An Introduction to Linear Solvers in OpenFOAM

Gregor Olenik (gregor.olenik@tum.de)

Chair of Computational Mathematics,

TUM School of Computation

Technical University of Munich

https://exasim-project.com/

Motivation

- Mainly concerned about system/fvSolution
- Build an understanding of different solver options and settings
- Ideally help to improve computational cost and speed up
- Encourage to measure and experiment with solvers

Won’t cover:

- In about one hour can’t present everything in great detail
- GPU solver, domain decomposition, workflows

Content

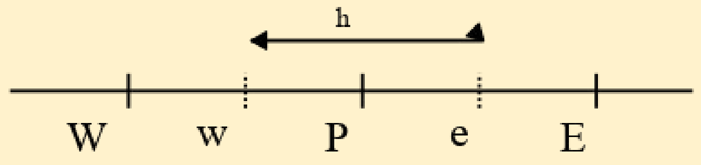
- Motivation
- Start with some theory and show where to select things in OpenFOAM
- Fundamentals Iterative Methods
- Preconditioners
- Multigrid Methods

Fundamentals

- Good introduction in Versteeg and Malalasekera:
  - Transport Eqn. (PDE) ⇨ Computational Grid ⇨ Set of linear equations of the Form Ax=b

$$\frac{d}{dx}(\rho u \phi) = \frac{d}{dy}\left(\Gamma \frac{d\phi}{dy}\right)$$

PDE



Computational Grid

$$a_P\phi_P - a_W\phi_W - a_E\phi_E = \mathrm{b}$$
$$⇨ \mathrm{Ax} = \mathrm{b}$$

Linear System

- **How to solve Ax=b for x ?**

Gaussian Elimination I

$$\begin{aligned} \boxed{2x} + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3 \end{aligned} \qquad \begin{aligned} &R_1 \\ &R_2 + 3/2\,R_1 \rightarrow R_2^* \\ &R_3 + R1 \rightarrow R_3^* \end{aligned}$$

$$\Downarrow$$

$$\begin{aligned} 2x + y - z &= 8 \\ \boxed{1/2\,y} + 1/2\,z &= 1 \\ 2y + z &= 5 \end{aligned}$$

$$\Downarrow$$

$$\begin{aligned} 2x + y - z &= 8 \\ 1/2\,y + 1/2\,z &= 1 \\ -z &= 1 \end{aligned}$$

$$\Rightarrow$$

$$\begin{aligned} 2x + y + z &= 8 \\ 1/2\,y + 1/2\,z &= 1 \\ \boxed{-z} &= 1 \end{aligned}$$

$$\Uparrow$$

$$\begin{aligned} 2x + y &= 7 \\ \boxed{1/2\,y} &= 3 \\ z &= -1 \end{aligned}$$

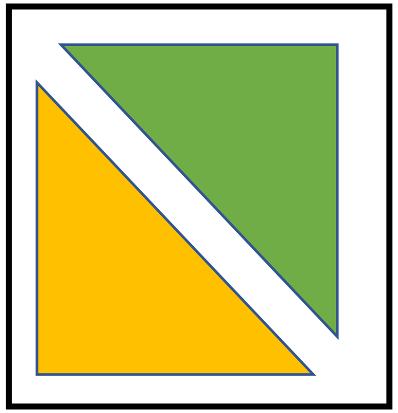
$$\Uparrow$$

$$\begin{aligned} x &= 2 \\ y &= 3 \\ z &= -1 \end{aligned}$$

Remarks:

- Usually performed via **LU decomposition**
- In the general case, solving Ax=b of size n by Gaussian elimination requires 2n^3/3 + O(n^2) operations. And requires n^2 + n storage.
- Using GE/LU has some drawbacks for sparse cases.
- Data dependencies -> limited parallelization potential
- Works for poorly conditioned cases

Gaussian Elimination II



- $\mathrm{A = LU} \Rightarrow \mathrm{LUx = b}$ with $\mathrm{Ux = y}$
- Solve:
    - 1. $\mathrm{Ly = b}$ for y
    - 2. $\mathrm{Ux = y}$ for x
- Obtain L and U via LU decomposition

$$\left(\begin{array}{c|c} a & \mathbf{w}^{\mathsf{T}} \\ \hline \mathbf{v} & A' \end{array}\right) = \left(\begin{array}{c|c} 1 & \mathbf{0}^{\mathsf{T}} \\ \hline \mathbf{l} & L^{(1)} \end{array}\right) \left(\begin{array}{c|c} u & \mathbf{u}^{\mathsf{T}} \\ \hline \mathbf{0} & U^{(1)} \end{array}\right) = \left(\begin{array}{c|c} u & \mathbf{u}^{\mathsf{T}} \\ \hline u\mathbf{l} & \mathbf{l}\mathbf{u}^{\mathsf{T}} + L^{(1)}U^{(1)} \end{array}\right)$$

note: $\mathbf{l} = \frac{1}{u}\mathbf{v}$

repeat for $lu^T + L^1U^1 = \mathrm{A}' - \mathrm{lu}^T$

- Lets check $\mathrm{L^1U^1 = A' - lu}^T$
- lu^T is an outer product
- A' and lu^T have non-zeros at different locations ⇨ number of non-zeros (nnz) per row decreases, **fill-in**

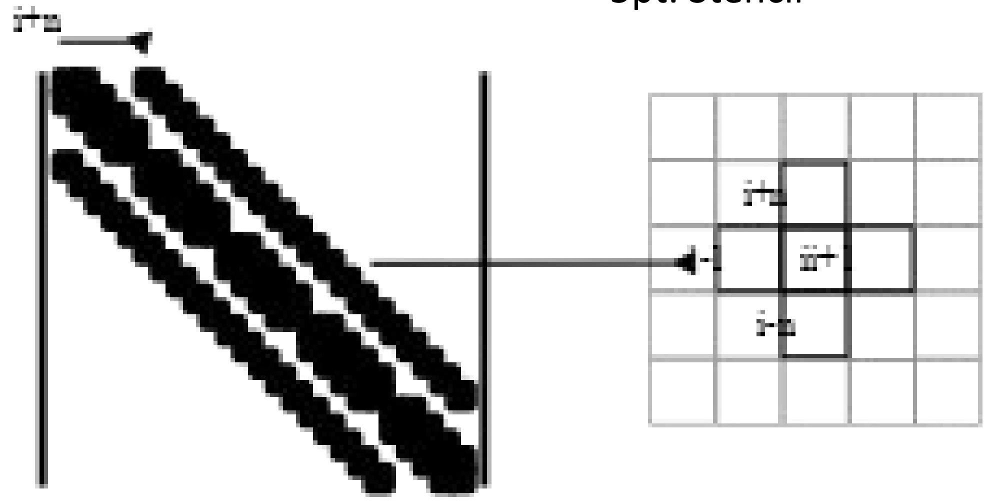
Example: $\begin{bmatrix}1\\0\\1\end{bmatrix} \times [\,1 \quad 0 \quad 1] = \begin{bmatrix}1 & 0 & 1\\0 & 0 & 0\\1 & 0 & 1\end{bmatrix}$

