PCPATCH: software for the topological construction of multigrid relaxation methods

PATRICK E. FARRELL, University of Oxford
MATTHEW G. KNEPLEY, University at Buffalo
LAWRENCE MITCHELL, Durham University
FLORIAN WECHSUNG, New York University

Effective relaxation methods are necessary for good multigrid convergence. For many equations, standard Jacobi and Gauß–Seidel are inadequate, and more sophisticated space decompositions are required; examples include problems with semidefinite terms or saddle point structure. In this paper we present a unifying software abstraction, PCPATCH, for the topological construction of space decompositions for multigrid relaxation methods. Space decompositions are specified by collecting topological entities in a mesh (such as all vertices or faces) and applying a construction rule (such as taking all degrees of freedom in the cells around each entity). The software is implemented in PETSc and facilitates the elegant expression of a wide range of schemes merely by varying solver options at runtime. In turn, this allows for the very rapid development of fast solvers for difficult problems.

1 INTRODUCTION

It is well known that geometric multigrid with Jacobi relaxation is an effective solver for many problems. For example, when applied to discretisations of: given a Lipschitz domain $\Omega \subset \mathbb{R}^3$, $f \in L^2(\Omega;\mathbb{R}^3)$ and $\alpha > 0$, find $u \in H^1(\Omega;\mathbb{R}^3)$ such that

$$(u, v) + \alpha(\nabla u, \nabla v) = (f, v) \text{ for all } v \in H^1(\Omega;\mathbb{R}^3), \tag{1.1}$$

geometric multigrid with Jacobi relaxation gives mesh-independent and α-robust convergence. However, for many other problems, this is not the case. For example, when applied to discretisations of: find $u \in H(\mathrm{curl}, \Omega)$ such that

$$(u, v) + \alpha(\nabla \times u, \nabla \times v) = (f, v) \text{ for all } v \in H(\mathrm{curl}, \Omega), \tag{1.2}$$

geometric multigrid with Jacobi relaxation gives neither mesh-independent nor α-robust convergence. A simple modification restores both properties: use a relaxation method that solves simultaneously for all degrees of freedom in the patch of cells around each vertex, excluding those on the boundary of the patch [Arnold et al. 2000]. The patch for the lowest-order Nédélec element of the first kind is shown in figure 1.

This same relaxation, which we refer to as a *vertex-star* iteration, arises in other contexts. It yields mesh-independent and parameter-robust convergence for the $H(\mathrm{div}, \Omega)$ and $H(\mathrm{curl}, \Omega)$ Riesz maps [Arnold et al. 1997, 2000], for nearly incompressible linear elasticity and Reissner–Mindlin plates [Schöberl 1999a,b], and for the Navier–Stokes equations [Benzi and Olshanskii 2006; Farrell et al. 2019]. When combined with a low-order solver, it yields p-independent convergence for high-order discretisations of symmetric second-order elliptic problems [Pavarino 1993].

Variants of the cellwise analogue, solving simultaneously for all degrees of freedom in a cell, have also been proposed many times in the literature [Bastian et al. 2012, 2019; Fischer 1997]. It was employed by Vanka as the relaxation in a nonlinear monolithic multigrid method for a marker-and-cell discretisation of the Navier–Stokes equations with piecewise constant pressures

Authors' addresses: Patrick E. Farrell, University of Oxford, Mathematical Institute, patrick.farrell@maths.ox.ac.uk; Matthew G. Knepley, University at Buffalo, Department of Computer Science and Engineering, knepley@gmail.com; Lawrence Mitchell, Durham University, Department of Computer Science, lawrence.mitchell@durham.ac.uk; Florian Wechsung, New York University, Courant Institute of Mathematical Sciences, wechsung@cims.nyu.edu.

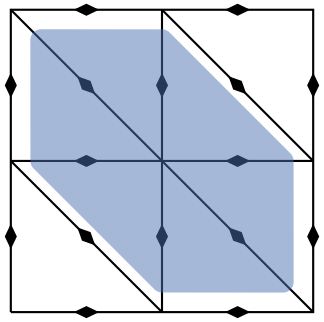

Fig. 1. Patch of cells resulting in robust relaxation for the problem of (1.2).

[Vanka 1986]. This idea can be generalised to other discretisations of saddle point problems by constructing patches that gather all degrees of freedom connected to a single degree of freedom for the Lagrange multiplier [MacLachlan and Oosterlee 2011]. For example, Vanka relaxation applied to a Taylor–Hood discretisation of the Navier–Stokes equations (with continuous piecewise linear pressures) builds patches around each vertex as shown in figure 2, including the velocity (but not pressure) degrees of freedom on the boundary of the cells around the vertex. This is a larger patch than the vertex-star.

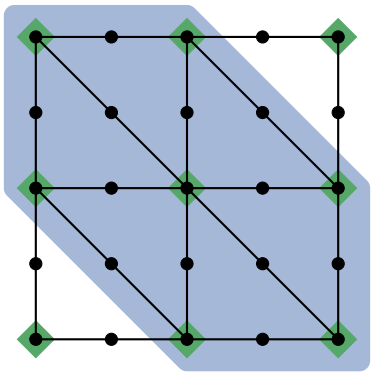

Fig. 2. Patch of cells for Vanka relaxation applied to a Taylor–Hood discretisation. The velocity degrees of freedom on the boundary of the patch are included, but the pressure degrees of freedom on the boundary are excluded.

In general, these relaxation methods can be viewed as additive or multiplicative Schwarz methods induced by a space decomposition [Xu 1992]

$$V = \sum_i V_i, \tag{1.3}$$

where V is the trial space of the discretisation and each V_i is constructed by gathering the degrees of freedom associated with a given subset of topological entities in the mesh[1]. The abstract formulation (1.3) also includes classical domain decomposition methods (when augmented with appropriate transmission conditions) [Chan and Mathew 1994; Dolean et al. 2015; Smith et al. 1996; Toselli and Widlund 2005]. Classical domain decomposition uses a relatively small number of large subspaces,

[1]By topological entity we mean a vertex, edge, face, or cell of the mesh.

whereas we are interested in making the subspaces as small as possible, while remaining effective for eliminating the relevant component of the error in a multigrid cycle.

There exists a large amount of software for classical domain decomposition [Hecht 2012; Jolivet et al. 2013; Zampini 2016], but due to this difference in patch size, the available software is not always well-suited for use as a relaxation method in a multigrid context. As a consequence, despite optimal multigrid relaxation methods being known for many hard problems, there are few (if any) general software implementations that allow easy access to them. For example, a Reynolds-robust multigrid scheme for the Navier–Stokes equations was developed in Benzi and Olshanskii [2006], but no general implementation of the full scheme was available until the work of Farrell et al. [2019] (based on PCPATCH).

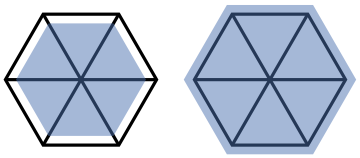

Fig. 3. The star of a vertex (left), and its closure (right).

PCPATCH constructs the subspaces as follows. Sets of topological entities are gathered according to a user-specified rule. For example, the vertex-star iteration loops over all vertices of the mesh and applies the *star operation* to gather a set of entities for each vertex. The star operation is a standard concept in algebraic topology: given a simplicial complex K, the star of a simplex p is the union of the interiors of all simplices that contain p [Munkres 1984, §2]. Concretely, the star of a vertex is the interiors of all edges, faces, and cells incidental to the vertex, along with the vertex itself, as shown on the left of figure 3. To define each subspace V_i from these sets of entities we gather all degrees of freedom supported on the entities, and set V_i to be the span of the associated basis functions.

On each V_i a small linear or non-linear problem must be solved. To do this, PCPATCH determines the cells involved in the assembly of this problem by an adjacency relation appropriate to the discretisation. For example, standard finite element assembly involving only cell integrals requires the *closure* of the entities in V_i. The closure of a set of simplices is the smallest simplicial subcomplex that contains each simplex in the set. Concretely, in the vertex-star example, this means adding all edges, vertices, and faces of cells in the set, i.e. including the boundary, as shown on the right of figure 3. We refer to this final set of entities as a *patch*. PCPATCH then invokes a user-provided callback function to assemble the matrix (and residual if nonlinear) of the problem on each cell of the patch. The local problem is then assembled using an appropriate local numbering, and solved (often with a direct factorisation, but not necessarily). The solution to the local problem results in a local update to the global solution that PCPATCH applies either additively or multiplicatively, as selected by the user.

The advantage of organising the software in this way is its great flexibility. For example, to implement Vanka relaxation for the marker-and-cell discretisation, the only change required is to specify that patches are created by iterating over cells using the closure operation to gather the entities in each patch; everything else follows automatically. To implement Vanka relaxation for Taylor–Hood, the only change required is to specify that patches are created by iterating over vertices using the closure of the star to gather entities in each patch. The callback structure separates the specification of a space decomposition from the implementation of the discretisation. Consequently, PCPATCH requires no new code to be written when attacking a problem in a new way.

