Report 2053622-TN-04: Evaluation of preconditioners of matrix-free operations in Nektar++ for anisotropic heat transport

Bin Liu, Chris Cantwell & Spencer Sherwin, Imperial College London
Mashy Green & David Moxey, King's College London

30th June 2022

1 Introduction

The numerical solution of the diffusion equation with anisotropy plays an important role in the simulation of plasma physics. The high degree of anisotropy in such problems can lead to numerical challenges in solving such systems using iterative Krylov-based algorithms, due to the resulting poor conditioning of the matrices. To address this, preconditioners are often employed to effectively reduce the condition number of the system being solved and thereby significantly reduce the number of iterations required for convergence and ultimately the time to solution. The choice of preconditioner may be governed by a number of factors, including the structure of the matrix and therefore the effectiveness of the preconditioner to improve the conditioning of the matrix, as well as the computational cost of evaluating the preconditioner.

In this report, we evaluate the performance of the preconditioners currently available, or under development, in Nektar++ [1] for solving the two-dimensional steady anisotropic thermal conduction problem. We assess their performance in terms of iteration count as well as computational cost.

2 Problem formulation

The general form for heat conduction in plasma is

$$\frac{3}{2}n\frac{dT}{dt} = \nabla \cdot (\boldsymbol{\kappa}_s \nabla T) + Q \tag{1}$$

Due to the ionized nature of plasma and the presence of a strong magnetic field, the dynamics of charged particles and the associated energy dissipation is highly anisotropic. Such

particles, e.g., electrons and ions, move rapidly in tight spiral orbits, known as gyro-orbits, along the magnetic field lines, but have only a small component of velocity along the normal direction to the magnetic field, e.g., $\kappa_\parallel \gg \kappa_\perp$, typically 10 orders of magnitude difference or more. The numerical consequence of anisotropic thermal conduction in the magnetized plasma is that the thermal conduction tensor, $\boldsymbol{\kappa}_s$, becomes non-diagonal. The temperature gradient is decomposed into three components/auxiliary vectors [2], defined with respect to the unit direction of magnetic field $\boldsymbol{b} = \boldsymbol{B}/|\boldsymbol{B}|$ as

$$\nabla_\parallel T = \boldsymbol{b}(\boldsymbol{b}\cdot\nabla T), \qquad \nabla_\perp T = (\boldsymbol{b}\times\nabla T)\times\boldsymbol{b}, \qquad \nabla_\wedge T = \boldsymbol{b}\times\nabla T, \tag{2}$$

where $\nabla T = \nabla_\parallel T + \nabla_\perp T$. $\nabla_\parallel T$ and $\nabla_\perp T$ are the auxiliary vectors along and normal to the magnetic field on the $\boldsymbol{b}-\nabla T$ plane respectively. The auxiliary vector $\nabla_\wedge T$ accounts for the direction of electromagnetic induction normal to the $\boldsymbol{b}-\nabla T$ plane.

It is shown by Goedbloed & Poedts [2] that if a second rank tensor $\boldsymbol{\kappa}_s$ representing the anisotropic transport coefficients is symmetric with respect to rotations about the magnetic field $\boldsymbol{b}$, it implies that $\boldsymbol{\kappa}_s$ can only have three independent elements, e.g., $\kappa_\parallel$, $\kappa_\perp$ and $\kappa_\wedge$, similar to the rotational tensor and have the form

$$\boldsymbol{\kappa}_s = \begin{bmatrix} \kappa_\perp & -\kappa_\wedge & 0 \\ \kappa_\wedge & \kappa_\perp & 0 \\ 0 & 0 & \kappa_\parallel \end{bmatrix}. \tag{3}$$

Assuming the magnetic field is parallel to the third axis of ∇T and the temperature gradient can be expressed as $\nabla T = (\partial_1 T)\hat{\boldsymbol{e}}_1 + (\partial_2 T)\hat{\boldsymbol{e}}_2 + (\partial_\parallel T)\hat{\boldsymbol{e}}_\parallel$ and

$$\begin{aligned} \nabla_\parallel T &= (\partial_\parallel T)\hat{\boldsymbol{e}}_\parallel, & (4)\\ \nabla_\wedge T &= -(\partial_2 T)\hat{\boldsymbol{e}}_1 + (\partial_1 T)\hat{\boldsymbol{e}}_2, & \\ \nabla_\perp T &= (\partial_1 T)\hat{\boldsymbol{e}}_1 + (\partial_2 T)\hat{\boldsymbol{e}}_2. & (5) \end{aligned}$$

Subsequently the tensor and vector product can be written as

$$\begin{aligned} \boldsymbol{\kappa}_s\cdot\nabla T &= \begin{bmatrix} \kappa_\perp & -\kappa_\wedge & 0 \\ \kappa_\wedge & \kappa_\perp & 0 \\ 0 & 0 & \kappa_\parallel \end{bmatrix} \cdot \begin{bmatrix} \partial_1 T \\ \partial_2 T \\ \partial_\parallel T \end{bmatrix} \\ &= \big((\kappa_\perp\partial_1 T) - (\kappa_\wedge\partial_2 T)\big)\hat{\boldsymbol{e}}_1 + \big((\kappa_\wedge\partial_1 T) + (\kappa_\perp\partial_2 T)\big)\hat{\boldsymbol{e}}_2 \\ &\quad (\kappa_\parallel\partial_\parallel T)\hat{\boldsymbol{e}}_\parallel \\ &= \kappa_\parallel\nabla_\parallel T + \kappa_\wedge\nabla_\wedge T + \kappa_\perp\nabla_\perp T \\ &= \kappa_\parallel\boldsymbol{b}(\boldsymbol{b}\cdot\nabla T) + \kappa_\perp\big(\nabla T - \boldsymbol{b}(\boldsymbol{b}\cdot\nabla T)\big) + \kappa_\wedge\boldsymbol{b}\times\nabla T. \end{aligned} \tag{6}$$

