LifeV, parallel framework, and applications

Simone Deparis
CMCS–MATHICSE–SB–EPFL
EPFL Lausanne

PDESoft 2014, Heidelberg, July 15th 2014

Acknowledgements

Prof. Alfio Quarteroni
Dr. Gwenol Granperrin
Davide Forti
Dr. Luca Dede

LifeV community `http://www.lifev.org`
Trilinos community `http://www.trilinos.org`
EPFL, Politecnico di Milano, INRIA, Emory University
FNS, FP6 (Haemodel), FP7 - ERC-AdvG (Mathcard), HP2C, Aneurisk, ...

LifeV project

Finite Element Library for the solution of PDEs

- C++, object oriented
- parallel and serial versions
- distributed under LGPL
- about 30 active developers

- CMCS – EPFL
- E(CM)^2 – Emory
- MOX – Polimi
- ESTIME– INRIA

Research code oriented to the development and test of new numerical methods and algorithms
Aim: effective tool for solving complex engineering problems

History

<2001 LifeI, LifeII, LifeIII: Fortran 2D finite element codes (CRS4, Politecnico di Milano, CMCS)

2001 early development of LifeV at Politecnico di Milano, INRIA, and EPFL (leaders A. Quarteroni, L. Formaggia, A. Veneziani, J.F. Gerbau)

2006 development of the parallelism in LifeV. Inclusion of Trilinos (G. Fourestey, S. Deparis).

2008 E(CM)^2 – Emory University joins the project

2011 Migration to git, tribits and cmake

Developers and target users

Research code oriented to the development and test of new numerical methods and algorithms
Developers are:

- Researchers
- Post docs
- PhD students

Target users are the developers themselves, master students, and other researchers.
Medium term target is to make LifeV more accessible to external users.

Licence ans copyright

LifeV is distributed under the LGPL 3 by a consortium of the main institutions involved.

Access to the development repository is given to people that accepts to transfer the copyright to the consortium. Authorship is of course kept to the author.

Examples of applications

- Darcy Solver: FE spaces: P1 for the pressure and low order Raviart-Thomas for the velocity.
- Three phase flow in the liver
- Cell culture in Orbitally Shaken Reactor (Bifluid, with level set and stabilization)
- Structural solver with non-linear and anisotropic materials
- Electrical activity in the heart (mono- and bi-domain solvers)
- Fluid-Structure interaction for vascular flows
- Geometrical multiscale modeling for vascular flows
- Integrated cardio-vascular simulations

Development tools

- TriBits and **CMake**
- nightly builds in opt and debug modes
- collaborative tool: **Git**
- **Redmine system** + GitoLite `http://cmcsforge.epfl.ch`,
- Google groups for mailing lists
- Google app educational for `lifev.org`
- Continuous distribution through **GitHub**
  `http://github.com/lifev/lifev`

Modules, aka mini-packages

LifeV is devided into modules, namely:

- **Core**: Finite Element basics, mesh classes, time discretization,
- BC interface: Boundary conditions
- Darcy
- Eta: assembly by expression templates
- FSI: Fluid-Structure interaction
- Heart
- Level set
- Navier Stokes
- One D FSI: 1D arterial model
- Structure
- Zero dimensional
- Multiscale
- Electrophysiology (not yet distributed)

They represent specific features. There are dependencies among modules
Each module includes a specific **testsuite**

Branch review scheme

Code submitted to the community comes as a **branch** to be merged into master.
Before merging, the code is submitted for open review to the developers. A specific reviewer is selected.

Review includes:

- compliance with the coding guidelines
- relevant tests
- (code analysis with Valgrind)
- list of reccomendations for further development (a todo list for after merging)

Until now, review has been quite selective, we are aiming at a little relaxation

Since January 2013 (Developers meeting), master is released on GitHub.

Why Trilinos (extract form Trilinos survey)

Were there alternatives to Trilinos that you investigated? If so, what made you decide to use Trilinos?

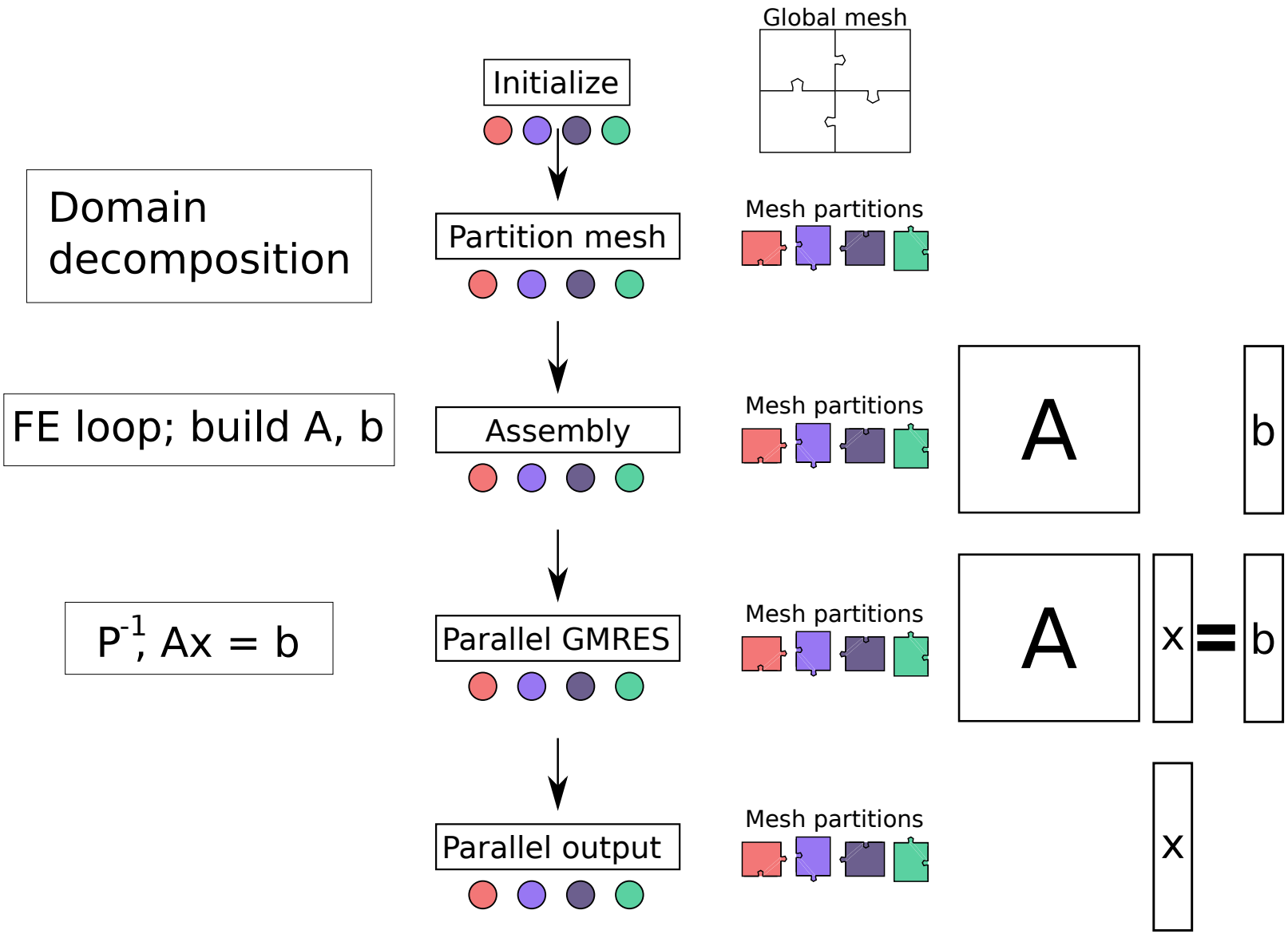
Yes, the most competitive alternative was petsc. However we found Trilinos more mature and with a larger set of tools (actually packages). Trilinos allowed us a smooth switch to MPI and offers interfaces to this API. It was also important that the source is written in c++ and that the following tools are provided:

- distributed sparse matrix data-structures
- linear system solvers and preconditioners
- parallel IO
- generalized eigenvalue solvers.

