Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 372 (2020) 113424
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Parallel block preconditioners for three-dimensional virtual element discretizations of saddle-point problems

F. Dassi[a,*], S. Scacchi[b]

[a] Dipartimento di Matematica, Università degli Studi di Milano-Bicocca, Via Cozzi, 20133 Milano, Italy
[b] Dipartimento di Matematica, Università degli Studi di Milano, Via Saldini 50, 20133 Milano, Italy

Received 14 May 2020; received in revised form 2 September 2020; accepted 2 September 2020
Available online 23 September 2020

Abstract

Several physical phenomena are described by systems of partial differential equations (PDEs) that, after space discretization, yield the solution of saddle point algebraic linear systems. In realistic three-dimensional numerical simulations, these linear systems are large scale and ill-conditioned, thus they require the development of effective solvers. The aim of this work is the construction and numerical validation of parallel block preconditioners for a set of three-dimensional saddle point problems discretized by the low order virtual element method (VEM). VEM is a recent numerical technology for the approximation of PDEs on polygonal and polyhedral meshes. We focus on the following systems of PDEs: stationary Maxwell equations in the mixed Kikuchi formulation; elliptic equations in mixed form; Stokes system; linear elasticity in the mixed Hellinger–Reissner formulation. We provide two parallel block preconditioners: one based on the approximate Schur complement and the other on a regularization technique. Several numerical experiments are run in parallel on a Linux cluster. We analyze the performance of the iterative solvers in terms of GMRES iterations and computational time. We verify the robustness of the solvers with respect to different polyhedral meshes and the scalability of both the assembling and solution time by varying the number of processors. The performance of the two iterative solvers is also compared with state-of-the-art parallel direct linear solvers.

Keywords: Virtual element method; Saddle-point linear systems; Parallel computing; Block preconditioners

1. Introduction

The Virtual Element Method (VEM) is a recent technology, first proposed in [1,2], for the numerical approximation of partial differential equations (PDEs) on polygonal or polyhedral grids; see also the reviews in [3,4]. In the last six years, VEM solution strategies have been constructed and analyzed for several problems governed by PDEs: scalar elliptic equations in primal [1,5,6] and mixed [7–9] form, elasticity [10–12], topology optimization [13,14], Stokes [15–17], Maxwell [18–20], parabolic and hyperbolic equations [21,22], Cahn–Hilliard [23] and further applications.

Analogously to the finite element case, VEM discretizations of scalar elliptic or elasticity equations in mixed form and further PDEs such as Stokes and Navier–Stokes equations or Maxwell equations, yield the solution of saddle-point linear algebraic systems, which are indefinite and ill-conditioned. As a consequence, the solution of such linear

* Corresponding author.
E-mail addresses: franco.dassi@unimib.it (F. Dassi), simone.scacchi@unimi.it (S. Scacchi).

https://doi.org/10.1016/j.cma.2020.113424

systems with iterative methods (see e.g. [24,25]) requires the development of robust and effective preconditioners, usually based on approximate block factorization, see [26–31]. Several Domain Decomposition preconditioners have also been proposed for finite element discretizations of this kind of problems, see [32–38]. So far, only a few studies have focused on the conditioning of the stiffness matrix resulting from VEM discretizations (see [6,39]) and on the development of preconditioners for VEM approximations of PDEs (see [40–43]).

The aim and novelty of the present work is the construction of parallel block preconditioners for the solution linear systems deriving from VEM discretizations of the following three-dimensional saddle point problems: stationary Maxwell equations in the mixed Kikuchi formulation [19]; elliptic equations in mixed form [9]; Stokes systems [17]; linear elasticity in the mixed Hellinger–Reissner formulation [12]. We develop two parallel block preconditioners, one based on the approximate Schur complement and one on a regularization technique. We compare the iterative methods against the parallel direct solver Mumps [44,45] and Pardiso [46–48] by performing several parallel tests on a Linux cluster with varying number of processors and type of polyhedral grid.

The rest of the paper is organized as follows: in Section 2, we briefly introduce the variational formulation of the saddle point problems and we describe the three-dimensional VEM discretizations of the model problems; in Section 3, we introduce the parallel block preconditioners used for the solution of the saddle point linear systems and we report the numerical experiments on a Linux cluster. Finally, in Section 4, we draw some conclusions about the analysis done in the numerical experiment part.

2. Saddle-point problems

In the present work, we consider different kind of variational problems which require the solution of a saddle-point linear system, i.e., the stiffness matrix can be written as

$$\mathcal{A} = \begin{bmatrix} A & B^T \\ B & \mathbf{0} \end{bmatrix}, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{m\times n}$, then the whole linear system $\mathcal{A} \in \mathbb{R}^{(n+m)\times(n+m)}$.

Many physical phenomena can be described via partial differential equations whose discretization leads to a linear system like the one in Eq. (1).

In a very general setting the variational formulation of such problems is the following. Let Ω be a bounded Lipschitz domain in $\mathbb{R}^d$, whose boundary is denoted by $\partial\Omega$. We look for a couple of functions $(\mathbf{u}, q) \in \mathbf{V} \times Q$ which satisfies

$$\begin{cases} \boldsymbol{a}(\mathbf{u}, \mathbf{v}) + \boldsymbol{b}(\mathbf{v}, p) &= c_1(\mathbf{v}) \quad \forall \mathbf{v} \in \mathbf{V} \\ \boldsymbol{b}(\mathbf{u}, q) &= c_2(q) \quad \forall q \in Q \end{cases}, \tag{2}$$

where $\boldsymbol{a}(\cdot,\cdot)$ and $\boldsymbol{b}(\cdot,\cdot)$ are bi-linear forms, while $c_1(\cdot)$ and $c_2(\cdot)$ are linear forms. More specifically, the linear operators $\boldsymbol{a}(\cdot,\cdot)$ and $\boldsymbol{b}(\cdot,\cdot)$ will define the matrix A and B of the linear system $\mathcal{A}$, see Eq. (1). According to the problem at hand the unknowns $\mathbf{u}$ and p represent different physical quantities and, consequently, the functional spaces $\mathbf{V}$ and Q have to be chosen accordingly. Suppose, for instance, that we are interested in a static fluid-dynamics. In such case the $\mathbf{u}$ variable represents the velocity vector field, while p is the scalar field which represents pressure.

In the last part of this section we show which are the problems taken into account for the solver and preconditioners analysis. More specifically, we will underline which are the continuous spaces ($\mathbf{V}$ and Q) and linear/bi-linear forms involved for each kind of problem, i.e., we define the forms $\boldsymbol{a}(\cdot,\cdot)$, $\boldsymbol{b}(\cdot,\cdot)$, $c_1(\cdot)$ and $c_2(\cdot)$ for each PDE taken into account.

Throughout the present work, we consider low-order virtual element discretizations of such problems. Since we are interested in the performances of different solvers, we will not go into the details about the virtual element discretization itself, but we will provide all the references so that the reader can find an in-depth analysis of such discretizations.

Mixed-form elliptic equations. Let us consider the continuous spaces

$$\mathbf{V}^{\mathcal{D}}(\Omega) := \left\{ \mathbf{v} \in H(\mathrm{div}\,;\Omega) \,:\, \mathbf{v}\cdot\mathbf{n} = u_N \text{ on } \partial\Omega \right\} \quad \text{and} \quad Q^{\mathcal{D}}(\Omega) := \left\{ q \in L^2(\Omega) \,:\, \int_\Omega q \,\mathrm{d}\Omega = 0 \right\},$$

and the linear/bi-linear forms:

$$\begin{aligned} &\boldsymbol{a}^{\mathcal{D}}(\mathbf{v},\boldsymbol{w}) : \mathbf{V}^{\mathcal{D}} \times \mathbf{V}^{\mathcal{D}} \to \mathbb{R}, \quad \boldsymbol{a}^{\mathcal{D}}(\mathbf{v},\boldsymbol{w}) := \int_{\Omega} \nu(x)\, \mathbf{v}\cdot\boldsymbol{w}\ \mathrm{d}\Omega, \\ &\boldsymbol{b}^{\mathcal{D}}(\mathbf{v},q) : \mathbf{V}^{\mathcal{D}} \times Q^{\mathcal{D}} \to \mathbb{R}, \quad \boldsymbol{b}^{\mathcal{D}}(\mathbf{v},q) := -\int_{\Omega} q\, \mathrm{div}\,(\mathbf{v})\ \mathrm{d}\Omega, \\ &c_2^{\mathcal{D}}(q) : Q^{\mathcal{D}} \to \mathbb{R}, \qquad\qquad c_2^{\mathcal{D}}(q) := -\int_{\Omega} f\, q\ \mathrm{d}\Omega, \end{aligned} \tag{3}$$

$c_1^{\mathcal{D}}(\mathbf{v})$ is identically zero, $\nu(x)$ is a positive piece-wise constant scalar function and $f \in L^2(\Omega)$. Such problem arises in the simulation of multi-phase in-compressible flow through porous media. We consider the virtual element spaces introduced in [9,49].