Sparsity

- First example all rows had entries (dense)
- For simulations matrix entries depend on computational grid/stencil.
- Storing only non-zero matrix entries can be beneficial. Assume a 2D grid with 1M cells -> 1Mx1M matrix, double precision,
    - dense: **9TByte** (n_rows**2 * sizeof(double))
    - COO: (values + row and column index 32bit int) **40MByte** (n_rows*nnz_row*sizeof(double) + 2n_rows*nnz_row*sizeof(int))
- Gauss elimination on sparse matrices will have larger storage requirement compared to original system. Makes GE impractical for larger problems, even for sparse matrix formats.

Sparsity Pattern 25x25 Matrix

5x5 Computational Grid
5pt. Stencil



Iterative Methods

Task:

- Find solution (x) to $Ax = b$
- Constraint: Problem: finding A^{-1} is expensive (Gaussian Elimination) and has huge storage requirements
- Idea: repeatedly improve a guessed solution x
  - Ingredients:
    - Error: $\mathbf{e} = \mathbf{x}^* - \mathbf{x}$
    - Norm: u $= ||\mathbf{u}|| = \sqrt{\boldsymbol{u} \cdot \boldsymbol{u}}$ ⇨ $||\mathbf{u}||^2 = \boldsymbol{u} \cdot \boldsymbol{u}$ needed for stopping criterion
    - Residual: $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$ if **e = 0** ⇨ **r=0** and **||r|| = 0**

Iterative Methods

- Principle methods

Direct	$x = A^{-1}b$	- Storage
Richardson	$x^{m+1} = x^m + \omega(b - Ax^m)$	+ Storage nnz + 2*n_Dofs
Jacobi	$x^{m+1} = D^{-1}(b - L + U)\,x^m$	+ Parallel
Gauss-Seidel	$L_* x^{m+1} = b - Ux^m$	+ Convergence, - Sequential

Jacobi vs. Gauss-Seidel vs. Richardson

$$c_{11}x_1 + c_{12}x_2 + c_{13}\,x_3 = b_1$$
$$c_{21}x_1 + c_{22}x_2 + c_{23}x_3 = b_2$$
$$c_{31}x_1 + c_{32}x_2 + c_{33}x_3 = b_3$$

Jacobi

solve for x_n repeatedly
wo updates

$$x_1^{m+1} = (b_1 - (c_{12}x_2^m + c_{13}x_3^m))/c_{11}$$
$$x_2^{m+1} = (b_2 - (c_{21}x_1^m + c_{23}x_3^m))/c_{22}$$
$$x_3^{m+1} = (b_3 - (c_{31}x_1^m + c_{32}x_2^m))/c_{33}$$

+ Parallel

Gauss-Seidel

solve for x_n repeatedly
w updates

$$x_1^{m+1} = (b_1 - (c_{12}x_2^m + c_{13}x_3^m))/c_{11}$$
$$x_2^{m+1} = (b_2 - (c_{21}x_1^{m+1} + c_{23}x_3^m))/c_{22}$$
$$x_3^{m+1} = (b_3 - (c_{31}x_1^{m+1} + c_{32}x_2^{m+1}))/c_{33}$$

+ Better convergence
- Sequential

Richardson

subtract residual

$$x_1^{m+1} = x^m - \omega\left(b_1 - c_{11}x_1^m - c_{12}x_2^m - c_{13}x_3^m\right)$$
$$x_2^{m+1} = x^m - \omega\left(b_2 - c_{21}x_1^m - c_{22}x_2^m - c_{23}x_3^m\right)$$
$$x_2^{m+1} = x^m - \omega\left(b_3 - c_{31}x_1^m - c_{32}x_2^m - c_{33}x_3^m\right)$$

+ Parallel

Richardson Iteration vs. Gaussian Elimination

Which does more work, for a typical CFD matrix of size nxn

Richardson

- $x^{m+1} = x^m + \ \omega\,(b - Ax^m)$
- 1SpMV, 2 VU, 1 SVP
- Assume 7pt stencil
- SpMV: $2n\ n_{NZR}$
- $n_{iter}\,(14\,n + 2n + n) = 17nn_{iter}$

+ VU, SVP Embarrassingly parallel

+ SpMV is parallelizable

Gaussian Elimination

LU Decomposition

- Only 3 Bands to eliminate
- Multiplication of pivot row (4 + Fill(row))
- Subtraction of pivot row 2
- 2 Fill ins per subtraction