Therefore the general form of anisotropic thermal conduction of magnetized plasma in Equation (1) can be recast into the form

$$\begin{aligned} \frac{3}{2}n\frac{dT}{dt} &= \nabla\cdot\Big[\kappa_\parallel\boldsymbol{b}(\boldsymbol{b}\cdot\nabla T) + \kappa_\perp\big(\nabla T - \boldsymbol{b}(\boldsymbol{b}\cdot\nabla T)\big) + \kappa_\wedge\boldsymbol{b}\times\nabla T\Big] + Q \\ &= \nabla\cdot\Big[\kappa_\parallel(\boldsymbol{b}\otimes\boldsymbol{b})\cdot\nabla T + \kappa_\perp(\boldsymbol{I} - \boldsymbol{b}\otimes\boldsymbol{b})\cdot\nabla T + \kappa_\wedge\boldsymbol{b}\times\nabla T\Big] + Q \\ &= \nabla\cdot\Big[\big((\kappa_\parallel - \kappa_\perp)(\boldsymbol{b}\otimes\boldsymbol{b}) + \kappa_\perp\boldsymbol{I}\big)\cdot\nabla T\Big] + \nabla\cdot[\kappa_\wedge\boldsymbol{b}\times\nabla T] + Q \\ &= \nabla\cdot\big(\boldsymbol{\kappa}_c\cdot\nabla T\big) + \nabla\cdot[\kappa_\wedge\boldsymbol{b}\times\nabla T] + Q, \end{aligned} \tag{7}$$

where $\boldsymbol{I}$ is an identity matrix. $\boldsymbol{\kappa}_c$ is defined as the thermal conductivity tensor,

$$\begin{aligned}\boldsymbol{\kappa}_c &= (\kappa_\parallel - \kappa_\perp)(\boldsymbol{b} \otimes \boldsymbol{b}) + \kappa_\perp \boldsymbol{I} \\ &= (\kappa_\parallel - \kappa_\perp) \begin{bmatrix} b_x^2 & b_x b_y \\ b_x b_y & b_y^2 \end{bmatrix} + \begin{bmatrix} \kappa_\perp & 0 \\ 0 & \kappa_\perp \end{bmatrix} \\ &= \begin{bmatrix} (\kappa_\parallel - \kappa_\perp) b_x^2 + \kappa_\perp & (\kappa_\parallel - \kappa_\perp) b_x b_y \\ (\kappa_\parallel - \kappa_\perp) b_x b_y & (\kappa_\parallel - \kappa_\perp) b_y^2 + \kappa_\perp \end{bmatrix}. \end{aligned} \tag{8}$$

If there is an angle θ defining the direction of the 2D magnetic field, $\boldsymbol{b}$ can be defined as $\boldsymbol{b} = [b_x, b_y]' = [cos(\theta), sin(\theta)]' = [cs, ss]'$, where the superscript ($'$) is a transpose operator. Consequently, $b_x^2 = cs^2$, $b_x b_y = cs\ ss$ and $b_y^2 = ss^2$.

In this study, since it is assumed that the anisotropic thermal conduction in magnetized plasma is two-dimensional, the induction direction $\kappa_\wedge$ in the third dimension (the term $\nabla \cdot [\kappa_\wedge \boldsymbol{b} \times \nabla T]$ in Equation (7)) is neglected. Therefore, the implemented strong form of two-dimensional anisotropic thermal conduction becomes

$$\frac{3}{2} n \frac{dT}{dt} = \nabla \cdot \begin{bmatrix} \big((\kappa_\parallel - \kappa_\perp) cs^2 + \kappa_\perp\big)\partial_x T + \big((\kappa_\parallel - \kappa_\perp) cs\ ss\big)\partial_y T \\ \big((\kappa_\parallel - \kappa_\perp) cs\ ss\big)\partial_x T + \big((\kappa_\parallel - \kappa_\perp) ss^2 + \kappa_\perp\big)\partial_y T \end{bmatrix} + Q. \tag{9}$$

For the purpose of this report to assess preconditioners, we consider the steady case where $\frac{\partial T}{\partial t} = 0$, that is

$$\nabla \cdot (\boldsymbol{\kappa}_c \cdot \nabla T) = Q. \tag{10}$$

2.1 Numerical Discretisation

In Nektar++, the anisotropic steady diffusion equation can be treated as an inhomogeneous Helmholtz equation with the scalar constant $\lambda = 0$, given by

$$\nabla \cdot (\boldsymbol{\kappa}_c \cdot \nabla T) - \lambda T = Q \tag{11}$$

where $\boldsymbol{\kappa}_c$, λ and Q are the anisotropic diffusion tensor, reaction coefficient and a forcing term in the field, respectively. Applying the Galerkin method of weighted residuals to Equation (11) and the divergence theorem on the Laplacian operator, its variational form satisfies

$$\begin{aligned}\int_\Omega [\nabla v^h (\boldsymbol{\kappa}_c \cdot \nabla T^h)] d\Omega + \lambda \int_\Omega [v^h T^h] d\Omega &= -\int_\Omega [v^h Q] d\Omega \\ &\quad + \int_\Gamma [v^h (\boldsymbol{\kappa}_c \cdot \nabla T^h) \cdot \boldsymbol{n}] d\Gamma \end{aligned} \tag{12}$$

for all test functions v^h. Now dropping the h superscript for clarity, we can expand T^h and v^h in terms of the basis functions on each element,

$$T^e = \sum_i \hat{T}^e \psi_i^e,$$

where the superscript ("e") refers to elemental quantities. Expressing as a matrix system, we arrive at

