# Topic I: Ideal-like modeling with NIMROD

# Topic II: Interfacing NIMROD with MUMPS

J R King, S E Kruger

Tech-X Corporation

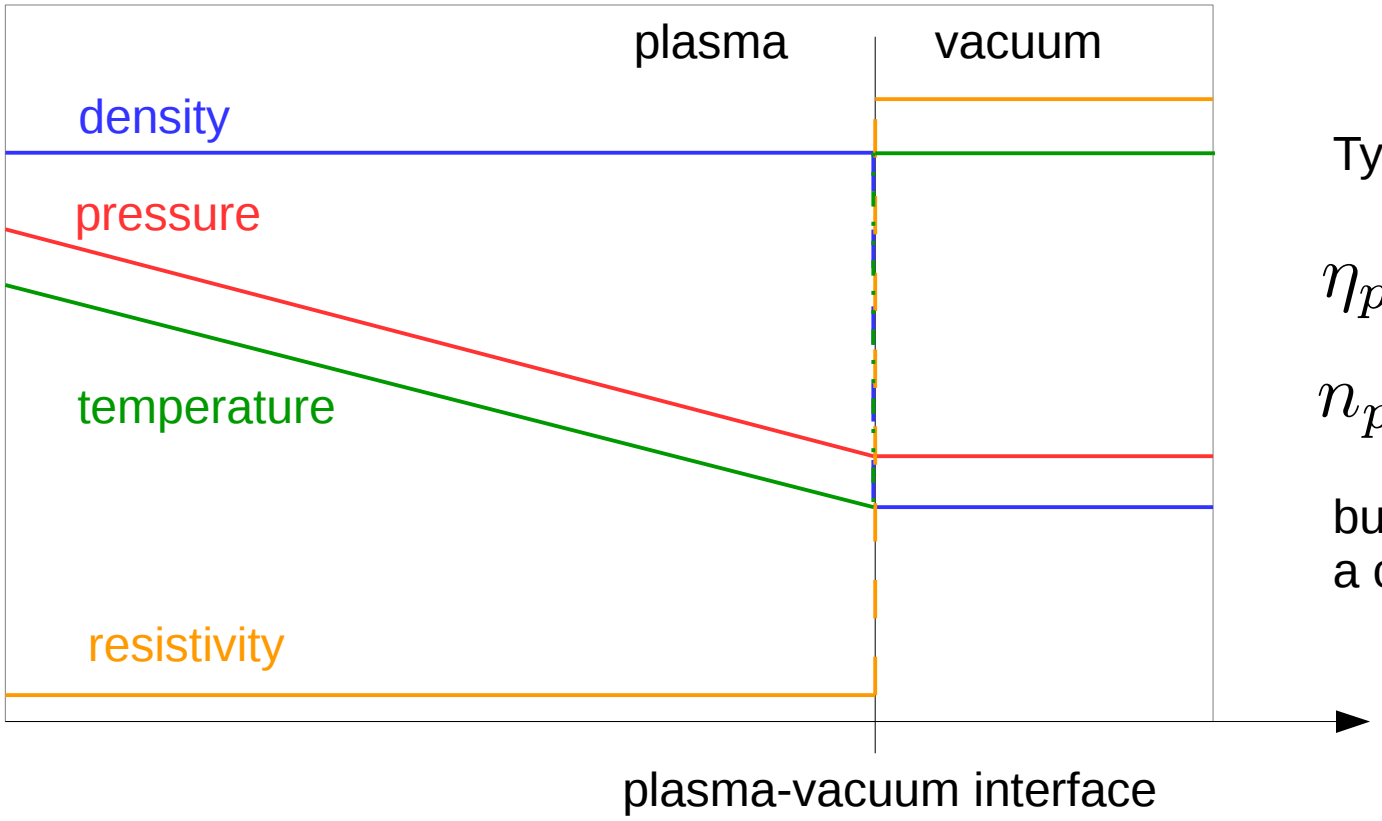
NIMROD Team Meeting
Sherwood 2014

# Why is ideal-like modeling important?

- Codes based on the energy principle typically compute an unstable mode based on the ideal-MHD response for a plasma surrounded by a vacuum region.
- These codes are widely used by both theorists and experimentalists.
- The vacuum is defined as zero density and zero current region.
- Ideal-like modeling is important for code benchmarks and to explicitly show differences in modeling with a low-temperature plasma is used outside the LCFS instead of a vacuum.
- From a NIMROD-centric point of view, these cases are not interesting.

# How does on implement a vacuum region in NIMROD?

- Zero current → large diffusivity
- Zero density → low density
- These constraints lead to challenging computations with discontinuous equilibrium quantities inside the computational domain.




Typical ratios are

$$\eta_p/\eta_v \simeq 10^{-7}$$

$$n_p/n_v \simeq 100$$

but these can vary on a case-to-case basis.

