Science and Technology Facilities Council

Time Stepping Techniques and Preconditioning for Hyperbolic and Anisotropic Elliptic Problems: Numerical Results

Technical Report 2060049-TN-03

Hussam Al Daas[*] Niall Bootland[*] Tyrone Rees[*] Sue Thorne[†]

September 2022

1 Introduction

Exascale targeted plasma modelling will require the efficient and accurate solution of systems of hyperbolic partial differential equations (PDEs), and corresponding elliptic problems, in the presence of highly anisotropic dynamics. The techniques used to solve such problems must scale with the computing power available and be robust enough to be portable to any emerging hardware.

Due to their high arithmetic intensity, high order finite elements/spectral element codes generally employ matrix-free techniques and are one of the few methods that allow the data movement to be limited to a level that would be amenable to exascale computing. For that reason, in this report we focus on linear systems arising from such discretisations. It is important that these systems are solved matrix-free, as the matrices arising from high order finite elements are significantly denser than their low order counterparts; this limits the choices of preconditioner available. Furthermore, preconditioners that are extremely successful for low order finite elements usually do not give satisfactory performance out-of-the-box as the polynomial degree increases.

In this report, we take the results of our literature and software review [13] and examine the methods of interest on a range of test problems that have been designed to incorporate the challenges that will be present when simulating plasma within a tokamak.

We first describe, in Section 2, the software test platform we use for our computations. In Section 3, we compare preconditioning strategies for highly anisotropic elliptic problems, where anisotropy is introduced within the problem definition, the underlying mesh and a combination of both. We then move to consider stationary hyperbolic problems in Section 4, using our findings when looking at high order discretisations of elliptic problems to inform the choice of preconditioners we test. Time-dependant hyperbolic problems are covered in Section 5 where we investigate various implicit time-stepping schemes and how they perform on several different examples, along with preconditioning for these problems. Finally, we draw conclusions in Section 6.

2 Test platform

The majority of our tests were run on STFC Hartree Centre's Scafell Pike, which is a Bull Sequana X1000 supercomputer. We ran our test problems on Scafell Pike's 'Regular' compute nodes, composed of 2x Intel Xeon Gold E5-6142 v5 (also known as Skylake) 16 core 2.6GHz (up to 3.7GHz). For these nodes, the system has a total of 192 GB of RAM across the 27,000 cores. The system uses a Mellanox EDR Infiniband interconnect. Scafell Pike uses IBM's LSF scheduler: we requested exclusive access to the nodes being used for our test runs. In our results, we provide "wall clock" times in seconds.

The majority of our test problems were run using MFEM with PETSC. In Table 1, we provide the software versions and compilation details.

[*]Scientific Computing Department, STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, OX11 0QX, UK.
[†]Hartree Centre, STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, OX11 0QX, UK. Email contact: sue.thorne@stfc.ac.uk

Software	Version	Compiler	Flags
MFEM	4.4	intel 2022	`-O3 -fp-model fast -mavx2`
PETSc	3.17	intel 2022	`-O3 -fp-model fast -mavx2`
Intel MPI	2022	intel 2022	default

Table 1: Software version, compiler and flags used for MFEM-based numerical examples.

3 Highly anisotropic elliptic problems

We first consider solving elliptic problems, which are PDEs without real characteristic surfaces, and thus have an infinite speed of propagation. As such, these problems typically model steady-state phenomena, which may correspond to long-time behaviour after transients have dissipated, or equilibrium solutions of hyperbolic problems. Further, elliptic equations tend to have smooth solutions and so are ideally suited to the use of high order finite element/spectral element methods (FEM/SEM) for their approximation.

The numerical solution of such problems requires the solution of a large linear system of equations,

$$A\mathbf{x} = \mathbf{b}, \tag{1}$$

where A is an $n \times n$ matrix, and $\mathbf{x}$ and $\mathbf{b}$ are n-vectors. Naive methods for solving such systems take $\mathcal{O}(n^3)$ operations and, hence, are infeasible for large problems. A large proportion of the work in a finite element simulation is in solving this linear system, and so it is vital that a robust and scalable method is chosen; identifying such a method is the target of this report. Typically, for such large linear systems, much of this work is in finding an effective preconditioner [36].

The presence of anisotropy, where properties depend upon the direction in which they are being measured, in the equations considered in fusion is an additional challenge. We will investigate anisotropy in two guises:

1. Anisotropy intrinsic to the model through physical parameters in the problem. Namely, through the diffusion coefficient, which can vary highly in different directions.

2. Anisotropy stemming from how we discretise the model. In particular, we consider that the meshes upon which we approximate the solution can be highly stretched in one direction compared to another. While such meshes may be inadvisable from an approximation theory perspective, in practice the constraints imposed by geometric and physical considerations along with ease of use mean such aspect ratios can arise in challenging applications such as fusion modelling.

Anisotropic problems can cause particular challenges for numerical solvers, where the scales involved can vary by many orders of magnitude within a single problem and result in a lack of robustness for many standard solution techniques. We will see this later in our numerical results.

In our study, we will consider both 2D and 3D highly anisotropic elliptic problems. For spatial dimension $d = 2$ or 3, let $\Omega = (0,1)^d$ and Γ be the boundary of Ω. We consider the model problem

$$-\nabla \cdot \alpha(\boldsymbol{x})\nabla u = f, \quad \boldsymbol{x} \in \Omega, \tag{2a}$$

$$u = 0, \quad \boldsymbol{x} \in \Gamma. \tag{2b}$$

This Poisson problem represents diffusion within a box. Following Xu and Zikatanov [37], we take the diffusion coefficient α to be a diagonal matrix with differing scales depending on directionality. For example, for $d = 2$ we choose

$$\alpha(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 \\ 0 & \epsilon \end{bmatrix} \tag{3}$$

while, for $d = 3$, we choose

$$\alpha(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

where $\epsilon > 0$ is a parameter which can be made increasingly small.

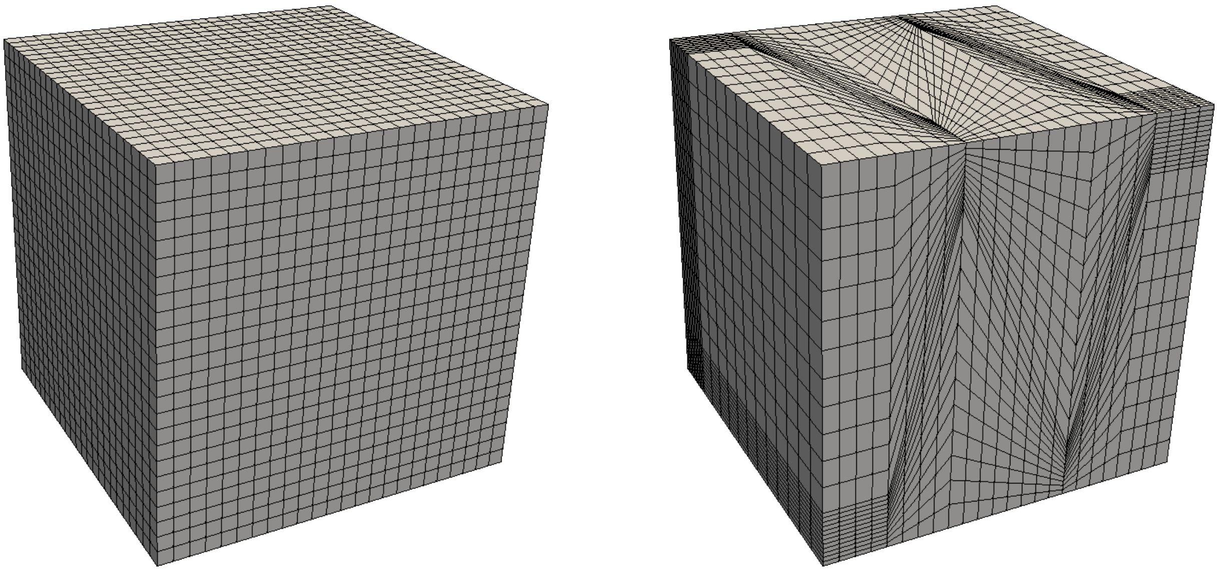
Figure 1: 3D Kershaw mesh with $\epsilon_y = \epsilon_z = 1$ (left) and $\epsilon_y = \epsilon_z = 0.3$ (right).

Following the suggestion of Farrell, we also solve this problem on anisotropic Kershaw meshes [24, 25]. In Figure 1, we show a 3D Kershaw mesh for $\epsilon_y = \epsilon_z = 1$ and $\epsilon_y = \epsilon_z = 0.3$, where decreasing the values of ϵ_y and ϵ_z from 1 will introduce anisotropy in the y and z dimension meshes, respectively. The case $\epsilon_y = \epsilon_z = 0.3$ is considered "hard" in [25].

We remark that the primary challenges stem, firstly, from the high order discretisation and, secondly, the anisotropy present. In considering preconditioning, we focus on approaches to tackle the use of high-order FEM and explore how they perform in the presence of anisotropy. Typically, a solver will first seek to ameliorate the intrinsic difficulties of the high order discretisation, reducing it to a lower order problem, and then apply a more well-established approach to treat this problem. We will consider two types of such high order solvers along with the option of tackling the high order system directly with standard low order solvers.

All of the solvers we consider below will be applied as preconditioners for a Krylov subspace method. For symmetric positive definite (SPD) matrices with an SPD preconditioner, the well-known Conjugate Gradient (CG) algorithm [20] is the method of choice, due to its efficient application via a three-term recurrence, and well-understood convergence properties. For non-symmetric linear systems or SPD linear systems with a non-SPD preconditioner, we use the Generalised Minimal Residual (GMRES) method [31]. Other Krylov methods are available; see, e.g., Greenbaum [17] for an introduction.

3.1 Preconditioning for elliptic problems

We first give a brief overview of the methods that we will compare in this section. For a more detailed comparison of the state of the art, including alternative methods, see the earlier survey reports [3, 5].

3.1.1 (Algebraic) Multigrid

Multigrid methods [10] are often amongst the most favourable methods used to solve elliptic problems, and can be highly efficient for solving large-scale problems. As the name suggests, multigrid methods are based around a hierarchy of multiple grids. Such methods exploit the fact that the high-frequency components of the error are damped effectively after only a few applications of a local relaxation method (for instance, classical iterative methods such as Jacobi or Gauss-Seidel), while low-frequency components are relatively smooth and so can be well-approximated on a coarser grid. The local relaxation method is often termed the "smoother" for this reason. By projecting onto a coarser grid, some of the original low-frequency components become high frequency on this grid and so can be treated by the smoother with low-frequency components then able to be transferred to a yet coarser mesh. Recursion on a hierarchy of grids yields a multilevel iterative approach. Since these grids are a geometric construction, typically given by a nested set of grids where one is a refinement of the next, such an approach is called geometric multigrid (GMG).

Where available, geometric multigrid methods can be very powerful tools. However, they rely on a hierarchy of grids being given or easily constructed. In practice, defining and realising such a set of grids may be difficult or impractical, especially in complex domains where meshing in itself is challenging. To this end, algebraic multigrid (AMG) methods [30, 37] have been developed which aim to provide a more black-box solution method, avoiding the requirement for a predefined grid structure. These methods have been generalised in many ways using heuristic arguments aimed at improving efficiency and robustness. Modern software implementations of AMG are complex but have been shown to provide efficient black-box solvers for certain types of problems, especially those stemming from second or fourth-order elliptic PDEs.