The remainder of the manuscript is organised as follows. The mathematical theory of space decompositions and subspace corrections is reviewed in section 2, with a particular emphasis on achieving parameter-robust convergence. The resulting computational abstractions developed in PCPATCH are described in section 3. The software is applied to several challenging problems in section 4, demonstrating its flexibility and utility. We conclude in section 5.

2 MATHEMATICAL BACKGROUND

For ease of exposition, we restrict ourselves in this section to linear variational problems, although PCPATCH also applies to nonlinear problems. Consider the problem: find $u \in V$ such that

$$a(u, v) = (f, v) \text{ for all } v \in V, \tag{2.1}$$

where V is a finite-dimensional Hilbert space, $a : V \times V \to \mathbb{R}$ is a bilinear form, and f is right hand side data in an appropriate function space. Many algorithms for solving (2.1) are induced by a space decomposition of V into subspaces V_i with

$$V = \sum_{i=1}^{J} V_i. \tag{2.2}$$

This means that for any $v \in V$ there exists $\{v_i\}_{i=1}^{J}$ with $v = \sum_{i=1}^{J} v_i$ and $v_i \in V_i$. This decomposition is usually not unique.

There are two main solver algorithms associated with this space decomposition. The additive variant, referred to as parallel subspace correction in Xu [1992], solves for updates to the solution in each subspace independently, and is shown in algorithm 1. Given an initial guess $u^k \in V$, the associated error equation is: find $e \in V$ such that

$$a(e, v) = (f, v) - a(u^k, v) \text{ for all } v \in V. \tag{2.3}$$

If we could compute e, then $u^k + e$ would be the exact solution. The idea of parallel subspace correction is to compute an approximation δu_i to e in each subspace V_i independently by solving the Galerkin projection of (2.3) onto V_i[2]. These updates δu_i are then combined, possibly after application of a weighting operator (often a partition of unity) w_i on each subspace.

input : Initial guess $u^k \in V$
input : Weighting operators $w_i : V_i \to V_i$
output: Updated guess $u^{k+1} \in V$

for $i = 1$ **to** J **do**
  Find $\delta u_i \in V_i$ such that

$$a(\delta u_i, v_i) = (f, v_i) - a(u^k, v_i) \text{ for all } v_i \in V_i.$$

end
$u^{k+1} \leftarrow u^k + \sum_{i=1}^{J} w_i(\delta u_i)$

Algorithm 1: Parallel subspace correction

The multiplicative algorithm, referred to by Xu [1992] as sequential subspace correction, solves for updates to the solution in each subspace sequentially, and is shown in algorithm 2. The difference to parallel subspace correction is that the updates from each subspace solve are immediately applied

[2]The analysis of Xu [1992] does not require that these solves are exact. However, in this paper we only consider exact subspace solves, both to simplify the exposition and because this is typically what is done in the context of multigrid relaxation.

to the current guess, which modifies the right hand side of the error equation for the next solve. The multiplicative variant typically exhibits better convergence, but the subsolves cannot be parallelised and the residual must be updated at each step.

input : Initial guess $u^k \in V$
output: Updated guess $u^{k+1} \in V$

for $i = 1$ **to** J **do**
  Find $\delta u_i \in V_i$ such that

$$a(\delta u_i, v_i) = (f, v_i) - a(u^{k+(i-1)/J}, v_i) \text{ for all } v_i \in V_i.$$

  $u^{k+i/J} \leftarrow u^{k+(i-1)/J} + \delta u_i$
end

Algorithm 2: Sequential subspace correction

Given basis functions $\{\phi_1, \ldots, \phi_N\}$ for V, the classical Jacobi and Gauß–Seidel iterations are induced by the space decomposition

$$V = \sum_{i=1}^{N} \operatorname{span}\{\phi_i\} \tag{2.4}$$

and by the additive and multiplicative algorithms respectively.

A domain decomposition method is induced by the space decomposition

$$V = V_0 + \sum_{i=1}^{J} V_i, \tag{2.5}$$

where V_0 is a coarse space and V_i consists of functions supported in some subdomain Ω_i of the domain $\Omega \subset \mathbb{R}^d$. Typically, the updates are combined with the additive algorithm[3].

A multigrid V-cycle is induced by the multiplicative algorithm applied to the space decomposition

$$V = \sum_{l=L}^{2} V_l + V_1 + \sum_{l=2}^{L} V_l, \tag{2.6}$$

where $V_1 \subset V_2 \subset \cdots \subset V_L = V$. Typically, each V_l is constructed on the refinement of the mesh used for V_{l-1}. The updates are performed with approximate solvers for $l > 1$. In fact, the approximate solver on each level $l > 1$ is often itself a subspace correction method.

Of course, the effectiveness of these solvers depends on the space decomposition chosen. We briefly recall some of the standard theory for subspace correction methods that describes what makes for an effective space decomposition. We define the operator $A : V \to V^*$ associated with the bilinear form via

$$\langle Au, v\rangle = a(u, v) \quad \text{for all } u, v \in V, \tag{2.7}$$

where V^* is the dual space of V. For each subspace we denote the inclusion $I_i : V_i \to V$ and its adjoint $I_i^* : V^* \to V_i^*$, and we define the restriction of A to V_i by

$$\langle A_i u_i, v_i\rangle = \langle AI_i u_i, I_i v_i\rangle \quad \text{for all } u_i, v_i \in V_i, \tag{2.8}$$

[3]In the domain decomposition context, the problem solved on each subspace is often modified, for example by imposing boundary conditions other than homogeneous Dirichlet. See Gander [2006] for details.

i.e. $A_i = I_i^* A I_i$. We assume that the patch solves are performed exactly. The parallel subspace correction preconditioner associated with the decomposition $\{V_i\}$ can then be expressed as

$$D^{-1} = \sum_i I_i A_i^{-1} I_i^*. \tag{2.9}$$

Denote $T = D^{-1}A = \sum_i T_i$, where $T_i = I_i A_i^{-1} I_i^* A$. Assuming that a is symmetric, bounded, and coercive and that D^{-1} will be used as a preconditioner in the conjugate gradient algorithm, then the goal is to estimate the condition number $\kappa(T)$ bounding the convergence of the Krylov method.

Theorem 2.1. *Assume that there exist constants c_0 and c_1 such that*

$$(Tu, Tu)_A \leq c_0 (Tu, u)_A \tag{2.10}$$

and

$$\inf_{\substack{u_i \in V_i \\ \sum_i u_i = u}} \sum_i \|u_i\|_{A_i}^2 \leq c_1 \|u\|_A^2 \tag{2.11}$$

for all $u \in V$. Then

$$\lambda_{\min}(T) \geq c_1^{-1} \quad \text{and} \quad \lambda_{\max}(T) \leq c_0. \tag{2.12}$$

A proof can be found in Xu [1992, Theorem 4.1]. We briefly comment on the two conditions. The first condition measures the interaction of the subspaces V_i. A useful tool is the interaction matrix Θ with entries Θ_{ij} defined by the smallest constants satisfying

$$|(T_i u, T_j v)_A| \leq \Theta_{ij} \left[(T_i u, u)_A\right]^{1/2} \left[(T_j v, v)_A\right]^{1/2} \quad \forall u, v \in V. \tag{2.13}$$

One can show that

$$c_0 \leq \rho(\Theta) \tag{2.14}$$

where $\rho(\Theta)$ is the spectral radius of Θ (Xu [1992, Lemma 4.6]). It follows that

$$\rho(\Theta) \leq \|\Theta\|_1 = \max_j \sum_i |\Theta_{ij}| \leq N_O, \tag{2.15}$$

where N_O is the maximal number of overlapping subspaces of any one subspace. In particular, this shows that the interaction can be estimated by purely topological arguments: in the case of the vertex-star on a regular mesh as shown in figure 1, the interaction can be bounded by $c_0 \leq N_O = 7$. In general, N_O is bounded on shape regular meshes.

The constant c_1 estimates the stability of the space decomposition and the eigenvalue bound follows from the classic statement

$$\|u\|_D^2 = \inf_{\substack{u_i \in V_i \\ \sum_i u_i = u}} \sum_i \|u_i\|_{A_i}^2, \tag{2.16}$$

found in Xu [2001, eqn. (4.11)]. We illustrate the calculation of c_1 in the next section.

2.1 Parameter-robust subspace correction

The problem (2.1) to be solved often depends on a parameter $\alpha \in \mathbb{R}$. For example, the linearised Navier–Stokes operator depends on the Reynolds number, while the equations of linear elasticity depend on the Poisson ratio. It is desirable to build multigrid methods with parameter-robust convergence, i.e. the number of iterations required for convergence does not vary substantially as α is varied. A requirement for achieving this is that the relaxation method is also parameter-robust. In some important cases, the construction of parameter-robust relaxation methods can be achieved by an appropriate choice of space decomposition. In particular, for nearly singular symmetric problems and when $\alpha \geq 0$, the appropriate choice is guided by results of Schöberl [1999a,b] (for the additive case) and Lee et al. [2007] (for the multiplicative case).

