OVERFLOW Training

Joseph Derlaga

Pieter Buning

Chip Jackson

15th Symposium on Overset Composite Grids and Solution Technology

November 1-3, 2022

Introduction

- Some philosophy
- Getting started, compiling, and input files
- Making use of changes in 2.2/2.3/2.4
- Suggestions
- Q&A

Setting the stage

- Please interrupt and ask questions!

Setting the stage

- Please interrupt and ask questions!
- Our goals for this tutorial are to…
  - Provide an overview of the OVERFLOW workflow without too many details
  - Discuss new features that have been added over the last 4 years

Setting the stage

- Whether we're talking low-fidelity or high-fidelity, CFD codes are nothing more than tools that implement models which are based on assumptions
  - Some models are grounded in physical arguments, some from a mathematical basis, and others are tied to experimental correlations; or any combination thereof

Setting the stage

- The impact these models have on a solution are often difficult to predict for anything beyond simple examples
  - Real problems are a mixture of simpler problems, so synthetic benchmarks can be misleading

Setting the stage

- We care about how well our solutions match real life despite all the assumptions as well as how fast we can get those solutions.
  - There are many assumptions that can provide useful results, just maybe not for your specific problem
  - When something goes ‘really wrong’ it is a safe bet to blame either the turbulence model or the grid
  - Corollary: Everything is wrong, but that’s ok!

Setting the stage

- We care about how well our solutions match real life despite all the assumptions as well as how fast we can get those solutions.
  - There are many assumptions that can provide useful results, just maybe not for your specific problem
  - When something goes ‘really wrong’ it is a safe bet to blame either the turbulence model or the grid
  - Corollary: Everything is wrong, but that’s ok!
    - As long as it’s not ‘really wrong’

What is OVERFLOW?

- Hybrid FD/FV Navier-Stokes solver for structured, overset grids
- Has a variety of flux schemes, turbulence/transition models, time advancement schemes, linear solvers, etc.
- Has been used across a wide variety of flow regimes

Changes (2018-2022)

- OVERFLOW 2.2
  - Mixed Peg5/DCF mode
  - Rotor disk improvements
- OVERFLOW 2.3 (Late 2019)
  - CFL ramping
  - Improved linearizations
  - Implicit physical boundary conditions
  - New BC’s, new QCR models, turb/transition model updates
- OVERFLOW 2.4 (Mid 2022)
  - GLOBAL_LINEAR_SOLVE (GLS)
  - Bug fixes…

Getting started

Step 0: Compiling

- Extracting the tarball
  - There are symlinks in the OVERFLOW distribution, you may need to untar with the -h option to maintain those symlinks
  - This problem will most likely manifest when you go to build the tools and get errors like ‘file not found’

Step 0: Compiling

- OVERFLOW has had a Make based build system for many, many years
  - Set the MPI_ROOT such that mpi.h is in $MPI_ROOT/include
  - Choose a compiler/architecture flag from Make.sys and run ./makeall <arch>
  - Pay attention to the architecture flag you choose if running on a heterogenous system, some executables may not be backwards compatible with older hardware

Step 0: Compiling

- A CMake based build system has been recently added and we recommend testing it out because it will replace the old Make system
  - Build happens in a separate directory from the source, avoids clutter
  - Can have multiple builds for different compilers/options and don't have to have multiple source directories/rebuild from scratch every time
  - Will automatically search for your MPI install based on mpiexec, but if something goes wrong, you can work around it
- Run ./cmakeall <F90 compiler> <C Compiler> <OpenMP Logical>
  - ./cmakeall gfortran gcc False
  - ./cmakeall mpifort mpicc True
  - Will automatically create source/bin & source/bin_dp

Step 0: Compiling

- If you do run into problems and need some help
  - Clean things up (make –f Makefile gfortran10 CMD=clean, or rm -rf <build_dir>)
  - Rebuild and pipe the build output into a file (./makeall gfortran10 &> made.out)
  - Send along that output file and the source/include/cmpltm.h file
- How does this help?
  - Starting clean will be sure to capture any problems that might get hidden by a rebuild
  - The last error message on the screen is seldom the most useful, it may just be the last output from a cascade of failures
  - The cmpltm.h file has useful info on loaded modules and you/we just might notice a conflict when you/we review it

Step 1: Geometry/Gridding

- OVERFLOW has companion software available from software.nasa.gov
  - Chimera Grid Tools for grid generation and other pre/post-processing tasks like X-ray creation or solution monitoring
  - Pegasus 5 for static overset grid assembly
- There are other options out there

Step 1: Geometry/Gridding, cont.

- OVERFLOW mode (static geometry)
  - User supplies all grids
  - Grid cutting/assembly done with Peg5 based on cell sizes
  - High-quality grid system

Step 1: Geometry/Gridding, cont.

- OVERFLOW mode (static geometry)
  - User supplies all grids
  - Grid cutting/assembly done with Peg5 based on cell sizes
  - High-quality grid system
- OVERFLOW-D mode (moving geometry/adaptation)
  - User supplies near-body (curvilinear) grids, X-rays, and hole cutting distances
  - OVERFLOW creates off-body (Cartesian) grids and cuts/assembles with DCF
  - Lower-quality than OVERFLOW mode, but fast enough for moving body cases

Step 1: Geometry/Gridding, cont.

- OVERFLOW mode (static geometry)
  - User supplies all grids
  - Grid cutting/assembly done with Peg5 based on cell sizes
  - High-quality grid system
- OVERFLOW-D mode (moving geometry/adaptation)
  - User supplies near-body (curvilinear) grids, X-rays, and hole cutting distances
  - OVERFLOW creates off-body (Cartesian) grids and cuts/assembles with DCF
  - Lower-quality than OVERFLOW mode, but fast enough for moving body cases
- Mixed mode
  - Assemble near-body grids with Peg5, then use DCF and X-rays to create, cut, and assemble off-body grids

Peg5/DCF Mixed Mode

- Why use mixed mode?
  - You have complicated geometry with features not limited to things like small gaps, many protuberances, etc. where creating X-rays and choosing the cutter distance is difficult
    - If Peg5 can't assemble the grids properly, consider rebuilding your grids!
  - Hole cutting is expensive, by eliminating many of the X-rays you can reduce the cost
    - Only cut IGXLIST=-1
    - Cut other near-body grids with a single X-ray rather than many
  - You want better quality near-body connectivity, but still need moving grid capability/adaptation

Peg5/DCF Mixed Mode

- See the test/capsule_sep_mixed_mode example
  - makegrids
- Generate near body grids and create the X-rays you want to cut other near-body grids and the background grids
  - makecapsule, makebooster
- Modify grid.in so that any separate bodies will not talk to each other when running Peg5, generate the connectivity, and then move the grids back together
  - makepeg
- Use XRINFO to specify cutters and OVERFLOW will use the XINTOUT file for the near-body assemblies

