Development of a Fast, Scalable Solver for the HiFi 3D Extended MHD Code

A. H. Glasser, PSI Center, University of Washington
V. S. Lukin, Naval Research Laboratory

Presented at the
CEMM and APS/DPP Meetings
Salt Lake City, Utah, November 13-18, 2011

Progress on 3D Scalable Solver

- 3D solver modules:
  - Static condensation
  - Physics-based preconditioning, Schur complement
- 3D test programs:
  - Nonlinear visco-resistive MHD in arbitrary geometry, boundary conditions
  - Ideal MHD waves in triply periodic cube
  - 3D GEM challenge problem
- PETSC interface:
  - Choice of solvers and preconditioners at each level through options database
  - Extensive runtime timing and profiling.
- Full ideal MHD Schur complement derived and implemented, including nonlinear and nonuniform terms.
- Scaling tests performed on hopper up to 32,768 cores, verifying scalability.

Scalable Parallel Solver for Extended MHD

- **Jacobian-Free Newton-Krylov (JFNK)**
  PETSc SNES solver with Physics-Based Preconditioning. Time-centered solution of full nonlinear system of equations.

- **Physics Based Preconditioning (PBP)**
  Chacón. Reduces full hyperbolic linear system to smaller parabolic systems.
  - Partition 1: Mass matrix **M**
    mass density, plasma pressure, magnetic fields, currents
  - Partition 2: Approximate Schur complement matrix **S**
    fluid momenta

- **Static Condensation (SC)**
  Exploits C^0 continuity of spectral element representation. Uses small, local direct solves to eliminates cell interior degrees of freedom in terms of cell boundaries.

- **Solution of Reduced, Condensed Linear Systems**
  - Solver: CG for SPD matrices, GMRES for non-SPD.
  - Preconditioners
    - Schwarz overlap preconditioned by core-wise SuperLU_DIST.
      Fast and efficient but not scalable, increasing number of Krylov iterations
    - Algebraic multigrid, Hypre/BoomerAMG.
      Scalable for limited range of test cases; smoother requires nodal basis.

Physics-Based Preconditioning

Factorization and Schur Complement

Linear System

$$\mathbf{L}\mathbf{u} = \mathbf{r}, \quad \mathbf{L} \equiv \begin{pmatrix} \mathbf{L}_{11} & \mathbf{L}_{12} \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{pmatrix}, \quad \mathbf{u} = \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix}, \quad \mathbf{r} = \begin{pmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{pmatrix}$$

Factorization

$$\mathbf{L} \equiv \begin{pmatrix} \mathbf{L}_{11} & \mathbf{L}_{12} \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{L}_{21}\mathbf{L}_{11}^{-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{L}_{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{S} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{L}_{11}^{-1}\mathbf{L}_{12} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}$$

Schur Complement

$$\mathbf{S} \equiv \mathbf{L}_{22} - \mathbf{L}_{21}\mathbf{L}_{11}^{-1}\mathbf{L}_{12}$$

Exact and Approximate Inverse

Preconditioned Krylov Iteration

Inverse

$$\mathbf{L}^{-1} = \begin{pmatrix} \mathbf{I} & -\mathbf{L}_{11}^{-1}\mathbf{L}_{12} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{L}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{L}_{21}\mathbf{L}_{11}^{-1} & \mathbf{I} \end{pmatrix}$$

Exact Solution

$$\mathbf{s}_1 = \mathbf{L}_{11}^{-1}\mathbf{r}_1, \quad \mathbf{s}_2 = \mathbf{r}_2 - \mathbf{L}_{21}\mathbf{s}_1$$

$$\mathbf{u}_2 = \mathbf{S}^{-1}\mathbf{s}_2, \quad \mathbf{u}_1 = \mathbf{s}_1 - \mathbf{L}_{11}^{-1}\mathbf{L}_{12}\mathbf{u}_2$$

Preconditioned Krylov Iteration

$$\mathbf{P} \approx \mathbf{L}^{-1}, \quad (\mathbf{L}\mathbf{P})\left(\mathbf{P}^{-1}\mathbf{u}\right) = \mathbf{r}$$

Outer iteration preserves full nonlinear accuracy.
Need approximate Schur complement $\mathbf{S}$
and scalable solution procedure for $\mathbf{L}_{11}$ and $\mathbf{S}$.

