NEPTUNE Technical Report 2047353-TN-02
Deliverables 1.1, 2.1 and 3.1

Linear systems of equations and preconditioners relating to the NEPTUNE Programme

A brief overview

Authors: V. Alexandrov, A. Lebedev, E. Sahin, S. Thorne

April 22, 2021

Contents

Chapter 1

Landscape of Preconditioners

1.1 Motivational Theory

Intermediate- and large-scale linear systems

$$A\vec{x} = \vec{b} \tag{1.1}$$

are most commonly solved using iterative methods such as

- (symmetric) successive over-relaxation ((S)SOR),
- Jacobi over-relaxation (JOR),
- conjugate gradient (CG),
- biconjugate gradient stabilized (BiCGstab),
- conjugate gradient squared (CGS),
- minimal residual method (MINRES),
- generalized minimal residual method (GMRES),
- quasi-minimal residual (QMR) and
- transpose-free QMR (TFQMR),

see [1], [8], [12], [24] or any university-level introduction to numerical mathematics, such as [30]. These methods do not compute the inverse system matrix A^{-1} or factors of A but approximate the solution via an iterative method of the form

$$\vec{x}_k = \vec{x}_{k-1} + \vec{s}_k \ . \tag{1.2}$$

Their convergence rates are typically bounded from above by expressions involving the condition number of the system matrix A, which is given here for the 2-norm:

$$\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2 \equiv \frac{\max_{\lambda_i \in \sigma(A)} |\lambda_i|}{\min_{\lambda_j \in \sigma(A)} |\lambda_j|} \ , \tag{1.3}$$

where $\sigma(A)$ denotes the spectrum of A. In general, the larger the value of the condition number, the slower the rate of convergence but other characteristics such as the eigenvalues being highly clustered into a few groups can significantly reduce the number of iterations required to achieve the desired level of accuracy. The above immediately shows that matrices which are almost singular, i.e., for which an eigenvalue λ_k exists s.t. $|\lambda_k| \approx 0$ the condition number becomes very large, are likely to have slow convergence. For example, this occurs when a finite-element discretization of a PDE is performed with very high spatial resolution or with almost degenerate finite elements.

Table 1.1: Summary of iterative methods with their properties and any preconditioner requirements. Here "s.p.d." denotes a symmetric positive definite matrix.

Iterative Method	Symmetric A	Non-symmetric A	Other requirements on A	Preconditioner requirements
SOR	Yes	Yes	diag(A) non-singular	N/A
SSOR	Yes	No	diag(A) non-singular	N/A
JOR	Yes	Yes	diag(A) non-singular	N/A
CG	Yes	No	positive definite	s.p.d.
BiCGSTAB	Yes	Yes	None	None
CGS	Yes	Yes	None	None
MINRES	Yes	No	None	s.p.d.
GMRES	Yes	Yes	None	None
QMR	Yes	Yes	None	None
TFQMR	Yes	Yes	None	None

To speed-up the iteration, one can use left and/or right preconditioners P_L and P_R, respectively, and solve the equivalent system

$$P_L A P_R \vec{y} = P_L \vec{b} \quad \text{where} \quad \mathrm{x} = \mathrm{P_R}\tilde{\mathrm{y}}, \tag{1.4}$$

and choose preconditioners for which $P_L A P_R$ has lower condition number than A or clusters the eigenvalues into a few groups such that the number of iterations is significantly reduced, and for which calculating the action of P_L and P_R multiplied by a vector is relatively cheap to initialise and then compute. For simplicity within this report, we will assume that one of P_L or P_R is the identity matrix and we will refer to the other preconditioner as P.

When considering a preconditioner for a given problem it is beneficial to keep in mind what category of PDE has given rise to the linear system at hand as well as the structure properties of the system matrix A. The latter are especially important if methods relying on the symmetry and positive definiteness of the system matrix A are employed (e.g., CG iteration). In such a case, usage of a preconditioner that is not symmetric and positive definite *may* slow the convergence of the iterative method or result in the method breaking down. We provide a summary of the different methods mentioned in Table 1.1.

