POLYTECHNIQUE MONTRÉAL
TECHNOLOGICAL UNIVERSITY

laboratoire CHAOS laboratory

Breaking free from the matrix: A stabilized Navier-Stokes perspective using Lethe

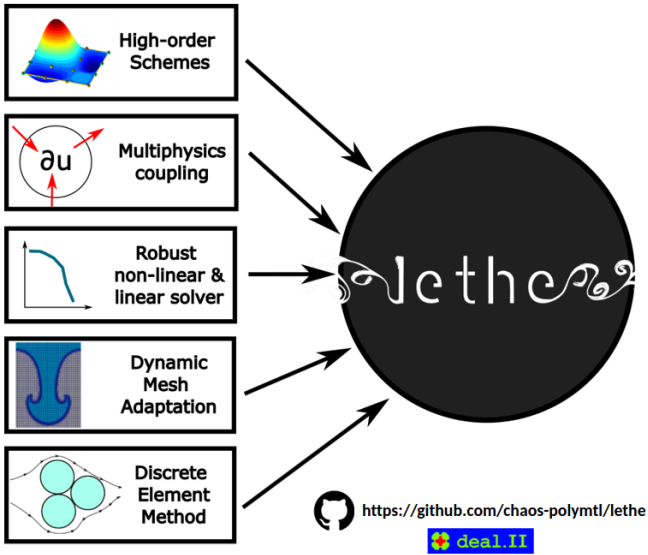
deal.II workshop

Laura Prieto Saavedra[1] Peter Munch[2] Bruno Blais[1]

[1]Polytechnique Montréal, laura.prieto-saavedra@polymtl.ca, bruno.blais@polymtl.ca
[2]Uppsala University, peter.munch@it.uu.se

Fort Collins, Tuesday 13th August, 2024

Open-source software: Lethe

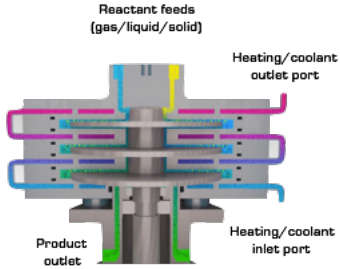


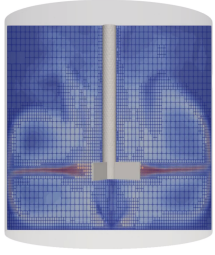
- Based on the deal.II library (Arndt et al., 2023)
- Created by Blais et al. (2020)
- 35 contributors
- User guide ($>$ 50 examples)
- Tests ($>$ 350 tests)

Context and motivation: process-intensified equipment

- Simulation of complex flow problems in chemical engineering is still a challenge: **rotating flows, transient flows** $\rightarrow \mathrm{Re} = [10^3, 10^5]$.
- Localized physical phenomena require very fine meshes $\rightarrow$ alternative: **locally refined meshes**.
- The algorithms for these very large simulations are memory bandwidth bounded, which led us to implement a **matrix-free solver**.
- Several challenges arise, among them: the development of a **matrix-free linear solver.**

In this presentation: overview of the matrix-free solver and the geometric multigrid preconditioner





Outline

1. Governing equations and discretization: incompressible Navier-Stokes equations
2. Implementation: from a matrix-based to a matrix-free approach
3. Linear solver: geometric multigrid preconditioner
4. Numerical tests
5. Outlook: what comes next?

1. Governing equations and discretization

Incompressible Navier-Stokes:

$$\nabla \cdot \boldsymbol{u} = 0 \text{ in } (0, T] \times \Omega$$

$$\partial_t \boldsymbol{u} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \frac{1}{\rho}\nabla p - \nabla \cdot \mathbb{D}(\boldsymbol{u}) = \boldsymbol{f} \text{ in } (0, T] \times \Omega$$

- Non-linear coupled PDE system that needs to be solved for a vector-valued velocity $\boldsymbol{u}$ and a scalar pressure p.
- Continuous Galerkin method.
- Tensor elements: quadrilateral and hexahedra.
- Newton's method for the non-linearity.
- For steady-state problems: $\partial_t \boldsymbol{u} = 0$.

1. Governing equations and discretization (Cont.)

Weak form:

$$F(\boldsymbol{u}, p) := (q, \nabla \cdot \boldsymbol{u})_\Omega + (\boldsymbol{v}, \partial_t \boldsymbol{u})_\Omega + (\boldsymbol{v}, (\boldsymbol{u} \cdot \nabla)\boldsymbol{u})_\Omega - (\nabla \cdot \boldsymbol{v}, p)_\Omega$$
$$+ \nu(\nabla \boldsymbol{v}, \nabla \boldsymbol{u})_\Omega + (\boldsymbol{n} \cdot \boldsymbol{v}, p)_{\partial\Omega} - (\boldsymbol{v}\boldsymbol{n}, \nu \nabla \boldsymbol{u})_{\partial\Omega} - (\boldsymbol{v}, \boldsymbol{f})_\Omega = 0$$

Linear system for each Newton iteration:

$$\underbrace{\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{B}^T & \boldsymbol{0} \end{bmatrix}}_{\boldsymbol{F}'} \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{P} \end{bmatrix} = - \underbrace{\begin{bmatrix} \boldsymbol{R}_u \\ \boldsymbol{R}_p \end{bmatrix}}_{\boldsymbol{R}}$$

This discretization has three limitations:

- Ladyzhenskaya–Babuška–Brezzi condition.
- Instabilities when Re increases.
- Saddle-point problem.