Step 2: Input Files

- OVERFLOW mode
  - over.namelist*
  - grid.in
  - mixsur or usurp files
  - XINTOUT
- OVERFLOW-D mode
  - over.namelist*
  - grid.in
  - mixsur or usurp files
  - xrays.in
  - Config.xml, Scenario.xml
- Peg5/DCF Mixed mode

  Union of OVERFLOW & OVERFLOW-D modes

*If using overrun/overrunmpi, this is what your namelist file gets copied to

NB: Filenames matter! 'Config.xml' .NE. 'config.xml'

Step 2: Input Files

- It is always a good idea to make sure you've got everything setup properly before submitting a full job
  - Set &OMIGLB IRUN=2 to double check that the DCF process is working
  - For rigid body motion cases, reset IRUN and turn off the flow solvers by setting ITER = ITERT = ITERC = 0 and then you can check for connectivity issues without doing flow solves
- You can do these checks on fewer processors, but your orphan points may change slightly during the real run

Rotor Disk Model

- Had major additions and a rewrite by Jasim Ahmad for OVERFLOW 2.3
- There is too much to cover here, but more details can be found in
  - doc/readmes/README.rotor_disk
  - test/xv15
- Designed for rotorcraft, but can be used for propellors
  - Need to convert between conventions

Rotor Disk Model

- Grid requirements
  - No coning in the grid, but can be approximately handled in the input file
  - J is the axis of rotation => JBCS:JBCE spreads the source term over planes
  - K is azimuthal => constant psi; axisymmetric, periodic, or manually split
  - L is radial => concentric circles increasing from root to tip

Rotor Disk Model

- Grid requirements
  - No coning in the grid, but can be approximately handled in the input file
  - J is the axis of rotation => JBCS:JBCE spreads the source term over planes
  - K is azimuthal => constant psi; axisymmetric, periodic, or manually split
  - L is radial => concentric circles increasing from root to tip
- Suggestions
  - This is a model; you're not trying to resolve detailed flow physics for the rotor surface
  - Use 10% chord spanwise spacing at tip and root, same for streamwise spacing when using JBCS=JBCE

Rotor Disk Model

- Varying levels of blade effect fidelity
  - MODEL = 0: Disk with uniform axial loading, set CTHRUST
  - MODEL = 1: Ideal lift curve slope + compressibility corrections
  - MODEL = -1: Ideal lift curve slope, no corrections, use for startup
  - MODEL = 2: Use C81 tables to set radial blade information
  - Tip and Root loss models

Rotor Disk Model

- Varying levels of blade effect fidelity
  - MODEL = 0: Disk with uniform axial loading, set CTHRUST
  - MODEL = 1: Ideal lift curve slope + compressibility corrections
  - MODEL = -1: Ideal lift curve slope, no corrections, use for startup
  - MODEL = 2: Use C81 tables to set radial blade information
  - Tip and Root loss models
- Can trim to thrust, pitch, and roll
  - Use collective/cyclic or rotor speed/cyclic

Rotor Disk Model

- Varying levels of blade effect fidelity
  - MODEL = 0: Disk with uniform axial loading, set CTHRUST
  - MODEL = 1: Ideal lift curve slope + compressibility corrections
  - MODEL = -1: Ideal lift curve slope, no corrections, use for startup
  - MODEL = 2: Use C81 tables to set radial blade information
  - Tip and Root loss models
- Can trim to thrust, pitch, and roll
  - Use collective/cyclic or rotor speed/cyclic
- Can specify motion
  - MOTIONFILE to get pilot input or blade deflections

CFL Ramping

- Uses Switched Evolution Relaxation to automatically ramp the CFL number based on how well the nonlinear residual is converging
- Inputs in &TIMACU
  - ITIME = 3 or 4 (constant CFL)
  - CFLMAX = initial CFL number
  - CFLMIN_LIMIT = if residual increases, do not go below this CFL value
  - CFLMAX_LIMIT = if residual decreases, do not ramp past this CFL value
  - RAMP_CFL = logical that turns ramping on or off

CFL Ramping

- If you’re running steady state and don’t know what to do for time stepping, this option is for you
  - But first, an aside…

Time Advancement

- OVERFLOW solves the nonlinear Navier-Stokes equations in either time-accurate or steady-state modes
  - At each ‘time’ step, we calculate an updated ΔQ, add that to Q, reevaluate the nonlinear residual => repeat
  - For explicit schemes, the ΔQ is calculated by basically scaling the nonlinear residual by an appropriate time step
  - For implicit schemes, we linearize a/the nonlinear operator and solve a linear problem to calculate ΔQ, but it also involves a time step
- The time step can be based off the CFL number (local time stepping) or a specified time step (global time stepping)

Time Advancement

- Time accurate
  - DTPHYS is how many grid units a particle will move at freestream velocity per time step
- Steady-state or subiterations (variable DT or CFL options)
  - DT is how many grid units a particle will move at sonic velocity per time step
  - ITIME = 0: time step = DT, variable CFL, limit with CFLMAX & CFLMIN
  - ITIME = 1 or 2: time step = DT scaled by metric Jacobian, ditto
  - ITIME = 3 or 4: time step = CFLMAX/(SUM(EIGENVALUES)), variable DT, limit with CFLMIN/CFLMAX

Time Advancement

- Dual or Newton time stepping?
  - Both schemes can reach 2nd order time accuracy if fully converged
  - Dual time stepping adds a pseudo-time term to the governing equations that helps iterate out various approximation errors and gives the user a little more control over the linear solver/implicit scheme
- DIRK schemes
  - Multistage schemes that trade off more work at each stage against improved time accuracy/larger time steps
  - Needs further investigation given recent code updates

CFL Ramping

- If you’re running steady state and don’t know what to do for time stepping, this option is for you
  - And… we’re back!

CFL Ramping

- If you’re running steady state and don’t know what to do for time stepping, this option is for you
  - Start with a CFLMAX = 1 and ramp from there
  - If you’re doing exploratory work, you can monitor the <runname>.out file and judge for yourself what a good CFLMAX_LIMIT should be
  - You can set different CFLMAX values per grid
  - On restart, reset your CFLMAX to the previous CFLMAX*RAMP_FACTOR

CFL Ramping

- If you’re running steady state and don’t know what to do for time stepping, this option is for you
  - Start with a CFLMAX = 1 and ramp from there
  - If you’re doing exploratory work, you can monitor the <runname>.out file and judge for yourself what a good CFLMAX_LIMIT should be
  - You can set different CFLMAX values per grid
  - On restart, reset your CFLMAX to the previous CFLMAX*RAMP_FACTOR
  - Which brings up another aside…

Restarting

- OVERFLOW will split grids as needed when running in parallel or if you set &GLOBAL -> MAX_GRID_SIZE
  - The split blocks have shared common nodes at the split location, but the solutions are probably going to be different at these shared points (unless you're running explicit time stepping!)
  - OVERFLOW reassembles the grids to create x.* and q.* files, so only one of these points can win
  - This mismatch is enough to cause spikes in the nonlinear residual on restart and is much more apparent as you get deeper convergence
- If you'd like to see the split grids, recompile with the -DDEBUG_SPLIT flag and you'll get x.split and q.split files

CFL Ramping

- If you’re running steady state and don’t know what to do for time stepping, this option is for you
  - Start with a CFLMAX = 1 and ramp from there
  - If you’re doing exploratory work, you can monitor the <runname>.out file and judge for yourself what a good CFLMAX_LIMIT should be
  - You can set different CFLMAX values per grid
  - On restart, reset your CFLMAX to what the previous CFLMAX*RAMP_FACTOR was
  - Back again!

