Computers & Fluids 92 (2014) 253–273

Contents lists available at ScienceDirect

Computers & Fluids

journal homepage: www.elsevier.com/locate/compfluid

# Parallel preconditioners for the unsteady Navier–Stokes equations and applications to hemodynamics simulations

Simone Deparis [a], Gwenol Grandperrin [a,*], Alfio Quarteroni [a,b]

[a] MATHICSE – Chair of Modelling and Scientific Computing (CMCS), EPFL, Station 8, CH – 1015 Lausanne, Switzerland
[b] MOX-Dipartimento di Matematica "F. Brioschi", Politecnico di Milano, I-20133 Milan, Italy

ARTICLE INFO

*Article history:*
Received 22 January 2013
Received in revised form 20 October 2013
Accepted 25 October 2013
Available online 4 November 2013

*Keywords:*
Pressure convection-diffusion preconditioner
SIMPLE preconditioner
Additive Schwarz preconditioner
Yosida preconditioner
Scalable parallel preconditioners
Finite element method
Tetrahedral unstructured mesh
Navier–Stokes equations
Hemodynamics applications

ABSTRACT

We are interested in the numerical solution of the unsteady Navier–Stokes equations on large scale parallel architectures. We consider efficient preconditioners, such as the Pressure Convection-Diffusion (PCD), the Yosida preconditioner, the SIMPLE preconditioner, and the algebraic additive Schwarz preconditioner, for the linear systems arising from finite element discretizations using tetrahedral unstructured meshes and time advancing finite difference schemes. To achieve parallel efficiency, we introduce approximate versions of these preconditioners, based on their factorizations where each factor can be either inverted exactly or using an add-hoc preconditioner. We investigate their strong scalability for both classical benchmark problems and simulations relevant to hemodynamics, using up to 8192 cores.


## 1. Introduction

In this work, we propose efficient, optimal, scalable preconditioners for the unsteady Navier–Stokes equations. We use the Finite Element Method (FEM) for the space discretization, based on tetrahedral unstructured meshes, and an implicit time discretization. To solve the nonlinear problem, we resort to a semi-explicit treatment of the convection term. We therefore have to repetitively solve a system of the type $\mathcal{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{x}$ is the unknown vector and both $\mathcal{A}$ and $\mathbf{b}$ change over the timesteps. This non-definite non-symmetric system is solved using preconditioned GMRES [1] or its variant FGMRES [2]. The efficiency of these iterative methods relies on the choice of a suitable preconditioner. A popular approach is to design a preconditioner based on an algebraic factorization of $\mathcal{A}$ which exploits its block structure, see, e.g., [3,4].

Physically based methods like SIMPLE (first introduced in [5]) or its generalizations SIMPLEC and SIMPLER [4], and Yosida methods [6,3] can be written as an approximate factorization of $\mathcal{A}$. Other methods are obtained with algebraic manipulations of the blocks of $\mathcal{A}$. This is the case of the Pressure Convection-Diffusion (PCD) preconditioner [7–9], based on an ideal preconditioner for which GMRES converges in at most two iterations [10]. Nevertheless, its implementation requires particular care.

Another option is to consider the Least-Squares Commutator (LSC) preconditioner (introduced in [11]), which can be built automatically, with higher computational cost, though. The convergence of LSC is independent of the mesh size and mildly dependent on the viscosity. A version for stabilized finite element discretizations has been introduced in [12]. More recently, the so-called Relaxed Dimensional Factorization (RDF) preconditioner has been introduced in [13] as an improvement to the Dimensional Splitting (DS) preconditioner [14]. Experimental results indicate independence of its convergence rate of the mesh size, and a mild dependence on the viscosity. The RDF deals quite well with stretched elements. These preconditioners usually offer suitable properties such as convergence independently of the domain discretization or robustness with respect to the variations on physical parameters. However, in general, the preconditioners above are not tested with more than 64 cores. Moreover, to the best of our knowledge these preconditioners are not designed and tested in the hemodynamics context.

Domain decomposition methods (DDMs) follow a different approach that is easier to parallelize; the main domain $\Omega$ is decomposed into several subdomains $\Omega_i, i = 1, \ldots, M$. Then, the

* Corresponding author.
*E-mail addresses:* simone.deparis@epfl.ch (S. Deparis), gwenol.grandperrin@epfl.ch (G. Grandperrin), alfio.quarteroni@epfl.ch (A. Quarteroni).

0045-7930/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.compfluid.2013.10.034

equations are solved iteratively on each subdomain with suitable boundary conditions depending on the neighboring domains to obtain the solution on the full domain $\Omega$. In this work, we first consider the additive Schwarz preconditioner. The locality of the algorithm (i.e., each domain is solved separately) is responsible for a raise in the number of iterations as the number of subdomains increases. The preconditioner is then improved by adding a coarse domain solver. In this work, we will form the subdomains and the coarse solver using algebraic techniques [15,16]. A quick review of DDM is available in e.g. [17] and, for a deeper understanding, the theory is presented in [18] or [19].

A further option relies on multigrid methods: instead of considering only one mesh/grid to solve a given problem, several levels of grids are used in coordination with some restriction and prolongation operators that transfer quantities from one level to another. The key idea behind multigrid is to smooth the error on the fine grids such that the remaining error can be well approximated on coarser grids, where the cost to compute the correction of such an error is cheaper. Multigrid is among the most efficient techniques for solving some specific partial differential equations (PDEs). We refer to [20] for an historical review of the development of multigrid over the last 30 years, which also includes many references, and to [21,22] for the theory of multigrid.

Our goal is to design an efficient preconditioner to solve problems relevant to hemodynamics on parallel architectures. We first consider a benchmark proposed by Ethier and Steinman [23], as well as a flow over an obstruction problem similar to the one used in [4]. We then focus to the development of largely scalable algorithms to improve cardiovascular applications. In particular, we also want to use the preconditioners developed here to improve those for complex problems such as in fluid structure interaction [24–26]. For this reason, we introduce a new test case which represents a cerebral blood vessel with an aneurysm. In the future, numerical simulations may become a tool that helps to take decisions about medical treatments in addition to the usual, e.g., radiography or magnetic resonance imaging (MRI). Among the most critical aspects of simulations in the medical field is the huge differences in the cardiovascular system from one patient to the other, the short computational time to solution to be able to make a diagnosis in case of emergency procedures, and the multiscale aspect of the models (i.e., the cardiovascular system involves from large arteries to small capillaries). We refer to [27] for an overview of techniques related to the modeling and the simulations of the cardiovascular system. The medical industry may benefit from efficient and reliable techniques to run patient specific simulations to help diagnosis in a reasonable time frame. Reducing the time to solution and increasing the problem size may also enable the validation of the models for large medical simulations. Typical medical problems may require to solve systems of millions of unknowns at each timestep. The only way to tackle them is to use parallel supercomputers, which are continuously developed and improved. It is not trivial to take advantage of such machines, though.

In what follows, we define preconditioners based on different factorizations of $\mathcal{A}$. In particular, we show that using "embedded" preconditioners to apply inverse of algebraic operators may lead to strongly scalable methods to solve the Navier–Stokes equations. We test our preconditioners using large problems and high number of processes (up to $N = 8192$). This paper focus on how state of the art preconditioners can be ported to large parallel platform. In Section 2, we describe the mathematical model used to solve the Navier–Stokes equations. In Section 3, we recall the different preconditioners that will be used in this work. In Section 4, we introduce some qualities that matter in the development of preconditioners for High Performance Computing (HPC). A new preconditioning strategy based on approximate inverses is proposed in Sections 5 and 6. Section 7 presents the scalability results obtained with a Cray XE6. Finally, some conclusions are drawn in Section 8.

## 2. Mathematical model

The Navier–Stokes equations for an incompressible viscous fluid read:

$$\frac{\partial}{\partial t}\mathbf{u} - \nu\Delta\mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u} + \nabla p = \mathbf{f} \qquad \text{in } \Omega, t > 0, \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0 \qquad \text{in } \Omega, t > 0, \tag{2}$$

$$\mathbf{u} = \mathbf{g}_D \qquad \text{on } \Gamma_D, t > 0,$$

$$\nu\frac{\partial\mathbf{u}}{\partial\mathbf{n}} - p\mathbf{n} = \mathbf{g}_N \qquad \text{on } \Gamma_N, t > 0,$$

$$\mathbf{u} = \mathbf{u}_0 \qquad \text{in } \Omega, t = 0,$$

where $\Omega$ is the fluid domain, $\Gamma_D$ and $\Gamma_N$ are the Dirichlet and Neumann parts of the boundary respectively, $\mathbf{u}$ is the fluid velocity, $p$ the pressure, $\nu$ the kinematic viscosity of the fluid, $\mathbf{f}$ the external forces, and $\mathbf{g}_D$ and $\mathbf{g}_N$ are assigned functions.

Let us introduce $H^1_D(\Omega) = \{f \in H^1(\Omega) | f = 0 \text{ on } \Gamma_D\}$, and let $V$ denote the space $\left[H^1_D(\Omega)\right]^3$, and $Q$ the space $L^2(\Omega)$. Then, multiplying by $\mathbf{v} \in V$ and integrating over $\Omega$ Eq. (1) and similarly Eq. (2) by $q \in Q$, we obtain the weak form as follows: find $(\mathbf{u}, p) \in V \times Q$ such that

$$\int_\Omega \frac{\partial\mathbf{u}}{\partial t}\mathbf{v}\,d\Omega + \int_\Omega \nu\nabla\mathbf{u} : \nabla\mathbf{v}\,d\Omega + \int_\Omega [(\mathbf{u}\cdot\nabla)\mathbf{u}]\cdot\mathbf{v}\,d\Omega - \int_\Omega p\nabla\cdot\mathbf{v}\,d\Omega$$
$$= \int_\Omega \mathbf{f}_{ext}\cdot\mathbf{v}\,d\Omega + \int_{\partial\Omega}\mathbf{g}_N\cdot\mathbf{v}\,ds \quad \forall\mathbf{v}\in V,$$
$$\int_\Omega q\nabla\cdot\mathbf{u}\,d\Omega = 0 \quad \forall q\in Q.$$

The weak form is discretized in time using a semi-implicit treatment of the nonlinear convective term and the time derivative is discretized using a first order finite difference, i.e., $(\mathbf{u}^{n+1}\cdot\nabla)\mathbf{u}^{n+1}$ is approximated by $(\mathbf{u}^n\cdot\nabla)\mathbf{u}^{n+1}$ and $\frac{\partial\mathbf{u}}{\partial t}$ by $\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t}$. The problem is then discretized in space, for instance by (stable) finite elements in space, see e.g. [17,28]. We introduce the space of finite elements:

$$X^r_h = \{\mathbf{v}_h \in C^0(\overline{\Omega}) | \mathbf{v}_{h|K} \in \mathcal{P}_r, \forall K \in \mathcal{T}_h\},$$

where $\mathcal{P}_r$ is the space of polynomials of degree less than or equal to $r$, and a regular triangulation $\mathcal{T}_h$ that forms an approximation $\Omega_h$ of $\Omega$ (see [28] for a formal definition). Then, we consider the discrete spaces for velocity and pressure denoted by $V_h = \left(X^2_h\right)^3 \cap V$, and $Q_h = X^1_h \cap Q$, respectively. Let $\mathbf{u}^n_h \in V_h$ and $p^n_h \in Q_h$ be approximations of the solutions $\mathbf{u}$ and $p$ at time $t_n$. We denote by $\boldsymbol{\varphi}_1, \ldots, \boldsymbol{\varphi}_{N_\mathbf{u}}$ the finite element basis of $V_h$ and by $\phi_1, \ldots, \phi_{N_p}$ the basis of $Q_h$. Then, we can write

$$\mathbf{u}^n_h(\mathbf{x}) = \sum_{i=1}^{N_\mathbf{u}} u^n_i\boldsymbol{\varphi}_i(\mathbf{x}), \qquad p^n_h(\mathbf{x}) = \sum_{i=1}^{N_p} p^n_i\phi_i(\mathbf{x}).$$

We approximate $(\mathbf{u}, p)$ at time $t_n$ by $(\mathbf{u}^n_h, p^n_h) \in V_h \times Q_h$ and, thanks to the Galerkin projection of $V_h \times Q_h$, we obtain the discrete system

$$\frac{1}{\Delta t}\int_{\Omega_h}\mathbf{u}^{n+1}_h\boldsymbol{\varphi}_i\,d\Omega + \int_{\Omega_h}\nabla\mathbf{u}^{n+1}_h : \nabla\boldsymbol{\varphi}_i\,d\Omega + \int_{\Omega_h}[(\mathbf{u}^n_h\cdot\nabla)\mathbf{u}^{n+1}_h]\cdot\boldsymbol{\varphi}_i\,d\Omega$$
$$-\int_{\Omega_h} p^{n+1}_h\nabla\boldsymbol{\varphi}_i\,d\Omega = \int_{\Omega_h}\mathbf{f}_{ext}(t_{n+1})\cdot\boldsymbol{\varphi}_i\,d\Omega$$
$$+\int_{\partial\Omega_h}\mathbf{g}_N(t_{n+1})\cdot\boldsymbol{\varphi}_i\,ds + \frac{1}{\Delta t}\int_{\Omega_h}\mathbf{u}^n_h,\boldsymbol{\varphi}_i\,d\Omega \qquad \forall i = 1,\ldots,N_\mathbf{u},$$
$$\int_{\Omega_h}\phi_j\nabla\mathbf{u}^{n+1}_h\,d\Omega = 0 \qquad \forall j = 1,\ldots,N_p.$$

**Table 1**
Summary of the operators required to build up each preconditioner.

| Operator | SIMPLE | Yosida | PCD |
|---|---|---|---|
| $F^{-1}$ | ✓ | ✓ | ✓ |
| $BD^{-1}B^T$ | ✓ | | |
| $A_p^{-1}$ | | | ✓ |
| $M_p^{-1}$ | | | ✓ |
| $BM_{u,\ell}^{-1}B^T$ | | ✓ | |

**Table 2**
Characteristic and non-dimensional measures; $\Delta t$ refers to the default value used.

| | | Ethier–Steinman | Obstruction | Aneurysm |
|---|---|---|---|---|
| $\nu$ | $\left[\frac{cm^2}{s}\right]$ | 0.01 | 0.035 | 0.035 |
| $L_{char}$ | (cm) | 2 | 1.5 | 0.35 |
| $U_{char}$ | $\left[\frac{cm}{s}\right]$ | 5 | 1 | 22 |
| $\Delta t$ | (s) | $10^{-3}$ | $10^{-3}$ | $10^{-3}$ |
| $\mathcal{R}e = L_{char}U_{char}/\nu$ | | 1000 | 42.9 | 220 |
| $T_{char} = L_{char}/U_{char}$ | (s) | $4.00 \times 10^{-1}$ | 1.50 | $1.59 \times 10^{-2}$ |
| $\Delta t^* = \Delta t/T_{char}$ | | $2.50 \times 10^{-3}$ | $6.67 \times 10^{-4}$ | $6.29 \times 10^{-2}$ |
| $h^*_{coarse} = h_{coarse}/L_{char}$ | | $8.65 \times 10^{-2}$ | $1.36 \times 10^{-1}$ | $1.69 \times 10^{-1}$ |
| $h^*_{fine} = h_{fine}/L_{char}$ | | $4.35 \times 10^{-2}$ | $6.80 \times 10^{-2}$ | $7.91 \times 10^{-2}$ |
| $CFL_{coarse} = \Delta t^*/h^*_{coarse}$ | | $2.89 \times 10^{-2}$ | $4.90 \times 10^{-3}$ | $3.73 \times 10^{-1}$ |
| $CFL_{fine} = \Delta t^*/h^*_{fine}$ | | $5.75 \times 10^{-2}$ | $9.80 \times 10^{-3}$ | $7.94 \times 10^{-1}$ |

**Table 3**
Monte Rosa Cray XE6 technical data.

| | |
|---|---|
| Number of service nodes | 40 |
| Number of compute nodes | 1496 |
| Number of processors per node | 2 × 16-core AMD Opteron Interlagos |
| Processors frequency | 2.1 GHz |
| Processors shared memory per node | 32 GB DDR2 |
| Peak performance | 402 Teraflop/s |
| Network | Gemini 3D torus |

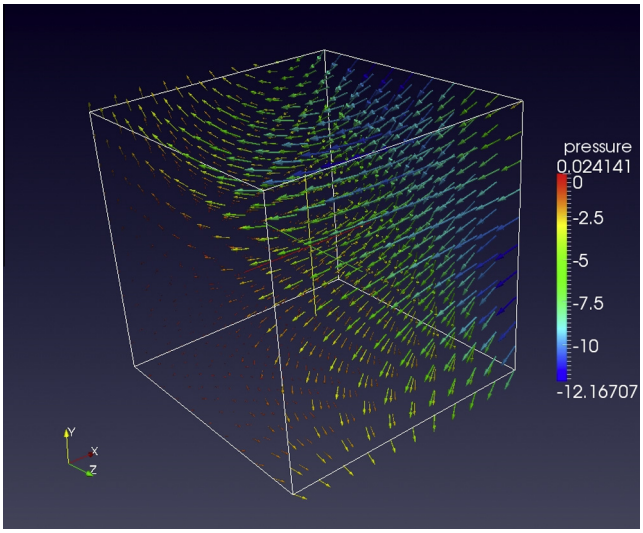



**Fig. 1.** A snapshot at time $t = 5 \times 10^{-3}$ of the solution of the Ethier–Steinman problem: pressure and velocity fields.

