Report 2047356-TN-12 (D1.3): Elliptic solvers within *Nektar++*

David Moxey, King's College London
Ben Dudson, Peter Hill, Ed Higgins, David Dickinson, & Steven Wright, University of York

March 30, 2022

Contents

1 Executive summary

This report focuses on the performance of the elliptic solver within *Nektar++* and its scaling in the limit of large processor counts. Building on the report 2047356-TN-02-2, which introduces a number of sample test cases for the elliptic solver, we test both the correctness and the functionality of the elliptic solver. This solver has been used within the anisotropic diffusion solver

provided under project 2048465, and whose proxyapp code can be found in the NEPTUNE GitHub repository. We discuss the formulation of an elliptic solver within a high-order finite element setting, the mapping of these formulation onto performant software and the preconditioning strategies that may be employed to solve such systems. Implementation can be found in:

- The solution of electrostatic potential for the Hasegawa-Wakatani proxyapp: `https://github.com/ExCALIBUR-NEPTUNE/nektar-driftwave`
- The solution of the Laplacian term as part of the anisotropic heat proxyapp: `https://github.com/ExCALIBUR-NEPTUNE/nektar-diffusion`
- The main *Nektar++* code base: `https://gitlab.nektar.info/nektar/nektar`

2 Introduction

In this report, we briefly summarise the discretisation, performance and implementation of elliptic solvers within the *Nektar++* framework. These examples will mostly focus on 'slab'-type evaluations suggested in report 2047356-TN-02-2 as evaluation cases for elliptic problems, and/or problems found in the computational fluid dynamics that *Nektar++* is typically used within. However they may equally well be generalised to other problems, such as the solution for electrostatic potential within plasma-based problems. As such, we will consider elliptic problems of the type

$$\nabla \cdot (\mathcal{D} \nabla u) = f \tag{1}$$

where $\mathcal{D}$ is a $d \times d$ tensor and d is the spatial dimension.

In the setting of a high-order finite element discretisation, there are a number of problem-specific challenges to overcome when considering a high-performance computing implementation of an elliptic solver:

- In parallel execution the sparse matrix representing the discretised operator is not usually explicitly constructed. This is because at higher-orders, a more computationally-efficient implementation strategy is typically to evaluate the operator on a per-element basis, and combine this with an assembly operation to mimic the *action* of the global operator on a given mesh. Indeed, the evaluation of the operator can be done either by constructing local elemental matrices explicitly, or evaluation of their action via local matrix-free operations (which is the focus of call 2048465).
- Because of this, the preconditioning of such problems becomes challenging. Although it is well-known that multigrid-type preconditioning works very effectively on elliptic problems when coupled with linear C^0 discretisations, their use in high-order simulations is not very effective.

To this end, in this report we outline the approaches being considered within *Nektar++* to address these challenges: in particular, the methods that are already currently available, but also those presently in development. The report is therefore structured as follows. In section 3, we consider the formulation of such an elliptic problem in a finite element setting. Section 4 briefly discusses the operator evaluation techniques that can be used in this setting. Then in section 5 we discuss some of the preconditioning options that may be used in this environment, followed by examination of test cases in section 6. Section 7 outlines current work being undertaken on the development of multigrid preconditioning strategies for *Nektar++*. Finally, section 8 presents conclusions and other development directions.

3 Formulation

This section gives a very brief overview of the formulation of an elliptic equation in the spectral/*hp* element method. Further details can be found in e.g. Karniadakis & Sherwin [1]. Consider the elliptic equation

$$\nabla \cdot (\mathcal{D}\nabla u) + \lambda u = -f \tag{2}$$

where $\mathcal{D}$ is a $d \times d$ tensor on a domain $\Omega \subset \mathbb{R}^d$. The starting point for any finite element solution is to express the equation in weak form by multiplying by a test function v from a suitable trial space $\mathcal{V}$, considering functions u in a test space $\mathcal{U}$, integrating the resulting equality and applying integration by parts to yield

$$(\mathcal{D}\nabla u, \nabla v)_\Omega - \langle \mathcal{D}\nabla u \cdot \vec{n}, v\rangle_{\partial\Omega} + \lambda(u, v)_\Omega = (f, v) \tag{3}$$

where the inner products $(\cdot,\cdot)$ and $\langle\cdot,\cdot\rangle$ are defined as

$$(u, v) = \int_\Omega uv \,\mathrm{d}x, \quad \langle u, v\rangle = \int_{\partial\Omega} uv \,\mathrm{d}s \tag{4}$$

and $\vec{n}$ is the outwards facing normal from the boundary $\partial\Omega$. In the Galerkin approach, we select $\mathcal{U} = \mathcal{V}$, and then select a finite-dimensional subspace of $\mathcal{U}$ given a mesh of N nonoverlapping elements Ω^e such that $\Omega = \bigcup_e^N \Omega^e$. In a continuous C^0 high-order setting this space is the piecewise-polynomial space

$$\mathcal{D}_k(\Omega) = \{v_h \in C^0(\Omega) \mid v_h|_{\Omega^e} \in \mathbb{P}_k(\Omega^e)\} \tag{5}$$

where $\mathbb{P}_k$ denotes a space of polynomial order less than or equal to k.