1.2 Preconditioner Classes

1.2.1 Scalings

A simple way to reduce the condition number of a matrix is to scale its rows/columns to be of approximately equal magnitude w.r.t. a given norm [24],[1]. Generally, the effectiveness of scaling preconditioners can be neglected when comparing them to the other classes listed below. They should, nevertheless, not be neglected, given that they are extremely cheap to compute and can be beneficial in cases where the computation of the matrix-vector product is highly susceptible to round-off errors (i.e., when using mixed precision computations).

1.2.1.1 Point-Jacobi

The simplest preconditioner is the so-called point-Jacobi preconditioner [30], [17], [8]. It is defined as

$$P := \mathrm{diag}(A)^{-1} \quad . \tag{1.5}$$

Due to its simplicity the preconditioner can be directly computed and applied to the iterations of the chosen method. A further benefit of the simplicity is the trivial parallelizability (there are no data-dependencies). The simplicity comes at the cost of effectiveness, this preconditioner will generally only slightly reduce the number of iterations required to achieve convergence.

1.2.1.2 Norm-based scaling

If one chooses

$$P = \mathrm{diag}\left(\left\{\frac{1}{d_{ii}}\right\}_{i=1}^{n}\right) := \mathrm{diag}\left(\left\{\frac{1}{\|\vec{a}_i\|_m}\right\}_{i=1}^{n}\right) \tag{1.6}$$

where $\vec{a}_i$ can be either the i-th row of the matrix A or its i-th column and $\|\cdot\|_m$ is the m norm of a vector, then the preconditioner can be trivially inverted for a direct application to the iteration vector. Such preconditioners are generally referred to as row-/column-scalings. Formally it can be proven that

$$\kappa_\infty(\tilde{P}A) \leq \kappa_\infty(PA) \tag{1.7}$$

for any scaling P if $\tilde{P}$ is computed using the 1 norm ($\|\vec{a}\|_1 = \sum_j |a_j|$) in (1.6), i.e., the preconditioner is an *optimal* scaling.

1.2.2 Incomplete Factorizations

Factorization methods that decompose a matrix into a product of matrices, i.e., $A = LU$, can be used as a basis to derive preconditioners. Note that the factors of the matrix *will generally become dense*, even if A is sparse. This is generally avoided by performing the factorization only for a pre-defined number of non-zero elements. Such a factorization is generally incomplete, hence the name of this preconditioner class. One strategy is to preserve the non-zero pattern of the matrix A, which results in zero-fill preconditioners.

Due to the sequential nature of the underlying factorization, these methods generally require a preliminary graph partitioning (element reordering) to be parallelized. The scalability of such methods is thus *limited* by the number of (strongly) connected components of the graph obtained if the matrix is interpreted as an adjacency matrix of a graph.

1.2.2.1 D-ILU

The second on the triviality scale is the D-ILU preconditioner [11]. It is based on the decomposition of the matrix A similar to the Jacobi iteration:

$$P := (D + L_A)D^{-1}(D + U_A) \quad , \tag{1.8}$$

where D is now not simply the diagonal of A but determined according to a different scheme and U_A, L_A are the strict upper/lower triangular parts of the matrix A.

1.2.2.2 ILU - Incomplete LU Decomposition

A solution of $LU\mathbf{x} = \mathbf{b}$, where L, U are obtained by LU-decomposition (Gaussian elimination) of A is equivalent to $\mathbf{x} = A^{-1}\mathbf{b}$. Here, L is a lower triangular matrix and U is an upper triangular matrix. As such one obvious choice for a preconditioner is the LU-decomposition of A. However, a full LU-decomposition may change a sparse matrix A into a dense one so we could, instead, only perform a step of the LU decomposition if and only if $a_{i,j} \neq 0$. This yields the so-called zero-fill incomplete LU factorization - in short: ILU(0).

ILU(k) denotes an ILU preconditioner with a user-defined fill-in level $k \geq 0$. Higher k correspond generally to a better approximation of the LU decomposition but result in a much denser preconditioner. As a rule $k > 3$ is seldom used [11].

ILUT(ρ,τ) is a threshold variant of ILU, in which entries are removed from the preconditioner if their magnitude falls under the threshold τ or the fraction ρ of additional values per row is exceeded.