→ **we overcome these by using stabilization techniques**

1. Governing equations and discretization (Cont.)

$$F + \underbrace{\sum_k (\tau \nabla \cdot q, \mathrm{R_m})_{\Omega_k}}_{\text{PSPG term}} + \underbrace{\sum_k (\tau (\boldsymbol{u} \cdot \nabla)\boldsymbol{v}, \mathrm{R_m})_{\Omega_k}}_{\text{SUPG term}} = 0 \quad \mathrm{R_m} : \text{momentum strong residual}$$

$$\tau = \left[\left(\frac{1}{\Delta t} \right)^2 + \left(\frac{2\|\boldsymbol{u}\|}{h} \right)^2 + 9 \left(\frac{4\nu}{h^2} \right)^2 \right]^{-1/2} \qquad h = \left(\frac{6 h_k}{\pi} \right)^{\frac{1}{3}}$$

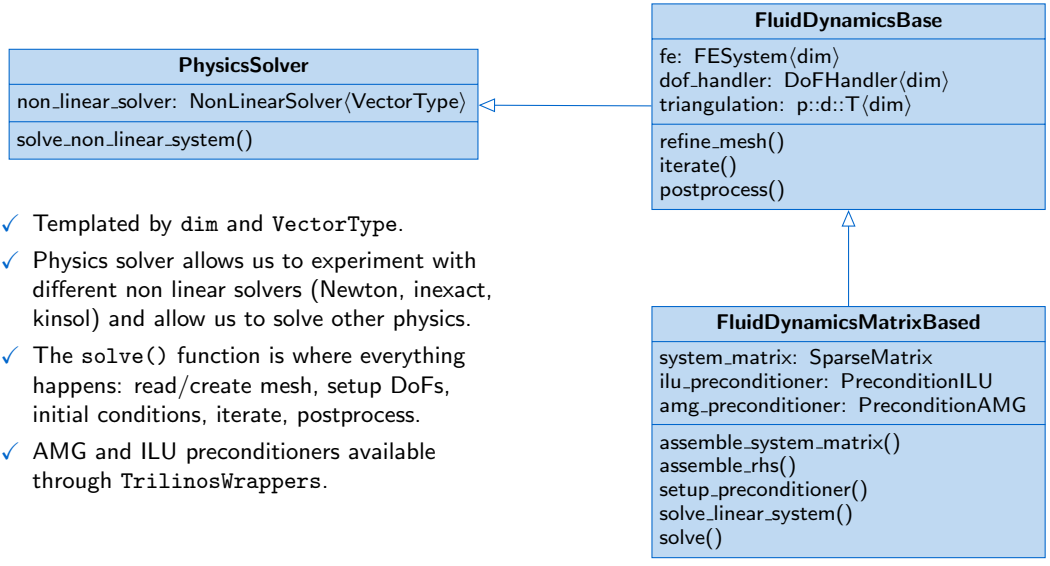
Modified linear system:

$$\underbrace{\begin{bmatrix} \widehat{\boldsymbol{A}} & \widehat{\boldsymbol{B}} \\ \boldsymbol{C} & \boldsymbol{D} \end{bmatrix}}_{\boldsymbol{F'}} \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{P} \end{bmatrix} = - \underbrace{\begin{bmatrix} \widehat{\boldsymbol{R}}_u \\ \widehat{\boldsymbol{R}}_p \end{bmatrix}}_{\boldsymbol{R}}$$

- We solve this problem in a fully-coupled monolithic way.
- At each nonlinear iteration, we solve the system using a preconditioned GMRES method.

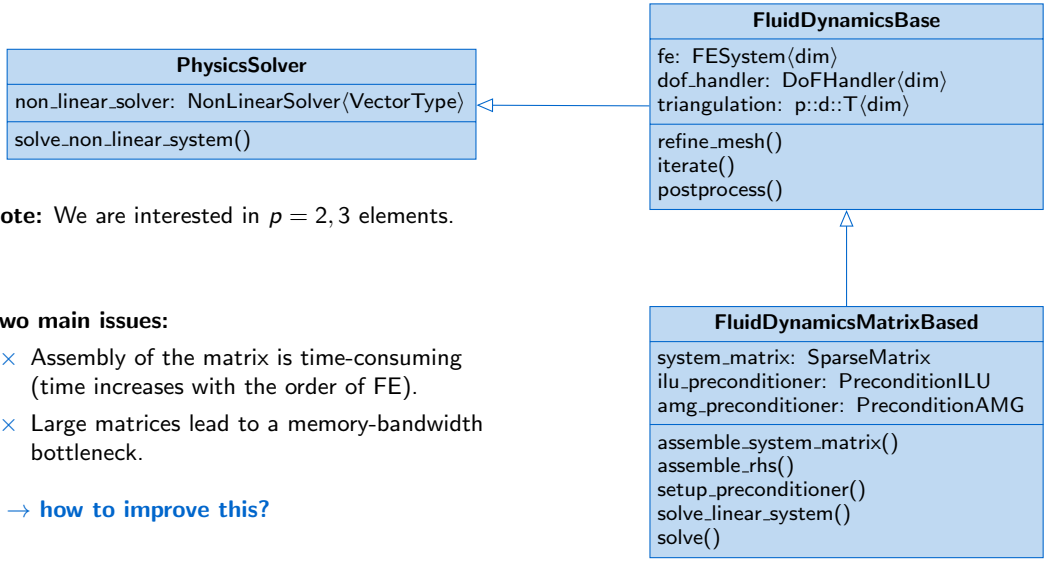
→ **how did we implement such a solver?**

2. Implementation: from a matrix-based to a matrix-free approach



- ✓ Templated by `dim` and `VectorType`.
- ✓ Physics solver allows us to experiment with different non linear solvers (Newton, inexact, kinsol) and allow us to solve other physics.
- ✓ The `solve()` function is where everything happens: read/create mesh, setup DoFs, initial conditions, iterate, postprocess.
- ✓ AMG and ILU preconditioners available through `TrilinosWrappers`.

2. Implementation: from a matrix-based to a matrix-free approach



Note: We are interested in $p = 2, 3$ elements.

Two main issues:

- × Assembly of the matrix is time-consuming (time increases with the order of FE).
- × Large matrices lead to a memory-bandwidth bottleneck.

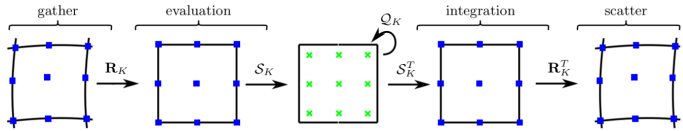
→ **how to improve this?**

2. Implementation: from a matrix-based to a matrix-free approach (Cont.)

Main idea: compute results of matrix-vector multiplication on-the-fly by defining an operator.

