components out of the operator source
          RefCountPtr<const LSCOperatorSource> lscOpSrcPtr
            = rcp_dynamic_cast<const LSCOperatorSource>(myLSCopSrcRcp);

          // Retrieve operators from the LSC operator source
          RefCountPtr<const LinearOpBase<double> > tmpBlockOp
            = lscOpSrcPtr->getOp();
          ConstLinearOperator<double> blockOp = tmpBlockOp;
          ConstLinearOperator<double> testF2 = blockOp.getBlock(0,0);
          ConstLinearOperator<double> testBt2 = blockOp.getBlock(0,1);
          ConstLinearOperator<double> testB2 = blockOp.getBlock(1,0);

          cerr << "Checking blocks after extracting them out the block op"
               << endl;
          cerr << "testF domain: " << testF2.domain().dim() << endl;
          cerr << "testF range: " << testF2.range().dim() << endl;
          cerr << "testBt domain: " << testBt2.domain().dim() << endl;
          cerr << "testBt range: " << testBt2.range().dim() << endl;
          cerr << "testB domain: " << testB2.domain().dim() << endl;
          cerr << "testB range: " << testB2.range().dim() << endl;

          Vector<double> testvec11 = testBt2 * rhs_press;
          cerr << "Bt * rhs_press = " << norm2(testvec11) <<endl;

          Vector<double> testvec22 = testB2 * testvec11;
          cerr << "B * Bt * rhs_press = " << norm2(testvec22) <<endl;

          Vector<double> testvec33 = testF2 * rhs_vel;
          cerr << "F * rhs_vel = " << norm2(testvec33) <<endl;

          Vector<double> testvec44 = testB2 * testF2 * testBt2 * rhs_press;
          cerr << "BFBt * rhs_press = " << norm2(testvec44) <<endl;
        }


      // 4) Build the LSC block preconditioner factory.
      RefCountPtr<PreconditionerFactoryBase<double> > merosPrecFac
        = rcp(
      new LSCPreconditionerFactory(
rcp(new Thyra::AztecOOLinearOpWithSolveFactory(aztecFParams)),
rcp(new Thyra::AztecOOLinearOpWithSolveFactory(aztecBBtParams))
)
      );

      RefCountPtr<PreconditionerBase<double> > Prcp
        = merosPrecFac->createPrec();

      merosPrecFac->initializePrec(myLSCopSrcRcp, &*Prcp);
```

```
// Checking that isCompatible and uninitializePrec at least
// compile and throw the intended exceptions for now.
// merosPrecFac->uninitializePrec(&*Prcp, &myLSCopSrcRcp);
// bool passed;
// passed = merosPrecFac->isCompatible(*(myLSCopSrcRcp.get()));


/* --- Now build a solver factory for outer saddle point problem --- */

// Set up parameter list and AztecOO solver
RefCountPtr<ParameterList> aztecSaddleParams
  = rcp(new ParameterList("aztecOOSaddleSolverFactory"));

RefCountPtr<LinearOpWithSolveFactoryBase<double> >
  aztecSaddleLowsFactory = rcp(new AztecOOLinearOpWithSolveFactory());

double saddleTol = 1.0e-6;

// forward solve settings
aztecSaddleParams->sublist("Forward Solve").set("Max Iterations", 500);
aztecSaddleParams->sublist("Forward Solve").set("Tolerance", saddleTol);
// aztecOO solver settings
aztecSaddleParams->sublist("Forward Solve")
  .sublist("AztecOO Settings").set("Aztec Solver", "GMRES");
aztecSaddleParams->sublist("Forward Solve")
  .sublist("AztecOO Settings").set("Aztec Preconditioner", "none");
aztecSaddleParams->sublist("Forward Solve")
  .sublist("AztecOO Settings").set("Size of Krylov Subspace", 500);

if(DEBUG > 0)
  {
    // turn on AztecOO output
    aztecSaddleParams->sublist("Forward Solve")
      .sublist("AztecOO Settings").set("Output Frequency", 1);
  }

aztecSaddleLowsFactory->setParameterList(aztecSaddleParams);

if(DEBUG > 0)
  {
    // Print out the parameters we've set.
    aztecSaddleLowsFactory->getParameterList()->print(cerr, 0,
                                                      true, false);
  }



// Set up the preconditioned inverse object and do the solve!
RefCountPtr<LinearOpWithSolveBase<double> > rcpAztecSaddle
  = aztecSaddleLowsFactory->createOp();
```

```
      // LinearOperator<double> epetraBlockOp = makeEpetraOperator(blockOp);

      // initializePreconditionedOp<double>(*aztecSaddleLowsFactory,
      //  epetraBlockOp.ptr(),
      //  Prcp,
      //  &*rcpAztecSaddle );

      //       initializePreconditionedOp<double>(*aztecSaddleLowsFactory,
      //   blockOp.ptr(),
      //   Prcp,
      //   &*rcpAztecSaddle );

      //       RefCountPtr<LinearOpBase<double> > tmpSaddleInv
      //  = rcp(new DefaultInverseLinearOp<double>(rcpAztecSaddle));

      //       LinearOperator<double> saddleInv = tmpSaddleInv;
      //       saddleInv.description();

      RefCountPtr<const LinearOpBase<double> > tmpPinv
        = Prcp->getRightPrecOp();
      ConstLinearOperator<double> Pinv = tmpPinv;

      LinearSolveStrategy<double> azSaddle
        = new AztecSolveStrategy(*(aztecSaddleParams.get()));

      ConstLinearOperator<double> saddleInv
= new InverseOperator<double>(blockOp * Pinv, azSaddle);

      // Do the solve!
      solnblockvec = saddleInv * rhs;

      // Check our results.
      Vector<double> residvec = blockOp * Pinv * solnblockvec - rhs;

      cerr << "norm of resid " << norm2(residvec) << endl;

      double normResvec = norm2(residvec);

      if(normResvec < 10.0*saddleTol)
{
  cerr << "Example PASSED!" << endl;
  return 0;
}
      else
{
  cerr << "Example FAILED!" << endl;
  return 1;
}
```

```
    } // end of try block


  catch(std::exception& e)
    {
      cerr << "Caught exception: " << e.what() << endl;
    }

  MPISession::finalize();

} // end of main()
```

3.3 SIMPLE Example

A Bug Reporting and Enhancement Requests

B Mailing Lists

References

[1] M. HEROUX, R. BARTLETT, V. HOWLE, R. HOEKSTRA, J. HU, T. KOLDA, R. LEHOUCQ, K. LONG, R. PAWLOWSKI, E. PHIPPS, A. SALINGER, H. THORNQUIST, R. TUMINARO, J. WILLENBRING, AND A. WILLIAMS, *An Overview of Trilinos*, Tech. Rep. SAND2003-2927, Sandia National Laboratories, 2003.

[2] D. KAY, D. LOGHIN, AND A. WATHEN, *A preconditioner for the steady-state Navier–Stokes equations*, SIAM J. Sci. Comput., 24 (2002), pp. 237–256.

[3] D. SILVESTER, H. ELMAN, D. KAY, AND A. WATHEN, *Efficient preconditioning of the linearized Navier–Stokes equations for incompressible flow*, J. Comp. Appl. Math., 128 (2001), pp. 261–279.


Sandia National Laboratories