SAND2016-7153C

Exceptional service in the national interest

Sandia National Laboratories

Block Preconditioners for Mixed Discretization MHD and Continuum Plasma Simulations Enabled by Teko

CCR
Center for Computing Research

Edward Phillips, Eric Cyr, John Shadid, and Ray Tuminaro

U.S. DEPARTMENT OF ENERGY
NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND2016-5003 C

Computational Challenges in Multiphysics

Sandia National Laboratories

Multiphysics systems are characterized by a myriad of complex, interacting, nonlinear multiple time- and length-scale physical mechanisms.

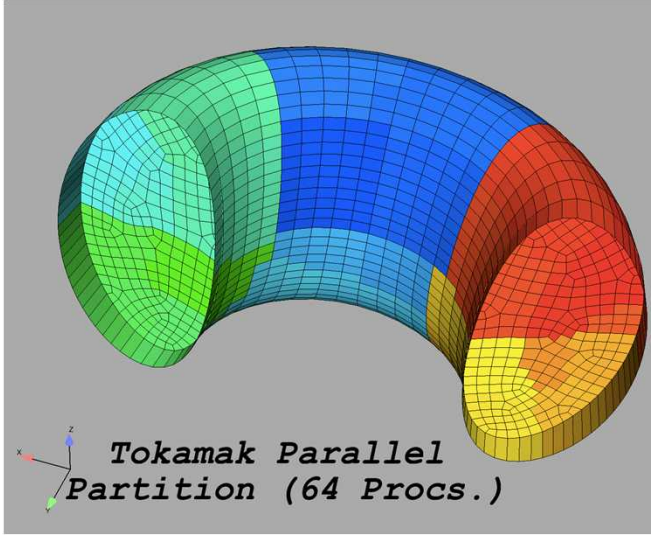
These mechanisms can:

- be dominated by one, or a few processes, that drive a short dynamical time-scale consistent with these dominating modes,
- consist of a set of widely separated time-scales that produce a stiff system response,
- nearly balance to evolve a solution on a dynamical time-scale that is long relative to the component time scales,
- or balance to produce steady-state behavior.

Mathematical Approach - develop:

- Stable formulations involving implicitness in time (e.g. fully-implicit or IMEX time integration) to allow resolution of dynamical time-scale of interest
- Stable or stabilized spatial discretizations, options for physically conforming discretizations (e.g. edge elements, face elements)
- Robust and efficient fully-coupled nonlinear/linear iterative solution methods based on Newton-Krylov (NK) methods
- Scalable and efficient preconditioners utilizing multi-level (AMG) methods (Fully-coupled AMG, physics-based, approx. block factorization)

Three Variants of Preconditioning



1. Domain Decomposition (Trilinos/Aztec & Ifpack)
   - 1 –level Additive Schwarz DD
   - ILU(k) Factorization on each processor (with overlap)
   - High parallel eff., non-optimal algorithmic scalability

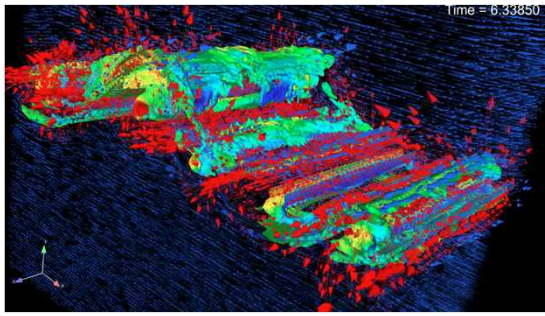
2. Multilevel Methods for Systems: ML pkg (Tuminaro, Sala, Hu, Siefert, Gee)

   Fully-coupled Algebraic Multilevel methods
   - Consistent set of DOF-ordered blocks at each node (e.g. Stabilized FE)
   - Uses block non-zero structure of Jacobian
   - Jacobi, GS, ILU(k) as smoothers
   - Can provide optimal algorithmic scalability