# Ideal-like cases introduce numerical challenges.

- Quadrature-point evaluations of a discontinuous (or nearly discontinuous) functions are prone to Gibbs-type phenomena with higher-order elements.
- Gradients of discontinuous functions are not well defined at the plasma-vacuum interface (PVI).
- Past modeling of these cases used a approximate step function at the PVI (such as a tanh), and converged to the limit of step function.
  - This approach was effective with simple equilibrium shaping.
  - This can take considerable resolution with non-trivial shaping, and may not be effective!

# These numerical challenges are evident in the density advance.

- Even without quadrature point overshoots, the FE weak form as implemented in NIMROD is not be amenable to discontinuous functions. Consider the RHS of the continuity equation where two regions are separated by the discontinuity in density and integrated by parts:

$$-\int dV_p \alpha \nabla \cdot (n_p \mathbf{v}) - \int dV_v \alpha \nabla \cdot (n_v \mathbf{v})$$
$$= \int dV_p \nabla \alpha \cdot \mathbf{v} n_p + \int dV_v \nabla \alpha \cdot \mathbf{v} n_v$$
$$- \int dS_{pv} \cdot \mathbf{v} n_p \alpha - \int dS_{vp} \cdot \mathbf{v} n_v \alpha - \int dS_w \cdot \mathbf{v} n_v \alpha$$

- The surface terms at the plasma-vacuum interface do not cancel and are missing in our formulation.
- Even when this term is not integrated by parts problems can still arise (but they are smaller problems).


# A new ideal-like advance has been implemented that avoids routines that evaluate gradients of n and T.

- This (linear only) advance uses v, B and p (instead of T) as fundamental variables. It is enabled when ideal_like=.true.
- Density can be advanced, but it does not affect v, B and p with an ideal-MHD model. In my experience the density advance results in unphysical solution for n.
- This avoid explicit evaluation of gradients of discontinuous quantities, but does not fix the quadrature overshoot problem.

# Analytic expressions can be used for the certain quadrature point evaluations.

- Quadrature-point evaluations in NIMROD are done 'on-the-fly' from the nodal basis functions.
- After these evaluations have completed, we post-adjust the values with analytic functions based on the vacuum region location on the mesh.
- This requires the PVI to be mesh aligned and known by NIMROD.
- Derivatives of density are set to zero throughout the domain and derivatives of p and J are set to zero in the vacuum.