Back Substitution

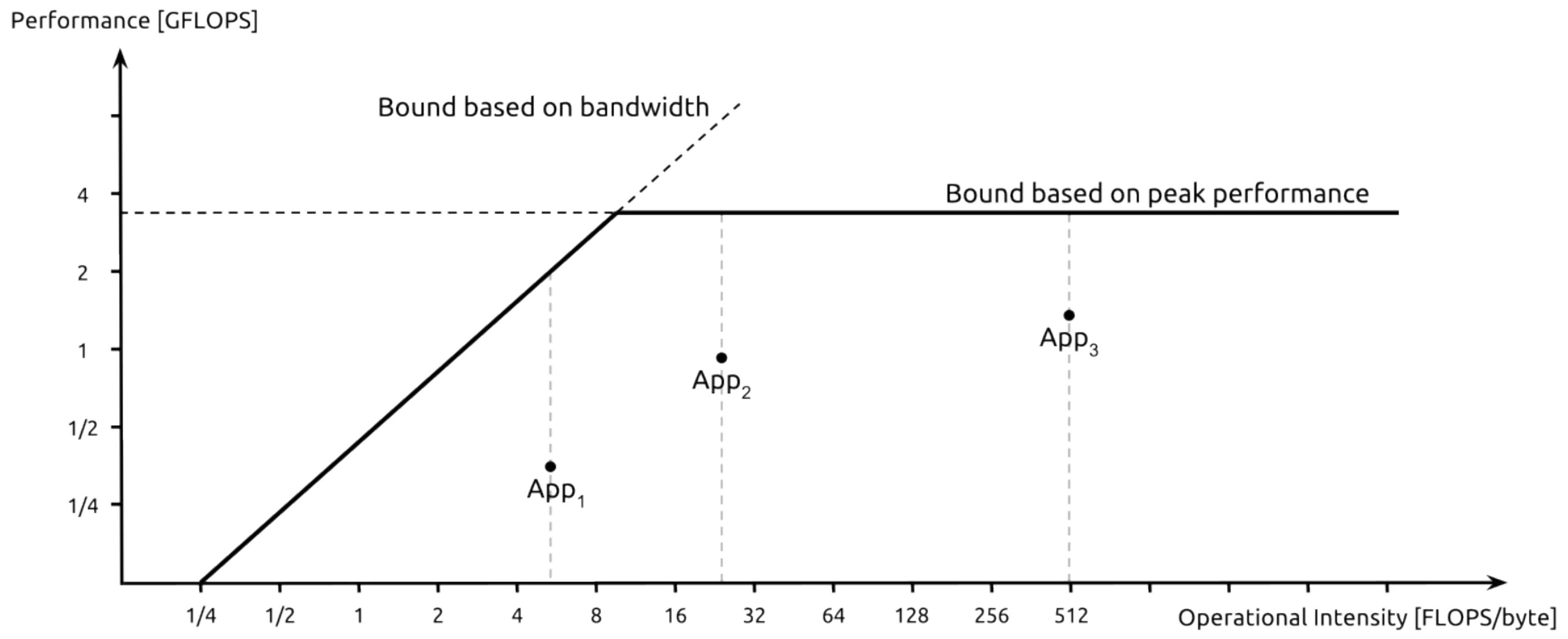
- 3*n*8*2 + Fill

Total = $3n4 + Fill\ + 42n\ = 56n$ + Fill

- Sequential/Hard to parallelize

Interlude Roofline Model

- Modern machines can perform floating point operations much faster then moving data into registers
- Arithmetic Intensity = Computational Work [FLOPS] / Amount of memory moved [Bytes]



Krylov Methods

- How to improve basic iterative methods?
- Richardson: $x_{m+1} = x_m + \omega(b - Ax_m) = x_m + \alpha p_m$
- Motivation for Krylov methods find optimal α and search direction **p**
- NB: with $x_0 = 0 \rightarrow x_1 = r_0,\ x_2 = 2r_0 - Ar_0$ … thus $x_k \in span\{r_0, Ar_0 \ldots A^{k-1} r_0\}$ which is called Krylov

  (sub)space
- For x_{m+1}find α such that error $||x - x_{m+1}||^2{}_{\mathrm{A}}$ is minimal, where $||a||_2 = \sqrt{}(a^T a)$ and $||a||_A = \sqrt{(a^T Aa)}$
- **A needs to be SPD to satisfy the A norm**
- $\frac{d}{d\alpha} ||x - x_{m+1}||^2{}_{\mathrm{A}} = 0 \rightarrow \frac{r_k^T r_k}{r_k^T A r_k}$
- Pick next search direction from Krylov space such that next search direction is orthogonal to all previous search directions

Conjugate Gradient Method (CG)

```
$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
if $\mathbf{r}_0$ is sufficiently small, then return $\mathbf{x}_0$ as the result
$\mathbf{p}_0 := \mathbf{r}_0$
$k := 0$
repeat
    $\alpha_k := \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$
    $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$
    $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$
    if $\mathbf{r}_{k+1}$ is sufficiently small, then exit loop
    $\beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}$
    $\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$
    $k := k + 1$
end repeat
return $\mathbf{x}_{k+1}$ as the result
```

- See literature for full derivation, eg.: A. Meister, Y. Saad or W. Hackbusch

// system/fvSolution

```
p
{
  solver  PCG;                 // 1
  preconditioner  none;    // 2
  tolerance        0.0001; // 3
  relTol  0;                   // 4
  maxIter 5000;                // 5
  minIter 0;                   // 6
}
pFinal                  // 4
{
$p;
tolerance  1e-06;    // 3
relTol     0;        // 4
}
```

1. Selects Preconditidioned CG (PCG):
   - `other options GAMG PBiCGStab PCG PPCG PPCR smoothSolver` (via banana method):
   - PCG typically a very simple and robust solver for poisson type equations, scales up to 10k per Core
   - GAMG typically faster but might experience worse scaling or issues when
2. Selects Preconditioner:
   - options `none DIC FDIC GAMG diagonal`
   - will talk about preconditioners in a moment
3. Absolute value of residual norm when to stop iteration process.
   - Be careful this is not ||r||_2 = ||Ax-b||_2
   - typical values for pressure 1e-4 ... 1e-6. If grid density increases tolerance might need to be adapted
4. Stop when a given ratio is reached $\alpha = r_0/r_i$
   - typical use case, outer iterations for projectio methods, predictor corrector methods like PIMPLE. Intermediate iterations might not need fully converged solution.
   - Ignored if 0, at least in pFinal it should be 0
5. Stop at n iterations (default 1000), to avoid dead lock. Be careful 1000 GAMG iterations are much more expensive.
   - treat solver as not converged if maxIter is reached
6. perform at least minIter iterations.
   - sometimes used to fix number of iterations for benchmarks minIter=maxIter
   - be careful in production (not recommended IMHO)

Preconditioner

- CG might fail to converge if condition number is large.
- Convergence of behavior CG:

$$\|e\|_A \leq 2\left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^k \|e_0\|_A$$