At each time step, these equations can be rewritten as a linear system of the form $\mathcal{A}\mathbf{x} = \mathbf{b}$ with

$$\mathcal{A} = \begin{pmatrix} F & B^T \\ -B & 0 \end{pmatrix}, \qquad \mathbf{x} = \begin{pmatrix} \mathbf{U}^{n+1} \\ \mathbf{p}^{n+1} \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} \mathbf{G} \\ \mathbf{0} \end{pmatrix}, \tag{3}$$

**Table 4**
Ethier–Steinman test case: number of Degrees of Freedom (DoF) and mesh size.

| Mesh | Velocity DoF | Pressure DoF | $h_{min}$ | $h_{average}$ | $h_{max}$ |
|---|---|---|---|---|---|
| Coarse | 206,763 | 9261 | 0.100 | 0.130 | 0.173 |
| Fine | 1,594,323 | 68,921 | 0.050 | 0.065 | 0.087 |

with $F = \frac{1}{\Delta t}M + A + C(\mathbf{U}^n)$, where $M$ is the fluid mass matrix, $A$ the stiffness matrix, and $C(\mathbf{U}^n)$ the (linearized) convection terms of the momentum equation. $B$ and $B^T$ are the discretized counterparts of the divergence operator and the gradient operator, respectively. $\mathbf{U}^{n+1} = (u_1, \ldots, u_{N_\mathbf{u}})^T$ is the vector of the velocity unknowns at time $t = t^{n+1}$ and $\mathbf{P}^{n+1} = (p_1, \ldots, p_{N_p})^T$ the one of the pressure unknowns. $\mathbf{G}$ is a known vector depending on the discretized source force, on $\mathbf{U}^n$, and on $\mathbf{f}, \mathbf{g}_D$, and $\mathbf{g}_N$. This is a simple but representative setting to test preconditioners for the Navier–Stokes equations. The considered discretization in time and space is sufficient to test the different preconditioners with the different benchmark problems. In particular, no stabilization techniques have been used in this work.

## 3. Preconditioners

In this section, we summarize how to form the core preconditioners that will be used in this work. We also comment briefly on their performance.

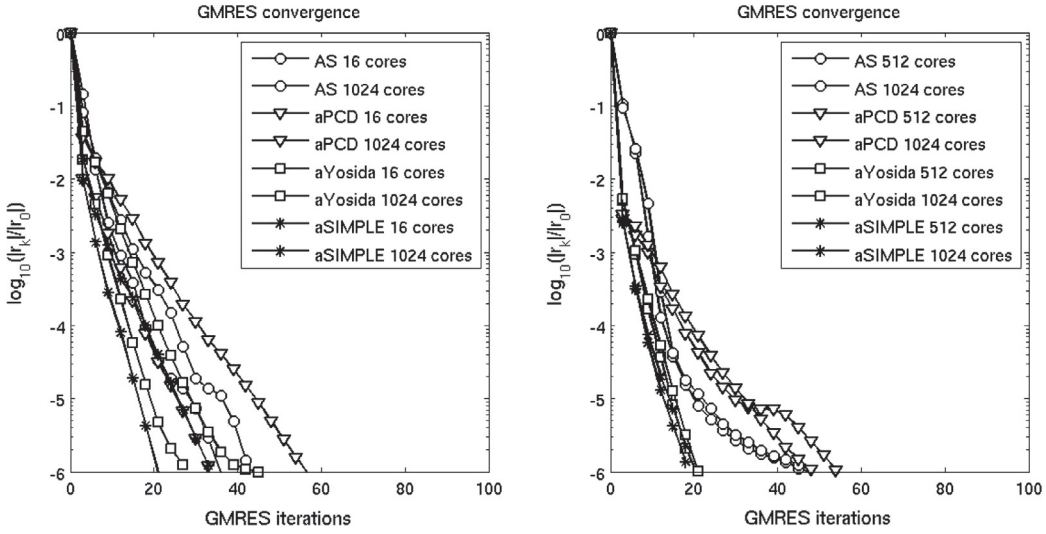
### 3.1. One and two level overlapping Additive Schwarz preconditioner

The additive Schwarz (AS) preconditioner is a popular preconditioner derived from domain decomposition theory. We recall here the basic concepts while we refer to [18,19,17] for more details and analysis. We consider a finite element mesh $\mathcal{T}_h$ of $\Omega$ and use it to assemble $\mathcal{A}\mathbf{x} = \mathbf{b}$. Let us assume that $\Omega$ is decomposed in $M$ subdomains $\Omega_i, i = 1, \ldots, M$. To guarantee convergence, we need to introduce an overlap between domains; several layer of boundary elements are therefore shared between neighbor domains. However it is also possible to defines domains and overlap based on the graph of the matrix entries; this leads to algebraic versions of the algorithms. In both cases, the partition domains, $\Omega_i, i = 1, \ldots, M$ have now common shared degrees of freedom. We denote by $N_h$ the total number of nodes of the triangulation that are internal to $\Omega$ and by $N_i$ those internal to $\Omega_i, i = 1, \ldots, M$. We now denote by $I = \{1, \ldots, N_h\}$ the set of indices of the nodes of $\Omega$ and $I_i = \{j_{i,1}, \ldots, j_{i,N_i}\}$ with $j_{i,k} \in I$, $i = 1, \ldots, M, k = 1, \ldots, N_i$, the set of nodes of $\Omega_i$. The matrix $\mathcal{A}$ contains the matrices $\mathcal{A}_i, i = 1, \ldots, M$, related to $\mathcal{A}$ by $\mathcal{A}_i = R_i \mathcal{A} R_i^T$, where $R_i$ is the restriction from $\Omega$ to $\Omega_i$ (which keeps only indices set $I_i$), and $R_i^T$ the prolongation from $\Omega_i$ to $\Omega$. The inverse of the AS preconditioner is defined as:

$$P_{AS}^{-1} = \sum_{i=1}^{M} R_i^T \mathcal{A}_i^{-1} R_i. \tag{4}$$

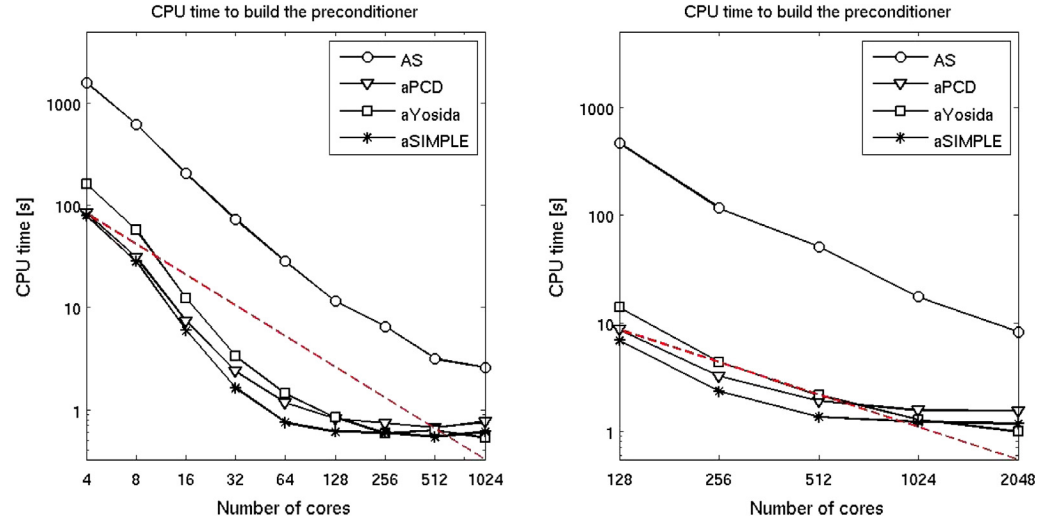
This preconditioner is not scalable. However, forming the preconditioner and applying it to a given vector is strongly scalable (provided that the overlap is not too large, otherwise the communication is a bottleneck). In our results, we are using the additive Schwarz preconditioner with an (algebraic) overlap of about three elements. This choice represents a good compromise that reduces communications and computations compared to higher overlap values while being efficient when used with GMRES. We run our code on a parallel system and we allocate one subdomain to each process. To improve the overall scalability of the AS, we also consider a 2-level additive Schwarz based on a coarse correction. In this case we add to the sum in Eq. (4) a coarse grid correction:

$$P_{2AS}^{-1} = P_{AS}^{-1} + R_0^T \mathcal{A}_0^{-1} R_0 = \sum_{i=0}^{M} R_i^T \mathcal{A}_i^{-1} R_i,$$




(a) Coarse mesh (b) Fine mesh

**Fig. 2.** Ethier–Steinman test case: convergence curves of the preconditioners for different number of cores.




(a) Coarse mesh (b) Fine mesh

**Fig. 3.** Ethier–Steinman test case: wall time needed to build the preconditioners.

where $R_0$ and $R_0^T$ are the restriction to the coarse level and the prolongation to the fine level, respectively, and $\mathcal{A}_0 = R_0 \mathcal{A} R_0^T$.

### 3.2. SIMPLE preconditioner

The Semi-Implicit Method for Pressure Linked Equations (SIMPLE) has been introduced in [5] as an iterative method which first solves the momentum equation and then updates the pressure field and the velocity field to conserve the mass by using the continuity Eq. (2). See also [29,11,4]. The method can be reinterpreted as if it were associated to a preconditioner,

$$P_{SIMPLE} = \begin{pmatrix} F & 0 \\ B & -\widetilde{S} \end{pmatrix} \begin{pmatrix} I & D^{-1}B^T \\ 0 & \alpha I \end{pmatrix}, \tag{5}$$

where $D$ is the diagonal of $F$, $\alpha \in (0, 1]$ is a parameter that damps the pressure update, and

$$\widetilde{S} = BD^{-1}B^T. \tag{6}$$

SIMPLE is efficient especially when the problem matrix is diagonally dominant. In particular, this is true in our case if $\Delta t$ is small enough. However the efficiency deteriorates as $\Delta t$ increases; in this case $\widetilde{S}$ is a poor approximation of the exact Schur complement $S$ of the approximation of the Schur complement.

### 3.3. Yosida preconditioner

Here, we consider a preconditioner inspired by the Yosida method presented, e.g., in [6,3,30,31]. This method can also be derived using the factorization:

$$\mathcal{A} = \begin{pmatrix} F & 0 \\ B & -S \end{pmatrix} \begin{pmatrix} I & F^{-1}B^T \\ 0 & I \end{pmatrix}.$$

The Schur complement is approximated using

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

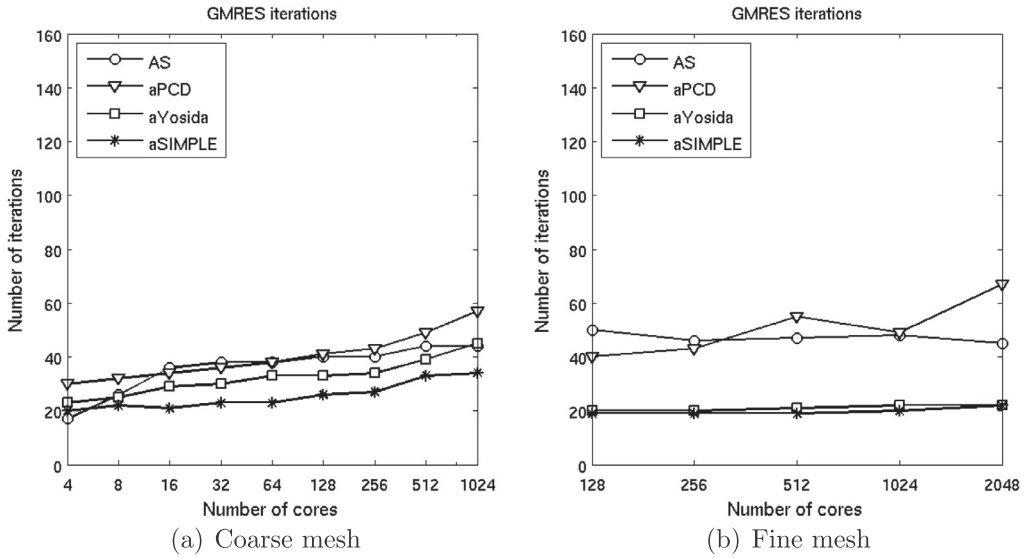



**Fig. 4.** Ethier–Steinman test case: number of GMRES iterations.

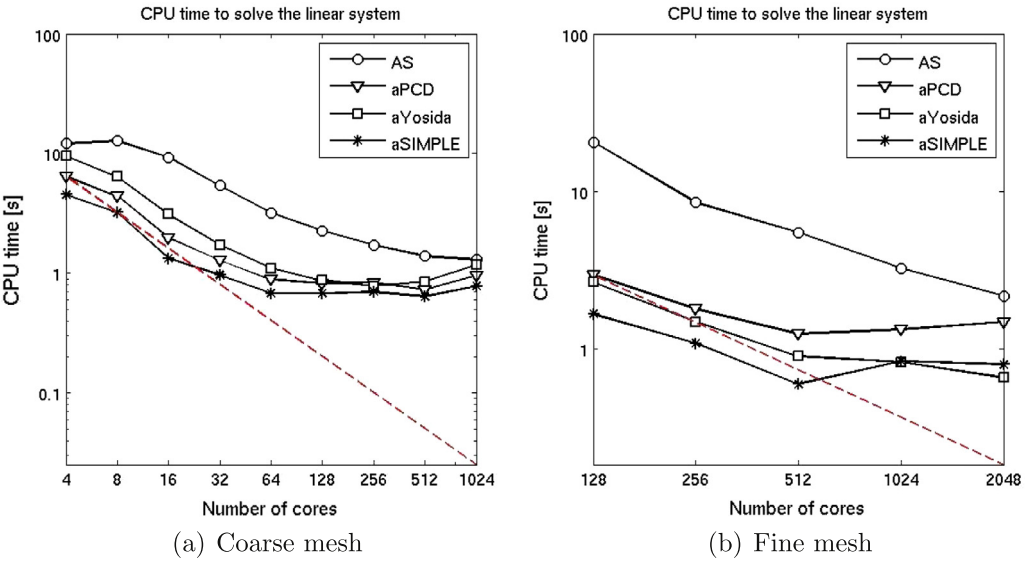



**Fig. 5.** Ethier–Steinman test case: wall time for the preconditioned iterations.

**Table 5**
Ethier–Steinman test case: sensitivity of the aPCD preconditioner w.r.t. $\nu$.

| $\nu$ | Cores | Coarse | | | Fine | | |
|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| 0.2 | 128 | 13 | 0.83 | 0.54 | 13 | 8.75 | 1.17 |
| | 256 | 14 | 0.74 | 0.54 | 13 | 3.26 | 0.78 |
| | 512 | 16 | 0.68 | 0.49 | 13 | 1.91 | 0.39 |
| | 1024 | 18 | 0.82 | 0.68 | 14 | 1.60 | 0.69 |
| 0.04 | 128 | 22 | 0.82 | 0.62 | 18 | 8.71 | 1.47 |
| | 256 | 23 | 0.74 | 0.64 | 19 | 3.26 | 0.98 |
| | 512 | 26 | 0.71 | 0.56 | 19 | 1.86 | 0.51 |
| | 1024 | 30 | 0.90 | 0.65 | 21 | 1.68 | 0.85 |
| 0.01 | 128 | 41 | 0.82 | 0.82 | 40 | 8.74 | 2.96 |
| | 256 | 43 | 0.74 | 0.84 | 43 | 3.22 | 1.80 |
| | 512 | 49 | 0.67 | 0.73 | 55 | 1.90 | 1.25 |
| | 1024 | 57 | 0.76 | 0.96 | 49 | 1.55 | 1.33 |