MHD Schur Complement, General Form

Schur Complement Equation

$$\frac{\partial^2}{\partial t^2}(\rho \mathbf{v}) + \nabla \cdot \dot{\mathsf{T}} = 0$$

$$\dot{\mathsf{T}} = \dot{\mathsf{T}}^\dagger = [\mathbf{B} \cdot \nabla \times (\mathbf{v} \times \mathbf{B}) - \gamma p \nabla \cdot \mathbf{v} - \mathbf{v} \cdot \nabla p]\, \mathsf{I} - \mathbf{B} \nabla \times (\mathbf{v} \times \mathbf{B}) - \nabla \times (\mathbf{v} \times \mathbf{B})\, \mathbf{B}$$

$$\frac{\partial^2}{\partial t^2}(\mathcal{J} \rho \mathbf{v} \cdot \nabla x_i) + \frac{\partial}{\partial x_j}\left(\mathcal{J} \dot{\mathsf{T}} : \nabla x_i \nabla x_j\right) = \mathcal{J} \dot{\mathsf{T}} : \nabla \nabla x_i$$

Scalar Components of Schur Complement

$$\begin{aligned}
\dot{T}_{ij} =& \dot{T}_{ji} \equiv \mathcal{J} \dot{\mathsf{T}} : \nabla x_i \nabla x_j \\
=& \left\{ \frac{1}{2} \epsilon_{klm} (\mathcal{J} \mathbf{B} \cdot \nabla x_k \times \nabla x_l) \frac{\partial}{\partial x_n} [(\mathcal{J} \mathbf{v} \cdot \nabla x_m)(\mathbf{B} \cdot \nabla x_n) - (\mathcal{J} \mathbf{v} \cdot \nabla x_n)(\mathbf{B} \cdot \nabla x_m)] \right. \\
& \left. -\gamma p \frac{\partial}{\partial x_k} (\mathcal{J} \mathbf{v} \cdot \nabla x_k) - (\mathcal{J} \mathbf{v} \cdot \nabla x_i) \frac{\partial p}{\partial x_i} \right\} (\nabla x_i \cdot \nabla x_j) \\
& - (\mathbf{B} \cdot \nabla x_i) \frac{\partial}{\partial x_k} [(\mathcal{J} \mathbf{v} \cdot \nabla x_j)(\mathbf{B} \cdot \nabla x_k) - (\mathcal{J} \mathbf{v} \cdot \nabla x_k)(\mathbf{B} \cdot \nabla x_j)] \\
& - (\mathbf{B} \cdot \nabla x_j) \frac{\partial}{\partial x_k} [(\mathcal{J} \mathbf{v} \cdot \nabla x_i)(\mathbf{B} \cdot \nabla x_k) - (\mathcal{J} \mathbf{v} \cdot \nabla x_k)(\mathbf{B} \cdot \nabla x_i)]
\end{aligned}$$

Uniform Field and Gradient Terms

$$\dot{T}_{ij} = S_{ijkl} \partial_k (\mathcal{J} \mathbf{v} \cdot \nabla x_l) + R_{ijk} (\mathcal{J} \mathbf{v} \cdot \nabla x_k), \quad S_{ijkl} \equiv \frac{\partial \dot{T}_{ij}}{\partial [\partial_k (\mathcal{J} \mathbf{v} \cdot \nabla x_l)]}, \quad R_{ijk} \equiv \frac{\partial \dot{T}_{ij}}{\partial (\mathcal{J} \mathbf{v} \cdot \nabla x_k)}$$