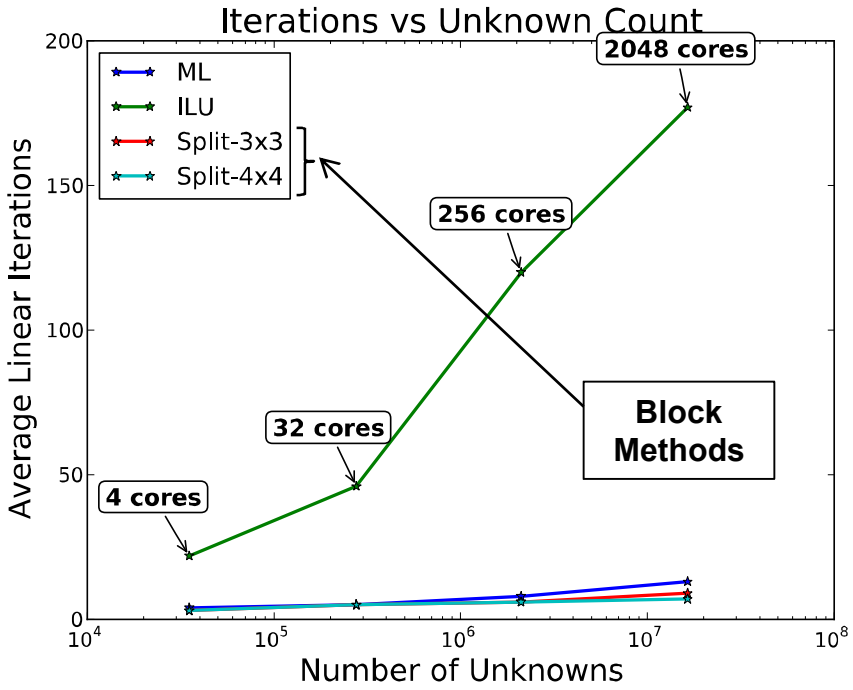
3. Approximate Block Factorization / Physics-based (Teko package)
   - Applies to mixed interpolation (FE), staggered (FV), physics compatible discretization approaches using segregated unknown blocking
   - Applies to systems where coupled AMG is difficult or might fail
   - Enables specialized AMG, e.g. H(grad), H(curl) to be applied to distinct discretizations.
   - Can provide optimal algorithmic scalability for coupled systems

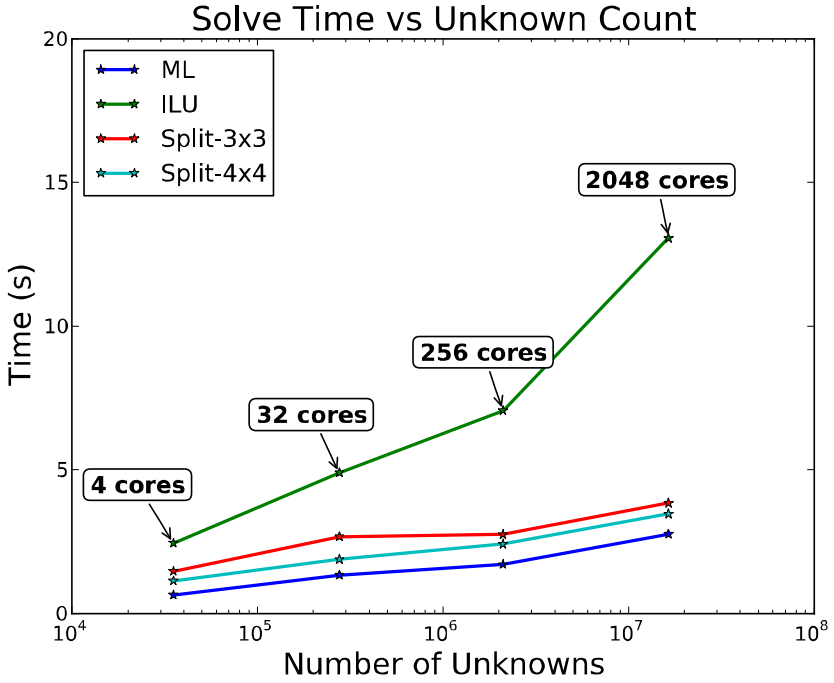
Comparison of Preconditioning Methods for MHD

3D HMKH [Re =10^4, Rem=10^4, M_A = 3; CFL ~0.125], SFE



FC-AMG – ILU(0), V(3,3); 3x3, 4x4 SIMPLEC and Gauss-Seidel





Fully coupled AMG limited to co-located nodal discretizations

Block Preconditioners

Split-3x3: 3x3 (SIMPLEC everywhere)

Preliminary Split-4x4: 4x4

Block Preconditioning and Time-Scales

A finite difference discretization of a 1D coupled convection-diffusion problem

$$\begin{pmatrix} \frac{1}{\Delta t}I + dD + aC & cC \\ cC & \frac{1}{\Delta t}I + dD + aC \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} R_u \\ R_v \end{pmatrix}$$

$$CFL_d = \frac{d\Delta t}{h^2}, CFL_a = \frac{a\Delta t}{h}, CFL_c = \frac{c\Delta t}{h}$$

$$\mathcal{P}_{GS} = \begin{pmatrix} \frac{1}{\Delta t}I + dD + aC & cC \\ 0 & \frac{1}{\Delta t}I + dD + aC \end{pmatrix} \quad \mathcal{P}_{SC} = \begin{pmatrix} \frac{1}{\Delta t}I + dD + aC & cC \\ 0 & S \end{pmatrix}$$

$$S = \frac{1}{\Delta t}I + dD + aC - c^2C(\frac{1}{\Delta t}I + dD + aC)^{-1}C$$