Trilinos' most used features

Tools	Epetra stack
Parallel linear algebra	Epetra
Matrix tools and extensions	EpetraExt
Graph operations	Zoltan & Isorropia
Iterative linear solver	AztecOO & Belos
Direct solvers	Amesos
DD preconditioners	Ifpack
Multi-level preconditioners	ML

Parallel finite element loop



E.g., Navier–Stokes equations
with G. Grandperring

Let Ω be a bounded domain in $\mathbb{R}^3$. The Navier–Stokes equations for an incompressible viscous flow read:

$$\begin{aligned}
\frac{\partial}{\partial t}\mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u} - \nu\Delta\mathbf{u} + \nabla p &= \mathbf{f} && \text{in } \Omega\times(0,T] \\
\nabla\cdot\mathbf{u} &= 0 && \text{in } \Omega\times(0,T] \\
\mathbf{u} &= \varphi && \text{on } \Gamma_D\times(0,T] \\
\nu\frac{\partial\mathbf{u}}{\partial\mathbf{n}} - p\mathbf{n} &= 0 && \text{on } \Gamma_N\times(0,T] \\
\mathbf{u} &= \mathbf{u}_0 && \text{at } t=0
\end{aligned}$$

where Γ_D and Γ_N are the Dirichlet and Neumann parts of the boundary respectively, $\mathbf{u}$ is the fluid velocity, p the pressure, ν the kinematic viscosity of the fluid, $\mathbf{f}$ the external forces, and φ a given function.

Discretization

Time discretization using e.g. semi-implicit Euler scheme:

$$\begin{aligned} \frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t} + \mathbf{u}^n \cdot \nabla \mathbf{u}^{n+1} - \nu \Delta \mathbf{u}^{n+1} + \nabla p^{n+1} &= \mathbf{f}^{n+1} && \text{in } \Omega \\ \nabla \cdot \mathbf{u}^{n+1} &= 0 && \text{in } \Omega \\ \mathbf{u}^{n+1} &= \boldsymbol{\varphi} && \text{on } \Gamma_D \\ \nu \frac{\partial \mathbf{u}^{n+1}}{\partial \mathbf{n}} - p^{n+1}\mathbf{n} &= 0 && \text{on } \Gamma_N \end{aligned}$$

FE discretization using $\mathbb{P}_2 - \mathbb{P}_1$ finite elements on tetrahedral unstructured meshes:

$$\begin{pmatrix} F(\mathbf{U}^n) & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{U}^{n+1} \\ \mathbf{P}^{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{G}^{n+1}(\mathbf{U}^n) \\ \mathbf{0} \end{pmatrix}$$

Designing a preconditioner for HPC

Wish list:

1. The algorithms involved to build and apply the preconditioner must be both weakly and strongly scalable.
2. The preconditioner should be optimal.
3. The preconditioner should be scalable in terms of number of iterations.
4. The preconditioner should be robust with respect to the physical parameter (e.g. viscosity ν).
   $\Rightarrow$ This property ensures that the preconditioner handles a wide range of Reynolds numbers; for vascular flows, the typical Reynolds number in large arteries ranges from $\Re e = 200$ up to $\Re e = 4000$ (ascending aorta).

Classical preconditioners for N–S

- SIMPLE

$$P^{-1}_{SIMPLE} = \begin{pmatrix} I & -\frac{1}{\alpha}D^{-1}B^T \\ 0 & \frac{1}{\alpha}I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -\tilde{S}^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -B & I \end{pmatrix} \begin{pmatrix} F^{-1} & 0 \\ 0 & I \end{pmatrix},$$

where $\alpha \in (0, 1]$ is a damping parameter, $\tilde{S} = BD^{-1}B^T$, and D is the diagonal of F,

Patankar, Spalding. A calculation procedure for heat, mass and momentum transfer in three dimensional parabolic flows. *International J. on Heat and Mass Transfer, 15:1787–1806, 1972.*

- Yosida

$$P^{-1}_{Yosida} = \begin{pmatrix} I & -F^{-1}B^T \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -B & I \end{pmatrix} \begin{pmatrix} F^{-1} & 0 \\ 0 & I \end{pmatrix},$$

with $S = \Delta t B M_{\mathbf{u}}^{-1} B^T$.

Quarteroni, Saleri, and Veneziani. Analysis of the Yosida method for the incompressible Navier–Stokes equations. *J. Math. Pures Appl., 1999.*

- PCD

$$P^{-1}_{PCD} = \begin{pmatrix} F^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & -B^T \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -A_p^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & F_p \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & M_p^{-1} \end{pmatrix}.$$

Silvester, Elman, Kay, Wathen. Efficient preconditioning of the linearized Navier-Stokes equations for incompressible flow. *J. Comput. Appl. Math., 2001.*

Elman, Tuminaro. Boundary conditions in approximate commutator preconditioners for the Navier–Stokes equations. *Electron. Trans. Numer. Anal., 2009.*

Approximate preconditioners for N–S

- Approximate SIMPLE (aSIMPLE)

$$P_{aSIMPLE}^{-1} = \begin{pmatrix} I & -\frac{1}{\alpha} D^{-1} B^T \\ 0 & \frac{1}{\alpha} I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -\hat{\tilde{S}}^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -B & I \end{pmatrix} \begin{pmatrix} \hat{F}^{-1} & 0 \\ 0 & I \end{pmatrix},$$

  where $\alpha \in (0, 1]$ is a damping parameter and $\tilde{S} = BD^{-1}B^T$.

- Approximate Yosida (aYosida)

$$P_{aYosida}^{-1} = \begin{pmatrix} I & -\hat{F}^{-1} B^T \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -\hat{S}^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -B & I \end{pmatrix} \begin{pmatrix} \hat{F}^{-1} & 0 \\ 0 & I \end{pmatrix},$$

  with $S = \Delta t B M_{\mathbf{u},\ell}^{-1} B^T$.

- Approximate PCD (aPCD)

$$P_{aPCD}^{-1} = \begin{pmatrix} \hat{F}^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & -B^T \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -\hat{A}_p^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & F_p \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & \hat{M}_p^{-1} \end{pmatrix}.$$

where $\hat{}$ denotes the use of a preconditioner to approximate the inverse

Deparis, Grandperrin, Quarteroni. Approximate preconditioners for the Navier-Stokes equations in hemodynamic simulations. *Computers & Fluids, vol. 92, p. 253-273, 2014.*

Inverses approximation
Details on the preconditioners

- $\hat{F}^{-1}$ and $(BM_{\mathbf{u},\ell}^{-1}B^T)^{-1}$ are replaced by a 2-level Schwarz preconditioner; the first level is applied without overlap with a coarse grid correction. The subdomain problems are solved using exact factorization.

- $\hat{A}_p^{-1}$ and $(BD^{-1}B^T)^{-1}$ are replaced by a V-cycle AMG with 2 sweeps of symmetric Gauss-Seidel as smoother (presmoothing only), exact factorization for the coarsest level.
The AMG is implemented in the `ML` package in `Trilinos`.

> Sala. An Object-Oriented Framework for the Development of Scalable Parallel Multilevel Preconditioners", *ACM Transactions on Mathematical Software, 2006.*

- $\hat{M}_p^{-1}$ is replaced by the inverse of the diagonal lumped mass matrix.