**Table 6**
Ethier–Steinman test case: sensitivity of the aSIMPLE preconditioner w.r.t. $\nu$.

| $\nu$ | Cores | Coarse | | | Fine | | |
|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| 0.2 | 128 | 18 | 0.63 | 0.60 | 14 | 6.91 | 1.31 |
| | 256 | 19 | 0.51 | 0.62 | 14 | 2.19 | 0.86 |
| | 512 | 23 | 0.53 | 0.56 | 15 | 1.23 | 0.50 |
| | 1024 | 24 | 0.75 | 0.64 | 15 | 1.18 | 0.78 |
| 0.04 | 128 | 24 | 0.63 | 0.67 | 15 | 6.92 | 1.40 |
| | 256 | 25 | 0.59 | 0.68 | 15 | 2.32 | 0.92 |
| | 512 | 30 | 0.53 | 0.64 | 16 | 1.38 | 0.56 |
| | 1024 | 31 | 0.77 | 0.72 | 17 | 1.27 | 0.89 |
| 0.01 | 128 | 26 | 0.61 | 0.68 | 19 | 6.90 | 1.67 |
| | 256 | 27 | 0.60 | 0.70 | 19 | 2.34 | 1.09 |
| | 512 | 33 | 0.54 | 0.64 | 19 | 1.35 | 0.60 |
| | 1024 | 34 | 0.61 | 0.78 | 20 | 1.21 | 0.84 |

**Table 7**
Ethier–Steinman test case: sensitivity of the aPCD preconditioner w.r.t. $\Delta t$.

| $\Delta t$ | Cores | Coarse | | | Fine | | |
|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| $10^{-4}$ | 128 | 20 | 0.82 | 0.61 | 21 | 8.72 | 1.67 |
| | 256 | 21 | 0.74 | 0.60 | 23 | 3.27 | 1.09 |
| | 512 | 24 | 0.69 | 0.73 | 22 | 1.89 | 0.56 |
| | 1024 | 27 | 0.76 | 0.63 | 24 | 1.58 | 0.96 |
| $10^{-3}$ | 128 | 41 | 0.82 | 0.82 | 40 | 8.74 | 2.96 |
| | 256 | 43 | 0.74 | 0.84 | 43 | 3.22 | 1.80 |
| | 512 | 49 | 0.67 | 0.73 | 55 | 1.90 | 1.25 |
| | 1024 | 57 | 0.76 | 0.96 | 49 | 1.55 | 1.33 |
| $10^{-2}$ | 128 | 292 | 0.81 | 4.96 | 446 | 8.75 | 54.04 |
| | 256 | 219 | 0.74 | 3.16 | 505 | 3.25 | 36.18 |
| | 512 | 240 | 0.72 | 2.67 | 379 | 1.90 | 13.00 |
| | 1024 | 257 | 0.95 | 3.81 | 348 | 1.49 | 8.95 |

**Table 8**
Ethier–Steinman test case: sensitivity of the aYosida preconditioner w.r.t. $\Delta t$.

| $\Delta t$ | Cores | Coarse | | | Fine | | |
|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| $10^{-4}$ | 128 | 37 | 0.84 | 0.92 | 33 | 13.96 | 4.25 |
| | 256 | 38 | 0.64 | 0.83 | 34 | 4.30 | 2.31 |
| | 512 | 41 | 0.52 | 0.88 | 35 | 2.06 | 1.32 |
| | 1024 | 43 | 0.57 | 1.20 | 35 | 1.31 | 1.21 |
| $10^{-3}$ | 128 | 33 | 0.84 | 0.87 | 20 | 13.97 | 2.67 |
| | 256 | 34 | 0.59 | 0.78 | 20 | 4.36 | 1.49 |
| | 512 | 39 | 0.63 | 0.85 | 21 | 2.14 | 0.90 |
| | 1024 | 45 | 0.53 | 1.18 | 22 | 1.28 | 0.83 |
| $10^{-2}$ | 128 | 72 | 0.84 | 1.49 | 79 | 13.95 | 10.12 |
| | 256 | 73 | 0.63 | 1.27 | 81 | 4.37 | 5.39 |
| | 512 | 85 | 0.54 | 1.62 | 85 | 2.06 | 3.10 |
| | 1024 | 97 | 0.58 | 2.20 | 89 | 1.31 | 2.44 |

where $M_{\mathbf{u}}$ denotes the mass matrix associated to the velocity shape functions. The smaller $\Delta t$ the better the approximation since as $\Delta t$ tends to zero the mass matrix $M_{\mathbf{u}}$ becomes the dominating term in $F$. To summarize the Yosida preconditioner is defined as follows:

$$P_Y = \begin{pmatrix} F & 0 \\ B & -\Delta t B M_{\mathbf{u}}^{-1} B^T \end{pmatrix} \begin{pmatrix} I & F^{-1}B^T \\ 0 & I \end{pmatrix}.$$

In practice, $M_{\mathbf{u}}^{-1}$ is replaced by the inverse of the diagonal D of $M_{\mathbf{u}}$ such that we can form explicitly $\widetilde{S}$ as in SIMPLE. Also in this case, the efficiency deteriorates when $\Delta t$ increases. A more sophisticated approach has been developed in [32] to obtain a preconditioner for time adaptive Navier–Stokes solver.

**Table 9**
Obstruction test case: number of Degrees of Freedom (DoF) and mesh size.

| Mesh | Velocity DoF | Pressure DoF | $h_{min}$ | $h_{average}$ | $h_{max}$ |
|---|---|---|---|---|---|
| Coarse | 1,304,130 | 58,579 | 0.023 | 0.093 | 0.204 |
| Fine | 9,244,263 | 401,041 | 0.014 | 0.048 | 0.102 |

### 3.4. Pressure Convection–Diffusion preconditioner

The Pressure Convection–Diffusion (PCD) [7–9] preconditioner reads

$$P_{PCD} = \begin{pmatrix} F & B^T \\ 0 & -M_p F_p^{-1} A_p \end{pmatrix}, \tag{7}$$

where $A_p = Bdiag(M_{\mathbf{u}})^{-1} B^T$ is a discrete weighted laplacian operator for the pressure space, $M_p$ is the mass matrix associated to the pressure, $F_p$ is counterpart for the pressure space of the operator $F$ (i.e., the discretized version of the operator $\mathcal{D}_t - \nu\Delta + \mathbf{u}\cdot\nabla$ for the pressure space with $\mathcal{D}_t$ being the contribution of the time discretization). $M_p F_p^{-1} A_p$ is an approximation of the Schur complement $S$ of $\mathcal{A}$. If the Schur complement was applied exactly, the resulting GMRES method would converge in at most two iterations [10,33], while the convergence rate when using $M_p F_p^{-1} A_p$ depends on the boundary conditions used to form $F_p$ [34]. We use the strategy proposed in [9]: Robin boundary conditions are imposed on the inflow boundaries while homogenous Neumann conditions are imposed on both the characteristic and outflow boundaries.

Empirical evidence indicates that the convergence rate is mildly dependent on the Reynolds number and is independent of the finite element mesh size $h$ [35,11,4].

In previous works [7,8], the algebraic operators $F^{-1}, A_p^{-1}$, and $M_p^{-1}$ were applied using GMRES iterations (i.e., a fixed number of iterations or as long as a tolerance is not reached). These inner iterations implies the use of FGMRES [2] or GMRESR [36] for solving the saddle point problem. The presence of the operator $F_p$ and the choice of the associated boundary conditions make the use of the PCD preconditioner more involved to form than the Yosida and SIMPLE preconditioners.

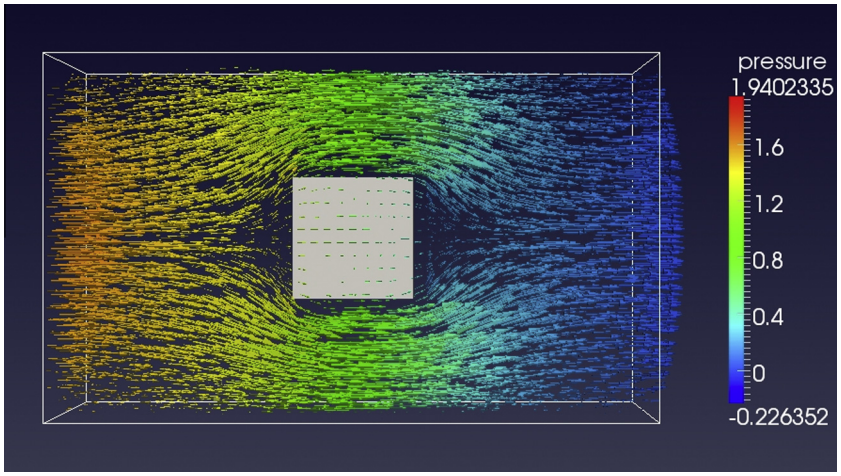
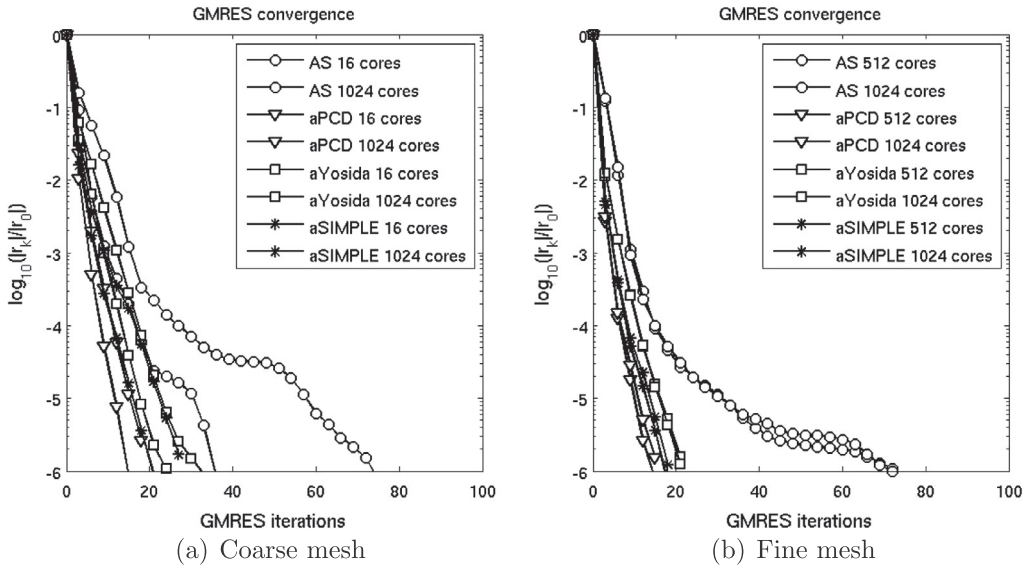
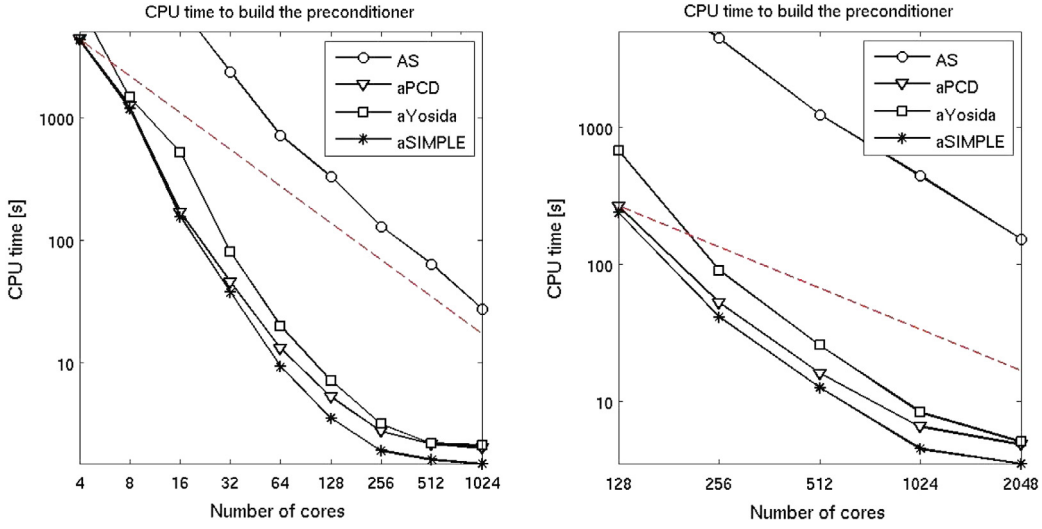



**Fig. 6.** A snapshot at time $t = 5 \times 10^{-3}$ of the solution of the obstruction problem: pressure and velocity fields.




(a) Coarse mesh (b) Fine mesh

**Fig. 7.** Obstruction test case: convergence curves of the preconditioners for different number of cores.




(a) Coarse mesh (b) Fine mesh

**Fig. 8.** Obstruction test case: wall time needed to build the preconditioners.

## 4. Preconditioners for high performance computing

Designing a preconditioner for High Performance Computing (HPC) requires special care; the preconditioner has to be efficient for the considered linear system without suffering of the parallelization required by supercomputers. We discuss in this section the must-have properties of a good preconditioner for HPC.

From now on, we use the term process to describe an atomic compute task of a parallel application. A parallel application consists of $P$ processes. A process can be either a thread or an MPI process. For a parallel application, one should evaluate the gain obtained with an increasing number of processes. A well known metric is the notion of strong scalability.

**Definition 1** (*Strong scalability*). Let $T_1$ and $T_P$ be the computational time needed to execute a task with fixed amount of computational work using one and $P$ processes, respectively. An application is said to be strongly scalable if

$$T_P = \frac{T_1}{P}.$$

As stated by Amdahl's law [37], the parts of an application that are not strongly scalable are bottlenecks for the entire application. In particular both building the preconditioner and applying the preconditioner to a vector have to be strongly scalable to guarantee that the preconditioned iterations for solving Navier–Stokes equations do not represent bottlenecks for the overall parallelization of a finite element code.

Provided that suitable resources are available, the notion of weak scalability is important in the perspective of solving bigger problems while keeping the computational time constant.

**Definition 2** (*Weak scalability*). Let $W_1$ and $W_2$ be the workload to solve a given problem using $P_1$ and $P_2$ processes, respectively. An application is said to be weakly scalable if, for any couple $(W_1, P_1)$ and $(W_2, P_2)$ such that

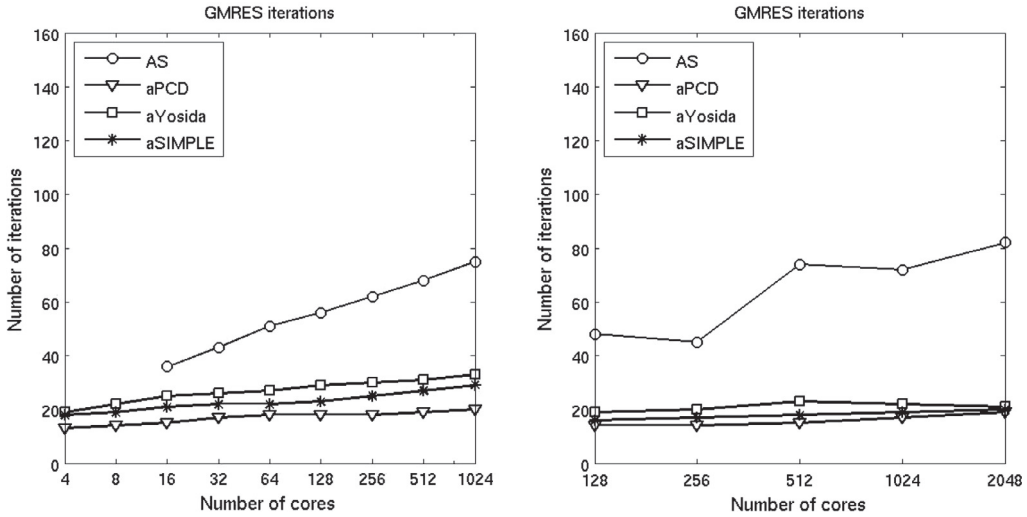



**Fig. 9.** Obstruction test case: number of GMRES iterations.

$$\frac{W_1}{P_1} = \frac{W_2}{P_2},$$

the computational time of the application is the same.