CFL_c	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}
GS	2	3			
J	3	5			
SC	2	2	2	2	2

$CFL_a = 1, CFL_d = 1$

CFL_c	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}
GS	2	3	5		
J	3	4	9		
SC	2	2	2	2	2

Schur complement is important when coupling time-scale is fast

Application to Continuum Plasma Simulations

- Electron plasma with a background magnetic field and density gradient
- Driven by an external current pulse
- Time-scales:

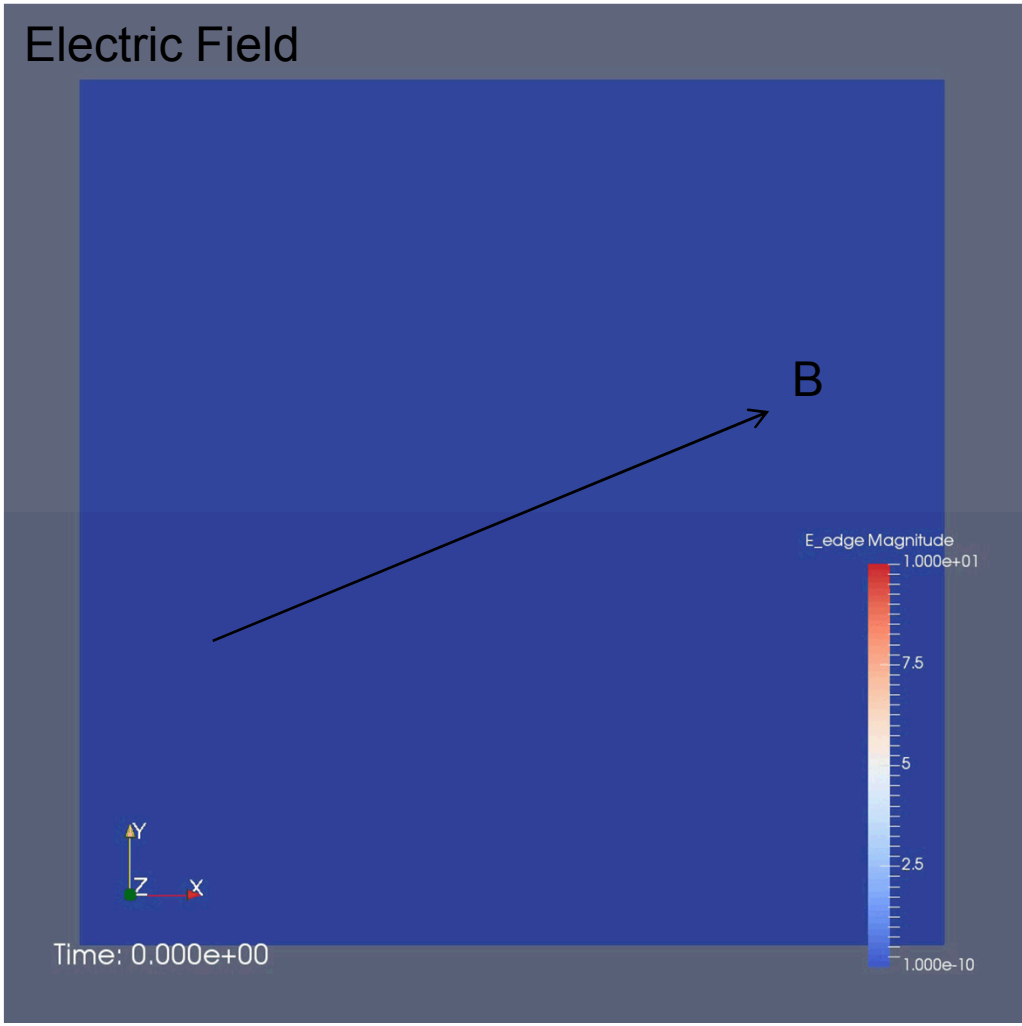
$$\tau_{EM} = \Delta x / c = 8 \times 10^{-8}$$

$$\tau_{\omega_{pe}} = \frac{1}{\sqrt{\frac{n_e q_e^2}{\epsilon_0 m_e}}} = 5 \times 10^{-7}$$

$$\tau_{\omega_{ce}} = \frac{1}{\frac{q_e B}{m_e}} = 1 \times 10^{-6}$$

$$\tau_{source} = 100 \times 10^{-9}$$

- Integrated implicitly to resolve current source
- Light waves are stiff and get stiffer with increased spatial resolution



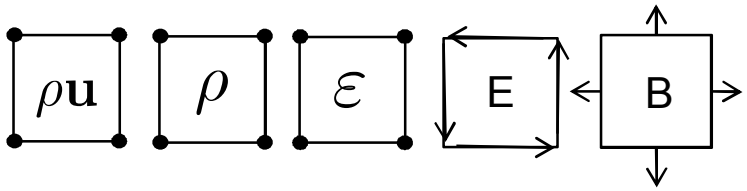
Simulation run with Richard Kramer, John Niederhaus, and Gregg Radtke