- Idea: Solve equivalent system with lower condition number

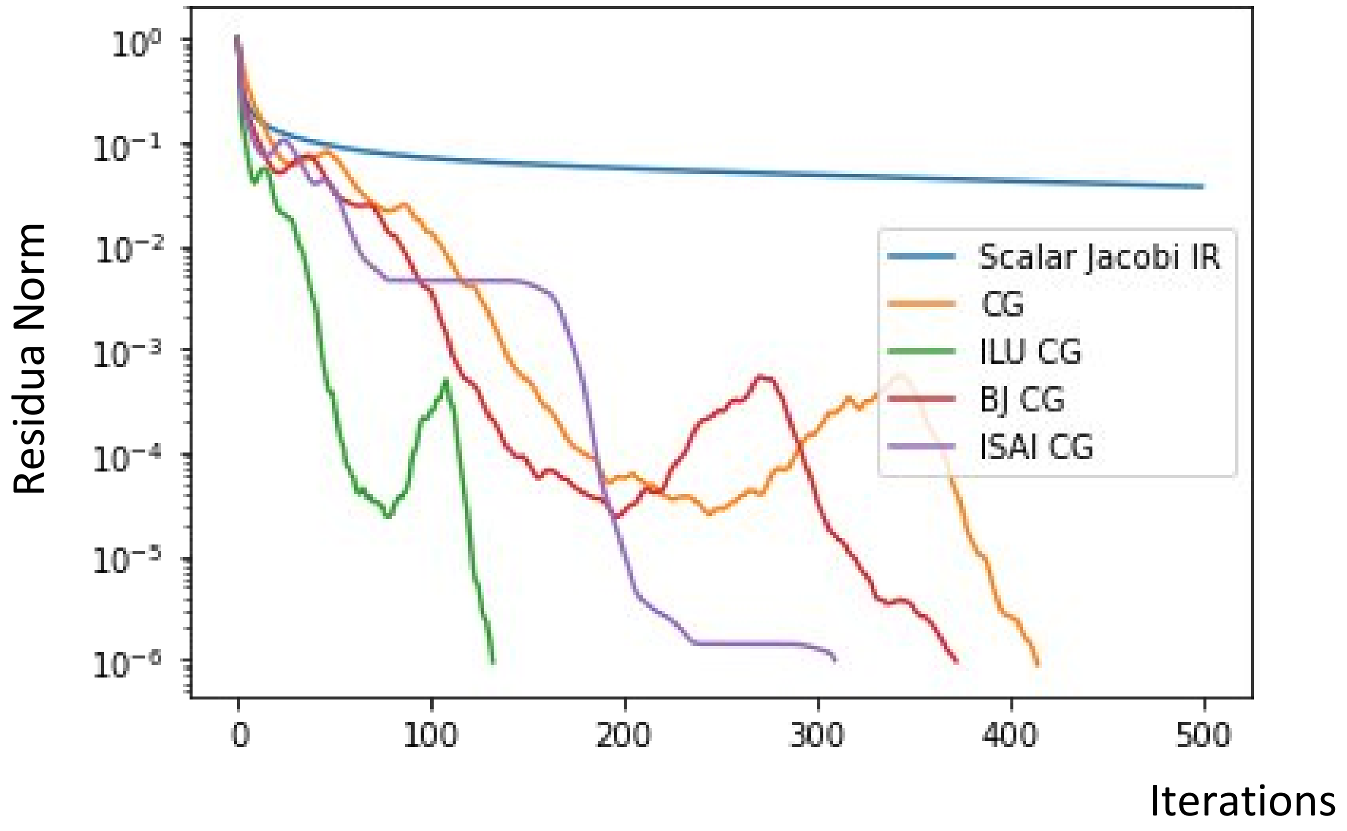
$$M^{-1}Ax = M^{-1}b$$

- easy to see that M=A is best preconditioner, but requires inverse of A. In PCG, however, only the residual vector is preconditioned $z = M^{-1}r$ or $Mz = r$
- Extra effort to generate M^{-1}
- Again, see literature for more details

Some (black box) preconditioners:

- **Diagonal** (cheap but usually not very effective): $M = D, D^{-1} = d_{ii}^{-1}$
- **Incomplete LU (ILU or in OF DILU)** do LU decomposition of A but keep only values which where previously at a non-zero location (no fill in). Works for non-symmetric A
- **Incomplete Cholesky (IC or in OF DIC)** for symmetric A: $M = LL^T \rightarrow LL^T z = r$
  - Factorize A into LL^T using modified Gaussian Elimination, but discard any fill-in
  - 1. Solve Ly = r for y 2. Solve $L^T z = y$ Simple to solve by backward substitution

Convergence Rates



Some observations:

- residual norm does not behave monotonically, even though error decreases monotonically. We minimize error but use norm as stopping criterion
- Preconditioner can improve convergence, but many factors influence behavior;
  - ordering, decomposition
- Note that less iterations does not necessarily mean less time.

Non symmetric variants

- CG only works for symmetric matrices (remember requirement for A-norm). Momentum, scalar transport equation don't yield symmetric linear system.
- Linear solver in OF for non-symmetric cases: GAMG, PBiCG, PBiCGStab, smoothSolver
- Idea of BiCG methods:
  - Solve a equivalent symmetric adjoint system $\begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \hat{x} \end{bmatrix} = \begin{bmatrix} \hat{b} \\ b \end{bmatrix}$
  - More work compared CG and some other challenges
- smoothSolver just a wrapper around:

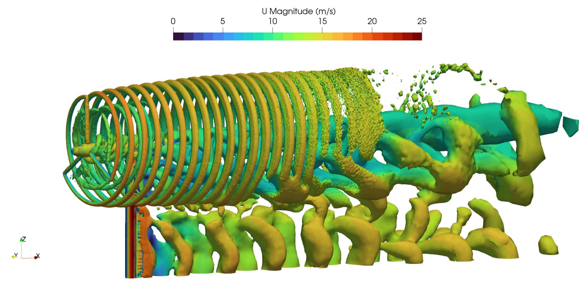
```
DILU DILUGaussSeidel GaussSeidel nonBlockingGaussSeidel symGaussSeidel
```

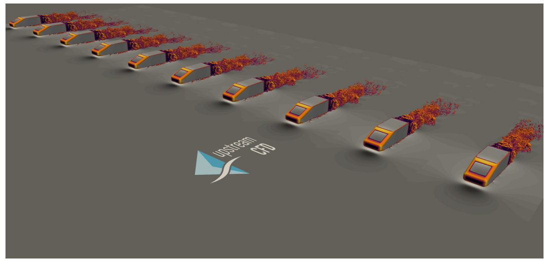
- GPU capable solver https://github.com/hpsim/OGL https://doi.org/10.1007/s11012-024-01806-1

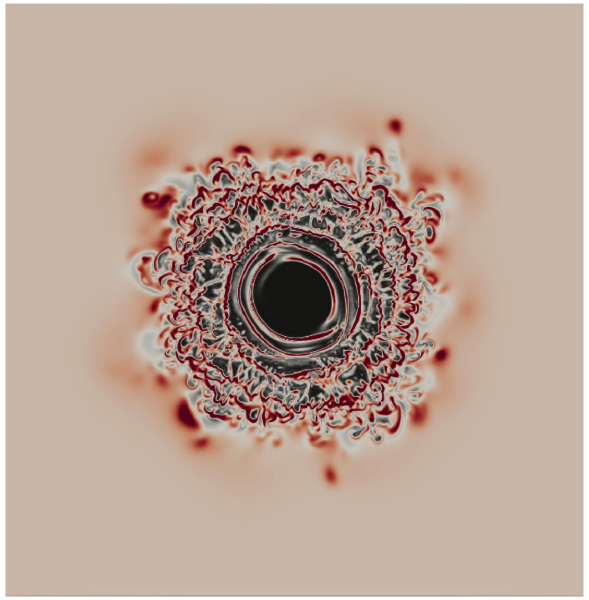
```
U
{
 solver           PBiCGStab;
 preconditioner   DILU;
 tolerance        1e-08;
 relTol           0.1;
 maxIter          1000;
}
```