However this is not enough, ideally a preconditioner must also be scalable in the following sense:

**Definition 3** (*Preconditioner scalability*). A preconditioner $\mathcal{P}$ of $\mathcal{A}$ is said to be scalable if the rate of convergence of the iterative method used to solve the preconditioned system does not deteriorate when the number of processes grows.

This problem can be readily understood. Let us assume that each preconditioned iteration can be performed using a strongly parallel algorithm with $T_{iter,1}$ and $T_{iter,P}$ the times to perform one iteration using 1 and $P$ processes, respectively. If we denote by $N_{iter}(P)$ the number of preconditioned iterations required to solve the linear system with $P$ processes, the time to perform the preconditioned iterations using $P$ processes is given by

$$N_{iter}(P)T_{iter,P} = N_{iter}(1)\frac{T_{iter,1}}{P}.$$

Clearly, we achieve strong scalability for the whole linear solve if and only if $N_{iter}(P) = C$ a suitable constant, i.e., when $\mathcal{P}$ is scalable.

Whether it is used for HPC or not, a preconditioner should not depend on the mesh size; finer meshes should be usable without requiring additional iterations. Moreover, the computational cost should remain acceptable as the problem size increases. We summarize these expectations in the following definition:

**Definition 4** (*Preconditioner optimality*). A preconditioner is said to be optimal if for $\mathcal{A} \in \mathbb{R}^{N\times N}$

1. the number of preconditioned iterations to achieve a given error tolerance is bounded with respect to the dimension $N$ of $\mathcal{A}$;

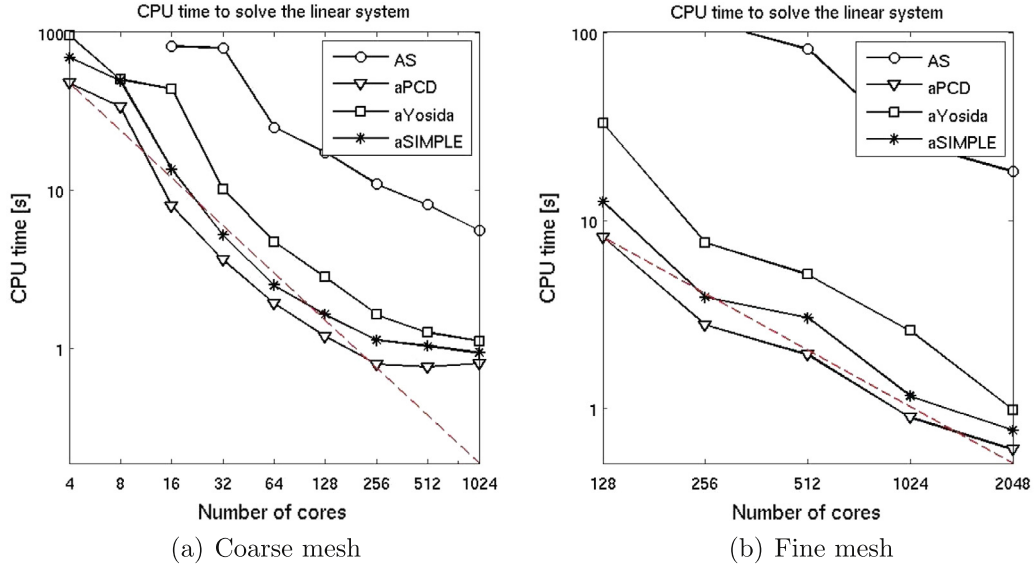



**Fig. 10.** Obstruction test case: wall time for the preconditioned iterations.

**Table 10**
Obstruction test case: sensitivity of the aPCD preconditioner w.r.t. $\nu$.

| $\nu$ | Cores | Coarse | | | Fine | | |
|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| 0.035 | 128 | 18 | 5.23 | 1.18 | 14 | 268.41 | 8.13 |
| | 256 | 18 | 2.75 | 0.79 | 14 | 52.97 | 2.77 |
| | 512 | 19 | 2.16 | 0.76 | 15 | 16.13 | 1.92 |
| | 1024 | 20 | 2.00 | 0.80 | 17 | 6.56 | 0.89 |
| 0.006 | 128 | 37 | 5.23 | 2.11 | 39 | 268.02 | 26.62 |
| | 256 | 40 | 2.74 | 1.37 | 39 | 53.19 | 8.09 |
| | 512 | 42 | 2.16 | 1.17 | 42 | 16.00 | 5.55 |
| | 1024 | 41 | 1.96 | 1.01 | 44 | 6.50 | 2.24 |
| 0.0015 | 128 | 69 | 5.21 | 3.87 | 77 | 267.57 | 62.98 |
| | 256 | 73 | 2.80 | 2.40 | 80 | 53.02 | 17.47 |
| | 512 | 77 | 2.11 | 1.84 | 88 | 16.11 | 12.20 |
| | 1024 | 78 | 2.15 | 1.65 | 93 | 6.63 | 5.19 |

**Table 11**
Obstruction test case: sensitivity of the aYosida preconditioner w.r.t. $\Delta t$.

| $\Delta t$ | Cores | Coarse | | | Fine | | |
|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| $10^{-4}$ | 128 | 36 | 7.11 | 3.43 | 29 | 681.18 | 50.35 |
| | 256 | 38 | 3.29 | 2.01 | 30 | 90.56 | 11.29 |
| | 512 | 39 | 2.21 | 1.52 | 32 | 26.30 | 7.14 |
| | 1024 | 41 | 1.80 | 1.24 | 33 | 8.46 | 4.10 |
| $10^{-3}$ | 128 | 29 | 7.12 | 2.81 | 19 | 683.25 | 32.78 |
| | 256 | 30 | 3.15 | 1.63 | 20 | 90.77 | 7.58 |
| | 512 | 31 | 2.19 | 1.25 | 23 | 25.77 | 5.15 |
| | 1024 | 33 | 2.12 | 1.11 | 22 | 8.35 | 2.58 |
| $10^{-2}$ | 128 | 50 | 7.12 | 4.74 | 38 | 683.78 | 67.03 |
| | 256 | 53 | 3.24 | 2.69 | 37 | 90.69 | 14.06 |
| | 512 | 57 | 2.50 | 2.17 | 42 | 26.16 | 9.34 |
| | 1024 | 61 | 1.79 | 1.65 | 42 | 8.48 | 5.35 |

**Table 12**
Obstruction test case: sensitivity of the aSIMPLE preconditioner w.r.t. $\Delta t$.

| $\Delta t$ | Cores | Coarse | | | Fine | | |
|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| $10^{-4}$ | 128 | 29 | 3.51 | 1.96 | 28 | 238.12 | 22.17 |
| | 256 | 29 | 1.89 | 1.26 | 28 | 41.26 | 6.49 |
| | 512 | 31 | 1.73 | 1.19 | 30 | 11.95 | 5.19 |
| | 1024 | 33 | 1.54 | 1.04 | 30 | 4.50 | 1.85 |
| $10^{-3}$ | 128 | 23 | 3.51 | 1.63 | 16 | 239.01 | 12.53 |
| | 256 | 25 | 1.90 | 1.12 | 17 | 41.42 | 3.90 |
| | 512 | 27 | 1.61 | 1.03 | 18 | 12.68 | 3.02 |
| | 1024 | 29 | 1.50 | 0.93 | 19 | 4.52 | 1.16 |
| $10^{-2}$ | 128 | 15 | 3.52 | 1.16 | 21 | 237.27 | 16.54 |
| | 256 | 16 | 1.90 | 0.86 | 21 | 41.34 | 4.91 |
| | 512 | 16 | 1.51 | 0.75 | 22 | 12.41 | 3.74 |
| | 1024 | 17 | 1.54 | 0.72 | 23 | 4.44 | 1.48 |

2. the total computational costs to assemble and to use the preconditioner increase linearly with respect to the dimension $N$ of $\mathcal{A}$.

**Definition 5** (*Preconditioner robustness*). A preconditioner is said to be robust if the convergence rate of the iterative method does not depend on the physical parameters (e.g., viscosity) that characterize the PDE.

The last property ensures that, in the Navier–Stokes case, the preconditioner handles a wide range of Reynolds numbers. For instance, for the simulation of blood flow in arterial system, the Navier–Stokes equations have to be solved for a wide range of Reynolds numbers from $\mathfrak{Re} \cong 0.003$ (capillary) to $\mathfrak{Re} \cong 4000$ (ascending aorta) [27].

## 5. Approximate preconditioners for the Navier–Stokes equations

In Section 4, we have seen that parallelism adds extra difficulties when designing good preconditioners. The additive Schwarz preconditioner perfectly illustrates this problematic situation (see Section 3.1) where as the number of domains grows (we affect one domain to each core) the convergence rate of GMRES deteriorates.

The inexact factorizations introduced in Sections 3.2–3.4 are used to define new preconditioners in Sections 5.1–5.3. The factors easily invertible are used as is, while the others are replaced by approximations based on parallel preconditioners well suited for those factors; for this reason we call them "embedded" preconditioners. The choice of good embedded preconditioners are discussed in Section 6. Embedded preconditioners have been successfully applied in the context of fluid structure interaction problems [26], where the approximations of the inverted blocks are computed as additive Schwarz preconditioners.

The factorization into block matrices improves the timing and memory usage, and allows for a modular implementation. As a matter of fact:

- preconditioning techniques for each factor can be adjusted separately;
- the condition number can be bounded in terms of the conditioning of each factor;
- if the matrix of one factor is constant, it is computed only once per simulation.

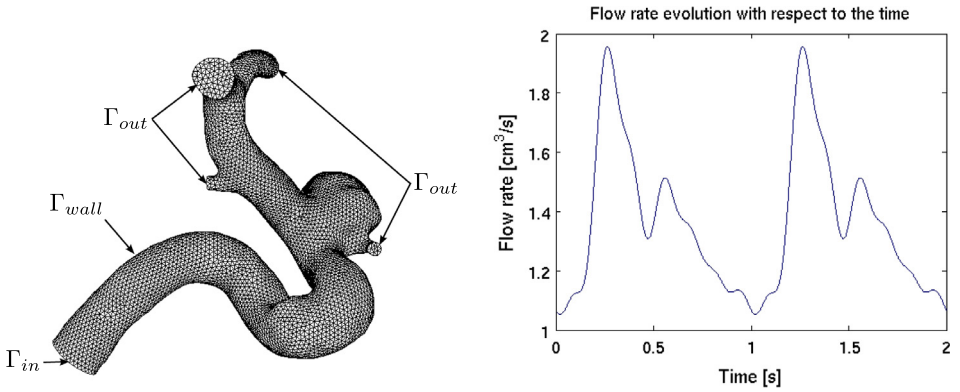
The purpose of a preconditioner is to reduce the iterations count of a linear solver; this is representative of the quality of the preconditioner to approximate the inverse of the matrix and of its ability to cluster the spectrum around one. However, we should not forget that what matters in the end is the time required to find an approximate solution to the Navier–Stokes equations. Therefore, it may be more efficient to apply less accurately a preconditioner if this lowers the time to solution. The difficulty is then to find a balance between keeping the number of iterations constant and as low as possible, while trying to lower the wall time as much as possible.

### 5.1. Approximate SIMPLE preconditioner

The approximate SIMPLE (aSIMPLE) preconditioner uses the factorization of the SIMPLE preconditioner from Section 3.2 as a starting point. Then, inverses of the algebraic operators are replaced by some suitable approximations denoted by a $\widehat{\ }$ ("hat") over the operators:

$$P^{-1}_{aSIMPLE} = \begin{pmatrix} D^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & -B^T \\ 0 & I \end{pmatrix} \begin{pmatrix} D & 0 \\ 0 & \frac{1}{\alpha} I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -\widehat{\widetilde{S}}^{-1} \end{pmatrix} \times \begin{pmatrix} I & 0 \\ -B & I \end{pmatrix} \begin{pmatrix} \widehat{F}^{-1} & 0 \\ 0 & I \end{pmatrix}.$$

Therefore, the aSIMPLE requires to build two approximations of $\widetilde{S}^{-1}$ and $F^{-1}$. Typically, these are defined starting from suitable preconditioners for $\widetilde{S}^{-1}$ and $F$, respectively.




(a) Mesh of the cerebral aneurysm (b) Flux evolution with respect to the time

**Fig. 11.** Mesh and inflow for the cerebral aneurysm.

**Table 13**
Coefficients for the flow rate function, [41].

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $a_k$ | 1.36 | −0.207 | −0.152 | 0.059 | 0.039 | 0.00286 | −0.0371 | −0.000759 |
| $b_k$ | | 0.176 | −0.0429 | −0.117 | 0.0385 | 0.0139 | 0.0166 | −0.0359 |

**Table 14**
Aneurysm test case: number of Degrees of Freedom (DoF) and mesh size.

| Mesh | Velocity DoF | Pressure DoF | $h_{min}$ | $h_{average}$ | $h_{max}$ |
|---|---|---|---|---|---|
| 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 |

### 5.2. Approximate Yosida preconditioner

The approximate Yosida preconditioner is derived from the preconditioner in Section 3.3 using the same strategy described in Section 5.1:

$$P^{-1}_{aYosida} = \begin{pmatrix} \widehat{F}^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & -B^T \\ 0 & I \end{pmatrix} \begin{pmatrix} F & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -\widehat{S}^{-1} \end{pmatrix} \times \begin{pmatrix} I & 0 \\ -B & I \end{pmatrix} \begin{pmatrix} \widehat{F}^{-1} & 0 \\ 0 & I \end{pmatrix}.$$

Note the similarity between aYosida and aSIMPLE, where in the latter $F$ is replaced by its diagonal in the first and third blocks. To reduce the computations, we replace $M_{\mathbf{u}}$ in the $\widehat{S}$ by its lumped version $M_{\mathbf{u},\ell}$ (i.e., a diagonal matrix where $\{M_{\mathbf{u},\ell}\}_{ii} = \sum_j |\{M_{\mathbf{u}}\}_{ij}|$).