One of the most widely used sets of AMG solvers is available through software library `HYPRE`, developed at Lawrence Livermore National Laboratory, and in particular for our tests we will investigate BoomerAMG [19]. Note that this tackles the high order system directly, without aiming to reduce it to a lower order problem first.

3.1.2 p-multigrid

The above discussion on multigrid details the classical approach, which can be thought of as reducing the number of DOFs on each grid by increasing the mesh spacing h; thus these are h-multigrid methods. For higher order FEM, another possibility for reducing DOFs is to coarsen by reducing the polynomial degree, rather than the size of the mesh: this is known as p-multigrid. Note that a clear nested set of discretisations is given by simply reducing the polynomial degree on the same mesh, alleviating the geometric difficulties arising with h-multigrid. The familiar multigrid recipe can be followed by smoothing, coarsening in p, and repeating typically until the lowest order discretisation (here $p = 1$) which can be solved via standard approaches, such as AMG, before propagating back up to the original problem.

We make use of the p-multigrid method available in the MFEM implementation of CEED BPS3 [25], which uses Chebyshev smoothing, and coarsen in powers of 2 from the problem order p to the lowest order ($p = 1$) discretisation (for instance, using $p = 8$, 4, 2 and 1). To solve the coarse (low-order) problem we use BoomerAMG. Note that, since we make use of h-multigrid on the coarse problem, this constitutes a hp-multigrid method, though more broadly such methods can interleave coarsening in h with coarsening in p.

Recently, a new hp-multigrid method was proposed in [11] based on vertex-star relaxation which was proved to be robust for high order FEM for the isotropic problem on regular Cartesian meshes. Note that for this method it is sufficient to coarsen straight from the high order problem to the lowest $p = 1$ problem. This approach is implemented in the software Firedrake and is not yet available in MFEM.

3.1.3 LOR preconditioning

One of the primary difficulties with using high-order finite elements is that the system matrix has increased coupling between degrees of freedom as p increases and, therefore, there is an increasing number of non-zero entries. The number of couplings, that is the number of non-zero entries per row, increases as $\mathcal{O}(p^d)$ and so the memory requirement for the storage of the system matrix, if assembled, increases as $\mathcal{O}(p^{2d})$. This motivates matrix-free approaches, where the use of sum factorisation techniques on tensor product elements can reduce the computational complexity down to $\mathcal{O}(dp^{d+1})$; see [27]. However, in terms of preconditioning, this prohibits methods which require the matrix to be built so as to access individual entries, such as AMG methods [30, 37].

One way to mitigate these issues is to base the preconditioner on a low-order discretisation of the problem, typically $p = 1$. This will yield a much sparser matrix which has a fixed cost to build, independent of p, thus allowing access to the wealth of experience in preconditioning low-order methods. The low-order refined (LOR) preconditioning strategy exploits this idea by discretising on a refined mesh, namely given by a structured grid of Gauss–Lobatto points within each high-order element. Note that these points are clustered towards the boundary of the element and so yield an anisotropic refined mesh which is not shape-regular with respect to p; see Figure 2. The purpose of this particular refined mesh is to ensure the spectral equivalence between the different discretisations, known as finite element method–spectral element method (FEM–SEM) equivalence [12]. In essence, this says that a good preconditioner for the LOR discretisation will also be a good preconditioner for the high-order discretisation.

While many choices could be made to precondition the LOR discretisation of the problem, see [28] and references therein, we stick to algebraic approaches which can be used out-of-the-box. In particular, we will apply BoomerAMG to the LOR system and use this precondition the high-order problem.

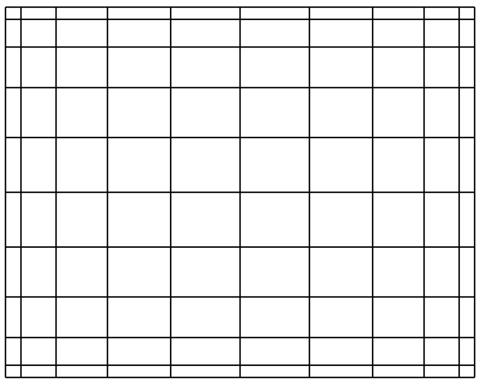

Figure 2: An example of a low-order refined (LOR) mesh on a quadrilateral element for $p = 10$. This uses 11 Gauss–Lobatto points in each dimension.

3.1.4 Overlapping Domain Decomposition Methods

Similar to multigrid methods, overlapping domain decomposition methods (ODDM) [14] are widely used for solving linear systems arising from the discretisation of PDEs. Although the method was originally designed as a solver on its own, it is most frequently used as a preconditioner for an appropriate Krylov method. As the name suggests, the domain in which the PDE is solved is decomposed into multiple overlapping subdomains, and in each of these subdomains the same PDE is defined, possibly with different boundary conditions. One iteration of the ODDM method solves the local PDEs in different subdomains concurrently and then glues the update solution by using a partition of unity that handles the overlapping regions. The method can be defined in an algebraic way, only requiring a sparse matrix, by partitioning the nodes in the sparsity graph of the matrix.

The concurrent solution of local problems makes ODDM very attractive for parallel computing. However, increasing the number of subdomains yields a higher number of iterations to reach convergence. Spectral coarse space (or second level) techniques are therefore used as a remedy. The idea behind a spectral coarse space is to judiciously define an artificial subdomain that encapsulates the information required to ensure fast convergence. Recent advances in ODDM allow the construction of coarse spaces to be done efficiently and algebraically [14, 2]. A generalised eigenvalue problem is solved in each subdomain concurrently to compute local requirements for an effective coarse space. For symmetric positive definite (SPD) matrices (e.g., those arising from elliptic PDEs), the coarse space can be constructed such that the condition number of the preconditioned matrix is upper bounded by any number specified by the user. Since the convergence of Krylov methods for SPD matrices is dependent on the condition number of the preconditioned matrix, this allows for well understood convergence behaviour.

It is worth mentioning that ODDM with spectral coarse space can be seen as a two-grid method where the smoothing is the local solve in the subdomain. However, the key difference with respect to multigrid methods is the coarsening strategy. The coarsening factor (the ratio between two consecutive levels) in multigrid methods is very small compared to that in ODDM. Recently, an algebraic recursive approach of effective spectral coarse spaces was proposed to produce an efficient multilevel ODDM [1].

The software package HPDDM [22] implements all up-to-date spectral multilevel ODDM. Although HPDDM optimises many computational kernels for multilevel ODDM, there is still a large room for improvement—more details are discussed in Section 3.2.7. Therefore, we will only present a synthetic strong scalability comparison against an algebraic multigrid preconditioner to illustrate the potential of ODDM when employed on exascale architectures.

3.1.5 Sparse approximate inverse preconditioning

The sparse approximate inverse (SPAI) algorithm [18] aims to provide an approximation to the inverse of the system matrix, then used as a preconditioner, under the condition that the approximation should be sparse. This is done by minimising the error between the preconditioned matrix and the identity in the Frobenius norm. This choice of norm allows for some inherent parallelisation since columns (or rows) of the approximate inverse can be calculated independently, each column then requiring the solution of a least-squares problem. Much of the effort is then in determining, adaptively, what sparsity pattern each column should possess in order to provide a good approximate inverse without including too many non-zero entries.

To test this approach we utilise the version of SPAI available in PETSc [9], which interfaces to code written by Stephen Barnard. As well as testing this method in its own right, we use this as a proxy for how well the Markov Chain Monte Carlo Matrix Inversion (MCMCMI) method [6] would perform on such problems. MCMCMI computes a sparse approximate inverse by performing a random walk on a graph. While it may be expected that the set up cost of MCMCMI could be more efficient than SPAI on large-scale hardware [4], the behaviour as h scales can be expected to be broadly similar. Since MCMCMI cannot be applied matrix-free and cannot be interfaced directly to a finite element package at this time, this test informed us whether this would be a worthwhile use of developer effort.

3.2 Numerical results for the elliptic problem

In Section3.2.1 we present some initial results that were obtained using Firedrake[1] We use the results of these tests to inform the methods implemented in MFEM, and run at a larger scale, both in terms of the problem size and the number of processors, on Scafell Pike. In all tests, unless otherwise stated, we use the preconditioned conjugate gradient method with a relative residual tolerance of 10^{-8} and a maximum number of 500 iterations; if this is reached before convergence we denote this in tables by "×". Should a particular test fail, we indicate this by marking "−" in the relevant table entry.

3.2.1 2D results using Firedrake

In this section, we provide results in Firedrake. These results are obtained on a desktop computer using 8 processors. We solve (2) with the right-hand side $f = 1$.

Firstly, in Table 2 we detail iteration counts for BoomerAMG [19] and the hp-multigrid method [11] for 1,050,625 degrees of freedom (DOFs). We see that the hp-multigrid method is robust in the isotropic case, but robustness is lost when introducing anisotropy in either the diffusion coefficient or the mesh. On the basis of this experience, we conclude this approach will struggle when applied to highly anisotropic problems. BoomerAMG performs a little better in most cases, although still has issues when applied to high degrees of anisotropy.

		ϵ_y		
ϵ	p	1	0.7	0.3
1	1	6	8	19
	2	7	10	25
	4	9	11	32
	8	8	14	70
10^{-3}	1	44	47	56
	2	38	52	58
	4	36	85	134
	8	9	124	242
10^{-6}	1	×	197	155
	2	373	195	144
	4	257	190	189
	8	6	267	334

(a) AMG

		ϵ_y		
ϵ	p	1	0.7	0.3
1	1	6	18	97
	2	6	18	97
	4	6	17	94
	8	6	16	77
10^{-3}	1	140	168	202
	2	124	134	182
	4	114	130	222
	8	89	115	206
10^{-6}	1	×	×	×
	2	×	×	×
	4	478	×	×
	8	253	400	399

(b) hp-multigrid

Table 2: Iteration counts using BoomerAMG or hp-multigrid in Firedrake for the 2D anisotropic elliptic problem with Kershaw meshes while varying the diffusion parameter ϵ, mesh parameter ϵ_y and FEM order p. Here the mesh refinement is changed with p so that each problem instance has the same number of DOFs, namely 1,050,625. The computations use 8 processors. Note that the problem size is smaller than in the subsequent MFEM tests.

[1]Thanks to Patrick Farrell and Pablo Brubeck for providing the initial code on which these tests were based.

When we consider the algebraic SPAI preconditioner, we found that for large problems this approach consistently fails to converge, and so can only give results for significantly smaller problems. Further, as there is no guarantee that the preconditioner is symmetric positive definite (SPD) we must use GMRES (we note that in our experience CG only works for $p = 1$). Results on a small problem with just 4,225 DOFs are given in Table 3a in order to assess robustness to anisotropy. We find that iteration counts are large, even for this small problem, and increase slightly as we decrease ϵ_y but can decrease as we decrease ϵ. Nonetheless, the approach generally deteriorates when anisotropy is present in both the diffusion coefficient and mesh. We note that iteration counts are broadly stable with p in this regime, increasing only mildly in most cases. However, the primary difficulty with this method is that is not robust as we increase the number of DOFs, by mesh refinement or increasing p for a fixed mesh. This is observed in the results of Table 3b where we see a large growth in iteration counts whenever we decrease h or increase p. We highlight, too, that this behaviour aligns with that reported when the MCMCMI was applied to realistic problems in a recent study [32]. Overall, the SPAI preconditioner lacks the robustness required to be considered as a solver at exascale for problems of this type.