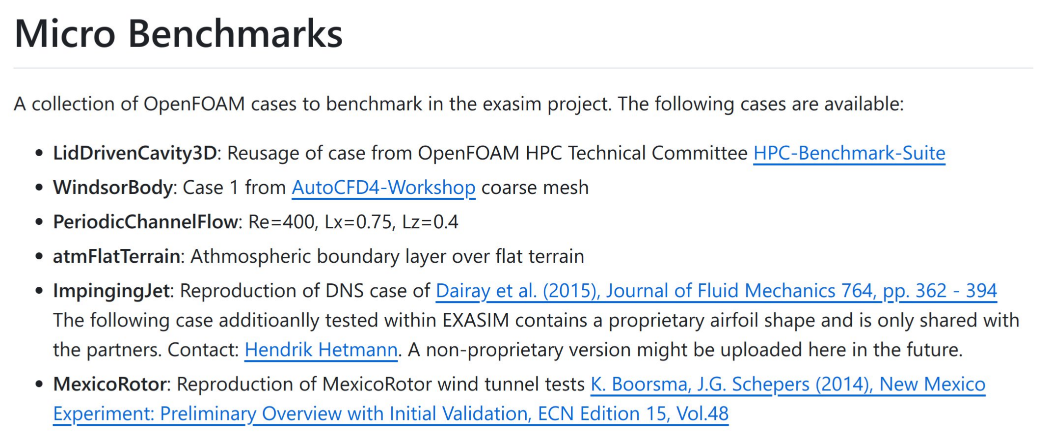
```
U
{
  solver       smoothSolver;
  smoother     GaussSeidel;
  tolerance    1e-8;
  relTol       0.1;
}
```

```
p
{
  solver      GAMG;
  smoother    GaussSeidel;
  tolerance   1e-7;
  relTol      0.01;
}
```









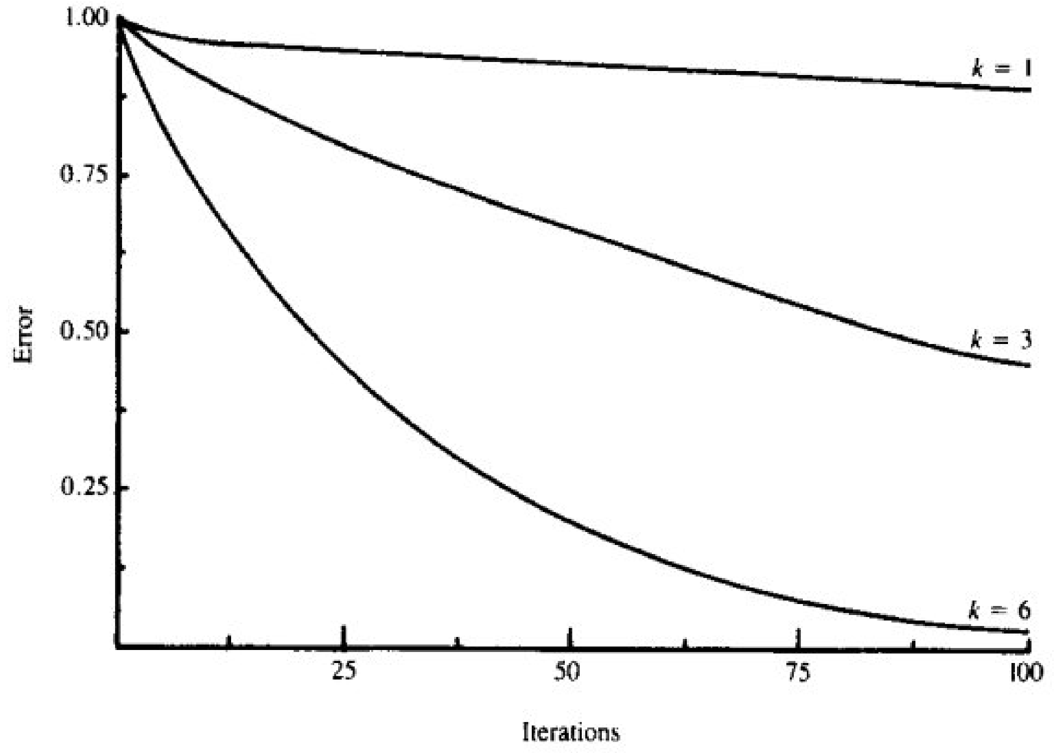
Micro Benchmarks

A collection of OpenFOAM cases to benchmark in the exasim project. The following cases are available:

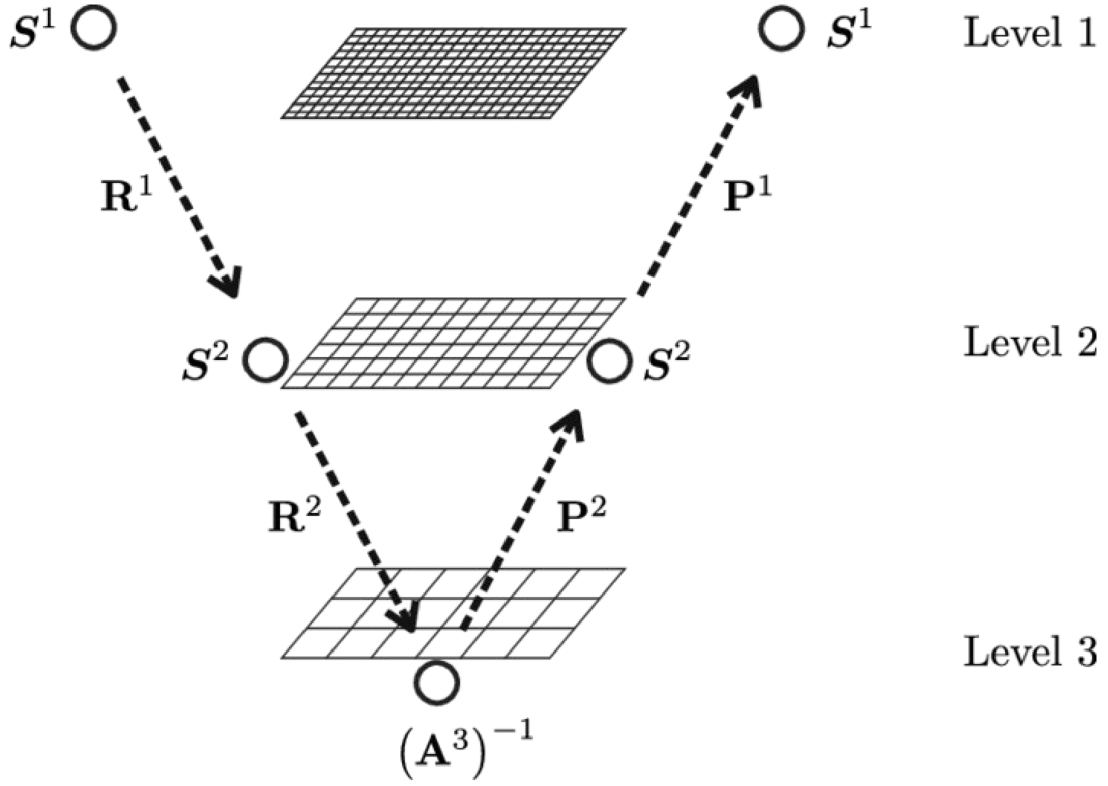
- **LidDrivenCavity3D**: Reusage of case from OpenFOAM HPC Technical Committee HPC-Benchmark-Suite
- **WindsorBody**: Case 1 from AutoCFD4-Workshop coarse mesh
- **PeriodicChannelFlow**: Re=400, Lx=0.75, Lz=0.4
- **atmFlatTerrain**: Athmospheric boundary layer over flat terrain
- **ImpingingJet**: Reproduction of DNS case of Dairay et al. (2015), Journal of Fluid Mechanics 764, pp. 362 - 394
  The following case additioanlly tested within EXASIM contains a proprietary airfoil shape and is only shared with the partners. Contact: Hendrik Hetmann. A non-proprietary version might be uploaded here in the future.
- **MexicoRotor**: Reproduction of MexicoRotor wind tunnel tests K. Boorsma, J.G. Schepers (2014), New Mexico Experiment: Preliminary Overview with Initial Validation, ECN Edition 15, Vol.48