The operator $\mathcal{A}$ allows us to rewrite the matrix-vector multiplication $\boldsymbol{Au}$ as follows:

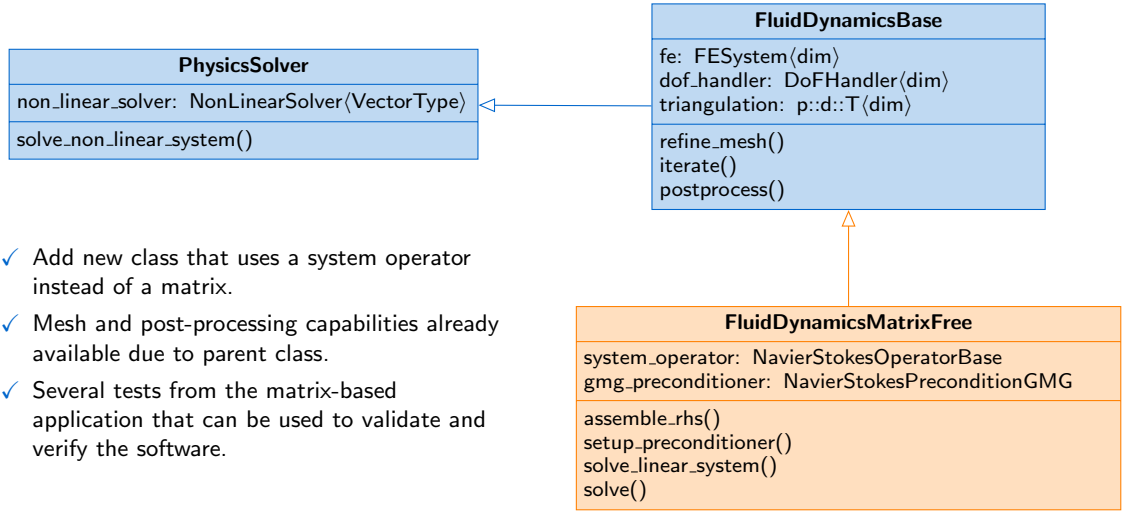
$$\boldsymbol{v} = \mathcal{A}(\boldsymbol{u}) = \left(\sum_k \boldsymbol{R}_k^T \boldsymbol{A}_k \boldsymbol{R}_k \right) \boldsymbol{u} = \sum_k \boldsymbol{R}_k^T \underbrace{\boldsymbol{\mathcal{S}}_k^T \boldsymbol{\mathcal{Q}}_k \boldsymbol{\mathcal{S}}_k}_{\boldsymbol{A}_k} \boldsymbol{R}_k \boldsymbol{u}$$



- We use the `deal.II` implementation by Kronbichler and Kormann (2012).
  Other libraries: MFEM (Anderson et al., 2021), libCEED (Brown et al., 2021).
- Uses cell-quadrature-approach, vectorization and sum-factorization.

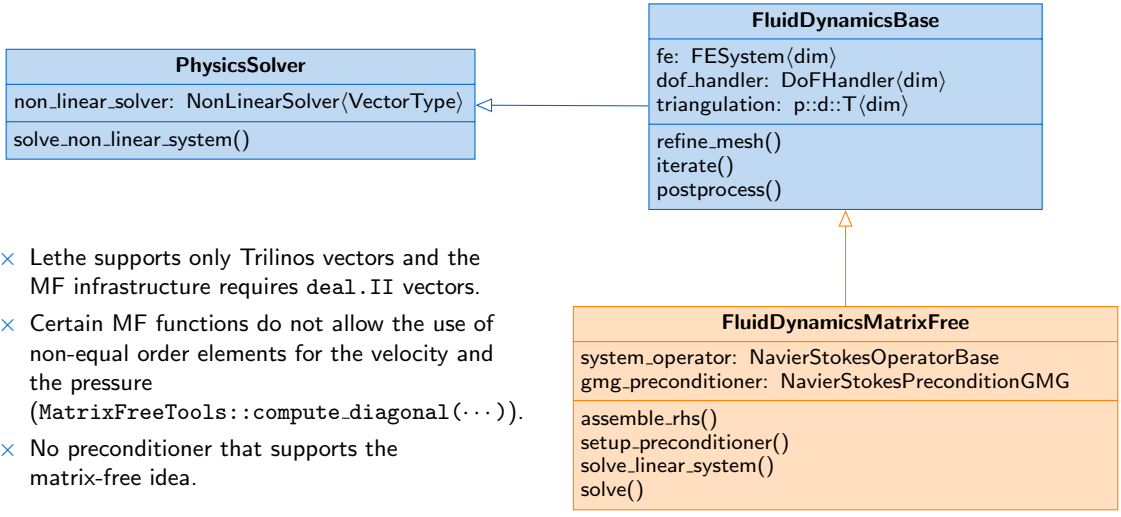
→ **how did we incorporate this idea in Lethe?**

2. Implementation: from a matrix-based to a matrix-free approach



- ✓ Add new class that uses a system operator instead of a matrix.
- ✓ Mesh and post-processing capabilities already available due to parent class.
- ✓ Several tests from the matrix-based application that can be used to validate and verify the software.

2. Implementation: from a matrix-based to a matrix-free approach



- × Lethe supports only Trilinos vectors and the MF infrastructure requires `deal.II` vectors.
- × Certain MF functions do not allow the use of non-equal order elements for the velocity and the pressure (`MatrixFreeTools::compute_diagonal(···)`).
- × No preconditioner that supports the matrix-free idea.

→ **how to implement such a preconditioner?**

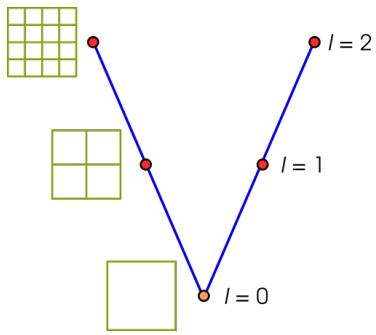
3. Linear solver: geometric multigrid preconditioner

A **geometric multigrid** was incorporated in our solver using the implementation available in `deal.II` (Munch et al., 2023; Clevenger et al., 2020).

> **Idea:** correct all components of the errors of the solution using a hierarchy of discretizations based on the geometrical refinement of the grid.