		ϵ_y		
ϵ	p	1	0.7	0.3
1	1	279	348	296
	2	389	357	379
	4	237	365	426
	8	324	355	486
10^{-3}	1	155	358	474
	2	110	359	476
	4	129	351	×
	8	132	376	×
10^{-6}	1	189	367	480
	2	186	356	470
	4	200	330	×
	8	163	396	×

(a) Results for the 2D anisotropic elliptic problem with Kershaw meshes while varying the diffusion parameter ϵ, mesh parameter ϵ_y and FEM order p for a fixed number of DOFs, namely 4,225.

	h				
p	1/8	1/16	1/32	1/64	1/128
1	9	20	55	279	×
2	40	101	389	×	×
4	74	237	×	×	×
8	324	×	×	×	×

(b) Results for the 2D isotropic elliptic problem with regular meshes while varying the mesh spacing h and FEM order p. Note that the number of DOFs is constant along diagonals from lower left to upper right.

Table 3: Iteration counts using SPAI in Firedrake for the 2D elliptic problem. On the left, we vary the anisotropy while keeping the number of DOFs fixed at 4,225. On the right, we vary the mesh spacing h and FEM order p. The computations use 8 processors. Note that problem sizes are much smaller than in other tables.

3.2.2 2D anisotropic elliptic problem with regular mesh

Since high order FEM offers advantages for future computing directions and research on solver technology is ongoing, the Poisson problem on Kershaw meshes has been posed as a benchmark problem by the Center for Efficient Exascale Discretizations (CEED) based in the USA. In particular, in [25] it is posed as a *bake-off problem for solvers,* denoted CEED BPS3. An implementation of this problem within the software MFEM is part of the CEED suite `https://github.com/CEED`, although this is in a private repository at the time of writing. We have adapted this implementation for our anisotropic test problems in order to run our solver tests. The problem setup in BPS3 consists of that given in (2) posed on straight-sided Kershaw meshes with a polychromatic right-hand side f with a known solution (in particular, we use the default structure level $n = 3$; see [25] for details); this is the setup we use in the remainder of this section.

In this section, we present numerical results for the 2D elliptic problem on a regular Cartesian mesh with anisotropy arising from the diffusion coefficient α, as given in (3). We choose the mesh spacing so that it varies with the FEM polynomial order p such that each problem instance has the same fixed number of degrees of freedom (DOFs), namely 67,125,249. For $p = 1$ this corresponds to an equispaced grid with $2^{13} + 1 = 8{,}193$ points in each dimension. We utilise 128 processors in each case giving approximately half a million DOFs per processor.

In Table 4, we provide iteration counts, the total computational run time (in seconds) along with the true error achieved in the solution by comparing it to the exact analytical solution. We vary the anisotropy through the diffusion parameter ϵ in (3) along with the FEM order p. We first note that, for the same number of DOFs, the error reduces as we increase p and this reduction can be several orders of magnitude compared to $p = 1$. Considering AMG on the full system, we see that iteration counts are low in the isotropic case ($\epsilon = 1$) but degrade as the anisotropy increases. In the highly anisotropic case $\epsilon = 10^{-6}$, AMG begins to really struggle with large iteration counts observed for all but the case $p = 8$ and we note that the error in the solution can be larger than when using the LOR approach. The p-multigrid method performs even better and is the most efficient solver in the isotropic case with iteration counts and time taken which are independent of p. However, iteration counts become much worse in the anisotropic case and the solver fails to converge within 500 iterations when $\epsilon = 10^{-6}$. These results suggest an overall lack of robustness for the AMG and p-multigrid solvers for highly anisotropic problems. The LOR approach proves more robust with relatively low iteration counts in all cases and we further note that the total run time remains small with very little increase as we increase p, unlike AMG on the full system where larger run times are observed.

It is interesting to note from Table 4 that LOR is far more robust than AMG for anisotropic problems of this type, even for low orders of p. We highlight that the poor performance of AMG for lower orders corresponds with the results obtained in Firedrake, reported in Table 2a, and we conclude this is a feature of the solver, not of the implementation. On the other hand, AMG appears to be fairly robust to this form of anisotropy for $p = 8$, which is intriguing.

		AMG			p-MG			LOR		
ϵ	p	It.	Tot. time	Error	It.	Tot. time	Error	It.	Tot. time	Error
1	1	9	6.9	5.7e-06				13	10.1	5.7e-06
	2	8	7.8	2.5e-07	9	6.6	2.5e-07	14	9.4	2.5e-07
	4	13	16.9	3.3e-09	8	4.7	3.1e-09	15	12.3	3.2e-09
	8	12	35.9	1.0e-09	9	5.1	3.3e-10	19	14.3	6.6e-10
10^{-3}	1	39	24.1	4.8e-06				12	11.3	4.8e-06
	2	40	28.7	2.5e-07	89	44.3	2.5e-07	17	11.8	2.5e-07
	4	40	43.2	3.2e-09	105	44.2	3.4e-09	17	12.4	3.1e-09
	8	11	34.6	3.0e-10	144	61.5	1.6e-09	17	12.4	3.5e-10
10^{-6}	1	463	200.4	4.5e-06				12	11.4	4.5e-06
	2	463	246.2	2.8e-07	×	×	×	17	11.8	2.5e-07
	4	397	326.9	1.5e-07	×	×	×	18	12.7	2.9e-09
	8	15	40.0	1.0e-09	×	×	×	18	12.6	2.7e-10

Table 4: Iteration counts, total time of computation (in seconds), and error in solution for the 2D anisotropic elliptic problem with regular meshes while varying the diffusion parameter ϵ and FEM order p. Here the mesh refinement is changed with p so that each problem instance has the same number of DOFs, namely 67,125,249. The computations use 128 processors.

3.2.3 2D elliptic problem with Kershaw meshes

We turn now to the case of anisotropy through the mesh. In this section, we provide results for the isotropic diffusion problem ($\epsilon = 1$) but discretised on an anisotropic Kershaw mesh [24]. The mesh anisotropy is controlled by the Kershaw parameter ϵ_y, with $\epsilon_y = 1$ being isotropic and decreasing ϵ_y giving increasingly stretched elements and thus a more anisotropic mesh. We run tests analogous to the previous section, now varying ϵ_y and p with a fixed number of DOFs.

Results are given in Table 5 and we first note that AMG on the full system is somewhat more robust for the parameters tested than in the previous example in Table 4. Similarly, p-multigrid now converges in all cases, however, there is still a large increase in iteration counts and thus time taken as the anisotropy increases. Both AMG and LOR now have some small degradation in iteration counts as p increases. While AMG on the full system generally has a lower iteration count over LOR, we see that the total run time is somewhat larger for higher p. The run time for LOR remains low but is now seen to slowly increase as p increases. We also note that anisotropy in the mesh appears to have a larger impact on the error achieved in the solution for this problem, nonetheless, good accuracy is still achieved for higher-order FEM.

		AMG			p-MG			LOR		
ϵ	p	It.	Tot. time	Error	It.	Tot. time	Error	It.	Tot. time	Error
1	1	9	6.9	5.7e-06				13	10.1	5.7e-06
	2	8	7.8	2.5e-07	9	6.6	2.5e-07	14	9.4	2.5e-07
	4	13	16.9	3.3e-09	8	4.7	3.1e-09	15	12.3	3.2e-09
	8	12	35.9	1.0e-09	9	5.1	3.3e-10	19	14.3	6.6e-10
0.7	1	10	10.7	1.3e-05				13	14.8	1.3e-05
	2	14	12.9	3.8e-07	14	8.8	3.8e-07	16	15.1	3.8e-07
	4	15	25.6	8.5e-09	18	8.7	8.5e-09	19	15.6	8.6e-09
	8	18	55.5	1.2e-09	26	12.3	9.4e-10	24	18.3	6.3e-10
0.3	1	19	14.7	1.9e-04				16	16.0	1.9e-04
	2	27	26.4	7.4e-06	66	32.7	7.4e-06	23	17.7	7.4e-06
	4	37	55.4	6.5e-07	108	45.1	6.5e-07	32	20.8	6.5e-07
	8	32	88.3	1.7e-07	169	72.1	1.7e-07	43	26.5	1.7e-07

Table 5: Iteration counts, total time of computation (in seconds), and error in solution for the 2D isotropic elliptic problem ($\epsilon = 1$) with Kershaw meshes while varying the mesh parameter ϵ_y and FEM order p. Here the mesh refinement is changed with p so that each problem instance has the same number of DOFs, namely 67,125,249. The computations use 128 processors.

3.2.4 2D anisotropic elliptic problem with Kershaw mesh

We now give results for both AMG on the full system and LOR when anisotropy is present in both the diffusion coefficient and the mesh. In this case, the anisotropy combines to make the problem particularly challenging. Indeed, we note that for $\epsilon = 10^{-6}$ and $\epsilon_y = 0.3$ the solution error is relatively large due to the poor approximation properties of approximating a highly anisotropic problem on a highly anisotropic mesh. It is therefore unsurprising that solvers will begin to struggle. In Table 6, we provide our results and first note that neither AMG nor LOR are robust in this regime. In all cases, iteration counts and timings increase with p. Nonetheless, LOR remains the fastest due to the lower setup costs.

			AMG			LOR		
ϵ	ϵ_y	p	It.	Tot. time	Error	It.	Tot. time	Error
0.7	10^{-3}	1	55	30.2	1.4e-03	55	30.0	1.4e-03
		2	149	106.2	2.4e-05	110	47.9	2.4e-05
		4	170	216.6	2.4e-07	216	97.2	2.3e-07
		8	247	557.2	7.8e-08	297	131.1	2.5e-08
0.7	10^{-6}	1	430	182.4	2.7e-03	85	41.4	2.7e-03
		2	×	×	×	314	121.3	1.9e-04
		4	×	×	×	×	×	×
		8	×	×	×	×	×	×
0.3	10^{-3}	1	70	36.7	7.4e-03	85	41.5	7.4e-03
		2	220	146.8	5.5e-04	191	77.9	5.5e-04
		4	418	468.0	3.4e-05	394	167.5	3.4e-05
		8	465	1020.0	3.7e-06	×	×	×
0.3	10^{-6}	1	458	196.1	1.0e-02	108	50.5	1.0e-02
		2	×	×	×	401	154.1	2.3e-03
		4	×	×	×	×	×	×
		8	×	×	×	×	×	×

Table 6: Iteration counts, total time of computation (in seconds), and error in solution for the 2D anisotropic elliptic problem with Kershaw meshes while varying the diffusion parameter ϵ, mesh parameter ϵ_y and FEM order p. Here the mesh refinement is changed with p so that each problem instance has the same number of DOFs, namely 67,125,249. The computations use 128 processors.

3.2.5 3D anisotropic elliptic problem with regular mesh

We move now to the 3D anisotropic problem, where the diffusion coefficient is given by (4). We present similar results to the 2D case, now for a 3D problem of size 16,974,593 DOFs. In Table 7, we see that both AMG and p-multigrid lack robustness as we increase the anisotropy in the diffusion coefficient while LOR remains fairly robust. Further, outside of the isotropic case, LOR gives both the lowest iteration counts and the fastest total time to solution. We also note that for some tests AMG failed to build a preconditioner and, thus, the test did not complete: these are marked "−".