The standard spectral/*hp* approach to discretise (2) starts with an expansion of u and v in terms of global basis functions $\Phi_n \in \mathcal{D}_k$, which are consequently represented as elemental basis functions ϕ_n^e:

$$u^\delta(\mathbf{x}) = \sum_{n=0}^{N_{\mathrm{dof}}-1} \hat{u}_n \Phi_n(\mathbf{x}) = \sum_{e=1}^{N}\sum_{n=1}^{k} \hat{u}_n^e \phi_n^e(\mathbf{x}) \tag{6}$$

where $\phi_n^e(\mathbf{x})$ is the n-th local expansion mode within the element Ω^e and $\hat{u}_n^e$ is the n-th local expansion coefficient within the element Ω^e. Approximating u and v in this manner (and assuming homogeneous Neumann boundary conditions for simplicity), we adopt a Galerkin discretisation of equation (2) where we find an approximate solution $u^\delta \in \mathcal{D}_k(\Omega)$ such that

$$\int_\Omega (\mathcal{D}\nabla u^\delta \nabla v^\delta + \lambda u^\delta v^\delta)\,\mathrm{d}\mathbf{x} = \int_\Omega v^\delta f d\mathbf{x} \quad \forall v^\delta \in D_k(\Omega). \tag{7}$$

This can be formulated in matrix terms as

$$\mathbf{H}\hat{\mathbf{u}} = \mathbf{f} \tag{8}$$

where $\mathbf{H} = \mathbf{L} + \lambda\mathbf{M}$ represents the Helmholtz matrix, $\mathbf{L}$ is the Laplacian matrix, λ is a positive constant and $\mathbf{M}$ the mass matrix. $\hat{\mathbf{u}}$ are the unknown global coefficients and $\mathbf{f}$ the inner product of the expansion basis with the forcing function. The solution of the discrete elliptic problem is therefore equivalent to the solution of the matrix problem.

4 Implementation strategies & operator evaluation

In this section we briefly outline the implementation strategies that may be employed in the solution of the elliptic matrix equation (8), with an emphasis on parallel execution. In this setting, we expect that the mesh is partitioned as a preprocessing step into R subdomains (weighted appropriately by the number of degrees of freedom, so as to load-balance the problem). Furthermore, the solution of the matrix equation is given through an appropriate Krylov solver such as the conjugate gradient method. The dominant cost in this process is therefore the computation of the **action** of $\mathbf{H}$, at the n-th iteration of such methods, i.e. $\mathbf{H}\hat{\mathbf{u}}_n$. The subsections below outline the general approaches that may be taken in a high-order setting.

4.1 Global matrix approach

Mathematically the easiest approach is to assemble the global system $\mathbf{H}$. To do this, we first ignore the mesh as a whole and construct instead a local elemental matrix $\mathbf{H}^e$ by considering the left hand integral of equation (7) on a single element Ω^e. Then, with knowledge of the mesh topology and the C^0 connectivity between neighbouring elements (and therefore their basis functions), we form an assembly matrix $\mathcal{A}$ which maps the vector of global coefficients $\hat{\mathbf{u}}$ to a vector of local coefficients $\hat{\mathbf{u}}_l$. In this manner, we may express the global matrix as

$$\mathbf{H} = \mathcal{A}^\top \left(\bigoplus_{e=1}^{N} \mathbf{H}^e \right) \mathcal{A}. \tag{9}$$

In a parallel environment, the sub-matrix of $\mathbf{H}$ is constructed for each rank, and then the action of the gather-scatter operations applied by $\mathcal{A}^\top$ and $\mathcal{A}$ are implemented via the external third-party library `gslib` [2][1].

From a practical perspective, the construction of $\mathbf{H}$ as a global sparse matrix is typically not recommended at any polynomial order $k > 1$, since the matrix itself admits a rich structure based on the locally-dense high-order element matrices which is disregarded in this setting. Attaining sufficient arithmetic intensity in operator evaluation is also highly challenging due to the large amount of indirection incurred via the use of standard sparse matrix data structures (e.g. compressed row/column format).

4.2 Local matrices

Instead, we may elect to consider that the operation is separated into three steps to obtain $\mathbf{v} = \mathbf{H}\mathbf{u}$:

1. obtain the vector of local coefficients from the scatter operation $\hat{\mathbf{u}}_l = \mathcal{A}\hat{\mathbf{u}}$;
2. compute the matrix-vector product

$$\hat{\mathbf{v}}_l = \left(\bigoplus_{e=1}^{N} \mathbf{H}^e \right) \mathbf{u}_l \tag{10}$$

3. gather the local coefficients as $\hat{\mathbf{v}} = \mathcal{A}^\top \hat{\mathbf{v}}_l$.

Mathematically this process is identicial in nature to eq. (9). However step 2 comes with the distinct advantage that the matrix-vector product is block diagonal: therefore we may precompute element matrices $\mathbf{H}^e$ as small dense matrices, which consequently greatly increases

[1] `https://github.com/Nek5000/gslib`

the arithmetic intensity of the method (even owing for the gather/scatter operations), despite the complexity remaining unchanged.

This approach has been the standard approach employed within many high-order codes. However, for large k in three-dimensions, even the local matrices start to increase greatly in size. We therefore should consider strategies for reducing the dimensionality or complexity of the problem in some manner. This is the consideration in the following two sections.

4.3 Static condensation

The spectral/*hp* expansion basis is obtained by considering interior modes, which have support only in the interior of the element, separately from boundary modes which are non-zero on the boundary of the element. We align the boundary modes across the interface of the elements to obtain a continuous global solution. For a three-dimensional problem, the boundary modes can be further decomposed into vertex, edge and face modes, defined as follows:

- vertex modes have support on a single vertex and the three adjacent edges and faces as well as the interior of the element;
- edge modes have support on a single edge and two adjacent faces as well as the interior of the element;
- face modes have support on a single face and the interior of the element.