Numerical results
Simulation protocol

- Linear problem solved at each timestep with right preconditioned GMRES;
- Stopping criteria based on the residual scaled by the right hand side:

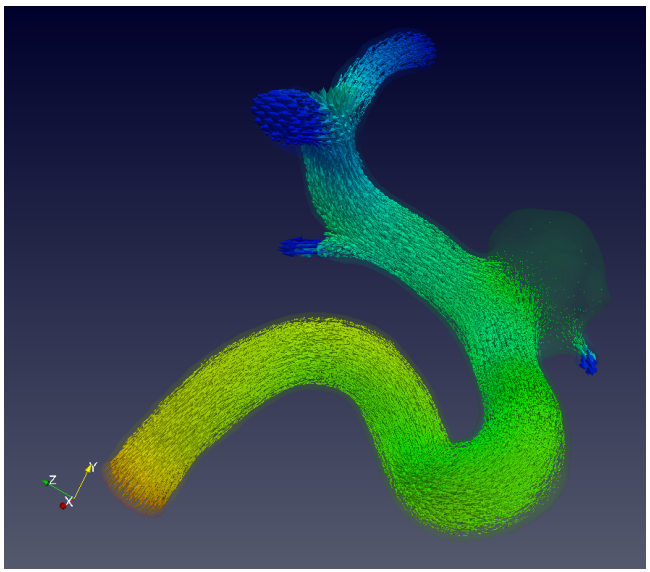
$$\|\mathbf{b} - \mathcal{A}\mathbf{x}_k\|_2 \leq 10^{-6}\|\mathbf{b}\|_2,$$

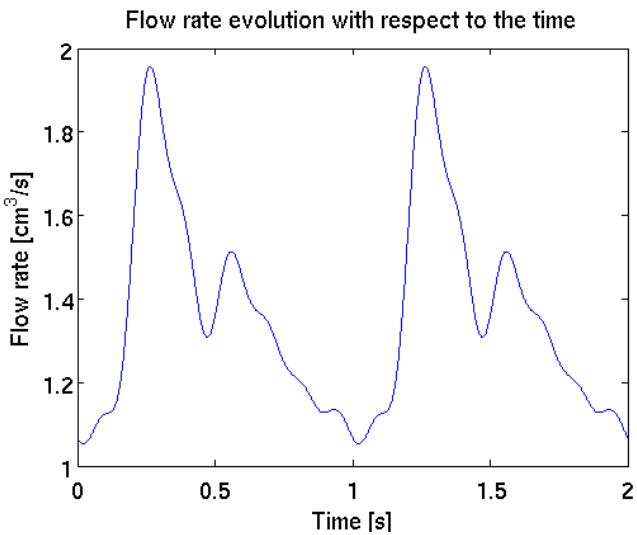
  where $\|\cdot\|_2$ denotes the ℓ_2 norm of the vector of the nodal finite element solution.
- GMRES is never restarted.
- The simulations were carried out using `LifeV` (www.lifev.org) on the Monte Rosa Cray XE6 at the CSCS, Lugano, Switzerland.

Number of nodes	1496
Number of processors per node	2x16-core AMD Interlagos
Processors frequency	2.1 GHz
Processors shared memory	32 GB DDR3
Peak performance	402 Teraflop/s.
Network	Gemini 3D torus

Flow in a rigid vessel

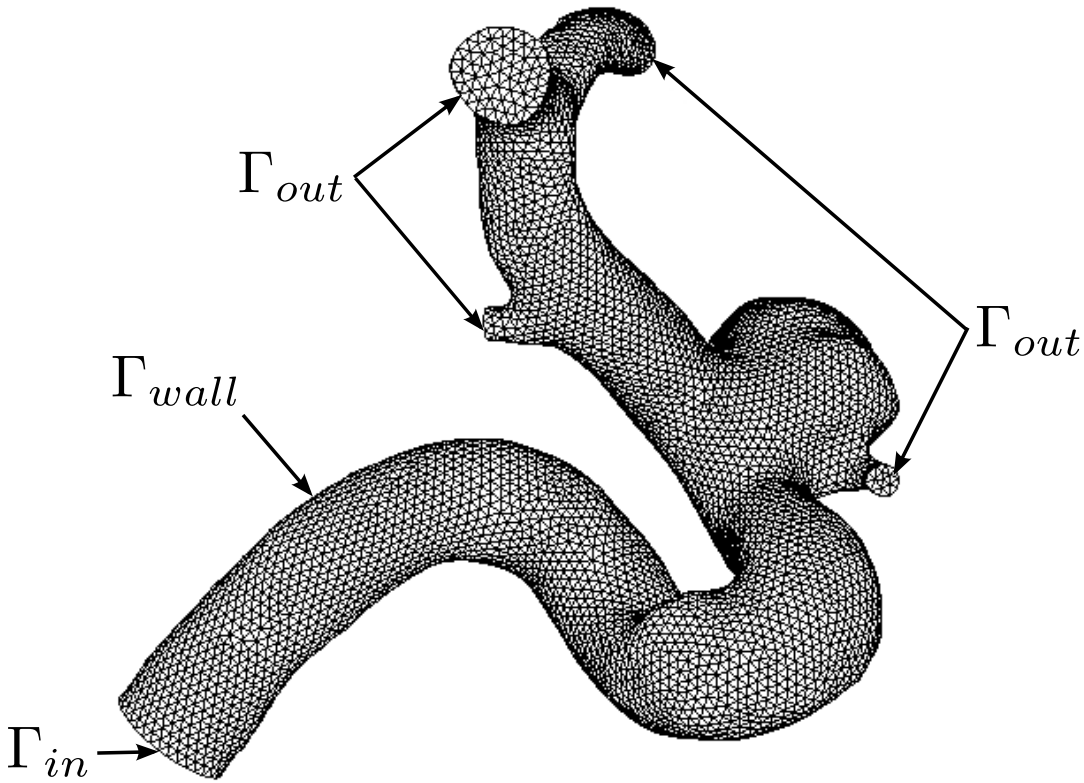
All our preconditioners are tested and tuned on a benchmark relevant for medical applications ($\nu = 0.035\ \frac{cm^2}{s}$, $\Re e = 400$).





Mesh	Velocity DoFs	Pressure DoFs	h_{min}	h_{av}	h_{max}
Coarse	597,093	27,242	0.015	0.035	0.059
Medium	4,557,963	199,031	0.005	0.018	0.051
Fine	35,604,675	1,519,321	0.0026	0.0097	0.0277

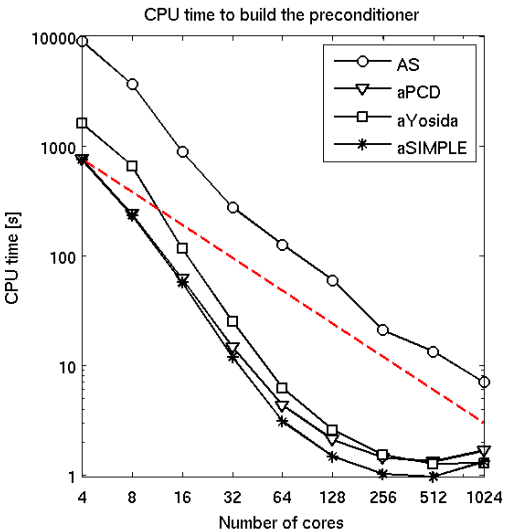
Flow in a rigid vessel (coarse grid)



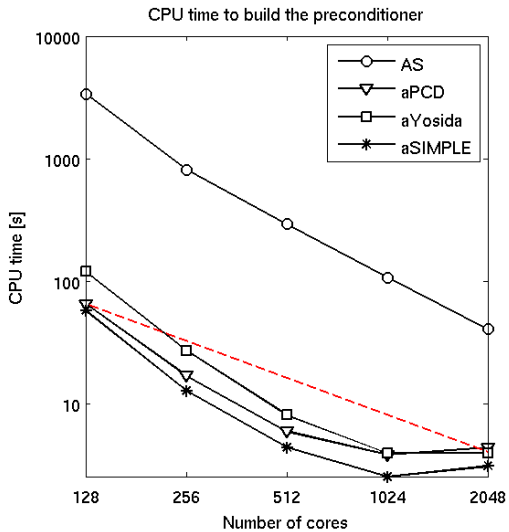
$$
\begin{aligned}
\mathbf{u} &= 0 && \text{on } \Gamma_{wall}, \\
\mathbf{u} &= \varphi_{flux}\mathbf{n} && \text{on } \Gamma_{in}, \\
\nu\frac{\partial \mathbf{u}}{\partial \mathbf{n}} - p\mathbf{n} &= 0 && \text{on } \Gamma_{out},
\end{aligned}
$$