Stokes problem. For such problem we consider the following functional spaces

$$\mathbf{V}^{\mathcal{S}}(\Omega) := [H_0^1(\Omega)]^3 \qquad \text{and} \qquad Q^{\mathcal{S}}(\Omega) := \left\{ q \in L^2(\Omega) \ :\ \int_{\Omega} q\ \mathrm{d}\Omega = 0 \right\}.$$

Then, we have the following forms

$$\begin{aligned} &\boldsymbol{a}^{\mathcal{S}}(\mathbf{v},\boldsymbol{w}) : \mathbf{V}^{\mathcal{S}} \times \mathbf{V}^{\mathcal{S}} \to \mathbb{R}, \qquad\qquad \boldsymbol{a}^{\mathcal{S}}(\mathbf{v},\boldsymbol{w}) := \int_{\Omega} \nu(x)\, \nabla\mathbf{v} \ : \nabla\boldsymbol{w}\ \mathrm{d}\Omega, \\ &\boldsymbol{b}^{\mathcal{S}}(\mathbf{v},q) : \mathbf{V}^{\mathcal{S}} \times Q^{\mathcal{S}} \to \mathbb{R}, \qquad\qquad \boldsymbol{b}^{\mathcal{S}}(\mathbf{v},q) := \int_{\Omega} q\, \mathrm{div}\,(\mathbf{v})\ \mathrm{d}\Omega, \\ &c_1^{\mathcal{S}}(\mathbf{v}) : \mathbf{V}^{\mathcal{S}} \to \mathbb{R}, \qquad\qquad c_1^{\mathcal{S}}(\mathbf{v}) := \int_{\Omega} \mathbf{f}\cdot\mathbf{v}\ \mathrm{d}\Omega, \end{aligned}$$

$c_2^{\mathcal{S}}(\mathbf{v})$ is identically zero, $\mathbf{f} \in [L^2(\Omega)]^3$. Stokes differential equations are widely used to describe the flow motion. In this paper we will use the virtual element discretization proposed in [17]. We refer to this paper for a deeper analysis about virtual element spaces and how discretize such forms.

Magneto-static problem. We take these functional spaces

$$\begin{aligned} &\mathbf{V}^{\mathcal{M}}(\Omega) := \left\{ \mathbf{v} \in [L^2(\Omega)]^3 \ :\ \mathbf{curl}(\mathbf{v}) \in [L^2(\Omega)]^3 \text{ with } \mathbf{v}\wedge\mathbf{n} = 0 \text{ on } \partial\Omega \right\} \quad \text{and} \\ &Q^{\mathcal{M}}(\Omega) := \left\{ q \in H^1(\Omega) \ :\ q = 0 \text{ on } \partial\Omega \right\}. \end{aligned}$$

In such case the linear and bi-linear forms of Eq. (2) become

$$\begin{aligned} &\boldsymbol{a}^{\mathcal{M}}(\mathbf{v},\boldsymbol{w}) : \mathbf{V}^{\mathcal{M}} \times \mathbf{V}^{\mathcal{M}} \to \mathbb{R}, \qquad\qquad \boldsymbol{a}^{\mathcal{M}}(\mathbf{v},\boldsymbol{w}) := \int_{\Omega} \mathbf{curl}(\mathbf{v})\cdot\mathbf{curl}(\boldsymbol{w})\ \mathrm{d}\Omega, \\ &\boldsymbol{b}^{\mathcal{M}}(\mathbf{v},q) : \mathbf{V}^{\mathcal{S}} \times Q^{\mathcal{S}} \to \mathbb{R}, \qquad\qquad \boldsymbol{b}^{\mathcal{M}}(\mathbf{v},q) := \int_{\Omega} \nabla q\cdot\mu\mathbf{v}\ \mathrm{d}\Omega, \\ &c_1^{\mathcal{M}}(\mathbf{v}) : \mathbf{V}^{\mathcal{S}} \to \mathbb{R}, \qquad\qquad c_1^{\mathcal{S}}(\mathbf{v}) := \int_{\Omega} \boldsymbol{j}\cdot\mathbf{curl}(\mathbf{v})\ \mathrm{d}\Omega, \end{aligned}$$

$c_2^{\mathcal{M}}(\mathbf{v})$ is identically zero, μ is the magnetic permeability and $\boldsymbol{j}$ is the current density, $\boldsymbol{j} \in H(\mathrm{div}\,;\Omega)$ such that $\mathrm{div}\,(\boldsymbol{j}) = 0$ in Ω. Such problem describes the distribution of the magnetic field in the presence of a current density $\boldsymbol{j}$. To have a virtual element discretization of such problem, we take into account the lowest-order formulation proposed in [19].

Hellinger–Reissner elasticity problem. We consider the functional spaces

$$\mathbf{V}^{\mathcal{E}}(\Omega) := \left\{ \mathbf{v} \in H(\mathbf{div};\Omega) \ :\ \mathbf{v} = \mathbf{v}^t \right\} \quad \text{and} \quad Q^{\mathcal{E}}(\Omega) := \left[L^2(\Omega) \right]^3,$$

where as usual the functional space $H(\mathbf{div};\Omega)$ denotes the space of tensors in $[L^2(\Omega)]^{3\times 3}$ whose divergence is the vector-valued operator in $[L^2(\Omega)]^3$. In this framework $\mathbf{V}^{\mathcal{E}}$ and $Q^{\mathcal{E}}$ represent the stress tensor and the displacement, respectively. We take the following linear/bi-linear forms to set the variational formulation based on the Hellinger–Reissner principle:

$$\begin{aligned} \boldsymbol{a}^{\mathcal{E}}(\mathbf{v},\boldsymbol{w}) &: \mathbf{V}^{\mathcal{E}}\times\mathbf{V}^{\mathcal{E}}\to\mathbb{R}, & \boldsymbol{a}^{\mathcal{E}}(\mathbf{v},\boldsymbol{w}) &:= \int_\Omega \mathbb{D}\mathbf{v} : \boldsymbol{w}\ \mathrm{d}\Omega,\\ \boldsymbol{b}^{\mathcal{E}}(\mathbf{v},q) &: \mathbf{V}^{\mathcal{E}}\times Q^{\mathcal{E}}\to\mathbb{R}, & \boldsymbol{b}^{\mathcal{E}}(\mathbf{v},q) &:= \int_\Omega \mathrm{div}\,(\mathbf{v})\cdot q\ \mathrm{d}\Omega,\\ c_2^{\mathcal{E}}(\mathbf{v}) &: Q^{\mathcal{E}}\to\mathbb{R}, & c_2^{\mathcal{E}}(q) &:= \int_\Omega \mathbf{f}\cdot q\ \mathrm{d}\Omega, \end{aligned}$$

$c_1^{\mathcal{E}}(\mathbf{v})$ is identically zero, $\mathbf{f}\in\left[L^2(\Omega)\right]^3$ is the loading term and $\mathbb{D}$ is a symmetric tensor associated with the elastic property of the material we are considering.

In the definition of this problem, we make an abuse of notation. Indeed, $q\in Q^{\mathcal{E}}$ is a vectorial function, but we do not use bold faced letters. We make this decision to further underline the saddle-point nature of the linear system arising form such formulation and, consequently, the common aspects with the previous three problems. To make a virtual element discretization of such problem, we refer to the theory proposed in [12].

Let Ω_h be a polyhedral discretization of a domain $\Omega\subset\mathbb{R}^3$. To solve one of the problem defined in Section 2, we follow a standard VEM approach, i.e., we define local spaces on each polyhedron $P\in\Omega_h$ and then we glue them together to get the global space.

Since the focus of this paper is a generic lower-order discretization of a continuous problem that provides a saddle-point linear system, we cannot describe in detail all the virtual functional spaces taken into account. We describe only the virtual element spaces used for a mixed-form elliptic problem and we demand the reader to [12,17,19] for a deeper description about the other ones.

However, at the end of this section we summarize the common aspects of the virtual element spaces involved to solve each problem described in Section 2.

2.1. Virtual element spaces for a mixed-form elliptic problem

To discretize the space $\mathbf{V}^{\mathcal{D}}(\Omega)$ we consider the virtual element space introduced in [9]. As we said before we focus on the definition of such space in a polyhedron P and, since we are interested in the lowest-order case, we show the degree one definition. Given a polyhedron P, the velocity field is defined as

$$\mathbf{V}_h^{\mathcal{D}}(P) := \Big\{ \mathbf{v}_h\in H(\mathrm{div}\,;P)\cap H(\mathbf{curl};P) : \mathbf{v}_h\cdot\mathbf{n}_F\in\mathbb{P}_1(F)\quad\forall F\in\partial P,\ \mathrm{div}\,(\mathbf{v}_h)\in\mathbb{P}_0(P),\quad \mathbf{curl}(\mathbf{v}_h)\in[\mathbb{P}_0(P)]^3 \Big\}. \tag{4}$$

To uniquely define a function inside such space, we introduce the following degrees of freedom:

- *normal face moments*

$$\int_F (\mathbf{v}_h\cdot\mathbf{n})\,p_1\,\mathrm{d}F\quad\forall p_1\in\mathbb{P}_1(F)\quad\text{and}\quad\forall F\in\partial P.$$

- *internal cross moments*

$$\int_P \mathbf{v}_h\cdot(\boldsymbol{x}\wedge\boldsymbol{p}_0)\,\mathrm{d}P\quad\forall\boldsymbol{p}_0\in[\mathbb{P}_0(P)]^3,$$

where $\boldsymbol{x}=(x,\ y,\ z)^t$.