It is worth emphasising that only certain choices of basis functions support such decompositions. In the modified basis of Karniadakis & Sherwin, such a structure is readily admitted for all elemental shape types. The classical Lagrange interpolant basis also admits the same structure for quadrilateral and hexahedral elements. However, other orthogonal bases such as the Legendre basis for triangular elements do not support this structure (which also complicates the imposition of C^0 conditions between elements).

When the discretisation is continuous, this strong coupling between vertices, edges and faces leads to a matrix of high condition number κ. Our aim is to reduce this condition number by applying specialised preconditioners. However, we can also leverage this structure to reduce the dimension of the parallel problem solved across the entire grid via **static condensation**. Utilising the above mentioned decomposition, we can write the matrix equation as

$$\begin{bmatrix} \mathbf{H}_{bb} & \mathbf{H}_{bi} \\ \mathbf{H}_{ib} & \mathbf{H}_{ii} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{u}}_b \\ \hat{\mathbf{u}}_i \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{f}}_b \\ \hat{\mathbf{f}}_i \end{bmatrix} \tag{11}$$

where the subscripts b and i denote the boundary and interior degrees of freedom respectively. This system then can be statically condensed allowing us to solve for the boundary and interior degrees of freedom in a decoupled manner. The statically condensed matrix is given by

$$\begin{bmatrix} \mathbf{H}_{bb} - \mathbf{H}_{bi}\mathbf{H}_{ii}^{-1}\mathbf{H}_{ib} & 0 \\ \mathbf{H}_{ib} & \mathbf{H}_{ii} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{u}}_b \\ \hat{\mathbf{u}}_i \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{f}}_b - \mathbf{H}_{bi}\mathbf{H}_{ii}^{-1}\hat{\mathbf{f}}_i \\ \hat{\mathbf{f}}_i \end{bmatrix}. \tag{12}$$

This is highly advantageous since by definition of our interior expansion this vanishes on the boundary, and so $\mathbf{H}_{ii}$ is block diagonal and thus can be trivially inverted as a preprocessing step. The above sub-structuring has reduced our problem to solving the boundary problem

$$\mathbf{S}_1\hat{\mathbf{u}} = \hat{\mathbf{f}}_1 \tag{13}$$

where $\mathbf{S_1} = \mathbf{H}_{bb} - \mathbf{H}_{bi}\mathbf{H}_{ii}^{-1}\mathbf{H}_{ib}$ and $\hat{\mathbf{f}}_1 = \hat{\mathbf{f}}_b - \mathbf{H}_{bi}\mathbf{H}_{ii}^{-1}\hat{\mathbf{f}}_i$. Although this new system typically has better convergence properties (i.e lower κ), the system is still ill-conditioned, leading to

a convergence rate of iterative-type solvers such as the conjugate gradient method that is prohibitively slow. For this reason we need to precondition $\mathbf{S}_1$. To do this we solve an equivalent system of the form:

$$\mathbf{M}^{-1}\left(\mathbf{S}_1\hat{\mathbf{u}} - \hat{\mathbf{f}}_1\right) = 0 \tag{14}$$

where the preconditioning matrix $\mathbf{M}$ is such that $\kappa\left(\mathbf{M}^{-1}\mathbf{S}_1\right)$ is less than $\kappa\left(\mathbf{S}_1\right)$ and speeds up the convergence rate. Within the conjugate gradient routine the same preconditioner $\mathbf{M}$ is applied to the residual vector $\hat{\mathbf{r}}_{k+1}$ of the conjugate gradient routine every iteration

$$\hat{\mathbf{z}}_{k+1} = \mathbf{M}^{-1}\hat{\mathbf{r}}_{k+1}. \tag{15}$$

We discuss specific preconditioning strategies in section 5.

4.4 Matrix-free operations

A final option is to omit the construction of local matrices **entirely** and instead evaluate their action via their summation forms instead. This is called the **matrix-free** approach and helps to further increase arithmetic intensity since matrix construction and storage is no longer required [3].

To demonstrate this approach, consider a straightforward quadrilateral element and further assume that $\mathcal{D}$ is the identity for simplicity. On this element type we use an expansion basis of the form $\{\phi_{pq}(\xi_1, \xi_2); p = 1, \ldots, P; q = 1, \ldots, P\}$. The approximated function is then given by the summation

$$u(\xi) = \sum_{p=1}^{P}\sum_{q=1}^{P}\hat{u}_{pq}\phi_{pq}(\xi_1, \xi_2) \tag{16}$$

and the two-dimensional discrete Helmholtz operator now reads

$$\begin{aligned}\boldsymbol{H}_\delta^e[m][n] = \lambda\sum_i\sum_j w_i w_j \phi_{pq}(\xi_{1i}, \xi_{2j})\phi_{rs}(\xi_{1i}, \xi_{2j})|\boldsymbol{J}_{ij}^e| \\ + \sum_i\sum_j w_i w_j (\boldsymbol{J}_{ij}^e)^{-T}\nabla\phi_{pq}(\xi_{1i}, \xi_{2j})\cdot(\boldsymbol{J}_{ij}^e)^{-1}\nabla\phi_{rs}(\xi_{1i}, \xi_{2j})|\boldsymbol{J}_{ij}^e|.\end{aligned} \tag{17}$$

where the indices i and j again span over the quadrature points, and the indices m and n are obtained as linear combinations of the indices i, j, p, q, r, and s that depend on the type of element used on the approximation. For specific details of this calculation the interested reader could consult Section 4.1.5 of the textbook [1].