CFL Ramping

- If you're running steady state and don't know what to do for time stepping, this option is for you
  - Start with a CFLMAX = 1 and ramp from there
  - If you're doing exploratory work, you can monitor the <runname>.out file and judge for yourself what a good CFLMAX_LIMIT should be
  - You can set different CFLMAX values per grid
  - On restart, reset your CFLMAX to what the previous CFLMAX*RAMP_FACTOR was
- For CFL ramping to work well, you need to have a linear solver that doesn't make too many approximations

Implicit Physical Boundary Conditions

- Without physical boundaries, we don’t have problems

Implicit Physical Boundary Conditions

- Without physical boundaries, we don’t have problems
  - Ignoring them in the linear solver means that we require the nonlinear solver to do all the heavy lifting
  - Rather than driving the bus, the boundary conditions are just along for the ride if they’re not included in the linearizations

Implicit Physical Boundary Conditions

- Without physical boundaries, we don’t have problems
  - Removing their affect from the linear solver means that we require the nonlinear solver to do all the heavy lifting
  - Rather than driving the bus, the boundary conditions are just along for the ride if they’re not included in the linearizations
- For OVERFLOW 2.3, physical boundaries that had nearest neighbor support were linearized
  - Technically, points where we apply strong boundary conditions aren’t really solution points, but treating them like they are is a convenience feature that works well with stationary solvers

Implicit Physical Boundary Conditions

- Greatly improves the ability of the solver to set up the gross flow field
  - Less time spent dealing with transient behavior and small fluctuations
- Can be more difficult to start problems
  - Can’t run from a cold start at very high CFL numbers, especially on coarse grids, because regions near boundaries now evolve faster and this means that high starting CFL numbers can destabilize everything… use CFL ramping!

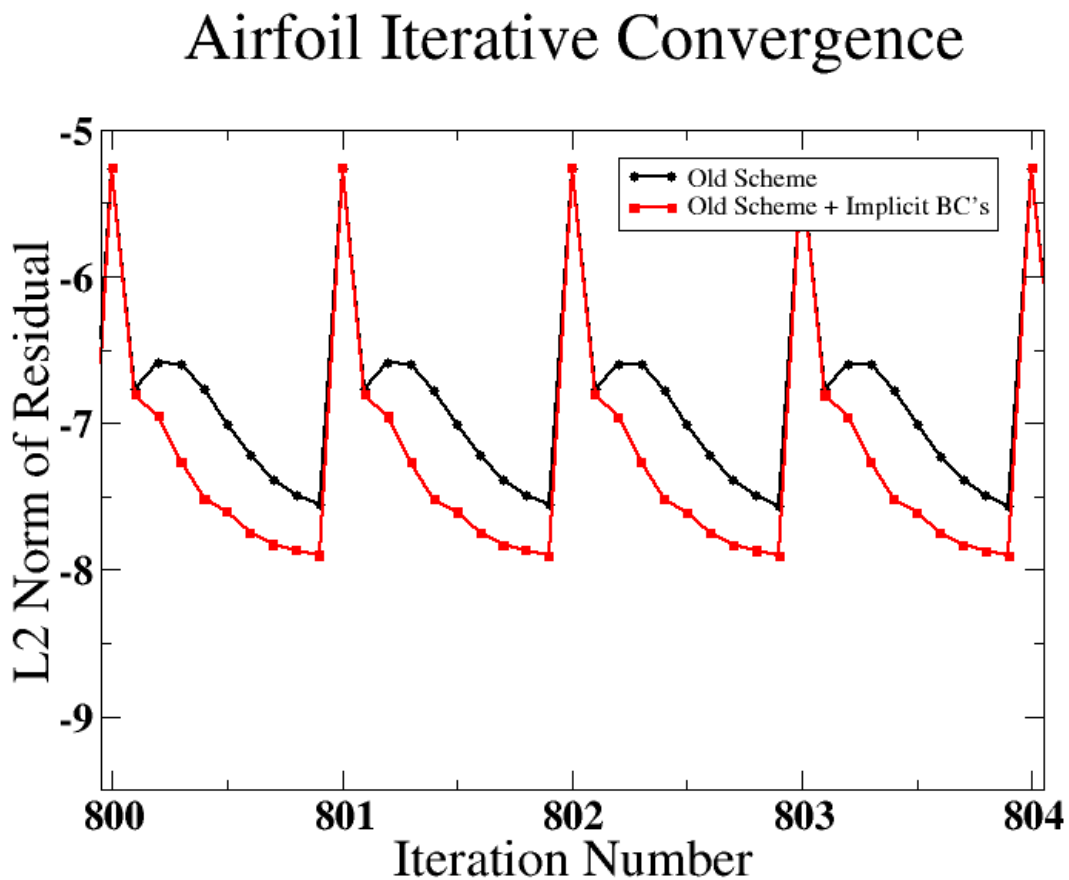
Improved Linearizations

- For implicit schemes, how well we asymptotically converge the nonlinear problem highly depends on how well our linearization matches the true linearization
  - Think of it as providing better directions