		AMG			p-MG			LOR		
ϵ	p	It.	Tot. time	Error	It.	Tot. time	Error	It.	Tot. time	Error
1	1	−	−	−				19	28.4	5.6e-03
	2	7	17.6	3.4e-02	8	42.5	3.4e-02	27	27.7	3.4e-02
	4	9	97.8	5.2e-02	7	5.6	5.2e-02	24	29.7	5.2e-02
	8	−	−	−	11	3.6	6.4e-02	22	35.7	6.4e-02
10^{-3}	1	50	37.4	5.5e-03				20	10.3	5.5e-03
	2	52	56.7	4.1e-02	94	34.0	4.1e-02	29	9.5	4.1e-02
	4	61	120.4	6.6e-02	97	21.9	6.6e-02	26	11.1	6.6e-02
	8	−	−	−	120	24.6	8.5e-02	23	10.9	8.5e-02
10^{-6}	1	181	66.1	5.5e-03				20	10.2	5.5e-03
	2	163	124.3	4.1e-02	134	46.1	4.1e-02	30	9.4	4.1e-02
	4	124	194.6	6.8e-02	151	34.3	6.8e-02	27	10.6	6.8e-02
	8	−	−	−	181	36.4	8.7e-02	24	10.4	8.7e-02

Table 7: Iteration counts, total time of computation (in seconds), and error in solution for the 3D anisotropic elliptic problem with regular meshes while varying the diffusion parameter ϵ and FEM order p. Here the mesh refinement is changed with p so that each problem instance has the same number of DOFs, namely 16,974,593. The computations use 128 processors.

3.2.6 3D elliptic problem with Kershaw meshes

In Table 8 we detail results on 3D Kershaw meshes. Again, we reach similar conclusions to the 2D case, namely that no method is fully robust but the most promising approach is LOR which takes the lowest total time and has only mildly increasing iteration counts as the anisotropy in the mesh increases. As such, we consider LOR to the method of choice for treating high order discretisation of highly anisotropic elliptic problems and will focus on this approach in subsequent sections.

		AMG			p-MG			LOR		
$\epsilon_y = \epsilon_z$	p	It.	Tot. time	Error	It.	Tot. time	Error	It.	Tot. time	Error
1.0	1	−	−	−				19	28.4	5.6e-03
	2	7	17.6	3.4e-02	8	42.5	3.4e-02	27	27.7	3.4e-02
	4	9	97.8	5.2e-02	7	5.6	5.2e-02	24	29.7	5.2e-02
	8	−	−	−	11	3.6	6.4e-02	22	35.7	6.4e-02
0.7	1	9	250.0	1.2e-02				21	61.3	1.2e-02
	2	11	59.8	2.8e-02	21	48.9	2.8e-02	29	63.8	2.8e-02
	4	17	149.7	4.6e-02	30	13.0	4.6e-02	28	47.0	4.6e-02
	8	−	−	−	48	10.9	4.5e-02	32	47.7	4.5e-02
0.3	1	17	155.4	2.0e-02				22	48.2	2.0e-02
	2	28	111.0	4.2e-02	162	79.5	4.9e-02	31	50.0	4.2e-02
	4	41	254.9	6.4e-02	207	48.8	1.3e-01	40	46.6	6.4e-02
	8	−	−	−	313	63.2	1.7e-01	56	53.0	6.4e-02

Table 8: Iteration counts, total time of computation (in seconds), and error in solution for the 3D isotropic elliptic problem ($\epsilon = 1$) with Kershaw meshes while varying the mesh parameter $\epsilon_y = \epsilon_z$ and FEM order p. Here the mesh refinement is changed with p so that each problem instance has the same number of DOFs, namely 16,974,593. The computations use 128 processors.

3.2.7 Inner solver: Multilevel Domain Decomposition or AMG

In the previous sections, we have seen that LOR is the most robust method in 2D and 3D for dealing with anisotropy in elliptic equations discretised by high-order finite elements. We now explore alternative methods to solve the low order system in the FEM–SEM equivalence. In particular, we present a strong scaling comparison between a spectral multilevel DD preconditioner (HPDDM) and an algebraic multigrid preconditioner (BoomerAMG) as the low-order solver for an LOR preconditioner.

The test case is the 2D elliptic problem with Kershaw mesh discretised on a grid of size 256×256 using 8th order FEM resulting in a system of size $\approx 4 \times 10^6$. The Kershaw parameter is $\epsilon_y = 0.7$. The test case is chosen to assess the strong scalability of the setup and solve phases when each processor owns a small number of DOFs without the necessity for employing a very large number of processors. The DD preconditioner constructs two grids on 32–512 processors and three grids on 1024 processors.

	32	64	128	256	512	1024
AMG	51	51	50	52	51	52
DD	68	61	54	47	50	54

Table 9: Iteration count for AMG and multilevel DD.

Table 9 presents the iteration count required to reach a relative residual norm 10^{-8} when using each preconditioner. These results show that both preconditioners are robust with respect to the number of processors on this test problem.

	AMG		DD	
lev	DOFs	nnz	DOFs	nnz
1	4198401	36923049	4198401	36923049
2	574905	5265911	5120	750400
3	162123	3136361	-	-
4	39601	1038607	-	-
5	8784	274304	-	-
6	1581	50063	-	-
7	231	6361	-	-
8	30	562	-	-
9	4	16	-	-

(a) Grid information on 128 processors

	AMG		DD	
lev	DOFs	nnz	DOFs	nnz
1	4198401	36923049	4198401	36923049
2	569530	5192580	40960	6505600
3	159448	3093544	80	2100
4	38404	1032022	-	-
5	7552	228534	-	-
6	1198	35398	-	-
7	174	4490	-	-
8	22	346	-	-
9	4	16	-	-

(b) Grid information on 1024 processors

Table 10: Grid and subgrid information for different architectures.

Table 10 contains the grid sizes along with the number of nonzero elements per grid for both AMG and the DD preconditioner. It is clear that the coarsening factors for DD are very large compared to the ones resulting from AMG. The largest coarsening factor for AMG is 7.7 while for the multilevel DD it reaches 820 on 128 processors and 512 on 1024 processors. The large coarsening factors obtained by the multilevel DD method allow us to reduce significantly the number of levels while maintaining the efficiency of the preconditioner. We also observe that the DD preconditioner produces grid matrices that are slightly denser than the ones obtained from AMG. This, in turn, has consequences on the time required to assemble these matrices during the setup phase and operate with them during the solve phase.

Figure 3 shows the total time required to solve the linear system up to a relative residual norm 10^{-8} using the multilevel DD and AMG preconditioners. For each preconditioner, the total time is broken down into the time spent during the setup phase and during the solve phase (the preconditioned Krylov subspace solver). Furthermore, the setup time for the DD preconditioner is broken down into two parts, the time required to assemble the operators on levels > 1 and the rest of the setup phase.

We notice from Figure 3 that the bottleneck for scalability for both preconditioners is the setup phase. Starting from 256 processors (approximately 5000 DOFs per processor), the setup phase for AMG struggles to scale and dominates the run time. Concerning the DD preconditioner, the computation of the matrices on levels > 1 dominates the run time (73% of the setup time). As expected, the rest of the computations during the setup scale very well. We give some details of this in Appendix A, and we will implement the suggested improvements in computing the matrices on levels > 1 in future work, which we expect to yield a more scalable setup time.

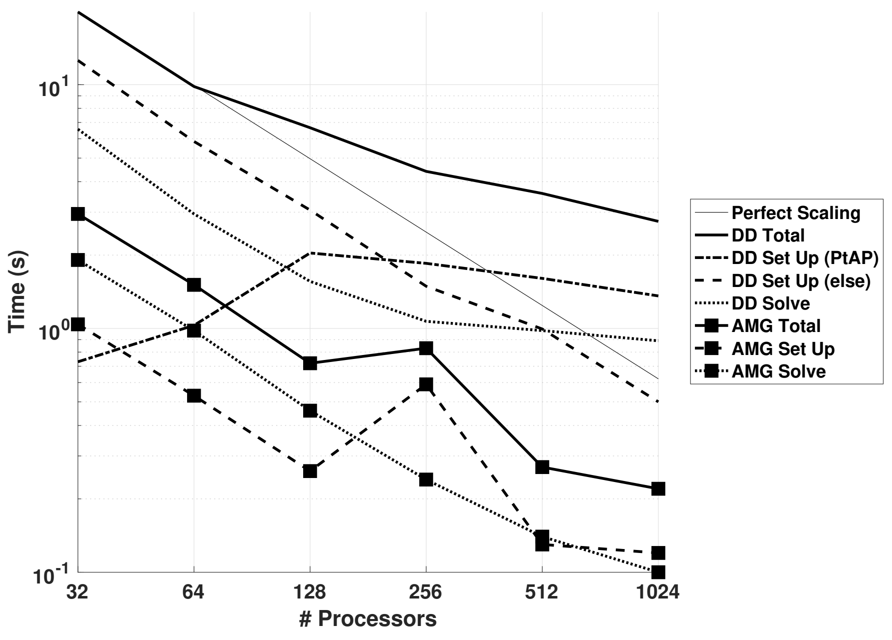

Figure 3: Strong scalability comparison between BoomerAMG and spectral multilevel DD (HPDDM).

4 Stationary hyperbolic problems

As a model for stationary hyperbolic problems, we use the advection-diffusion equation. In our examples, we consider the square domain $\Omega = (-1,1)^2$ and utilise Dirichlet boundary conditions on the boundary $\Gamma = \partial\Omega$ so that the problem is given by

$$\begin{aligned} -\epsilon\nabla^2 u + w(\boldsymbol{x})\cdot\nabla u = 0, &\qquad \boldsymbol{x}\in\Omega,\\ u = u_D, &\qquad \boldsymbol{x}\in\Gamma, \end{aligned}$$

where $w(\boldsymbol{x})$ is the so-called wind vector defining the advection field and u_D defines the values on the boundary, both of which must be specified.

When the parameter ϵ is small, the problem is dominated by advection and solutions may often exhibit steep gradients (boundary layers) in some parts of the domain. This presents challenges both in discretising and solving such problems. The degree to which the problem is advection-dominated is typically expressed by the Peclet number

$$\mathscr{P} = \frac{|w|L}{\epsilon},$$

where L is the characteristic length scale of the domain Ω and $|w|$ is a measure of the size of the wind. The larger the Peclet number $\mathscr{P}$, the more advection dominates and so the more challenging the problem becomes. In our examples, the wind is chosen such that $|w| = O(1)$ and the domain is fixed so that by varying ϵ we vary $\mathscr{P}$.

To discretise, we use high-order continuous Galerkin finite elements and make use of a sufficiently fine mesh resolution. When $\mathscr{P}$ is large and advection is dominant, the continuous Galerkin solution may exhibit spurious oscillations if the mesh is not sufficiently refined to capture any layers in the solution. This is quantified by the mesh Peclet number

$$\mathscr{P}_h = \frac{\mathscr{P}h}{2L} = \frac{|w|h}{2\epsilon},$$

which should be lower than the unity to avoid such oscillations. One way to mitigate this lack of stability is to use a streamline diffusion discretisation [21, 15] instead, which formulates a Petrov–Galerkin method. However, MFEM does not support bilinear form integrators required for the second order streamline upwind

Petrov–Galerkin. Nonetheless, we employ a first order streamline diffusion stabilization scheme when necessary, i.e., when $\mathscr{P}_h < 1$. An alternative approach is to use a discontinuous Galerkin (DG) method but we shall not consider such an approach in this section.