On the other hand, to get a numerical approximation of $Q^{\mathcal{D}}(\Omega)$, we use the space of element-wise constant polynomials, i.e.,

$$Q_h^{\mathcal{D}}(P) := \Big\{ q_h\in L^2(P)\ :\ q_h\in\mathbb{P}_0(P)\Big\}.$$

In such space a function is determined by one degree of freedom. In the virtual element framework, we take the following degree of freedom

- the *internal moment*

$$\int_P q_h \, \mathrm{d}P \, . \tag{5}$$

Although a function $\mathbf{v}_h$ is virtual, it is possible to compute its divergence and the polynomial defined on each face F [9]. Moreover, we define the L^2-projection operator $\mathbf{\Pi}^0 : \mathbf{V}_h^{\mathcal{D}}(P) \to [\mathbb{P}_1(P)]^3$ such that

$$\int_P \mathbf{\Pi}^0 \mathbf{v}_h \cdot \boldsymbol{p}_1 \, \mathrm{d}P = \int_P \mathbf{v}_h \cdot \boldsymbol{p}_1 \, \mathrm{d}P \, . \tag{6}$$

Since the function $\mathbf{v}_h$ is virtual the main issue is associated with the right-hand side of Eq. (6). Indeed, starting from the degrees of freedom and the polynomial identity

$$\boldsymbol{p}_1 = \nabla p_2 + \boldsymbol{x} \wedge \boldsymbol{p}_0,$$

we have

$$\begin{aligned}
\int_P \mathbf{v}_h \cdot \boldsymbol{p}_1 \, \mathrm{d}P &= \int_P \mathbf{v}_h \cdot (\nabla p_2) \, \mathrm{d}P + \int_P \mathbf{v}_h \cdot (\boldsymbol{x} \wedge \boldsymbol{p}_0) \, \mathrm{d}P \\
&= -\int_P \operatorname{div}(\mathbf{v}_h) \, p_2 \, \mathrm{d}P + \int_{\partial P} (\mathbf{v}_h \cdot \mathbf{n}) \, p_2 \, \mathrm{d}F + \int_P \mathbf{v}_h \cdot (\boldsymbol{x} \wedge \boldsymbol{p}_0) \, \mathrm{d}P \\
&= -\int_P \operatorname{div}(\mathbf{v}_h) \, p_2 \, \mathrm{d}P + \sum_{F \in \partial P} \int_F (\mathbf{v}_h \cdot \mathbf{n}_F) \, p_2 \, \mathrm{d}F + \int_P \mathbf{v}_h \cdot (\boldsymbol{x} \wedge \boldsymbol{p}_0) \, \mathrm{d}P \, .
\end{aligned}$$

Since we can get both the polynomials $(\mathbf{v}_h \cdot \mathbf{n}_F)$ on each face and the constant divergence of $\mathbf{v}_h$ via the degrees of freedom, we can compute the right-hand side and the projection operator $\mathbf{\Pi}^0$ too.

2.2. Operators for a mixed-form elliptic problem

To proceed with the discretization of a mixed-form elliptic problem, we introduce some linear/bi-linear operators that are the discrete counterpart of the ones defined in Eq. (3). For this particular problem we define the following local forms:

- *flux operator:*

$$\boldsymbol{a}_{h,P}^{\mathcal{D}}(\mathbf{v}_h, \boldsymbol{w}_h) \coloneqq \int_P \nu(x) \, \mathbf{\Pi}^0 \mathbf{v}_h \cdot \mathbf{\Pi}^0 \boldsymbol{w}_h \, \mathrm{d}P + s_P(\mathbf{v}_h - \mathbf{\Pi}^0 \mathbf{v}_h, \boldsymbol{w}_h - \mathbf{\Pi}^0 \boldsymbol{w}_h) \, , \tag{7}$$

  where s_P is any symmetric and positive definite bi-linear form which scales as the $\boldsymbol{a}(\cdot, \cdot)$. In this paper we choose the Euclidean scalar product associated with the degrees of freedom of $\mathbf{V}_h^{\mathcal{D}}(P)$ multiplied by the volume of P and the value of $\nu(x)$ at its barycenter [2,7], i.e.,

$$s_P(\mathbf{v}_h, \boldsymbol{w}_h) = \nu(\boldsymbol{x}_P) \, |P| \sum_{i=1}^{\#dof_P} dof_i(\mathbf{v}_h) \, dof_i(\boldsymbol{w}_h),$$

  where $\#dof_P$ are the number of degrees of freedom associated with a function in $\mathbf{V}_h^{\mathcal{D}}(P)$ and $dof_i : \mathbf{V}_h^{\mathcal{D}}(P) \to \mathbb{R}$ is a linear functional that associates to a function in $\mathbf{V}_h^{\mathcal{D}}(P)$ the value of its ith degree of freedom.
- *divergence operator:*

$$\boldsymbol{b}_P^{\mathcal{D}}(\mathbf{v}_h, q_h) \coloneqq \int_P \operatorname{div}(\mathbf{v}_h) \, q_h \, \mathrm{d}P \, , \tag{8}$$

- *right-hand side operator:*

$$c_{2,h,P}^{\mathcal{D}}(q_h) \coloneqq -\int_P f \, q_h \, \mathrm{d}P.$$

Then, the global one is obtained by the sum of all these local contributions, i.e.,

$$\boldsymbol{a}^{\mathcal{D}}(\mathbf{v},\ \boldsymbol{w}) \approx \boldsymbol{a}_h^{\mathcal{D}}(\mathbf{v}_h,\ \boldsymbol{w}_h) = \sum_{P\in\Omega_h} \boldsymbol{a}_{h,P}^{\mathcal{D}}(\mathbf{v}_h, \boldsymbol{w}_h)\,,$$

$$\boldsymbol{b}^{\mathcal{D}}(\mathbf{v},\ q) \approx \boldsymbol{b}^{\mathcal{D}}(\mathbf{v}_h,\ q_h) = \sum_{P\in\Omega_h} \boldsymbol{b}_P^{\mathcal{D}}(\mathbf{v}_h, q_h)\,,$$

$$c_2^{\mathcal{D}}(q) \approx c_{2,h}^{\mathcal{D}}(q_h) = \sum_{P\in\Omega_h} c_{2,h,P}^{\mathcal{D}}(q_h)\,.$$

2.3. Saddle-point common aspects

In this subsection we show the common characteristic of the proposed saddle-point linear systems. We focus on each sub-matrix of $\mathcal{A}$ defined in Eq. (1).

Sub-matrix A. In all the proposed saddle-point problems this sub-matrix is built by an operator similar to the one in Eq. (7). Indeed, there is always a first part which depends on a suitable L^2-projection operator and a second part with a stabilization. Consequently such global matrix is the sum of two matrices,

$$A = C + S,$$

where C involves projection operators, the so-called *consistency matrix*, and S is the *stabilization matrix*. The matrix C gives the right accuracy of the method with respect to the polynomial degree. However, since the virtual element space is richer than the polynomial space, the matrix C is singular, i.e., its rank is lower than its dimension. The role of the matrix S is to correct the rank of C so that the whole matrix A has full rank.

To get the matrix S, we use a similar stabilization operator. Indeed in all the problems we use the stabilization s_P based on the dof_i linear operator and it is properly scaled by the data of the problem at hand. Consider for instance a mixed-form elliptic or a Stokes problem. In such case we use the evaluation of the $\nu(x)$ function at the barycenter. While, when we have an elasticity problem via the Hellinger–Reissner principle, we scale the stabilization with the trace of the linear tensor $\mathbb{D}$.

Sub-matrix B. In a Maxwell problem such matrix has a structure similar to the sub-matrix A. Indeed it has both a projection and a stabilization part. However, B becomes more interesting if we are considering the other problems taken into account.

In all these cases the matrix B is computed *exactly* in the virtual element spaces. Consider for instance the Darcy problem described in Section 2.2. If we analyze more into the details of the definition of the local operator $\mathbf{b}_P$, Eq. (8), we observe that we know exactly all the functions involved in the computation of such integral. Indeed, q_h is a polynomial and $\operatorname{div}(\mathbf{v}_h)$ is a constant polynomial, see the definition of $\mathbf{V}_h^{\mathcal{D}}(P)$ in Eq. (4). Both polynomials are computable from the degrees of freedom of the spaces so, although the function $\mathbf{v}_h$ is virtual, we are able to compute such integral.

Notice that to better underline this fact, we omit the subscript h in the definition of $\boldsymbol{b}_P^{\mathcal{D}}$, see Eq. (8). Indeed, if we are able to compute integrals of polynomial in polyhedrons, we get the exact value of this bi-linear operator.

As already mentioned, a similar observation holds for both Stokes and Hellinger–Reissner problems. More specifically, in the first one we have exactly the same configuration of a Mixed problem, i.e., the divergence of a virtual function $\mathbf{v}_h \in \mathbf{V}_h^{\mathcal{S}}(P)$ is a constant polynomial and $q_h \in Q_h^{\mathcal{S}}(P)$ is a polynomial too. Then, in an Hellinger–Reissner problem the divergence of $\mathbf{v}_h \in \mathbf{V}_h^{\mathcal{E}}(P)$ and the displacement space $Q_h^{\mathcal{D}}(P)$ are rigid body motions, i.e., vectorial polynomials of the form

$$\mathbf{r}(\boldsymbol{x}) = \boldsymbol{\alpha} + \boldsymbol{\omega}(\boldsymbol{x} - \boldsymbol{x}_P),$$

where $\boldsymbol{\alpha}, \boldsymbol{\omega} \in [\mathbb{P}_0(P)]^3$ and $\boldsymbol{x}_P$ is the barycenter of the polyhedron P. In [12] it is shown that the degrees of freedom of the space $\mathbf{V}_h^{\mathcal{E}}(P)$ are chosen in such a way that the known coefficients of $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$ are computable.