Main components:

- Operators for each level: $\mathcal{A}^{(l)}, \mathcal{A}^{(0)}$
- **Smoother** for each level
- Restriction operator from fine l to coarse level $l-1$: $R_l^{(l-1)}$
- Prolongation operator from coarse $l-1$ to fine level l: $P_{(l-1)}^{l}$
- **Coarse-grid solver**

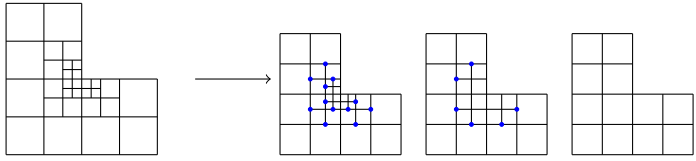


→ **for locally-refined meshes there are different geometric multigrid methods**

3. Linear solver: geometric multigrid preconditioner (Cont.)

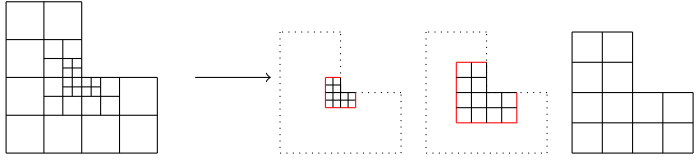
In this work we consider two approaches already implemented in `deal.II`:

- **global coarsening (GC):**



No internal interfaces, smoothers consider hanging nodes.

- **local smoothing (LS):**



Internal interfaces (edges), efficient transfer between MG levels, no hanging nodes.

→ **what do we know about these two approaches?**

3. Linear solver: geometric multigrid preconditioner (Cont.)

Munch P., Heister T., Prieto Saavedra L. and Kronbichler M. (2023). Efficient distributed matrix-free multigrid methods for locally refined meshes for FEM computations. *ACM Transactions on Parallel Computing*.

- Considered a Poisson and a Stokes problem with a block preconditioner.
- Evaluated the performance of both multigrid approaches using different metrics, among them:
  - **Vertical efficiency:** indication of how much data needs to be exchanged during inter-grid transfer → low number: large volume of communication.
  - **Workload efficiency:** measure that considers the amount of cells processed by each processor → low value: load imbalance during smoothing.

Main findings:

- For serial simulations, **LS is faster than GC**: total number of cells to perform smoothing on is lower and there is no cost for evaluating hanging-nodes constraints.
- For parallel simulations, **GC is faster than LS**: better load balance.

→ **how did we implement both of these approaches in Lethe?**

3. Linear solver: geometric multigrid preconditioner (Cont.)

NavierStokesPreconditionGMG
mg_operators: OperatorType mg_smoother: MGSmootherPrecondition mg_transfer_ls: LSTransferType mg_transfer_gc: GCTransferType mg_coarse: MGCoarseGridBase mg: Multigrid ls_mg_preconditioner: PreconditionMG gc_mg_preconditioner: PreconditionMG
NavierStokesPreconditionGMG() initialize(···) vmult(···)

- In the constructor: we set up the triangulations, the level operators (with constraints), and the transfer operators.
- In the initialize: transfer relevant quantities to mg levels, evaluate non linear terms for all operators, create smoother (with ev estimation), setup coarse grid solver and create the final preconditioner.

→ **This architecture allow us to reuse several parts of the multigrid across transient iterations...**

3. Linear solver: geometric multigrid preconditioner (Cont.)

How to choose the smoother?

- We use a relaxation scheme $x_{n+1} = x_n + \omega P^{-1} r_n$ (`PreconditionRelaxation`), where P is the diagonal of the operator.
- We calculate the ω using the eigenvalues for every level.

What about the coarse-grid solver?

- Implement different options: AMG, ILU, GMRES preconditioned by AMG/ILU, direct solver.

How to further reduce the cost of the preconditioner?

- Reuse the preconditioner across Newton iterations.
- Neglect terms involving the Hessian in the jacobian.
- Use of `FE_Q_iso_Q1` elements in coarse-grid level.

→ the solver is tested for different steady/transient problems

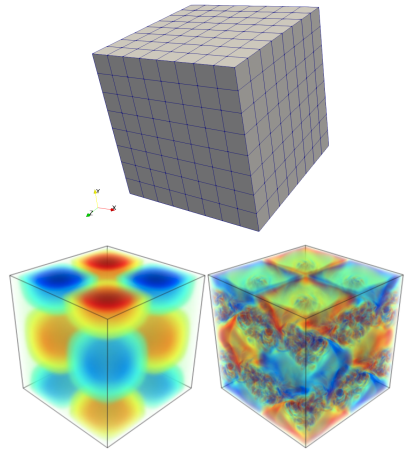
4. Numerical tests: Taylor-Green vortex

Benchmark to study vortex dynamics and the energy cascade:

- Transient problem: fixed CFL$=1$, BDF2.
- Domain: $\Omega = (-\pi, \pi)^3$.
- Periodic boundary conditions.
- $\mathrm{Re} = 1600$.
- $Q_p Q_p$ elements with $p = 1, 2, 3$.
- Initial condition:

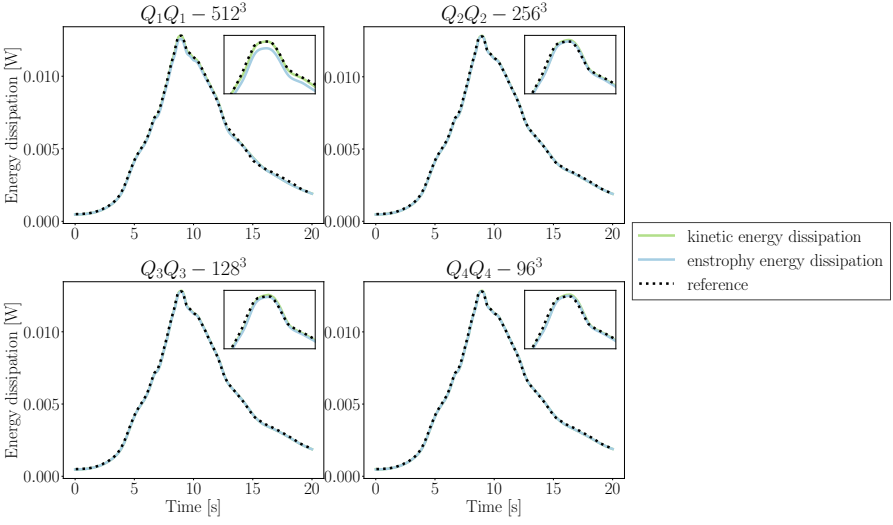
$$\boldsymbol{u} = \begin{pmatrix} \sin(x)\cos(y)\cos(z) \\ -\cos(x)\sin(y)\cos(z) \\ 0 \end{pmatrix}$$