Before we state the results, we consider a simple example. Let V be an H^1_0-conforming finite element space, and let

$$a(u, v) = (\nabla u, \nabla v) + \alpha(\nabla \cdot u, \nabla \cdot v), \tag{2.17}$$

for $\alpha \geq 0$. In the simple case of Jacobi smoothing, then the decomposition $u = \sum u_i$ is unique and using a standard inverse estimate we obtain

$$\begin{aligned} \|u\|_D^2 = \sum_i \|u_i\|_A^2 &\preceq (1+\alpha) \sum_i \|u_i\|_1^2 \preceq \frac{1+\alpha}{h^2} \sum_i \|u_i\|_0^2 \\ &\preceq (1+\alpha) h^{-2} \|u\|_0^2 \preceq (1+\alpha) h^{-2} \|u\|_A^2, \end{aligned} \tag{2.18}$$

and hence $c_1 \sim (1+\alpha)h^{-2}$.

We make two observations. First, the bound grows as the mesh is refined. This is well known and is the reason why Jacobi relaxation by itself is not effective but must be used in a multigrid hierarchy. Second, the bound increases with α: Jacobi relaxation is not parameter-robust. To avoid this blow-up, the space decomposition needs to *decompose the nullspace of the singular operator* that is scaled with α.

This argument was made rigorous by Schöberl [Schöberl 1999b, Theorem 4.1] for the case of the parallel subspace correction method, and a similar result also holds for the sequential subspace correction method [Lee et al. 2007, Theorem 4.2].

Theorem 2.2 (Parameter robust parallel subspace correction). *Consider a problem of the form: for $\alpha \geq 0$, find $u \in V$ such that*

$$a(u, v) = (f, v) \quad \text{for all } v \in V, \tag{2.19}$$

where

$$a(u, v) = a_0(u, v) + \alpha b(\Lambda u, \Lambda v) \tag{2.20}$$

and $\Lambda : V \to Q$ for some space Q. Assume that the mapping Λ is a continuous linear map and that b is symmetric, bounded, and coercive on Q and that a_0 is symmetric, bounded, and coercive on V. Denote the kernel by

$$\mathcal{N} = \{u \in V : b(\Lambda u, q) = 0 \, \forall \, q \in Q\}. \tag{2.21}$$

A space decomposition

$$V = \sum_i V_i \tag{2.22}$$

defines a subspace correction method that is robust with respect to α if in addition to (2.10) and (2.11) the decomposition satisfies the following properties:

(1) the pair $V \times Q$ is inf-sup stable for the mixed problem induced by

$$B((u, p), (v, q)) = a_0(u, v) - b(\Lambda u, q) - b(\Lambda v, p), \tag{2.23}$$

(2) the splitting is stable for the V-norm:

$$\inf_{\substack{u_i \in V_i \\ \sum_i u_i = u}} \sum_i \|u_i\|_V^2 \leq c_1 \|u\|_V^2 \quad \text{for all } u \in V \tag{2.24}$$

for some c_1,

(3) the splitting is stable in the operator norm on the kernel $\mathcal{N}$:

$$\inf_{\substack{u_i \in V_i \cap \mathcal{N} \\ \sum_i u_i = u}} \sum_i \|u_i\|_{A_i}^2 \leq c_2 \|u\|_A^2 \quad \text{for all } u \in V \cap \mathcal{N} \tag{2.25}$$

for some α-independent c_2.

We highlight that in particular the condition (2.25) requires that

$$\mathcal{N} = \sum_i (V_i \cap \mathcal{N}). \tag{2.26}$$

In other words, a crucial property for such nearly singular systems is that any kernel function can be written as a sum of kernel functions in the subspaces.

We give an example of the construction of subspaces $\{V_i\}$ so that (2.26) is satisfied for the $H(\mathrm{curl})$ Riesz map in three dimensions (1.2). We consider the de Rham complex and the Whitney subcomplex

$$\begin{array}{ccccccc} H^1 & \xrightarrow{\text{grad}} & H(\mathrm{curl}) & \xrightarrow{\text{curl}} & H(\mathrm{div}) & \xrightarrow{\text{div}} & L^2 \\ \downarrow & & \downarrow & & \downarrow & & \downarrow \\ \Sigma & \xrightarrow{\text{grad}} & V & \xrightarrow{\text{curl}} & W & \xrightarrow{\text{div}} & Q \end{array}. \tag{2.27}$$

Here Σ is the familiar space of continuous Lagrange finite elements of order k, V is the Nédélec finite element of the first kind of order k, W is the Raviart–Thomas element of order k, and Q is the space of discontinuous Lagrange finite elements of order $k-1$. This sequence is exact on simply-connected domains, and hence for every $u \in H(\mathrm{curl})$ with $\nabla \times u = 0$, we can find a $\Phi \in H^1$ such that $\nabla\Phi = u$. This property carries over to the discrete complex: for $u \in V$ with $\nabla \times u = 0$ there exists a $\Phi \in \Sigma$ with $\nabla\Phi = u$. Given a basis $\{\Phi_j\}$ for Σ, we can write $\Phi = \sum_j \alpha_j \Phi_j$ and defining

$$u_j = \alpha_i \nabla \Phi_j, \tag{2.28}$$

we observe that

$$\sum_j u_j = u \quad \text{and} \quad \nabla \times u_j = 0 \text{ for all } j. \tag{2.29}$$

We now need to find subspaces $\{V_i\}$ such that for every u_j there exists a V_i with $u_j \in V_i$. To do this, we examine the support of the basis functions. Since the standard finite element basis for Σ has basis functions supported in the stars of vertices, the same holds for each u_j, as it is simply the gradient of such a function. Consequently, defining V_i to be the functions supported in the star of vertex i gives the desired kernel capturing property (2.26).

Obtaining the bound in (2.25) is harder and requires bounds on the potential Φ in terms of u. We note that such bounds are usually known for the infinite-dimensional sequence and also hold for the discrete sequence when commuting projections exist. For more detail on multigrid relaxation methods derived from exterior calculus we refer to Arnold et al. [2006].

2.2 Nonlinear relaxation

The idea of subspace correction also applies to the nonlinear case. Consider the problem: find $u \in V$ such that

$$F(u; v) = 0 \text{ for all } v \in V, \tag{2.30}$$

where $F : V \times V \to \mathbb{R}$ is the residual. Given an initial guess u^k, the associated error equation is to find $e \in V$ such that $F(u^k + e; v) = 0$ for all $v \in V$. The associated additive algorithm that computes approximations to e in each subspace is given in algorithm 3. Initial theoretical results on this algorithm were established by Dryja and Hackbusch [1997]. This algorithm is often used as a nonlinear preconditioner for an outer Newton method [Brune et al. 2015; Cai and Keyes 2002].

We note that in the context of finite elements the residual F usually has a local nature and when testing with functions in a patch spanning the space V_i, only the part of u^k that overlaps with

input : Initial guess $u^k \in V$
input : Weighting operators $w_i : V_i \to V_i$
output: Updated guess $u^{k+1} \in V$

for $i = 1$ **to** J **do**
  Find $\delta u_i \in V_i$ such that
$$F(u^k + \delta u_i; v_i) = 0 \text{ for all } v_i \in V_i.$$
end
$u^{k+1} \leftarrow u^k + \sum_{i=1}^{J} w_i(\delta u_i)$

Algorithm 3: Parallel subspace correction for the nonlinear problem (2.30)

V_i is relevant. To formalise this property we assume that there exist subspaces $\overline{V_i} \supset V_i$, injection operators $\iota_i : V \to \overline{V_i}$, and local residuals $F_i : \overline{V_i} \to V_i^*$ such that

$$F(u^k + \delta u_i; v_i) = F_i(\iota(u^k) + \delta u_i; v_i) \quad \text{for all } v_i \in V_i,\ \delta u_i \in V_i. \tag{2.31}$$

This means that the local Newton solvers do not need to know the full state, but only the relevant part of the state given by $\iota_i(u^k)$; this is important for parallelisation. As a consequence, in the implementation we solve for $\delta u_i \in V_i$ such that

$$F_i(\iota_i(u^k) + \delta u_i; v_i) = 0 \text{ for all } v_i \in V_i. \tag{2.32}$$

3 COMPUTATIONAL ABSTRACTIONS

Even though mathematical formulations of parameter robust multigrid relaxation schemes have been known for several decades, implementations of these schemes have been missing from general purpose linear algebra and discretisation packages such as PETSc [Balay et al. 2019], Trilinos [Heroux et al. 2005], deal.II [Bangerth et al. 2007], DUNE [Blatt et al. 2016], Firedrake [Rathgeber et al. 2016], and FEniCS [Logg and Wells 2010; Logg et al. 2012], due to the difficulty of defining the space decomposition and resulting local problems in a general and composable way. A popular algebraic approach for implementing subspace correction methods, such as provided by PETSc in `PCASM`, is to obtain index sets that define the space decomposition and construct the local operators algebraically by extracting submatrices from the global assembled matrix. NGSolve [Schöberl 2014] also supports this algebraic approach, augmented with the ability to obtain the index sets through topological queries [NGSolve 2017]. Although successful, using such an interface to construct multigrid relaxation schemes has several disadvantages. It does not allow for matrix-free implementations (which can offer significant efficiency advantages at high order) or efficient nonlinear relaxation (since each local nonlinear update would require the assembly of the global Jacobian). In light of these issues, we take a different approach: separating the topological decomposition and subproblem assembly into different stages, and using a callback interface that we now describe.