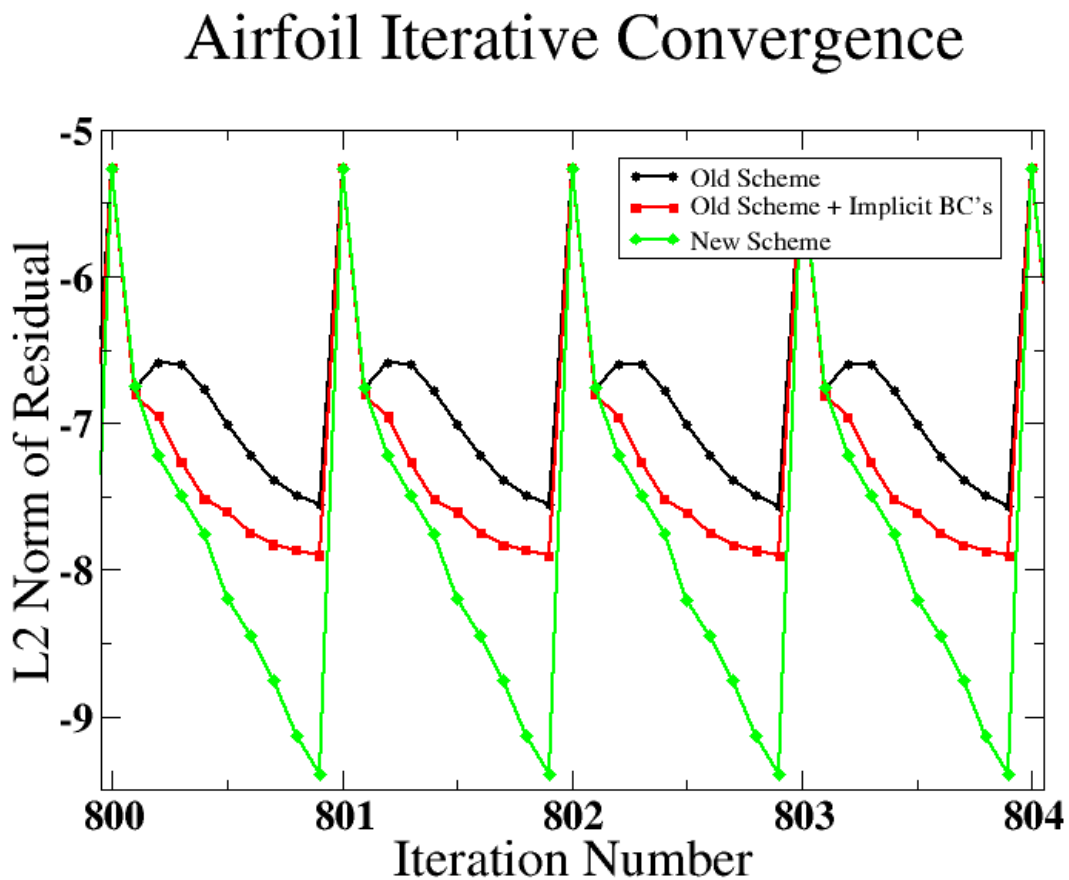
Improved Linearizations

- For implicit schemes, how well we asymptotically converge the nonlinear problem highly depends on how well our linearization matches the true linearization
  - Think of it as providing better directions
- Option to use ILHS=26/27/28 or 16/17/18 to get 32- or 64-bit linearizations of *FV* inviscid fluxes when using IRHS = 4/5/6
  - Change from ILHS=6/7/8 which uses linearization of *FD* Steger-Warming fluxes
  - In general, there is less dissipation, so supersonic flows might be a bit more unstable
  - 64-bit path is more costly, but can get you past some startup problems without segfaulting

OVERFLOW 2.2 vs 2.3



OVERFLOW 2.2 vs 2.3



Time check

- We’ve covered the changes to 2.2/2.3, now onto 2.4!

Global Linear Solve/GLS/RAS

- Tries to address a few problems
  - Poor convergence / sensitivity to grid splitting
  - Issues with supersonic cases not getting as many benefits as subsonic
  - Nonlinear residual jumps on restarts due to solution differences at common nodes

Global Linear Solve/GLS/RAS

- Tries to address a few problems
  - Poor convergence / sensitivity to grid splitting
  - Issues with supersonic cases not getting as many benefits as subsonic
  - Nonlinear residual jumps on restarts due to solution differences at common nodes
- Implicit overset boundaries
  - &GLOBAL -> GLOBAL_LINEAR_SOLVE = .TRUE./.FALSE.

Global Linear Solve/GLS/RAS

- Rather than performing block-Jacobi solves of independent linear problems, couple blocks together during the linear solver instead of just at the nonlinear solver
  - Exchange ΔQ after each symmetric sweep so that blocks that are not near the action have a more implicit treatment
  - Allows blocks that are near physical boundaries to avoid Dirichlet like block boundaries that can amplify transient behavior