The discrete Helmholtz operator can be further split into the mass operator $\boldsymbol{M}_\delta^e$ and the Laplacian operator $\boldsymbol{L}_\delta^e$ yielding

$$\boldsymbol{H}_\delta^e = \lambda\boldsymbol{M}_\delta^e + \boldsymbol{L}_\delta^e. \tag{18}$$

The mass operator can be rewritten in matrix form as

$$\boldsymbol{M}_\delta^e = \boldsymbol{B}^T\boldsymbol{W}\boldsymbol{B}, \tag{19}$$

where the dense basis matrix $\boldsymbol{B}$ contains terms of the form $\phi_{pq}(\xi_{1i}, \xi_{2j})$, and the components of the diagonal matrix $\boldsymbol{W}$ are the weights $w_i w_j|\boldsymbol{J}_{ij}^e|$.

Noting that the derivative of the basis functions with respect to the local coordinates for two-dimensional expansions is

$$\nabla\phi(\xi_{1i}, \xi_{2j}) = \begin{pmatrix}\dfrac{\partial\phi}{\partial\xi_{1i}} \\ \dfrac{\partial\phi}{\partial\xi_{2j}}\end{pmatrix}, \tag{20}$$

the inverse of the Jacobian matrix $(\boldsymbol{J}^e)^{-1}$ is

$$(\boldsymbol{J}^e)^{-1} = \begin{pmatrix} \dfrac{\partial \xi_1}{\partial x_1} & \dfrac{\partial \xi_2}{\partial x_1} \\ \dfrac{\partial \xi_1}{\partial x_2} & \dfrac{\partial \xi_2}{\partial x_2} \end{pmatrix}, \tag{21}$$

and the Laplacian operator can be expanded as

$$\begin{aligned} \boldsymbol{L}_\delta^e = \sum_i \sum_j w_i w_j \\ \Bigg[\left(\frac{\partial \xi_1}{\partial x_1} \frac{\partial \phi_{pq}}{\partial \xi_1} + \frac{\partial \xi_2}{\partial x_1} \frac{\partial \phi_{pq}}{\partial \xi_2} \right) \left(\frac{\partial \xi_1}{\partial x_1} \frac{\partial \phi_{rs}}{\partial \xi_1} + \frac{\partial \xi_2}{\partial x_1} \frac{\partial \phi_{rs}}{\partial \xi_2} \right) \\ + \left(\frac{\partial \xi_1}{\partial x_2} \frac{\partial \phi_{pq}}{\partial \xi_1} + \frac{\partial \xi_2}{\partial x_2} \frac{\partial \phi_{pq}}{\partial \xi_2} \right) \left(\frac{\partial \xi_1}{\partial x_2} \frac{\partial \phi_{rs}}{\partial \xi_1} + \frac{\partial \xi_2}{\partial x_2} \frac{\partial \phi_{rs}}{\partial \xi_2} \right) \Bigg]_{ij} |\boldsymbol{J}_{ij}^e| \,. \end{aligned} \tag{22}$$

We can now rewrite the Laplacian operator in matrix form as

$$\boldsymbol{L}_\delta^e = \boldsymbol{U}^T \boldsymbol{W} \boldsymbol{U} + \boldsymbol{V}^T \boldsymbol{W} \boldsymbol{V}, \tag{23}$$

with $\boldsymbol{U}$ and $\boldsymbol{V}$ being

$$\boldsymbol{U} = \left(\boldsymbol{\Lambda} \left(\frac{\partial \xi_1}{\partial x_1} \right) \boldsymbol{D}_{\xi_1} + \boldsymbol{\Lambda} \left(\frac{\partial \xi_2}{\partial x_1} \right) \boldsymbol{D}_{\xi_2} \right) \boldsymbol{B} \tag{24}$$

$$\boldsymbol{V} = \left(\boldsymbol{\Lambda} \left(\frac{\partial \xi_1}{\partial x_2} \right) \boldsymbol{D}_{\xi_1} + \boldsymbol{\Lambda} \left(\frac{\partial \xi_2}{\partial x_2} \right) \boldsymbol{D}_{\xi_2} \right) \boldsymbol{B} \,. \tag{25}$$

Here $\boldsymbol{\Lambda}(f)$ is is a diagonal matrix containing terms of the form $f(\xi_{1i}, \xi_{2j})$, and $\boldsymbol{D}_{\xi_i}$ is a block-diagonal differentiation matrix with entries $\left.\frac{\mathrm{d}h_p(\xi_1)}{\mathrm{d}\xi_1}\right|_{\xi_{1i}}$ where $h_p(\xi)$ is the one-dimensional Lagrange polynomial.

In this form, we have written the Helmholtz matrix mostly in terms of element-independent matrices: the $\mathbf{D}_\xi$ and quadrature locations and weights are entirely defined on the standard element. The only elemental-specific information required is therefore the geometric factors and their associated Jacobian. When combined further with sum-factorisation to reduce the cost of the summation from a complexity of $\mathcal{O}(P^{2d})$ to $\mathcal{O}(P^{d+1})$, matrix-free evaluation gives performant and vectorisation-ameanable approach to operator evaluation (as shown in [3]).

4.5 Solvers in *Nektar++*

Nektar++ exposes some of these approaches within the `GlobalSysSoln` classes, which implement these strategies:

- `Full` solves operate on the full operator;
- `StaticCond` solves operate on the statically-condensed solver;
- `MultiLevelStaticCond` solves apply the static condensation approach repeatedly to further reduce the system size.