1.2.2.3 IC - Incomplete Cholesky Decomposition

Applying the same methodology to the Cholesky decomposition for symmetric positive definite matrices, the IC(k) (incomplete Cholesky decomposition with fill-level k) factorization is obtained. Similar to the common Cholesky decomposition [1], it requires $\sim \frac{1}{2}$ as many operations to compute, as ILU and the resulting decomposition is symmetric, positive definite, making it a prime candidate for an incomplete factorization preconditioner for symmetric (positive definite) matrices and methods which rely on the symmetry of the linear operator (e.g. CG). The same caveats regarding fill-in as for ILU(k) apply.

1.2.2.4 Additive Factorization - Splitting

Methods which rely on a splitting of the matrix A into $A = L+D+R$ which are, by themselves, *not* modified are designated "splitting methods" and can roughly be assigned to the class of factorization methods. In the following, we assume that D is defined as for the Jacobi preconditioner, L is the strict lower triangular part of A and R is the strict upper triangular part of A.

While we provide the preconditioner matrices below *these are generally never computed* and an application of any splitting preconditioner corresponds to the execution of one iteration step of the corresponding iterative solver method.

1.2.2.4.1 Jacobi This preconditioner, which is also considered in Section 1.5, can be considered a splitting or scaling preconditioner, since it corresponds to the preconditioner matrix $P = D^{-1}$.

1.2.2.4.2 (Symmetric) Gauss-Seidel The preconditioner matrix of the simple Gauss-Seidel preconditioner is

$$P = (D + L)^{-1} . \tag{1.9}$$

For the symmetric form, the preconditioner matrix is

$$P = (D + R)^{-1} D (D + L)^{-1} . \tag{1.10}$$

1.2.2.4.3 (Symmetric) SOR This preconditioner is equivalent to the Successive Overrelaxation (SOR) method, resulting in the preconditioner matrix

$$P = \omega (D + \omega L)^{-1} . \tag{1.11}$$

Here ω is a parameter supplied by the user with $\omega \in (0, 2)$. Its symmetrised counterpart is given by

$$P = \omega (2 - \omega)(D + \omega R)^{-1} D (D + \omega L)^{-1} . \tag{1.12}$$

The (s)SOR preconditioners correspond to shortened ($\omega < 1$) or prolonged ($\omega > 1$) Gauss-Seidel steps and, as such, can be attempted once it has been established that the (s)GS preconditioners do not result in the desired reduction of step number. Since an iterative solution is, simply put, an update of the proposed solution $\vec{x}_k$ at step k by a correction - the residual $\vec{r}_k$ - one may introduce an importance weighting of the correction ω: $\vec{x}_{k+1} = \vec{x}_k + \omega \vec{r}_k$. Dependent upon the problem and the current approximation $\vec{x}_k$ a larger or smaller correction to $\vec{x}_k$ can be performed.

1.2.3 Approximate Inverses

Instead of computing an approximate factorization of the sparse matrix A sequentially, one may directly approximate its inverse by minimizing

$$\|I - BA\|_2 \tag{1.13}$$

w.r.t. B in one go. This class of preconditioners generally yields better to parallelisation attempts, at the cost of higher memory complexity, since (1.13) essentially factorizes into independent least-squares approximations. Note that while for the true inverse holds $A^{-1}A = I = AA^{-1}$ this does not hold for approximate inverses, i.e. a right-inverse obtained using $\|I - AB\|_2$ will generally differ from a left inverse, obtained from $\|I - BA\|_2$.

1.2.3.1 SPAI - SParse Approximate Inverse

The method enlarges the non-zero pattern of the approximant B dynamically until the minimization problem is solved to within a provided tolerance. Minimisation of the residual results in the method providing robust preconditioners at the cost of being time-consuming. This approach does not guarantee that the approximate inverse B of a symmetric matrix will be symmetric.

1.2.3.2 FSAI - Factorized Sparse Approximate Inverse

This variant of SPAI does not approximate $B = A^{-1}$ directly but rather the Cholesky factors of B, i.e., L in $A^{-1} \approx L^t L$. Usage of Cholesky factors imposes the same restrictions on the matrix A as the (incomplete) Cholesky decomposition 1.2.2.3 - A has to be symmetric positive definite. If A is s.p.d., then FSAI is well-defined, the converse - in general - does not hold.