In the rest of this section, we consider two test cases corresponding to two different wind vectors. In each case, we compare two variants of algebraic multigrid preconditioners: BoomerAMG and ℓ-AIR [26]. The ℓ-AIR approach is tailored to non-symmetric problems such as advection-diffusion. These preconditioners are used either directly or as the low-order solver within a LOR preconditioner. When LOR is employed, the matrix is partially assembled (matrix-free) and the algebraic multigrid preconditioner is set up using the discretised PDE on the refined mesh with order $p = 1$ finite elements.

4.1 Advection–diffusion with a shear wind

We first consider the case of a constant shear wind, as in Example 6.1.3 of [15], where

$$w(\boldsymbol{x}) = (-\sin(\pi/6), \cos(\pi/6))^T,$$

and

$$u_D(\boldsymbol{x}) = \begin{cases} 1 & \text{if } x = 1, \\ 1 & \text{if } y = -1 \text{ and } 0 \le x \le 1, \\ 0 & \text{otherwise.} \end{cases}$$

The solution u features an internal layer of width $\mathcal{O}(\sqrt{\epsilon})$, emanating from the discontinuity of the boundary data at $(0, -1)$ and following the wind vector w, as well as an exponential boundary layer of width $\mathcal{O}(\epsilon)$ along the top of the domain.

In the results that follow, we vary the Peclet number $\mathscr{P}$ by reducing the value of ε from 10^{-1} to 10^{-4} to assess the robustness of the algebraic multigrid and the LOR preconditioners with respect to $\mathscr{P}$.

ε	p	Full				LOR			
		AMG		ℓ-AIR		AMG		ℓ-AIR	
		It.	Tot. time	It.	Tot. time	It.	Tot. time	It.	Tot. time
10^{-1}	1	6	1.4	×	×	10	2.2	×	×
	2	7	1.9	×	×	12	2.3	×	×
	4	11	4.9	×	×	13	3.6	×	×
	8	10	10.7	37	31.7	15	5.0	×	×
10^{-2}	1	5	1.3	38	5.4	8	2.0	34	6.0
	2	6	1.7	22	5.3	10	2.1	35	5.5
	4	8	4.3	54	20.2	11	3.7	27	6.1
	8	7	8.4	12	12.6	12	4.0	×	×
10^{-3}	1	5	1.4	31	5.2	7	2.2	22	4.1
	2	6	1.8	34	7.5	9	2.0	21	3.7
	4	6	4.8	×	×	10	3.6	11	3.4
	8	6	8.0	8	9.8	10	4.3	12	3.8
10^{-4}	1	6	3.3	40	12.1	8	3.9	19	5.2
	2	7	4.1	6	3.2	9	3.9	35	9.3
	4	7	6.5	14	9.8	9	4.4	29	7.7
	8	8	12.1	×	×	11	4.9	17	5.3

Table 11: Iteration counts for the 2D advection-diffusion equation with a shear wind. Here the mesh refinement is changed with p so that each problem instance has the same number of DOFs, namely 67,125,249.

Table 11 presents the iteration count required to a reach relative residual norm smaller than 10^{-8} as well as the total run time. The polynomial order of the finite element method is doubled starting from $p = 1$ up to $p = 8$ and the mesh is coarsened correspondingly so that the number of degrees of freedom is fixed at 67,125,249. We notice that for all preconditioning variants the iteration count is small, showing their effectiveness. Although using AMG on the full system requires the fewest number of iterations, the increased computational cost incurred by increasing the polynomial order prohibits the Full-AMG variant from being the fastest. The variant

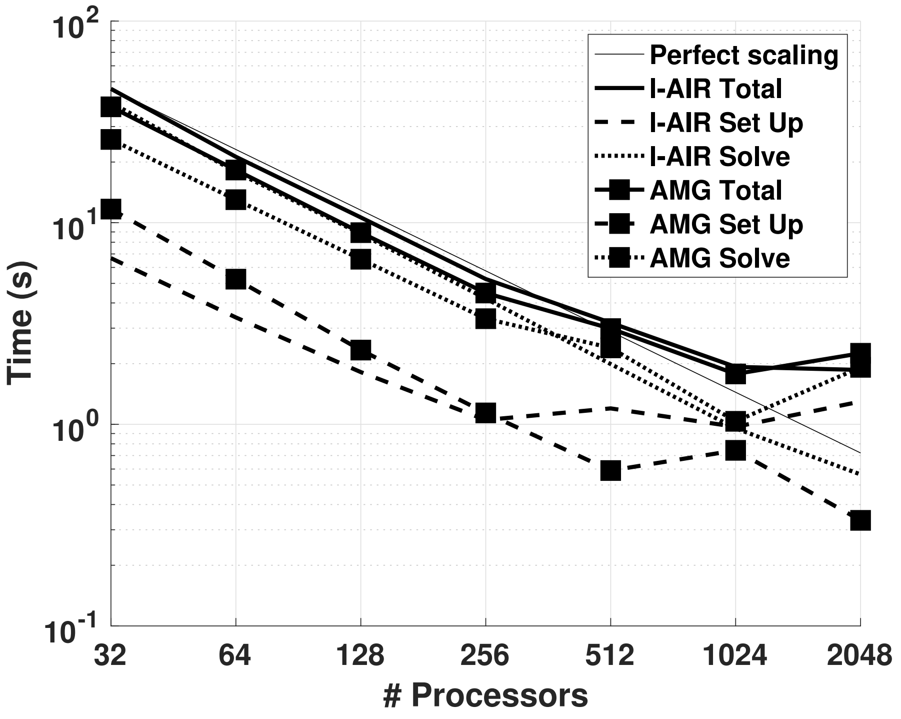

Figure 4: Strong scalability of the advection-diffusion with shear wind problem and LOR preconditioner. The viscosity parameter is $\varepsilon = 10^{-4}$.

LOR-AMG has slightly larger iteration counts (at most 5 extra iterations compared to Full-AMG) but requires the least amount of time for each order. Furthermore, the total run time increases only mildly when increasing the order while the iteration count is robust with respect to the order. We also observe robustness with respect to the Peclet number $\mathscr{P}$ as it increases by decreasing ε and so making the problem more advection-dominated.

Figure 4 provides results for a strong scalability test with BoomerAMG and ℓ-AIR when used as part of a LOR preconditioner for the linear system arising from discretising the 2D advection-diffusion problem with a shear wind. In particular, we use the case $\varepsilon = 10^{-4}$ on a Cartesian grid of size 1024×1024 with 8th order FEM. The number of iterations to reach convergence is 11 for AMG and ranges between 17 and 20 for ℓ-AIR. We can observe that both preconditioners achieve scalability for the setup and solve phases.

4.2 Advection–diffusion with a recirculating wind

As a second example, we consider the case of a recirculating wind, as in Example 6.1.4 of [15], where

$$w(\boldsymbol{x}) = \left(2y(1-x^2), -2x(1-y^2)\right)^T,$$

and

$$u_D(\boldsymbol{x}) = \begin{cases} 1 & \text{if } x = 1, \\ 0 & \text{otherwise.} \end{cases}$$

This provides a simple model of heat in a cavity where the right-hand wall is kept hot ($u_D = 1$) and the heat is advected by a recirculating current. The discontinuities in boundary data at the right-hand corners yield nearby boundary layers in the solution.

ε	p	Full				LOR			
		AMG		ℓ-AIR		AMG		ℓ-AIR	
		It.	Tot. time	It.	Tot. time	It.	Tot. time	It.	Tot. time
10^{-1}	1	6	1.4	×	×	10	2.3	×	×
	2	7	2.4	×	×	12	2.3	×	×
	4	10	4.9	×	×	13	3.6	×	×
	8	10	10.9	43	36.6	15	4.5	×	×
10^{-2}	1	5	1.3	68	10.3	8	2.0	42	7.4
	2	6	1.7	38	8.0	10	2.1	42	6.7
	4	9	4.6	×	×	11	3.5	42	8.6
	8	9	10.1	16	16.2	12	4.0	×	×
10^{-3}	1	8	1.9	47	8.2	10	2.5	15	3.4
	2	9	2.9	62	15.0	12	2.6	15	3.1
	4	11	7.2	37	18.4	13	4.2	26	6.9
	8	12	13.3	13	13.6	15	4.9	18	5.1
10^{-4}	1	×	×	17	8.0	×	×	11	4.4
	2	9	5.7	36	11.4	10	4.0	13	3.7
	4	9	7.4	17	10.7	10	4.4	15	4.8
	8	9	12.8	44	43.6	11	4.5	×	×

Table 12: Iteration counts for the 2D advection-diffusion equation with a recirculating wind. Here the mesh refinement is changed with p so that each problem instance has the same number of DOFs, namely = 67,125,249.

Table 12 contains the iteration count required to reach a relative residual norm smaller than 10^{-8} as well as the total run time. The setup is similar to that for Table 11 in the previous section and we broadly see similar trends in the results. Namely, LOR-AMG is the fastest approach for higher-order p and provides the most robust solver.

5 Time-dependent hyperbolic problems

In this section, we turn our attention to time-dependent hyperbolic problems, we must consider how to advance solutions in time. We first discretise in space to obtain a semi-discrete problem, turning the problem into an ordinary differential equation (ODE) in time. To numerically solve the ODE, we use a time-stepping scheme which can either be explicit or implicit. Explicit schemes update the solution based solely on values at previous time-steps and so are relatively cheap to use, however, they typically suffer from stability issues which enforce a constraint on how large the time-step can be. This is particularly true of so-called stiff ODEs, which often occur in physically relevant applications. Since performing many small time steps becomes costly and can accumulate errors, implicit methods are often favourable when high solution accuracy is required. In an implicit scheme, the solution at the next time-step is computed through solving linear systems rather than each degree of freedom only depending on past values. This provides more stable time-stepping schemes but comes with the additional cost of solving the associated linear systems.

Here we consider the most promising types of implicit time-stepping schemes. Most of these approaches belong to a broad class of ODE solvers called Runge–Kutta (RK) methods, which are popular and include many well-known classical schemes such as Crank–Nicolson. Before continuing, we note that there are two notions of stability which can be helpful in characterising time-stepping schemes based on solving the stiff problem

$$\partial_t u(t) - ku(t) = 0, \quad u(0) = 1,$$

where $k \in \mathbb{C}$ with $\mathrm{Re}(k) < 0$. If, for a fixed time-step, the numerical scheme approaches 0 as $t \to \infty$ then the scheme is said to be A-stable. If, in addition, this holds in a single time-step as the time-step tends to infinity then the scheme is said to be L-stable. We note that L-stability is a stronger property than A-stability.

5.1 Implicit time-stepping schemes

We shall consider several classes of implicit time-stepping methods. Within each class, there can be many schemes. Primarily they are categorised by the order they achieve, stability properties (A-stable or L-stable) and then other identifying characteristics, such as an underlying approach they are based on. We only consider approaches that solve over a single time-step rather than multistep methods which, as described in [13], have several drawbacks such as the fact that they are more memory intensive due to requiring the storage of the solution at several past time-steps, and that there are no implicit multistep methods that are A-stable and of order greater than two.

5.1.1 Classical implicit schemes

The most well-known time-stepping schemes are classical methods, which typically have relatively low order. The simplest implicit scheme is the backward Euler method, which uses a first-order finite difference to approximate the time derivative with all other terms being prescribed at the current time-step, hence requiring the solution of a linear system.