The precise evaluation strategy depends on the actual solver being used to solve the resulting systems:

- `Direct` solves use LAPACK to solve the matrix system, and can be used in serial only;
- `Iterative` solves use a conjugate gradient (symmetric) or GMRES method (non-symmetric) to solve the system, either in serial or parallel;
- `Xxt` uses the $XX^\top$ direct parallel solver [4] to solve the system;
- `PETSc` uses the PETSc linear algebra package for the system.

The combination of these strategies results in a concrete implementation. For example:

- `DirectFull` solves the full matrix system with LAPACK.
- `IterativeFull` does the same, but using CG/GMRES. In this case the matrix evaluation would be done via a matrix-free approach.
- `IterativeStaticCond` solves the statically-condensed system using an iterative approach. This is the default method in parallel.
- `DirectMultiLevelStaticCond` solves the multi-level statically condensed system using a direct solver. This is the default method in serial execution.

5 Preconditioning

Within the *Nektar++* framework a number of preconditioners are available to speed up the convergence rate of the conjugate gradient routine. 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).

Name	**Dimensions**	**Discretisations**
`Null`	All	All
`Diagonal`	All	All
`FullLinearSpace`	2/3D	CG
`LowEnergyBlock`	3D	CG (static cond.)
`Block`	2/3D	All
`FullLinearSpaceWithDiagonal`	All	CG
`FullLinearSpaceWithLowEnergyBlock`	2/3D	CG (static cond.)
`FullLinearSpaceWithBlock`	2/3D	CG