3. Parallel preconditioners

The c++ parallel code we have developed is based on the PETSc library from Argonne National Laboratory [50–52]. Such library is built on the MPI standard and offers advanced data structures and routines for the parallel

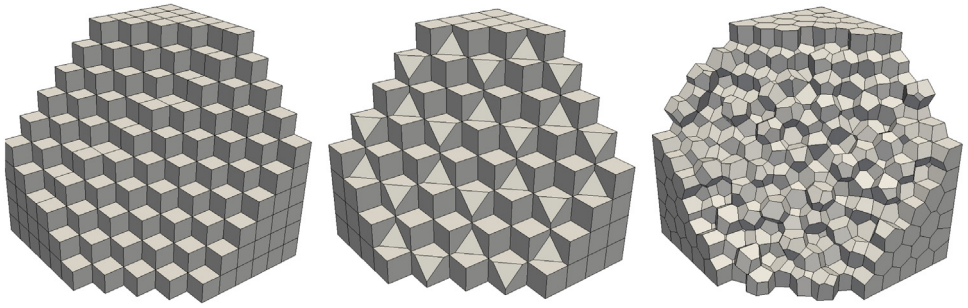

Fig. 1. Hexahedral (Cube, left), octahedral (Octa, middle) and Voronoi (CVT, right) meshes of the unit cube.

solution of partial differential equations, such as basic vector and matrix operations and even more complex linear and non-linear equation solvers. In our c++ code, vectors and matrices are built and sub-assembled in parallel on each processor.

To solve the saddle-point linear system $\mathcal{A}$, see Eq. (1), we use the parallel GMRES method provided by the PETSc library, employing a zero initial guess and a stopping criterion of 10^{-10} reduction of the relative residual. We consider two types of block-diagonal preconditioners (see e.g. [29–31]) of the form

$$\mathcal{B}_D = \begin{bmatrix} \mathrm{B}_1 & 0 \\ 0 & \mathrm{B}_2 \end{bmatrix}: \tag{9}$$

- **Block-Schur** where

$$\begin{aligned} B_1^{-1} &= \text{ diagonal or Algebraic Multigrid preconditioner for } A \\ B_2^{-1} &= \text{ exact solution of the approximate Schur complement } S \end{aligned} \tag{10}$$

  with $S = -B\ diag(A)^{-1}B^T$. As Algebraic Multigrid preconditioner we use the GAMG solver of PETSc or BoomerAMG [53], provided within the Hypre library [54]. Since we did not observe significant improvements changing the parameters of the Algebraic Multigrid solvers, all the next results are obtained using the default values. For the inversion of S at each preconditioning step we use the parallel multifrontal direct solver Mumps [44,45].
- **Block-Reg** where

$$\begin{aligned} B_1^{-1} &= \text{ Algebraic Multigrid preconditioner for } A + B^T W^{-1} B, \\ B_2^{-1} &= W^{-1} \end{aligned} \tag{11}$$

  with $W = \gamma I$, for a suitable parameter $\gamma > 0$. To our knowledge, there is no theoretical support for the choice of γ. In the numerical tests, we study the dependence of the solver on the choices $\gamma = h,\ h^2,\ h^3$. As Algebraic Multigrid preconditioner we use the GAMG solver of PETSc.

In the following tests we compare the previous two block-diagonal preconditioners and two parallel direct solvers Mumps and Pardiso [46–48] considering the model problems introduced in Section 2.

3.1. Numerical results

In the numerical tests we use the Linux cluster INDACO of the University of Milan, constituted by 16 nodes, each carrying 2 processors INTEL XEON E5-2683 V4 2.1 GHz, with 16 cores each (www.indaco.unimi.it). We consider three types of polyhedral meshes discretizing the unit cube: hexahedral (**Cube**), octahedral (**Octa**) and Voronoi (**CVT**), see Fig. 1. We underline that the first two mesh types are regular but, although **CVT** meshes do not have stretched polygons, they present small and distorted faces/edges next to regularly shaped ones, see the details in Fig. 2. Consequently, when we are dealing with such mesh type, the proposed preconditioners will be also tested against such "bad" mesh configurations. Since we are not interested in verifying the convergence rate

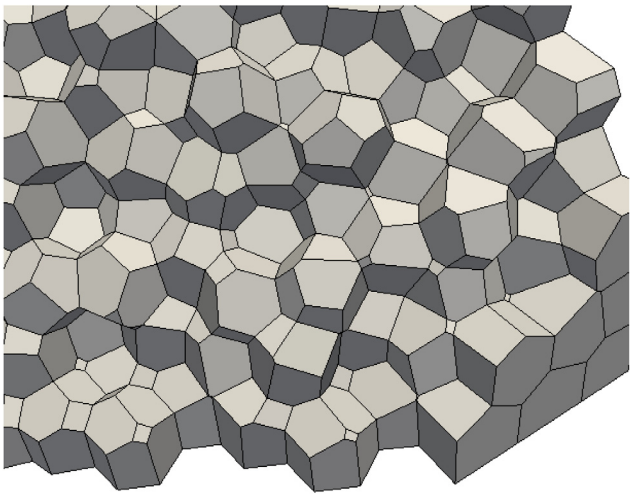

Fig. 2. A detail of one CVT mesh to underline that such kind of mesh presents some bad features such as distorted faces and small edges/faces next to big ones.

of each method, the exact solution of the problem is irrelevant. Indeed, it changes only the right hand side and it will not affect the behavior of the solver or preconditioner used.

Regarding the boundary conditions, for the Hellinger–Reissner problem we set homogeneous Neumann boundary conditions, while we consider Dirichlet boundary conditions for all the other ones.

We consider two classes of test. In the first one we make a sensitivity analysis of the diagonal preconditioner for A, i.e., B_1, and the parameter γ in the Block-Reg preconditioner. Then, in the last set of examples we make a strong scaling test among the direct solvers MUMPS and Pardiso and the iterative solvers Block-Schur and Block-Reg with the best choice of preconditioner derived by the previous test.

3.2. Test 1: choice of sub-block preconditioners

In this test, we investigate the effect of the choice of the B_1 sub-block preconditioner and of parameter γ on the performance of **Block-Schur** and **Block-Reg** preconditioners, respectively. For **Block-Schur**, we consider the following choices of B_1 sub-block preconditioners: diagonal ($diag(A)$), GAMG ($gamg(A)$) and BoomerAMG ($boom(A)$). For **Block-Reg**, we consider the following choices of γ parameter: h, h^2 and h^3, where h denotes the mesh size.

3.2.1. Maxwell equations solver

The **Cube**, **Octa** and **CVT** meshes considered consist of 46 656 elements (198 505 dofs), 30 375 elements (173 656 dofs) and 8000 elements (136 865 dofs), respectively. The results reported in Table 1 show that:

- for **Block-Schur**, the most effective choice, in terms of CPU time, of B_1 sub-block is the diagonal preconditioner, being about twice (1.5 times) as fast as the GAMG preconditioner in case of **Cube** (**CVT**) mesh. The BoomerAMG preconditioner does not converge in case of **Octa** and **CVT** meshes;
- for **Block-Reg**, the most effective choice, in terms of CPU time, of γ parameter is $\gamma = h^2$ on all meshes.

3.2.2. Mixed elliptic equations solver

The **Cube**, **Octa** and **CVT** meshes considered consist of 17 576 elements (234 573 dofs), 15 552 elements (233 281 dofs) and 8000 elements (192 861 dofs), respectively. The results reported in Table 2 show that:

- for **Block-Schur**, the most effective choice, in terms of CPU time, of B_1 sub-block is the diagonal preconditioner, being about 1.5 (3) times as fast as the GAMG preconditioner in case of **Octa** (**CVT**) mesh. Even in this case, the BoomerAMG preconditioner does not converge in case of **Octa** and **CVT** meshes;

Table 1
Choice of block B_1 and parameter γ in Block-Schur and Block-Reg preconditioners, respectively, for Maxwell equations solver. $p :=$ number of procs; nel := number of elements; T_{sol} := solution time in seconds; it := GMRES iterations; NC := not converged.

Maxwell equations solver								
Block-Schur preconditioner, p = 16								
Mesh	nel	dofs	$B_1 = diag(A)$		$B_1 = gamg(A)$		$B_1 = boom(A)$	
			it	T_{sol}	it	T_{sol}	it	T_{sol}
Cube	46 656	198 505	382	110	551	216	3863	2171
Octa	30 375	173 656	646	148	429	150	NC	–
CVT	8000	136 865	696	367	853	540	NC	–
Block-Reg preconditioner, p = 16								
Mesh	nel	dofs	$\gamma = h$		$\gamma = h^2$		$\gamma = h^3$	
			it	T_{sol}	it	T_{sol}	it	T_{sol}
Cube	46 656	198 505	1485	144	348	44	342	47
Octa	30 375	173 656	2767	301	528	71	768	101
CVT	8000	136 865	3788	494	669	120	863	157

Table 2
Choice of block B_1 and parameter γ in Block-Schur and Block-Reg preconditioners, respectively, for mixed elliptic equations solver. $p :=$ number of procs; nel := number of elements; T_{sol} := solution time in seconds; it := GMRES iterations; NC := not converged.