### 5.3. Approximate Pressure Convection–Diffusion preconditioner

We approximate $F^{-1}, M_p^{-1}$ and $A_p^{-1}$ by suitable embedded preconditioners $\widehat{F}^{-1}, \widehat{M}_p^{-1}$ and $\widehat{A}_p^{-1}$, and we refer to this new preconditioner as Approximate Pressure Convection-Diffusion (aPCD). The aPCD preconditioner is factorized in the same way as the PCD,

$$P^{-1}_{aPCD} = \begin{pmatrix} \widehat{F}^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & -B^T \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -\widehat{A}_p^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & F_p \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & \widehat{M}_p^{-1} \end{pmatrix}. \quad (8)$$

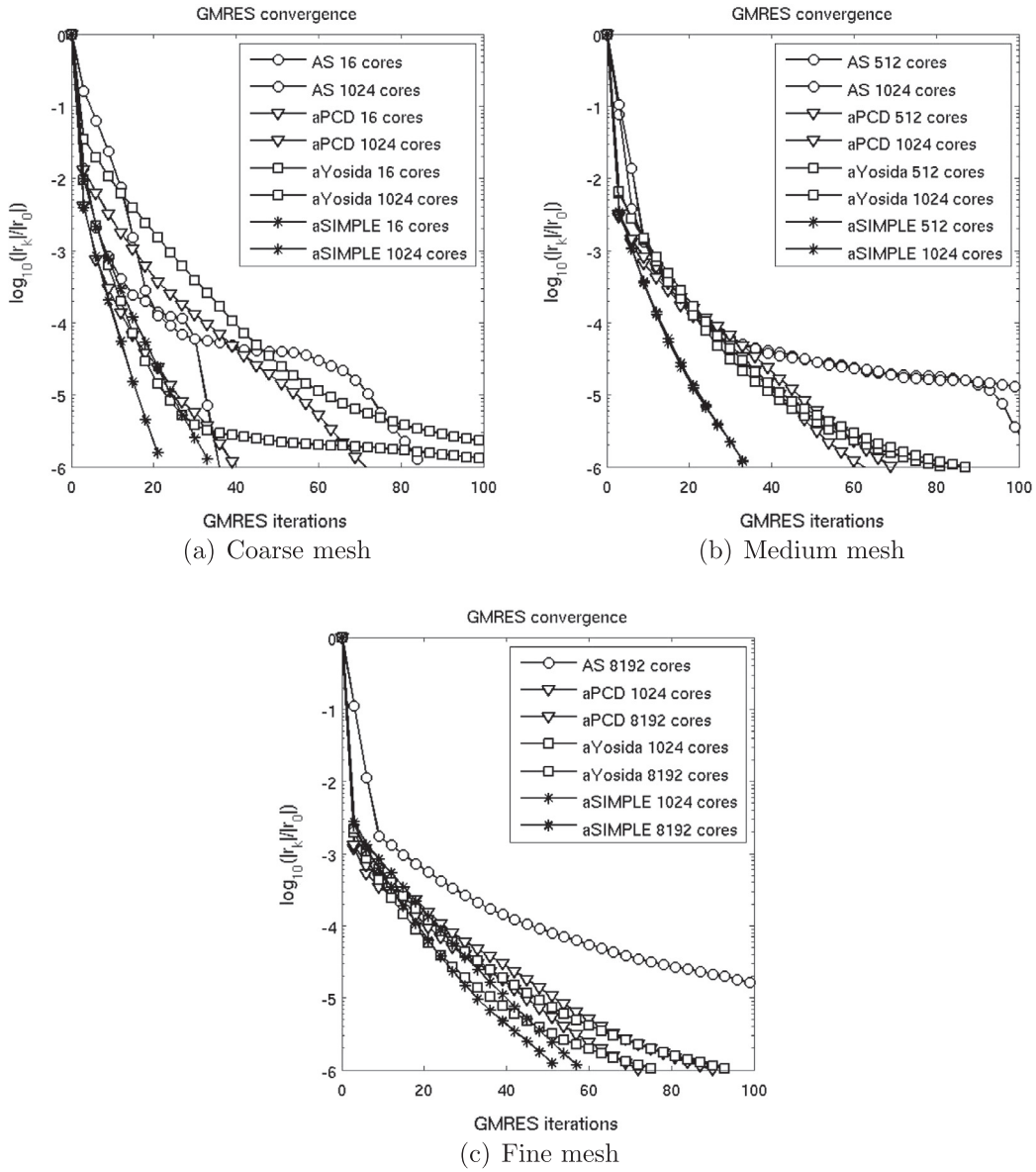
The choice of the approximations is crucial to achieve good efficiency with $P_{aPCD}$. Typically, we expect the number of iterations to be higher (the inverses of the algebraic operators are approximated by preconditioners) but on the other hand we remove the subiterations used in other papers, e.g., [4], to apply the algebraic operators inverses. Therefore, we are able to use GMRES instead of FGMRES [2] or GMRESR [36] to solve the saddle point problem, which saves some memory usage.

## 6. Strategies for applying the algebraic operators

The efficiency of the preconditioners presented in Sections 5.1,5.2,5.3 relies on that of the embedded preconditioners. More precisely, we would like those preconditioners to enjoy the properties described in Section 4. To obtain efficient parallel algorithms we need to properly balance the workload among the different collaborative parallel tasks. At the beginning of the simulation the mesh of the geometry is partitioned into subdomains using [ofortt]`METIS`[cfortt]/[ofortt]`ParMETIS`[cfortt] [15,16] and we affect one domain per core. This partition is useful to do the finite element assembly of the matrices but also play a key role in the build of the embedded preconditioners. In particular, it is used to define the subdomains in Schwarz type preconditioners.

In Table 1, we summarize the list of the algebraic operators encountered so far that require an approximation of their inverse, then in what follows, we indicate the best strategy for their application.

- $F^{-1}, BM^{-1}_{u,\ell}B^T$. We use a 2-level Schwarz preconditioner. This includes a coarse grid correction (in our case algebraic) that provides a correction from the whole domain: Additive-Schwarz (1 level) with minimal overlap (one layer of elements) is used as a pre-smoother. An LU factorization is used for the (exact) solve on each subdomain. No post-smoothing is applied. The solve at the coarse level is performed using one sweep of Gauss–Seidel method. The coarse correction is computed using one parallel task only. This is a serial bottleneck that fortunately does not affect too much the computation since the Gauss–Seidel iteration is fast to perform. Note that applying Additive Schwarz with minimal overlap (equivalent to a block Jacobi preconditioner) avoids communication. This choice has a positive impact on the cost to compute the preconditioner since




(a) Coarse mesh (b) Medium mesh (c) Fine mesh

**Fig. 12.** Aneurysm test case: convergence curves of the preconditioners for different number of cores.

reducing the overlap reduces the communication between parallel tasks. The LU factorization computational cost is in the order of $\mathcal{O}(N^3)$, where N is the size of the matrix of the local problem. For sparse matrix, however, we can expect a better complexity depending on the finite element matrix bandwidth; if $p$ and $q$ are the lower and the upper bandwidth, respectively, then if $N \gg p$ and $N \gg q$, the cost reduces to $\mathcal{O}(N)$ [38]. We explicitly compute $BM_{u,\ell}^{-1}B^T$ to construct the preconditioner.

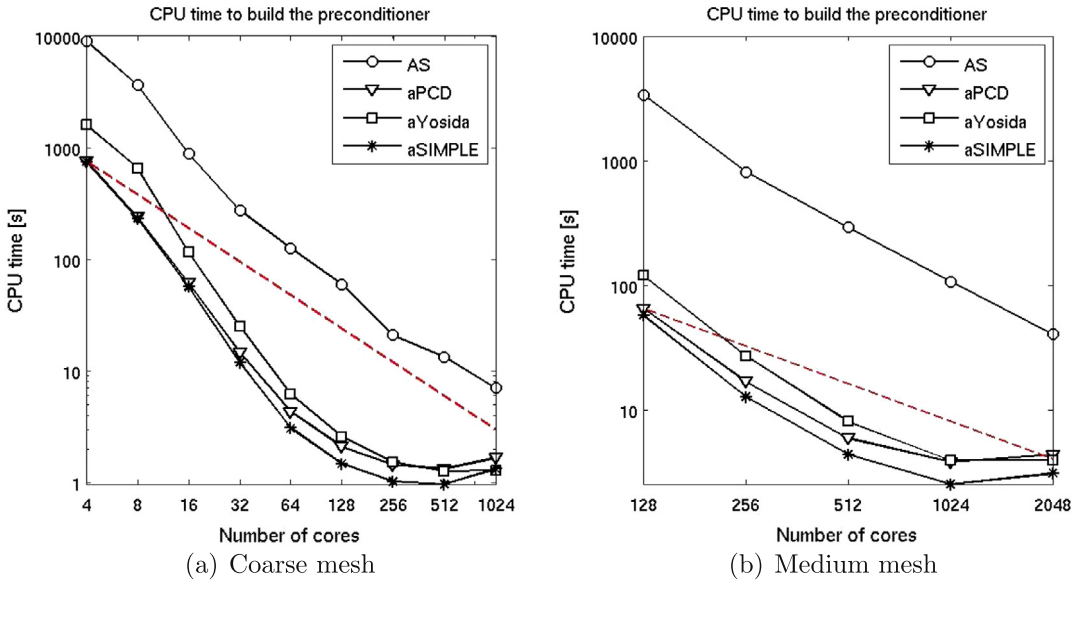
- $M_p^{-1}$ is replaced by the inverse of the lumped diagonal version of $M_p$ (this was already proposed in [4]). The approximation is fast to build and applying it is a embarrassingly parallel task.
- $A_p^{-1}, BD^{-1}B^T$. Multigrid provides good efficiency and strong scalability results for pseudo-Laplacian algebraic operators, it is therefore a natural choice for the approximation of both $A_p$ and $BD^{-1}B^T$. For our simulations we use symmetric Gauss–Seidel to perform pre smoothing only, while the solve on the coarse mesh uses an LU factorization. There is a small serial bottleneck due to the coarse solve, which is performed with one of the parallel tasks. We explicitly compute $BD^{-1}B^T$ to construct the preconditioner.

These approximations were obtained by choosing the most efficient techniques to solve each operator using up to 1024 processes, although we then tested the preconditioners using up to 8192 processes. In particular, we tested Multilevel Schwarz preconditioners as well as multigrid preconditioners.

## 7. Numerical results

We test our preconditioners in three different cases: an analythical solution on a cube [23], an obstruction problem inspired from [4], and a cerebral aneurysm. Table 2 reports the characteristic and non-dimensional quantities used for each problem. In Section 7.5, we focus on testing the weak scalability of the preconditioners on a thoracic aorta.

We have chosen the parameters to cover different Reynolds numbers. Note that $\Delta t$ is the same for the three benchmarks, but $\Delta t^*$ spans over three orders of magnitude.




(a) Coarse mesh (b) Medium mesh

(c) Fine mesh

**Fig. 13.** Aneurysm test case: wall time needed to build the preconditioners.

We solve the associated linear problem (3) at each time step with a right preconditioned GMRES method, without restart. We recompute the preconditioner at each timestep. We use the initial guess $\mathbf{x} = \mathbf{0}$ to make sure that our initial vector is far from the true finite element solution, and to guarantee a similar iterations count from one timestep to another. In practice, better choices can be made, e.g., by taking as initial guess the finite element solution $\mathbf{u}^n$ at the previous timestep. The stopping criteria is set to $10^{-6}$ and based on the residual scaled by the right hand side:

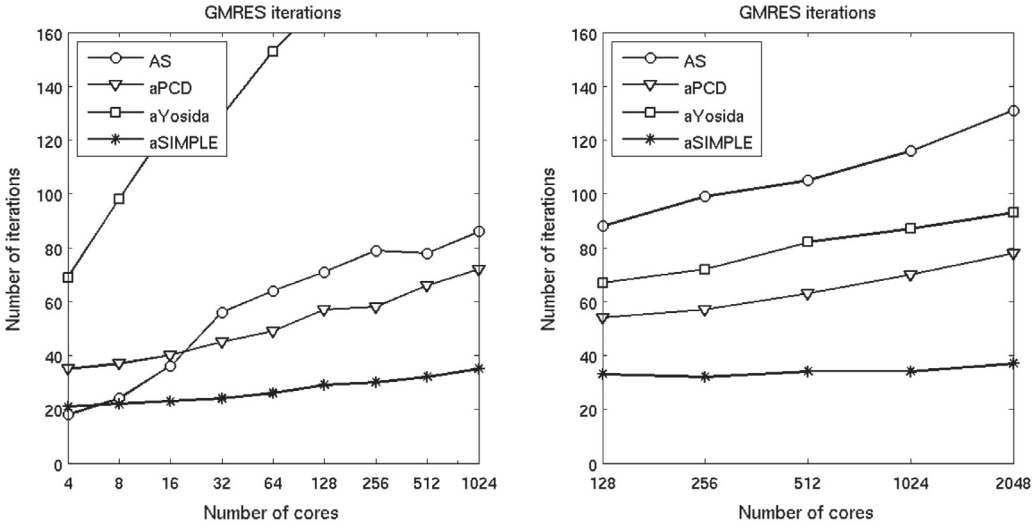
$$\|\mathbf{r}_k\|_2 \leqslant 10^{-6}\|\mathbf{b}\|_2, \tag{9}$$

where $\mathbf{r}_k$ is the residual at the $k$th iteration, and $\|\cdot\|_2$ denotes the $\ell_2$ norm of the vector of the nodal finite element solution.

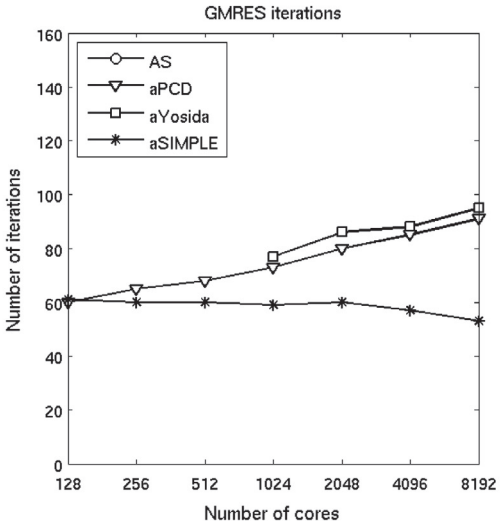
For each test case, and correspondingly to different number of processes, we show the convergence curves of the residual for the GMRES iterations; this allows to understand if the iterations are stagnating or if the use of the preconditioner remains efficient until the goal tolerance is reached. Then, we show strong scalability curves in terms of wall time, taking therefore the communication costs into account. We also show the scalability curve of the preconditioner; we expect the number of iterations to remain constant as the number of cores grows. Finally, we test the sensitivity of the preconditioners with respect to the viscosity $\nu$ and the timestep $\Delta t$; the timestep is particularly relevant for the Yosida preconditioner in which $F$ is approximated by $\frac{1}{\Delta t}M_u$.

### 7.1. Implementation

We have implemented the aPCD, aSIMPLE, and aYosida preconditioners in [ofortt]`LifeV`[cfortt] [39], a C++ finite element library under the LGPL license. This library makes intensive use of [ofortt]`Trilinos`[cfortt] [40], in particular the multilevel preconditioners via [ofortt]`ML`[cfortt] and the (1 level) additive Schwarz preconditioners via [ofortt]`Ifpack`[cfortt]. Our iterative solver (GMRES) is based on the [ofortt]`Belos`[cfortt] package. Whenever coarsening the mesh is required, aggregations are computed using [ofortt]`METIS`[cfortt]/[ofortt]`ParMETIS`[cfortt] [15,16].




(a) Coarse mesh (b) Medium mesh




(c) Fine mesh

**Fig. 14.** Aneurysm test case: number of GMRES iterations.

All the computations are carried out using Monte Rosa, a Cray XE6 supercomputer at the Swiss National Supercomputing Centre (CSCS), cf. Table 3.

### 7.2. Ethier–Steinman problem

Ethier and Steinman proposed in [23] a test case for which an analytic solution of the Navier–Stokes equations is known; taking the Navier–Stokes equations with $\rho = 1$ and $\mathbf{f} = \mathbf{0}$ and the domain $\Omega = (-1,1)^3$ (Fig. 1), the solution $(\mathbf{u}^e, p^e)$ reads:

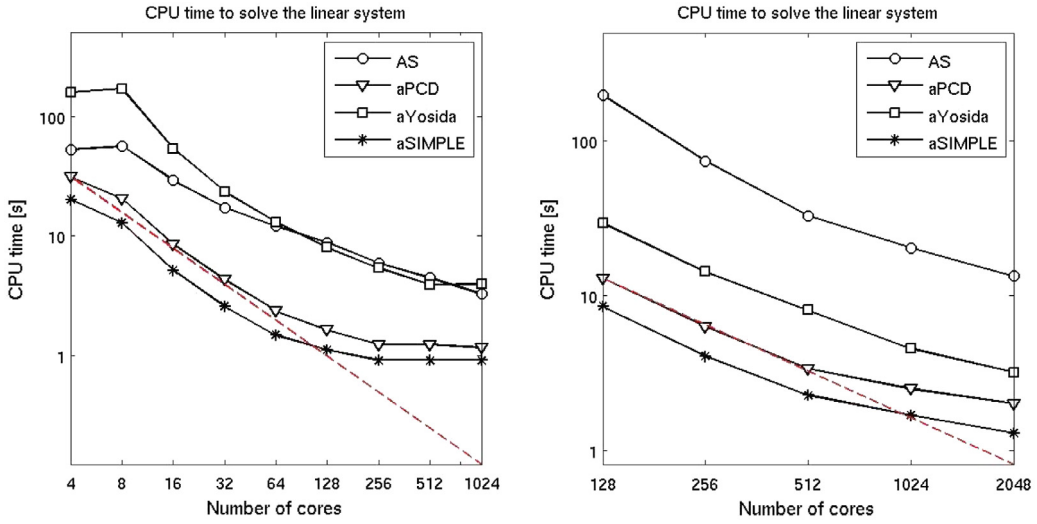
$$\mathbf{u}^e = -ae^{-vb^2t}\begin{pmatrix} e^{ax_1}\sin(ax_2+bx_3) + e^{ax_3}\cos(ax_1+bx_2) \\ e^{ax_2}\sin(ax_3+bx_1) + e^{ax_1}\cos(ax_2+bx_3) \\ e^{ax_3}\sin(ax_1+bx_2) + e^{ax_2}\cos(ax_3+bx_1) \end{pmatrix},$$

$$\begin{aligned} p^e = -\frac{a^2e^{-2vb^2t}}{2}\big(&2\sin(ax_1+bx_2)\cos(ax_3+bx_1)e^{a(x_2+x_3)} \\ &+ 2\sin(ax_2+bx_3)\cos(ax_1+bx_2)e^{a(x_1+x_3)} \\ &+ 2\sin(ax_3+bx_1)\cos(ax_2+bx_3)e^{a(x_1+x_2)} + e^{2ax_1} + e^{2ax_2} + e^{2ax_3}\big), \end{aligned}$$

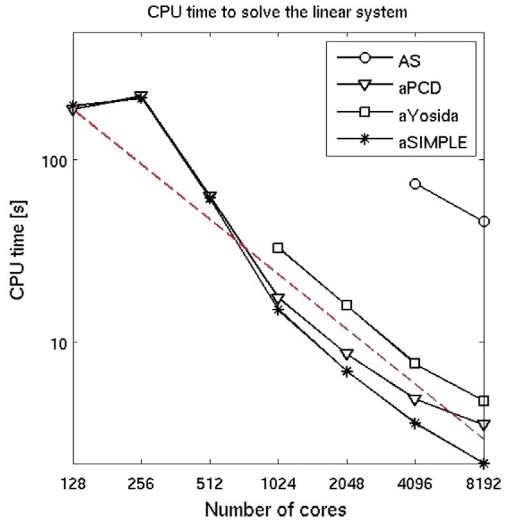
where here we take $a = \frac{\pi}{4}$ and $b = \frac{\pi}{2}$. We use this solution to generate Dirichlet boundary conditions on every faces but the front face (i.e., $y = -1$), where we impose instead a Neumann boundary condition. We use two structured meshes made of tetrahedra with the properties showed in Table 4 and a timestep $\Delta t = 10^{-3}$. The characteristic length chosen is the side of the domain.