$$p = \frac{1}{16}(\cos(2x) + \cos(2y))(\cos(2z) + 2)$$



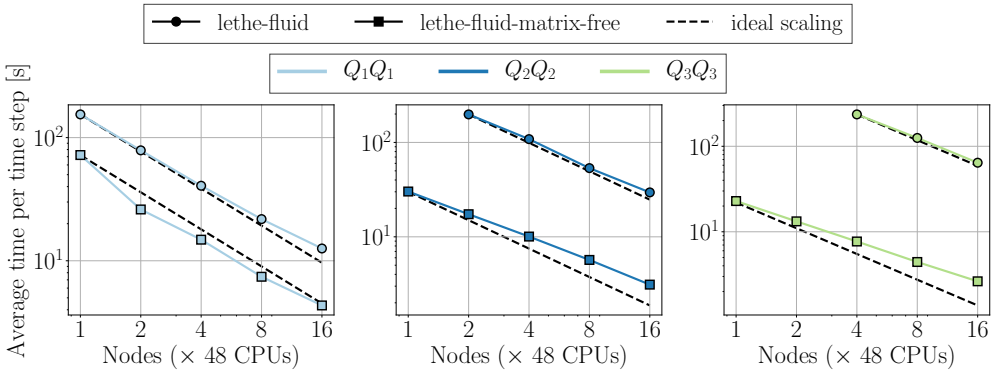
$\rightarrow$ **the problem allows us to study parallel scalability**

4. Numerical tests: Taylor-Green vortex (Cont.)



- For the next analysis, we focus on the measurements over the interval $(6, 10]$.

4. Numerical tests: Taylor-Green vortex (Cont.)

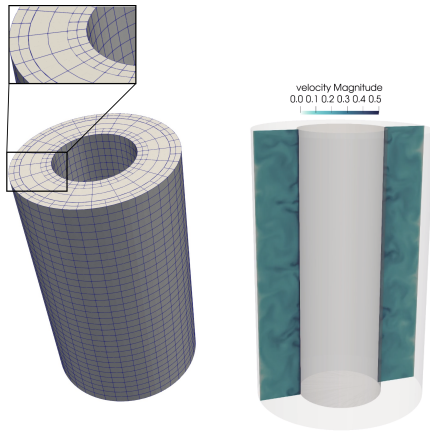


- Matrix-free faster than matrix-based: ∼3x for Q_1Q_1, ∼10x for Q_2Q_2, ∼28x Q_3Q_3.
- ∼ 34M DoFs

4. Numerical tests: turbulent Taylor-Couette

Complex turbulent flow problem:

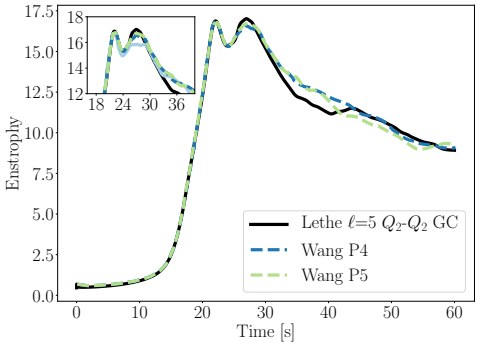
- Annular flow between two coaxial cylinders.
- Inner cylinder with a fixed angular velocity.
- Outer cylinder is static.
- Curved walls.
- Transient: BDF2, fixed CFL=1.
- $\mathrm{Re} = 4000$.
- $Q_p Q_p$ elements with $p = 1, 2$.
- **Global static mesh refinement** ℓ with one additional refinement next to the walls.
- Simulation time: 60s.



→ **allow us to study serial and large-scale parallel behavior of the preconditioners**

4. Numerical tests: turbulent Taylor-Couette - validation

The quality of the results is evaluated through the enstrophy and compared to the reference results from Wang and Jourdan (2021).

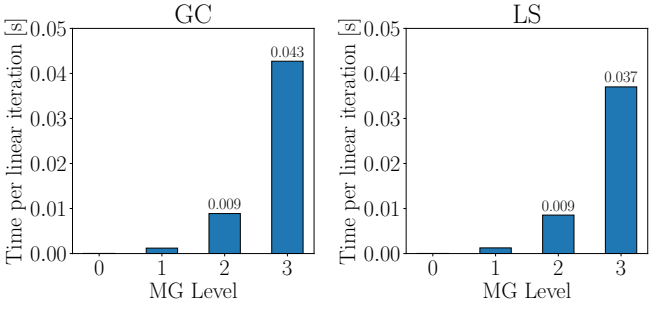


- For the next analysis, we focus on the measurements over the interval $(15, 30]$.

4. Numerical tests: turbulent Taylor-Couette - serial run

We analyze average number of linear iterations per Newton iteration (#its) and the total simulation time (t):

ℓ	Q_1Q_1				Q_2Q_2			
	GC		LS		GC		LS	
	#its	$t[s]$	#its	$t[s]$	#its	$t[s]$	#its	$t[s]$
2	10	82	15	98.7	9	754	12	640

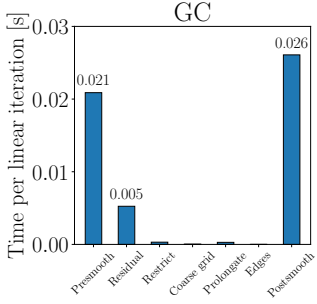


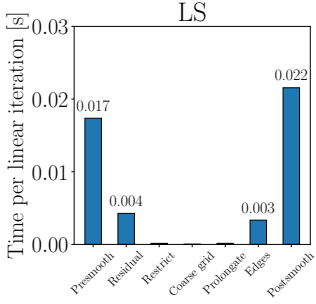
- GC is faster in terms of total run time (less iterations); but each linear iteration is more expensive.
- One GC V-cycle is 22% slower than a LS V-cycle:
  - More cells in the case of GC ($\sim$12%).
  - Application of expensive hanging node constraints (Munch et al., 2022).