Multifluid 5-Moment Plasma Model

Sandia National Laboratories

Density	$\frac{\partial \rho_a}{\partial t} + \nabla \cdot (\rho_a \mathbf{u}_a) = \sum_{b \neq a} (n_a \rho_b \bar{\nu}_{ab}^{+} - n_b \rho_a \bar{\nu}_{ab}^{-})$
Momentum	$\frac{\partial (\rho_a \mathbf{u}_a)}{\partial t} + \nabla \cdot (\rho_a \mathbf{u}_a \otimes \mathbf{u}_a + p_a I + \Pi_a) = q_a n_a (\mathbf{E} + \mathbf{u}_a \times \mathbf{B})$ $- \sum_{b \neq a} \left[\rho_a (\mathbf{u}_a - \mathbf{u}_b) n_b \bar{\nu}_{ab}^{M} + \rho_b \mathbf{u}_b n_a \bar{\nu}_{ab}^{+} - \rho_a \mathbf{u}_a n_b \bar{\nu}_{ab}^{-} \right]$
Energy	$\frac{\partial \varepsilon_a}{\partial t} + \nabla \cdot ((\varepsilon_a + p_a) \mathbf{u}_a + \Pi_a \cdot \mathbf{u}_a + \mathbf{h}_a) = q_a n_a \mathbf{u}_a \cdot \mathbf{E} + Q_a^{src}$ $- \sum_{b \neq a} \left[(T_a - T_b) k \bar{\nu}_{ab}^{E} - \rho_a \mathbf{u}_a \cdot (\mathbf{u}_a - \mathbf{u}_b) n_b \bar{\nu}_{ab}^{M} - n_a \bar{\nu}_{ab}^{+} \varepsilon_b + n_b \bar{\nu}_{ab}^{-} \varepsilon_a \right]$
Charge and Current Density	$q = \sum_k q_k n_k \qquad \mathbf{J} = \sum_k q_k n_k \mathbf{u}_k$
Maxwell's Equations	$\frac{1}{c^2} \frac{\partial \mathbf{E}}{\partial t} - \nabla \times \mathbf{B} + \mu_0 \mathbf{J} = \mathbf{0} \qquad \nabla \cdot \mathbf{E} = \frac{q}{\epsilon_0}$ $\frac{\partial \mathbf{B}}{\partial t} + \nabla \times \mathbf{E} = \mathbf{0} \qquad \nabla \cdot \mathbf{B} = 0$

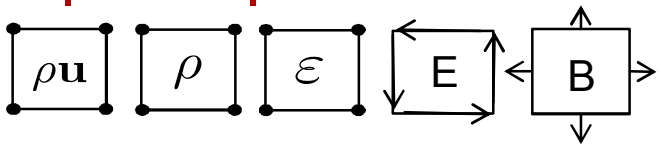
Nodal discretization for fluids,
Compatible discretization for electromagnetics



Fully Discrete Two-Fluid System

$$
\left(\begin{array}{cccccc|cc}
\mathbf{D}_{\rho_i} & \mathbf{K}^{\rho_i}_{\rho_i u_i} & 0 & \mathbf{Q}^{\rho_i}_{\rho_e} & 0 & 0 & 0 & 0 \\
\mathbf{D}^{\rho_i u_i}_{\rho_i} & \mathbf{D}_{\rho_i u_i} & 0 & \mathbf{Q}^{\rho_i u_i}_{\rho_e} & \mathbf{Q}^{\rho_i u_i}_{\rho_e u_e} & 0 & \mathbf{Q}^{\rho_i u_i}_{E} & \mathbf{Q}^{\rho_i u_i}_{B} \\
\mathbf{D}^{\mathcal{E}_i}_{\rho_i} & \mathbf{D}^{\mathcal{E}_i}_{\rho_i u_i} & \mathbf{D}_{\mathcal{E}_i} & \mathbf{Q}^{\mathcal{E}_i}_{\rho_e} & \mathbf{Q}^{\mathcal{E}_i}_{\rho_e u_e} & \mathbf{Q}^{\mathcal{E}_i}_{\mathcal{E}_e} & \mathbf{Q}^{\mathcal{E}_i}_{E} & 0 \\
\mathbf{Q}^{\rho_e}_{\rho_i} & 0 & 0 & \mathbf{D}_{\rho_e} & \mathbf{K}^{\rho_e}_{\rho_e u_e} & 0 & 0 & 0 \\
\mathbf{Q}^{\rho_e u_e}_{\rho_i} & \mathbf{Q}^{\rho_e u_e}_{\rho_i u_i} & 0 & \mathbf{D}^{\rho_e u_e}_{\rho_e} & \mathbf{D}_{\rho_e u_e} & 0 & \mathbf{Q}^{\rho_e u_e}_{E} & \mathbf{Q}^{\rho_e u_e}_{B} \\
\mathbf{Q}^{\mathcal{E}_e}_{\rho_i} & \mathbf{Q}^{\mathcal{E}_e}_{\rho_i u_i} & \mathbf{Q}^{\mathcal{E}_e}_{\mathcal{E}_i} & \mathbf{D}^{\mathcal{E}_e}_{\rho_e} & \mathbf{D}^{\mathcal{E}_e}_{\rho_e u_e} & \mathbf{D}_{\mathcal{E}_e} & \mathbf{Q}^{\mathcal{E}_e}_{E} & 0 \\
\hline
0 & \mathbf{Q}^{E}_{\rho_i u_i} & 0 & 0 & \mathbf{Q}^{E}_{\rho_e u_e} & 0 & \mathbf{Q}_E & \mathbf{K}^{E}_{B} \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & \mathbf{K}^{B}_{E} & \mathbf{Q}_B
\end{array}\right)
\left(\begin{array}{c}
\rho_i \\
\rho_i \mathbf{u}_i \\
\mathcal{E}_i \\
\rho_e \\
\rho_e \mathbf{u}_e \\
\mathcal{E}_e \\
\hline
\mathbf{E} \\
\hline
\mathbf{B}
\end{array}\right)
$$