$$n = n_{dens} + (n_{edge} - n_{dens})H(\psi_{PVI})$$

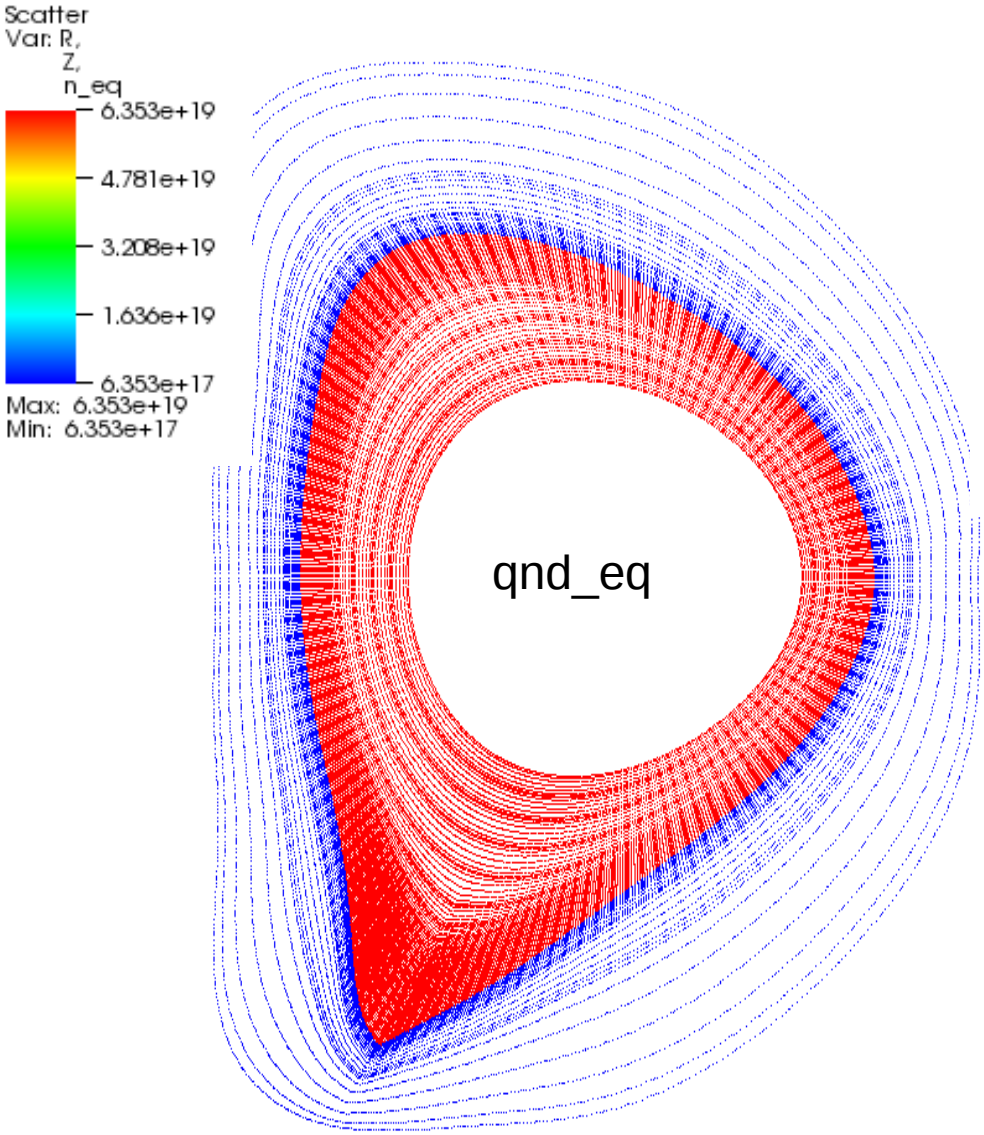
$$Diff = 1 + (dvac - 1)H(\psi_{PVI})$$

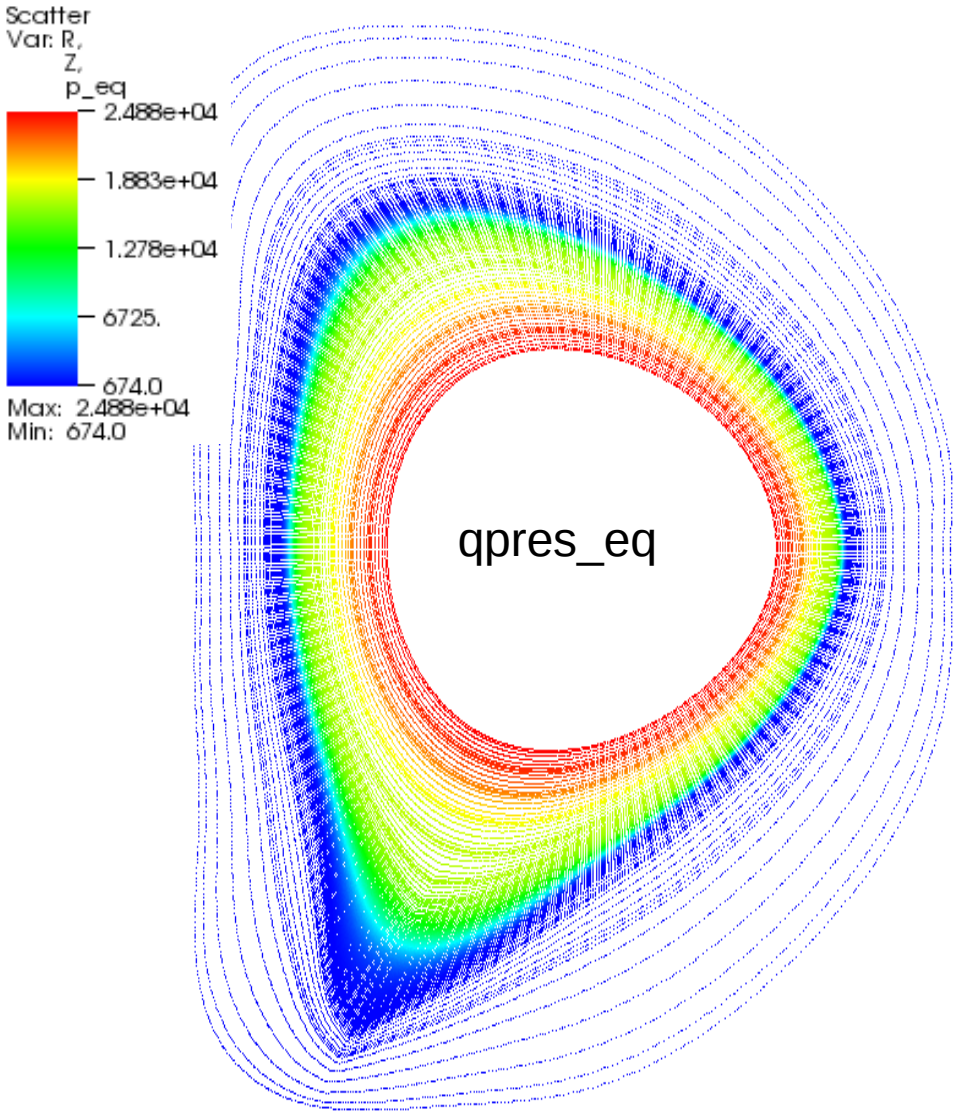
$$p(\psi > \psi_{PVI}) = pressure\_gauge$$