Time to build the preconditioners

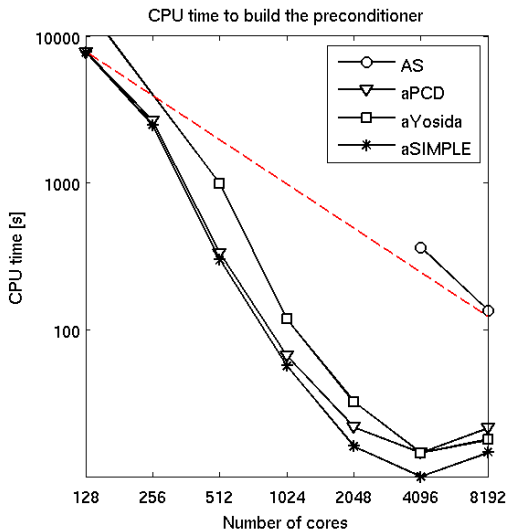
Coarse
max. 1024 cores



Medium
max. 2048 cores



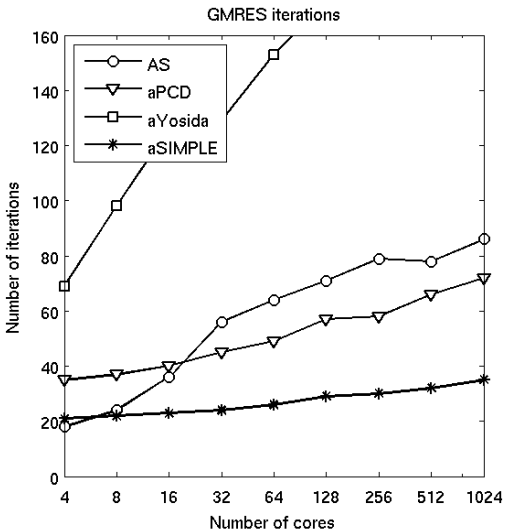
Fine
max. 8192 cores



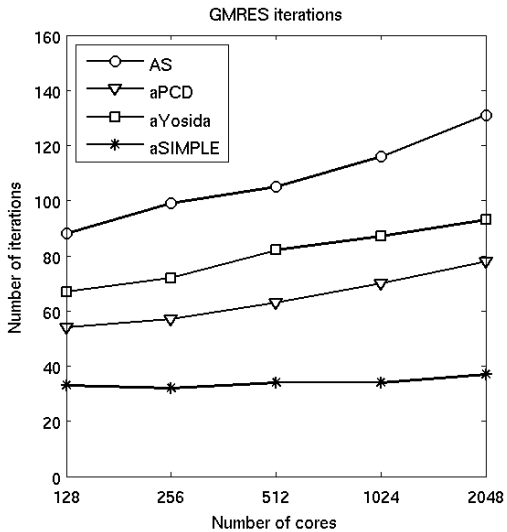
- The time to build the different preconditioner is superlinear with respect to the number of cores due to the computation of the local LU factorizations.
- When the assembly time goes below a given threshold, the communication time overcomes the computation time for aPCD, aSIMPLE, and aYosida.
- The AS preconditioner is clearly slower to build.

Number of GMRES iterations

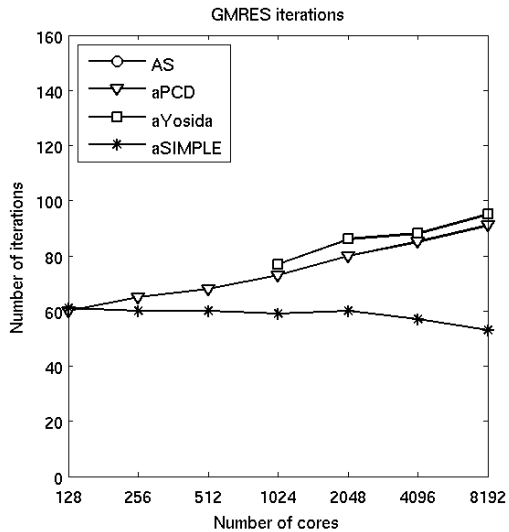
Coarse
max. 1024 cores



Medium
max. 2048 cores



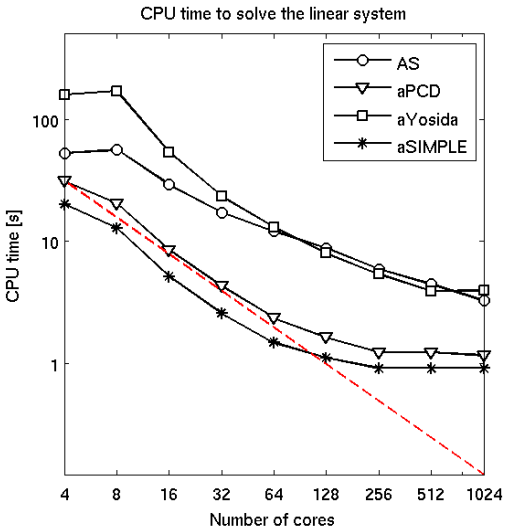
Fine
max. 8192 cores



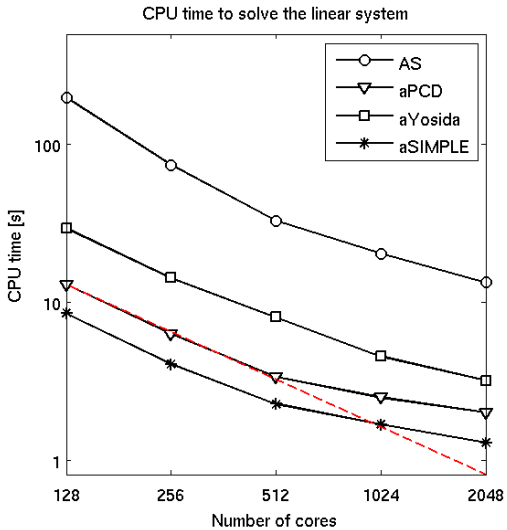
- aSIMPLE is scalable (flat curves).
- With aPCD, the iterations count is moderately increasing
- Problems of convergence are encountered with coarse mesh and aYosida.
- GMRES converges slower when the AS preconditioner is used.

Time to solve the linear system

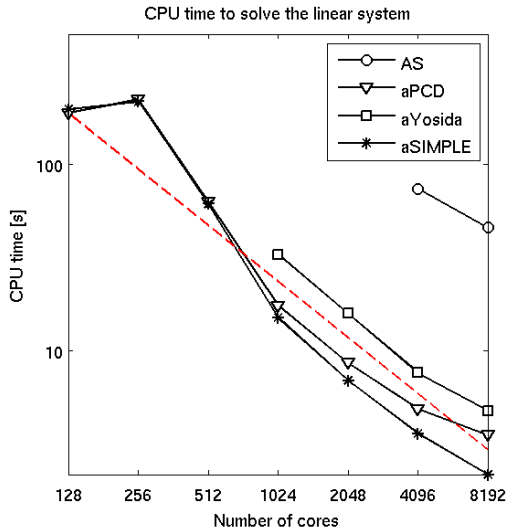
Coarse
max. 1024 cores



Medium
max. 2048 cores



Fine
max. 8192 cores



- Using the coarse mesh, the AS prec. is not strongly scalable.
- Under ~ 1 s. the communication time dominates the wall time (coarse mesh)
- Using the medium and fine meshes, the preconditioners are strongly scalable.