Focus on block methods that segregate by discretization and approximate coupling through Schur complements



- Fast waves arise from off-diagonal coupling (electromagnetic waves, plasma waves for each species, circularly polarized waves) and can range over many orders of magnitude
- Disparate discretizations make it difficult to apply monolithic multigrid solvers

Reblocked System

$$\begin{pmatrix} Q_B & K_E^B & 0 \\ K_B^E & Q_E & Q_F^E \\ Q_B^F & Q_E^F & D_F \end{pmatrix} \begin{pmatrix} \mathbf{B} \\ \mathbf{E} \\ \mathbf{F} \end{pmatrix} = \begin{pmatrix} R_B \\ R_E \\ R_F \end{pmatrix}$$

- Block fluid DoFs together $\mathbf{F} = (\rho_i, \rho_i \mathbf{u}_i, \mathcal{E}_i, \rho_e, \rho_e \mathbf{u}_e, \mathcal{E}_e)$
- Use upper triangular factor of block LU decomposition as preconditioner

$$\mathcal{P} = \begin{pmatrix} Q_B & K_E^B & 0 \\ 0 & S_E & Q_F^E \\ 0 & 0 & S_F \end{pmatrix} \begin{aligned} S_E &= Q_E - K_B^E Q_B^{-1} K_E^B, \\ S_F &= D_F - (Q_E^F - Q_B^F Q_B^{-1} K_E^B) S_E^{-1} Q_F^E \end{aligned}$$

Fully coupled system captured in two Schur complement approximations and three subsolves (face, edge, node)

Schur Complement Approximations

Sandia National Laboratories

$$\mathcal{P} = \begin{pmatrix} Q_B & K_E^B & 0 \\ 0 & S_E & Q_F^E \\ 0 & 0 & S_F \end{pmatrix}$$

$$S_E = Q_E - K_B^E Q_B^{-1} K_E^B,$$

$$S_F = D_F - (Q_E^F - Q_B^F Q_B^{-1} K_E^B) S_E^{-1} Q_F^E$$

- Grad-div augmented approximation developed for Maxwell equations applied to S_E
  - Shown to be robust when integrating at time-scales several orders of magnitude slower than the speed of light
  - $S_E \approx T_E Z_E^{-1} Q_E$

$$T_E = Q_E - K_B^E Q_B^{-1} K_E^B + G Q_\rho^{-1} G^t$$

$$Z_E = Q_E + G Q_\rho^{-1} G^t$$

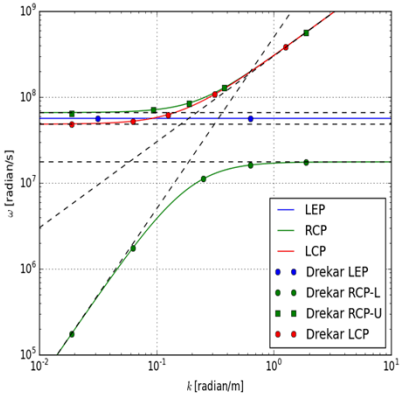
  - T_E and Q_E solves more amenable to traditional multigrid than S_E