Mixed elliptic equations solver								
Block-Schur preconditioner, p = 16								
Mesh	nel	dofs	$B_1 = diag(A)$		$B_1 = gamg(A)$		$B_1 = boom(A)$	
			it	T_{sol}	it	T_{sol}	it	T_{sol}
Cube	17 576	234 573	66	6	39	7	68	7
Octa	15 552	233 281	79	6	48	10	NC	–
CVT	8000	192 861	142	6	72	20	NC	–
Block-Reg preconditioner, p = 16								
Mesh	nel	dofs	$\gamma = h$		$\gamma = h^2$		$\gamma = h^3$	
			it	T_{sol}	it	T_{sol}	it	T_{sol}
Cube	17 576	234 573	120	12	109	13	110	12
Octa	15 552	233 281	158	17	172	18	263	22
CVT	8000	192 861	143	25	117	24	209	28

- for **Block-Reg**, all choices γ parameter are comparable in terms of CPU time, whereas $\gamma = h^2$ seems to be the most effective in terms of GMRES iterations.

3.2.3. Stokes equations solver

The **Cube**, **Octa** and **CVT** meshes considered consist of 8000 elements (238 764 dofs), 4608 elements (166 180 dofs) and 2000 elements (154 068 dofs), respectively. The results reported in Table 3 show that:

- for **Block-Schur**, the most effective choice, in terms of CPU time, of B_1 sub-block is the diagonal preconditioner, being about 3 (4) times as fast as the GAMG preconditioner in case of **Cube** (**Octa**) mesh. The GAMG preconditioner does not converge in case of **CVT** mesh, whereas the BoomerAMG preconditioner does not converge in case of both **Octa** and **CVT** meshes;
- for **Block-Reg**, the most effective choice, in terms of both GMRES iterations and CPU time, of γ parameter is $\gamma = h$ on all meshes.

Table 3
Choice of block B_1 and parameter γ in Block-Schur and Block-Reg preconditioners, respectively, for Stokes equations solver. $p :=$ number of procs; nel := number of elements; T_{sol} := solution time in seconds; it := GMRES iterations; NC := not converged.

Stokes equations solver								
Block-Schur preconditioner, p = 16								
Mesh	nel	dofs	$B_1 = diag(A)$		$B_1 = gamg(A)$		$B_1 = boom(A)$	
			it	T_{sol}	it	T_{sol}	it	T_{sol}
Cube	8000	238 764	1793	67	789	215	1146	73
Octa	4608	166 180	1397	38	552	171	NC	–
CVT (p = 8)	2000	154 068	1262	38	NC	–	NC	–
Block-Reg preconditioner, p = 16								
Mesh	nel	dofs	$\gamma = h$		$\gamma = h^2$		$\gamma = h^3$	
			it	T_{sol}	it	T_{sol}	it	T_{sol}
Cube	8000	238 764	270	163	1652	265	3868	533
Octa	4608	166 180	441	162	2259	303	7453	749
CVT	2000	154 068	245	317	336	330	1208	460

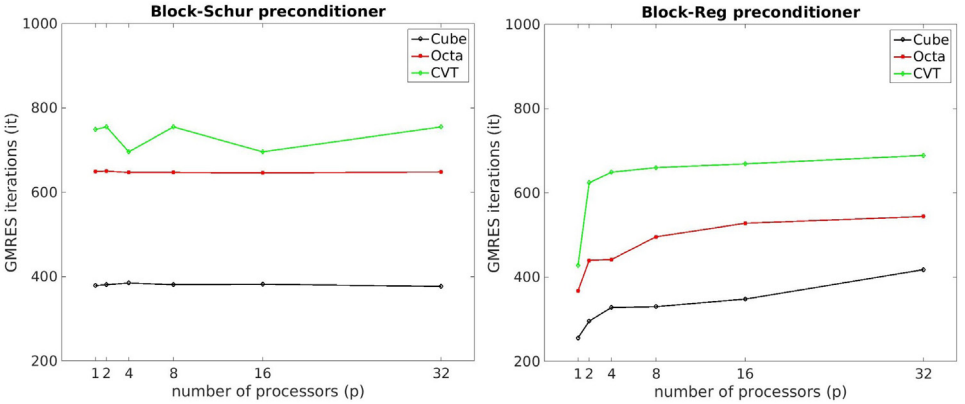

Fig. 3. Strong scaling test, Maxwell equations solver. GMRES iterations of Block-Schur (left) and Block-Reg (right) preconditioners as a function of the number of processors.

3.2.4. Hellinger–Reissner equations solver

For the sake of conciseness, we have not reported the analogous test for the Hellinger–Reissner equations, because the Block-Schur preconditioner fails on all meshes and the performance of the Block-Reg preconditioner is not significantly sensitive with respect to the three choices (h, h^2, h^3) of the γ parameter.

3.3. Test 2: strong scaling

In this strong scaling test, we investigate the performance of the two iterative solvers **Block-Schur** and **Block-Reg**, when the number of processors increase, keeping fixed the total dofs. For **Block-Schur**, we consider as B_1 sub-block the diagonal preconditioner ($diag(A)$), whereas, for **Block-Reg**, we take $\gamma = h^2$ for Maxwell, Mixed elliptic and Hellinger–Reissner equations solvers and $\gamma = h$ for Stokes. Denoting by p the number of processors, we recall that the *parallel efficiency* E_p is defined as

$$E_p := \frac{\text{CPU time with 1 procs}}{p(\text{CPU time with } p \text{ procs})}.$$

The two iterative solvers are compared in terms of CPU time with the parallel direct solvers **Mumps** and **Pardiso**.

Table 4
Strong scaling test for the Maxwell equations solver. p := number of procs; T_{ass} := assembling time in seconds; T_{sol} := solution time in seconds; it := GMRES iterations; E_p := parallel efficiency; OoM := out of memory.

Maxwell equations solver										
Cube mesh with 46 656 elements, k = 1, dofs = 198 505										
Pardiso $T_{sol} = 174$										
p	T_{ass}	E_p	Mumps		Block-Schur			Block-Reg		
			T_{sol}	E_p	it	T_{sol}	E_p	it	T_{sol}	E_p
1	58	–	3740	–	379	704	–	256	231	–
2	23	126%	1968	95%	381	411	86%	296	129	89%
4	14	104%	1173	80%	385	239	74%	328	89	65%
8	7	104%	591	79%	381	183	48%	330	54	53%
16	4	91%	352	66%	382	110	40%	348	44	33%
32	2	91%	195	60%	377	96	23%	418	29	25%
Octa mesh with 30 375 elements, k = 1, dofs = 173 656										
Pardiso $T_{sol} = 76$										
p	T_{ass}	E_p	Mumps		Block-Schur			Block-Reg		
			T_{sol}	E_p	it	T_{sol}	E_p	it	T_{sol}	E_p
1	44	–	OoM	–	649	712	–	368	334	–
2	18	122%	OoM	–	650	496	72%	440	211	79%
4	11	100%	OoM	–	647	275	65%	442	133	63%
8	6	92%	1345	–	647	247	36%	496	91	46%
16	2	137%	1077	62%	646	148	30%	528	71	29%
32	1	137%	941	36%	648	204	11%	544	181	6%
CVT mesh with 8000 elements, k = 1, dofs = 136 865										
Pardiso $T_{sol} = 516$										
p	T_{ass}	E_p	Mumps		Block-Schur			Block-Reg		
			T_{sol}	E_p	it	T_{sol}	E_p	it	T_{sol}	E_p
1	28	–	OoM	–	749	1869	–	428	467	–
2	13	108%	OoM	–	755	1093	85%	624	356	66%
4	7	100%	2717	–	696	695	67%	649	247	47%
8	3	117%	1761	77%	755	490	48%	660	161	36%
16	2	87%	1416	48%	696	367	32%	669	120	24%
32	1	87%	1070	32%	755	866	7%	689	72	20%

3.3.1. Maxwell equations solver

The **Cube**, **Octa** and **CVT** meshes considered consist of 46 656 elements (198 505 dofs), 30 375 elements (173 656 dofs) and 8000 elements (136 865 dofs), respectively. The results of the strong scaling test are reported in Table 4 and Fig. 3.

We first observe that, for all polyhedral meshes, the CPU times needed to assemble the stiffness matrix and the right hand side (T_{ass}) are completely scalable, with parallel efficiency always greater than 80%. Both **Block-Schur** and **Block-Reg** iterative solvers are scalable in terms of GMRES iterations, which seem to approach constant values when the number of processors increase, as clearly shown by the plots in Fig. Instead, in terms of CPU times the two iterative solvers present a good scalability up to 16 processors.

In case of **Cube** mesh, the most effective solver is **Block-Reg** ($p = 32$, $T_{sol} = 29$ s), being about 3, 6 and 6 times as fast as the **Block-Schur** ($p = 32$, $T_{sol} = 96$ s), **Mumps** ($p = 32$, $T_{sol} = 195$ s) and **Pardiso** ($T_{sol} = 174$ s), respectively.