Variational MultiScale modeling ([Baziliev et al. 2007])

L. Dede and D. Forti

Variational formulation:

- We define $\mathcal{V} = (H_0^1(\Omega))^3$, $\mathcal{Q} = L_0^2(\Omega)$ and $\mathcal{V} = \mathcal{V} \times \mathcal{Q}$.
- Find $\mathbf{U} = \{\mathbf{u}, p\} \in (0, T] \rightarrow \mathcal{V}$ such that $\forall \mathbf{W} = \{\mathbf{w}, q\} \in \mathcal{V}$

$$B(\mathbf{W}, \mathbf{U}) = B_1(\mathbf{W}, \mathbf{U}) + B_2(\mathbf{W}, \mathbf{U}, \mathbf{U}) = (\mathbf{W}, \mathbf{F})$$

being:

$$B_1(\mathbf{W}, \mathbf{U}) = (\mathbf{w}, \mathbf{u}_t) + (\nabla^s \mathbf{w}, 2\nu \nabla^s \mathbf{u}) + (q, \nabla \cdot \mathbf{u}) - (\nabla \cdot \mathbf{w}, p)$$

$$B_2(\mathbf{W}, \mathbf{U}, \mathbf{U}) = -(\nabla \mathbf{w}, \mathbf{u} \otimes \mathbf{u})$$

$$(\mathbf{W}, \mathbf{F}) = (\mathbf{w}, \mathbf{f})$$

Variational MultiScale modeling

Scale separation

Consider a direct-sum decomposition of $\mathcal{V}$ into coarse-scale and fine-scale components:

$$\mathcal{V} = \bar{\mathcal{V}} \oplus \mathcal{V}', \tag{1}$$

being $\bar{\mathcal{V}}$ a finite dimensional subspace (e.g. Finite Elements). From (1), with the aid of a projector $\bar{\mathbb{P}}$, we consider:

$$\mathbf{U} = \bar{\mathbf{U}} + \mathbf{U}',$$

with:

$$\begin{aligned} \bar{\mathbf{U}} &= \bar{\mathbb{P}}\,\mathbf{U}, \\ \mathbf{U}' &= (I - \bar{\mathbb{P}})\,\mathbf{U}. \end{aligned}$$

Variational Multiscale modeling

The original problem becomes:

Find $\bar{\mathbf{U}} \in \bar{\mathcal{V}}$ and $\mathbf{U}' \in \mathcal{V}'$ such that:

$$\begin{aligned} &\text{(coarse-scale problem)} && B(\bar{\mathbf{W}}, \bar{\mathbf{U}} + \mathbf{U}') = (\bar{\mathbf{W}}, \mathbf{F}) && \forall \bar{\mathbf{W}} \in \bar{\mathcal{V}} \\ &\text{(fine-scale problem)} && B(\mathbf{W}', \bar{\mathbf{U}} + \mathbf{U}') = (\mathbf{W}', \mathbf{F}) && \forall \mathbf{W}' \in \mathcal{V}' \end{aligned}$$

being:

$$\begin{aligned} B(\bar{\mathbf{W}}, \bar{\mathbf{U}} + \mathbf{U}') = \; & B_1(\bar{\mathbf{W}}, \bar{\mathbf{U}}) + B_1(\bar{\mathbf{W}}, \mathbf{U}') \\ & + B_2(\bar{\mathbf{W}}, \bar{\mathbf{U}}, \bar{\mathbf{U}}) + B_2(\bar{\mathbf{W}}, \bar{\mathbf{U}}, \mathbf{U}') \\ & + B_2(\bar{\mathbf{W}}, \mathbf{U}', \bar{\mathbf{U}}) + B_2(\bar{\mathbf{W}}, \mathbf{U}', \mathbf{U}'), \end{aligned}$$

and

$$\begin{aligned} B(\mathbf{W}', \bar{\mathbf{U}} + \mathbf{U}') = \; & B_1(\mathbf{W}', \bar{\mathbf{U}}) + B_1(\mathbf{W}', \mathbf{U}') \\ & + B_2(\mathbf{W}', \bar{\mathbf{U}}, \bar{\mathbf{U}}) + B_2(\mathbf{W}', \bar{\mathbf{U}}, \mathbf{U}') \\ & + B_2(\mathbf{W}', \mathbf{U}', \bar{\mathbf{U}}) + B_2(\mathbf{W}', \mathbf{U}', \mathbf{U}') \end{aligned}$$

Variational Multiscale modeling

Approximation of the fine scale:

We aim at solving only the problem defined for the coarse-scale:

$$B(\bar{\mathbf{W}}, \bar{\mathbf{U}} + \mathbf{U}') = (\bar{\mathbf{W}}, \mathbf{F}) \quad \forall \bar{\mathbf{W}} \in \bar{\mathcal{V}},$$

which depends on the fine-scale component $\mathbf{U}'$.
Rather than solving explicitly for the fine-scale, $\mathbf{U}'$ is approximated as:

$$\mathbf{U}' \simeq F'(\bar{\mathbf{U}}, \mathrm{Res}(\bar{\mathbf{U}})),$$

where $F'(\cdot, \cdot)$ depends only on coarse scale components and $\mathrm{Res}(\bar{\mathbf{U}})$ is the coarse-scale residual.

Variational Multiscale model:

Find $\bar{\mathbf{U}} \in \bar{\mathcal{V}}$ such that[a]:

$$B(\bar{\mathbf{W}}, \bar{\mathbf{U}} + F'(\bar{\mathbf{U}}, \mathrm{Res}(\bar{\mathbf{U}}))) = (\bar{\mathbf{W}}, \mathbf{F}) \quad \forall \bar{\mathbf{W}} \in \bar{\mathcal{V}}$$

[a] We redefine $\bar{\mathbf{U}}$ as the new coarse-scale component.

Variational Multiscale modeling

Let $\bar{\mathcal{V}}$ be the finite dimensional space obtained by Finite Elements, i.e. the coarse spaces with the unknown and test variables.

Approximation of $\mathbf{U}'$

$\mathbf{U}' \simeq -\tau \text{Res}(\mathbf{U}_h)$, with:

$$\tau = \begin{pmatrix} \tau_M \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3^T & \tau_C \end{pmatrix} \quad \text{and} \quad \text{Res}(\mathbf{U}_h) = \begin{pmatrix} \mathbf{r}_M(\mathbf{u}_h, p_h) \\ r_C(\mathbf{u}_h) \end{pmatrix}.$$

The residuals $\mathbf{r}_M(\mathbf{u}_h, p_h)$ and $r_C(\mathbf{u}_h)$ read:

$$\begin{aligned} \mathbf{r}_M(\mathbf{u}_h, p_h) &= \mathbf{u}_{h\,t} + \mathbf{u}_h \cdot \nabla \mathbf{u}_h + \nabla p_h - \nu \Delta \mathbf{u}_h - \mathbf{f}, \\ r_C(\mathbf{u}_h) &= \nabla \cdot \mathbf{u}_h. \end{aligned}$$

The stabilization parameters τ_M and τ_C read (element-wise):

$$\tau_M = \left(\frac{4}{\Delta t^2} + \frac{\|\mathbf{u}_h\|^2}{h^2} + C \frac{\nu^2}{h^4} \right)^{-1/2}, \quad \tau_c = \left(\frac{\tau_M}{h^2} \right)^{-1},$$

being h the diameter of the mesh element and Δt the characteristic timestep.