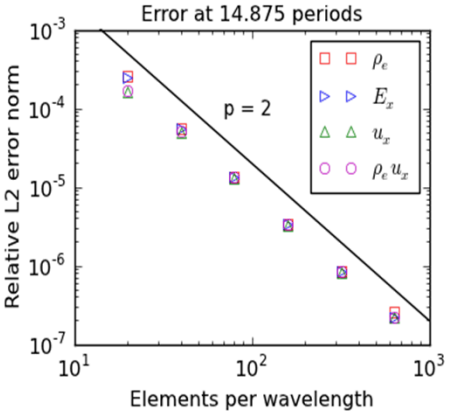
- SIMPLEC approximation for S_F
  - Replace embedded inverses with absolute row sum inverses
  - Regard as a perturbation of D_F
  - Co-located degrees of freedom allow for fully coupled AMG to be applied to this operator

Initial Weak Scaling for Electron / Ion Plasma Oscillation

Sandia National Laboratories

$$\tau_{EM} \approx 3.3 \times 10^{-13} \quad \text{on coarsest mesh}$$
$$\tau_{\omega_{pe}} \approx 1.1 \times 10^{-10}$$
$$\tau_{\omega_{pi}} \approx 4.8 \times 10^{-9}$$
$$\Delta t = 0.1 * \tau_{\omega_{pe}}$$





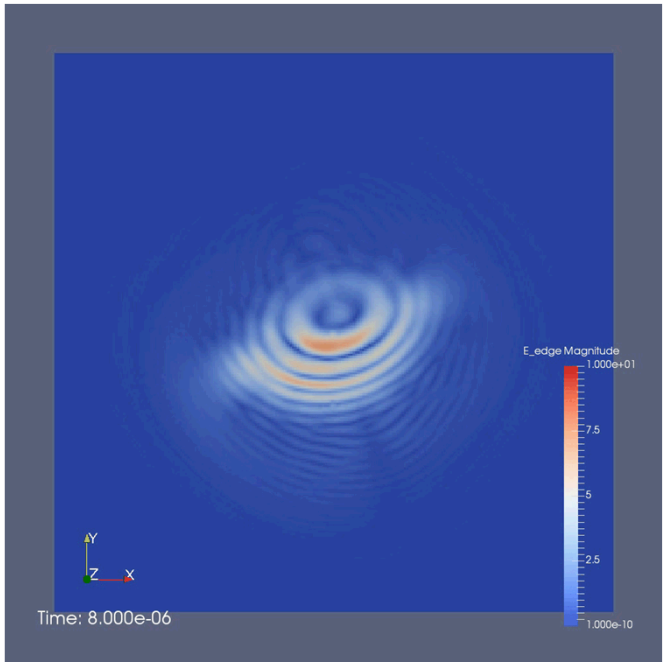
Dispersion and convergence results by John Niederhaus and Gregg Radtke

N	P	Linear its / Newton	Solve time / linear solve	$\frac{\Delta t_{imp}}{\Delta t_{exp}}$
100	1	4.18	0.2	300
200	2	4.21	0.22	600
400	4	4.27	0.23	1.2E+3
800	8	4.4	0.26	2.4E+3
1600	16	4.51	0.35	4.8E+3
3200	32	4.89	0.42	9.6E+3
6400	64	6.21	0.61	1.9e+4

Weak Scaling for 2D Problem

Sandia National Laboratories

Number of Processors	Average Linear Iterations	Average Time Per Newton Step
16	1.02	8.2075
64	1.04	8.5362
256	1.04	8.9625
1024	1.09	9.1713



120x120, 240x240, 480x480, 960x960 meshes
Approximately 22,000 DoFs per processor

Simulation made feasible through block preconditioning

Teko: A Package for Block Preconditioning

Sandia National Laboratories

- Part of Trilinos framework
  (released in Fall 2010 in Trilinos 10.6)
- Allows easy manipulation of blocks
- Tool for rapid prototyping and efficient implementation
- Use Trilinos solvers & preconditioners for subsolves
- Easy to pull in auxiliary operators and perform algebraic operations

Teko Example: Plasma Preconditioner

Sandia National Laboratories

- Create a factory for your preconditioner, inheriting from the abstract BlockPreconditionerFactory class
- Given a blocked operator, the factory builds the preconditioner as a linear operator

```
// Declaration of the preconditioner factory
class PlasmaPreconditionerFactory
   : public Teko::BlockPreconditionerFactory {
public:
   // Function inherited from BlockPreconditionerFactory
   Teko::LinearOp
     buildPreconditionerOperator(Teko::BlockedLinearOp & blo,
                                 Teko::BlockPreconditionerState & state) const;
```