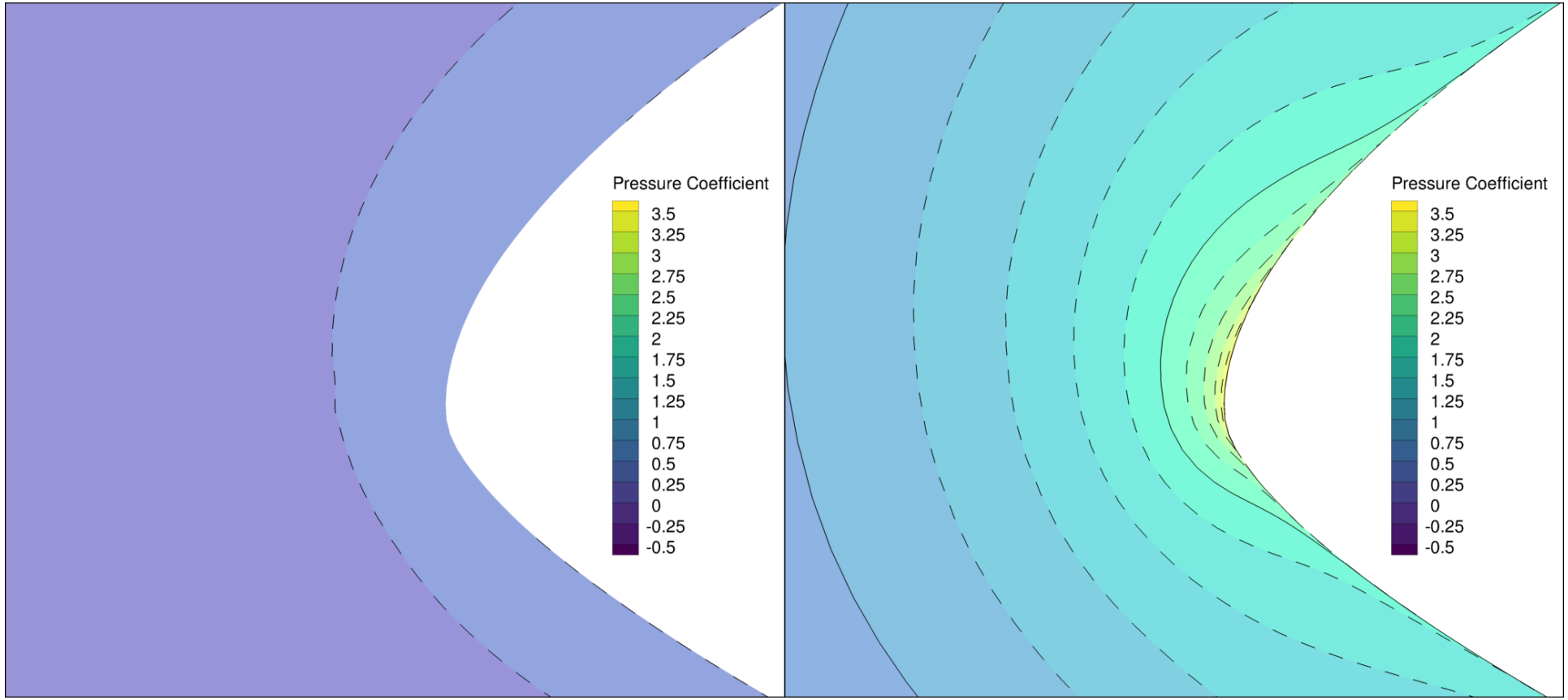
Global Linear Solve/GLS/RAS

- Rather than performing block-Jacobi solves of independent linear problems, couple blocks together during the linear solver instead of just at the nonlinear solver
  - Exchange ΔQ after each symmetric sweep so that blocks that are not near the action have a more implicit treatment
  - Allows blocks that are near physical boundaries to avoid Dirichlet like block boundaries that can amplify transient behavior
- Does not try to perfectly recreate SSOR without grid splitting, we're just trying to be more implicit
  - Trying to create a better preconditioner, too