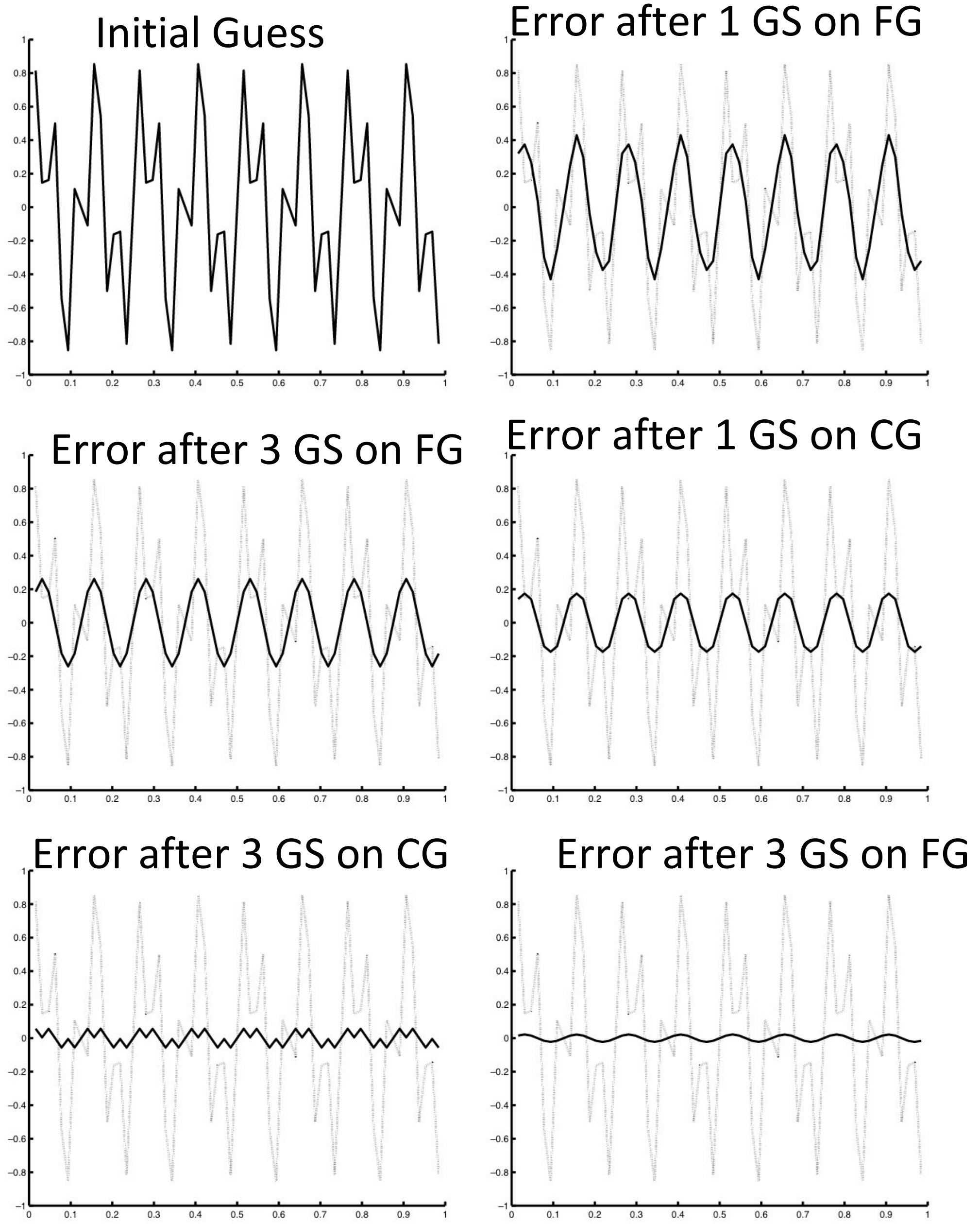
On github: exasim-project/micro-benchmarks