In case of **Octa** mesh, the most effective solvers are **Block-Reg** ($p = 16$, $T_{sol} = 71$ s) and **Pardiso** ($T_{sol} = 76$ s), being about 2 and 13 times as fast as the **Block-Schur** ($p = 16$, $T_{sol} = 148$ s) and **Mumps** ($p = 32$, $T_{sol} = 941$ s), respectively.

Table 5
Strong scaling test for the mixed elliptic equations solver. p := number of procs; T_{ass} := assembling time in seconds; T_{sol} := solution time in seconds; it := GMRES iterations; E_p := parallel efficiency.

Mixed elliptic equations solver										
Cube with 17 576 elements, k = 1, dofs = 234 573										
Pardiso $T_{sol} = 49$										
p	T_{ass}	E_p	Mumps		Block-Schur			Block-Reg		
			T_{sol}	E_p	it	T_{sol}	E_p	it	T_{sol}	E_p
1	35	–	240	–	67	15	–	71	20	–
2	17	103%	121	99%	67	9	83%	97	18	56%
4	10	87%	66	91%	66	6	62%	101	13	38%
8	6	73%	43	70%	66	5	37%	101	11	23%
16	3	73%	26	58%	66	4	23%	109	9	14%
32	1	109%	15	50%	66	5	9%	124	9	7%
Octa mesh with 15 552 elements, k = 1, dofs = 233 281										
Pardiso $T_{sol} = 36$										
p	T_{ass}	E_p	Mumps		Block-Schur			Block-Reg		
			T_{sol}	E_p	it	T_{sol}	E_p	it	T_{sol}	E_p
1	42	–	574	–	79	14	–	72	25	–
2	19	110%	281	102%	79	8	87%	124	27	46%
4	12	87%	151	95%	79	6	58%	162	24	26%
8	6	87%	97	74%	79	5	35%	156	16	19%
16	4	66%	54	66%	79	5	17%	172	14	11%
32	2	66%	33	54%	79	5	9%	172	39	2%
CVT mesh with 8000 elements, k = 1, dofs = 192 861										
Pardiso $T_{sol} = 281$										
p	T_{ass}	E_p	Mumps		Block-Schur			Block-Reg		
			T_{sol}	E_p	it	T_{sol}	E_p	it	T_{sol}	E_p
1	71	–	2136	–	142	11	–	55	35	–
2	34	104%	1084	98%	142	8	69%	81	55	32%
4	22	81%	585	91%	142	5	55%	103	45	19%
8	12	74%	346	77%	142	5	27%	114	25	17%
16	6	74%	202	66%	142	5	14%	117	17	13%
32	3	74%	117	57%	142	5	7%	121	10	11%

In case of **CVT** mesh, the most effective solver is again **Block-Reg** ($p = 32$, $T_{sol} = 72$ s), being about 5, 14 and 7 times as fast as the **Block-Schur** ($p = 16$, $T_{sol} = 367$ s), **Mumps** ($p = 32$, $T_{sol} = 1070$ s) and **Pardiso** ($T_{sol} = 516$ s), respectively.

3.3.2. Mixed elliptic equations solver

The **Cube**, **Octa** and **CVT** meshes considered consist of 17 576 elements (234 573 dofs), 15 552 elements (233 281 dofs) and 8000 elements (192 861 dofs), respectively. The results of the strong scaling test are reported in Table 5 and Fig. 4.

For all polyhedral meshes, the assembling CPU time for the stiffness matrix and the right hand side (T_{ass}) are scalable, with parallel efficiency always greater than 65%. Both **Block-Schur** and **Block-Reg** iterative solvers are scalable in terms of GMRES iterations, as confirmed by the plots in Fig. 4. In terms of CPU times, the two iterative solvers do not exhibit a scalable behavior. This could be attributed to the fact that the linear system is not sufficiently large, and its solution is quite effective even with 1 processor. We could not test larger problems because Mumps failed with more dofs and we did not have a comparison.

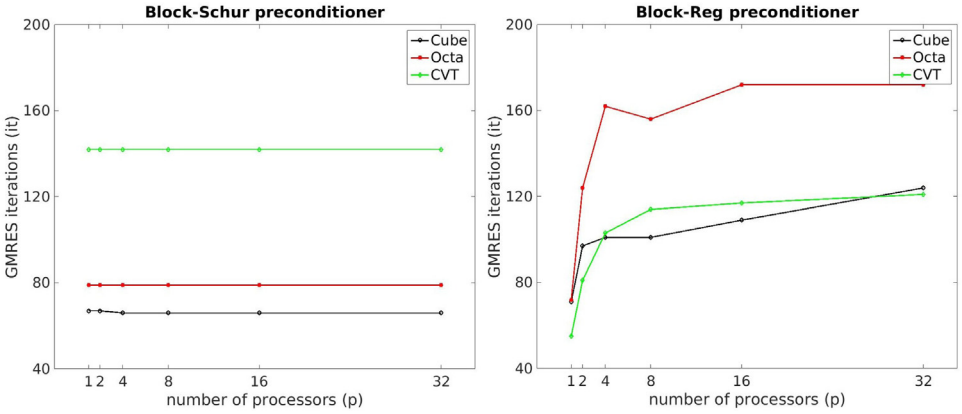

Fig. 4. Strong scaling test, mixed elliptic equations solver. GMRES iterations of Block-Schur (left) and Block-Reg (right) preconditioners as a function of the number of processors.

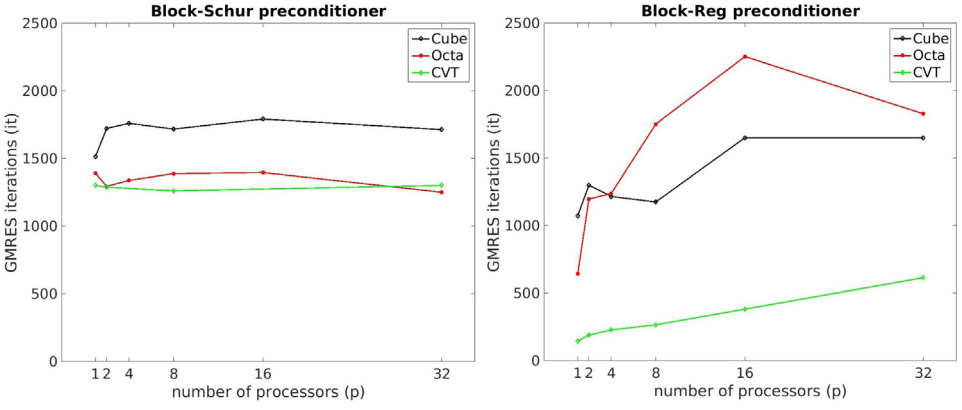

Fig. 5. Strong scaling test, Stokes equations solver. GMRES iterations of Block-Schur (left) and Block-Reg (right) preconditioners as a function of the number of processors.

In case of **Cube** mesh, the most effective solver is **Block-Schur** ($p = 16$, $T_{sol} = 4$ s), being about 2, 3 and 12 times as fast as the **Block-Reg** ($p = 16$, $T_{sol} = 9$ s), **Mumps** ($p = 32$, $T_{sol} = 15$ s) and **Pardiso** ($T_{sol} = 49$ s), respectively.

In case of **Octa** mesh, the most effective solver is **Block-Schur** ($p = 8$, $T_{sol} = 5$ s), being about 3, 6 and 7 times as fast as the **Block-Reg** ($p = 16$, $T_{sol} = 14$ s) and **Mumps** ($p = 32$, $T_{sol} = 33$ s) and **Pardiso** ($T_{sol} = 36$ s), respectively.

In case of **CVT** mesh, the most effective solver is again **Block-Schur** ($p = 4$, $T_{sol} = 5$ s), being about twice, 23 and 56 times as fast as the **Block-Reg** ($p = 32$, $T_{sol} = 10$ s), **Mumps** ($p = 32$, $T_{sol} = 117$ s) and **Pardiso** ($T_{sol} = 281$ s), respectively.

3.3.3. Stokes equations solver

The **Cube**, **Octa** and **CVT** meshes considered here consist of 8000 elements (238 764 dofs), 4608 elements (166 180 dofs) and 2000 elements (154 068 dofs), respectively. The results of the strong scaling test are reported in Table 6 and Fig. 5.

For all polyhedral meshes, the assembling CPU time for the stiffness matrix and the right hand side (T_{ass}) exhibit a very good scalability, with parallel efficiency always greater than 70%. Both **Block-Schur** and **Block-Reg** iterative solvers are scalable in terms of GMRES iterations, which seem to approach constant values when the number of processors increases, as confirmed by the plots in Fig. 5. However, iteration counts are very large, yielding a loss of

Table 6
Strong scaling test for the Stokes equations solver. p := number of procs; T_{ass} := assembling time in seconds; T_{sol} := solution time in seconds; it := GMRES iterations; E_p := parallel efficiency; NC := not converged.