$$\boldsymbol{L}^e\hat{\boldsymbol{T}}^e + \lambda \boldsymbol{M}^e\hat{\boldsymbol{T}}^e = -\boldsymbol{B}\boldsymbol{W}^e\boldsymbol{Q}^e + \boldsymbol{\Gamma} \tag{13}$$

where the matrices $\boldsymbol{L}^e$, $\boldsymbol{M}^e$ and $\boldsymbol{B}^e$ respectively are the discrete representations of Laplacian, mass and basis evaluation operators. $\boldsymbol{W}^e$ is a diagonal matrix of quadrature weights and $\hat{\boldsymbol{T}}^e$ is the matrix of elemental basis functions ϕ_i evaluated at the quadrature points, ξ_j. For the non-zero surface flux, the vector $\boldsymbol{\Gamma}$ is introduced to represent the boundary integral term in equation (12). In terms of fundamental operators, $\boldsymbol{L}^e$ and $\boldsymbol{M}^e$ can be written as

$$\boldsymbol{L}^e = (\boldsymbol{D}^e\boldsymbol{B})\boldsymbol{g}^e(\boldsymbol{D}^e\boldsymbol{B})^T\boldsymbol{W}^e \tag{14a}$$

$$\boldsymbol{M}^e = \boldsymbol{B}\boldsymbol{W}^e(\boldsymbol{B})^T \tag{14b}$$

where $\boldsymbol{D}$, $\boldsymbol{B}$ and $\boldsymbol{W}$ respectively are the one-dimensional derivative matrix, basis matrix and diagonal weight of numerical integration in the standard region element $\xi \subset [-1,1]^d$ (uniquely defined for each element type). The metric, $\boldsymbol{g}^e$, incorporates the diffusion tensor $\boldsymbol{\kappa}_c$ and the inverse of Jacobian matrix $\boldsymbol{J}$, as

$$\begin{aligned}
\boldsymbol{g}^e &= (\boldsymbol{J}^e)^{-1}\boldsymbol{\kappa}_c(\boldsymbol{J}^e)^{-T} = \begin{bmatrix} g_{00} & g_{01} \\ g_{10} & g_{11} \end{bmatrix} \\
\boldsymbol{\kappa}_c &= \begin{bmatrix} \kappa_{00} & \kappa_{01} \\ \kappa_{10} & \kappa_{11} \end{bmatrix}; \quad (\boldsymbol{J}^e)^{-1} = \begin{bmatrix} j_{00} & j_{01} \\ j_{10} & j_{11} \end{bmatrix} \\
g_{00} &= \kappa_{00}j_{00}^2 + 2\kappa_{01}j_{00}j_{01} + \kappa_{11}j_{01}^2 \\
g_{01} &= g_{10} = \kappa_{00}j_{00}j_{10} + \kappa_{01}(j_{01}j_{10} + j_{00}j_{11}) + \kappa_{11}j_{01}j_{11} \\
g_{11} &= \kappa_{00}j_{10}^2 + 2\kappa_{01}j_{10}j_{11} + \kappa_{11}j_{11}^2.
\end{aligned} \tag{15}$$

3 Implementation

The equations above are implemented in the nektar-diffusion solver, reported on previously [3]. In this section, we highlight the currently available preconditioners, as well as a p-multigrid preconditioner which is under development.

3.1 Matrix types

The global linear system constructed through the discretisation of the Helmholtz operator can be solved as a *full matrix* system explicitly, or by first *statically condensing* the matrix. The former solves for all degrees of freedom using a single linear system. The latter proceeds as follows. The degrees of freedom in each element are reordered to put those on the boundaries of the element first, followed by those on the interior. Due to the choice of modified Legendre basis, the interior degrees of freedom are naturally decoupled from neighbouring elements. Combining these block matrices from all elements, we construct an assembled element-block

matrix system consisting of a boundary-boundary block, and interior-interior block and two off-diagonal boundary-interior blocks. We can therefore express the system to solve as:

$$\begin{bmatrix} \boldsymbol{M}_b & \boldsymbol{M}_c \\ \boldsymbol{M}_c^\top & \boldsymbol{M}_i \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_b \\ \boldsymbol{x}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_b \\ \boldsymbol{f}_i \end{bmatrix}$$

Performing block elimination, we solve this system in two steps: first by using the iterative conjugate gradient solver with the Schur complement matrix to calculate the boundary degrees of freedom, then second by solving for the remaining interior degrees of freedom by trivially inverting each elemental block. Due to the reduced rank of the linear system, this approach generally provides improved performance over the full matrix system solve.

3.2 Available preconditioners

Within the Nektar++ framework a number of preconditioners are available to speed up the convergence rate of the conjugate gradient solver. The table below summarises each method, the dimensions of elements which are supported, and also the discretisation type support which can either be continuous (CG) or discontinuous (hybridizable DG). We also note which type of linear systems the preconditioners can be applied to.

Name	**Dimensions**	**Discretisations**	**Matrix-type**
None	All	All	All
Diagonal (Jacobi)	All	All	All
Block	2/3D	All	All
LowEnergyBlock	3D	CG	StaticCond
FullLinearSpace	All	CG	StaticCond
FullLinearSpaceWithDiagonal	All	CG	StaticCond
FullLinearSpaceWithBlock	2/3D	CG	StaticCond
FullLinearSpaceWithLowEnergyBlock	3D	CG	StaticCond

Table 1: Available preconditioners in Nektar++. Only those supporting 2D domains and continuous Galerkin projection are considered in this report. Two matrix-types can be used: (multi-level) static condensation and full matrix.

The default preconditioner with the iterative conjugate gradient solver in Nektar++ is the *Diagonal* preconditioner. The preconditioner and solution method can be chosen through the session file. For example, to enable *FullLinearSpace* preconditioner with the statically-condensed system one can use:

```
<I PROPERTY="GlobalSysSoln"    VALUE="IterativeStaticCond" />
<I PROPERTY="Preconditioner"   VALUE="FullLinearSpace" />
```

Alternatively one can have more control over different preconditioners for each solution field by using the *GlobalSysSoln* section of the XML file (for more details, consult the user guide of Nektar++). We now discuss the structure of each of the preconditioners.