`PCPATCH` separates the assembly of subproblems on patches into three stages. First we decompose the mesh into patches of entities whose degrees of freedom will be updated simultaneously in each subproblem. For this we use PETSc's `DMPLEX` mesh abstraction [Knepley and Karpeev 2005, 2009; Lange et al. 2015], which gives us a simple dimension-independent topological query language. For example, it offers the star and closure operations that often arise when describing patches. Second, we gather the degrees of freedom in the function space associated to the entities in each patch and build local numberings for restrictions of the function space to the patch. Third, these

numberings are supplied to callbacks to a discretisation engine[4] to conduct the assembly of the subproblem (Jacobian and possibly residual) on the patch. The callbacks receive the current local state, a list of entities in the global mesh to assemble over, and the degree of freedom mappings for the local function space. The C signatures of these callbacks are shown in listing 1. By separating the assembly of patches in this way, the abstraction is robust to the type of mesh and element employed, the PDE being solved, and the global numbering of degrees of freedom.

```
int ComputeJacobian(PC pc, PetscInt point, Vec state, Mat J, IS cells,
  PetscInt ndof, PetscInt *dofNumbering, PetscInt *dofNumberingWithBoundary);

int ComputeResidual(PC pc, PetscInt point, Vec state, Vec F, IS cells,
  PetscInt ndof, PetscInt *dofNumbering, PetscInt *dofNumberingWithBoundary);
```

Listing 1. Function signatures of the Jacobian and residual callbacks for assembly of subproblems on each patch.

The data structures representing the local function spaces are deliberately kept lightweight. In particular, no mesh object representing the subdomain is built. For example, information about which facets are on the boundary of the patch is not stored. This is in contrast to the domain decomposition approach used in the high-level FreeFem++ library [Hecht 2012], which does build fully-fledged meshes on each subdomain. Building submeshes facilitates code re-use (no callback interface is required) and is efficient when $O(1)$ subdomains per MPI process are employed, but is computationally expensive in time and memory in the multigrid context where many small subdomains are expected. The strategy we use is to mark the patch entities within the larger mesh. The discretisation engine can use this information to query the global mesh if necessary. This allows rapid assembly without the overhead of object creation while still allowing us access to topological queries so that we can, for example, construct the patch boundary on the fly.

Once each patch of entities defining the degrees of freedom to be updated in a patch solve is constructed, we additionally gather all entities in the stencil of the operator to be assembled on the patch. We term this step *patch completion*. For example, if the operator involves integration over cells of the mesh, we extend the patch to include all entities which lie in the closure of the cells we integrate over. If the operator contains facet terms, the patch completion is more involved, gathering all entities in the closure of the support of facets in the patch. This process is illustrated in figure 4.

For the solution of the subproblems on each patch, we use a PETSc KSP (for linear problems) and SNES (for nonlinear problems). This enables full flexibility in configuring the patch solves, for example the use of inexact inverses using iterative methods. For the common case where the patch is small and an exact inverse is desired, we implement a small amount of special-case code to explicitly form the inverse on each patch in setup and then solve the subproblems by matrix-vector multiplication. This offers substantial speedups for many problems.

The numberings required on each patch depend on the solution algorithm employed. In all cases, we build a numbering for the degrees of freedom that are updated with each patch solve. This excludes all degrees of freedom on the boundary of the completed patch. In the multiplicative case, an additional numbering is necessary to describe the degrees of freedom in the global residual that are updated with each patch solve. This includes degrees of freedom on the boundary of the completed patch that are not fixed by global Dirichlet conditions. Finally, in the nonlinear case, we

[4]Currently Firedrake and PETSc's PetscDS are supported; interfacing to other discretisation engines is possible.

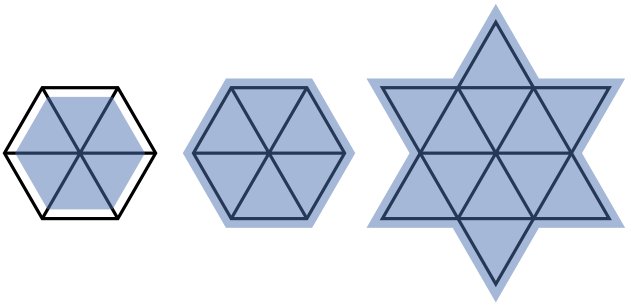

Fig. 4. Vertex-star patch (left), its completion when the problem requires only cell integrals (centre), and completion when facet integrals are also present (right).

also require a numbering for the state vector on the patch. This includes all degrees of freedom in the completed patch, including those subject to global Dirichlet conditions. Only those numberings necessary for the solution algorithm employed are constructed. In the PETSc DS implementation, a Section numbering object is created for each patch in the same way that a Section is used for the global function [Balay et al. 2019, §10.2].

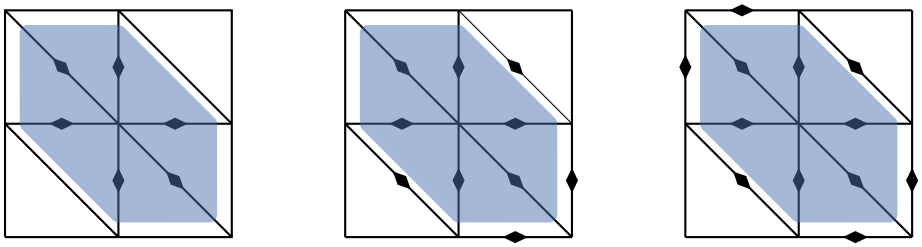

Fig. 5. Illustration of which degrees of freedom appear in each numbering for the vertex-star patch of figure 1, with global Dirichlet conditions applied on the top and left of the domain. A numbering for the degrees of freedom to be updated (left) is always required; if multiplicative updates are selected, degrees of freedom on the patch boundary that are not on the global boundary (center) must also be numbered; finally, for nonlinear problems, a numbering for all degrees of freedom supported on the patch is built (right).

PETSc as a whole, and DMPLEX in particular, are well-suited to parallel computations. The topological mesh is domain-decomposed across participating parallel processes. With a good decomposition of the mesh, parallel load balance is obtained and the data structures have been used with excellent scalability at $O(10000 - 100000)$ processes [Hapla et al. 2020; Parsani et al. 2021]. With this domain-decomposition to hand, assembly of residuals and Jacobians is naturally parallel across processes.

Globally multiplicative updates require frequent synchronisation between the parallel subdomains to communicate updated residuals. As a consequence, in parallel PCPATCH offers either additive or multiplicative relaxation within a subdomain, but only additive updates between subdomains. If multiplicative updates within a subdomain are chosen, as the number of processes is increased the convergence will approach that of the additive mode. This has the consequence that additive local updates should be performed if convergence rates independent of the parallel decomposition are desired.

We require that the mesh is decomposed with enough overlap such that any patch problem can be constructed entirely locally. This choice avoids excessive communication when assembling patches.

DMPLEX conveniently allows for the specification of the appropriate overlap when distributing the mesh [Lange et al. 2015]. However, this strategy implies a performance tradeoff: larger patches may give better convergence, but they cause an increase in the sizes of messages sent over the network due to increased parallel overlap. Currently, the user must arrange that the mesh decomposition has enough overlap for the patch specification before the application of PCPATCH. A future enhancement will use the patch specification to define the required overlap and automatically expand the mesh overlap as necessary using the existing overlap distribution algorithm provided by DMPLEX.

PCPATCH offers two mechanisms for the user to configure the space decomposition. In the basic case, the user specifies the dimension (or codimension) of entities to iterate over and a patch construction type. In the more advanced case, the user provides a callback that explicitly enumerates the entities that each patch contains. For example, the vertex-star space decomposition of figure 1 is obtained by specifying a `star` construction type around dimension-0 entities (vertices). PCPATCH presently provides three pre-configured construction types: `star`, which gathers entities the topological star of the given iteration entity; `vanka`, which gathers entities in the closure of the star of the given iteration entity; and `pardecomp`, which creates a single patch per parallel subdomain containing all locally visible entities. The callback mechanism can implement more complex patches, such as line- and plane-smoothers for advection-dominated problems, or space decompositions induced by a particular mesh structure such as Alfeld (barycentric) or Powell–Sabin refinement. This callback has access to the full DMPLEX object and can therefore construct arbitrary space decompositions, subject to the constraint that each patch must be local to a single parallel process.