Variational Multiscale modeling

Semi-discrete formulation:

Find $\mathbf{U}_h$ such that $\forall \mathbf{W}_h \in \bar{\mathcal{V}}_h$:

$$(\mathbf{w}_h, \mathbf{u}_{ht}) - (\nabla \mathbf{w}_h, \mathbf{u}_h \otimes \mathbf{u}_h) - (\nabla \cdot \mathbf{w}_h, p_h) + (\nabla^s \mathbf{w}_h, 2\nu \nabla^s \mathbf{u}_h) + (q, \nabla \cdot \mathbf{u}_h)$$

$$\text{VMS} \begin{cases} \text{SUPG} \begin{cases} +(\mathbf{u}_h \cdot \nabla \mathbf{w}_h, \tau_M \mathbf{r}_M(\mathbf{u}_h, p_h)) + (\nabla q_h, \tau_M \mathbf{r}_M(\mathbf{u}_h, p_h)) \\ +(\nabla \cdot \mathbf{w}_h, \tau_C r_C(\mathbf{u}_h)) \end{cases} \\ \text{LES} \begin{cases} +(\mathbf{u}_h \cdot (\nabla \mathbf{w}_h)^T, \tau_M \mathbf{r}_M(\mathbf{u}_h, p_h)) \\ -(\nabla \mathbf{w}_h, \tau_M \mathbf{r}_M(\mathbf{u}_h, p_h) \otimes \tau_M \mathbf{r}_M(\mathbf{u}_h, p_h)) \end{cases} \end{cases}$$

$$= (\mathbf{w}_h, \mathbf{f})$$

Implementation aspects in LifeV

Packages used: Core, ETA, Navier-Stokes

Highlights on the implementation of VMS-BDF

Find $\mathbf{U}_h$ such that $\forall \mathbf{W}_h \in \bar{\mathcal{V}}_h$:

$$
\begin{aligned}
&(\mathbf{w}_h, \frac{\alpha}{\Delta t}\mathbf{u}_h^{n+1}) - (\nabla \mathbf{w}_h, \mathbf{u}_h^{n+1} \otimes \mathbf{u}_h^{*,n+1}) \\
&\quad - (\nabla \cdot \mathbf{w}_h, p_h^{n+1}) + (\nabla^s \mathbf{w}_h, 2\nu \nabla^s \mathbf{u}_h^{n+1}) + (q_h, \nabla \cdot \mathbf{u}_h^{n+1}) \\
&\text{VMS} \quad \begin{cases} \text{SUPG} & \begin{cases} +(\mathbf{u}_h^{n+1,*} \cdot \nabla \mathbf{w}_h, \tau_M \mathbf{r}_M^{n+1}) + (\nabla q_h, \tau_M \mathbf{r}_M^{n+1}) \\ +(\nabla \cdot \mathbf{w}_h, \tau_C r_C(\mathbf{u}_h^{n+1})) \end{cases} \\ \text{LES} & \begin{cases} +(\mathbf{u}_h^{*n+1} \cdot (\nabla \mathbf{w}_h)^T, \tau_M \mathbf{r}_M^{n+1}) \\ -(\nabla \mathbf{w}_h, \tau_M \mathbf{r}_M^{*,n+1} \otimes \tau_M \mathbf{r}_M^{n}) \end{cases} \end{cases} \\
&\quad = (\mathbf{w}_h, \mathbf{f}) + (\mathbf{w}_h, \frac{\mathbf{u}_h^{n,rhs}}{\Delta t})
\end{aligned}
$$

$$(\mathbf{u}_h^{n+1,*} \cdot \nabla \mathbf{w}_h, \tau_M \mathbf{r}_M^{n+1}) =$$
$$(\mathbf{u}_h^{n+1,*} \cdot \nabla \mathbf{w}_h, \tau_M(\alpha/\Delta t\, \mathbf{u}_h^{n+1} - \underbrace{\mathbf{u}_h^{bdf}/\Delta t}_{NSrhs(0)} + \mathbf{u}_h^{n+1,*} \cdot \nabla \mathbf{u}_h^{n+1} + \underbrace{\nabla p_h^{n+1}}_{NSmatrix(0,1)} - \nu \Delta \mathbf{u}_h^{n+1}))$$

```
integrate(
elements(M_uFESpace.mesh()),
M_uFESpace.qr(),
M_ETUFE, // test  w -> phi_i
TAU_M/value(M_dt)*dot(value(M_ETUFE,uExtr)*grad(phi_i),value(M_ETUFE,uBdf))
) >> NSrhs->block(0);

integrate(
elements(M_uFESpace.mesh()),
M_uFESpace.qr(),
M_ETUFE, // test  w -> phi_i
M_ETPFE, // trial p -> phi_j
TAU_M*dot(value(M_ETUFE,uExtr)*grad(phi_i),grad(phi_j))
) >> NSmatrix->block(0,1);
```

ML preconditioner for VMS

Preconditioner

We used MultiGrid preconditioners from the ML package of Trilinos

- Default parameters setting: NSSA, nonsymmetric smoothed aggregation variant for highly nonsymmetric operators
- `max_levels` $= 3$
- `cycle_applications` $= 3$
- `pde_equations` $= 4$
- smoother: Gauss-Seidel, 3 sweeps
- aggregation: `type = Uncoupled-MIS`

Preconditioner built on the system matrix

Solver

We used the GMRES method through Belos from Trilinos, tolerance $= 10^{-8}$ (on the relative residual criterion), right-preconditioning.