- This factory builds the upper triangular block preconditioner

$$\mathcal{P} = \begin{pmatrix} Q_B & K_E^B & 0 \\ 0 & S_E & Q_F^E \\ 0 & 0 & S_F \end{pmatrix}$$

$$S_E = Q_E - K_B^E Q_B^{-1} K_E^B,$$
$$S_F = D_F - (Q_E^F - Q_B^F Q_B^{-1} K_E^B) S_E^{-1} Q_F^E$$

Extracting subblocks

Sandia National Laboratories

Blocking determined by physics application

```
<ParameterList name="Assembly">
    <Parameter name="Workset Size" type="int" value="100"/>
    <Parameter name="Use Tpetra" type="bool" value="true"/>
    <Parameter name="Field Order" type="string" value="blocked: B_face E_edge ELECTRON_RHO_UX - ELECTRON_RHO_UY - ELECTRON_RHO_UZ - ELECTRON_DENSITY"/>
</ParameterList>
```

```
Teko::LinearOp PlasmaPreconditionerFactory
::buildPreconditionerOperator(Teko::BlockedLinearOp & blo,
                              Teko::BlockPreconditionerState & state) const
{
   // Check that blocked operator is correct size
   int rows = Teko::blockRowCount(blo);
   int cols = Teko::blockColCount(blo);
   TEUCHOS_ASSERT(rows==3);
   TEUCHOS_ASSERT(cols==3);

   // Extract subblocks
   const Teko::LinearOp Q_B     = Teko::getBlock(0,0,blo);
   const Teko::LinearOp K_EB    = Teko::getBlock(0,1,blo);

   const Teko::LinearOp K_BE    = Teko::getBlock(1,0,blo);
   const Teko::LinearOp Q_E     = Teko::getBlock(1,1,blo);
   const Teko::LinearOp Q_FE    = Teko::getBlock(1,2,blo);

   const Teko::LinearOp Q_BF    = Teko::getBlock(2,0,blo);
   const Teko::LinearOp Q_EF    = Teko::getBlock(2,1,blo);
   const Teko::LinearOp D_F     = Teko::getBlock(2,2,blo);
```

$$\begin{pmatrix} Q_B & K_E^B & 0 \\ K_B^E & Q_E & Q_F^E \\ Q_B^F & Q_E^F & D_F \end{pmatrix}$$

Accessing Auxiliary Operators and Performing Matrix Algebra

- Auxiliary operators (mass matrix and gradient) assembled by physics application
- Accessed through request handler object

```
<ParameterList name="Auxiliary Physics Blocks">

    <ParameterList name="Plasma">

        <ParameterList>
            <Parameter name="Type" type="string" value="Auxiliary Mass Matrix"/>
            <Parameter name="DOF Name" type="string" value="AUXILIARY_NODE"/>
            <Parameter name="Basis Type" type="string" value="HGrad"/>
            <Parameter name="Basis Order" type="int" value="1"/>
            <Parameter name="Integration Order" type="int" value="2"/>
        </ParameterList>

        <ParameterList>
            <Parameter name="Type" type="string" value="Auxiliary Weak Gradient"/>
            <Parameter name="Scalar Name" type="string" value="AUXILIARY_NODE"/>
            <Parameter name="Vector Name" type="string" value="AUXILIARY_EDGE"/>
            <Parameter name="Basis Type" type="string" value="HCurl"/>
            <Parameter name="Basis Order" type="int" value="1"/>
            <Parameter name="Integration Order" type="int" value="2"/>
        </ParameterList>

    </ParameterList>

</ParameterList>
```

- Easy functions for transpose, extracting inverse diagonal operators, adding and multiplying operators

```
// Grab auxiliary operators
const Teko::LinearOp Q_rho   = getRequestHandler()->request<Teko::LinearOp>(Teko::RequestMesg("Mass Matrix"));
const Teko::LinearOp G       = getRequestHandler()->request<Teko::LinearOp>(Teko::RequestMesg("Weak Gradient"));
const Teko::LinearOp Gt      = Teko::explicitTranspose(G);

// Construct electric field Schur complement
const Teko::LinearOp idQ_B   = Teko::getInvDiagonalOp(Q_B,Teko::AbsRowSum);
const Teko::LinearOp S_E     = Teko::explicitAdd(Q_E,Thyra::scale(-1.0,Teko::explicitMultiply(K_BE,idQ_B,K_EB)));

// Grad-div augmentation
const Teko::LinearOp idQ_rho = Teko::getInvDiagonalOp(Q_rho,Teko::AbsRowSum);
const Teko::LinearOp GGt     = Teko::explicitMultiply(G,idQ_rho,Gt);
const Teko::LinearOp T_E     = Teko::explicitAdd(S_E,GGt);
const Teko::LinearOp Z_E     = Teko::explicitAdd(Q_E,GGt);

// Construct fluids Schur complement
```

$$S_E \approx T_E Z_E^{-1} Q_E \qquad \begin{aligned} T_E &= Q_E - K_B^E Q_B^{-1} K_E^B + G Q_\rho^{-1} G^t \\ Z_E &= Q_E + G Q_\rho^{-1} G^t \end{aligned}$$