The default is the `Diagonal` preconditioner. The above preconditioners are specified through the `Preconditioner` option of the `SOLVERINFO` section in the session file. For example, to enable `FullLinearSpace` one can use:

```
<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). The following sections specify the details for each method.

5.1 Diagonal

Diagonal (or Jacobi) preconditioning is amongst the simplest preconditioning strategies. In this scheme one takes the global matrix $\mathbf{H} = (h_{ij})$ and computes the diagonal terms h_{ii}. The preconditioner is then formed as a diagonal matrix $\mathbf{M}^{-1} = (h_{ii}^{-1})$.

5.2 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

$$\mathbf{M}^{-1} = (\mathbf{S}_1^{-1})_{vv}. \tag{26}$$

Since the mesh associated with higher order methods is relatively coarse compared with traditional 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 present is handled using the XX^T library [4]. XX^T is a parallel direct solver for problems of the form $\mathbf{A}\hat{\mathbf{x}} = \hat{\mathbf{b}}$ based around a sparse factorisation of the inverse of $\mathbf{A}$. To precondition utilising this methodology the linear subspace is gathered from the expansion and the preconditioned residual within the CG routine is determined by solving

$$(\mathbf{S}_1)_{vv}\hat{\mathbf{z}} = \hat{\mathbf{r}}. \tag{27}$$

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

5.3 Block

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

$$\mathbf{M}^{-1} = \begin{bmatrix} (\mathbf{S}_1^{-1})_{vv} & 0 & 0 \\ 0 & (\mathbf{S}_1^{-1})_{eb} & 0 \\ 0 & 0 & (\mathbf{S}_1^{-1})_{ef} \end{bmatrix} \tag{28}$$

where $\text{diag}[(\mathbf{S}_1)_{vv}]$ is the diagonal of the vertex modes, $(\mathbf{S}_1)_{eb}$ and $(\mathbf{S}_1)_{fb}$ 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.

5.4 Low energy preconditioner

Low energy basis preconditioning follows the methodology proposed by Sherwin & Casarin. In this method a new basis is numerically constructed from the original basis which allows the Schur complement matrix to be preconditioned using a block preconditioner. The method is outlined briefly in the following.

Elementally the local approximation $\mathbf{u}^\delta$ can be expressed as different expansions lying in the same discrete space V^δ

$$\mathbf{u}^\delta(\mathbf{x}) = \sum_i^{\dim(V^\delta)} \hat{u}_{1i}\phi_{1i}(x) = \sum_i^{\dim(V^\delta)} \hat{u}_{2i}\phi_{2j}(x). \tag{29}$$

Since both expansions lie in the same space it is possible to express one basis in terms of the other via a transformation, i.e.

$$\phi_2 = \mathbf{C}\phi_1 \implies \hat{\mathbf{u}}_1 = C^T\hat{\mathbf{u}}_2. \tag{30}$$

Applying this to the Helmholtz operator it is possible to show that

$$\mathbf{H}_2 = \mathbf{C}\mathbf{H}_1\mathbf{C}^T. \tag{31}$$

For sub-structured matrices ($\mathbf{S}$) the transformation matrix ($\mathbf{C}$) becomes

$$\mathbf{C} = \begin{bmatrix} \mathbf{R} & 0 \\ 0 & \mathbf{I} \end{bmatrix}. \tag{32}$$

Hence the transformation in terms of the Schur complement matrices is:

$$\mathbf{S}_2 = \mathbf{R}\mathbf{S}_1\mathbf{R}^T. \tag{33}$$

Typically the choice of expansion basis ϕ_1 can lead to a Helmholtz matrix that has undesirable properties i.e poor condition number. By choosing a suitable transformation matrix $\mathbf{C}$ it is possible to construct a new basis, numerically, that is amenable to block diagonal preconditioning, given by

$$\mathbf{S}_1 = \begin{bmatrix} \mathbf{S}_{vv} & \mathbf{S}_{ve} & \mathbf{S}_{vf} \\ \mathbf{S}_{ve}^T & \mathbf{S}_{ee} & \mathbf{S}_{ef} \\ \mathbf{S}_{vf}^T & \mathbf{S}_{ef}^T & \mathbf{S}_{ff} \end{bmatrix} = \begin{bmatrix} \mathbf{S}_{vv} & \mathbf{S}_{v,ef} \\ \mathbf{S}_{v,ef}^T & \mathbf{S}_{ef,ef} \end{bmatrix}. \tag{34}$$

Applying the transformation $\mathbf{S}_2 = \mathbf{R}\mathbf{S}_1\mathbf{R}^T$ leads to the following matrix

$$\mathbf{S}_2 = \begin{bmatrix} \mathbf{S}_{vv} + \mathbf{R}_v\mathbf{S}_{v,ef}^T + \mathbf{S}_{v,ef}\mathbf{R}_v^T + \mathbf{R}_v\mathbf{S}_{ef,ef}\mathbf{R}_v^T & [\mathbf{S}_{v,ef} + \mathbf{R}_v\mathbf{S}_{ef,ef}]\mathbf{A}^T \\ \mathbf{A}[\mathbf{S}_{v,ef}^T + \mathbf{S}_{ef,ef}\mathbf{R}_v^T] & \mathbf{A}\mathbf{S}_{ef,ef}\mathbf{A}^T \end{bmatrix} \tag{35}$$

where $\mathbf{A}\mathbf{S}_{ef,ef}\mathbf{A}^T$ is given by

$$\mathbf{A}\mathbf{S}_{ef,ef}\mathbf{A}^T = \begin{bmatrix} \mathbf{S}_{ee} + \mathbf{R}_{ef}\mathbf{S}_{ef}^T + \mathbf{S}_{ef}\mathbf{R}_{ef}^T + \mathbf{R}_{ef}\mathbf{S}_{ff}\mathbf{R}_{ef}^T & \mathbf{S}_{ef} + \mathbf{R}_{ef}\mathbf{S}_{ff} \\ \mathbf{S}_{ef}^T + \mathbf{S}_{ff}\mathbf{R}_{ef}^T & \mathbf{S}_{ff} \end{bmatrix}. \tag{36}$$

To orthogonalise the vertex-edge and vertex-face modes, it can be seen from the above that

$$\mathbf{R}_{ef}^T = -\mathbf{S}_{ff}^{-1}\mathbf{S}_{ef}^T \tag{37}$$

and for the edge-face modes

$$\mathbf{R}_v^T = -\mathbf{S}_{ef,ef}^{-1}\mathbf{S}_{v,ef}^T. \tag{38}$$