3.2.1 Diagonal

Diagonal (or Jacobi) preconditioning is amongst the simplest preconditioning strategies and is particularly effective when the matrix is diagonally dominant. In this scheme one extracts the diagonal terms $h_i i$ of the matrix and forms the preconditioner as a diagonal matrix $\boldsymbol{M}^{-1} = (h_{ii}^{-1})$

3.2.2 Block

Block preconditioning of the C^0 continuous system is defined by

$$\boldsymbol{M}^{-1} = \begin{bmatrix} (\boldsymbol{S}_1^{-1})_{vv} & 0 & 0 \\ 0 & (\boldsymbol{S}_1^{-1})_{eb} & 0 \\ 0 & 0 & (\boldsymbol{S}_1^{-1})_{ef} \end{bmatrix} \tag{16}$$

where diag$[(\boldsymbol{S}_1)_{vv}]$ is the diagonal of the vertex modes, $(\boldsymbol{S}_1)_{eb}$ and $(\boldsymbol{S}_1)_{fb}$ are block diagonal matrices corresponding to coupling of an edge (or face) with itself i.e ignoring the coupling to other edges and faces. This preconditioner is best suited for two dimensional problems.

3.2.3 Linear space

The linear space (or coarse space) of the matrix system is that containing degrees of freedom corresponding only to the vertex modes in the high-order system. Preconditioning of this space is achieved by forming the matrix corresponding to the coarse space and inverting it, so that

$$\boldsymbol{M}^{-1} = (\boldsymbol{S}_1^{-1})_{vv} \tag{17}$$

Since the mesh associated with higher order methods is relatively coarse compared with tradi- tional finite element discretisations, the linear space can usually be directly inverted without memory issues. However such a methodology can be prohibitive on large parallel systems, due to a bottleneck in communication.

In Nektar++ the inversion of the linear space preconditioner is handled using the XX^T library [4]. XX^T is a parallel direct solver for problems of the form $\boldsymbol{A}\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}$ based around a sparse factorisation of the inverse of $\boldsymbol{A}$. To precondition utilising this methodology the linear sub- space is gathered from the expansion and the preconditioned residual within the CG routine is determined by solving

$$(\boldsymbol{S}_1)_{vv}\hat{\boldsymbol{z}} = \hat{\boldsymbol{r}} \tag{18}$$

The preconditioned residual $\hat{\boldsymbol{z}}$ is then scattered back to the respective location in the global degrees of freedom.

3.2.4 Geometrically informed Algebraic multigrid (AMG) preconditioner

It is well-known that multigrid methods are preferred method for solving the resulting symmetric linear system with optimal and mesh-independent convergence. However, when used with unstructured meshes, the use of geometric multigrid (GMG) is not straightforward, requiring the use of algebraic multigrid (AMG) instead. Unlike GMG, where the coarse-grid operators are typically obtained via geometric coarsening and rediscretizations, AMG does not use the geometric information but instead builds the coarse grid operators algebraically. Algebraic coarsening results in a loss of sparsity, leading to poor performance and scalability, especially for large-scale problems. This is usually addressed by employing sparsity control techniques, such as dropping values below a specified threshold at coarser levels. Such measures can address the sparsity issues, albeit with some loss of convergence rates. Additionally, they introduce an additional parameter that needs to be tuned, making the overall solver or preconditioner less robust. This issue is exacerbated for high-order operators that are denser to begin with, and get even denser on coarser grids, leading to poor scalability.

In some sense the linear space preconditioner outlined in Section 3.2 can be viewed as a form of multigrid, where in a single-level V-cycle the restriction operator is the extraction of linear modes and no smoothing is performed. However the reliance on $X^T X$ imposes natural scalability restrictions so that beyond 10^4 cores, and the lack of smoothing does not guarantee many of the convergence properties that a multigrid method should theoretically provide.

In work currently being developed[1], to circumvent this problem we are considering a geometrically informed algebraic multigrid approach (GIAMG), implemented in the *Saena* library, in which we perform two phases:

- p-coarsening: in which the problem is coarsened by reducing the polynomial order of the basis functions and degrees of freedom within each element, resulting in a local coarsening; followed by
- algebraic h-coarsening: also known as algebraic multigrid (AMG), in which the mesh is coarsened based on the specific matrix connectivity properties.

This is visualised in Figure 1. We note that for the grids obtained using p-coarsening, we effectively increase the sparsity as we go to coarser levels. This enables us to offset the loss of sparsity encountered when generating coarser levels using AMG. This plays a major part in ensuring that the overall scalability remains good. In addition, if the high-order system is generated using a modal basis, then the restriction operation is simply an injection, i.e., a lower order mode is injected to the coarser level. This indicates the prolongation and restriction operators will only have zero- or single-valued entries. As a result, it is extremely efficient to perform p-coarsening for such high-order systems. Note

[1] *A geometrically informed algebraic multigrid preconditioned iterative approach for solving high-order finite element systems.* S. Xu, M. Rasouli, R. M. Kirby, D. Moxey and H. Sundar, under review in Numerical Linear Algebra with Applications.