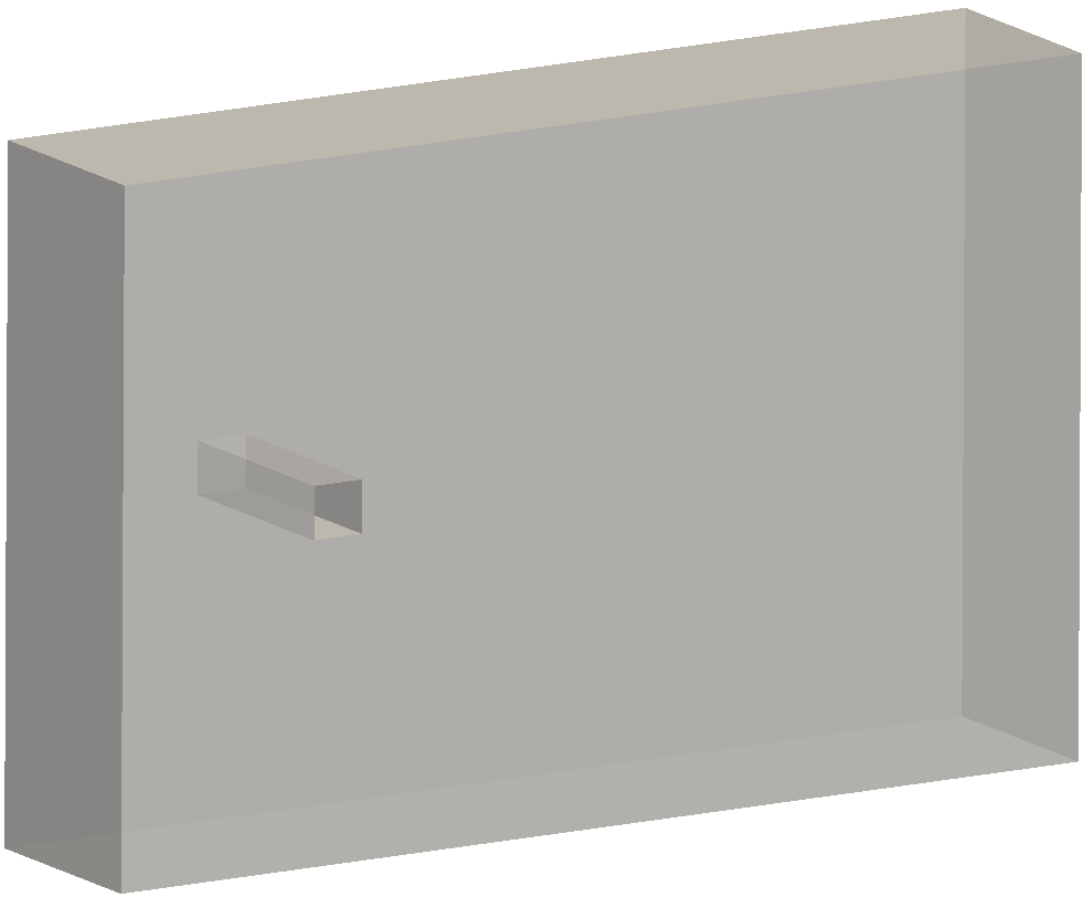
Numerical results

Benchmark problem

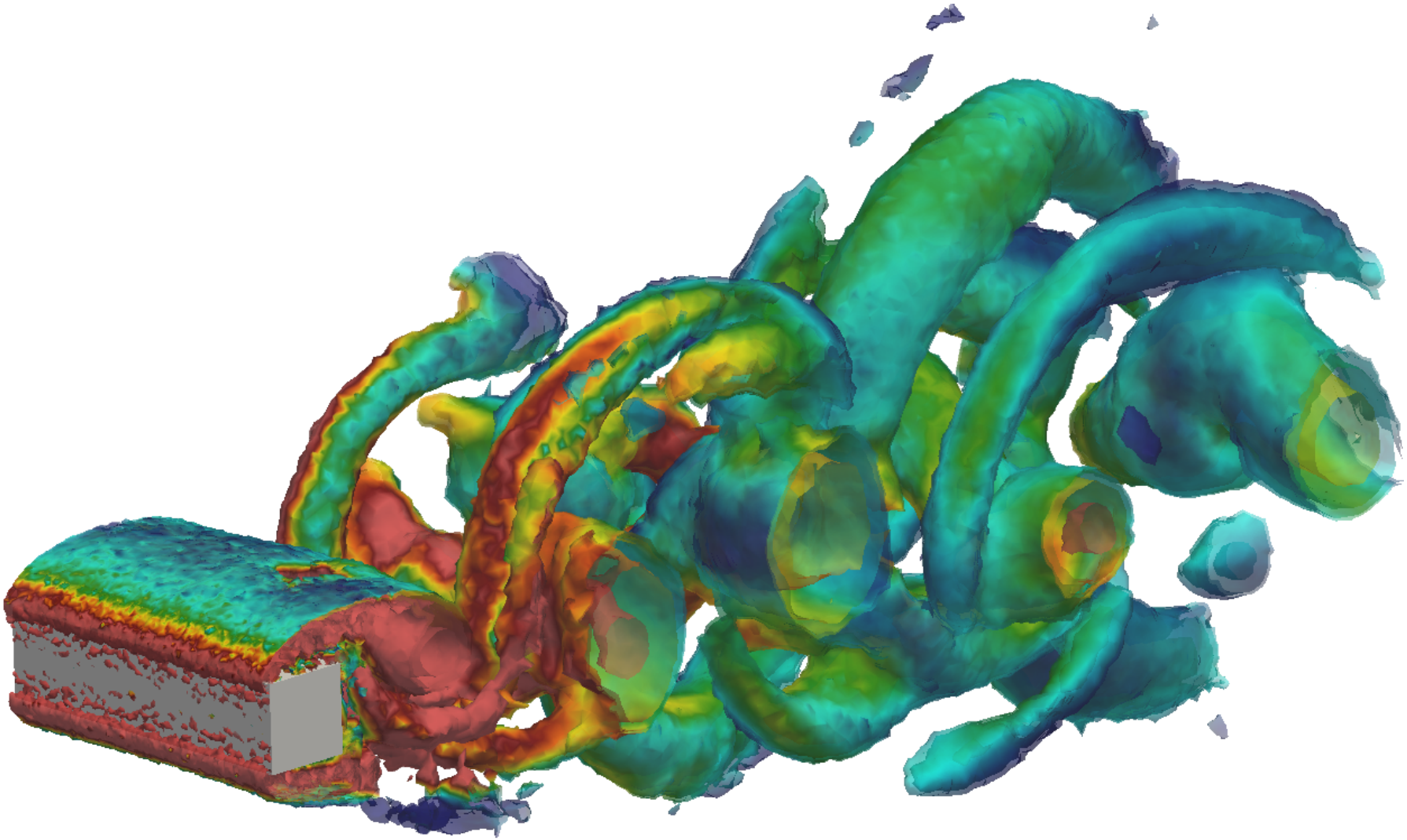
We consider the flow past a squared cylinder [Koobus and Farhat, 2004] (available experimental data). Reynolds = 22000.

dofs = 1,323,056

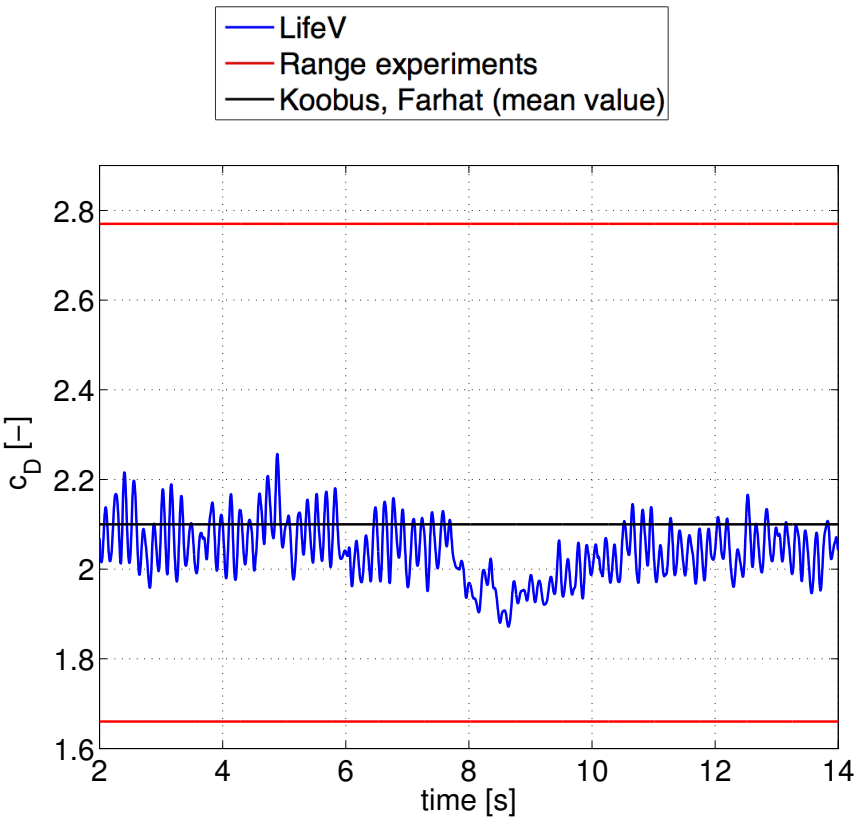
$$\begin{cases} u_{IN}(t) = \dfrac{t}{t_R} U_{IN}, \ t <= t_R \\ u_{IN}(t) = U_{IN}, \ t > t_R \end{cases}$$