Fig. 2 shows the residual with respect to the number of GMRES iterations using either 16 or 1028 cores for the coarse mesh, and 128 or 1028 cores for the fine mesh. We observe that both aSIMPLE and aYosida preconditioners are very efficient; indeed the convergence curve for both of them is almost a straight line. The convergence rate of the other preconditioners is slightly slower but no plateau is observed except for the AS preconditioner around $10^{-5}$ with the fine mesh. With the coarse mesh, using a higher number of cores increases the number of iterations significantly. This phenomenon is also observed for the fine mesh, although here this dependence is much milder.

Fig. 3 reports the curves for the time to build the preconditioners: following the steps described in Section 6, given the block matrices we have to form approximate inverses involved in its factorization using the core preconditioners in Section 3. The dotted line represents a perfect scaling, showing strong scalability. The




(a) Coarse mesh (b) Medium mesh




(c) Fine mesh

**Fig. 15.** Aneurysm test case: wall time for the preconditioned iterations.

**Table 15**
Aneurysm test case: sensitivity of the additive Schwarz preconditioner w.r.t. $v$.

| $v$ | Cores | Coarse | | | Medium | | | Fine | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| 0.154 | 128 | 76 | 60.36 | 9.19 | 117 | 3367.10 | 263.79 | – | – | – |
| | 256 | 85 | 20.90 | 6.31 | 131 | 814.63 | 99.27 | – | – | – |
| | 512 | 94 | 13.17 | 5.20 | 145 | 292.07 | 46.61 | – | – | – |
| | 1024 | 110 | 6.97 | 4.29 | 164 | 108.35 | 29.50 | – | – | – |
| 0.035 | 128 | 71 | 59.37 | 8.79 | 88 | 3382.21 | 197.82 | – | – | – |
| | 256 | 79 | 20.88 | 5.90 | 99 | 812.26 | 73.91 | – | – | – |
| | 512 | 78 | 13.38 | 4.44 | 105 | 292.17 | 32.85 | – | – | – |
| | 1024 | 86 | 7.00 | 3.26 | 116 | 107.55 | 20.36 | – | – | – |
| 0.0077 | 128 | 78 | 59.97 | 9.50 | 94 | 3384.52 | 210.57 | – | – | – |
| | 256 | 84 | 20.85 | 6.28 | 111 | 813.15 | 83.28 | – | – | – |
| | 512 | 94 | 13.31 | 5.37 | 107 | 294.21 | 33.50 | – | – | – |
| | 1024 | 98 | 7.04 | 3.77 | 128 | 107.77 | 22.67 | – | – | – |

super-linear behavior observed is due to the use of the local LU factorizations to build the preconditioners. Clearly, aPCD, aYosida, and aSIMPLE preconditioners are much faster to compute than the AS preconditioner. We also observe that the curve is stagnating when the computational time is about one second; in this situation, the communication probably represents a serial bottleneck.

**Table 16**
Aneurysm test case: sensitivity of the aPCD preconditioner w.r.t. $\nu$.

| $\nu$ | Cores | Coarse | | | Medium | | | Fine | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| 0.154 | 128 | 26 | 2.10 | 0.91 | 29 | 65.47 | 6.71 | 33 | 7859.95 | 101.30 |
| | 256 | 27 | 1.52 | 0.77 | 33 | 16.92 | 3.65 | 35 | 2617.54 | 115.88 |
| | 512 | 30 | 1.32 | 0.76 | 33 | 5.99 | 1.70 | 40 | 330.90 | 34.64 |
| | 1024 | 33 | 1.52 | 0.71 | 43 | 3.81 | 1.68 | 46 | 65.72 | 10.58 |
| 0.035 | 128 | 57 | 2.10 | 1.63 | 54 | 65.30 | 12.95 | 60 | 7828.87 | 188.59 |
| | 256 | 58 | 1.45 | 1.23 | 57 | 16.96 | 6.28 | 65 | 2641.00 | 222.54 |
| | 512 | 66 | 1.33 | 1.23 | 63 | 5.93 | 3.37 | 68 | 331.47 | 63.22 |
| | 1024 | 72 | 1.66 | 1.16 | 70 | 3.85 | 2.49 | 73 | 66.87 | 17.42 |
| 0.0077 | 128 | 133 | 2.12 | 4.03 | 174 | 65.08 | 48.67 | 138 | 7848.56 | 457.40 |
| | 256 | 138 | 1.43 | 2.84 | 196 | 16.95 | 26.22 | 143 | 2638.94 | 532.93 |
| | 512 | 157 | 1.34 | 2.61 | 216 | 6.06 | 15.07 | 154 | 332.19 | 176.10 |
| | 1024 | 170 | 1.63 | 2.40 | 234 | 3.89 | 9.69 | 166 | 66.56 | 45.64 |

**Table 17**
Aneurysm test case: sensitivity of the aYosida preconditioner w.r.t. $\nu$.

| $\nu$ | Cores | Coarse | | | Medium | | | Fine | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| 0.154 | 128 | 151 | 2.58 | 7.05 | 106 | 120.00 | 47.72 | – | – | – |
| | 256 | 164 | 1.51 | 4.64 | 106 | 27.15 | 21.42 | – | – | – |
| | 512 | 179 | 1.30 | 4.15 | 112 | 8.23 | 11.26 | – | – | – |
| | 1024 | 196 | 1.50 | 4.05 | 120 | 3.93 | 6.41 | 101 | 119.09 | 43.86 |
| 0.035 | 128 | 172 | 2.56 | 8.02 | 67 | 119.66 | 29.43 | – | – | – |
| | 256 | 180 | 1.53 | 5.38 | 72 | 27.14 | 14.25 | – | – | – |
| | 512 | 182 | 1.26 | 3.90 | 82 | 8.08 | 8.06 | – | – | – |
| | 1024 | 217 | 1.29 | 3.95 | 87 | 3.95 | 4.54 | 77 | 118.68 | 32.72 |
| 0.0077 | 128 | 225 | 2.59 | 11.17 | 228 | 119.94 | 113.26 | – | – | – |
| | 256 | 257 | 1.55 | 8.07 | 291 | 27.05 | 68.50 | – | – | – |
| | 512 | 300 | 1.35 | 7.31 | 357 | 8.28 | 43.90 | – | – | – |
| | 1024 | 413 | 1.28 | 8.03 | 370 | 3.98 | 24.70 | 227 | 118.82 | 110.78 |

**Table 18**
Aneurysm test case: sensitivity of the aSIMPLE preconditioner w.r.t. $\nu$.

| $\nu$ | Cores | Coarse | | | Medium | | | Fine | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| 0.154 | 128 | 22 | 1.46 | 0.91 | 44 | 57.84 | 11.47 | 89 | 7726.28 | 293.74 |
| | 256 | 22 | 1.06 | 0.77 | 42 | 12.67 | 5.24 | 85 | 2461.63 | 313.57 |
| | 512 | 23 | 0.99 | 0.76 | 41 | 4.19 | 2.70 | 85 | 299.20 | 92.40 |
| | 1024 | 25 | 1.15 | 0.67 | 40 | 2.82 | 1.89 | 83 | 57.02 | 21.86 |
| 0.035 | 128 | 29 | 1.47 | 1.11 | 33 | 57.59 | 8.50 | 61 | 7744.19 | 197.40 |
| | 256 | 30 | 1.02 | 0.91 | 32 | 12.74 | 4.05 | 60 | 2481.39 | 216.91 |
| | 512 | 32 | 0.97 | 0.91 | 34 | 4.38 | 2.26 | 60 | 300.12 | 61.30 |
| | 1024 | 35 | 1.32 | 0.91 | 34 | 2.55 | 1.68 | 59 | 56.88 | 15.01 |
| 0.0077 | 128 | 46 | 1.46 | 1.59 | 61 | 57.75 | 16.06 | 56 | 7748.71 | 181.66 |
| | 256 | 50 | 1.09 | 1.32 | 65 | 12.74 | 8.16 | 58 | 2479.67 | 209.47 |
| | 512 | 57 | 0.97 | 1.28 | 72 | 4.27 | 4.92 | 59 | 298.75 | 60.60 |
| | 1024 | 66 | 1.28 | 1.35 | 72 | 2.44 | 2.97 | 58 | 56.76 | 14.88 |

The number of iterations with respect to the number of subdomains (i.e. cores) used to solve the linear problem are reported in Fig. 4. As desired, it remains approximatively constant (flat curves). With the fine mesh, we note that both aYosida and aSIMPLE preconditioners need less iterations to converge, they converge in half the number of iterations of the others. The AS preconditioner is as scalable as the other preconditioners. However, we will see that on the other benchmarks the scalability of the AS preconditioner deteriorates. Comparing the coarse and the fine mesh results we observe only a mild dependence on $h$.

The scalability of the wall time to solve the linear system with GMRES is reported in Fig. 5. With the coarse mesh, all the preconditioners scale well up to 64 cores. However, the deterioration observed for higher cores number is due to communication; the time to solve the system is around one second. Using the fine mesh, all the preconditioners are strongly scalable, the fastest preconditioner being aSIMPLE.

We also study the sensitivity of the preconditioners with respect to $\nu$. The timing and iterations counts for aPCD, and aSIMPLE are reported in Tables 5 and 6, respectively. The AS and aYosida preconditioners results are independent of $\nu$ in the experimentation and, therefore, do not need to be reported. The results for the aPCD are independent of $h$ but depends on $\nu$ and on the number of cores. The mesh independence is to be expected since the

**Table 19**
Aneurysm test case: sensitivity of the additive Schwarz preconditioner w.r.t. $\Delta t$.

| $\Delta t$ | Cores | Coarse | | | Medium | | | Fine | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| $10^{-4}$ | 128 | 91 | 60.17 | 11.05 | 102 | 3346.12 | 228.08 | – | – | – |
| | 256 | 103 | 20.89 | 7.55 | 130 | 814.06 | 98.79 | – | – | – |
| | 512 | 108 | 13.33 | 6.01 | 133 | 292.45 | 43.03 | – | – | – |
| | 1024 | 117 | 6.99 | 4.47 | 142 | 107.41 | 24.68 | – | – | – |
| $10^{-3}$ | 128 | 71 | 59.37 | 8.79 | 88 | 3382.21 | 197.82 | – | – | – |
| | 256 | 79 | 20.88 | 5.90 | 99 | 812.26 | 73.91 | – | – | – |
| | 512 | 78 | 13.38 | 4.44 | 105 | 292.17 | 32.85 | – | – | – |
| | 1024 | 86 | 7.00 | 3.26 | 116 | 107.55 | 20.36 | – | – | – |
| $10^{-2}$ | 128 | 87 | 59.73 | 10.60 | 124 | 3379.68 | 284.55 | – | – | – |
| | 256 | 94 | 20.90 | 6.91 | 138 | 813.60 | 104.75 | – | – | – |
| | 512 | 106 | 13.38 | 5.96 | 154 | 292.60 | 48.96 | – | – | – |
| | 1024 | 125 | 6.94 | 4.67 | 176 | 107.48 | 30.66 | – | – | – |

**Table 20**
Aneurysm test case: sensitivity of the aPCD preconditioner w.r.t. $\Delta t$.

| $\Delta t$ | Cores | Coarse | | | Medium | | | Fine | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| $10^{-4}$ | 128 | 27 | 2.08 | 0.93 | 28 | 65.15 | 6.51 | 24 | 7871.28 | 73.25 |
| | 256 | 27 | 1.46 | 0.77 | 29 | 16.96 | 3.21 | 24 | 2628.06 | 77.89 |
| | 512 | 29 | 1.34 | 0.75 | 32 | 5.98 | 1.64 | 27 | 332.92 | 22.53 |
| | 1024 | 33 | 1.66 | 0.74 | 37 | 3.78 | 1.39 | 28 | 66.56 | 6.34 |
| $10^{-3}$ | 128 | 57 | 2.10 | 1.63 | 54 | 65.30 | 12.95 | 60 | 7828.87 | 188.59 |
| | 256 | 58 | 1.45 | 1.23 | 57 | 16.96 | 6.28 | 65 | 2641.00 | 222.54 |
| | 512 | 66 | 1.33 | 1.23 | 63 | 5.93 | 3.37 | 68 | 331.47 | 63.22 |
| | 1024 | 72 | 1.66 | 1.16 | 70 | 3.85 | 2.49 | 73 | 66.87 | 17.42 |
| $10^{-2}$ | 128 | 123 | 2.10 | 3.59 | 108 | 65.40 | 27.48 | 130 | 7848.83 | 430.18 |
| | 256 | 131 | 1.45 | 2.67 | 123 | 17.02 | 14.76 | 160 | 2621.14 | 602.76 |
| | 512 | 140 | 1.34 | 2.27 | 132 | 5.98 | 7.96 | 175 | 330.11 | 208.80 |
| | 1024 | 177 | 1.68 | 2.69 | 142 | 3.62 | 5.27 | 193 | 66.60 | 55.02 |

**Table 21**
Aneurysm test case: sensitivity of the aYosida preconditioner w.r.t. $\Delta t$.

| $\Delta t$ | Cores | Coarse | | | Medium | | | Fine | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| $10^{-4}$ | 128 | 28 | 2.58 | 1.39 | 22 | 120.38 | 9.51 | – | – | – |
| | 256 | 30 | 1.55 | 1.06 | 24 | 27.22 | 4.78 | – | – | – |
| | 512 | 33 | 1.27 | 0.94 | 26 | 8.24 | 2.70 | 31 | 983.80 | 70.58 |
| | 1024 | 36 | 1.28 | 0.92 | 26 | 3.93 | 1.45 | 32 | 118.06 | 13.24 |
| $10^{-3}$ | 128 | 172 | 2.56 | 8.02 | 67 | 119.66 | 29.43 | – | – | – |
| | 256 | 180 | 1.53 | 5.38 | 72 | 27.14 | 14.25 | – | – | – |
| | 512 | 182 | 1.26 | 3.90 | 82 | 8.08 | 8.06 | – | – | – |
| | 1024 | 217 | 1.29 | 3.95 | 87 | 3.95 | 4.54 | 77 | 118.68 | 32.72 |
| $10^{-2}$ | 128 | 1177 | 2.58 | 122.38 | 1151 | 119.48 | 1848.45 | – | – | – |
| | 256 | 1315 | 1.53 | 83.54 | 1222 | 27.25 | 532.72 | – | – | – |
| | 512 | 1585 | 1.23 | 72.59 | 1272 | 8.30 | 288.02 | – | – | – |
| | 1024 | 1942 | 1.31 | 70.53 | 1351 | 3.95 | 168.16 | 399 | 118.59 | 221.54 |