1.2.3.3 AINV - Approximate INVerse

Similar to FSAI this method approximates the inverses of triangular factors which, if fully computed, transform the matrix A into a diagonal matrix: $L^{-1}AU^{-1} = D$.

1.2.3.4 MCMCMI - Markov Chain Monte Carlo Matrix Inversion

This method computes a sparse approximate inverse approximation via a random walk on the graph defined by interpreting A as an adjacency matrix. We discuss this further in Section 1.2.5.2.

1.2.4 Multigrid Methods

Similar to the splitting preconditioners, multigrid preconditioners originate from multigrid solution methods. The latter are intended to accelerate the convergence of, e.g., Gauss-Seidel iterations for large systems by essentially coarsening the solution vector $\vec{x}$ (e.g., by selecting only every second element), computing a solution with $\vec{x}$ as the right-hand side vector $\vec{b}$ of a smaller linear system and finally interpolating the coarse solution onto the original solution and updating the original $\vec{x}$ [31].

Originally, the methods were conceived for the solution of elliptic PDE where highly oscillatory components of the residual solution can be damped by solving computing a solution on a coarser mesh and updating the solution on the finer mesh. *Algebraic* multigrid methods (AMG) abstract the geometric picture of a mesh away and attempt to replicate the procedure given only the system matrix A. Multigrid methods vary by the choice of the coarsening and interpolation operators as well as of the coarsening/refinement cycles. These methods are very well suited for linear systems resulting from the discretization of elliptic PDE. They are especially suited for approaches where iterative mesh refinement is utilised. Furthermore, these methods scale well, provided the utilised solver is well-parallelised. In the case of a strongly heterogeneous system (i.e., multiscale system) or of strongly irregular meshes (e.g., mesh refinement at a sharp tip) these methods are known to fail.

1.2.4.1 *l*AIR

Classical AMG method variants are available for both symmetric and nonsymmetric problems although the former are more widely known and used. The *l*AIR preconditioner is a variation on classical AMG for nonsymmetric matrices [22]. The method is based on a local approximation to an ideal restriction operator, which is coupled with F-relaxation. For a given mesh with

vertex set V, the set V is partitioned into F-points and C-points, where C-points represent vertices on the coarse grid. The matrix A can then be symbolically ordered into the following block form:

$$A = \begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix},$$

where A_{ff} corresponds to the F-points. F-relaxation improves the solution at the F-points and this accuracy is then distributed at the C-points via the coarse-grid correction (ideal restriction):

$$R = \begin{pmatrix} -A_{cf}A_{ff}^{-1} & I \end{pmatrix}.$$

It has been shown to be a robust solver for various discretizations of the advection-diffusion-reaction equation in regimes ranging from purely advective to purely diffusive and including time-dependent and steady-state problems.

1.2.5 Stochastic Methods

The methods presented above are deterministic and generally touch each value of the matrix at least once during the computation. Stochastic methods aim to reduce computational costs by utilising only the "important subset" of the matrix/vector entries.

A fairly accurate interpretation of stochastic methods would be as "measurement", where a systematic error (equivalent to the tolerances for deterministic methods) is to be balanced with a stochastic measurement error. Usage of stochastic methods for the computation of preconditioners uses the fact that a preconditioner does not have to be *excellent* to be usable, but instead has to be effective and quick to compute.

1.2.5.1 SP -Stochastic Projection

The basic idea is fairly simple and the implementation follows roughly equation (2.13) of [34]. The main idea is to project the solution vector successively and orthogonally onto arbitrarily chosen subspaces of the row-space of the matrix until the accumulated effect leads the iteration into the subspace of the true solution. An obvious extension to block-projections has been mentioned in [34] with the iteration step given as follows:

$$\vec{x}_{k+1} = \vec{x}_k + A_i^t \left(A_i A_i^t\right)^{-1} \left(\vec{b}_i - A_i \vec{x}_k\right) . \tag{1.14}$$

Here A_i is a randomly selected block of rows of the matrix and $\vec{b}_i$ the corresponding subset of entries of the right-hand-side vector of

$$A\vec{x} = \vec{b} . \tag{1.15}$$

The intuitive simplicity of this approach is paid for by its performance. Furthermore, the computation of a matrix inverse in each step is required. Depending on the block size of the computation of said inverse, or a solution of a dense system, may incur a significant cost.