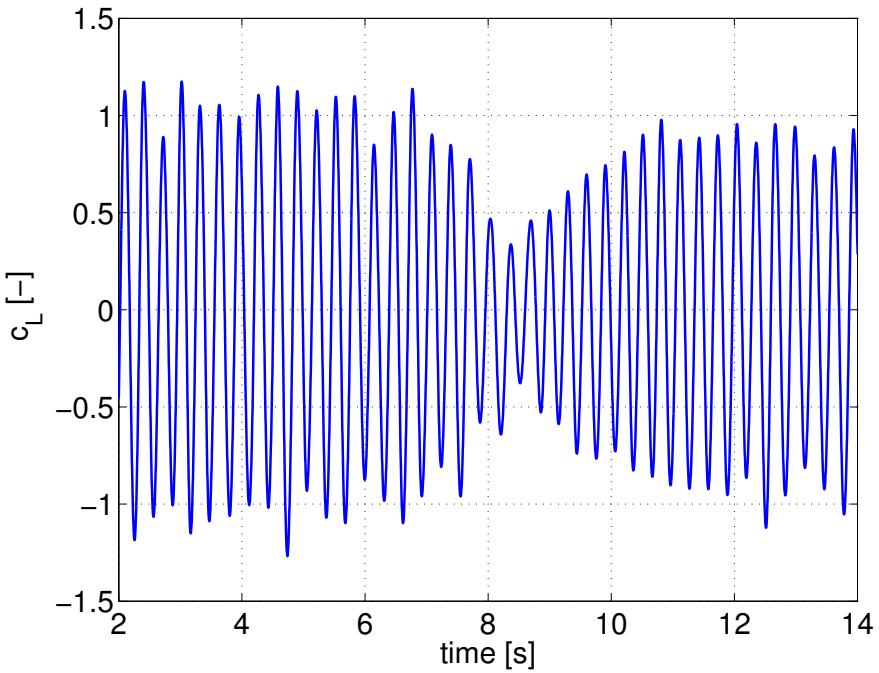


Vortex structures - Lambda 2 criterion



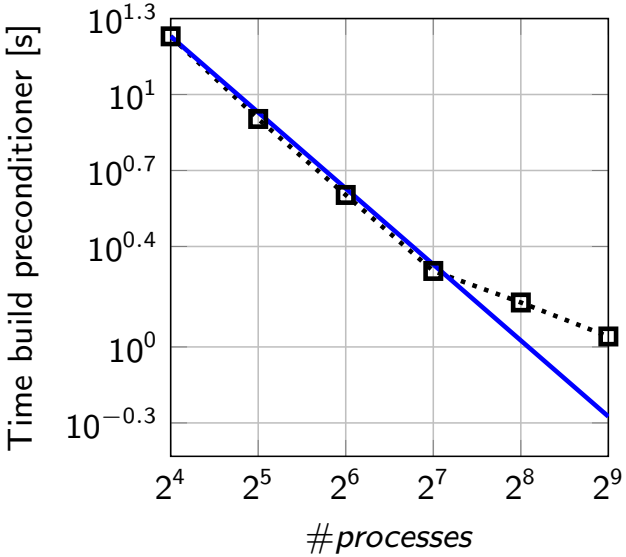
Results - Validation

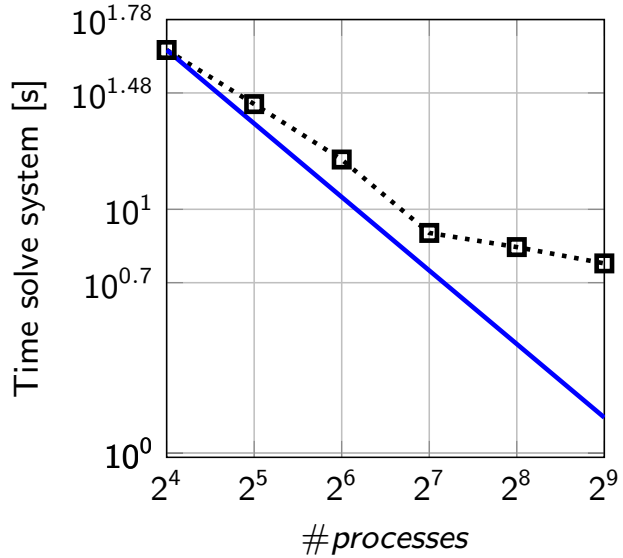


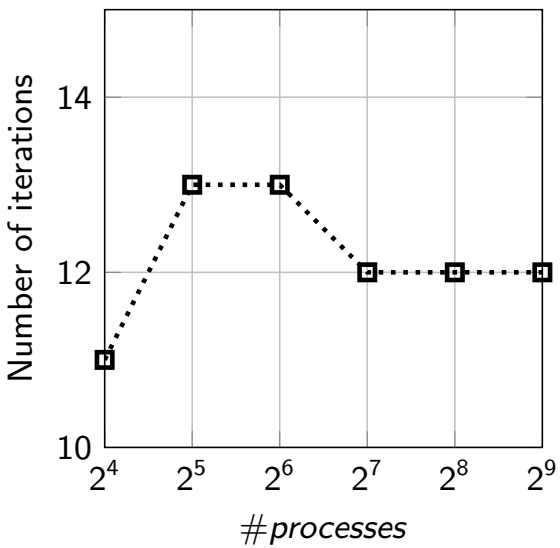


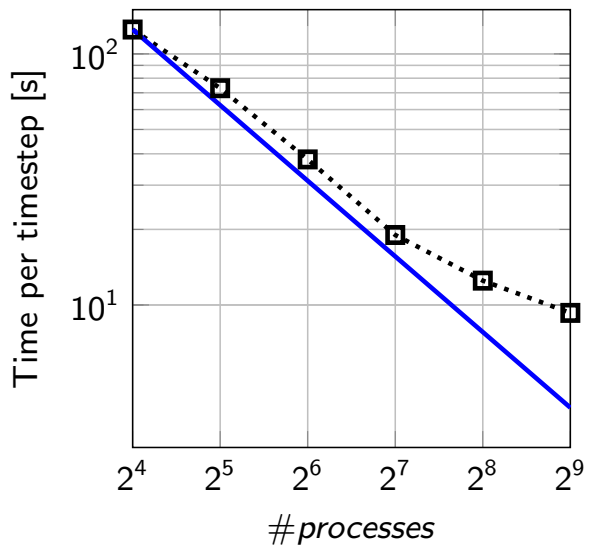
	$\bar{c_D}$	rms(c_D)	Strouhal
Experimental data [Rodi et al, 1997]	[1.66 - 2.77]	[0.1 - 0.27]	[0.07 - 0.15]
VMS-BDF	2.04	0.113	0.146
[Koobus and Farhat, 2004]	2.1	0.18	0.136

Results









Simulations were run on the cluster Bellatrix at the EPF Lausanne (each node with 2 Sandy Bridge processors running at 2.2 GHz, with 8 cores each, 32 GB of RAM, Infiniband QDR 2:1 connectivity, and GPFS filesystem).

Conclusions and next steps

Conclusions:

- Implementation of fluid solver based on VMS-BDF
- Efficient semi-implicit time-advancing scheme
- Validation towards a benchmark problem
- Thanks to Trilinos solvers and ML preconditioner achieved good parallel performances

Next steps:

- Full analysis of turbulence's scales
- Scalability study on finer meshes
- Performance analysis and comparison with a fully implicit solver (nonlinear iterations)

LifeV developers

Adrien Lefieux, Pavia; Alessandro Melani, MOX; Alessandro Veneziani, E(CM)^2; Alessio Fumagalli, MOX; Alexis Aposporidis, E(CM)^2; Antonio Cervone, MOX; Claudia Colciago, CMCS; Christian Vergara, MOX; Cristiano Malossi, CMCS; Davide Forti, CMCS; Guido Iori, MOX; Gwenol Grandperrin, CMCS; Hélène Ruffieux, EPFL; Jean Bonnemain, CMCS; Laura Cattaneo, MOX; Luca Bertagna, MOX; Luca Formaggia, MOX; Lucia Mirabella, CFM Lab; Marta D'Elia, E(CM)^2; Matteo Pozzoli, MOX; Mauro Perego, CS - Florida State Univ; Michel Kern, ESTIME - INRIA; Nur Fadel, MOX; Paolo Crosetto, CMCS; Paolo Tricerri, CMCS; Radu Popescu, CMCS; Ricardo Ruiz Baier, CMCS; Rocco Lancellotti, MOX; Samuel Quinodoz, CMCS; Simone Deparis, CMCS; Simone Pezzuto, MOX; Simone Rossi, CMCS; Simone Stangalino, MOX; Tiziano Passerini, E(CM)^2; Toni Lassila, CMCS; Tricerri Paolo, CMCS; Umberto Villa, E(CM)^2; ...