$$p_e(\psi > \psi_{PVI}) = pe\_frac \times pressure\_gauge$$

$$T_\alpha = \frac{p_\alpha}{k_b n_\alpha} \qquad J(\psi > \psi_{PVI}) = 0$$

# Optional output allows plotting of the analytic quadrature point evaluations.



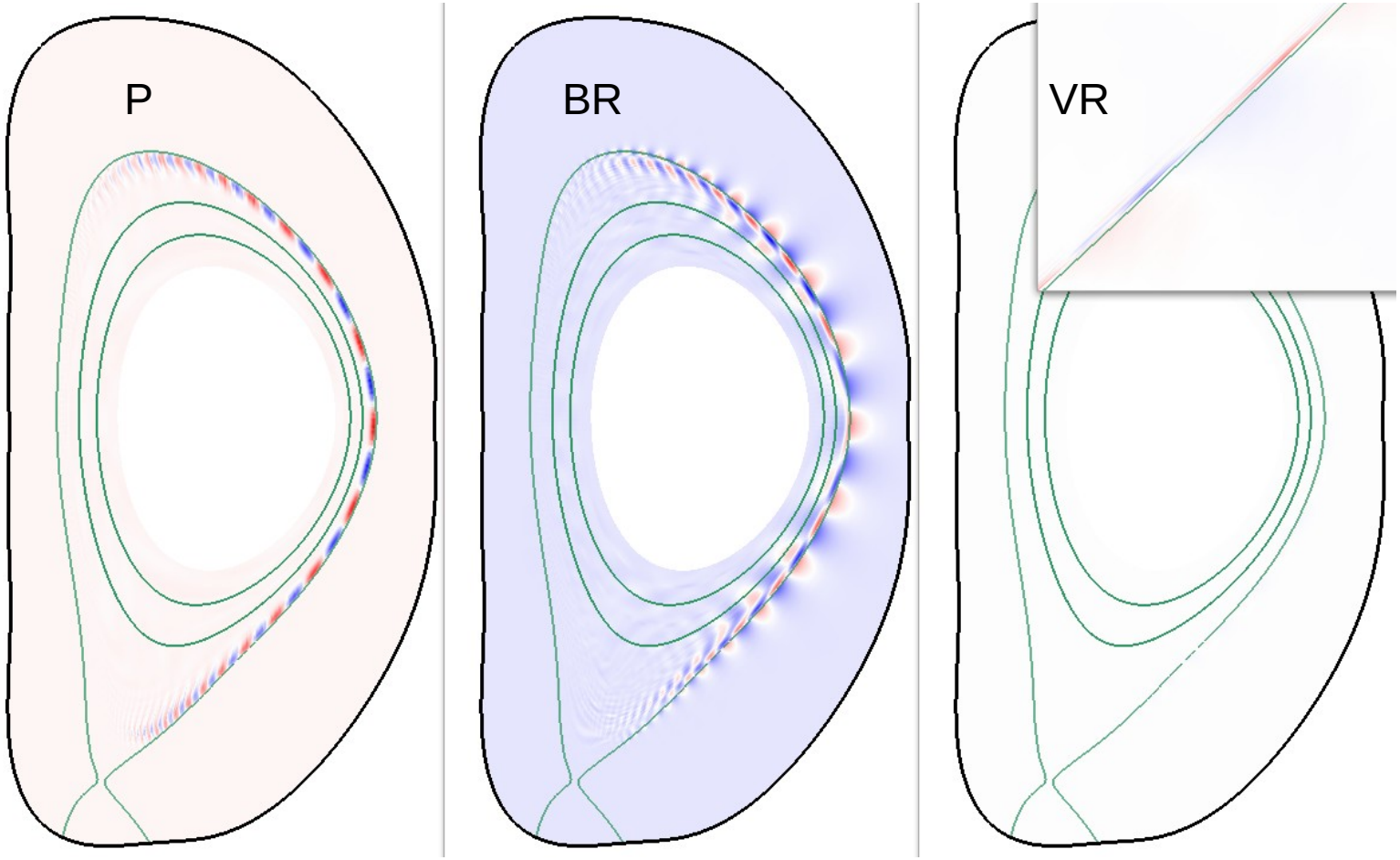
# Example modeling on an under resolved EHO case: a default without the ideal-like modifications.

- There are severe problems with the velocity and density fields.