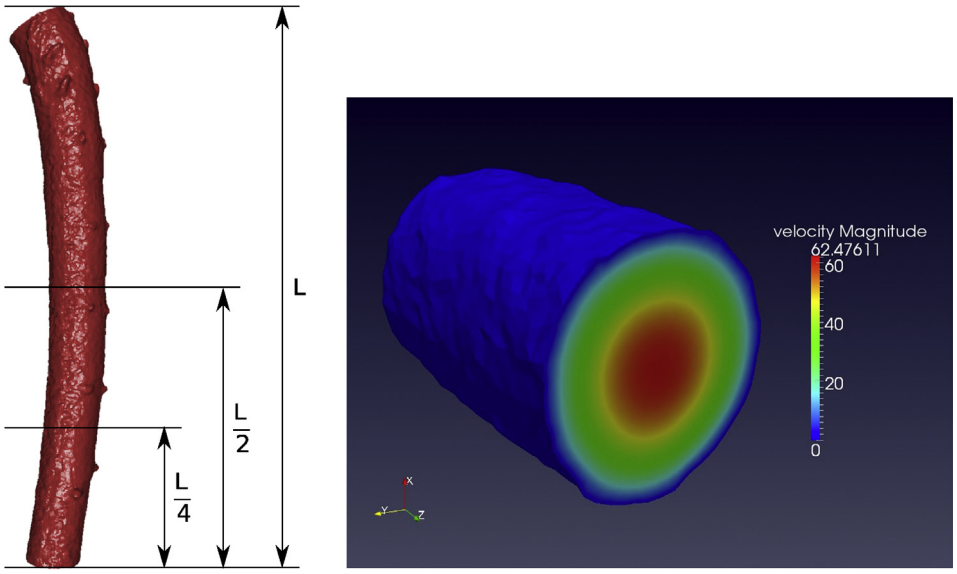
eigenvalues of $\mathcal{A}$ preconditioned with PCD can be bounded with constants, which are independent of $h$, see, e.g., [34]. For every fixed value of $\nu$, aSIMPLE is scalable for every number of cores with the fine mesh, while the number of iterations moderately depends on it with the coarse mesh; aSIMPLE mildly depends on $\nu$, though. Note also that when aSIMPLE is used, GMRES converges in less iterations when using the fine mesh. The iterations count, the time to build the AS preconditioner, and the time to solve the linear system are all independent of $\nu$.

Finally we study the impact of the choice of $\Delta t$ on the preconditioned iterations. The timing using AS or aSIMPLE is only mildly dependent on $\Delta t$, and are therefore not reported and remains scalable using different values of $\Delta t$. With the aPCD (see Table 7), $\Delta t$ strongly affects the convergence. We observe a mild dependence on $h$ for $\Delta t = 10^{-4}$ and $\Delta t = 10^{-3}$; using these values the preconditioner is almost scalable. As expected because of its definition, the aYosida preconditioner needs more iteration with a large timestep (see Table 8). We also observe that aYosida is scalable provided that the fine mesh is used.

To summarize, all the proposed preconditioners outperform the AS preconditioner. Overall, aSIMPLE turns out to have the fastest time to solution (build the preconditioner and solving the system) and the lowest number of iterations for a broad range of values of $\nu$ and $\Delta t$.

**Table 22**
Aneurysm test case: sensitivity of the aSIMPLE preconditioner w.r.t. $\Delta t$.

| $\Delta t$ | Cores | Coarse | | | Medium | | | Fine | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Iter | Setup time | Iter time | Iter | Setup time | Iter time | Iter | Setup time | Iter time |
| $10^{-4}$ | 128 | 26 | 1.47 | 1.04 | 22 | 57.36 | 5.75 | 21 | 7721.74 | 66.07 |
| | 256 | 28 | 1.04 | 0.88 | 24 | 12.74 | 3.07 | 22 | 2492.84 | 76.35 |
| | 512 | 31 | 1.00 | 0.89 | 25 | 4.27 | 1.62 | 23 | 299.85 | 21.67 |
| | 1024 | 33 | 1.11 | 0.78 | 26 | 2.45 | 1.22 | 25 | 56.60 | 6.23 |
| $10^{-3}$ | 128 | 29 | 1.47 | 1.11 | 33 | 57.59 | 8.50 | 61 | 7744.19 | 197.40 |
| | 256 | 30 | 1.02 | 0.91 | 32 | 12.74 | 4.05 | 60 | 2481.39 | 216.91 |
| | 512 | 32 | 0.97 | 0.91 | 34 | 4.38 | 2.26 | 60 | 300.12 | 61.30 |
| | 1024 | 35 | 1.32 | 0.91 | 34 | 2.55 | 1.68 | 59 | 56.88 | 15.01 |
| $10^{-2}$ | 128 | 75 | 1.37 | 2.51 | 110 | 57.74 | 30.61 | 208 | 7744.24 | 756.56 |
| | 256 | 85 | 0.91 | 1.98 | 101 | 12.72 | 13.39 | 189 | 2490.54 | 772.38 |
| | 512 | 92 | 1.11 | 2.05 | 94 | 4.34 | 6.61 | 170 | 301.34 | 221.01 |
| | 1024 | 104 | 1.30 | 1.98 | 81 | 2.55 | 3.61 | 151 | 56.68 | 43.71 |




(a) Full vessel (b) Velocity magnitude in the vessel

**Fig. 16.** Mesh and solution of the artery problem.

**Table 23**
Vessel test case: weak scalability of the preconditioners (coarse mesh).

| | 64 cores | 128 cores | 256 cores | W. Scalability | W. Scalability |
|---|---|---|---|---|---|
| Total DoF | 512,747 | 1,079,563 | 2,363,158 | 128/64 | 256/128 |
| Tot. DoF/Num. cores | 8,012 | 8,434 | 9,231 | 1.05 | 1.09 |
| *Additive Schwarz* | | | | | |
| Iter | 42 | 60 | 97 | 1.43 | 1.62 |
| Setup time | 215.60 | 255.40 | 294.97 | 1.18 | 1.15 |
| Iter time | 9.85 | 16.65 | 30.76 | 1.69 | 1.85 |
| *aPCD* | | | | | |
| Iter | 101 | 104 | 114 | 1.03 | 1.10 |
| Setup time | 3.83 | 4.96 | 6.14 | 1.30 | 1.24 |
| Iter time | 4.55 | 5.30 | 6.66 | 1.16 | 1.26 |
| *aYosida* | | | | | |
| Iter | 127 | 140 | 144 | 1.10 | 1.03 |
| Setup time | 5.09 | 7.17 | 8.42 | 1.41 | 1.17 |
| Iter time | 9.10 | 12.38 | 14.08 | 1.36 | 1.14 |
| *aSIMPLE* | | | | | |
| Iter | 48 | 50 | 54 | 1.04 | 1.08 |
| Setup time | 2.52 | 3.51 | 4.28 | 1.39 | 1.22 |
| Iter time | 2.29 | 2.87 | 3.53 | 1.25 | 1.23 |

**Table 24**
Vessel test case: weak scalability of the preconditioners (fine mesh).

| | 128 cores | 256 cores | 512 cores | W. Scalability | W. Scalability |
|---|---|---|---|---|---|
| Total DoF | 2,000,361 | 4,256,516 | 9,208,310 | 256/128 | 512/256 |
| Tot. DoF/Num. cores | 15,628 | 16,627 | 17,985 | 1.06 | 1.08 |
| *Additive Schwarz* | | | | | |
| Iter | 52 | 75 | 121 | 1.44 | 1.61 |
| Setup time | 832.72 | 989.90 | 1403.25 | 1.19 | 1.42 |
| Iter time | 29.27 | 59.52 | 139.59 | 2.03 | 2.35 |
| *aPCD* | | | | | |
| Iter | 129 | 144 | 159 | 1.12 | 1.10 |
| Setup time | 13.66 | 14.95 | 19.48 | 1.09 | 1.30 |
| Iter time | 12.35 | 15.54 | 19.12 | 1.26 | 1.23 |
| *aYosida* | | | | | |
| Iter | 235 | 249 | 278 | 1.06 | 1.12 |
| Setup time | 22.01 | 23.18 | 31.28 | 1.05 | 1.35 |
| Iter time | 43.41 | 48.01 | 62.92 | 1.11 | 1.31 |
| *aSIMPLE* | | | | | |
| Iter | 59 | 65 | 75 | 1.10 | 1.15 |
| Setup time | 10.25 | 10.78 | 14.33 | 1.05 | 1.33 |
| Iter time | 5.84 | 6.67 | 8.91 | 1.14 | 1.34 |

### 7.3. Obstruction problem

The obstruction problem is a benchmark problem already used in [4]. It consists of a cubic body immersed in a liquid which diverts the fluid flow. The computational domain is the subregion of the parallelepiped $\Omega = (-0.75, 0.75) \times (-2.5, 2.5) \times (-1.5, 1.5)$, external to the cubic body $\Omega_{cube} = (-0.5, 0.5)^3$ (Fig. 6).

This domain is discretized with tetrahedra to obtain two unstructured meshes with two refinements (see Table 9). A velocity profile is imposed at inflow while homogeneous natural boundary conditions are imposed at outflow. No-slip boundary conditions are imposed on the remaining faces (including those of the small cube). The timestep is $\Delta t = 10^{-3}$. The characteristic length represents the width of the inlet.

Fig. 7 shows the residual computed at each GMRES iteration. The aSIMPLE preconditioner needs more iterations to converge than for the Ethier–Steinman test case, but the curves are still straight. For both aYosida and aSIMPLE preconditioners, the number of iterations increases when using a larger number of cores. The situation improves when the fine mesh is used. The aPCD curves behavior depends mildly on the choice of the mesh. Small plateaux are observed for the AS preconditioner for both the coarse and the fine mesh.

Fig. 8 shows the strong scalability curves for the time to build the preconditioners. The dotted line represents a perfect scaling. The same observations as for the Ethier–Steinman hold for this problem. The stagnation of the curve for the fine mesh is noticeable. Again, the aSIMPLE preconditioner is faster to form.

For this second problem, we see that the aPCD, aSIMPLE, and aYosida have a scalable behavior (i.e., the iteration count is independent of the number of cores) up to the number of cores shown taken into account, i.e., 1024 and 2048, respectively. Moreover, their iterations count is lower than for the AS preconditioner. All the cases considered confirm that the AS preconditioner is not scalable. The aPCD preconditioner is the one with the lowest iterations count (see Fig. 9).

On both meshes, all the preconditioners but the AS are strongly scalable for solving the linear system, as shown in Fig. 10. On the fine mesh, this occurs only after that the computational time is below one second and dominated by the communications.

We consider now the sensitivity of the preconditioners with respect to $v$. When aSIMPLE and aYosida are considered, the number of iterations to solve the system depends mildly on $v$ and the mesh size $h$ and is independent of the number of cores. Those of *AS* depends mildly on $h$ and $v$, but depends on $h$. When using the aPCD (see Table 10) preconditioner the GMRES convergence depends on the value of $v$. Moreover, in every case, the iteration count is mildly dependent on the number of cores, i.e., the preconditioner is practically scalable. The results depend only mildly on $h$.

We now discuss the results for different values of $\Delta t$. When using aPCD, the preconditioned iterations converge almost independently of $\Delta t$ (as long as $\Delta t$ is less than or equal to $10^{-3}$ s), of the number of cores, and of $h$. The convergence rate for the AS preconditioner is mildly affected by $\Delta t$ with both the coarse and the fine mesh. Its efficiency depends strongly on the number of cores, and mildly on the space discretization step $h$. Those results are only affected by $\Delta t$, and we therefore do not report the full results for AS and aPCD. Using the aYosida preconditioner, the convergence rates of GMRES is mildly dependent on $\Delta t$ as shown in Table 11. However the dependence on $\Delta t$ becomes moderate with the fine mesh. With this mesh, the convergence rate is almost independent of the number of cores. aSIMPLE efficiency depends moderately on $\Delta t$, but is independent of $h$ and on the number of cores, see Fig. 12. (See Table 12).

All in all, aSIMPLE makes the computations faster than the other preconditioners, while AS is at least one order of magnitude slower. However, for $\Delta t = 10^{-4}, 10^{-3}$ s. and with aPCD, GMRES converges in less iterations than aSIMPLE.

### 7.4. Flow in a cerebral aneurysm

We consider now the simulation of blood flow in a cerebral aneurysm, that is a localized blood-filled deformation in a blood vessel wall. The computational domain $\Omega$ represents an artery where an aneurysm has developed (see Fig. 11a). The diameter of the inlet $\Gamma_{in}$ measures 0.35 cm and is chosen as characteristic length.

We carried out our computations on a coarse, a medium, and a fine unstructured mesh of tetrahedra[1]; details about the three meshes are reported in Table 14. In this section, due to the amount of memory and time resources required by aYosida and AS preconditioners, some results are not available (as indicated by a "-" in the table). The particular geometry affects the domain decomposition; indeed, automatic partitioning of the mesh tends to produce subdomains that are portions of the vessel. In this configuration,

[1] The mesh of the aneurysm is available on http://www.lifev.org in three refinement versions.

only a given amount of data proportional to the square of the radius of the vessel is sent via MPI communications between the different domains. Fig. 11a shows the mesh used to perform the simulations, for a timestep $\Delta t = 10^{-3}$.

Blood is modeled by the Navier–Stokes equations and has a density $\rho = 1\left[\frac{g}{cm^3}\right]$ and a viscosity $\mu = 0.035\left[\frac{cm^2}{s}\right]$. An approximation of the flow rate at the inlet $\Gamma_{in}$ has been provided by [41] as

$$\varphi(t) = a_0 + \sum_{k=1}^{7} a_k \cos\left(\frac{2\pi k t}{T}\right) + b_k \sin\left(\frac{2\pi k t}{T}\right), \tag{10}$$

where $T$ denotes the period of the cardiac cycle, and $a_k$ and the $b_k$ are given in Table 13 (the coefficients have been scaled to correspond to our geometry). The function $\varphi(t)$ is presented in Fig. 11b. For our computations we take $T = 1$ and the flow rate is imposed by a flat inlet profile on $\Gamma_{in}$. Homogenous Neumann boundary conditions are imposed on $\Gamma_{out}$, and no-slip boundary conditions are imposed on the vessel wall $\Gamma_{wall}$.

We carry out a strong scalability test with up to 8192 cores with the finest mesh. Fig. 12 shows the residual with respect to the number of GMRES iterations. With aSIMPLE the convergence of GMRES is the fastest. With AS, plateaux are observed with the different meshes. Using aPCD, the convergence is almost mesh independent and quite fast. Finally, using aYosida, we observe a slow convergence with the coarse mesh, while for the two finer meshes, it is similar to that of aPCD.

The strong scalability curves for the time to build the preconditioners shown in Fig. 13 are stagnating when the time goes below ten seconds. For this real case problem, the wall time required by AS preconditioner is one order of magnitude larger than that of aPCD, aSIMPLE, and aYosida preconditioners.

For this benchmark, we observe in Fig. 14 that the number of iterations for the AS preconditioner grows dramatically with the coarse and the medium mesh. This is consistent with what we observed about the reduction of the plateau in Fig. 12. Using the aPCD, the number of iterations is moderately increasing for the medium mesh and the fine mesh. Using the coarse mesh and aYosida, the iterations count considerably increases with the number of processes. When using both the medium and the fine meshes, the number of iterations is similar to the one using aPCD. Finally, the aSIMPLE preconditioner clearly outperforms the other preconditioners since the iterations count remains almost constant.

The results on strong scalability for solving the linear system are presented in Fig. 15. As expected, the AS preconditioner is neither optimal nor robust, and has to be avoided in this context. With the coarse mesh, the aYosida preconditioner behaves as the AS preconditioner. In all the cases, the time to perform the preconditioned iterations is much higher than the time required by the aPCD, aSIMPLE or aYosida preconditioners with the medium and fine meshes. With the coarse mesh, we report good scalability up to 128–256 cores. Here, the communication time seems to represent the bottleneck. Using the medium size mesh, we have excellent strong scalability up to 1024 cores. Finally, using the fine mesh, strong scalability is observed up to 8192 cores. We did not use more cores since the forming operation does not scale beyond 4096 cores.