GLS motivation

Original SSOR, 1 processor

Improved SSOR, 1 processor

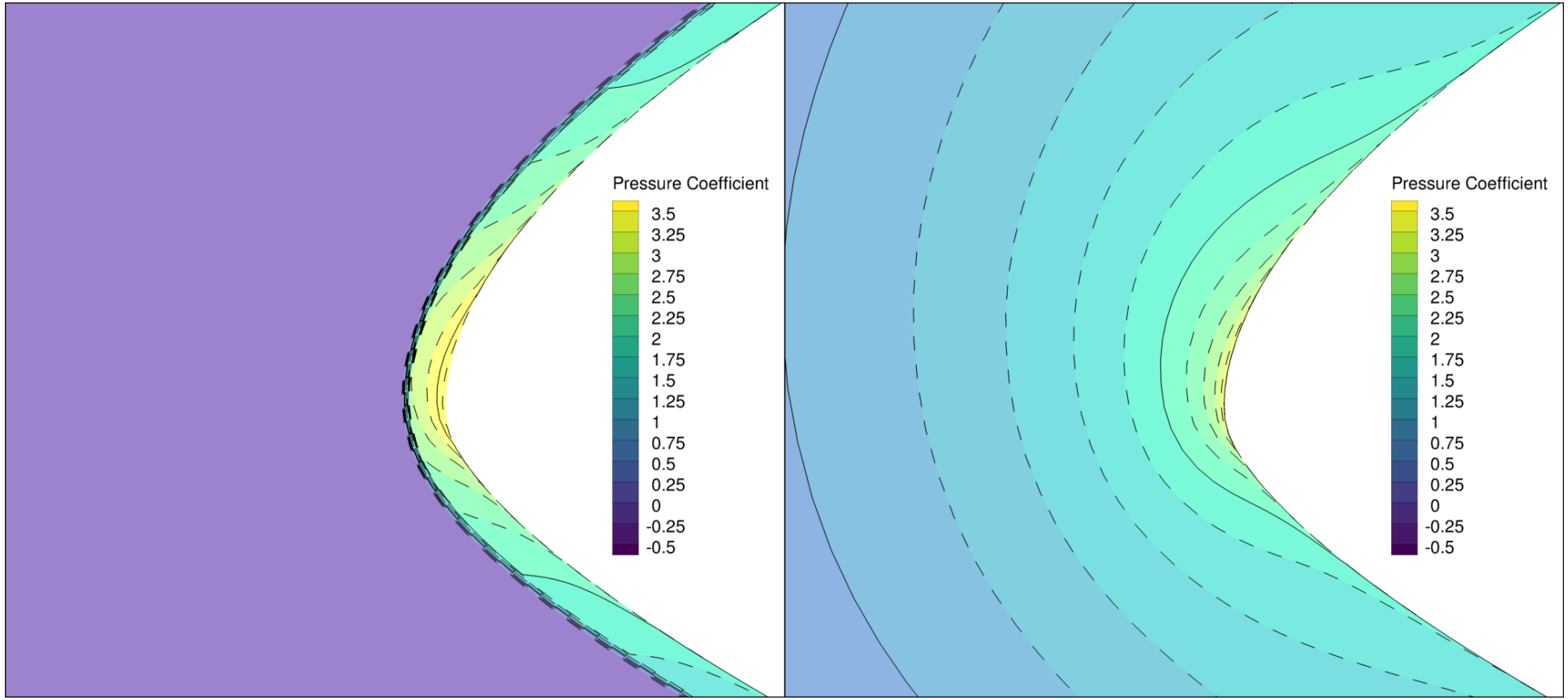


Impulsive start, one nonlinear step

GLS motivation

Improved SSOR, 8 processors

Improved SSOR, 1 processor

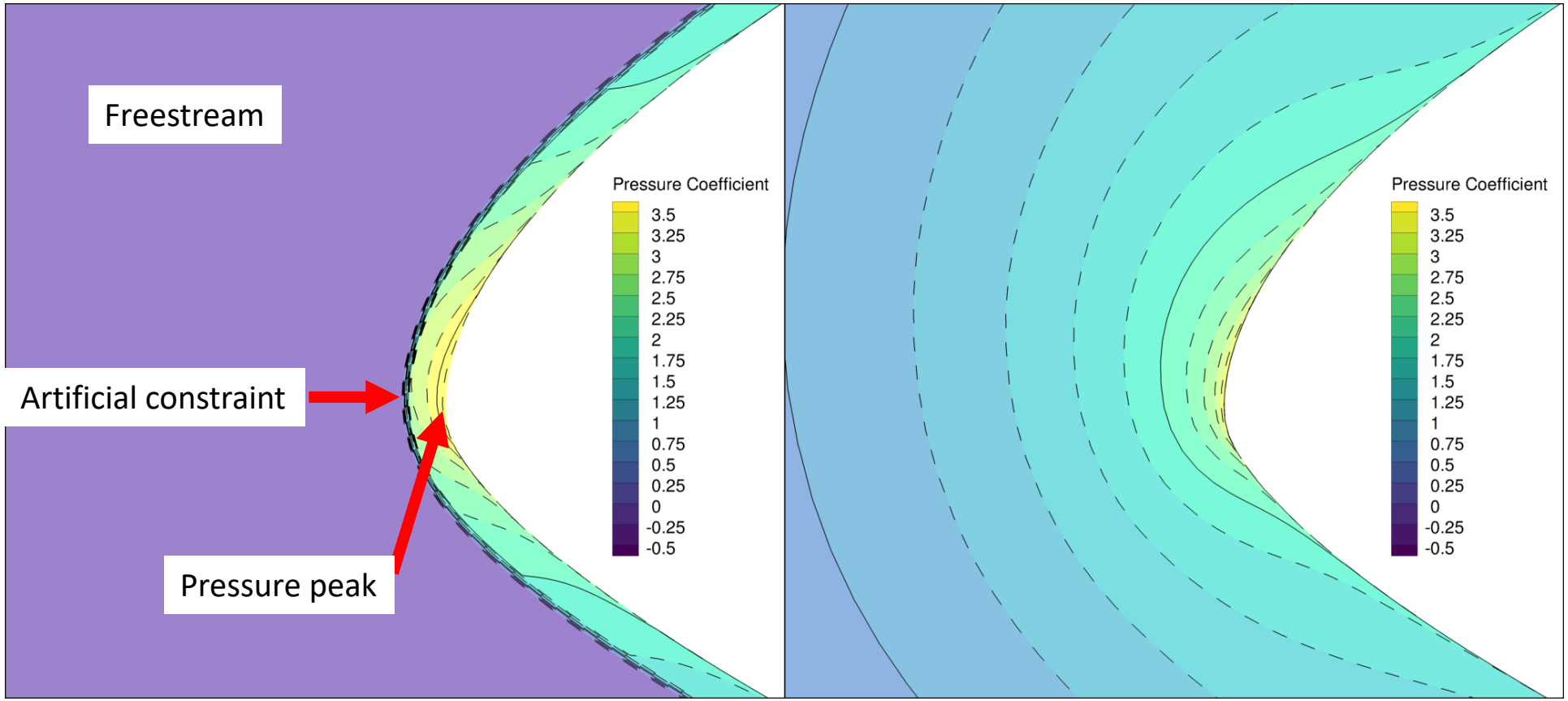


Impulsive start, one nonlinear step

GLS motivation

Improved SSOR, 8 processors

Improved SSOR, 1 processor

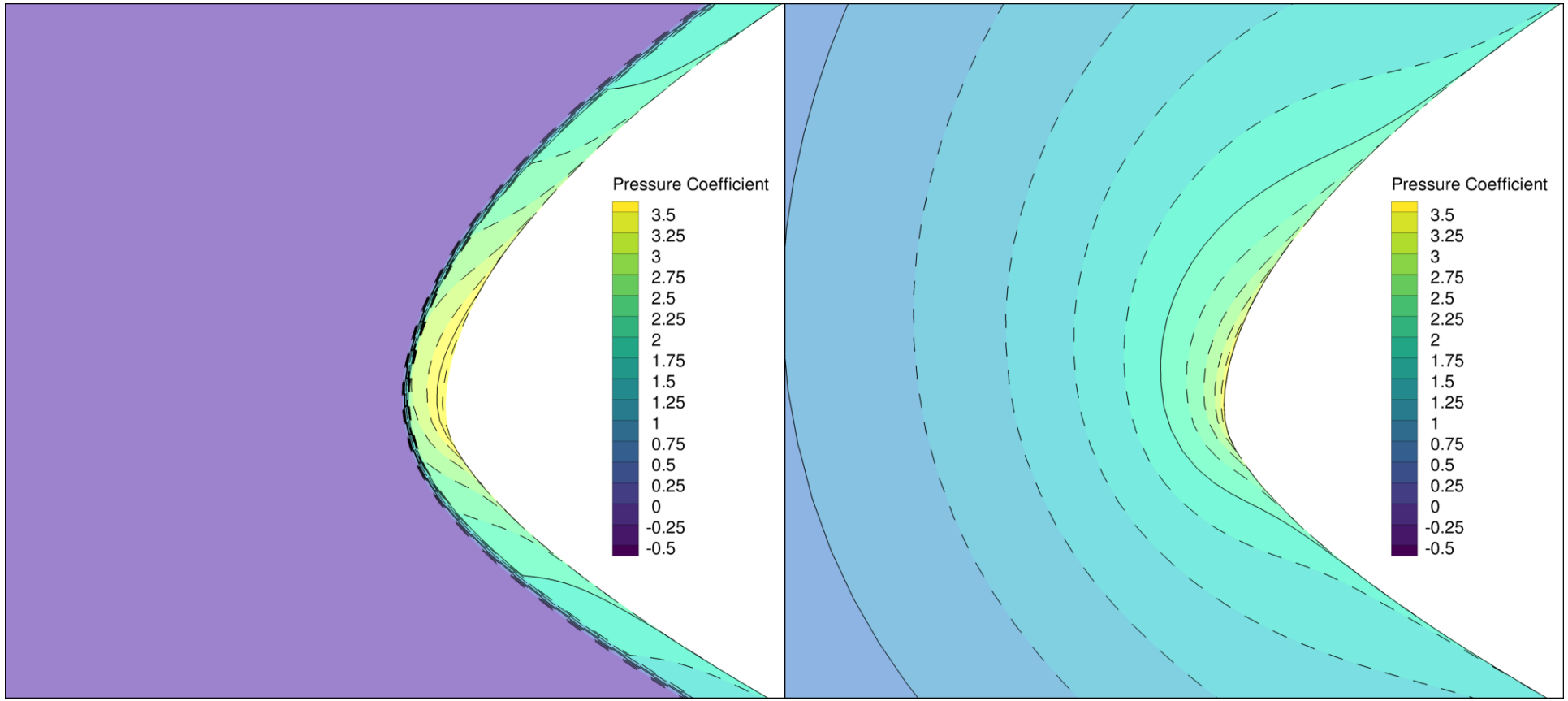


Impulsive start, one nonlinear step

GLS motivation

Improved SSOR, 8 processors

Improved SSOR, 1 processor

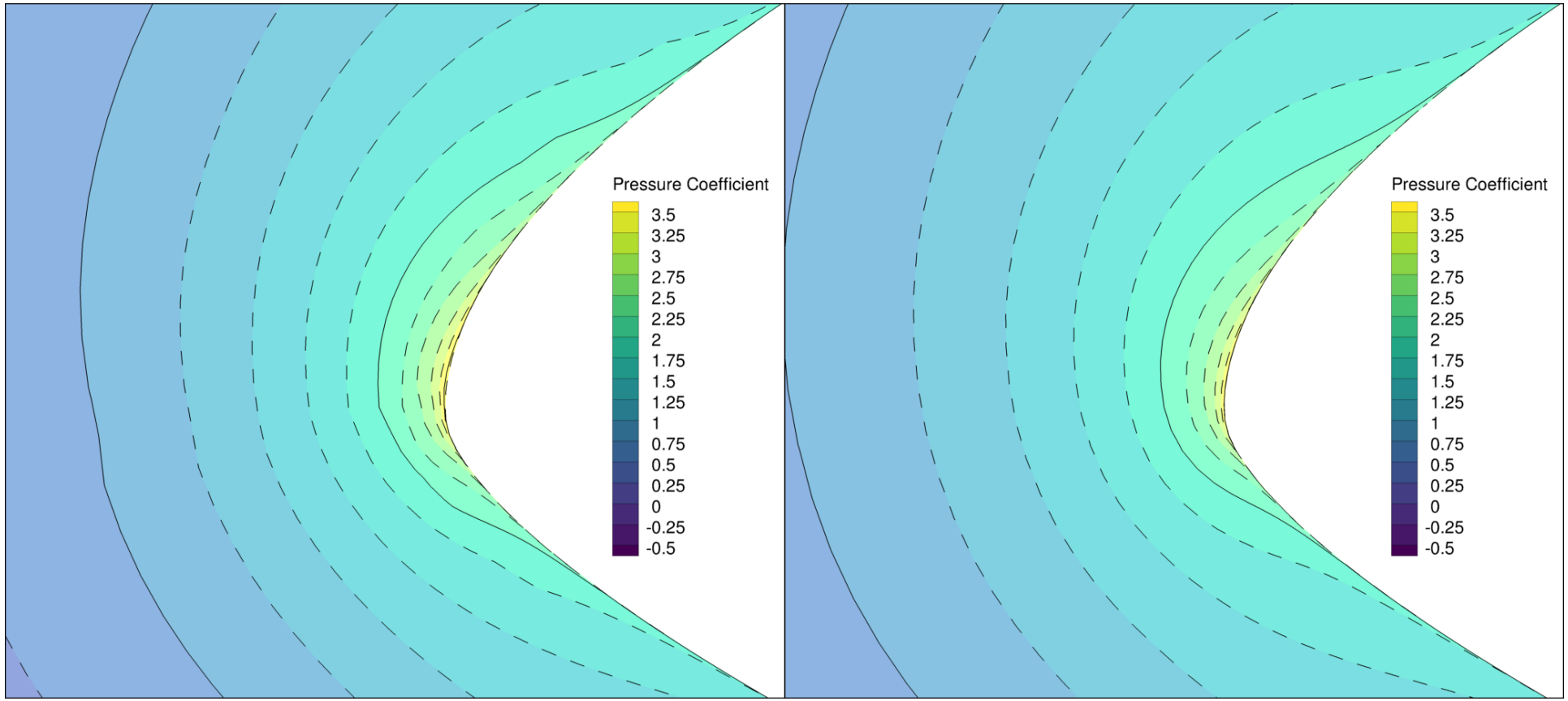


Impulsive start, one nonlinear step

GLS in action

Parallel SSOR, 8 processors

Improved SSOR, 1 processor

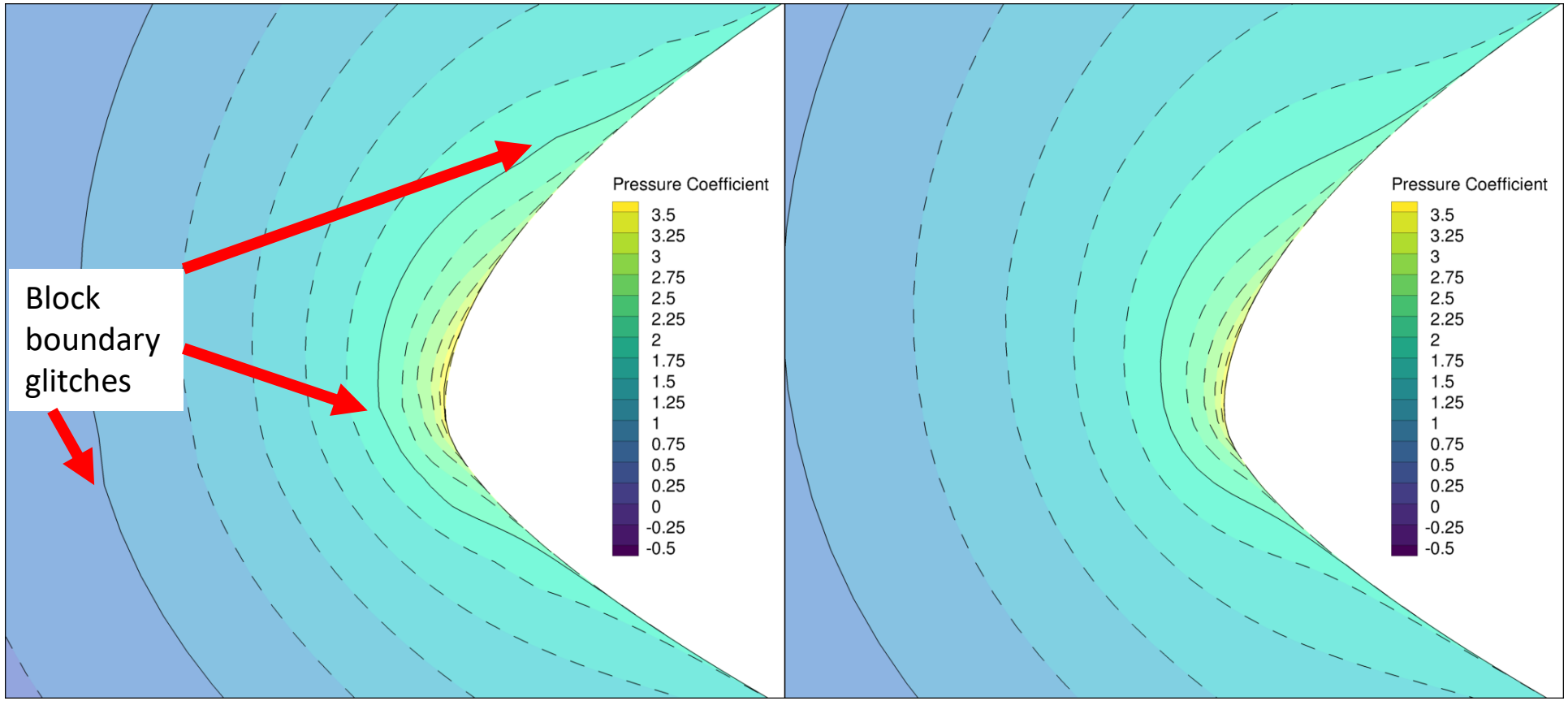


Impulsive start, one nonlinear step

GLS in action

Parallel SSOR, 8 processors

Improved SSOR, 1 processor

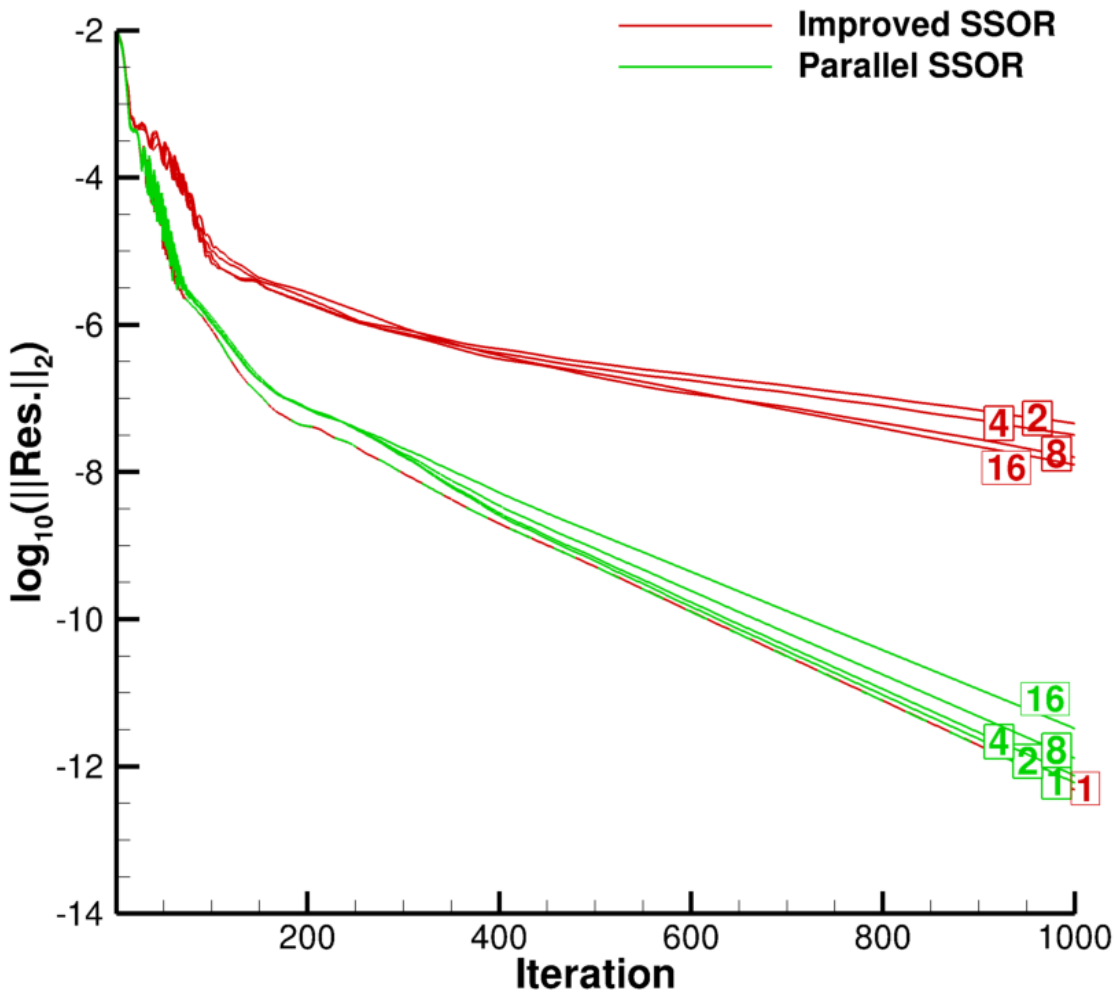


Impulsive start, one nonlinear step

Global Linear Solve/GLS/RAS

- You should not see convergence rates decrease as much when running with more processors

GLS in action



Global Linear Solve/GLS/RAS

- You should not see convergence rates decrease as much when running with more processors
  - You can still get unlucky with grid splitting and run into a stability issue, but it is not as common when using GLS

Global Linear Solve/GLS/RAS

- You should not see convergence rates decrease as much when running with more processors
  - You can still get unlucky with grid splitting and run into a stability issue, but it is not as common when using GLS
- You'll see more time spent in CBCXCH/CDQXCH
  - Approximately ILHSIT times more than before