Multigrid I



Error components for Jacobi Iteration
Source: Briggs et al. 2000



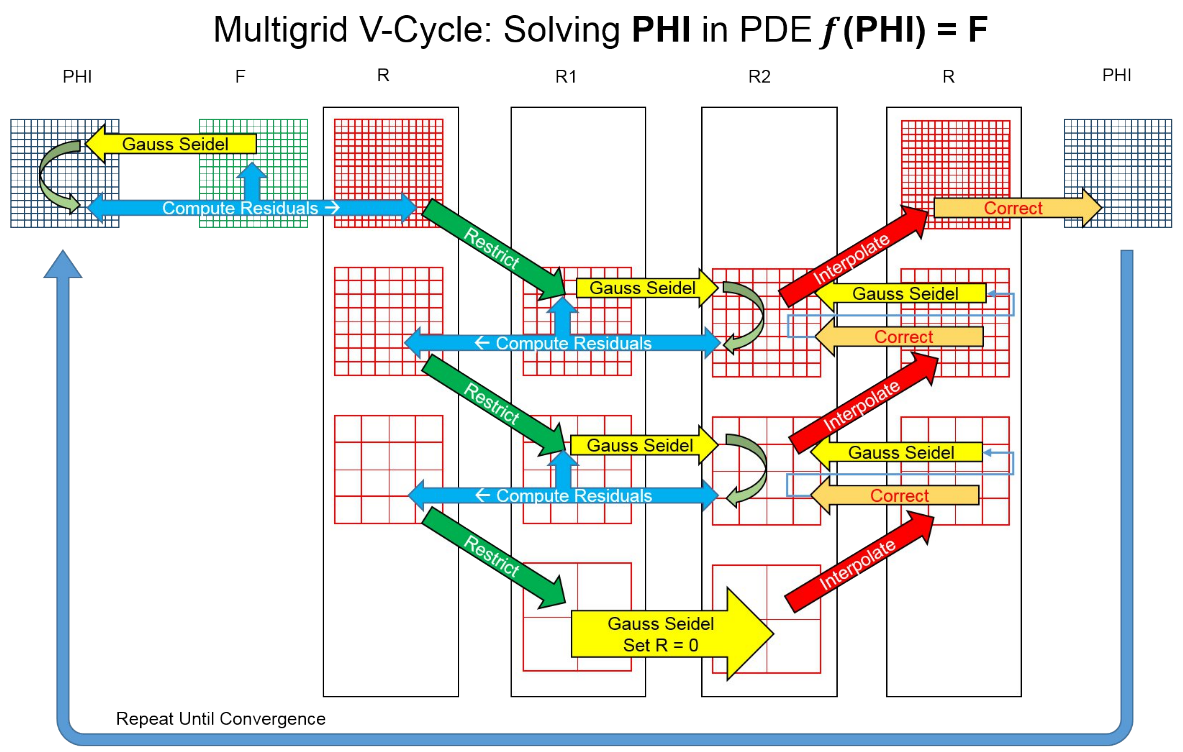


Source: Briggs et al. 2000

- Error always wrt. fine grid.
- GS on FG becomes less effective after few iterations
- Switch to CG (Restriction) for further iterations
- Projection (Prolongation) of CG solution on FG has spurious oscilations
- Perform post smoothing on FG.

Multigrid II

- Many flavors of Multigrid:
  - Geometric Multigrid (GAMG) if geometric data for restriction and prolongation is available
  - Algebraic Multigrid (AMG) otherwise
- Various cycles V, W
- All kinds of smoothers are possible Gauss-Seidel, pre, post-smoothing, or both



Multigrid Options

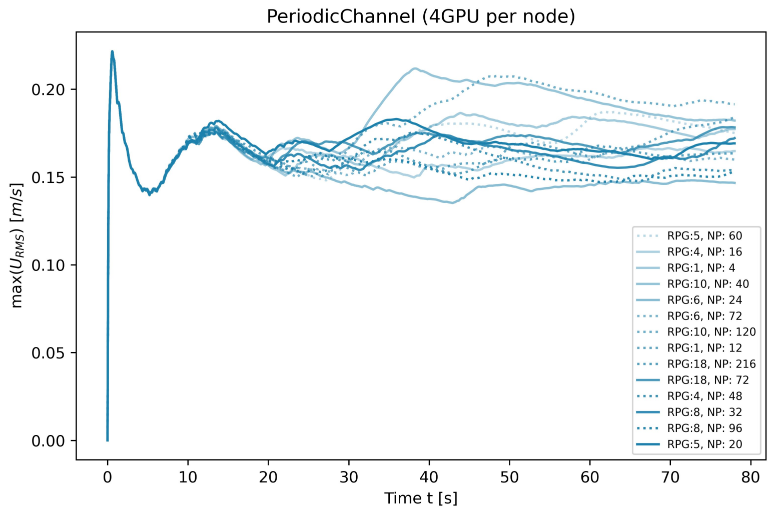
```
// fvSolution
p
{
    solver          GAMG;
    tolerance       1e-06;
    relTol          0.1;
    smoother        GaussSeidel;
// alternative smoother
// DIC DICGaussSeidel FDIC
GaussSeidel nonBlockingGaussSeidel
symGaussSeidel
}
```

```
// GAMGSolver.C

cacheAgglomeration_(true),
nPreSweeps_(0),
preSweepsLevelMultiplier_(1),
maxPreSweeps_(4),
nPostSweeps_(2),
postSweepsLevelMultiplier_(1),
maxPostSweeps_(4),
nFinestSweeps_(2),
interpolateCorrection_(false),
scaleCorrection_(matrix.symmetric
()),
directSolveCoarsest_(false),
```