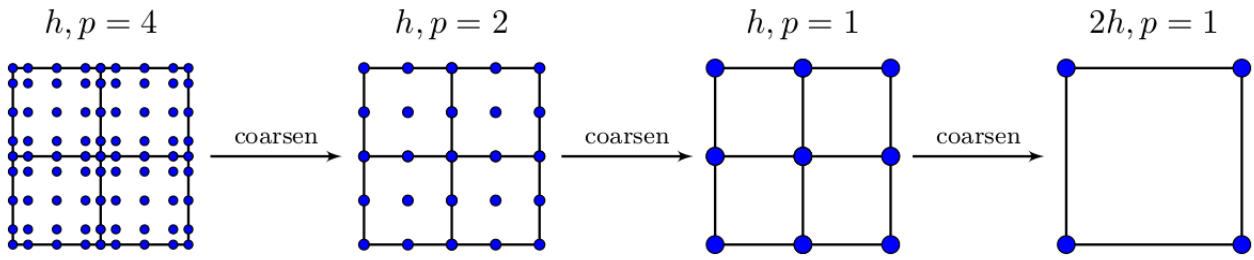

Figure 1: Illustration of the coarsening strategy of GIAMG for quadrilateral elements with nodal basis of polynomial order p = 4. First, We p-coarsen from p = 4 to p = 2, and then again p-coarsen from p = 2 to p = 1. The total number of elements—and therefore the mesh—does not change during p-coarsening. Once we reach linear elements, we perform h-coarsening to obtain a coarser mesh and smaller system.

when using modal basis, the sparsity of coarser level matrices is further reduced due to the injection during interpolation, compared to using a nodal basis.

The integration of the Saena code within Nektar++ is currently a work in progress, but our initial version of this can be found on the main Nektar++ repository under the `feature/saena` branch, which can be enabled through the `NEKTAR_USE_SAENA` CMake setting. Note that this is presently experimental and thus should be considered unstable. We also note that there are significant performance optimisations that are still required to make this production-ready, and thus walltimes may not be representative of the potential for this method at present. Our current focus is on attaining greater performance of the solver in parallel execution, the reduction of setup times in construction of the matrix systems, as well as other potential improvements such as the use of matrix-free methods within the V-cycle.

4 Comparison of available preconditioners

Preconditioners were assessed using the steady anisotropic heat transport problem described in 2. The computational domain is a unit square region, $(x, y) = ([0, 1], [0, 1])$, as shown in Figure 2. The traction-free boundary condition is imposed along the top, bottom and right boundaries. A hyperbolic tangent function is imposed along the left boundary to represent the profile of the SOL, the green line in Figure 2. The magnetic field direction $\boldsymbol{B}$ is shown as the red solid line, which has an angle θ with respect to the x-axis.

The performance of the preconditioners was assessed on the following metrics:

- the number of iterations required for the CG algorithm to converge;
- the total execution time for the solve;

The different preconditioners were compared with an anisotropy of $k_{\parallel} = 10^{24}$ and $k_{\perp} = 0$, with a transport angle of between 0° and 45° with respect to the x-axis. Greater angular

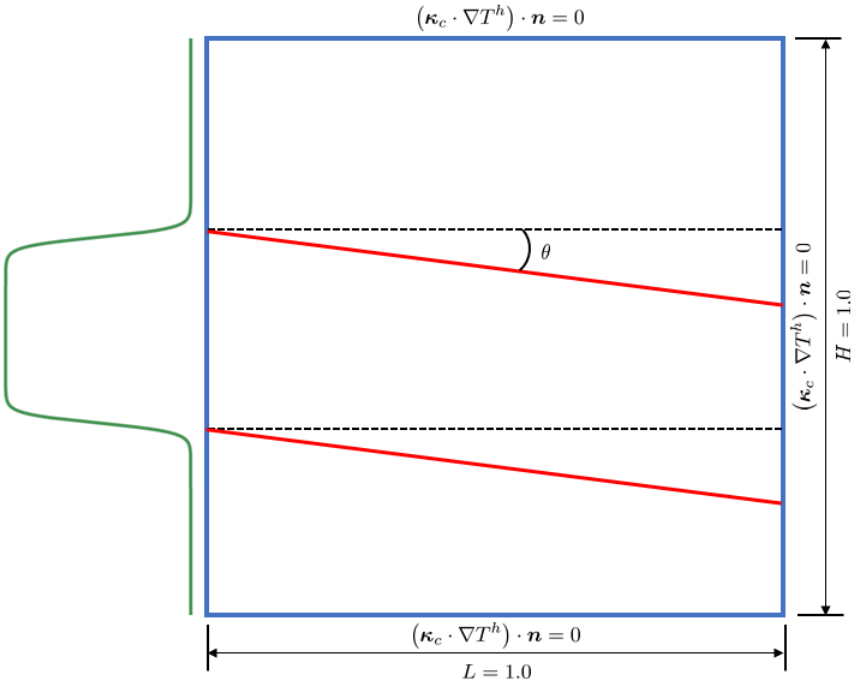

Figure 2: Schematic diagram of computational domain showing imposed temperature profile on the left boundary and angle of anisotropy. Neumann boundary conditions are imposed on the upper, lower and right boundaries.

resolution was used for small angles, since these are of most interest to the NEPTUNE project.

4.1 Experimental details

Results were obtained by executing the *nektar-diffusion* solver to solve the Helmholtz problem arising from the steady system. Preconditioners applied to the full matrix system and statically condensed system were assessed. For the full matrix system, the matrix-free kernels were used. These are not available for the statically condensed system. All tests use a polynomial order of 4. The specifications of the CPU hardware used to assess the performance are listed in Table 2. The solver was executed using 36 MPI ranks to fully utilise all floating-point units on the machine.

4.2 Full matrix system

Initially, we investigate the impact of the anisotropic ratio on the performance of the different preconditioners. The preconditioners investigated in this case are the diagonal preconditioner and the Saena p-multigrid preconditioner. The parallel diffusivity is fixed at 1.0, while the perpendicular diffusivity is reduced from 1.0 towards zero to obtain different anisotropic

Architecture	Intel(R) Xeon(R)
Model	E5-2697 v4
Clock speed	2.3 Ghz
L3 Cache size	46080 KB
Number of Sockets	2
Cores per socket	18

Table 2: Specification of CPU used for obtaining the results presented in this report.