Mathematica Program: Formulation

```
xvec = Table[x[i],{i,3}];
vvec = Table[v[i][xvec],{i,3}];
bvec = Table[b[i][xvec],{i,3}];
bbvec = Table[bb[i][xvec],{i,3}];

gradv = Table[D[vvec[[l]], xvec[[k]]], {k, 3}, {l, 3}];
divv = Sum[gradv[[i,i]], {i, 3}];

d1 = Sum[bbvec[[m]]*
    D[vvec[[m]]bvec[[n]]-vvec[[n]]bvec[[m]],xvec[[n]]],
    {m, 3},{n, 3}];
d2 = -cs2[xvec] divv -
    Sum[vvec[[i]] D[p[xvec], xvec[[i]]], {i, 3}];
d = Simplify[d1 + d2];

t1 = d*IdentityMatrix[3];
t2 = Simplify[Table[bvec[[i]]*Sum[
    D[vvec[[j]] bvec[[k]] - vvec[[k]] bvec[[j]], xvec[[k]]],
    {k, 3}], {i, 3}, {j, 3}]];

t = Simplify[t1 - t2 - Transpose[t2]];

s = Simplify[Table[D[t[[i, j]], gradv[[k, l]]],
    {i, 3}, {j, 3}, {k, 3}, {l, 3}]];

r = Simplify[Table[D[t[[i, j]], vvec[[k]]],
    {i, 3}, {j, 3}, {k, 3}]];
```

Mathematica Program: Simplification

```
rule = {cs2[xvec] → cs2, p[xvec] → p,
   p^({1,0,0})[xvec] → px, p^({0,1,0})[xvec] → py, p^({0,0,1})[xvec] → pz,
   b[1][xvec] → bx, b[2][xvec] → by, b[3][xvec] → bz,
   bb[1][xvec] → bx, bb[2][xvec] → by, bb[3][xvec] → bz,
   b[1]^({1,0,0})[xvec] → bxx, b[2]^({1,0,0})[xvec] → byx, b[3]^({1,0,0})[xvec] → bzx,
   b[1]^({0,1,0})[xvec] → bxy, b[2]^({0,1,0})[xvec] → byy, b[3]^({0,1,0})[xvec] → bzy,
   b[1]^({0,0,1})[xvec] → bxz, b[2]^({0,0,1})[xvec] → byz, b[3]^({0,0,1})[xvec] → bzz,
   bb[1]^({1,0,0})[xvec] → bxx, bb[2]^({1,0,0})[xvec] → byx, bb[3]^({1,0,0})[xvec] → bzx,
   bb[1]^({0,1,0})[xvec] → bxy, bb[2]^({0,1,0})[xvec] → byy, bb[3]^({0,1,0})[xvec] → bzy,
   bb[1]^({0,0,1})[xvec] → bxz, bb[2]^({0,0,1})[xvec] → byz, bb[3]^({0,0,1})[xvec] → bzz};
s = Simplify[s /. rule];
r = Simplify[r /. rule];
```

```
sxx = s[[1, vec, 1, vec]];
syx = s[[1, vec, 2, vec]];
szx = s[[1, vec, 3, vec]];

sxy = s[[2, vec, 1, vec]];
syy = s[[2, vec, 2, vec]];
szy = s[[2, vec, 3, vec]];

sxz = s[[3, vec, 1, vec]];
syz = s[[3, vec, 2, vec]];
szz = s[[3, vec, 3, vec]];

rx = r[[vec, 1, vec]];
ry = r[[vec, 2, vec]];
rz = r[[vec, 3, vec]];
```

Mathematica Program: Output

```
dir = "~/Desktop";
SetDirectory[dir];

file = "smat.f";

Write[file, sxx];
Write[file, syx];
Write[file, szx];

Write[file, sxy];
Write[file, syy];
Write[file, szy];

Write[file, sxz];
Write[file, syz];
Write[file, szz];

Close[file];

file = "rmat.f";

Write[file, rx];
Write[file, ry];
Write[file, rz];

Close[file];
```