4. Numerical tests: turbulent Taylor-Couette - serial run

We analyze average number of linear iterations per Newton iteration (#its) and the total simulation time (t):

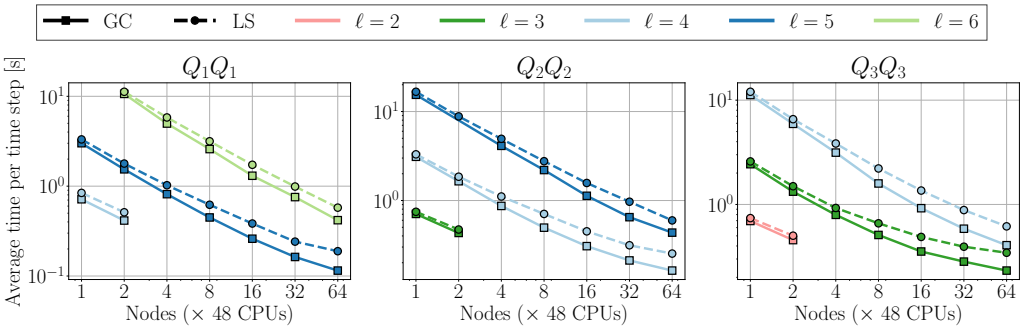
ℓ	Q_1Q_1				Q_2Q_2			
	GC		LS		GC		LS	
	#its	$t[s]$	#its	$t[s]$	#its	$t[s]$	#its	$t[s]$
2	10	82	15	98.7	9	754	12	640





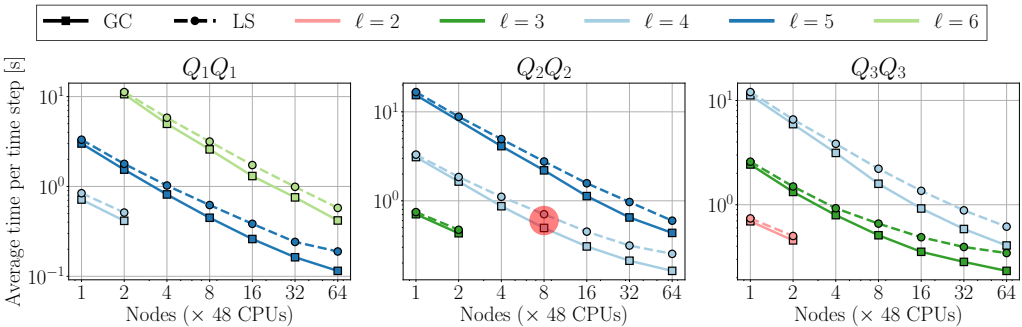
- Coarse grid is cheap for this case, as it has only 20 cells. Coarse-grid solver: AMG.
- Pre- and postsmoothing are the most expensive components of the algorithm. LS is faster than GC.
- Edges (application of non-zero Dirichlet BCs) cost is significant in LS.

4. Numerical tests: turbulent Taylor-Couette - large-scale parallel behavior



- Up to 6M cells in Q_1Q_1, 1M cells in Q_2Q_2, 150K cells in Q_3Q_3
- GC faster than LS in all cases for all refinements ℓ.
- GC scales better as we increase the number of nodes.

4. Numerical tests: turbulent Taylor-Couette - large-scale parallel behavior

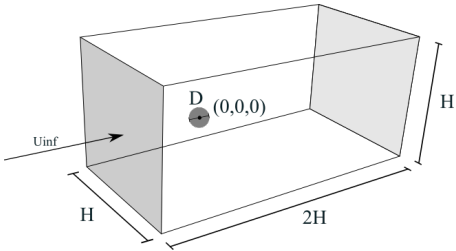


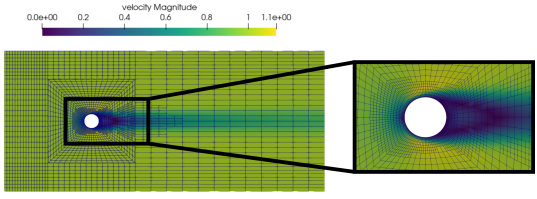
- Up to 6M cells in Q_1Q_1, 1M cells in Q_2Q_2, 150K cells in Q_3Q_3
- GC faster than LS in all cases for all refinements ℓ.
- GC scales better as we increase the number of nodes.
- GC is 1.3x faster than LS. For the same case with one node, LS and GC need the same amount of time.

4. Numerical tests: flow around a sphere

Challenging external flow problem:

- Fixed entrance velocity $U_\infty = 1$ in the x direction.
- Steady-state, $\mathrm{Re} = 100$. Initial condition: ramp up Re starting with $\mathrm{Re} = 10$.
- No slip boundary conditions around the sphere and slip boundary conditions on the wall.
- $Q_p Q_p$ elements with $p = 1, 2$.
- Initial global refinement ℓ.
- **Dynamic mesh refinement:** using Kelly error estimator on the pressure.



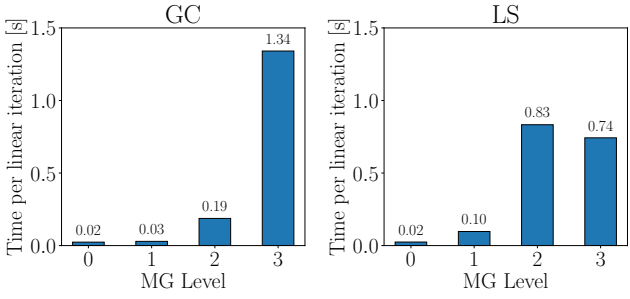


→ **allow us to study serial and moderately parallel behavior of the preconditioners**

4. Numerical tests: flow around a sphere - serial run

We analyze average number of linear iterations per Newton iteration (#its) and the total simulation time (t):

ℓ	Q_1Q_1				Q_2Q_2			
	GC		LS		GC		LS	
	#its	$t[s]$	#its	$t[s]$	#its	$t[s]$	#its	$t[s]$
1	20	37	20	37	24	207	24	207
2	27	173	27	182	42	363	38	346

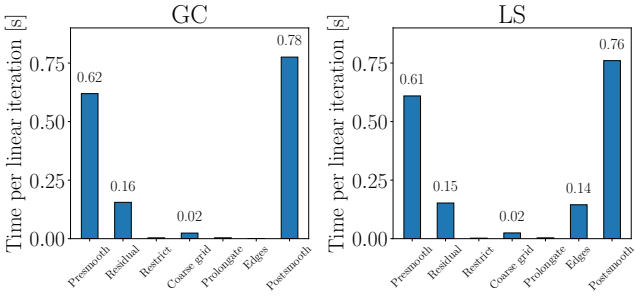


- In this case, the coarse mesh has 1024 cells. We use GMRES preconditioned by ILU.
- Approximately same number of iterations and similar time to solution for both methods.
- Iterations increase a lot with increasing order $\rightarrow$ possibly due to simple smoother.