Building Approximate Inverses

Sandia National Laboratories

```
// Build inverses
Teuchos::RCP<const Teko::InverseLibrary> invLib  = getInverseLibrary();
Teuchos::RCP<Teko::InverseFactory> invQB_Factory = invLib->getInverseFactory("MueLu Q_B");
Teuchos::RCP<Teko::InverseFactory> invTE_Factory = invLib->getInverseFactory("MueLu T_E");
Teuchos::RCP<Teko::InverseFactory> invSF_Factory = invLib->getInverseFactory("MueLu S_F");
const Teko::LinearOp invQB   = Teko::buildInverse("invQB",*invTE_Factory,Q_B);
const Teko::LinearOp invTE   = Teko::buildInverse("invTE",*invTE_Factory,T_E);
const Teko::LinearOp invSF   = Teko::buildInverse("invSF",*invTE_Factory,S_F);
const Teko::LinearOp invQE   = Teko::getInvDiagonalOp(Q_E,Teko::AbsRowSum);
```

- Inverses constructed from user supplied parameter lists
- Access to multigrid (ML, MueLu), incomplete factorizations (IFPACK), direct solves (Amesos), Krylov methods (Aztec, Belos) through Stratimikos interface to Trilinos packages

```
<ParameterList name="MueLu Q_B">
  <Parameter name="Type" type="string" value="MueLu-Tpetra"/>

  <Parameter name="verbosity"                          type="string"   value="high"/>
  <Parameter name="number of equations"                type="int"      value="1"/>
  <Parameter name="max levels"                         type="int"      value="10"/>
  <Parameter name="multigrid algorithm"                type="string"   value="unsmoothed"/>
  <Parameter name="sa: use filtered matrix"            type="bool"     value="true"/>
  <Parameter name="sa: damping factor"                 type="double"   value="0.0"/>
  <Parameter name="coarse: max size"                   type="int"      value="1000"/>
  <Parameter name="coarse: type"                       type="string"   value="KLU2"/>
  <Parameter name="transpose: use implicit"            type="bool"     value="false"/>
  <Parameter name="problem: symmetric"                 type="bool"     value="true"/>

  <!-- =================  AGGREGATION  ================= -->
  <Parameter name="aggregation: type"                  type="string"   value="uncoupled"/>
  <Parameter name="aggregation: drop scheme"           type="string"   value="classical"/>
  <Parameter name="aggregation: drop tol"              type="double"   value="0.0"/>

  <!-- =================  SMOOTHERS  ================= -->
  <Parameter name="smoother: pre or post"              type="string"   value="both"/>
  <Parameter name="smoother: type"                     type="string"   value="RELAXATION"/>
  <ParameterList name="smoother: params">
    <Parameter name="relaxation: type"                 type="string" value="Gauss-Seidel"/>
    <Parameter name="relaxation: sweeps"               type="int"    value="5"/>
    <Parameter name="relaxation: damping factor"       type="double" value="1"/>
  </ParameterList>
</ParameterList>
<!-- end "MueLu Q_B" -->
```

Constructing the Preconditioner

- Convenient functions for constructing block upper, lower, and diagonal inverse operators
- Just provide the block diagonal inverse operators (approximate inverses of Schur complement approximations) and the off-diagonal blocks

```
// Build inverse diagonal
std::vector<Teko::LinearOp> invDiag(3);
invDiag[0] = invQB;
invDiag[1] = Teko::multiply(invQE,Teko::multiply(invTE,Z_E));
invDiag[2] = invSF;

// Construct block upper triangular operator
Teko::BlockedLinearOp U = Teko::createBlockedOp();
Teko::beginBlockFill(U,3,3);
{
  Teko::setBlock(0,0,U,Q_B);
  Teko::setBlock(0,1,U,K_EB);
  Teko::setBlock(1,1,U,S_E);
  Teko::setBlock(1,2,U,Q_FE);
  Teko::setBlock(2,2,U,S_F);
}
Teko::endBlockFill(U);

// Build preconditioner inverse
return(Teko::createBlockUpperTriInverseOp(U,invDiag));
```

$$S_E \approx T_E Z_E^{-1} Q_E \qquad \begin{aligned} T_E &= Q_E - K_B^E Q_B^{-1} K_E^B + G Q_\rho^{-1} G^t \\ Z_E &= Q_E + G Q_\rho^{-1} G^t \end{aligned}$$

$$\mathcal{P} = \begin{pmatrix} Q_B & K_E^B & 0 \\ 0 & S_E & Q_F^E \\ 0 & 0 & S_F \end{pmatrix}$$

Summary

- Multiphysics result in difficult linear systems with many coupled fields and fast time-scales leading to stiff modes
- Preconditioning is particularly challenging when disparate discretizations are used and fast time-scales are associated with off-diagonal couplings
- Block preconditioners are attractive in this situation
- Effectiveness of block preconditioning was shown for mixed a discretization multifluid continuum plasma model
- The Teko package is a powerful tool that allows user friendly definition of block preconditioners as well as efficient implementation