Schur Complement, Fortran Code

```
 sxx=RESHAPE((/-by**2-bz**2-cs2,bx*by,bx*bz,bx*by,-bx**2,vzero,
$      bx*bz,vzero,-bx**2/),(/n,3,3/))
 syx=RESHAPE((/-bx*by,bx**2-bz**2-cs2,by*bz,-by**2,bx*by,vzero,
$      -by*bz,2*bx*bz,-bx*by/),(/n,3,3/))
 szx=RESHAPE((/-bx*bz,by*bz,bx**2-by**2-cs2,-by*bz,-bx*bz,
$      2*bx*by,-bz**2,vzero,bx*bz/),(/n,3,3/))
 sxy=RESHAPE((/bx*by,-bx**2,vzero,by**2-bz**2-cs2,-bx*by,bx*bz,
$      2*by*bz,-bx*bz,-bx*by/),(/n,3,3/))
 syy=RESHAPE((/-by**2,bx*by,vzero,bx*by,-bx**2-bz**2-cs2,by*bz,
$      vzero,by*bz,-by**2/),(/n,3,3/))
 szy=RESHAPE((/-by*bz,-bx*bz,2*bx*by,bx*bz,-by*bz,
$      -bx**2+by**2-cs2,vzero,-bz**2,by*bz/),(/n,3,3/))
 sxz=RESHAPE((/bx*bz,vzero,-bx**2,2*by*bz,-bx*bz,-bx*by,
$      -by**2+bz**2-cs2,bx*by,-bx*bz/),(/n,3,3/))
 syz=RESHAPE((/-by*bz,2*bx*bz,-bx*by,vzero,by*bz,-by**2,
$      bx*by,-bx**2+bz**2-cs2,-by*bz/),(/n,3,3/))
 szz=RESHAPE((/-bz**2,vzero,bx*bz,vzero,-bz**2,by*bz,bx*bz,by*bz,
$      -bx**2-by**2-cs2/),(/n,3,3/))
```

- Similar expressions for rx, ry, rz, involving gradients of **B**.
- Transformation from **v** to ρ**v**.

Using PETSc Runtime Options to Choose Solvers and Preconditioners

Fortran Source Code

```
c-----------------------------------------------------------------------
c     create linear solver, assign prefix, read options.
c-----------------------------------------------------------------------
      CALL KSPCreate(comm,ctv%ksp,ierr)
      CALL KSPSetOptionsPrefix(ctv%ksp,TRIM(prefix),ierr)
      CALL KSPSetFromOptions(ctv%ksp,ierr)
```

PETSc Runtime File `petopt`, Options for Schur Solver

```
-schur_ksp_type cg
-schur_ksp_rtol 1e-8
-schur_ksp_max_it 500
-schur_pc_type hypre
-schur_pc_hypre_type boomeramg
-schur_pc_hypre_boomeramg_max_iter 1
-schur_pc_hypre_boomeramg_tol 0
-schur_pc_hypre_boomeramg_coarsen_type HMIS
-schur_pc_hypre_boomeramg_interp_type ext+I
-schur_pc_hypre_boomeramg_strong_threshold .5
-schur_pc_hypre_boomeramg_relax_type_down SOR/Jacobi
-schur_pc_hypre_boomeramg_relax_type_up backward-SOR/Jacobi
-schur_pc_hypre_boomeramg_grid_sweeps_all 2
-schur_pc_hypre_boomeramg_truncfactor .5
-schur_pc_hypre_boomeramg_P_max 4
```

Ideal MHD Waves

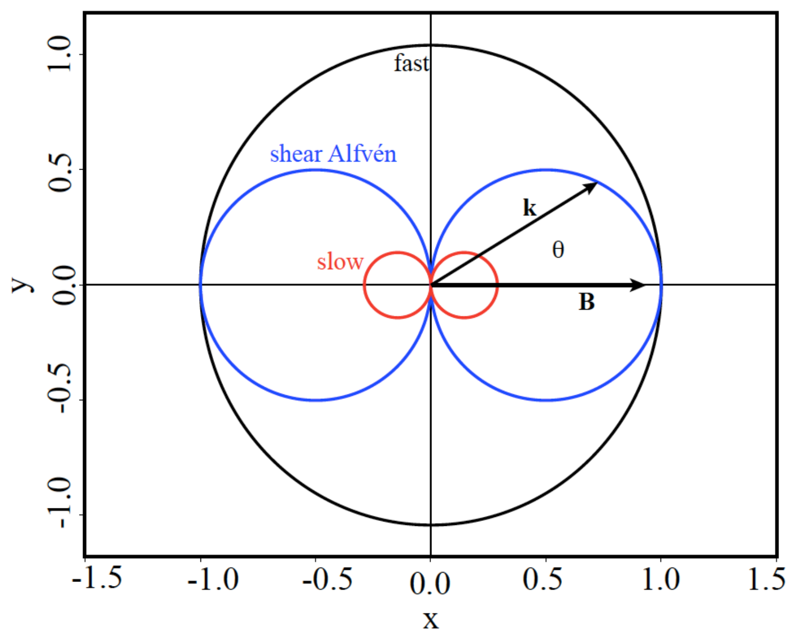
Shear Alfvén Waves, Fast and Slow Magnetosonic Waves