1.2.5.2 MCMCMI - Markov Chain Monte Carlo Matrix Inversion

This method computes a sparse approximate inverse by performing a random walk on the graph defined by interpreting the matrix A as an adjacency matrix. The entries of the inverse are computed by utilising the Neumann series:

$$A^{-1} = \sum_{i=0}^{\infty} (I - A)^i . \tag{1.16}$$

This requires $\rho(A) < 1$ in general, but by proper scaling can be used even when this condition is not fulfilled. The benefit of this method is that it does not require the matrix A to be explicitly known and the computational cost of computing *either one row of the inverse or one element of the solution vector* scales as $\mathcal{O}(NT)$, where N is the number of Markov chains and T the mean length of a chain.

1.3 Further Remarks

1.3.1 On Parallelism

As has been remarked above, many of the factorization methods require a prior graph-partitioning to restructure the system matrix - ideally in a block-diagonal form. The Gauss-Seidel preconditioning iteration, for instance, carries an explicit data dependency. This can be circumvented by usage of so-called wave-front parallelization [13] but is, as a rule, not part of the solver implementation.

Approximate inverses are generally much more amenable to parallelization due to the aforementioned independence of the least-squares problems that need to be solved. Stochastic methods such as MCMCMI fall into the same category.

1.3.1.1 Domain decomposition

The decomposition of the computational simulation domain with a local ordering of the nodes of the mesh improves the feasibility of factorization methods by reducing the effects of data dependencies and concentrating matrix entries relevant for the local physical domain on the local computational domain. This in turn simplifies the partitioning of the system matrix into block-diagonal form (with or without overlap). An additive Schwarz preconditioner performs a factorization on the (overlapping) blocks in parallel, with an additive averaging on the overlaps.

1.3.2 On Matrices

Most of the aforementioned methods can be used matrix-free, since they generally only require the ability to determine the elements of the matrix and compute a matrix-vector product. Incomplete factorizations with fill-in (ILU(k), IC(k)) are less amenable to usage with matrix-free methods due to the need to compute the fill indices of the matrix elements and to create a factorization with a non-zero pattern larger than the original system matrix.

Scalings and approximate inverses are more amenable to use with matrix-free methods due to the only requirement being the computation of the matrix elements.

1.4 Summary of Preconditioners

In Table 1.2, we summarise the preconditioners as well as their prerequisites and limitations. We distinguish between regular (i.e., non-singular, invertible) matrices and general matrices. The latter do not have to be square, in which case one computes an approximate pseudo-inverse, rather than an approximate true inverse.

The column "CG usable" indicates whether a preconditioner computed with a given method is usable with an iterative solver which requires a symmetric, positive matrix (represented by the CG iteration), i.e., whether the method produces a symmetric, positive definite preconditioner. This information is intended as a rough guide only, since asymmetric preconditioners (e.g., as computed by SPAI or MCMCMI) may still work as intended.

Constraints on the system matrix given in the second column stem, in part, from theoretical considerations. Methods such as IC, (s)SOR, FSAI may still compute a valid preconditioner if the system matrix is indefinite but the theory guarantees that, with exact arithmetic, these methods will not fail if the matrix is symmetric and positive definite.

Table 1.2: Summary of preconditioners and their application scenarios. Here "s.p.d." denotes a symmetric positive definite matrix, "regular" an invertible matrix, "general" a general (non-square) matrix and "d.d." a dominant diagonal. "GP" abbreviates "graph partitioning" and "WP" "wavefront parallelization"[13] .

Preconditioner	System matrix	PDE type	Matrix-free	A required explicitly	parallelizable	CG usable
Jacobi	$0 \notin \text{diag}(A)$, regular	general	Yes	No	Yes	Yes
Scaling	regular	general	Yes	No	Yes	Yes
D-ILU	$0 \notin \text{diag}(A)$, regular	general	Yes	No	via GP	Yes
ILU($k \geq 0$)/ILUT(ρ, τ)	regular	general	No	Yes	via GP	No
IC($k \geq 0$)	s. p. d.	general (elliptic)	No	Yes	via GP	Yes
(s)GS	regular, d.d.	general	Yes	No	via WP	sGS only
(s)SOR	s. p. d.	general (elliptic)	Yes	No	via WP	sSOR only
SPAI	general	general	No	No	Yes	No
FSAI	s. p. d.	general (elliptic)	No	No	Yes	Yes
AINV	regular	general	No	No	limited	No
AMG	regular	elliptic, hyperbolic, parabolic	Yes	No	Yes	Yes
lAIR	regular	elliptic, parabolic	Yes	No	undetermined	-
SP	general	general	Yes	No	Yes	No
MCMCMI	general	general	No	No	Yes	No