4 APPLICATIONS

4.1 $H(\mathrm{div}, \Omega)$ and $H(\mathrm{curl}, \Omega)$ Riesz maps

Consider the following problem: for a bounded Lipschitz domain $\Omega \subset \mathbb{R}^3$, $\alpha > 0$ and $f \in (H(\mathrm{div}, \Omega))^*$, find $u \in H(\mathrm{div}, \Omega)$ such that

$$u - \alpha \nabla \nabla \cdot u = f \quad \text{in } \Omega, \tag{4.1a}$$

$$\nabla \cdot u = 0 \quad \text{on } \partial\Omega. \tag{4.1b}$$

For $\alpha = 1$, this is the application of the $H(\mathrm{div}, \Omega)$ Riesz map. Similarly, the application of the $H(\mathrm{curl}, \Omega)$ Riesz map entails solving the problem: for $f \in (H(\mathrm{curl}, \Omega))^*$ and $\alpha = 1$, find $u \in H(\mathrm{curl}, \Omega)$ such that

$$u + \alpha \nabla \times \nabla \times u = f \quad \text{in } \Omega, \tag{4.2a}$$

$$\nabla \times u = 0 \quad \text{on } \partial\Omega. \tag{4.2b}$$

These equations often arise as subproblems in the construction of fast preconditioners for more complex systems involving solution variables in $H(\mathrm{div}, \Omega)$ and $H(\mathrm{curl}, \Omega)$ [Mardal and Winther 2011]. However, the operators $I - \alpha\nabla\nabla\cdot$ and $I + \alpha\nabla\times\nabla\times$ are not elliptic due to the infinite-dimensional kernels of the divergence and curl operators, and hence standard relaxation schemes such as Jacobi and Gauß–Seidel iteration do not yield effective multigrid schemes. This can be overcome by employing relaxation schemes where the space decomposition does capture the kernel, as in (2.26). Arnold, Falk, and Winther proved that the star iteration constructed around vertices does satisfy (2.26) and hence yields a mesh-independent and α-robust multigrid scheme for both problems [Arnold et al. 2000]. The key solver options for expressing the vertex-star relaxation scheme are given in listing 2. A complete Firedrake code for solving this problem can be found in appendix A. All calculations for the examples in this section were performed on 28 cores of a 28-core Intel Xeon Gold 5120 CPU @2.20GHz workstation with 192GB of RAM running Ubuntu 18.04.3.

```
solver_parameters = {
    ...
    "ksp_type": "cg",
    "pc_type": "mg",
    "mg_levels_ksp_type": "richardson",
    "mg_levels_pc_type": "python",
    "mg_levels_pc_python_type": "firedrake.PatchPC",
    "mg_levels_patch_pc_patch_local_type": "additive",
    "mg_levels_ksp_richardson_scale": 1/3,
    "mg_levels_patch_pc_patch_construct_type": "star",
    "mg_levels_patch_pc_patch_construct_dim": 0,
    ...
}
```

Listing 2. Solver options for PCPATCH to implement a damped additive vertex-star iteration.

# refinements	# degrees of freedom	α				
		10^0	10^1	10^2	10^3	10^4
1	1.26×10^4	14	14	14	14	14
2	9.84×10^4	15	15	15	15	15
3	7.78×10^5	15	16	16	16	16
4	6.18×10^6	16	16	16	16	16

Table 1. Number of conjugate gradient iterations to solve the Raviart–Thomas discretisation of (4.1) using multigrid and the vertex-star relaxation.

We present results for the vertex-star relaxation applied to the lowest-order $H(\mathrm{div}, \Omega)$-conforming Raviart–Thomas discretisation of (4.1) with varying mesh refinement and α in table 1. The domain and data were given by $\Omega = (0, 2)^3$ and $f = \big(2yz(1 - x^2), 2xz(1 - y^2), 2xy(1 - z^2)\big)$. The solver employed the conjugate gradient method preconditioned by a full multigrid cycle, with the relaxation applied additively with a damping of $1/3$. The coarse grid had $5 \times 5 \times 5$ elements and the solver was deemed to have converged when the preconditioned residual norm had decreased by ten orders of magnitude. As expected from the theory of Arnold et al., the solver enjoys both mesh-independence and α-robustness.

For the $H(\mathrm{div}, \Omega)$ problem (4.1), Arnold et al. also prove robustness of the edge-star relaxation. This can be expressed using the option `"mg_levels_patch_pc_patch_construct_dim": 1`. Similarly robust results were observed for this space decomposition, applied additively with a damping of $1/4$ (not shown). However, the absolute convergence was poorer, with typically 40–47 Krylov iterations required for convergence instead of 14–16.

The results for the analogous experiment with vertex-star relaxation applied to the lowest-order $H(\mathrm{curl}, \Omega)$-conforming discretisation of (4.2) with Nédélec elements of the first kind are presented in table 2. The same domain, data, and solver were employed, except that the additive vertex-star relaxation was applied with a damping of $1/2$. The solver is again mesh-independent and α-robust. For this problem the edge-star relaxation is not effective (in fact it coincides with standard Jacobi or Gauß–Seidel).

# refinements	# degrees of freedom	α				
		10^0	10^1	10^2	10^3	10^4
1	7.93×10^3	18	19	19	19	19
2	5.97×10^4	19	19	19	19	19
3	4.63×10^5	20	20	20	20	20
4	3.64×10^6	20	20	20	20	20

Table 2. Number of conjugate gradient iterations to solve the Nédélec discretisation of (4.2) using multigrid and the vertex-star relaxation.

The vertex-star relaxation also yields mesh independent and α-robust results for higher-order discretisations, as well as for Brezzi–Douglas–Marini elements and Nédélec elements of the second kind.

4.2 Nearly incompressible linear elasticity

The equations of linear elasticity in the isotropic homogeneous case can be written as: given $\Omega \subset \mathbb{R}^d$ and $f \in \left(H_0^1(\Omega;\mathbb{R}^d)\right)^*$, find $u \in H_0^1(\Omega;\mathbb{R}^d)$ such that

$$-\mu\nabla \cdot E(u) - \gamma\nabla\nabla \cdot u = f \quad \text{in } \Omega, \tag{4.3a}$$

where $E(u) = \frac{1}{2}\left(\nabla u + \nabla u^T\right)$. Here μ and γ are positive real parameters describing the material in question. The difficult case is when the material is nearly incompressible, with $\gamma \to \infty$. As guided by the abstract theory, the central task is to choose a space decomposition $V_h = \sum_i V_i$ that captures the kernel of the divergence operator. For $H^1(\Omega;\mathbb{R}^d)$-conforming Lagrange elements this is a subtle question that depends on the dimension d and polynomial degree k; for brevity we only consider the case $d = 2$ here.

An important insight is gained by studying the de Rham complex in two dimensions

$$\mathbb{R} \to H^2(\Omega) \xrightarrow{\nabla^\perp} H^1(\Omega;\mathbb{R}^d) \xrightarrow{\nabla\cdot} L^2(\Omega) \to 0. \tag{4.4}$$

If Ω is simply connected then this de Rham complex is exact, i.e. every divergence-free function in $H^1(\Omega;\mathbb{R}^d)$ is the rotated gradient of a function in $H^2(\Omega)$. The existence of a suitable space decomposition for degree k vector fields thus hinges on the existence of a local basis for $C^1(\Omega)$-conforming degree $k+1$ scalar fields. An important result of Morgan and Scott [1975] guarantees the existence of a local basis that is captured by a vertex-star iteration for $k \geq 4$. This vertex-star iteration can be easily implemented with `PCPATCH` via the options given in listing 2.

We consider the continuous Lagrange discretisation of (4.3) with $\Omega = (0,1)^2$, $f = (1,1)$, $\mu = 1$ and γ varying from 10^0 to 10^4. The degree of the polynomials employed on each cell k was varied from $k = 1,\ldots,6$. The problem was solved with the conjugate gradient method preconditioned by a full multigrid cycle and vertex-star relaxation with damping of $1/3$. The base mesh was a 10×10 grid and five levels of mesh refinement were used for all problems.

The results are presented in table 3. The results are strikingly different for $k < 4$ and $k \geq 4$. For $k < 4$, the solver exhibits strong γ-dependence, with the iteration counts blowing up as γ increases. However, for $k \geq 4$, the iteration counts are γ-robust. This is exactly as one would expect from the results of Morgan and Scott, which guarantee that the star iteration captures the kernel in the sense of (2.26) for $k \geq 4$. This is a striking illustration of the sharpness of the abstract theory.

k	# degrees of freedom	γ				
		10^0	10^1	10^2	10^3	10^4
1	2.06×10^5	32	68	189	541	> 1000
2	8.22×10^5	15	28	74	256	> 1000
3	1.85×10^6	12	18	37	102	310
4	3.28×10^6	10	13	16	16	16
5	5.13×10^6	9	11	14	14	13
6	7.38×10^6	9	11	12	12	11

Table 3. Number of conjugate gradient iterations to solve the continuous Lagrange discretisation of (4.3) using full multigrid cycles and additive vertex-star relaxation. For $k \geq 4$ the vertex-star iteration captures the kernel of the divergence operator, while for $k < 4$ it does not.