4. Numerical tests: flow around a sphere - serial run

We analyze average number of linear iterations per Newton iteration (#its) and the total simulation time (t):

ℓ	Q_1Q_1				Q_2Q_2			
	GC		LS		GC		LS	
	#its	$t[s]$	#its	$t[s]$	#its	$t[s]$	#its	$t[s]$
1	20	37	20	37	24	207	24	207
2	27	173	27	182	42	363	38	346

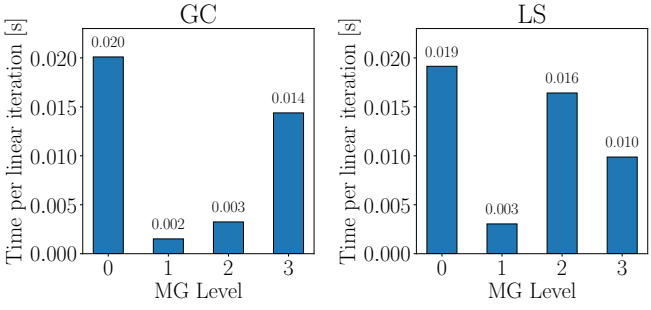


- Pre- and postsmoothing are the most expensive components of the algorithm.
- Edges cost is significant for LS.

4. Numerical tests: flow around a sphere - parallel run (192 processes)

We analyze average number of linear iterations per Newton iteration (#its) and the total simulation time (t):

ℓ	Q_1Q_1				Q_2Q_2			
	GC		LS		GC		LS	
	#its	$t[s]$	#its	$t[s]$	#its	$t[s]$	#its	$t[s]$
1	20	3.46	20	3.46	25	7.84	25	12.7
2	27	4.13	27	5.22	42	7.51	39	12.2

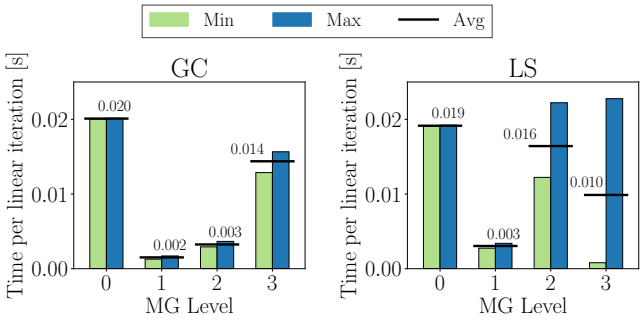


- Speed up (LS/GC) of 1.2x for Q_1Q_1, 1.6x for Q_2Q_2
- Same number of iterations for both, but GC faster than LS in all cases.
- The coarse-grid problem is the bottleneck.

4. Numerical tests: flow around a sphere - parallel run (192 processes)

We analyze average number of linear iterations per Newton iteration (#its) and the total simulation time (t):

ℓ	Q_1Q_1				Q_2Q_2			
	GC		LS		GC		LS	
	#its	$t[s]$	#its	$t[s]$	#its	$t[s]$	#its	$t[s]$
1	20	3.46	20	3.46	25	7.84	25	12.7
2	27	4.13	27	5.22	42	7.51	39	12.2



- LS has load imbalance for the finest levels.
- GC:
  - Vertical efficiency: 7.48%
  - Workload efficiency: 99%
- LS:
  - Vertical efficiency: 99.83%
  - Workload efficiency: 52%.
- The workload efficiency is the most relevant metric, since the smoother is dominating the overall cost of the v-cycle.

5. Outlook

What comes next?

- Evaluate other smoothers to improve performance for steady-state cases.
- Evaluate the efficiency of the preconditioners for other benchmarks.
- Experiment with p-multigrid.
- Compare stabilization techniques.
- Couple the matrix-free solver with other physics (e.g., heat transfer, VOF).

Thank you!

lethe

https://github.com/chaos-polymtl/lethe

References I

Anderson, R., Andrej, J., Barker, A., Bramwell, J., Camier, J.-S., Cerveny, J., Dobrev, V., Dudouit, Y., Fisher, A., Kolev, T., Pazner, W., Stowell, M., Tomov, V., Akkerman, I., Dahm, J., Medina, D., and Zampini, S. (2021). Mfem: A modular finite element methods library. *Computers & Mathematics with Applications*, 81:42–74. Development and Application of Open-source Software for Problems with Numerical PDEs.

Arndt, D., Bangerth, W., Bergbauer, M., Feder, M., Fehling, M., Heinz, J., Heister, T., Heltai, L., Kronbichler, M., Maier, M., Munch, P., Pelteret, J.-P., Turcksin, B., Wells, D., and Zampini, S. (2023). The `deal.II` library, version 9.5. *Journal of Numerical Mathematics*, 31(3):231–246.

Blais, B., Barbeau, L., Bibeau, V., Gauvin, S., El Geitani, T., Golshan, S., Kamble, R., Mirakhori, G., and Chaouki, J. (2020). Lethe: An open-source parallel high-order adaptative cfd solver for incompressible flows. *SoftwareX*, 12:100579.

Brown, J., Abdelfattah, A., Barra, V., Beams, N., Camier, J.-S., Dobrev, V., Dudouit, Y., Ghaffari, L., Kolev, T., Medina, D., Pazner, W., Ratnayaka, T., Thompson, J., and Tomov, S. (2021). libceed: Fast algebra for high-order element-based discretizations. *Journal of Open Source Software*, 6(63):2945.

Clevenger, T. C., Heister, T., Kanschat, G., and Kronbichler, M. (2020). A flexible, parallel, adaptive geometric multigrid method for fem. *ACM Trans. Math. Softw.*, 47(1).