Backward Euler is only first order, and higher order schemes can also be devised. One of the most popular classical schemes is the second order Crank–Nicolson method, which can be derived from the trapezium rule. We will consider a similar second-order method, known as the implicit midpoint rule, which is a geometric integrator and within the family of Gauss–Legendre Runge–Kutta methods. Note that both backwards Euler and the implicit midpoint rule are one-stage RK methods: we go straight to the solution at the next time step rather than having internal stages to compute first. To achieve higher order, the RK methods we now discuss utilise more than one stage.

5.1.2 SDIRK schemes

Diagonally implicit Runge–Kutta (DIRK) methods, see [23], are RK methods that have a lower-triangular Runge–Kutta matrix, which describes the set of s stages to be performed. Such methods are attractive since this reduces the implicit solves to a series of $n \times n$ systems rather than a larger $ns \times ns$ system. In particular, we consider singly diagonally implicit Runge–Kutta (SDIRK) methods which, in addition, have an RK matrix with a constant diagonal. These methods are typically differentiated by their order, whether they are A-stable or L-stable, and the number of stages s which they use.

5.1.3 Fully implicit RK schemes

As noted in [23, 13], while DIRK methods are enticing due to their simplified structure, there are a number of disadvantages of such schemes, especially for stiff problems where they may suffer from so-called order reduction, resulting in lower accuracy than their formal order would suggest. Fully implicit Runge–Kutta methods [34, 33], which utilise the full RK matrix, can overcome these limitations. However, due to the challenging linear systems to be solved they have generally been considered impractical. Nonetheless, with recent advances, such schemes look more promising when the linear algebra is treated carefully and their ability to provide a high-order robust solver is worthy of investigation.

Common families of fully implicit RK methods include Gauss–Legendre, Gauss–Lobatto and Gauss–Radau schemes. Such families also encompass classical techniques such as backward Euler (equivalent to RadauIIA) and Crank–Nicolson (equivalent to LobattoIIIA). In particular, we consider methods in the Gauss–Legendre family, denoted IRK-Gauss, and the Gauss-Lobatto family, denoted IRK-LobIIIC.

5.1.4 IMEX schemes

The main motivation behind implicit-explicit (IMEX) schemes [8] is that the time scale of different terms in the underlying equation may be quite different; some terms may be well approximated by an explicit time-stepping scheme, while others require an implicit method for stability and accuracy. Hence, if we split such terms and treat one set with an explicit method and the other set with an implicit method, then we may hope to gain the best of each method.

However, the split is not always clear and can be somewhat problem dependent, even for the same PDE model, as it may have differing physics regimes. IMEX methods may also suffer from accuracy issues and leak stiffness. While they can be applied successfully, see examples discussed in [13], for these reasons we have chosen not to pursue these schemes for the test problems we consider here. We feel that they are best when targeted to a particular equation or model. We envisage that in more challenging applications and multiphysics simulations, IMEX schemes should be investigated when a specific set of equations and parameter regimes has been established.

5.2 Time-dependent advection problem

To start our study of time-dependent hyperbolic problems, we examine the basic case of an advection equation on the square domain $\Omega = (-1,1)^2$, given by

$$\partial_t + w(\boldsymbol{x}) \cdot \nabla u = 0, \qquad \boldsymbol{x} \in \Omega,\ t \in (0,T], \tag{5a}$$

$$u(0,\boldsymbol{x}) = u_0(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega,\ t = 0, \tag{5b}$$

with periodic boundary conditions in space. We consider the simple example of a constant advection field $w(\boldsymbol{x}) = (1,0)^T$ so that the exact solution consists of the initial condition $u_0(\boldsymbol{x})$ travelling to the right and, due to the periodic domain, returning to the initial state when t is a multiple of 2. That is, $u(2j,\boldsymbol{x}) = u_0(\boldsymbol{x})$ for all $j \in \mathbb{N}$ for the exact solution. This allows us to straightforwardly measure the error in the time-stepping solution at $T = 2j$ by comparing it with the (discrete) initial condition.

To implement this problem, we modify MFEM Example 9 [7] and use the initial condition

$$u_0(\boldsymbol{x}) = \frac{1}{16}\operatorname{erfc}(10(x-0.45))\operatorname{erfc}(-10(x+0.45))\operatorname{erfc}(10(y-0.25))\operatorname{erfc}(-10(y+0.245)),$$

where erfc is the complementary error function. This represents a "bump" in the centre of the domain which is then advected to the right. To begin our exploration of time-stepping schemes, we consider the classical approaches of the backwards Euler scheme and the Implicit Midpoint Rule along with several higher order methods in the family of implicit Runge–Kutta methods. In particular, we consider singly diagonally implicit Runge–Kutta (SDIRK) methods as given in Table 13 (see, e.g., [16, Appendix B] for further details), which include the aforementioned classical approaches.

Scheme	Stability	Order
Backwards Euler	L	1
SDIRK22	L	2
SDIRK33	L	3
Implicit Midpoint Rule	A	2
SDIRK23	A	3
SDIRK24	A	4

Table 13: Time-stepping schemes used for the advection problem tests.

To discretise in space, we use discontinuous Galerkin (DG) finite elements of order equal to the order of the time-stepping scheme used, namely in the range $p \in [1,4]$. To precondition the linear systems that arise, we consider two approaches: a simple block ILU preconditioner and the specialist ℓ-AIR preconditioner tailored for non-symmetric problems such as advection. The time-step Δt is chosen of the same order as Δx, in particular given by $\Delta t = \frac{3}{2}\Delta x$, where $\Delta x = \frac{1}{6}2^{-r}$ with r being a mesh refinement parameter for the regular Cartesian mesh used.

In Table 14 we tabulate for the L-stable time-stepping schemes the total time to solution, the L_2-error in the solution at time $T \in \{2,8\}$ and the average number of iterations for each linear system solved, each for a sequence of mesh refinements. Table 15 provides similar results for the A-stable methods. We first note that ℓ-AIR always converges to within the relative tolerance of 10^{-8} in a single iteration and, aside from the smallest problem sizes, gives the fastest time to solution, with block ILU typically requiring two or three iterations. In terms of the error, the higher order schemes typically give smaller errors for the same mesh refinement, as expected, but also yield a larger run time. Further, the A-stable methods are faster than their equivalent order L-stable method but are slightly less accurate.

If we consider which scheme is the fastest to reach a given error, however, we see that the extra work required in the fourth order method SDIRK34 is not worth the effort in terms of the run time, with SDIRK33 giving a similar error in the solution more cheaply. Conversely, the third order SDIRK23 scheme manages to achieve an order of magnitude smaller error on the finest meshes used compared to the second order SDIRK22 scheme and so can make the extra work worthwhile. If we consider which method attains an error below, for example, 10^{-3} at $T = 2$ in the fastest time we find that SDIRK23 is quickest at 11.65s followed by SDIRK33 at 16.97s. If we change to consider a final time $T = 8$, however, then we see that SDIRK33 is quickest at 65.93s followed by SDIRK34 at 147.43s. Thus, in this case, we see that the extra stability of SDIRK33 or higher order of SDIRK34 can pay off. As a general rule, the higher order methods are more efficient to attain a given accuracy compared to lower order methods such as the classical Backwards Euler scheme or Implicit Midpoint Rule, but this should be balanced by the consideration of stability, especially when solving over a long time interval. On balance, the L-stable method SDIRK33 performs best over the range of tests here, being favourable for larger T and only marginally slower than SDIRK23 for small T.

				Backwards Euler ($p = 1$)			SDIRK22 ($p = 2$)			SDIRK33 ($p = 3$)		
T	Δt	r	Prec.	Time	Error	It.	Time	Error	It.	Time	Error	It.
2	1/32	3	ILU	0.04	1.6e-01	3.8	0.09	2.2e-02	2	0.43	7.6e-03	2
			ℓ-AIR	0.09	1.6e-01	1	0.13	2.2e-02	1	0.35	7.6e-03	1
	1/64	4	ILU	0.16	1.1e-01	3.5	0.95	6.1e-03	2.3	4.60	1.3e-03	2.9
			ℓ-AIR	0.15	1.1e-01	1	0.64	6.1e-03	1	2.19	1.3e-03	1
	1/128	5	ILU	1.86	7.6e-02	3	7.43	1.6e-03	2	28.29	1.7e-04	2
			ℓ-AIR	0.79	7.6e-02	1	4.36	1.6e-03	1	16.97	1.7e-04	1
	1/256	6	ILU	9.75	4.8e-02	2.4	60.47	3.9e-04	2	219.91	2.2e-05	2
			ℓ-AIR	6.52	4.8e-02	1	38.25	3.9e-04	1	141.42	2.2e-05	1
8	1/32	3	ILU	0.09	3.1e-01	3.2	0.28	6.5e-02	2	1.53	2.1e-02	2
			ℓ-AIR	0.13	3.1e-01	1	0.29	6.5e-02	1	1.09	2.1e-02	1
	1/64	4	ILU	0.58	2.2e-01	3.8	3.61	2.3e-02	2.3	17.41	4.5e-03	2.9
			ℓ-AIR	0.43	2.2e-01	1	2.14	2.3e-02	1	8.05	4.5e-03	1
	1/128	5	ILU	5.08	1.6e-01	3	29.00	6.2e-03	2	111.16	6.7e-04	2
			ℓ-AIR	2.89	1.6e-01	1	16.87	6.2e-03	1	65.93	6.7e-04	1
	1/256	6	ILU	39.46	1.1e-01	2.4	239.12	1.6e-03	2	869.66	8.7e-05	2
			ℓ-AIR	25.28	1.1e-01	1	150.11	1.6e-03	1	558.15	8.7e-05	1

Table 14: Total time to solution, error in the solution and average iteration counts for the preconditioned linear system solves for L-stable time-stepping schemes on the advection problem with final time T, time-step Δt and mesh refinement parameter r. The computations use 64 processors.

				Implicit Midpoint ($p = 2$)			SDIRK23 ($p = 3$)			SDIRK34 ($p = 4$)		
T	Δt	r	Prec.	Time	Error	It.	Time	Error	It.	Time	Error	It.
2	1/32	3	ILU	0.07	4.2e-02	3	0.38	1.8e-02	3	1.57	1.5e-02	4
			ℓ-AIR	0.08	4.2e-02	1	0.28	1.8e-02	1	0.72	1.5e-02	1
	1/64	4	ILU	0.61	1.2e-02	3	3.14	3.9e-03	3	11.54	2.4e-03	3.5
			ℓ-AIR	0.35	1.2e-02	1	1.57	3.9e-03	1	4.91	2.4e-03	1
	1/128	5	ILU	3.77	3.2e-03	2	23.24	5.8e-04	2.7	79.97	2.1e-04	3
			ℓ-AIR	2.28	3.2e-03	1	11.65	5.8e-04	1	38.00	2.1e-04	1
	1/256	6	ILU	30.26	8.1e-04	2	146.59	7.5e-05	2	559.75	1.4e-05	2.5
			ℓ-AIR	19.78	8.1e-04	1	96.73	7.5e-05	1	311.01	1.4e-05	1
8	1/32	3	ILU	0.20	1.0e-01	3	1.37	4.1e-02	3	5.99	3.2e-02	4
			ℓ-AIR	0.17	1.0e-01	1	0.79	4.1e-02	1	2.42	3.2e-02	1
	1/64	4	ILU	2.22	4.3e-02	3	11.92	1.2e-02	3	45.02	7.5e-03	3.5
			ℓ-AIR	1.14	4.3e-02	1	5.66	1.2e-02	1	18.50	7.5e-03	1
	1/128	5	ILU	14.52	1.3e-02	2	90.56	2.2e-03	2.7	314.59	8.1e-04	3
			ℓ-AIR	8.50	1.3e-02	1	44.91	2.2e-03	1	147.43	8.1e-04	1
	1/256	6	ILU	120.20	3.2e-03	2	581.34	3.0e-04	2	2230.29	5.6e-05	2.5
			ℓ-AIR	76.00	3.2e-03	1	380.21	3.0e-04	1	1231.55	5.6e-05	1

Table 15: Total time to solution, error in the solution and average iteration counts for the preconditioned linear system solves for A-stable time-stepping schemes on the advection problem with final time T, time-step Δt and mesh refinement parameter r. The computations use 64 processors.