Chapter 2

Application Cases

2.1 Introduction

Due to the limited duration of the NEPTUNE Preconditioning Project, the variety of problems and test cases available within the NEPTUNE programme [2] has to be reduced to a manageable level. In the following we present the equations and test cases that will be considered within the given project.

2.2 Elliptic Problems

For elliptic problems, System 2-2 from [2] is our priority, which consists of a 2-D elliptic solver in complex geometry. BOUT++ [9] already has some test cases and Nektar++ [35] also has some suitable cases. BOUT++ has finite difference examples whilst Nektar++ uses finite and spectral/*hp* elements for its discretizations. There is a solver in BOUT++ that sets up a matrix problem and calls PETSc [29], and another implementation of the same problem which calls HYPRE [10]. Since BOUT++ uses finite differences and not finite, high order, elements, there are plans within the NEPTUNE Programme to implement the Nektar++ version over the next 6 months or so.

2.3 Hyperbolic Problems

For hyperbolic problems, System 2-3 from [2] is our priority case. There is already a BOUT++ test case called SD1D [9] that models the dynamics along the magnetic field, utilises finite differences and is a matrix-free implementation that uses SUNDIALS [40]. A version of this test problem is going to be set up using Nektar++ during the next few months. For the dynamics across the magnetic field, there are 2D problems like Hasegawa-Wakatani, which are a similar to incompressible fluid dynamics:

$$\frac{\partial n}{\partial t} = -\{\phi, n\} + \alpha(\phi - n) - \kappa\frac{\partial \phi}{\partial z} + D_n\nabla_\perp^2 n \tag{2.1a}$$

$$\frac{\partial \omega}{\partial t} = -\{\omega, n\} + \alpha(\omega - n) + D_\omega\nabla_\perp^2 \omega \tag{2.1b}$$

$$\nabla^2\phi = \omega\ . \tag{2.1c}$$

Here n is the plasma number density, $\omega := \vec{b}_0\cdot\nabla\times\vec{v}$ is the vorticity with $\vec{v}$ being the $\vec{E}\times\vec{B}$ drift velocity in a constant magnetic field and $\vec{b}_0$ is the unit vector in the direction of the equilibrium magnetic field. The operator $\{\cdot,\cdot\}$ in (2.1) is the Poisson bracket.

Let us assume for simplicity, that n, ω, ϕ are discretized using the same basis in space and an implicit Euler method is used for the time evolution, then the following set of non-linear equations is obtained from (2.1):

$$\begin{aligned}
0 &= M(\vec{n}_i - \vec{n}_{i-1}) + \Delta t \left(\text{diag}\left(L_x\vec{\phi}_i\right) L_z\vec{n}_i - \text{diag}\left(L_z\vec{\phi}_i\right) L_x\vec{n}_i \right. \\
&\quad \left. - \alpha M\left(\vec{\phi}_i - \vec{n}_i\right) - \kappa L_z\vec{\phi}_i - D_n K\vec{n}_i\right) && \text{(2.2a)} \\
0 &= M\left(\vec{\omega}_i - \omega_{i-1}\right) + \Delta t \left(\text{diag}\left(L_x\vec{\phi}_i\right) L_z\vec{\omega}_i - \text{diag}\left(L_z\vec{\phi}_i\right) L_x\vec{\omega}_i \right. \\
&\quad \left. - \alpha M\left(\vec{\phi}_i - \vec{n}_i\right) - D_\omega K\vec{\omega}_i\right) && \text{(2.2b)} \\
0 &= K\vec{\phi}_i - M\vec{\omega}_i \ , && \text{(2.2c)}
\end{aligned}$$

where K, M are the stiffness and mass matrices respectively and L_x, L_z are transport matrices and represent the discretized versions of ∂_x, ∂_z [6]. Note that the first two terms in the Δt bracket in the first two equations (the terms with two L operators) are straightforward discretizations of the Poisson bracket $\{f, g\} = \partial_x f \partial_z g - \partial_z f \partial_x g$ where the recasting of, e.g. $L_z\vec{\phi}$, is necessary to ensure that the result of $\text{diag}\left(L_z\vec{\phi}_i\right) L_x\vec{n}_i$ is again a vector (a discretized function).