$\gamma$=5.3e5 $s^{-1}$ (ELITE: 1.4e5 $s^{-1}$)

$k^2_{divB}$=10

Case parameters:
(high res)
mx=36 (48)
my=128 (512)
poly_degree=4
n=11
Ndens/nedge=100
dvac=1e7
pressure_gauge=
3.5
S=1.1e8
Pm=0.001
p_computation=
'at nodes'



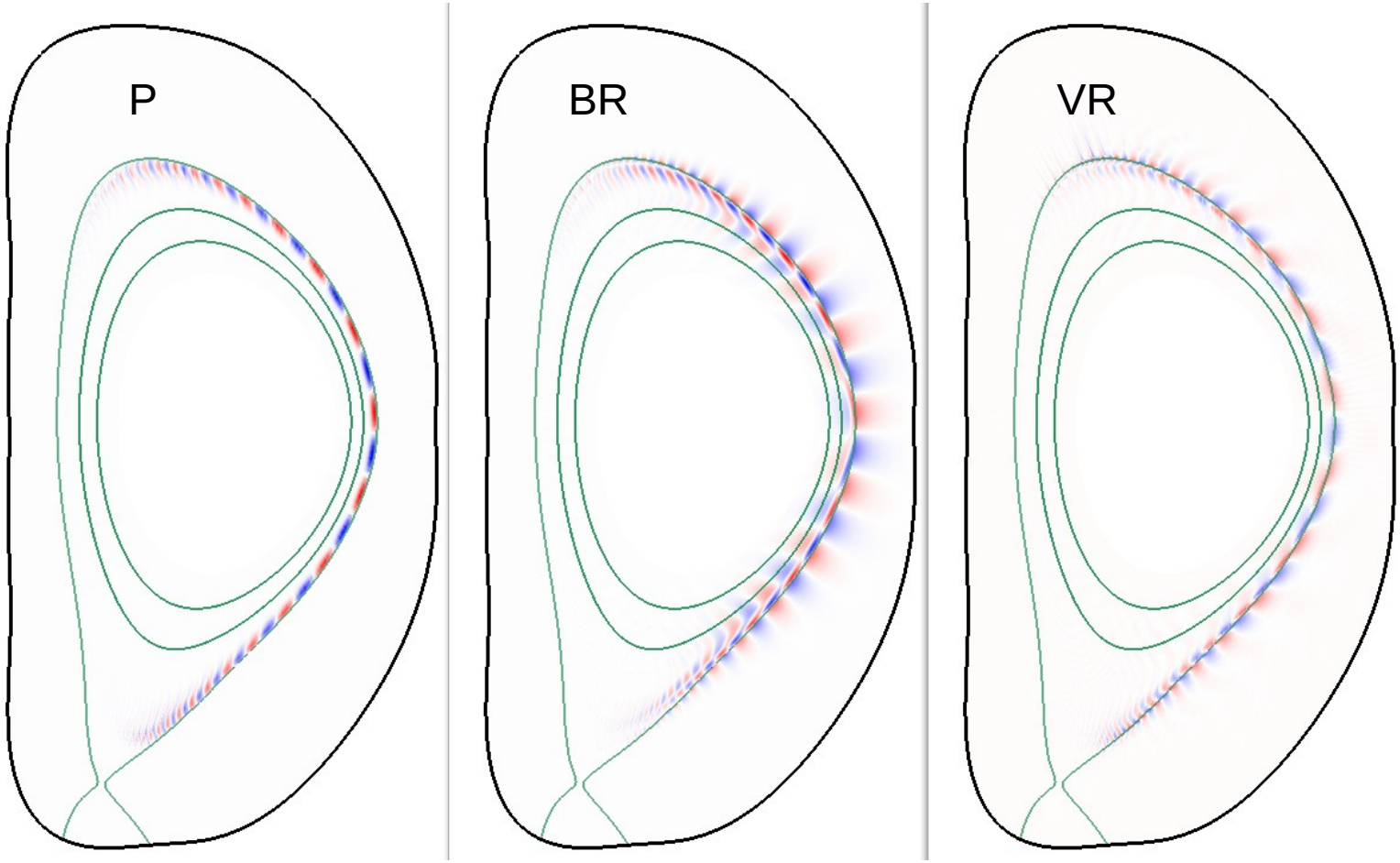
# Example modeling on an under resolved EHO case: with new ideal-like modifications.

- Fields are cleaner and the mode growth rate is more reasonable.

$\gamma$=5.1e5 $s^{-1}$ (ELITE: 1.4e5 $s^{-1}$)

$k^2_{divB}$=3

Case parameters:
(high res)
mx=36 (48)
my=128 (512)
poly_degree=4
n=11
Ndens/nedge=100
dvac=1e7
pressure_gauge=
3.5
S=1.1e8
Pm=0.001
ideal_like=.true.
set_vac_quadpts=
.true.