  - MPI processes that are not well balanced will cause other processes to wait to receive data, so rather than just spending time idle, processes are now busy waiting for the slow process to send

Global Linear Solve/GLS/RAS

- You should not see convergence rates decrease as much when running with more processors
  - You can still get unlucky with grid splitting and run into a stability issue, but it is not as common when using GLS
- You'll see more time spent in CBCXCH/CDQXCH
  - Approximately ILHSIT times more than before
  - MPI processes that are not well balanced will cause other processes to wait to receive data, so rather than just spending time idle, processes are now busy waiting for the slow process to send
- For supersonic cases, use GLS + ILHS = 6/7/8
  - Linearization of Steger-Warming fluxes in a *FV* context + implicit BCs, etc.

General suggestions for OVERFLOW 2.3/2.4

General suggestions for OVERFLOW 2.4

- There’s no good reason not to switch to 2.4
  - Only a slight change that causes wall functions to recompute some data

General suggestions for OVERFLOW 2.4

- There’s no good reason not to switch to 2.4
  - Only a slight change that causes wall functions to recompute some data
- If you’re running steady state, use CFL ramping

General suggestions for OVERFLOW 2.4

- There’s no good reason not to switch to 2.4
  - Only a slight change that causes wall functions to recompute some data
- If you’re running steady state, use CFL ramping
- Grid sequencing is a double-edged sword
  - Implicit BC’s and coarse grids have strange interactions, but GLS helps

General suggestions for OVERFLOW 2.4

- There's no good reason not to switch to 2.4