Stokes equations solver										
Cube with 8000 elements, k = 2, dofs = 238 764										
Pardiso $T_{sol} = 354$										
p	T_{ass}	E_p	Mumps		Block-Schur			Block-Reg		
			T_{sol}	E_p	it	T_{sol}	E_p	it	T_{sol}	E_p
1	2481	–	2542	–	1516	157	–	163	178	–
2	1266	98%	1486	85%	1724	104	75%	220	263	34%
4	709	87%	771	82%	1761	76	52%	248	276	16%
8	367	84%	471	67%	1719	64	31%	255	266	8%
16	189	82%	271	59%	1793	63	16%	270	163	7%
32	99	78%	166	48%	1715	71	7%	294	98	6%
Octa mesh with 4608 elements, k = 2, dofs = 166 180										
Pardiso $T_{sol} = 106$										
p	T_{ass}	E_p	Mumps		Block-Schur			Block-Reg		
			T_{sol}	E_p	it	T_{sol}	E_p	it	T_{sol}	E_p
1	1857	–	466	–	1392	85	–	200	168	–
2	950	98%	332	70%	1295	58	73%	259	251	33%
4	536	87%	143	81%	1339	42	51%	343	284	15%
8	269	86%	95	61%	1389	39	27%	416	253	8%
16	137	85%	49	59%	1397	38	14%	441	162	6%
32	72	81%	33	44%	1252	455	1%	441	97	5%
CVT mesh with 2000 elements, k = 2, dofs = 154 068										
Pardiso $T_{sol} = 1557$										
p	T_{ass}	E_p	Mumps		Block-Schur			Block-Reg		
			T_{sol}	E_p	it	T_{sol}	E_p	it	T_{sol}	E_p
1	2159	–	5839	–	1303	140	–	106	276	–
2	1119	96%	3860	76%	1289	88	79%	153	665	21%
4	619	87%	1685	87%	NC	–	–	172	684	10%
8	326	83%	1134	64%	1262	38	46%	202	419	8%
16	174	77%	666	55%	NC	–	–	245	317	5%
32	95	71%	371	49%	1302	39	11%	266	184	5%

scalability in terms of CPU times. In case of **CVT** mesh, **Block-Schur** does not converge with 4 and 16 processors. Further research is needed to develop robust iterative solvers for this VEM approximation of the Stokes equations.

Nevertheless, in case of **Cube** mesh, the most effective solver is **Block-Schur** ($p = 16$, $T_{sol} = 63$ s), being about twice and 5 times as fast as the **Block-Reg** ($p = 32$, $T_{sol} = 162$ s), **Mumps** ($p = 32$, $T_{sol} = 166$ s) and **Pardiso** ($T_{sol} = 354$ s), respectively.

In case of **Octa** mesh, the performances of **Block-Schur** ($p = 16$, $T_{sol} = 38$ s) and **Mumps** ($p = 32$, $T_{sol} = 33$ s) are comparable, being about 7 times and twice as fast as the **Block-Reg** ($p = 16$, $T_{sol} = 276$ s) and **Pardiso** ($T_{sol} = 106$ s), respectively.

In case of **CVT** mesh, the most effective solver is again **Block-Schur** ($p = 8$, $T_{sol} = 38$ s), being about 4, 9 and 40 times as fast as the **Block-Reg** ($p = 32$, $T_{sol} = 182$ s), **Mumps** ($p = 32$, $T_{sol} = 371$ s) and **Pardiso** ($T_{sol} = 1557$ s), respectively.

3.3.4. Hellinger–Reissner equations solver

The **Cube** and **Octa** meshes considered here consist of 10 648 elements (264 264 dofs) and 9000 elements (253 200 dofs), respectively. The **Block-Schur** solver did not converge on any kind of polyhedral mesh, while

Table 7
Strong scaling test for the Hellinger–Reissner equations solver. $p :=$ number of procs; $T_{ass} :=$ assembling time in seconds; $T_{sol} :=$ solution time in seconds; it := GMRES iterations; $E_p :=$ parallel efficiency.

Hellinger–Reissner equations solver							
Cube with 10 648 elements, k = 1, dofs = 264 264							
Pardiso $T_{sol} = 155$							
p	T_{ass}	E_p	Mumps		Block-Reg		
			T_{sol}	E_p	it	T_{sol}	E_p
1	272	–	1484	–	2124	301	–
2	135	101%	924	80%	2124	150	100%
4	69	98%	460	81%	2124	85	88%
8	35	97%	262	71%	2124	52	72%
16	19	89%	142	65%	2124	44	43%
32	10	85%	86	54%	2124	25	38%
Octa mesh with 9000 elements, k = 1, dofs = 253 200							
Pardiso $T_{sol} = 122$							
p	T_{ass}	E_p	Mumps		Block-Reg		
			T_{sol}	E_p	it	T_{sol}	E_p
1	331	–	540	–	2255	297	–
2	174	95%	352	77%	2255	159	93%
4	93	89%	190	71%	2255	87	85%
8	49	84%	108	62%	2255	57	65%
16	28	74%	54	62%	2255	51	36%
32	15	69%	36	47%	2255	29	32%

the **Block-Reg** solver did not converge on the **CVT** meshes. The results of the strong scaling test are reported in Table 7.

As in the previous tests, for both polyhedral meshes, the assembling CPU time for the stiffness matrix and the right hand side (T_{ass}) exhibit a very good scalability, with parallel efficiency always greater than 70%. The **Block-Reg** iterative solver is scalable in terms of GMRES iterations, which remain constant when the number of processors increases. CPU times exhibit also a quite good scalability, since they reduce significantly with the growth of the number of processors.

In case of **Cube** mesh, the most effective solver is **Block-Reg** ($p = 32$, $T_{sol} = 25$ s), being about 3 and 6 as fast as **Mumps** ($p = 32$, $T_{sol} = 86$ s) and **Pardiso** ($T_{sol} = 155$ s), respectively.

In case of **Octa** mesh, the best performances of **Block-Reg** ($p = 32$, $T_{sol} = 29$ s) and **Mumps** ($p = 32$, $T_{sol} = 36$ s) are comparable, being about 4 times as fast as **Pardiso** ($T_{sol} = 122$ s).

4. Conclusions

In the present work, we have developed and studied numerically parallel block preconditioners for a set of three-dimensional saddle point problems discretized by low order virtual elements. We have restricted the investigation to the following systems of PDEs: stationary Maxwell equations in the mixed Kikuchi formulation; elliptic equations in mixed form; Stokes system; linear elasticity in the mixed Hellinger–Reissner formulation. Two block preconditioners have been proposed: one based on the approximate Schur complement (Block-Schur) and the other on a regularization technique (Block-Reg). Several numerical experiments have been conducted in parallel on a Linux cluster in order to study the performance of the iterative solvers in terms of GMRES iterations and computational time. We verify the robustness of the solvers with respect to different polyhedral meshes and the scalability of both the assembling and solution time by varying the number of processors. We have also compared the two iterative solvers with the parallel direct linear solvers Mumps and Pardiso. The results have shown that:

- in case of Maxwell equations, the most effective solver is Block-Reg, being comparable or in the range of 6–14 times as fast as the Mumps and Pardiso solvers, depending on the type of polyhedral mesh;

- in case of mixed elliptic equations, the most effective solver is Block-Schur, being in the range of 3–50 times as fast as the Mumps and Pardiso solvers, depending on the type of polyhedral mesh;
- in case of Stokes equations, the most effective solver is Block-Schur, being comparable or in the range of 2–40 times as fast as the Mumps and Pardiso solvers, depending on the type of polyhedral mesh;
- in case of Hellinger–Reissner equations, the most effective solver is Block-Reg, being in the range of 3–6 times as fast as the Mumps and Pardiso solvers, depending on the type of polyhedral mesh.

The main limitations of this study are the use of low order virtual elements and the consideration of only symmetric problems. Future research should be devoted to the construction of parallel iterative solvers robust with respect to higher order virtual element discretizations and non-symmetric problems, such as linear systems deriving from discretizations of the Navier–Stokes equations.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to acknowledge INDAM-GNCS, Italy for the support. They further thank Prof. L. Beirão da Veiga for the fruitful discussions on the subject of this paper. Moreover Franco Dassi was partially supported by the European Research Council through the H2020 Consolidator Grant (Grant No. 681162) CAVE — Challenges and Advancements in Virtual Elements. These supports are gratefully acknowledged.