4.3 Stokes equations

We consider the incompressible Newtonian Stokes equations: given $\mu > 0$, a bounded Lipschitz domain $\Omega \subset \mathbb{R}^d$, $f \in \left(H^1(\Omega; \mathbb{R}^d)\right)^*$ and $g \in H^{1/2}(\partial\Omega_D; \mathbb{R}^d)$, find the velocity and pressure $(u, p) \in V \times Q := H^1(\Omega; \mathbb{R}^d) \times L^2(\Omega)$ such that

$$-\nabla \cdot 2\mu E(u) + \nabla p = f \quad \text{in } \Omega, \tag{4.5a}$$

$$\nabla \cdot u = 0 \quad \text{in } \Omega, \tag{4.5b}$$

$$u = g \quad \text{on } \partial\Omega_D, \tag{4.5c}$$

$$(-pI + \mu E(u)) \cdot n = 0 \quad \text{on } \partial\Omega_N, \tag{4.5d}$$

where n is the outward unit normal to $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N$, and I is the $d \times d$ identity matrix. If $|\partial\Omega_N| = 0$ then the pressure is only defined up to a constant and the pressure trial space $Q = L^2_0(\Omega)$ is used instead. The pressure acts as a Lagrange multiplier for enforcing the divergence constraint. This equation is a fundamental problem in fluid mechanics and a great many discretisations and solvers have been proposed for it [Brandt and Livne 2011; Elman et al. 2014; Turek 1999]. In this section we discuss the implementation of a monolithic multigrid method with Vanka relaxation [Vanka 1986].

Monolithic multigrid methods apply the multigrid philosophy to the entire block-structured problem (i.e. solving for velocity and pressure together). It is therefore necessary to develop appropriate relaxation methods that dampen the error of the coupled problem. Note that Jacobi or Gauß–Seidel would not work for an inf-sup stable discretisation of (4.5), as the saddle point structure means that there are zero entries on the main diagonal. One popular strategy for implementing appropriate relaxation methods for saddle point problems is Vanka relaxation, where patches are defined by gathering all degrees of freedom connected to a single degree of freedom of the Lagrange multiplier [MacLachlan and Oosterlee 2011].

We consider the classical regularised lid-driven cavity benchmark [Elman et al. 2014, §3.1.3], discretised using the inf-sup stable CG2-CG1 Taylor–Hood discretisation. As shown in figure 2, the associated Vanka relaxation is defined by the closure of the star around each vertex (as the pressure degrees of freedom are located at vertices). The key solver options are given in listing 3. The solver employed GMRES as the outer Krylov solver preconditioned by multigrid V-cycles, using two iterations of Chebyshev-accelerated Vanka relaxation on each multigrid level. The pressure nullspace was dealt with by explicitly passing a basis for the nullspace to the Krylov method (in this case, the vector of constant pressures). We highlight the option

```
solver_parameters = {
    ...
    "ksp_type": "gmres",
    "pc_type": "mg",
    "mg_levels_ksp_type": "chebyshev",
    "mg_levels_ksp_max_it": 2,
    "mg_levels_pc_type": "python",
    "mg_levels_pc_python_type": "firedrake.PatchPC",
    "mg_levels_patch_pc_patch_local_type": "additive",
    "mg_levels_patch_pc_patch_partition_of_unity": False,
    "mg_levels_patch_pc_patch_construct_type": "vanka",
    "mg_levels_patch_pc_patch_construct_dim": 0,
    "mg_levels_patch_pc_patch_exclude_subspaces": "1",
    ...
}
```

Listing 3. Solver options for PCPATCH to implement Vanka relaxation for a Taylor–Hood discretisation.

`"mg_levels_patch_pc_patch_exclude_subspaces": "1"`; this excludes from the patch pressure degrees of freedom (in the subspace indexed by 1) other than that at the vertex around which the patch is built, ensuring that each patch contains exactly one pressure degree of freedom. Without this, the patch would include the pressures at other vertices connected by an edge to the base vertex. The base grid was a uniform 20×20 grid of quadrilaterals. The solver was deemed to have converged when the Euclidean norm of the residual had decreased by ten orders of magnitude. The results are shown in table 4. The solver enjoys mesh independence and ν-robustness. However,

# refinements	# degrees of freedom	ν				
		10^0	10^1	10^2	10^3	10^4
1	1.48×10^4	14	14	14	14	14
2	5.84×10^4	14	14	14	14	14
3	2.32×10^5	14	14	14	14	14
4	9.25×10^5	14	14	14	14	14

Table 4. Number of GMRES iterations to solve the Taylor–Hood discretisation of (4.5) using multigrid and Vanka relaxation.

these properties do not hold for Vanka relaxation applied to the Navier–Stokes equations [Turek 1999] and a more complex algorithm must be used to achieve ν-robustness in this case [Benzi and Olshanskii 2006; Farrell et al. 2019].

4.4 Semilinear Allen–Cahn equation

As a final example, we consider the semilinear Allen–Cahn equation: given $\Omega \subset \mathbb{R}^d$ and $f \in H^{-1}$, find $u \in H^1_0(\Omega)$ such that

$$-\nabla^2 u + u^3 - u = f. \tag{4.6}$$

We consider full approximation scheme (FAS) nonlinear multigrid methods for high-order discretisations of this problem [Brandt and Livne 2011]. While PETSc has had an implementation of

FAS since 2011, no general nonlinear relaxation methods were available; users had to implement by hand a custom nonlinear Gauß–Seidel algorithm on a case-by-case basis. This deficiency is remedied by SNESPATCH, the nonlinear analogue of PCPATCH.

As with any multigrid scheme, the choice of appropriate relaxation is key. When nonlinear problems are solved with Newton's method on each patch, the residual and Jacobian on the patch must be calculated at each iteration. If many patches are present this assembly cost can be quite expensive [Brabazon et al. 2014]. We therefore compare two different relaxation methods: a nonlinear star iteration[5], and an overlapping Schwarz iteration induced by the parallel decomposition [Dryja and Hackbusch 1997]. Specifically, in the latter scheme a nonlinear problem is solved independently on each core, and the updates averaged on the overlap, using Newton's method and LU factorisation as the inner solver. The overlap is of closure-star type, i.e. the degrees of freedom on the closure of the star of all entities that the process owns are solved for.

We consider a continuous Lagrange discretisation of varying degree p of (4.6). We employ quadrilateral elements to take advantage of sum factorisation [Homolya et al. 2017] and tensor-product Gauß–Legendre–Lobatto quadrature rules [Karniadakis and Sherwin 2005] for efficiency at high-order.

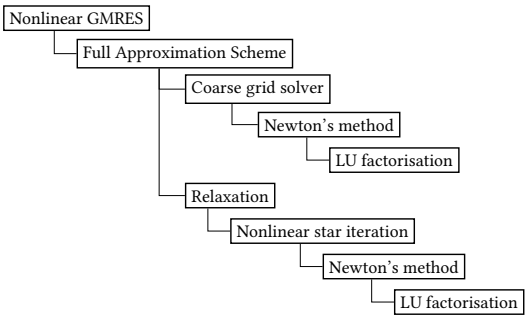

Fig. 6. An outline of the NGMRES/FAS/star algorithm for solving (4.6).

In both solvers, we employ cascadic multigrid cycles [Bornemann and Deuflhard 1996], using Newton's method with direct factorisations for both the coarse grid problem and all patch solves. No global matrix is assembled or solved, only small local or coarse problems. All relaxation updates are performed with partition of unity weighting. In both cases we use FAS as a nonlinear right preconditioner for a nonlinear GMRES iteration, analogous to preconditioning linear GMRES using a standard multigrid scheme [Brune et al. 2015; Oosterlee and Washio 2000]. For illustration, a diagram of the solver with star relaxation is given in figure 6.

For reference, we also compare to a standard Newton-Krylov scheme where the linearised Jacobians are solved with FGMRES, preconditioned by matrix-free cascadic multigrid cycles with two Chebyshev-accelerated Jacobi sweeps as relaxation. This latter scheme does not use PCPATCH; the diagonal of the matrix is assembled directly. The solves were performed with 4 multigrid refinements of a uniform 20×20 base grid. The results are shown in table 5.