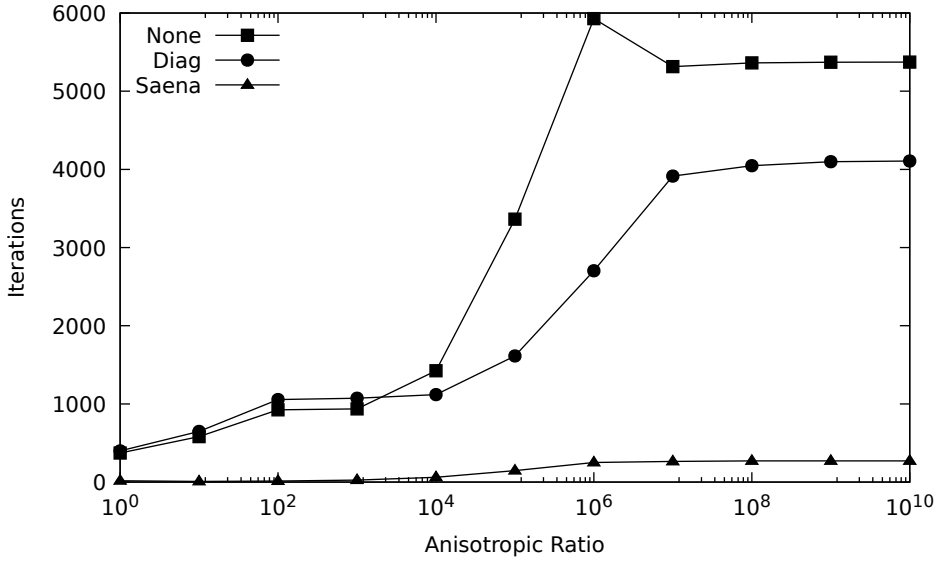

Figure 3: Full-matrix iteration count for different preconditioners as a function of anisotropic ratio.

ratios $k_{\parallel}/k_{\perp}$. The angle of diffusivity is 0° and is therefore grid aligned.

Figure 3 shows how the iteration count of these preconditioners changes with anisotropic ratio. No preconditioning typically attracts the highest number of iterations, as would be expected, with the diagonal preconditioner reducing this noticeably, but only at higher anisotropic ratios. The Saena preconditioner has a substantially lower iteration count across the entire range of angles considered.

To understand the real-world impact of these preconditioners, Figure 4 shows the overall time taken to complete the solve. At low anisotropic ratios, no preconditioning actually provides the fastest solve time. This is likely because of the already low condition number of the matrix and grid alignment of the diffusivity tensor; diagonal preconditioning and the Saena preconditioner do not provide a sufficient reduction in condition number to offset their additional cost in this regime. At higher anisotropic ratios when the matrix becomes more ill-conditioned, both preconditioners are effective at reducing both iteration count and overall solve time. Although not shown, with $k_{\perp}$ set to zero the effect of changing $k_{\parallel}$ on the iteration count of the solve was found to be minimal.

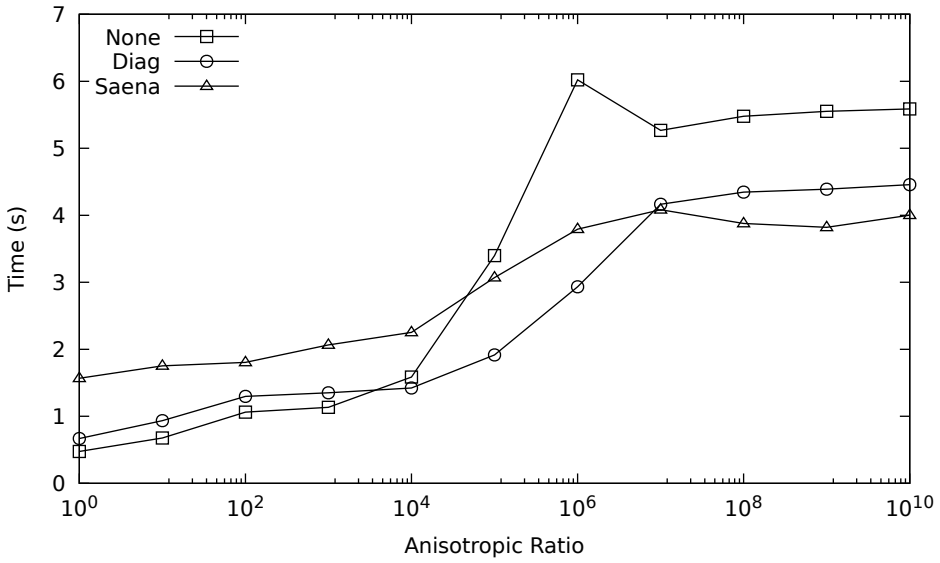

Figure 4: Full-matrix solve time for different preconditioners as a function of anisotropic ratio.

We next investigate the impact of diffusivity angle on the performance of the preconditioners. A non-zero diffusivity angle leads to a non-grid aligned field and therefore also introduces a change in the conditioning of the matrix system.

Figure 5 shows the change in the iteration count as diffusivity angle is varied for the two preconditioners and the baseline case of no preconditioning. There is a clear and substantial reduction in iteration count for small deviations from the grid-aligned case, across both preconditioners. The iteration count then saturates above approximately 10°. The diagonal preconditioner reduces the iteration count by approximately one-third for low angles, although provides negligible benefit for high angles. The Saena preconditioner reduces the iteration count substantially, but its benefit is lost by its substantially higher cost, as evident in Figure 6, where in fact using no preconditioning out-performs it for larger angles.

4.3 Static-condensed matrix system

We next investigate the performance of preconditioners in the context of using a statically condensed matrix system. Due to the requirement to substructure the matrix, the matrix-free kernels were not used for these tests.