Kronbichler, M. and Kormann, K. (2012). A generic interface for parallel cell-based finite element operator application. *Computers and Fluids*, 63:135–147.

Munch, P., Heister, T., Saavedra, L. P., and Kronbichler, M. (2023). Efficient distributed matrix-free multigrid methods on locally refined meshes for FEM computations. *ACM Transactions on Parallel Computing*, pages 1–34.

Munch, P., Ljungkvist, K., and Kronbichler, M. (2022). Efficient application of hanging-node constraints for matrix-free high-order fem computations on cpu and gpu. In *High Performance Computing: 37th International Conference, ISC High Performance 2022, Hamburg, Germany, May 29 – June 2, 2022, Proceedings*, page 133–152, Berlin, Heidelberg. Springer-Verlag.

Wang, Z. J. and Jourdan, E. (2021). Benchmark for scale-resolving simulation with curved walls: the Taylor Couette flow. *Advances in Aerodynamics*, 3(1).

Numerical implementation: the cell matrix

In reality, the local element matrix can be decomposed as the product of three matrices:

$$\boldsymbol{A}_k = \boldsymbol{\mathcal{S}}_K^T \boldsymbol{\mathcal{Q}}_K \boldsymbol{\mathcal{S}}_K \tag{1}$$

But how? For this, we need to take a closer look to the diffusion term in the Jacobian expression:

$$F'(\boldsymbol{u}, p)[\boldsymbol{\delta u}, \delta p] = \cdots + \nu(\nabla \boldsymbol{v}, \nabla \boldsymbol{\delta u})_\Omega + \cdots \tag{2}$$

If we insert the discretization, this looks like this:

$$\begin{aligned}
\nu(\nabla \boldsymbol{v}, \nabla \boldsymbol{\delta u})_\Omega &= \nu \sum_k (\nabla_{\boldsymbol{x}} \boldsymbol{v}_k, \nabla_{\boldsymbol{x}} \boldsymbol{u}_k)_{\Omega_k} \\
&= \nu \sum_k V_{k,i}^T (\nabla_{\boldsymbol{x}} \phi_i, U_{k,j} \nabla_{\boldsymbol{x}} \phi_j)_{\Omega_k} \\
&= \nu \sum_k V_{k,i}^T \underbrace{\int_{\Omega_k} (\nabla_{\boldsymbol{x}} \phi_i)^T \nabla_{\boldsymbol{x}} \phi_j U_{k,j} \mathrm{d}\boldsymbol{x}}_{1}
\end{aligned}$$

where $\nabla_{\boldsymbol{x}}$ is the gradient in physical space.

$\longrightarrow$ now we use the **cell-quadrature approach** for 1

4. Numerical implementation: cell-quadrature approach

Reminder: in finite elements we define a mapping to go from the physical space $\boldsymbol{x}$ to the reference space $\boldsymbol{\xi}$, which introduces the Jacobian matrix $\boldsymbol{J}$ in several terms. In addition, we use numerical quadrature to approximate the integral: $\int F(\boldsymbol{\xi})\mathrm{d}\boldsymbol{\xi} \simeq \sum_q \omega_q g(\boldsymbol{\xi}_q)$.

We can further develop the expression as:

$$\begin{aligned}\int_{\Omega_k} (\nabla_{\boldsymbol{x}}\phi_i)^T \nabla_{\boldsymbol{x}}\phi_j U_{k,j}\mathrm{d}\boldsymbol{x} &= \int_{\Omega_k} \left(\boldsymbol{J}_k^T \nabla_{\boldsymbol{\xi}}\phi_i\right)^T \left(\boldsymbol{J}_k^{-T} \nabla_{\boldsymbol{\xi}}\phi_j\right) U_{k,j} \left(\det \boldsymbol{J}_k\right) \mathrm{d}\boldsymbol{\xi} \\ &= \sum_q \underbrace{(\nabla_{\boldsymbol{\xi}_q}\phi_i)^T}_{\boldsymbol{\mathcal{S}}_k^T} \underbrace{\boldsymbol{J}_k^T \omega_q (\det \boldsymbol{J}_k) \boldsymbol{J}_k^{-T}}_{\boldsymbol{\mathcal{Q}}_k} \underbrace{(\nabla_{\boldsymbol{\xi}_q}\phi_j)}_{\boldsymbol{\mathcal{S}}_k} U_{k,j}\end{aligned}$$

From left to right:

1. Transform the vector values on the local DoFs to a vector of gradients in quadrature points.
2. Multiply the gradients by the integration weights and Jacobian information.
3. Apply the second gradient to have a vector of Laplacian values on the cell DoFs.

Multigrid algorithm

Algorithm 1 Solves $Ax = b$ approximately

function MultigridVCycle(l, $A^{(l)}$, $x^{(l)}$, $b^{(l)}$)
if $l = 0$ **then**
  $x^{(0)} \leftarrow$ CoarseGridSolver($A^{(0)}$, $x^{(0)}$, $b^{(0)}$) (e.g. Algebraic Multigrid (AMG))
else
  $x^{(l)} \leftarrow$ Smoother ($A^{(l)}$, $x^{(l)}$, $b^{(l)}$)
  $r^{(l)} \leftarrow b^{(l)} - A^{(l)} x^{(l)}$
  $b^{(l-1)} \leftarrow R_l^{(l-1)} r^{(l)}$
  $x^{(l-1)} \leftarrow$ MultigridVCycle ($l - 1$, $A^{(l-1)}$, 0, $b^{(l-1)}$)
  $x^{(l)} \leftarrow x^{(l)} + P_{(l-1)}^{l} x^{(l-1)}$
  $x^{(l)} \leftarrow$ Smoother($A^{(l)}$, $x^{(l)}$, $b^{(l)}$)
  return $x^{(l)}$
end if

$\longrightarrow$ the difference between multigrid methods relies in the way the **hierarchy of discretizations** is defined

Flow around a sphere: pressure coefficient

$$C_D = C_p + C_\tau = \frac{\mathrm{F}_p}{\frac{1}{2}\rho U_\infty^2 \left(\frac{\pi}{4} D^2\right)} + \frac{\mathrm{F}_\tau}{\frac{1}{2}\rho U_\infty^2 \left(\frac{\pi}{4} D^2\right)}$$