For $p \leq 5$, FAS with large patches is more efficient than FAS with small patches; for larger p, FAS with star relaxation is substantially more efficient. However, in all cases the schemes based on FAS are not competitive with Newton–Krylov for this problem. We expect that the code would be even

[5]Since a star iteration and pointwise Jacobi must assemble over the same cells for each nonlinear iteration, it makes sense to update all degrees of freedom in the star at the same time.

p	# degrees of freedom	Solver type		
		NGMRES/FAS/star	NGMRES/FAS/pardecomp	Newton/FGMRES/MG/Jacobi
3	9.24×10^5	6.5/5	4.0/3	1.5/2
4	1.64×10^6	8.6/4	6.2/2	2.2/2
5	2.56×10^6	11.6/4	8.7/1	4.0/2
6	3.69×10^6	13.9/3	17.1/1	6.3/2
7	5.02×10^6	19.5/3	27.5/1	9.6/2
8	6.56×10^6	21.9/2	45.0/1	12.6/2
9	8.30×10^6	32.2/2	71.3/1	18.3/2
10	1.02×10^7	42.5/2	115.7/1	24.5/2

Table 5. Time in seconds/number of outer nonlinear iterations for three different solver strategies applied to discretisations of varying degree p of (4.6) using multigrid and Vanka relaxation.

faster if nonlinear star iteration were used as a nonlinear preconditioner for Newton's method [Cai and Keyes 2002].

5 CONCLUSION

We have presented PCPATCH, a multigrid relaxation framework whose software abstractions are motivated by the abstract subspace corrections framework of Xu [1992]. By using a callback, rather than algebraic, interface PCPATCH supports nonlinear and matrix-free relaxation in the same generic way. By maintaining topological information in multigrid hierarchies, and utilising a simple topological query language, it offers a flexible and extensible interface to a broad range of optimal relaxation methods. We have demonstrated its application for the easy development of efficient multigrid solvers for both linear and non-linear problems.

CODE AVAILABILITY

For reproducibility, we cite archives of the exact software versions used to produce the results in this paper. All major Firedrake components as well as the code used to obtain the shown iteration counts and runtimes have been archived on Zenodo [Firedrake-Zenodo 2019]. An installation of Firedrake with components matching those used to produce the results in this paper can by obtained following the instructions at https://www.firedrakeproject.org/download.html.

ACKNOWLEDGMENTS

This work is supported by the Engineering and Physical Sciences Research Council under grants EP/R029423/1, EP/V001493/1, and EP/L015803/1. LM also acknowledges support from the UK Fluids Network [EPSRC grant number EP/N032861/1] for funding a visit to Oxford. MGK acknowledges partial funding for this work from U.S. Department of Energy under grant DE-AC02-06CH11357.

A COMPLETE CODE LISTING FOR THE $H(\mathrm{div}, \Omega)$ RIESZ MAP

```
from firedrake import *

# Create the base mesh and multigrid hierarchy
N = 5
nref = 2
distribution_parameters = {
```

```
    "partition": True, "overlap_type": (DistributedMeshOverlapType.VERTEX, 1)
}
# Discretise Ω = (0,2)^3
base = BoxMesh(N, N, N, 2, 2, 2, distribution_parameters=distribution_parameters)
mh = MeshHierarchy(base, nref)
mesh = mh[-1]

# Declare Raviart-Thomas function space
V = FunctionSpace(mesh, "Raviart-Thomas", degree=1)

# Set up the PDE
u = Function(V)
v = TestFunction(V)
(x, y, z) = SpatialCoordinate(mesh)
f = as_vector([2*y*z*(1-x**2), 2*x*z*(1-y**2), 2*x*y*(1-x**2)])
alpha = Constant(1000)
F = inner(u, v)*dx + alpha*inner(div(u), div(v))*dx - inner(f, v)*dx

# Specify solver options
solver_parameters = {
    "mat_type": "matfree",
    "snes_type": "ksponly",
    "ksp_type": "cg",
    "ksp_max_it": 100,
    "ksp_rtol": 1.0e-10,
    "ksp_atol": 0.0,
    "ksp_monitor_true_residual": None,
    "pc_type": "mg",
    "pc_mg_type": "full",
    "mg_levels_ksp_type": "richardson",
    "mg_levels_ksp_richardson_scale": 1/3,
    "mg_levels_ksp_max_it": 1,
    "mg_levels_ksp_convergence_test": "skip",
    "mg_levels_pc_type": "python",
    "mg_levels_pc_python_type": "firedrake.PatchPC",
    "mg_levels_patch_pc_patch_save_operators": True,
    "mg_levels_patch_pc_patch_construct_type": "star",
    "mg_levels_patch_pc_patch_construct_dim": 0,
    "mg_levels_patch_pc_patch_sub_mat_type": "seqdense",
    "mg_levels_patch_sub_ksp_type": "preonly",
    "mg_levels_patch_sub_pc_type": "lu",
    "mg_coarse_pc_type": "python",
    "mg_coarse_pc_python_type": "firedrake.AssembledPC",
    "mg_coarse_assembled_pc_type": "lu",
    "mg_coarse_assembled_pc_factor_mat_solver_type": "mumps",
}
```

```
# Solve the problem
solve(F == 0, u, solver_parameters=solver_parameters)
```

REFERENCES

D. N. Arnold, R. Falk, and R. Winther. 1997. Preconditioning in $H(\mathrm{div})$ and applications. *Math. Comp.* 66, 219 (1997), 957–984. https://doi.org/10.1090/S0025-5718-97-00826-0

Douglas N. Arnold, Richard S. Falk, and Ragnar Winther. 2000. Multigrid in $H(\mathrm{div})$ and $H(\mathrm{curl})$. *Numer. Math.* 85, 2 (2000), 197–217. https://doi.org/10.1007/pl00005386

Douglas N. Arnold, Richard S. Falk, and Ragnar Winther. 2006. Finite element exterior calculus, homological techniques, and applications. *Acta Numerica* 15 (2006), 1–155. https://doi.org/10.1017/S0962492906210018

Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dmitry Karpeyev, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. 2019. *PETSc Users Manual.* Technical Report ANL-95/11 - Revision 3.12. Argonne National Laboratory.

W. Bangerth, R. Hartmann, and G. Kanschat. 2007. deal.II – a General Purpose Object Oriented Finite Element Library. *ACM Trans. Math. Software* 33, 4 (2007), 24/1–24/27.

Peter Bastian, Markus Blatt, and Robert Scheichl. 2012. Algebraic multigrid for discontinuous Galerkin discretizations of heterogeneous elliptic problems. *Numerical Linear Algebra with Applications* 19, 2 (2012), 367–388. https://doi.org/10.1002/nla.1816

Peter Bastian, Eike H. Müller, Steffen Müthing, and Marian Piatkowski. 2019. Matrix-free multigrid block-preconditioners for higher order Discontinuous Galerkin discretisations. *J. Comput. Phys.* 394, 1 (2019), 417–439. https://doi.org/10.1016/j.jcp.2019.06.001 arXiv:1805.11930 [math.NA]

M. Benzi and M. A. Olshanskii. 2006. An augmented Lagrangian-based approach to the Oseen problem. *SIAM Journal on Scientific Computing* 28, 6 (2006), 2095–2113. https://doi.org/10.1137/050646421

M. Blatt, A. Burchardt, A. Dedner, Ch. Engwer, J. Fahlke, B. Flemisch, Ch. Gersbacher, C. Gräser, F. Gruber, Ch. Grüninger, D. Kempf, R. Klöfkorn, T. Malkmus, S. Müthing, M. Nolte, M. Piatkowski, and O. Sander. 2016. The Distributed and Unified Numerics Environment, Version 2.4. *Archive of Numerical Software* 4, 100 (2016), 13–29. https://doi.org/10.11588/ans.2016.100.26526

F. A. Bornemann and P. Deuflhard. 1996. The cascadic multigrid method for elliptic problems. *Numer. Math.* 75, 2 (1996), 135–152. https://doi.org/10.1007/s002110050234

K.J. Brabazon, M.E. Hubbard, and P.K. Jimack. 2014. Nonlinear multigrid methods for second order differential operators with nonlinear diffusion coefficient. *Computers & Mathematics with Applications* 68, 12 (2014), 1619–1634. https://doi.org/10.1016/j.camwa.2014.11.002

Achi Brandt and Oren E. Livne. 2011. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics* (2nd ed.). Classics in Applied Mathematics, Vol. 67. SIAM.

P. R. Brune, M. G. Knepley, B. F. Smith, and X. Tu. 2015. Composing scalable nonlinear algebraic solvers. *SIAM Rev.* 57, 4 (2015), 535–565. https://doi.org/10.1137/130936725

X.-C. Cai and D. E. Keyes. 2002. Nonlinearly preconditioned inexact Newton algorithms. *SIAM Journal on Scientific Computing* 24, 1 (2002), 183–200. https://doi.org/10.1137/S106482750037620X

T. F. Chan and T. P. Mathew. 1994. Domain decomposition algorithms. *Acta Numerica* 3 (1994), 61–143. https://doi.org/10.1017/s0962492900002427

V. Dolean, P. Jolivet, and F. Nataf. 2015. *An Introduction to Domain Decomposition Methods.* SIAM. https://doi.org/10.1137/1.9781611974065

M. Dryja and W. Hackbusch. 1997. On the nonlinear domain decomposition method. *BIT Numerical Mathematics* 37, 2 (1997), 296–311. https://doi.org/10.1007/bf02510214

Howard C. Elman, David J. Silvester, and Andrew J. Wathen. 2014. *Finite Elements and Fast Iterative Solvers: with applications in incompressible fluid dynamics* (2nd ed.). Oxford University Press.