References

[1] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L.D. Marini, A. Russo, Basic principles of virtual element methods, Math. Models Methods Appl. Sci. 23 (1) (2013) 199–214.
[2] L. Beirão da Veiga, F. Brezzi, L.D. Marini, A. Russo, The hitchhiker's guide to the virtual element method, Math. Models Methods Appl. Sci. 24 (08) (2014) 1541–1573.
[3] N. Bellomo, F. Brezzi, G. Manzini, Recent techniques for pde discretizations on polyhedral meshes, Math. Models Methods Appl. Sci. 24 (8) (2014) 1453–1455.
[4] L. Beirão da Veiga, A. Ern, Preface, ESAIM Math. Model. Numer. Anal. 50 (3) (2016) 633–634.
[5] L. Beirão da Veiga, A. Chernov, L. Mascotto, A. Russo, Basic principles of hp virtual elements on quasiuniform meshes, Math. Models Methods Appl. Sci. 26 (08) (2016) 1567–1598.
[6] F. Dassi, L. Mascotto, Exploring high-order three dimensional virtual elements: Bases and stabilizations, Comput. Math. Appl. 75 (9) (2018) 3379–3401.
[7] L. Beirão da Veiga, F. Brezzi, L.D. Marini, A. Russo, Mixed virtual element methods for general second order elliptic problems on polygonal meshes, ESAIM Math. Model. Numer. Anal. 50 (3) (2016) 727–747.
[8] F. Brezzi, R.S. Falk, L.D. Marini, Basic principles of mixed virtual element methods, ESAIM Math. Model. Numer. Anal. 48 (4) (2014) 1227–1240.
[9] F. Dassi, S. Scacchi, Parallel solvers for virtual element discretizations of elliptic equations in mixed form, Comput. Math. Appl. (2019).
[10] L. Beirão da Veiga, C. Lovadina, D. Mora, A virtual element method for elastic and inelastic problems on polytope meshes, Comput. Methods Appl. Mech. Engrg. 295 (2015) 327–346.
[11] E. Artioli, S. de Miranda, C. Lovadina, L. Patruno, A stress/displacement virtual element method for plane elasticity problems, Comput. Methods Appl. Mech. Engrg. 325 (2017) 155–174.
[12] F. Dassi, C. Lovadina, M. Visinoni, A three-dimensional Hellinger–Reissner virtual element method for linear elasticity problems, Comput. Methods Appl. Mech. Engrg. 364 (2020) 112910.
[13] A.L. Gain, G.H. Paulino, L.S. Duarte, I.F.M. Menezes, Topology optimization using polytopes, Comput. Methods Appl. Mech. Engrg. 293 (2015) 411–430.
[14] P.F. Antonietti, M. Bruggi, S. Scacchi, M. Verani, On the virtual element method for topology optimization on polygonal meshes: A numerical study, Comput. Math. Appl. 74 (5) (2017) 1091–1109.
[15] A. Cangiani, V. Gyrya, G. Manzini, The nonconforming virtual element method for the stokes equations, SIAM J. Numer. Anal. 54 (6) (2016) 3411–3435.
[16] L. Beirão da Veiga, C. Lovadina, G. Vacca, Divergence free virtual elements for the stokes problem on polygonal meshes, ESAIM Math. Model. Numer. Anal. 51 (2) (2017) 509–535.
[17] L. Beirão da Veiga, F. Dassi, G. Vacca, The stokes complex for virtual elements in three dimensions, Math. Models Methods Appl. Sci. 30 (3) (2020) 477–512.
[18] L. Beirão da Veiga, F. Brezzi, F. Dassi, L.D. Marini, A. Russo, Virtual element approximation of 2d magnetostatic problems, Comput. Methods Appl. Mech. Engrg. 327 (2017) 173–195.

[19] L. Beirão da Veiga, F. Brezzi, F. Dassi, L.D. Marini, A. Russo, Lowest order virtual element approximation of magnetostatic problems, Comput. Methods Appl. Mech. Engrg. 332 (2018) 343–362.
[20] L. Beirão da Veiga, F. Brezzi, F. Dassi, L. Marini, A. Russo, A family of three-dimensional virtual elements with applications to magnetostatics, SIAM J. Numer. Anal. 56 (5) (2018) 2940–2962.
[21] G. Vacca, L. Beirão da Veiga, Virtual element methods for parabolic problems on polygonal meshes, Numer. Methods Partial Differential Equations 31 (6) (2015) 2110–2134.
[22] G. Vacca, Virtual element methods for hyperbolic problems on polygonal meshes, Comput. Math. Appl. 74 (5) (2017) 882–898.
[23] P.F. Antonietti, L. Beirão da Veiga, S. Scacchi, M. Verani, A C^1 virtual element method for the Cahn-Hilliard equation with polygonal meshes, SIAM J. Numer. Anal. 54 (1) (2016) 34–57.
[24] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, 2003.
[25] W. Zulehner, Analysis of iterative methods for saddle point problems: a unified approach, Math. Comp. 71 (238) (2001) 479–506.
[26] O. Axelsson, Iterative Solution Methods, Cambridge University Press, 1994.
[27] G.H. Golub, C. Greif, On solving block-structured indefinite linear systems, SIAM J. Sci. Comput. 24 (6) (2003) 2076–2092.
[28] V. Simoncini, Block triangular preconditioners for symmetric saddle-point problems, Appl. Numer. Math. 49 (1) (2004) 63–80.
[29] M. Benzi, G.H. Golub, J. Liesen, Numerical solution of saddle point problems, Acta Numer. 14 (2005) 1–137.
[30] K.-A. Mardal, R. Winther, Preconditioning discretizations of systems of partial differential equations, Numer. Linear Algebra Appl. 18 (1) (2011) 1–40.
[31] O. Axelsson, R. Blaheta, P. Byczanski, J. Karátson, B. Ahmad, Preconditioners for regularized saddle point problems with an application for heterogeneous Darcy flow problems, J. Comput. Appl. Math. 280 (2015) 141–157.
[32] A. Klawonn, Block-triangular preconditioners for saddle point problems with a penalty term, SIAM J. Sci. Comput. 19 (1) (1998) 172–184.
[33] A. Klawonn, L.F. Pavarino, A comparison of overlapping Schwarz methods and block preconditioners for saddle point problems, Numer. Linear Algebra Appl. 7 (1) (2000) 1–25.
[34] T.P. Mathew, Schwarz alternating and iterative refinement methods for mixed formulations of elliptic problems, part I: Algorithms and numerical results, Numer. Math. 65 (1) (1993) 445–468.
[35] T.P. Mathew, Schwarz alternating and iterative refinement methods for mixed formulations of elliptic problems, part II: Convergence theory, Numer. Math. 65 (1) (1993) 469–492.
[36] L.F. Pavarino, Indefinite overlapping Schwarz methods for time-dependent Stokes problems, Comput. Methods Appl. Mech. Engrg. 187 (1–2) (2000) 35–51.
[37] L.F. Pavarino, O.B. Widlund, Balancing Neumann-Neumann methods for incompressible Stokes equations, Comm. Pure Appl. Math. 55 (3) (2001) 302–335.
[38] S. Zampini, X. Tu, Multilevel balancing domain decomposition by constraints deluxe algorithms with adaptive coarse spaces for flow in porous media, SIAM J. Sci. Comput. 39 (4) (2017) A1389–A1415.
[39] L. Mascotto, Ill-conditioning in the virtual element method: Stabilizations and bases, Numer. Methods Partial Differential Equations 34 (4) (2018) 1258–1281.
[40] S. Bertoluzza, M. Pennacchio, D. Prada, BDDC and FETI-DP for the virtual element method, Calcolo 54 (4) (2017) 1565–1593.
[41] S. Bertoluzza, M. Pennacchio, D. Prada, FETI-DP for the three dimensional virtual element method, SIAM J. Numer. Anal. 58 (3) (2020) 1556–1591.
[42] P.F. Antonietti, L. Mascotto, M. Verani, A multigrid algorithm for the p-version of the virtual element method, ESAIM: Math. Model. Numer. Anal. 52 (1) (2018) 337–364.
[43] J.G. Calvo, On the approximation of a virtual coarse space for domain decomposition methods in two dimensions, Math. Models Methods Appl. Sci. 28 (07) (2018) 1267–1289.
[44] P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, J. Koster, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM J. Matrix Anal. Appl. 23 (1) (2001) 15–41.
[45] P.R. Amestoy, A. Guermouche, J.-Y. L'Excellent, S. Pralet, Hybrid scheduling for the parallel solution of linear systems, Parallel Comput. 32 (2) (2006) 136–156.
[46] A. De Coninck, B. De Baets, D. Kourounis, F. Verbosio, O. Schenk, S. Maenhout, J. Fostier, Needles: Toward large-scale genomic prediction with marker-by-environment interaction, 203 (1) (2016) 543–555.
[47] F. Verbosio, A. De Coninck, D. Kourounis, O. Schenk, Enhancing the scalability of selected inversion factorization algorithms in genomic prediction, J. Comput. Sci. 22 (Supplement C) (2017) 99–108.
[48] D. Kourounis, A. Fuchs, O. Schenk, Towards the next generation of multiperiod optimal power flow solvers, IEEE Trans. Power Syst. PP (99) (2018) 1–10.
[49] L. Beirão da Veiga, F. Brezzi, L.D. Marini, A. Russo, H(div) and H(curl)-conforming virtual element methods, Numer. Math. 133 (2016) 303–332.
[50] S. Balay, S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, D.A. May, L.C. McInnes, R.T. Mills, T. Munson, K. Rupp, P. Sanan, B.F. Smith, S. Zampini, H. Zhang, H. Zhang, PETSc Web page, 2018, http://www.mcs.anl.gov/petsc.
[51] S. Balay, S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, D.A. May, L.C. McInnes, R.T. Mills, T. Munson, K. Rupp, P. Sanan, B.F. Smith, S. Zampini, H. Zhang, H. Zhang, PETSc Users Manual, Technical Report ANL-95/11 - Revision 3.9, Argonne National Laboratory, 2018.
[52] S. Balay, W.D. Gropp, L.C. McInnes, B.F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: E. Arge, A.M. Bruaset, H.P. Langtangen (Eds.), Modern Software Tools in Scientific Computing, Birkhäuser Press, 1997, pp. 163–202.
[53] V.E. Henson, U.M. Yang, BoomerAMG: A parallel algebraic multigrid solver and preconditioner, Appl. Numer. Math. 41 (2002) 155–177.
[54] Lawrence Livermore National Laboratory. hypre: High Performance Preconditioners.