Here it is important to consider the form of the expansion basis since the presence of $\mathbf{S}_{ff}^{-1}$ will lead to a new basis which has support on all other faces; this is problematic when creating a C^0 continuous global basis. To circumvent this problem when forming the new basis, the decoupling is only performed between a specific edge and the two adjacent faces in a symmetric standard region. Since the decoupling is performed in a rotationally symmetric standard region the basis does not take into account the Jacobian mapping between the local element and global coordinates, hence the final expansion will not be completely orthogonal.

The low energy basis creates a Schur complement matrix that although it is not completely orthogonal can be spectrally approximated by its block diagonal contribution. The final form of the preconditioner is:

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathrm{diag}[(\mathbf{S}_2)_{vv}] & 0 & 0 \\ 0 & (\mathbf{S}_2)_{eb} & 0 \\ 0 & 0 & (\mathbf{S}_2)_{fb} \end{bmatrix}^{-1} \tag{39}$$

where $\mathrm{diag}[(\mathbf{S}_2)_{vv}]$ is the diagonal of the vertex modes, $(\mathbf{S}_2)_{eb}$ and $(\mathbf{S}_2)_{fb}$ are block diagonal matrices corresponding to coupling of an edge (or face) with itself i.e. ignoring the coupling to other edges and faces.

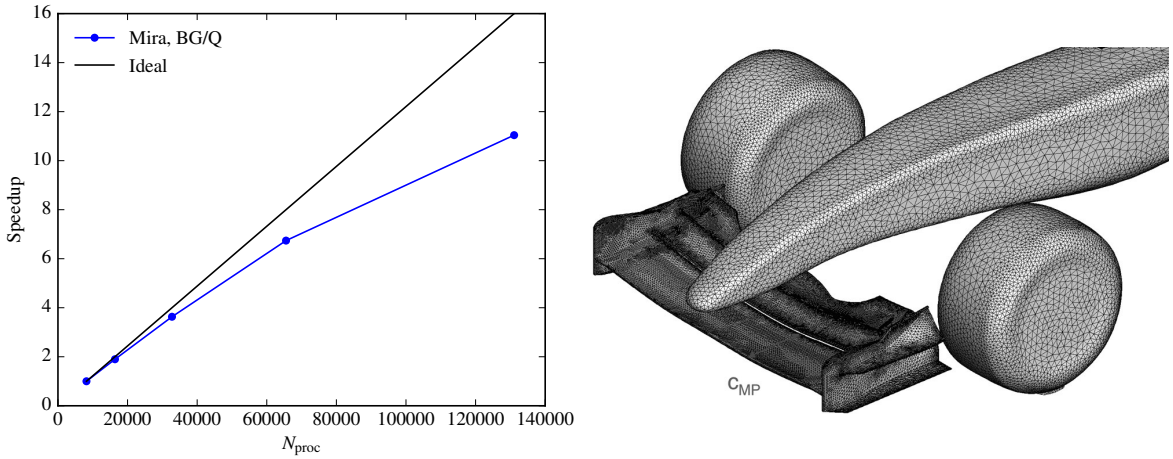

Figure 1: Strong scaling of the preconditioned conjugate gradient solver, run on a mesh of a Formula 1 front wing section (pictured right), consisting of 3.3M tetrahedral and prismatic elements. Speedup is calculated relative to runs on 8192 processors.

6 Performance

In this section we briefly outline performance of the elliptic solver from the perspective of parallel efficiency for the `IterativeStaticCond` solver. We note that other NEPTUNE deliverables (particularly those from project 2048465) focus on the outline performance properties of matrix-free operators. *Nektar++* has been used on a wide variety of HPC platforms. We report on two of the largest simulations to date:

- A diffusion problem over a complex geometry of a Formula 1 car on the Mira supercomputer, which shows general speedup of the elliptic solver under simple preconditioning strategies.
- A more realistic fluid dynamics simulation of an Elemental RP1 car on the supercomputer ARCHER2.

6.1 Large-scale parallel testing

As part of validation of the solver at large scales, simulations were performed on both the BG/Q at ANL and the Cray XK7 at ORNL for preliminary testing of Nektar++ on extremely large core counts. In Fig. 1, we present the results of strong scaling simulations up to 131K processors on the BG/Q, where we measure relative speedup when compared to a simulation on 8192 processors. The starting point for these tests is a complex topology of a Formula 1 front wing section, which is 6th order accurate in space. We then measure the mean execution time for a single iteration of the conjugate gradient solver, which represents the most processor- and communication-intensive part of the solver. At 131K processors, there are only 25 elements per process. This of course is not a representative example of a full complex fluid dynamics solver, since the setup is deliberately simple: a diffusion equation with simple Jacobi preconditioning. However it demonstrates that the core solve itself is readily scalable to high core counts.

6.2 Fluid dynamics simulations

Results are shown in Figure 2 for a strong scaling test using early access to the ARCHER2 facility, using between 2k and 50k cores. The test case under investigation is a fifth-order incompressible fluid dynamics simulation of a track car configuration at Reynolds number of

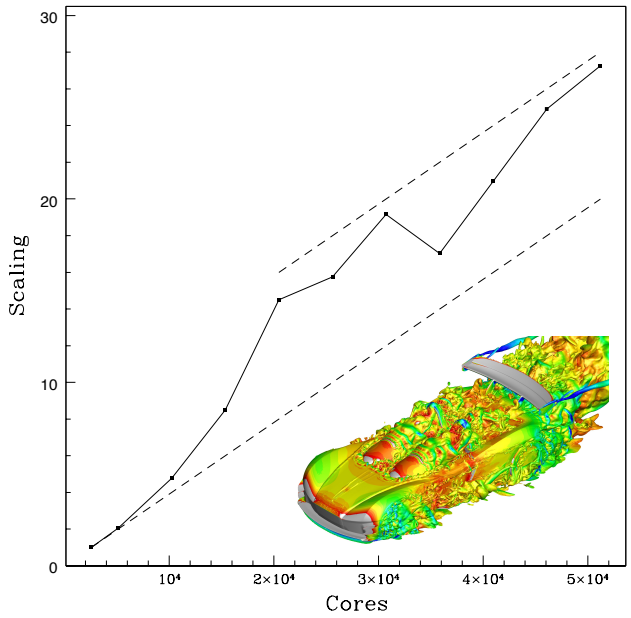

Figure 2: Strong scaling of the Elemental RP1 track car simulation on the ARCHER2 facility.

approximately 1 million, on a mesh of mixed tetrahedral/prismatic elements, so that at 50k cores we observe around 70 elements per computing core. This therefore represents a mesh of around double the density of the test case above. Preconditioning in this case uses the low-energy block preconditioner for all simulation variables.

At lower core counts, relatively high memory usage per node for this more realistic case means that superlinear scaling is observed. However beyond around 20,000 cores, memory per node drops sufficiently so that cache usage provides more effective performance, thereby bringing scaling into a more linear regime. We note that these results were obtained very shortly after early access was made available for the 4-cabinet system in November 2021, and so may differ from results available now. However it also highlights the general performance capability of the code for a more realistic problem.

7 Ongoing developments in preconditioning

In the test cases discussed above, one potential limitation to scalability is the preconditioning strategies being used, particularly when considering Poisson problems of the form

$$\nabla^2 p = -f \tag{40}$$