Since the equations are non-linear they have to be solved e.g. via Newton's method, which requires the Jacobian of the system. The latter has the following form:

$$J := \begin{bmatrix} A & 0 & E \\ B & C & G \\ 0 & -M & K \end{bmatrix} \ , \tag{2.3}$$

where the constituent matrices are the following

$$\begin{aligned}
A &= M + \Delta t \left(\text{diag}\left(L_x\vec{\phi}_i\right) L_z - \text{diag}\left(L_z\vec{\phi}_i\right) L_x + \alpha M - D_n K\right) && \text{(2.4a)} \\
B &= \alpha \Delta t M && \text{(2.4b)} \\
C &= M + \Delta t \left(\text{diag}\left(L_x\vec{\phi}_i\right) L_z - \text{diag}\left(L_z\vec{\phi}_i\right) L_x - D_\omega K\right) && \text{(2.4c)} \\
E &= \Delta t \left(\text{diag}\left(L_z\vec{n}_i\right) L_x - \text{diag}(L_x\vec{n}_i) L_z - \alpha M - \kappa L_z\right) && \text{(2.4d)} \\
G &= \Delta t \left(\text{diag}(L_z\vec{\omega}_i) L_x - \text{diag}(L_x\vec{\omega}_i) L_z\right) \ . && \text{(2.4e)}
\end{aligned}$$

Alternatively the last equation of (2.2) can be used to eliminate $\vec{\phi}_i$ by virtue of $\vec{\phi}_i = K^{-1}M\vec{\omega}_i$. The Jacobian of the reduced system is then of the form

$$\tilde{J} := \begin{bmatrix} P & R \\ Q & S \end{bmatrix} \ , \tag{2.5}$$

with the constituent matrices

$$\begin{aligned}
P &= M + \Delta t \left(\text{diag}\left(L_x K^{-1} M\vec{\omega}_i\right) L_z - \text{diag}\left(L_z K^{-1} M\vec{\omega}_i\right) L_x + \alpha M - D_n K\right) && \text{(2.6a)} \\
Q &= \alpha \Delta t M && \text{(2.6b)} \\
R &= \Delta t \left(\text{diag}\left(L_z\vec{n}_i\right) L_x K^{-1} M - \text{diag}(L_x\vec{n}_i) L_z K^{-1} M - \alpha M K^{-1} M - \kappa L_z K^{-1} M\right) && \text{(2.6c)} \\
S &= M + \Delta t \left(\text{diag}\left(L_x K^{-1} M\vec{\omega}_i\right) L_z + \text{diag}\left(L_z\vec{\omega}_i\right) L_x K^{-1} M - \alpha M K^{-1} M - D_\omega K\right) && \text{(2.6d)}
\end{aligned}$$

As mentioned above, even if the stiffness matrix K is sparse its inverse will generally be dense. Hence, for the discretization methods of interest for the project, the Jacobian of the reduced system $\tilde{J}$ will not be sparse. Therefore, one of the questions is whether to (a) use the

reduced system but have a dense Jacobian, or (b) treat the elliptic problem as a constraint but have a sparse, larger and more ill-conditioned Jacobian. Option (a) is normally done but experiments have also been done in BOUT++ (b) in order to use PETSc's matrix coloring to extract an approximate matrix for pre- conditioning from our matrix-free code. Nektar++ also has an implementation of the Hasegawa-Wakatani problem.

2.4 Comments

When possible, Nektar++ examples should have higher priority than BOUT++ due to the expectation that finite element and spectral discretizations will be the method of choice for future simulations. In addition, we should expect adaptive hp-refinement to be used. Nektar++ is moving towards matrix-free implementations. For Nektar++, we need to keep in contact with David Moxey.