Patrick E. Farrell, Lawrence Mitchell, and Florian Wechsung. 2019. An augmented Lagrangian preconditioner for the 3D stationary incompressible Navier–Stokes equations at high Reynolds number. *SIAM Journal on Scientific Computing* 41 (2019), A3073–A3096. Issue 5. https://doi.org/10.1137/18M1219370 arXiv:1810.03315 [math.NA]

Firedrake-Zenodo. 2019. Software used in 'PCPATCH: software for the topological construction of multigrid relaxation methods'. https://doi.org/10.5281/ZENODO.3552611

Paul F. Fischer. 1997. An Overlapping Schwarz Method for Spectral Element Solution of the Incompressible Navier–Stokes Equations. *J. Comput. Phys.* 133, 1 (1997), 84 – 101. https://doi.org/10.1006/jcph.1997.5651

M. Gander. 2006. Optimized Schwarz Methods. *SIAM J. Numer. Anal.* 44, 2 (2006), 699–731. https://doi.org/10.1137/S0036142903425409

Vaclav Hapla, Matthew G. Knepley, Michael Afanasiev, Christian Boehm, Martin van Driel, Lion Krischer, and Andreas Fichtner. 2020. Fully Parallel Mesh I/O using PETSc DMPlex with an Application to Waveform Modeling. *SIAM Journal on Scientific Computing* (2020). arXiv:2004.08729 [cs.MS] To appear.

F. Hecht. 2012. New development in freefem++. *Journal of Numerical Mathematics* 20, 3-4 (2012). https://doi.org/10.1515/jnum-2012-0013

Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. 2005. An overview of the Trilinos project. *ACM Trans. Math. Softw.* 31, 3 (2005), 397–423. https://doi.org/10.1145/1089014.1089021

M. Homolya, R. C. Kirby, and D. A. Ham. 2017. Exposing and exploiting structure: optimal code generation for high-order finite element methods. arxiv:1711.02473.

P. Jolivet, F. Hecht, F. Nataf, and C. Prud'homme. 2013. Scalable domain decomposition preconditioners for heterogeneous elliptic problems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '13*. https://doi.org/10.1145/2503210.2503212

G. E. Karniadakis and S. J. Sherwin. 2005. *Spectral/hp Element Methods for Computational Fluid Dynamics* (second ed.). Oxford University Press.

Matthew G. Knepley and Dmitry A. Karpeev. 2005. *Flexible representation of computational meshes.* Technical Report ANL/MCS-P1295-1005. Argonne National Laboratory.

Matthew G. Knepley and Dmitry A. Karpeev. 2009. Mesh Algorithms for PDE with Sieve I: Mesh Distribution. *Scientific Programming* 17, 3 (2009), 215–230. https://doi.org/10.1155/2009/948613

Michael Lange, Matthew G. Knepley, and Gerard J. Gorman. 2015. Flexible, Scalable Mesh and Data Management Using PETSc DMPlex. In *Proceedings of the 3rd International Conference on Exascale Applications and Software* (Edinburgh, UK) *(EASC '15)*. University of Edinburgh, Edinburgh, Scotland, UK, 71–76. arXiv:1505.04633 [cs.MS] http://dl.acm.org/citation.cfm?id=2820083.2820097

Young-Ju Lee, Jinbiao Wu, Jinchao Xu, and Ludmil Zikatanov. 2007. Robust subspace correction methods for nearly singular systems. *Mathematical Models and Methods in Applied Sciences* 17, 11 (2007), 1937–1963. https://doi.org/10.1142/s0218202507002522

Anders Logg and Garth N. Wells. 2010. DOLFIN: Automated Finite Element Computing. *ACM Trans. Math. Softw.* 37, 2, Article 20 (April 2010), 28 pages. https://doi.org/10.1145/1731022.1731030

Anders Logg, Garth N. Wells, and Johan Hake. 2012. DOLFIN: a C++/Python finite element library. In *Automated Solution of Differential Equations by the Finite Element Method*, Anders Logg, Kent-Andre Mardal, and Garth Wells (Eds.). Lecture Notes in Computational Science and Engineering, Vol. 84. Springer Berlin Heidelberg, 173–225. https://doi.org/10.1007/978-3-642-23099-8_10

S. P. MacLachlan and C. W. Oosterlee. 2011. Local Fourier analysis for multigrid with overlapping smoothers applied to systems of PDEs. *Numerical Linear Algebra with Applications* 18, 4 (2011), 751–774. https://doi.org/10.1002/nla.762

Kent-Andre Mardal and Ragnar Winther. 2011. Preconditioning discretizations of systems of partial differential equations. *Numerical Linear Algebra with Applications* 18, 1 (2011), 1–40. https://doi.org/10.1002/nla.716

John Morgan and L. Ridgway Scott. 1975. A nodal basis for C^1 piecewise polynomials of degree $n \geq 5$. *Math. Comp.* 29, 131 (1975), 736. https://doi.org/10.2307/2005284

James R. Munkres. 1984. *Elements of Algebraic Topology.* CRC Press.

NGSolve 2017. Building blocks for programming preconditioners. https://ngsolve.org/docu/latest/i-tutorials/unit-2.1.2-blockjacobi/blockjacobi.html#A-Block-Jacobi-preconditioner (last accessed: 2019/10/21).

C. W. Oosterlee and T. Washio. 2000. Krylov Subspace Acceleration of Nonlinear Multigrid with Application to Recirculating Flows. *SIAM Journal on Scientific Computing* 21, 5 (2000), 1670–1690. https://doi.org/10.1137/s1064827598338093

Matteo Parsani, Radouan Boukharfane, Irving Reyna Nolasco, David C. Del Rey Fernández, Stefano Zampini, Bilel Hadri, and Lisandro Dalcin. 2021. High-order accurate entropy-stable discontinuous collocated Galerkin methods with the summation-by-parts property for compressible CFD frameworks: Scalable SSDC algorithms and flow solver. *J. Comput. Phys.* 424 (2021), 109844. https://doi.org/10.1016/j.jcp.2020.109844

L. F. Pavarino. 1993. Additive Schwarz methods for the p-version finite element method. *Numer. Math.* 66, 1 (1993), 493–515. https://doi.org/10.1007/bf01385709

Florian Rathgeber, David A. Ham, Lawrence Mitchell, Michael Lange, Fabio Luporini, Andrew T. T. McRae, Gheorghe-Teodor Bercea, Graham R. Markall, and Paul H. J. Kelly. 2016. Firedrake: automating the finite element method by composing abstractions. *ACM Trans. Math. Software* 43, 3 (2016), 24:1–24:27. https://doi.org/10.1145/2998441 arXiv:1501.01809

J. Schöberl. 1999a. Multigrid methods for a parameter dependent problem in primal variables. *Numer. Math.* 84, 1 (1999), 97–119. https://doi.org/10.1007/s002110050465

J. Schöberl. 1999b. *Robust Multigrid Methods for Parameter Dependent Problems.* Ph.D. Dissertation. Johannes Kepler Universität Linz, Linz, Austria.

Joachim Schöberl. 2014. *C++ implementation of finite elements in NGSolve.* Technical Report ASC Report 30/2014. Vienna University of Technology.

B. Smith, P. Bjorstad, and W. Gropp. 1996. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press.

A. Toselli and Olof B. Widlund. 2005. *Domain Decomposition Methods — Algorithms and Theory.* Springer Berlin Heidelberg. https://doi.org/10.1007/b137868

Stefan Turek. 1999. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach.* Springer.

Surya P. Vanka. 1986. Block-implicit multigrid solution of Navier-Stokes equations in primitive variables. *J. Comput. Phys.* 65, 1 (1986), 138–158. https://doi.org/10.1016/0021-9991(86)90008-2

Jinchao Xu. 1992. Iterative methods by space decomposition and subspace correction. *SIAM Rev.* 34, 4 (1992), 581–613. https://doi.org/10.1137/1034116

Jinchao Xu. 2001. The method of subspace corrections. *J. Comput. Appl. Math.* 128, 1 (2001), 335–362. https://doi.org/10.1016/S0377-0427(00)00518-5

S. Zampini. 2016. PCBDDC: A Class of Robust Dual-Primal Methods in PETSc. *SIAM Journal on Scientific Computing* 38, 5 (2016), S282–S306. https://doi.org/10.1137/15m1025785