in solving for pressure p or, in the case of plasma-related applications, electrostatic potential ϕ. It is well-known that multigrid methods are preferred method for solving the resulting symmetric linear systems 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

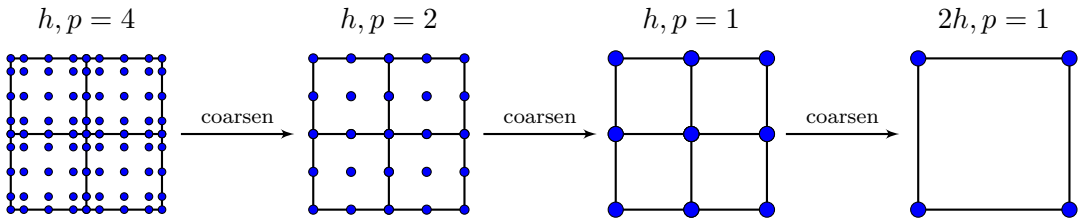

Figure 3: Illustration of the coarsening strategy of GIAMG for quadrilateral elements with nodal basis of polynomial order $p = 4$. First, We p-coarsen from $p = 4$ to $p = 2$, and then again p-coarsen from $p = 2$ to $p = 1$. The total number of elements—and therefore the mesh—does not change during p-coarsening. Once we reach linear elements, we perform h-coarsening to obtain a coarser mesh and smaller system.

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 5 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 $XX^\top$ 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[2], to circumvent this problem we are considering a *geometrically informed* algebraic multigrid approach (GIAMG), 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 3. 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 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.

To test this approach, we consider the sample pin-type breeder blanket configuration used for reactor cooling, shown in figure 4. We focus on an under-resolved DNS at $\mathrm{Re} = 10^4$ with a coarse mesh consisting of around 150000 tetrahedral and prismatic elements. The coarse nature of these simulations, together with large changes in velocity across a small region at the end of the pin, together with high aspect ratio elements at the walls, makes the systems particularly challenging to precondition at high order.

We plot the convergence results using GIAMG for p from 1 to 5 as shown in Figure 4. For this

[2] *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.

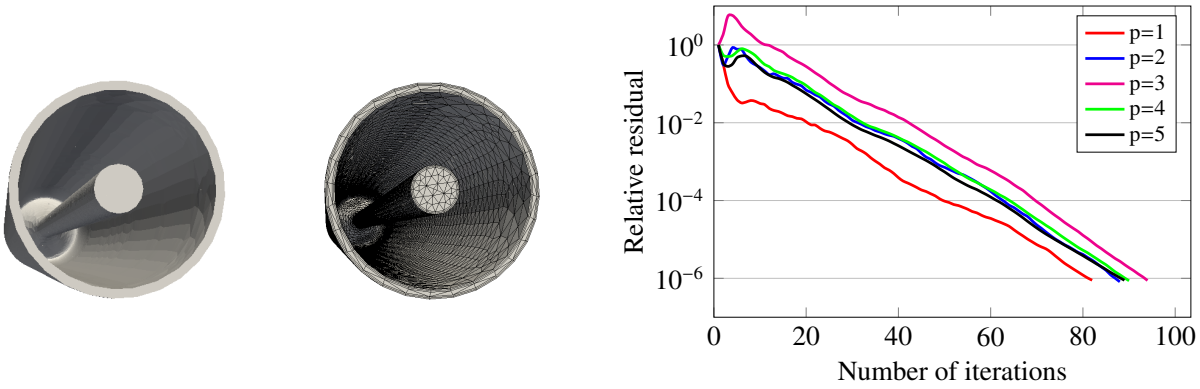

Figure 4: Geometry and convergence properties for the sample case.

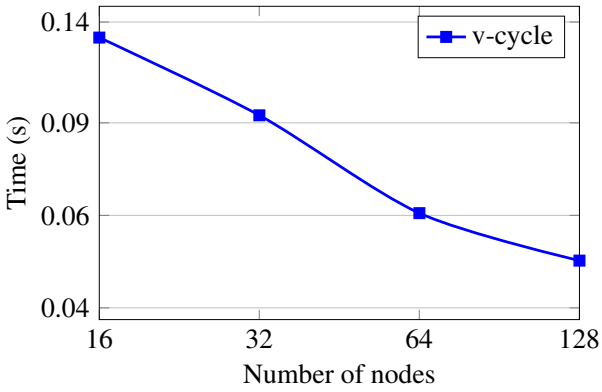

Figure 5: Strong scaling of the GIAMG approach.

more complex operator, we set the convergence criterion to be the relative residual less than 10^{-6}. In this case, we only reduce the basis order by 1 at each level during p-coarsening followed by 6 additional levels of coarsening using smoothed aggregation. We perform 2 Chebyshev iterations in pre- and postsmoothing. The sizes and numbers of nonzeros of the coarsest level matrices in these cases are slightly different, varying from 303×303 to 315×315, and 32338 to 34191, respectively, which are all reasonably small at the coarsest level for the superLU solver. We can see the number of iterations does not significantly increase as we increase p order. The convergence of $p = 3$ case looks slightly inferior to other cases. However, it still behaves within a reasonable range, and the convergence pattern is similar to that in other cases.

When compared against the standard Jacobi-type preconditioning which is extremely cheap and communication-free to evaluate, the GIAMG approach is clearly more expensive per conjugate gradient iteration. However, the GIAMG significantly reduces the number of iterations required to converge from roughly 3500 iterations for Jacobi-preconditioned solves to around 80-90 iterations. Scaling of the V-cycle from between 16 to 128 nodes (with 56 cores per node, so 7168 MPI ranks at the larger end) is shown in Figure 5, which demonstrates the potential for this method for larger-scale problems.

It should be noted that this approach is not yet widely available with *Nektar++* but we are working to integrate this over the coming months to improve elliptic solver performance in a wide range of problems. There are also challenges to be overcome in terms of 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.

8 Conclusions

This report has briefly outlined the approaches being undertaken within *Nektar++* and NEPTUNE for the solution of elliptic problems. The core performance of the solver itself demonstrates high performance for both on-node and parallel execution. Some of the challenges facing further development have been outlined, with a particular emphasis on the development of multigrid methods. The further development of these techniques, and particularly their use on different architectures such as GPUs, is also a key focus of the development team which we discuss in other NEPTUNE deliverables.

References

[1] George Karniadakis and Spencer Sherwin. *Spectral/Hp Element Methods for Computational Fluid Dynamics*. Oxford University Press, 2013.

[2] Jing Gong, Stefano Markidis, Erwin Laure, Matthew Otten, Paul Fischer, and Misun Min. Nekbone performance on GPUs with OpenACC and CUDA Fortran implementations. *The Journal of Supercomputing*, 72(11):4160–4180, 2016.

[3] David Moxey, Roman Amici, and Mike Kirby. Efficient Matrix-Free High-Order Finite Element Evaluation for Simplicial Elements. *SIAM J. Sci. Comput.*, 42(3):C97–C123, January 2020.

[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.