# Open issues for discussion.

- Convergence can still be challenging to achieve. Parameters dvac, $n_{edge}$ and pressure_gauge must be chosen carefully.
- It is not clear what the numerically optimal value for pressure_gauge is (physically, it can take any value).
  - pressure_gauge sets the temperature (and thus the sound speed) in the vacuum region
  - If too large, experience has shown unmagnetized sound waves can corrupt the computation and cause an over-stable mode (as pointed out by Carl).

# Ideal-like modeling summary

- Ideal-like modeling is much easier and this should aid code comparisons.
  - However, it still isn't *easy.* But it used to be extremely difficult.
- These ideal-like cases are helped by two code modifications:
  - Analytic specification of equilibrium values at the quadrature points, where appropriate.
  - Advance of v, B and p and the primitive variables (which require no evaluations of discontinuous equilibrium quantities).

# Topic II: Interfacing NIMROD with MUMPS

# Why interface NIMROD with MUMPS?

- Interfacing NIMROD with MUMPS (“MUltifrontal Massively Parallel Solver”, http://mumps.enseeiht.fr) is interesting because it uses a different numerical approach than SuperLU_DIST (right-looking factorization/solve), so differences in performance can be algorithmic not just implementation based (see http://crd-legacy.lbl.gov/~xiaoye/SuperLU/SparseDirectSurvey.pdf).
- MUMPS can also be built with OpenMP threading which provides a mixed-parallelism model not present with SuperLU (except through BLAS calls).
- Previously, Ben Jamroz reported a significant speed-up with MUMPS relative to SuperLU_DIST for real NIMROD matrices interfaced to the solvers through PETSc.
- Personally, I hope to use MUMPS to efficiently run edge cases with a higher-resolution poloidal mesh by achieving better scaling and memory usage.


# There are some differences between the MUMPS interface and other solvers (SuperLU_DIST, PaStiX).

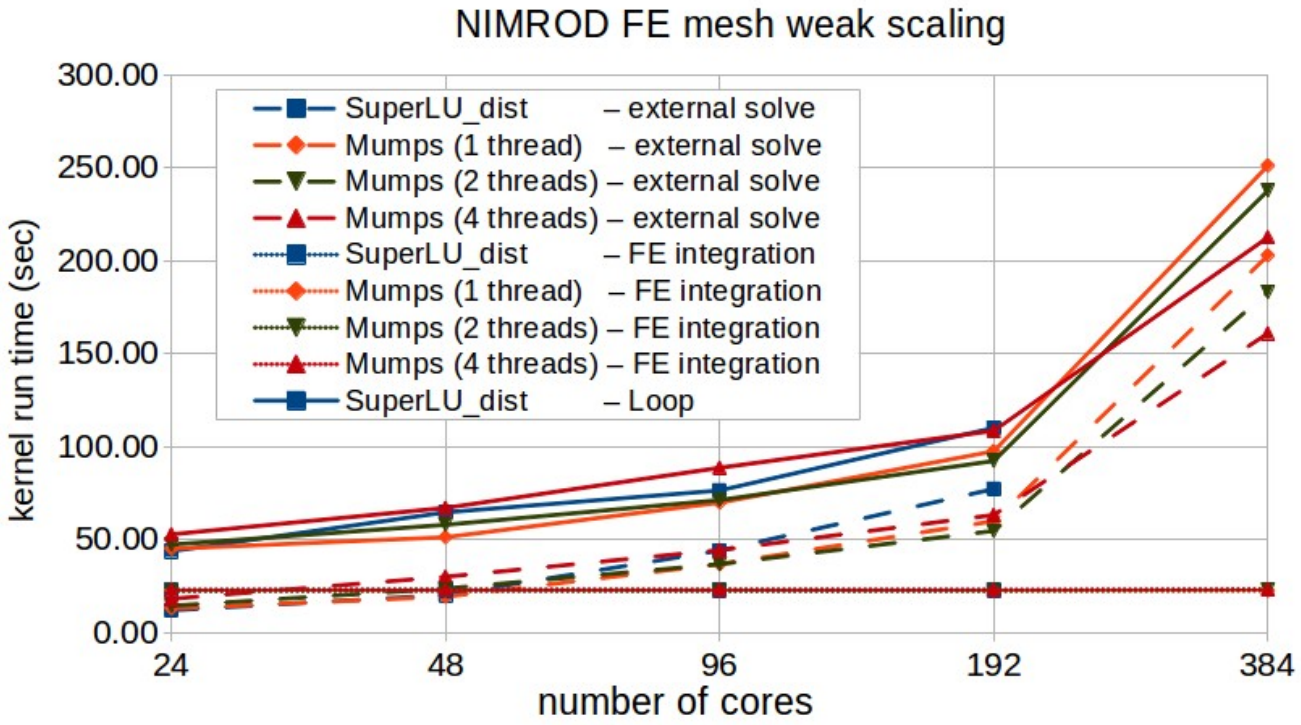
- The matrix sparsity pattern for MUMPS is a coordinate list format (row, column) instead of compressed sparse row/column. This requires modifications to iter_utils.F90 during the matrix construction.
- Although the distributed matrix is passed to MUMPS during factorization (as with slu_dstm/a), the global RHS is required during the solve. This requires an MPI_allgather call similar to slu_dist.
- MUMPS can return a distributed solution; however, there no guarantee that the data locality matches the input matrix rows (unlike SuperLU_DIST)! Thus the values must be communicated after each solve.
- MUMPS can also simply return the global solution vector on node zero. This combined with an MPI_broadcast proved to be ~ 20% slower than SuperLU_DIST.