$$\frac{\omega^2}{k^2} = c_A^2 \cos^2\theta, \quad \frac{1}{2}\left\{\left(c_A^2 + c_S^2\right) \pm \left[\left(c_A^2 + c_S^2\right)^2 - 4c_A^2 c_S^2 \cos^2\theta\right]^{1/2}\right\}$$

Condition Number

$$\kappa(\mathbf{S}) \sim \left(\frac{k_{\text{max}}}{k_{\text{min}}}\right)^2 \left(\frac{\omega_{\text{fast}}}{\omega_{\text{slow}}}\right)^2$$

Friedrichs Diagram, $\beta = 10\%$



Weak Scaling: Test Problem

- Linear ideal MHD traveling waves in a triply periodic uniform cube.
- 3D **k**-vector; 3D uniform **B**-vector specified by spherical angles about **k** vector. Continuous control of angle $\theta = 45^\circ, 75^\circ$ between **k** and **B.**
- Initialize to pure slow wave, amplitude delta = 10^{-3}.
- Unit cell: 1 full wavelength in each direction, nx = ny = nz = 4, np =4 and 6.
- Each core has one unit cell, doubled in each direction on each increment. Hopper.nersc.gov, 16 cores/node, 2GB/core, ncore = 64, 512, 4,096, 32,768.
- Dependent variables: density, 3 momenta, 3 vector potentials, 3 currents, pressure, nqty = 11.
- Largest test problem size 78M variables, one Jacobian evaluation,
  64 time steps to one full slow wave period.

Weak Scaling: Results

➢ **Solution Procedure 1: GMRES on fully coupled problem, nqty = 11.**

- Preconditioner: Additive Schwarz, overlap 1, SuperLU on each core
- Results: scales well with np = 4. Memory failure with np = 6.

➢ **Solution Procedures 2 and 3: Physics Based Preconditioning**

❖ Mass matrix, $nqty_1 = 8$: GMRES with Additive Schwarz and SuperLU on each core.