  - Only a slight change that causes wall functions to recompute some data
- If you're running steady state, use CFL ramping
- Grid sequencing is a double-edged sword
  - Implicit BC's and coarse grids have strange interactions, but GLS helps
- If a problem is failing
  - Run with &GLOBAL => DEBUG = 2 and examine the q.time file for crazy CFL numbers, adjust &TIMACU parameter or DTPHYS as needed
  - Reduce ILHSIT to avoid oversolving the linear system
  - Try running with ILHS = 16/17/18, it will either get past the problem or fail more gracefully

General suggestions for OVERFLOW 2.4

- Always turn on GLS
  - If you have a problem that converges well for you, it may have no effect
  - If a problem has large scale unsteadiness, you might not notice a difference in the norms, but the difference is there
  - We've seen other problems that do not converge/fail run fine with GLS
  - There are cases where problems with one grid may leak over into other grids

General suggestions for OVERFLOW 2.4

- Always turn on GLS
  - If you have a problem that converges well for you, it may have no effect
  - If a problem has large scale unsteadiness, you might not notice a difference in the norms, but the difference is there
  - We've seen other problems that do not converge/fail run fine with GLS
  - There are cases where problems with one grid may leak over into other grids
- When running with GLS
  - If using SA and grid sequencing, use NQT=104
  - Have seen cases that fail with NQT=102 that run with 104 and vice versa
  - If a problem does converge worse, examine your grid system

Step: N+1

- What’s coming next?

Questions?

- Answers?