5.3 Time-dependent advection–diffusion problem

We consider in this section the scalar advection-diffusion equation,

$$\partial_t u + \alpha \cdot \nabla f(u) = \nabla \cdot (\mu \cdot \nabla u) + s(t,x,y), \qquad (t,x,y) \in (0,T] \times (-1,1)^2 \tag{6a}$$

$$u(0,x,y) = u_0(x,y) \qquad (x,y) \in (-1,1)^2 \tag{6b}$$

for constant vectors $\alpha = (0.85, 1.0)^\top$ and $\mu = (0.3, 0.25)^\top$ and final time $T > 0$. An initial condition is given as

$$u(0,x,y) = u_0(x,y) = \left(\sin(\frac{\pi}{2}(x-1))\right)^4 \left(\sin(\frac{\pi}{2}(y-1))\right)^4,$$

and periodic spatial boundaries are used. Two options for the advection flux f are considered, a linear function $f(u) = u$ that yields a linear PDE, and a quadratic function $f(u) = u^2$ that yields a nonlinear PDE. Test problems with manufactured solutions are implemented to facilitate convergence studies. The manufactured solution is given as:

$$u(t,x,y) = \left(\sin\left(\frac{\pi}{2}(x-1-\alpha_1 t)\right)\sin\left(\frac{\pi}{2}\left(y-1-\alpha_2 t\right)\right)\right)^4 e^{-(\mu_1+\mu_2)t},$$

so that the test problem represents a propagation and dissipation of the initial condition with wave speed α and diffusivity μ. The solution-independent term s is computed correspondingly to the exact solution and the function f.

We use am arbitrarily high-order finite difference method in 2D. In all of the experiments, we use the preconditioner BoomerAMG for solving the arising linear systems. As the actual implementation of the fully implicit RK methods does not support using LOR, we employ BoomerAMG on the fully assembled system.

In order to compare the different families of time-stepping schemes using different orders, we choose to adjust the spatial discretisation so that all methods end up with a fixed number of DOF per stage. Therefore, we had only selected the time-stepping schemes having one of the orders $\{2,4,8\}$. That is, we do not include results for the Gauss-Radau family.

5.3.1 The linear case

Table 16 presents the weighted average Krylov iteration count per time step to solve (using a weight of 1 for a 1×1 block diagonal system corresponding to a real eigenvalue of the Butcher table matrix, and a weight of 2 for 2×2 block diagonal system corresponding to the pair of complex eigenvalues of the Butcher table matrix), the total run time, and the L_2-error norm. The results in the table demonstrate that combining high-order discretisation in space and time achieves its most benefit when the time stepping scheme is fully implicit. Indeed, for orders 2, 4, and 8, fully implicit RK methods (IRK-Gauss and IRK-LobIIIC) achieve the best accuracy. Furthermore, the IRK-Gauss method is the fastest for all orders.

Scheme	Stability	Order	Avg. It.	Error	Tot. time
SDIRK	A	4	21	2.83692e-07	26.688
SDIRK	L	2	14	4.65815e-06	68.837
SDIRK	L	4	35	2.72948e-09	41.372
IRK-Gauss	L	2	7	8.38129e-06	35.141
IRK-Gauss	L	4	18	2.64616e-09	18.690
IRK-Gauss	L	8	44	1.35212e-11	**13.814**
IRK-LobIIIC	L	2	32	1.37162e-05	88.737
IRK-LobIIIC	L	4	31	4.03505e-09	31.775
IRK-LobIIIC	L	8	58	7.00163e-12	17.723

Table 16: Time window = 2 seconds. L_2-error norm and total time comparison between different time-stepping schemes. Linear advection–diffusion test case discretised with higher order finite difference in space (order equal to the order of the time stepping scheme). The time step is chosen as $\Delta t = 2h$, where h is the mesh size. The number of DOFs in space is the same for all cases and equal 4,194,304. All tests run on 64 processors.

Table 17 presents a scalability study of the fully implicit RK method IRK-Gauss of order 8 applied to the linear time-dependent advection-diffusion equation. The grid size is 1024×1024. The size of the problem is fixed and the number of processors is quadrupled starting from 16 up to 1024. Although the number of iterations is

# Processors	16	64	256	1024
Tot. time	1432.4	486.0	143.1	84.7
Gain factor	1	2.95	10.0	16.9
Avg. It.	42	42	42	44

Table 17: Number of time steps = 512.Total time is reported for the strong scalability test study using IRK-Gauss (L-stable) of order 8. The linear problem is discretised on a grid of size 1024×1024 with the order 8 finite difference method. The number of processors increased by a factor of 4. The gain factor is the proportion of the total time on P processors divided by the total time on 16 processors. The L_2-error = 6.2×10^{-12}.

robust with respect to the number of processors, the scalability is not perfect; the most probable reason is the overhead incurred by assembling the full matrix. As mentioned earlier, the current implementation of the fully implicit RK method does not allow the use of a LOR preconditioner. We expect that scalability can be achieved once a matrix-free method is used and the LOR preconditioner is combined with AMG. We have come to this conclusion for two reasons. First, in Section 4 we demonstrated that the LOR-AMG preconditioner scales very well for stationary problems. Second, in Table 18 we give the results of a scalability study of the IRK-Gauss method of order 8, but using a discretisation in space of order 2; in this regime the assembly of the matrix and the set up of the preconditioner do not suffer from the overhead of the high-order discretisation in space, and the gain factor demonstrates this accordingly.

# Processors	16	64	256	1024
Tot. time	469.1	146.7	33.3	9.3
Gain factor	1	3.2	14.0	50.4
Avg. It.	38	40	40	42

Table 18: Number of time steps = 21. Total time is reported for the strong scalability test study using IRK-Gauss (L-stable) of order 8. The linear problem is discretised on a grid of size 4096×4096 with the order 2 finite difference method. The number of processors increased by a factor of 4. The gain factor is the proportion of the total time on P processors divided by the total time on 16 processors. The L_2-error = 3.3×10^{-8}.

5.3.2 The nonlinear case

Table 19 presents the weighted average Krylov iteration count per time step to solve (using a weight of 1 for 1×1 block diagonal system corresponding to a real eigenvalue of the Butcher table matrix, and a weight of 2 for 2×2 block diagonal system corresponding to the pair of complex eigenvalues of the Butcher table matrix), the total run time, and the L_2-error norm for our non-linear test case. As for the linear case, the results in the table demonstrate that combining high-order discretisation in space and time achieves its most benefit when the time stepping scheme is fully implicit. The results here follow the same pattern as in the linear case: for orders 2, 4, and 8, fully implicit RK methods (IRK-Gauss and IRK-LobIIIC) achieve the best accuracy, and the IRK-Gauss method is again the fastest for all orders.

Table 20 contains a scalability study of the fully implicit RK method IRK-Gauss of order 8 applied to the nonlinear time-dependent advection-diffusion equation. The grid size is 1024×1024. The size of the problem is fixed and the number of processors is quadrupled starting from 16 up to 1024.

Although the number of iterations is robust with respect to the number of processors, the scalability is not perfect. As with the linear case, the reason for not achieving perfect scalability is probably due to the overhead incurred by assembling the full matrix, and we expect that scalability can be achieved once a matrix-free method is used and the LOR preconditioner is combined with AMG. We again run perform a scalability study of the IRK-Gauss method of order 8, but using a discretisation in space of order 2, giving the results in Table 21. As we saw in the linear case, the scalability is more favourable in this regime, where the assembly of the matrix and the set up of the preconditioner do not suffer from the overhead of the high-order discretisation in space.

Scheme	Stability	Order	Avg Newton It.	Avg. Krylov It.	Error	Tot. time
SDIRK	A	4	2.1	41	1.74236e-07	36.2014
SDIRK	L	2	2	24	3.02704e-06	94.3833
SDIRK	L	4	2	64	1.96809e-09	53.5953
IRK-Gauss	L	2	2	12	5.63955e-06	47.8599
IRK-Gauss	L	4	2	34	2.24606e-09	25.3041
IRK-Gauss	L	8	2.6	94	7.59273e-09	**18.9245**
IRK-LobIIIC	L	2	2	40	9.63594e-06	130.601
IRK-LobIIIC	L	4	2	57	2.73404e-09	44.7401
IRK-LobIIIC	L	8	2.7	132	7.34143e-09	27.0011

Table 19: Time window = 2 seconds. L_2-error norm and total time comparison between different time-stepping schemes. Nonlinear advection–diffusion test case discretised with higher order finite difference in space (order equal to the order of the time stepping scheme). The time step is chosen as $\Delta t = 2h$, where h is the mesh size. The number of DOFs in space is the same for all cases and equal $4,194,304$. All tests run on 64 processors.

# Processors	16	64	256	1024
Tot. time	2069.2	722.2	199.3	113.8
Gain factor	1	2.9	10.4	18.2
Avg. It.	72	74	72	78

Table 20: Number of time steps = 512. Total time is reported for the strong scalability test study using IRK-Gauss (L-stable) of order 8. The nonlinear problem is discretised on a grid of size 1024×1024. The number of processors increased by a factor of 4. The gain factor is the proportion of the total time on P processors divided by the total time on 16 processors. The L_2-error norm is 3.1×10^{-11}. The number of Newton iterations per time step is 2.

# Processors	16	64	256	1024
Tot. time	665.5	220.5	49.6	13.8
Gain factor	1	3.0	13.4	48.2
Avg. It.	68	70	70	72

Table 21: Number of time steps = 21. Total time is reported for the strong scalability test study using IRK-Gauss (L-stable) of order 8. The nonlinear problem is discretised on a grid of size 4096×4096 with the order 2 finite difference method. The number of processors increased by a factor of 4. The gain factor is the proportion of the total time on P processors divided by the total time on 16 processors. The L_2-error = 4.8×10^{-8}.

6 Conclusions

In this report, we presented a study around preconditioning linear systems arising from high-order discretisation (in space and time) of stationary and time-dependent PDEs. Through the extensive numerical experiments we performed, we observed the following: regarding spatial discretisation, recent techniques for matrix-free preconditioning are effective, efficient, and scalable. The p-MG method is the most effective for isotropic diffusion equations. However, once anisotropy is present either in the mesh or in the PDE coefficient, p-MG struggles. The LOR preconditioner combined with any available efficient preconditioner for linear systems arising from low-order discretised PDEs proved to be scalable and efficient for highly anisotropic diffusion equations and advection-diffusion ones.

Concerning temporal discretisation, we observed that high-order fully implicit time-stepping schemes yield high accuracy when combined with high-order spatial discretisation. The Gauss-Legendre family with orders 2, 4 and 8 was the most efficient in terms of run time and accuracy when compared to other time-stepping families with the same order.