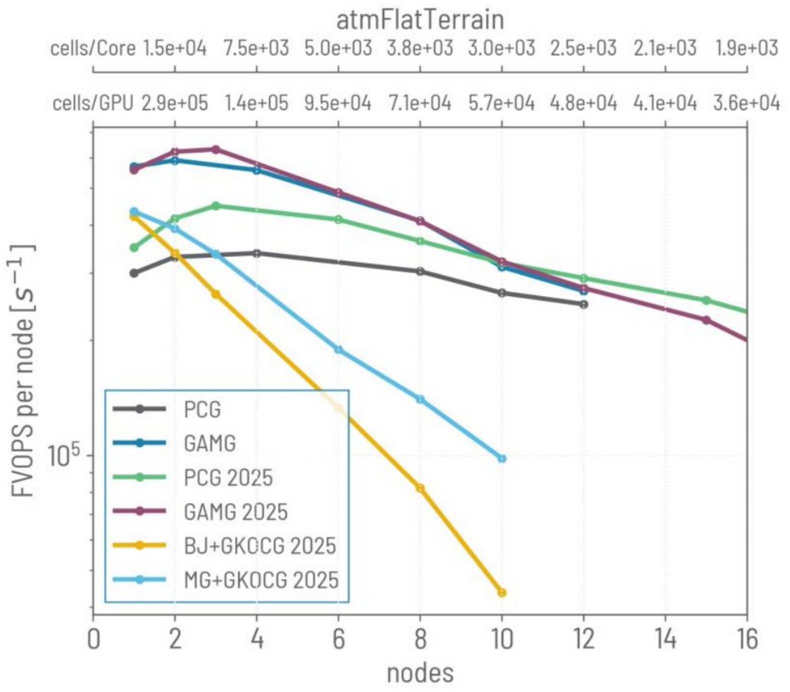
How to choose solver

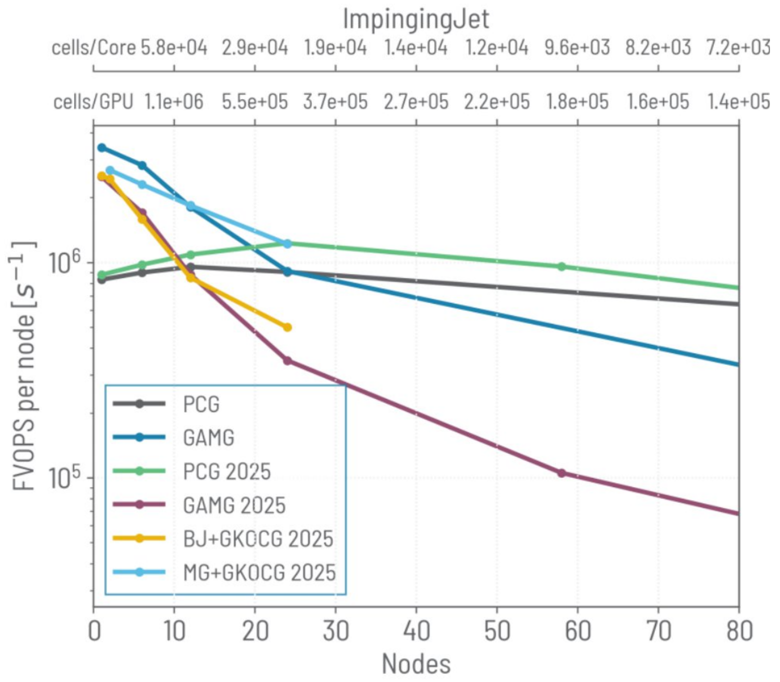
- No general answer, measure performance (timings) for your problem
- (Don’t be afraid of trying out solver, in the worst case fail to converge)
- Check also whether results are independent of tolerances, sample enough data for statistics

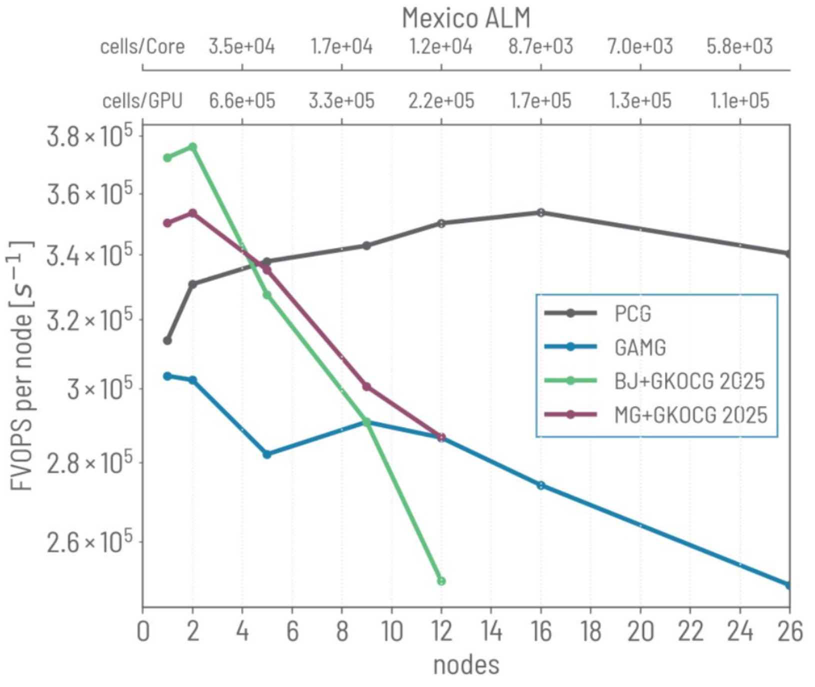


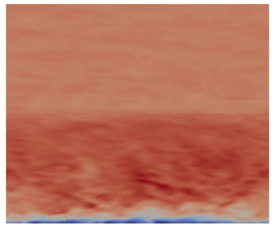
- GAMG for pressure, smoothSolver for Momentum (transient). For larger cases/many subdomains CG might be better. BiCGStab non-symmetric, steady-state.

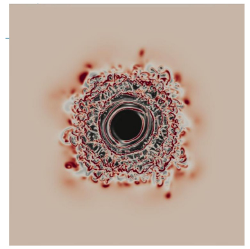
Some Benchmark results

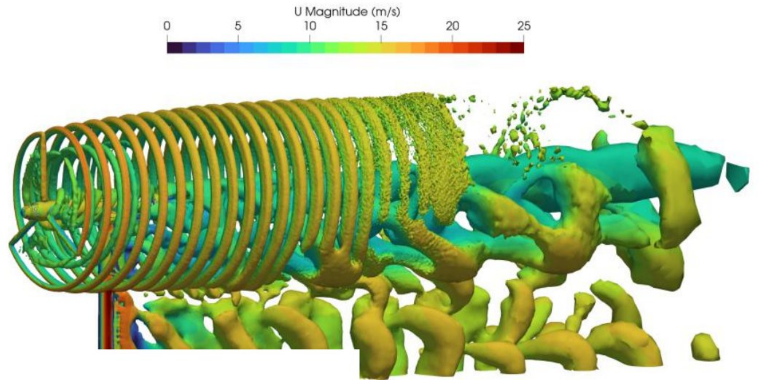












- Typically GAMG better performance on fewer nodes
- Scaling study required

Performance tweaks

- Renumber mesh can improve cache access on unstructured meshes
- Good meshes can have a huge impact on performance
- 10000-30000 cells per core are a reasonable number for domain decomposition (cache), see Galeazzo et. al
- Check DD for load balancing
- For large cases with many subdomains communication can become the bottleneck, switch from GAMG -> PPCG
- Don’t assume mpirun –bind-to core

Literature

- Meister, Andreas. *Numerik Linearer Gleichungssysteme*. Vol. 4. Wiesbaden: Vieweg+ Teubner, 2011.
- Saad, Yousef. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2003.
- Hackbusch, Wolfgang. *Iterative solution of large sparse systems of equations*. Vol. 95. New York: Springer, 1994.
- Galeazzo, Flavio Cesar Cunha, et al. "Understanding superlinear speedup in current HPC architectures." *IOP Conference Series: Materials Science and Engineering*. Vol. 1312. No. 1. IOP Publishing, 2024.
- Briggs, William L., Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial*. Society for Industrial and Applied Mathematics, 2000.
- Olenik, Gregor, et al. "Towards a platform-portable linear algebra backend for OpenFOAM." *Meccanica* (2024): 1-14.
- Gärtner, Jan Wilhelm, et al. "Testing Strategies for OpenFOAM Projects." *OpenFOAM® Journal* 5 (2025): 115-130.