# The main challenge of the implementation was the redistribution of the solution.

- The code on the right shows the transfer_soln subroutine which is called after the solve phase.
- This code involves asynchronous point-to-point communication where local values are sorted while waiting.
- Our initial experience is the values are NOT local relative to the matrix row indices passed during factorization.
- There is another routine (setup_comm_arrays) which distributes the communication array information on the first solve after any factorization step.

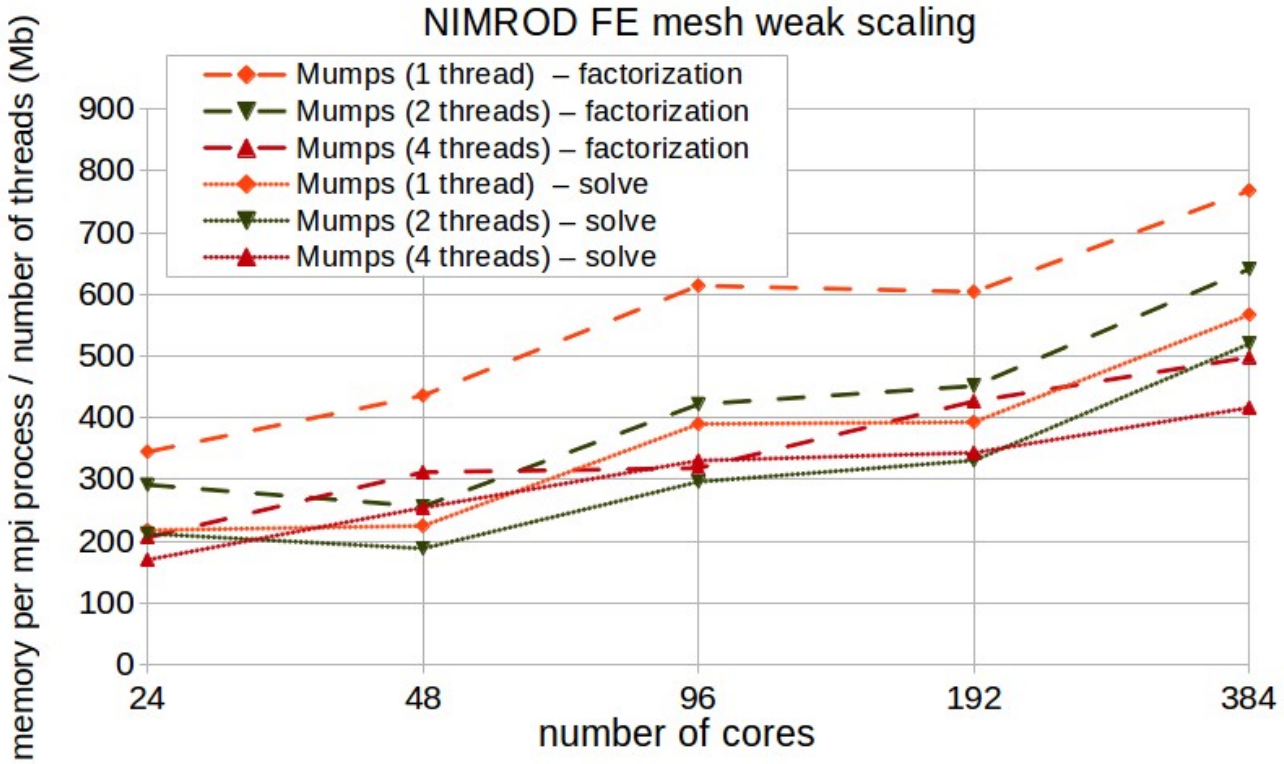
```
!-----------------------------------------------------------------------
! subprogram 3. transfer_soln
! swap the distrubted solution among nodes.
!-----------------------------------------------------------------------
SUBROUTINE transfer_soln(spp,mumps_par,sol)
  USE pardata, ONLY: nprocs_layer,comm_layer,node_layer
  IMPLICIT NONE
  TYPE(sparsity_pattern), INTENT(INOUT) :: spp
  TYPE(zmumps_struc), INTENT(INOUT) :: mumps_par
  COMPLEX(r8), INTENT(INOUT) :: sol(:)
  INTEGER(i4) :: iproc,ii,is,ierr
  COMPLEX(r8), ALLOCATABLE :: send_sol(:,:),recv_sol(:,:)
  INTEGER(i4), ALLOCATABLE :: mpi_requests(:,:)
  INTEGER(i4), ALLOCATABLE :: mpi_statuses(:,:)
!-----------------------------------------------------------------------
! send data then transfer to the local soln vector while waiting
!-----------------------------------------------------------------------
  ALLOCATE(send_sol(spp%maxsend,0:nprocs_layer-1))
  ALLOCATE(recv_sol(spp%maxrecv,0:nprocs_layer-1))
  ALLOCATE(mpi_requests(0:nprocs_layer-1,2))
  ALLOCATE(mpi_statuses(mpi_status_size,0:nprocs_layer-1))
  mpi_requests=mpi_request_null
  DO iproc=0,nprocs_layer-1
    IF(iproc == node_layer) CYCLE
    IF(spp%mumps_nsend(iproc)>0) THEN
      DO ii=1,spp%mumps_nsend(iproc)
        send_sol(ii,iproc)=                                          &
          mumps_par%sol_loc(spp%mumps_sisol(iproc)%irow(ii))
      ENDDO
      CALL mpi_isend(send_sol(:,iproc),spp%mumps_nsend(iproc),       &
                     mpi_nim_comp,iproc,0_i4,comm_layer,             &
                     mpi_requests(iproc,1),ierr)
    ENDIF
    IF(spp%mumps_nrecv(iproc)>0) THEN
      CALL mpi_irecv(recv_sol(:,iproc),spp%mumps_nrecv(iproc),       &
                     mpi_nim_comp,iproc,0_i4,comm_layer,             &
                     mpi_requests(iproc,2),ierr)
    ENDIF
  ENDDO
  DO ii=1,spp%nlsol
    sol(spp%mumps_lisol(2,ii))=mumps_par%sol_loc(spp%mumps_lisol(1,ii))
  ENDDO
!-----------------------------------------------------------------------
! recieve data and sort
!-----------------------------------------------------------------------
  DO is=1,spp%ncrecv
    CALL mpi_waitany(nprocs_layer,mpi_requests(:,2),iproc,           &
                     mpi_statuses,ierr)
    iproc=iproc-1 ! 0 indexing
    DO ii=1,spp%mumps_nrecv(iproc)
      sol(spp%mumps_risol(iproc)%irow(ii))=recv_sol(ii,iproc)
    ENDDO
  ENDDO
  CALL mpi_waitall(nprocs_layer,mpi_requests(:,1),mpi_statuses,ierr)
  DEALLOCATE(send_sol,recv_sol,mpi_requests,mpi_statuses)
END SUBROUTINE transfer_soln
```