❖ Schur complement, $nqty_2 = 3$: CG with

✧ Procedure 2: Additive Schwarz and SuperLU on each core

✧ Procedure 3: Hypre/BoomerAMG algebraic multigrid

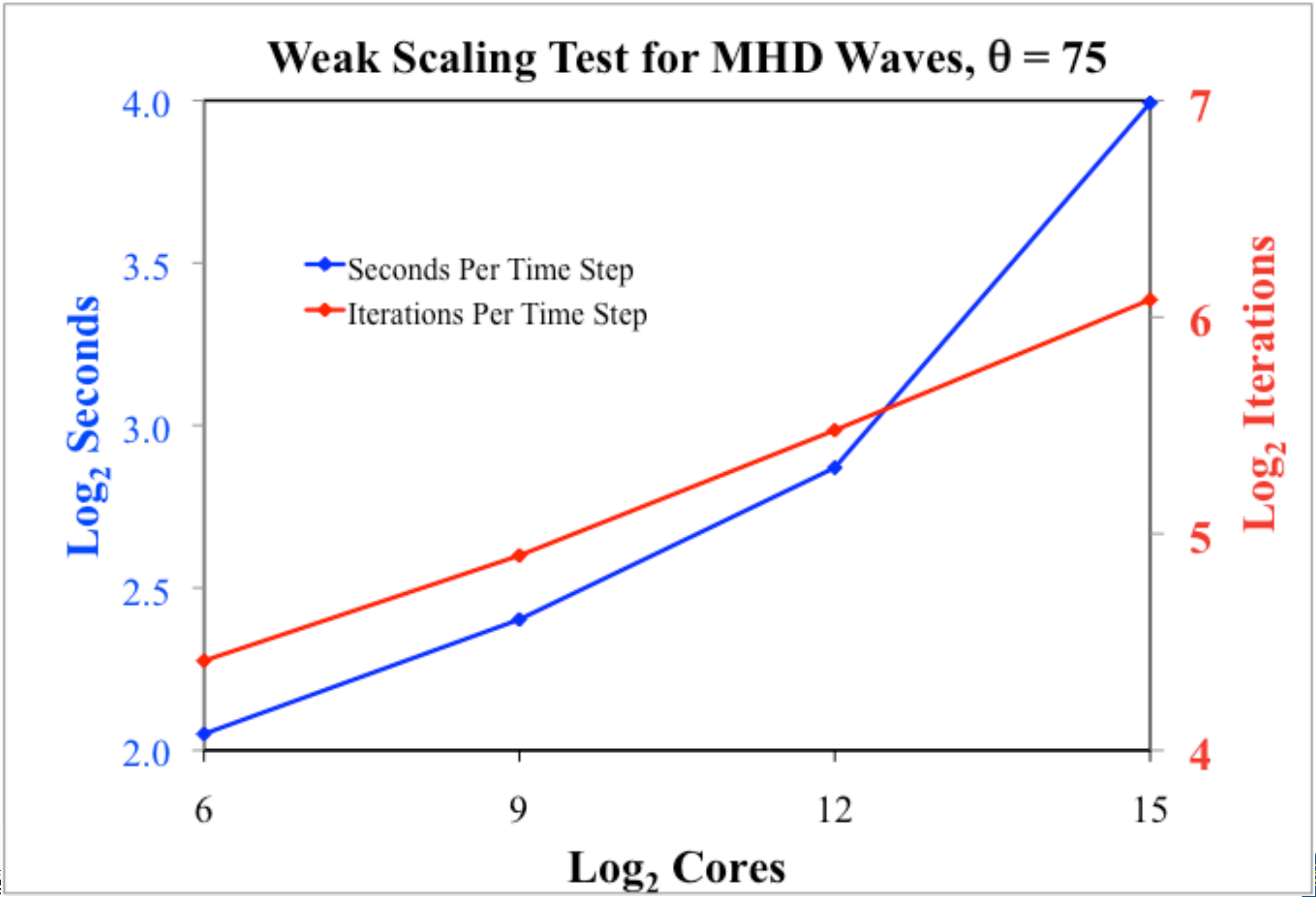
❖ Results:

✧ Both 2 and 3 scale well for np = 4.

✧ Procedure 2 scales poorly on condition number and ksp iterations for np = 6.

✧ Procedure 3 scales well.

Scaling for Algebraic Multigrid Schur Solve, np = 6



The Jed Brown Fix

- Scalable parallel solver development is based on the assumption that the most computationally intensive operation is linear system solution.
- This has been reasonably achieved, with time to solution increasing by a factor of 4 while the number of dependent variables and cores increases by a factor of a 512. There isn't room on hopper to increase the cores by another factor of 8.
- For np = 6, with one Jacobian and 64 time steps, matrix formation takes more than half the run time and most of the storage. The time gets rapidly worse for nonlinear problems with multiple Jacobian evaluations, and for np > 6.
- Jed Brown, "Efficient Nonlinear Solvers for Nodal High-Order Finite Elements in 3D," J. Sci. Comput. **45**, 48-63 (2010).
- Cause: full matrix blocks coupling all polynomial basis functions.
- Cure: form approximate Jacobian using linear finite element discretization on the grid of GLL nodal points; use as preconditioner for matrix-free solution methods.
- Greatly reduces number of nonzero matrix elements; accelerates matrix formation and matrix-vector multiplication; reduces storage. Replaces static condensation.
- Retains benefits of high-order methods while minimizing cost.

Remaining Work for Scalable Solver

- Debug and verify nonlinear, nonuniform **R** terms in ideal MHD Schur complement; drift waves, GEM challenge.
- Hall terms: asymmetry; higher condition number.
- Polar boundary conditions
- Multiblock version of HiFi
- More applied physics problems.
- The Jed Brown fix: computation, storage, and matrix-vector multiplication of Jacobian matrix for higher np.