Figure 7 shows the iteration count of the conjugate gradient solver when using the different preconditioners, for different angles of diffusivity. It is evident that the iteration count is particularly high for the grid-aligned diffusivity, but quickly reduces for non-aligned diffusivity, as for the full-matrix case. Additionally, as with the full matrix case, the diagonal preconditioner performs well for low-angles, but provides little advantage at high angles of diffusivity. For the statically condensed case, we also consider the full linear space preconditioner. For angles $\geq 1°$, this provides comparable or better iteration counts than the diagonal

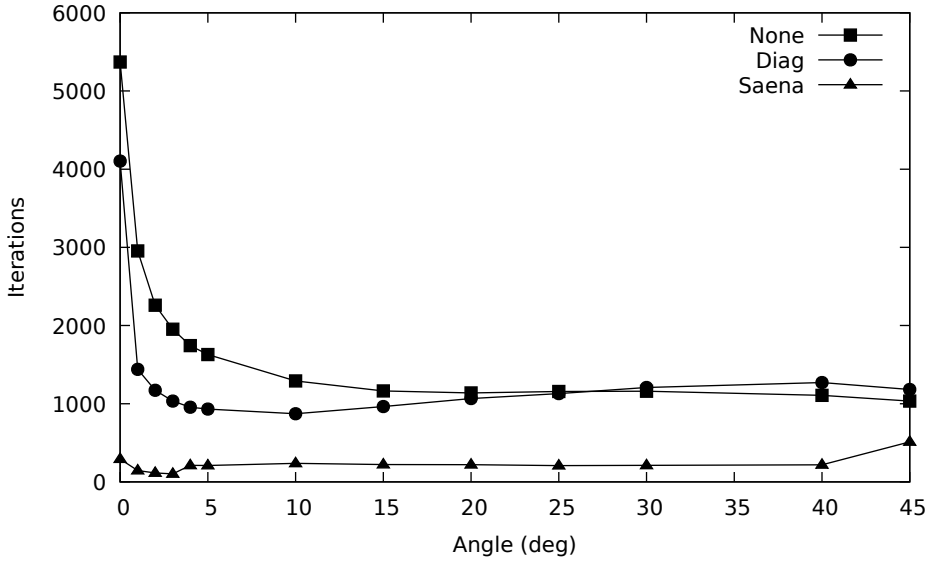

Figure 5: Iteration count of the conjugate gradient solver when using no preconditioner, the diagonal preconditioner or the experimental Saena preconditioner.

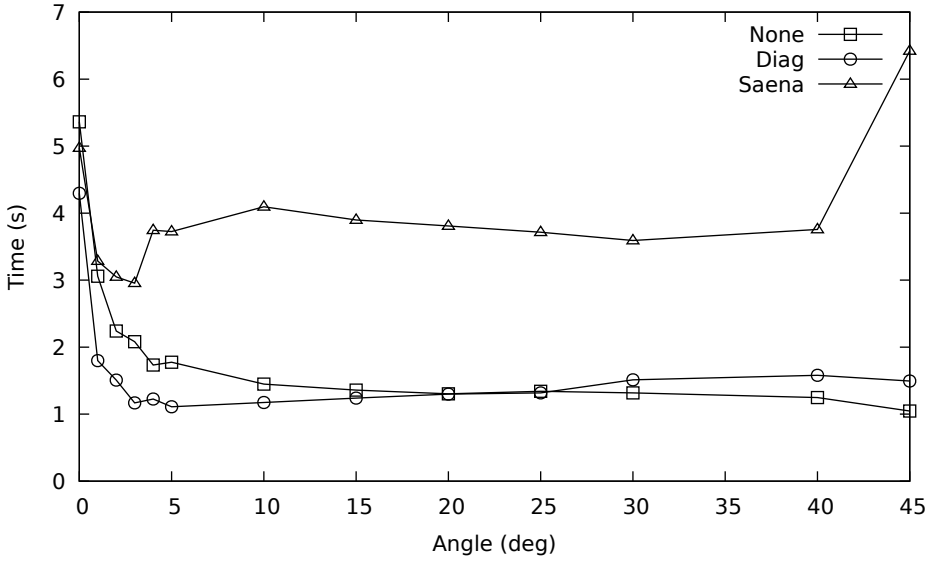

Figure 6: Full-matrix solve time for different angles of diffusivity, when using no preconditioner, a diagonal preconditioner or the experimental Saena preconditioner.

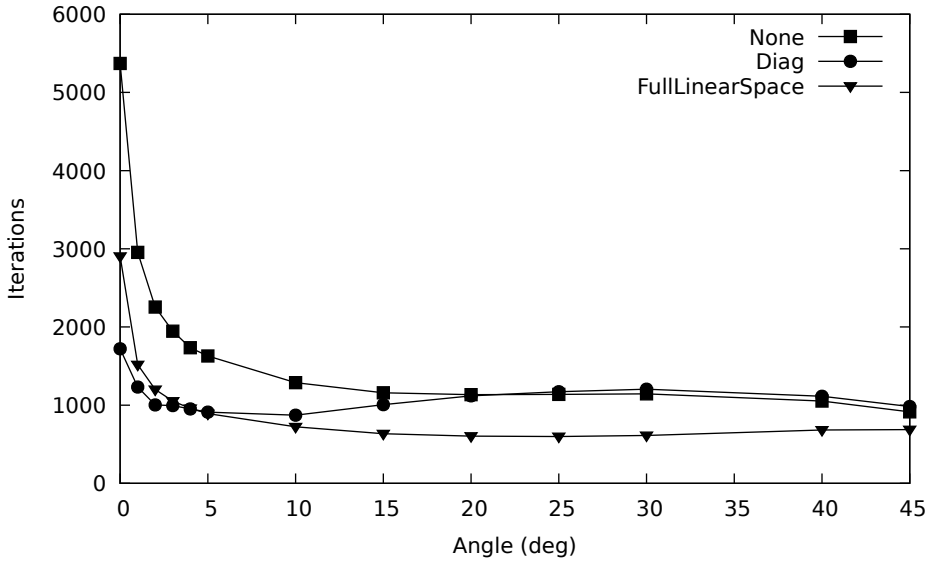

Figure 7: Statically condensed matrix conjugate gradient iteration count for different angles of diffusivity when using no preconditioner, the diagonal preconditioner or the linear space preconditioner.

preconditioner.