We now consider different values for $\nu$ to investigate the robustness of the different preconditioners. The results are reported in Tables 15–18. With the aPCD, the convergence depends on $\nu$ and moderately on the number of cores. However, as in the previous problems, we observe $h$ independence. The aYosida preconditioner depends on $h$, on $\nu$, and it is not scalable. aSIMPLE depends mildly on $h$, and on $\nu$. However, on most of the tested configurations, aSIMPLE is scalable and is the most efficient.

We analyze now the dependence on $\Delta t$ for the aneurysm problem using the results reported in Tables 19–22. The results with aPCD preconditioner also depend on $\Delta t$, on the number of cores, and moderately on the mesh size $h$. The aYosida is strongly affected by $\Delta t$. With $\Delta t$ small enough, aYosida behaves independently of $h$ and moderately of the number of cores. With $\Delta t = 10^{-4}$ and $\Delta t = 10^{-3}$, aSIMPLE is scalable**, while with** $\Delta t = 10^{-2}$, its efficiency deteriorates. The convergence rate using the AS preconditioner depends on $h$, moderately on $\Delta t$, and on the number of cores.

On this benchmark, both aPCD and aSIMPLE preconditioners are better than the others; more specifically aSIMPLE offers a faster time to solution when accounting for both the preconditioner assembly and the solution of the linear system.

### 7.5. Weak scalability study with a flow in the thoracic aorta

For the sake of completeness, we also want to address the issue of the weak scalability of the preconditioners. Suppose that we want to consider a section of an artery twice as long, and correspondingly we double the number of resources available, i.e. the number of cores: can the numerical solution be computed with the same wall time? The domain here considered represents a portion of variable size of the thoracic aorta, cf. Fig. 16a. The velocity at the inlet has the following profile

$$\mathbf{u}_{inlet}(\mathbf{x}) = \begin{cases} \cos\left(\frac{\pi\|\mathbf{x}-\mathbf{c}_{inlet}\|}{2r_{inlet}}\right) v_{peak}\mathbf{n}_{inlet} & \text{if } \|\mathbf{x}-\mathbf{c}_{inlet}\| < r_{inlet} \\ 0 & \text{otherwise} \end{cases}$$

where $r_{inlet}$ is the radius of the aorta at the inlet, $\mathbf{c}_{inlet}$ is the center point of the inlet surface, $\mathbf{n}_{inlet}$ is the normal to the inlet surface, and $v_{peak}$ is the peak velocity observed in the thoracic aorta, i.e. 60 cm/s [42]. No-slip boundary conditions are imposed on the walls, while homogenous Neumann conditions are imposed at the outflow.

For our computation, we consider two meshes representing different refinements of the thoracic aorta. Each mesh is then considered in three versions: one quarter, one half, or the full vessel (see Fig. 16a).

Tables 23 and 24 present the weak scalability results in terms of number of iterations and wall time. The three first columns contain the number of iterations to solve the system and the wall time for building the preconditioner and for solving the linear system using the preconditioned GMRES iterations; the two last columns display the ratio of the first by the second column and the ratio of the second by the third column, respectively. A ratio close to the ratio between the number of degrees of freedom per cores given in the header of the fourth and fifth columns indicates that weak scalability is achieved.

The AS preconditioner is not weakly scalable when solving the system (number of iterations and thus timing). However, building the preconditioner is almost weakly scalable; this comes from the fact that the size of the subdomains considered by the AS preconditioner is almost the same. The three other preconditioners behave similarly in terms of weak scalability. The number of iterations is not kept constant, though. AS is the least efficient preconditioner for this benchmark test.

## 8. Conclusions

In this work, we focused on developing preconditioners suitable for high performance computing on parallel machines with many cores. We have paid specific attention to results relevant to hemodynamics simulations. Our preconditioners have been tested on three distinctive features: scalability, optimality, and robustness. We proposed a strategy to develop approximate preconditioners which are fast to build and to apply. We considered aSIMPLE, aYosida, and aPCD preconditioners. In all our numerical tests, they clearly outperform the overlapping additive Schwarz preconditioner. On the one hand, aSIMPLE exhibits the fastest time to be built,

the lowest iterations number, and the lowest time to solution in almost all the situations that we have covered. On the other hand, aYosida is sensitive to both the timestep $\Delta t$ (by construction) and the viscosity coefficient $\nu$. Finally, the aPCD preconditioner turns out to be the candidate that competes well with aSIMPLE, although it is less straightforward to build.

## Acknowledgements

We acknowledge the European Research Council Advanced Grant "Mathcard, Mathematical Modelling and Simulation of the Cardiovascular System", Project ERC-2008-AdG 227058, and the Swiss Platform for High-Performance and High-Productivity Computing (HP2C). This work was supported by a grant from the Swiss National Supercomputing Centre (CSCS) under project IDs h02 and s392. We gratefully acknowledge the CSCS for providing us the CPU resources for our simulations. Many thanks to Professors Andy Wathen and Michele Benzi for their comments at the International Conference On Preconditioning Techniques For Scientific And Industrial Applications, Bordeaux, France. We are also grateful to Drs. Mike Heroux, Andrew Salinger, Jeremie Gaidamour, Mark Hoemmen, Jonathan Hu, Chris Siefert, and Professor John Shadid from SANDIA National Labs for their help, remarks, and advices. Finally we acknowledge the research project Aneurisk for supplying the geometry of the cerebral aneurysm [43].

## References

[1] Saad Y, Schultz MH. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J Sci Statist Comput 1986;7(3):856–69. http://dx.doi.org/10.1137/0907058. http://dx.doi.org/10.1137/0907058.

[2] Saad Y. A flexible inner-outer preconditioned GMRES algorithm. SIAM J Sci Comput 1993;14(2):461–9. http://dx.doi.org/10.1137/0914028. http://dx.doi.org/10.1137/0914028.

[3] Quarteroni A, Saleri F, Veneziani A. Factorization methods for the numerical approximation of Navier–Stokes equations. Comput Methods Appl Mech Eng 2000;188(1–3):505–26. http://dx.doi.org/10.1016/S0045-7825(99)00192-9. http://dx.doi.org/10.1016/S0045-7825(99)00192-9.

[4] Elman H, Howle VE, Shadid J, Shuttleworth R, Tuminaro R. A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier–Stokes equations. J Comput Phys 2008;227(3):1790–808. http://dx.doi.org/10.1016/j.jcp.2007.09.026. http://dx.doi.org/10.1016/j.jcp.2007.09.026.

[5] Patankar SV, Spalding DB. A calculation procedure for heat, mass and momentum transfer in three dimensional parabolic flows. Int J Heat Mass Transfer 1972;15:1787–806. http://dx.doi.org/10.1016/0017-9310(72)90054-3.

[6] Quarteroni A, Saleri F, Veneziani A. Analysis of the Yosida method for the incompressible Navier–Stokes equations. J Math Pures Appl 1999(9 78 5):473–503. http://dx.doi.org/10.1016/S0021-7824(99)00027-6. http://dx.doi.org/10.1016/S0021-7824(99)00027-6.

[7] Silvester D, Elman H, Kay D, Wathen A. Efficient preconditioning of the linearized Navier–Stokes equations for incompressible flow. J Comput Appl Math 2001;128(1–2):261–79. http://dx.doi.org/10.1016/S0377-0427(00)00515-X. http://dx.doi.org/10.1016/S0377-0427(00)00515-X [Numerical analysis 2000, vol. VII, Partial differential equations].

[8] Kay D, Loghin D, Wathen A. A preconditioner for the steady-state Navier–Stokes equations. SIAM J Sci Comput 2002;24(1):237–56. http://dx.doi.org/10.1137/S106482759935808X. http://dx.doi.org/10.1137/S106482759935808X.

[9] Elman HC, Tuminaro RS. Boundary conditions in approximate commutator preconditioners for the Navier–Stokes equations. Electron Trans Numer Anal 2009;35:257–80.

[10] Murphy MF, Golub GH, Wathen AJ. A note on preconditioning for indefinite linear systems. SIAM J Sci Comput 2000;21(6):1969–72. http://dx.doi.org/10.1137/S1064827599355153. http://dx.doi.org/10.1137/S1064827599355153.

[11] Elman H, Howle VE, Shadid J, Shuttleworth R, Tuminaro R. Block preconditioners based on approximate commutators. SIAM J Sci Comput 2006;27(5):1651–68. http://dx.doi.org/10.1137/040608817. http://dx.doi.org/10.1137/040608817.

[12] Elman H, Howle VE, Shadid J, Silvester D, Tuminaro R. Least squares preconditioners for stabilized discretizations of the Navier–Stokes equations. SIAM J Sci Comput 2007;30(1):290–311. http://dx.doi.org/10.1137/060655742. http://dx.doi.org/10.1137/060655742.

[13] Benzi M, Ng M, Niu Q, Wang Z. A relaxed dimensional factorization preconditioner for the incompressible Navier–Stokes equations. J Comput Phys 2011;230(16):6185–202. http://dx.doi.org/10.1016/j.jcp.2011.04.001. http://dx.doi.org/10.1016/j.jcp.2011.04.001.

[14] Benzi M, Guo X-P. A dimensional split preconditioner for Stokes and linearized Navier–Stokes equations. Appl Numer Math 2011;61(1):66–76. http://dx.doi.org/10.1016/j.apnum.2010.08.005. http://dx.doi.org/10.1016/j.apnum.2010.08.005.

[15] Karypis G, Schloegel K, Kumar V. METIS: a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Tech rep, Univ MN; 1998.

[16] Karypis G, Schloegel K, Kumar V. ParMETIS: parallel graph partitioning and sparse matrix ordering library. Tech rep, Univ MN; 2003.

[17] Quarteroni A. Numerical models for differential problems. MS&A. Modeling, simulation and applications, vol. 2. Italia (Milan): Springer-Verlag; 2009. doi:10.1007/978-88-470-1071-0. http://dx.doi.org/10.1007/978-88-470-1071-0.

[18] Quarteroni A, Valli A. Domain decomposition methods for partial differential equations. Numerical mathematics and scientific computation. New York: The Clarendon Press Oxford University Press, Oxford Science Publications; 1999.

[19] Toselli A, Widlund O. Domain decomposition methods—algorithms and theory. Springer series in computational mathematics, vol. 34. Berlin: Springer-Verlag; 2005.

[20] Trottenberg U, Clees T. Multigrid software for industrial applications – from mg00 to samg. In: Hirschel E, Krause E, editors. Notes on numerical fluid mechanics and multidisciplinary design, vol. 100. Berlin (Heidelberg): Springer; 2009. p. 423–36. doi:10.1007/978-3-540-70805-6_33. http://dx.doi.org/10.1007/978-3-540-70805-6_33.

[21] Hackbusch W. Multi-Grid methods and applications. Springer series in computational mathematics. Berlin (Heidelberg): Springer-Verlag; 1985.

[22] Trottenberg U, Oosterlee CW, Schüller A. Multigrid. San Diego (CA): Academic Press Inc.; 2001 [with contributions by A. Brandt, P. Oswald , K. Stüben].

[23] Ethier CR, Steinman DA. Exact fully 3D Navier–Stokes solutions for benchmarking. Int J Numer Methods Fluids 1994;19(5):369–75. http://dx.doi.org/10.1002/fld.1650190502.

[24] Badia S, Quaini A, Quarteroni A. Splitting methods based on algebraic factorization for fluid–structure interaction. SIAM J Sci Comput 2008;30(4):1778–805. http://dx.doi.org/10.1137/070680497. http://dx.doi.org/10.1137/070680497.

[25] Badia S, Quaini A, Quarteroni A. Modular vs. non-modular preconditioners for fluid–structure systems with large added-mass effect. Comput Methods Appl Mech Eng 2008;197(49–50):4216–32. http://dx.doi.org/10.1016/j.cma.2008.04.018. http://dx.doi.org/10.1016/j.cma.2008.04.018.

[26] Crosetto P, Deparis S, Fourestey G, Quarteroni A. Parallel algorithms for fluid–structure interaction problems in haemodynamics. SIAM J Sci Comput 2011;33(4):1598–622. http://dx.doi.org/10.1137/090772836. http://dx.doi.org/10.1137/090772836.

[27] Formaggia L, Quarteroni A, Veneziani A, editors. Cardiovascular mathematics, MS&A. Modeling, simulation and applications, vol. 1. Italia (Milan): Springer-Verlag; 2009. doi:10.1007/978-88-470-1152-6. URL http://dx.doi.org/10.1007/978-88-470-1152-6 [modeling and simulation of the circulatory system].

[28] Quarteroni A, Valli A. Numerical approximation of partial differential equations. Springer series in computational mathematics, vol. 23. Berlin: Springer-Verlag; 1994.

[29] Pernice M, Tocci MD. A multigrid-preconditioned Newton–Krylov method for the incompressible Navier–Stokes equations. SIAM J Sci Comput 2001;23(2):398–418. http://dx.doi.org/10.1137/S1064827500372250. http://dx.doi.org/10.1137/S1064827500372250 [(electronic), copper Mountain Conference; 2000].

[30] Gervasio P. Convergence analysis of high order algebraic fractional step schemes for time-dependent stokes equations. SIAM J Numer Anal 2008;46(4):1682–703. http://dx.doi.org/10.1137/070682800. arXiv:http://epubs.siam.org/doi/pdf/10.1137/070682800. http://epubs.siam.org/doi/abs/10.1137/070682800.

[31] Veneziani A. A note on the consistency and stability properties of Yosida fractional step schemes for the unsteady stokes equations. SIAM J Numer Anal 2009;47(4):2838–43. http://dx.doi.org/10.1137/080724484. arXiv:http://epubs.siam.org/doi/pdf/10.1137/080724484. http://epubs.siam.org/doi/abs/10.1137/080724484.

[32] Veneziani A, Villa U. ALADINS: an ALgebraic splitting time adaptive solver for the incompressible Navier–Stokes equations. J Comput Phys 2013;238(0021-9991):359–75.

[33] Ipsen ICF. A note on preconditioning nonsymmetric matrices. SIAM J Sci Comput 2001;23(3):1050–1. http://dx.doi.org/10.1137/S1064827500377435. http://dx.doi.org/10.1137/S1064827500377435.

[34] Elman HC, Silvester DJ, Wathen AJ. Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics. Numerical mathematics and scientific computation. New York: Oxford University Press; 2005.

[35] ur Rehman M, Vuik C, Segal G. A comparison of preconditioners for incompressible Navier–Stokes solvers. Int J Numer Methods Fluids 2008;57(12):1731–51. http://dx.doi.org/10.1002/fld.1684. http://dx.doi.org/10.1002/fld.1684.

[36] van der Vorst HA, Vuik C. GMRESR: a family of nested GMRES methods. Numer Linear Algebra Appl 1994;1(4):369–86. http://dx.doi.org/10.1002/nla.1680010404. http://dx.doi.org/10.1002/nla.1680010404.

[37] Amdahl GM. Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18–20, 1967, spring joint computer conference, AFIPS '67 (Spring). New York (NY, USA): ACM; 1967. p. 483–5. doi: http://doi.acm.org/10.1145/1465482.1465560. http://doi.acm.org/10.1145/1465482.1465560.
[38] Golub GH, Van Loan CF. Matrix computations. Johns Hopkins studies in the mathematical sciences. Baltimore (MD): Johns Hopkins University Press; 1996.
[39] LifeV user manual; 2010. <http://www.lifev.org>.
[40] Heroux MA, Bartlett RA, Howle VE, Hoekstra RJ, Hu JJ, Kolda TG, et al. An overview of the trilinos project. ACM Trans Math Softw 2005;31(3):397–423. http://dx.doi.org/10.1145/1089014.1089021.
[41] Baek H, Jayaraman MV, Richardson PD, Karniadakis GE. Flow instability and wall shear stress variation in intracranial aneurysms. J R Soc Interface 2010;7:967–88.
[42] Nichols WW, O'Rourke MF, Hartley C, McDonald DA. McDonald's blood flow in arteries: theoretical, experimental, and clinical principles. 4th ed. London (New York): Arnold, Oxford University Press; 1998.
[43] Sangalli LM, Secchi P, Vantini S, Veneziani A. A case study in exploratory functional data analysis: geometrical features of the internal carotid artery. J Am Statist Assoc 2009;104(485):37–48. http://dx.doi.org/10.1198/jasa.2009.0002. <http://dx.doi.org/10.1198/jasa.2009.0002>.