# A weak-scaling study shows MUMPS does not scale or perform significantly better than SuperLU_DIST.




- This case uses 50 linear time steps with 192 poly_degree 6 FEs per processor. Performance data is from Edison at NERSC.
- The range of v-matrix rows and NNZ for the smallest to largest cases is 1.5e5-2.4e6 and 2.0e7-3.2e8, respectively
- Threading helps scaling, but the effect is small.
- MUMPS is slightly faster, the MUMPS case with 2 threads has a 17% faster loop time with 192 processors.
- SuperLU_DIST runs out of memory with the last case at 384 processors.

# However, the weak scaling of the MUMPS memory usage is very good.




- When NIMROD is interfaced with MUMPS, larger cases can be run than with SuperLU_DIST.
- Threading further decreases the MUMPS (and NIMROD) memory footprint.

# NIMROD interface with MUMPS summary

- The performance of MUMPS is not significantly better than SuperLU_DIST in terms of time to solution or scaling.
- However, the memory scaling of MUMPS is found to be better than SuperLU_DIST (in addition to good performance with threads). Thus MUMPS may offer advantages for large cases.

# NIMROD with MUMPS OpenMP compilation

- Both NIMROD and MUMPS are compiled with OpenMP threading enabled.
- MUMPS uses ScaLAPACK, BLACS and BLAS through the threaded version of Cray's libsci.
- The binary is also linked to Google's tcmalloc (thread-caching malloc) to avoid thread locks during memory allocation/deallocation in threaded regions.
- MUMPS has many ordering options (ParMetis, Pt-Scotch, PORD, etc). PORD was found to be best before the distributed solution interface was implemented, and thus further optimization of the ordering may be possible.