The solve times for these cases are compared in Figure 8. Notably, these times are lower than those in Figure 6; this is primarily due to the reduced system size leading to a more efficient solution procedure, even without the availability of the matrix-free kernels. The diagonal preconditioner provides a performance gain for low angles of diffusivity, but this quickly becomes of negligible benefit at higher angles. While the full linear space preconditioner provided reduced iteration counts, it in fact is slower than not using a preconditioner at all.

4.4 Limitations

The test case considered is that of a steady heat transport problem. Performance measurements include the complete set-up and solution of the linear system. These results are therefore not representative of time-integrating an unsteady heat transport problem. This is particularly evident for the full-linear space preconditioner, due to the high cost of setting up the preconditioner. Using it requires the direct parallel Cholesky factorisation and inversion of the resulting preconditioner matrix. In the case of this problem, this dominates the cost. For time-dependent problems where the preconditioner would be applied repeatedly at each time-step, this set-up cost would only occur once and so the preconditioner would provide a greater advantage.

We also note that timings were found to be quite variable for the configuration tested. Repeated runs of the same case might give variability of up to 10% in the measured runtime. This could due to the allocation of resources by the operating system, I/O contention and the movement of processes between NUMA regions. While the minimum runtime from multiple

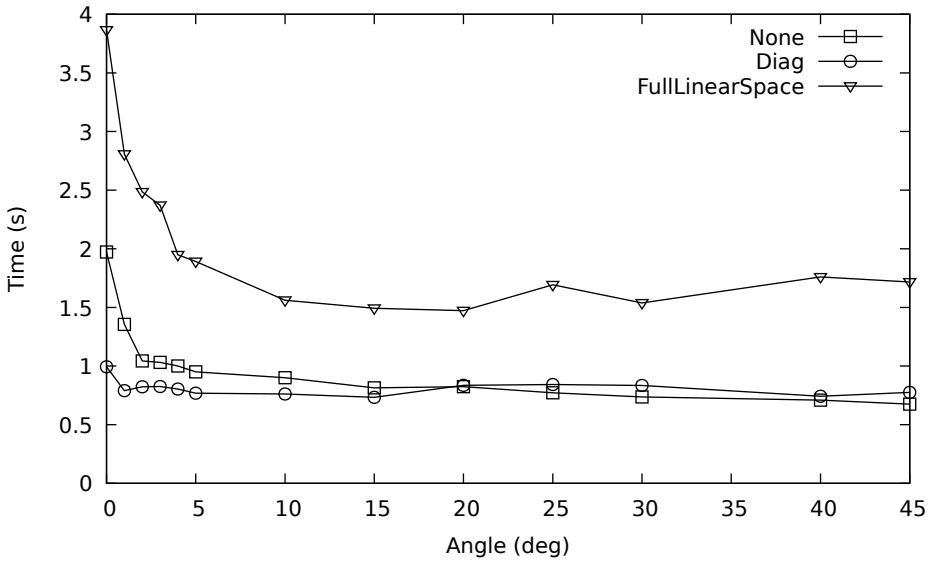

Figure 8: Statically condensed matrix solve time for different angles of diffusivity, when using no preconditioner, the diagonal preconditioner or the linear space preconditioner.

runs was recorded, this might explain some of the irregularities evident in the timing plots.

Finally, the performance of the Saena preconditioner will improve significantly with continued development. At present, the use of matrix-free operations was not possible with the current version of the library, but is expected to be available in a future version.

5 Conclusion

This report summarises the relevant available preconditioners in Nektar++ and their suitability to accelerate the conjugate gradient solver when solving the steady anisotropic heat transport formulation. Since it is anticipated that field-aligned grids will likely be used, the diffusivity angle is likely to be low. At current stage, it was found that the diagonal preconditioner is the most effective preconditioner currently available to improve the convergence rate of the iterative solver for the steady problem. However, iteration counts are still high and further work is needed to explore the use of other preconditioners for this problem. Unsteady problems, where a linear system is repeatedly solved will show different performance characteristics, particularly for the linear space and p-multigrid preconditioners, since the expensive set-up stage will only be required once, and further experiments are required to assess this case.

References

[1] Chris D Cantwell, David Moxey, Andrew Comerford, Alessandro Bolis, Gabriele Rocco, Gianmarco Mengaldo, Daniele De Grazia, Sergey Yakovlev, J-E Lombard, Dirk

Ekelschot, et al. Nektar++: An open-source spectral/hp element framework. *Computer physics communications*, 192:205–219, 2015.

[2] JP Hans Goedbloed, JP Goedbloed, and Stefaan Poedts. *Principles of magnetohydrodynamics: with applications to laboratory and astrophysical plasmas*. Cambridge university press, 2004.

[3] Bin Liu, Chris Cantwell, and Spencer Sherwin. Matrix-free kernel for the anisotropic laplacian operator for x86 architectures. Technical report, Imperial College London / UKAEA, 2021.

[4] Nicolas Offermans, Adam Peplinski, Oana Marin, Elia Merzari, and Philipp Schlatter. Performance of preconditioners for large-scale simulations using nek5000. In *Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2018*, pages 263–272. Springer, Cham, 2020.

[5] George Em Karniadakis, George Karniadakis, and Spencer Sherwin. *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press on Demand, 2005.