Summary

- The most promising method for solving matrices from high order elliptic PDEs is to precondition with a low order finite element operator: low-order solvers have become robust enough to solve the resulting (challenging) linear systems[29].
- Multigrid and Domain Decomposition-based preconditioners are the leading contenders for performant parallel iterative linear solvers on anisotropic elliptic problems of low order.
- There are several excellent algebraic multigrid packages available that are highly parallel. However, to get good performance, it is vital to tune the parameters for a given problem. A well-implemented geometric multigrid method would give superior performance to an algebraic multigrid.
- The HPDDM domain decomposition method, based on the GenEO method [35], seemed promising with a good trade-off between performance and ease of setup; the recent development of fully algebraic variations [2, 3] make this even more the case. However, implementation improvements for these methods are needed for them to compete against highly-tuned multigrid implementations.
- Higher order fully implicit time-stepping schemes show spectacular performance in terms of run time and accuracy when compared to other higher-order time-stepping schemes.

References

[1] H. Al Daas, L. Grigori, P. Jolivet, and P.-H. Tournier. A multilevel Schwarz preconditioner based on a hierarchy of robust coarse spaces. *SIAM Journal on Scientific Computing*, 43(3):A1907–A1928, 2021/07/01 2021.

[2] H. Al Daas, P. Jolivet, and T. Rees. Efficient algebraic two-level schwarz preconditioner for sparse matrices, 2022.

[3] H. Al Daas, P. Jolivet, and J. A. Scott. A robust algebraic domain decomposition preconditioner for sparse normal equations, 2022. In press. preprint:https://arxiv.org/abs/2107.09006.

[4] V. Alexandrov and O. A. Esquivel-Flores. Towards Monte Carlo preconditioning approach and hybrid Monte Carlo algorithms for Matrix Computations. *Computers & Mathematics with Applications*, 70(11): 2709–2718, Dec. 2015. ISSN 0898-1221. doi: 10.1016/j.camwa.2015.08.035.

[5] V. Alexandrov, A. Lebedev, E. Sahin, and S. Thorne. Linear systems of equations and preconditioners relating to the NEPTUNE Programme. NEPTUNE Technical Report 2047353-TN-04, CCFE Culham, Apr. 2021.

[6] V. N. Alexandrov. Efficient parallel Monte Carlo methods for matrix computations. *Mathematics and computers in Simulation*, 47(2-5):113–122, 1998.

[7] R. Anderson, J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. C. V. Dobrev, Y. Dudouit, A. Fisher, T. Kolev, W. Pazner, M. Stowell, V. Tomov, I. Akkerman, J. Dahm, D. Medina, and S. Zampini. MFEM: A modular finite element methods library. *Computers & Mathematics with Applications*, 81:42–74, 2021. doi: 10.1016/j.camwa.2020.06.009.

[8] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri. Implicit-Explicit Runge-Kutta methods for time-dependent partial differential equations. *Applied Numerical Mathematics*, 25(2):151–167, Nov. 1997. ISSN 0168-9274. doi: 10.1016/S0168-9274(97)00056-1.

[9] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, D. Karpeyev, D. Kaushik, M. Knepley, D. May, L. Curfman McInnes, R. Mills, T. Munson, K. Rupp, P. Sanan, B. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc Users Manual. 2019.

[10] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial.* SIAM, second edition, 2000. ISBN 978-0-898714-62-3.

[11] P. D. Brubeck and P. E. Farrell. A scalable and robust vertex-star relaxation for high-order FEM, 2021.

[12] C. Canuto, M. Y. Hussaini, A. Quarteroni, and T. A. Zang. *Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics.* Springer Science & Business Media, 2007. doi: 10.1007/978-3-540-30728-0.

[13] H. A. Daas, T. Rees, E. Sahin, and H. S. Thorne. A Review of Time Stepping Techniques and Preconditioning for Hyperbolic and Anisotropic Elliptic Problems. Technical Report 2060049-TN-01, UKAEA, 2022.

[14] V. Dolean, P. Jolivet, and F. Nataf. *An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation.* SIAM, Dec. 2015. ISBN 978-1-61197-405-8.

[15] H. Elman, D. Silvester, and A. Wathen. *Finite Elements and Fast Iterative Solvers: With Applications in Incompressible Fluid Dynamics.* Numerical Mathematics and Scientific Computation. Oxford University Press, Oxford, second edition, 2014. ISBN 0-19-967880-X.

[16] S. Friedhoff and B. S. Southworth. On "optimal" h-independent convergence of parareal and multigrid-reduction-in-time using runge-kutta time integration. *Numerical Linear Algebra with Applications*, 28(3): e2301, 2021.

[17] A. Greenbaum. *Iterative Methods for Solving Linear Systems.* SIAM, Philadelphia, 1997.

[18] M. J. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853, 1997.

[19] V. E. Henson and U. M. Yang. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41(1):155–177, Apr. 2002. ISSN 0168-9274. doi: 10.1016/S0168-9274(01)00115-5.

[20] M. R. Hestenes and E. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *J. Res. Nat. Bur. Stand.*, 49:409–436, 1952.

[21] T. J. R. Hughes and A. N. Brooks. A multi-dimensional upwind scheme with no crosswind diffusion. In T. J. R. Hughes, editor, *Finite Element Methods for Convection Dominated Flows*, volume 34 of *AMD*, pages 19–35. ASME, New York, 1979.

[22] P. Jolivet, J. E. Roman, and S. Zampini. KSPHPDDM and PCHPDDM: Extending PETSc with advanced Krylov methods and robust multilevel overlapping Schwarz preconditioners. *Computers & Mathematics with Applications*, 84:277–295, Feb. 2021. ISSN 0898-1221. doi: 10.1016/j.camwa.2021.01.003.

[23] C. A. Kennedy and M. H. Carpenter. Diagonally Implicit Runge-Kutta Methods for Ordinary Differential Equations. A Review. Technical Report NF1676L-19716, Mar. 2016.

[24] D. S. Kershaw. Differencing of the diffusion equation in Lagrangian hydrodynamic codes. *Journal of Computational Physics*, 39(2):375–395, Feb. 1981. ISSN 0021-9991. doi: 10.1016/0021-9991(81)90158-3.

[25] T. Kolev. CEED-MS36: High-order algorithmic developments and optimizations for large-scale GPU-accelerated simulations. Technical Report LLNL-TR-821011, Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States), Mar. 2021.

[26] T. Manteuffel, J. Ruge, and B. Southworth. Nonsymmetric algebraic multigrid based on local approximate ideal restriction (*l*air). *SIAM Journal on Scientific Computing*, 6(40):4105–4130, 2018.

[27] S. A. Orszag. Spectral methods for problems in complex geometries. *Journal of Computational Physics*, 37 (1):70–92, Aug. 1980. ISSN 0021-9991. doi: 10.1016/0021-9991(80)90005-4.

[28] W. Pazner. Efficient low-order refined preconditioners for high-order matrix-free continuous and discontinuous galerkin methods. *SIAM Journal on Scientific Computing*, 42(5):A3055–A3083, 2022/03/17 2020.

[29] W. Pazner, T. Kolev, and C. Dohrmann. Low-order preconditioning for the high-order finite element de rham complex, 2022.

[30] J. W. Ruge and K. Stüben. Algebraic multigrid. In *Multigrid Methods*, chapter 4, pages 73–130. SIAM, 1987. doi: 10.1137/1.9781611971057.ch4.

[31] Y. Saad and M. H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, 1986. doi: 10.1137/0907058.

[32] E. Sahin, A. Lebedev, M. Abaļenkovs, and V. Alexandrov. Usability of Markov Chain Monte Carlo Preconditioners in Practical Problems. In *2021 12th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, pages 44–49, Nov. 2021. doi: 10.1109/ScalA54577.2021.00011.

[33] B. S. Southworth, O. Krzysik, and W. Pazner. Fast solution of fully implicit runge–kutta and discontinuous galerkin in time for numerical pdes, part ii: Nonlinearities and daes. *SIAM Journal on Scientific Computing*, 44(2):A636–A663, 2022.

[34] B. S. Southworth, O. Krzysik, W. Pazner, and H. D. Sterck. Fast Solution of Fully Implicit Runge–Kutta and Discontinuous Galerkin in Time for Numerical PDEs, Part I: the Linear Setting. *SIAM Journal on Scientific Computing*, 44(1):A416–A443, 2022.

[35] N. Spillane, V. Dolean, P. Hauret, F. Nataf, C. Pechstein, and R. Scheichl. Abstract robust coarse spaces for systems of PDEs via generalized eigenproblems in the overlaps. *Numerische Mathematik*, 126(4):741–770, 2014. ISSN 0945-3245.

[36] A. J. Wathen. Preconditioning. *Acta Numerica*, 24:329–376, May 2015. ISSN 0962-4929, 1474-0508. doi: 10.1017/S0962492915000021.

[37] J. Xu and L. Zikatanov. Algebraic multigrid methods. *Acta Numerica*, 26:591–721, May 2017. ISSN 0962-4929, 1474-0508. doi: 10.1017/S0962492917000083.

A Appendix: HPDDM

In this appendix, we describe some of the technical details of the setup of the HPDDM preconditioner. In particular, we describe the cause of the lack of scalability observed in Figure 3 and provide a suggestion about how it can be overcome.

The current implementation of HPDDM associates a subdomain on the finest level with each processor. For subsequent levels, the user has to specify the number of processors handling each level. Using $L+1$ levels, $L > 0$, each processor handling a subdomain on level $l \leq L$ performs the following:

1. Factor the local matrix and the local auxiliary matrix using a sparse direct solver.
2. Compute a specified number of generalised eigenvectors locally.
3. Assemble the local part of the next level matrix.
4. If involved in handling the next level, receive information from processors in the same aggregate. Otherwise, send data to the representing processor on the next level.
5. If $l < L$, compute the local auxiliary matrices required in the next level
6. and if $l = L$ and the processor is involved in handling the following level, it participates in factoring the coarse matrix at level $L+1$.

The first three steps are entirely local and so they scale optimally. The third step involves local computation and moving data with neighbouring subdomains and should scale optimally as well. However, we noticed that—especially when the number of DOFs per processor is small—this very step is dominating the computation time during the setup phase. The main reason for this step being not scalable is purely implementation related. The PETSc interface to HPDDM does not, for the time being, allow data gathering to be performed optimally, namely via a binary tree communication. Furthermore, the computations involved in this step, as well as the previous step, ($P^T AP$ where A and P are sparse) are optimised based on the sparsity pattern only. However, in practice the column vectors in P stem from local generalised eigenvalue problems involving the diagonal blocks in A. Therefore, if D is the block diagonal matrix obtained from A and $A = D + B$, we have

$P^T AP = P^T DP + P^T BP$, where $P^T DP$ is a diagonal matrix in exact arithmetic and $P^T BP$ is a hugely-sparse matrix containing a small number of dense blocks. As for the current implementation the matrix $P^T AP$ stores entries based on the sparsity patterns of A, since the sparsity pattern of P is tightly related to that of A.

The upshot of this is that a huge reduction in the density of the matrices on levels > 1 can be achieved by considering the origin of the column vectors in P. This would lead to:

- a reduction in the computation cost in steps 1, 2, and 3 in the next level as well as steps 5 and 6 on the same level, and
- a reduction in the amount of data sent/received in step 4.

We believe this would make the HPDDM implementation more efficient when applied to large numbers of processors.