Chapter 3

Implementations

3.1 Introduction

For scientific computing, there are widely-used platforms such as (among others) PETSc/TAO [29], Trilinos, BOUT++ [9], Nektar++[35] and SUNDIALS [40] available. According to the prioritised equations in Chapters 2, there are several studies using these platforms accessible in the literature.

3.2 Elliptic Problems

As indicated in Chapter 2, the Grad-Shafranov equations is used to generate spectrally accurate magnetic fields to use in other proxyapps [2]. There are highly-scalable, multi-physics implementations for Grad-Shafranov using finite difference [20] [41] and finite element [28] methods with PETSc as well as finite element methods [32] [5] with Trilinos. To solve another equation of interest, the non-Boussinesq vorticity equation, [3] used preconditioned Conjugate Gradient with a Block–Jacobi preconditioner from PETSc. In [18], an inexact Newton-Krylov approach with PETSc was used to solve a similar problem. In addition, [4] uses a matrix-free method from PETSc to solve a non-Boussinesq formulation of polythermal ice flow. In [33], similar problems were solved with massive MPI parrallelism using Krylov methods with preconditioners (ILU(k) and block Jacobi) from PETSc. As discussed in Chapter 2, there are implementations and suitable cases available in BOUT++ [9] and Nektar++ [35]. There are also tools which use the mixture of platforms such as libMesh [16] and CoolFluiD[19] framework specialised for plasma and multi-physics simulations.

3.3 Hyperbolic Problems

As stated in Chapter 2, the hyperbolic case focuses on system 2-3 of [2], of which several related implementations and test cases exist in BOUT++ and Nektar++.

In the literature, several studies exist in which highly parallelized solvers for a 2-fluid model have been considered and implemented. In [23], the linearized two-fluid MHD equations were solved using a matrix-free Newton-Krylov method of solution in XTOR-2F (PETSc) with MPI parallelism. In [39], an MPI parallelized Vlasov Fokker-Planck (VFP) set of equations was solved (IMPACT [15]) using BICGstab as solver with an ILU preconditioner provided by PETSc. In [21], a system of multi-fluid equations was solved using GMRES with a parallel Additive Schwarz preconditioner provided by PETSc. VFP-based equations are also solved by a package introduced in [25] and based on PETSc with MPI parallelism. In addition, there is a package [38] available within Trilinos.

3.4 Other libraries of interest to the NEPTUNE Programme

Clearly, the preconditioners from PETSc and Trilinos are already highly-used and their applicability for future exascale implementations will need exploring. However, our literature searches also identified other interesting libraries libraries.

DUNE is an open-source high-performance iterative solver that has an implementation of the GenEO preconditioner [36] for strongly anisotropic elliptic partial differential equations implemented within dune-pdelab. This preconditioner has a MPI implementation: good weak and strong scalability demonstrated on ARCHER. We note that there is a version of the GenEO preconditioner is available in PETSc.

HYPRE [10] is a library of scalable linear solvers and multigrid methods from Lawrence Livermore National Laboratory. MPI implementations are provided.

HSL_MI20 is an implementation of an algebraic multigrid preconditioner for nonsymmetric systems from the HSL library [14]. Unfortunately, parallelism is only provided through the use of parallelized Level 3 BLAS subroutine calls and it is not suitable for matrix-free/operator only provision of A.

MUMPS [26] is a Fortran 95 library that utilizes MPI and OpenMP to solve large sparse systems via direct methods and would be a good option for comparing iterative methods with direct methods. The library is developed within a consortium of people from CERFACS, ENS Lyon, INPT(ENSEEIHT)-IRIT, Inria, Mumps Technologies and the University of Bordeaux.

PaStiX (Parallel Sparse matriX package) [27] is a scientific library that provides a high performance parallel solver for very large sparse linear systems based on direct methods. The library can also be used to form preconditioners. As well as using MPI and OpenMP, the most recent versions of the library also support the use of GPUs.

The STRUMPACK [37] library provides linear algebra routines and linear system solvers for sparse and for dense rank-structured linear systems. The library has Fortran and C interfaces and can also be used to compute incomplete LU factorization-based preconditioners. Parallization is provided via MPI and OpenMP but the preconditioner routines do not currently support GPUs.

We also direct the reader to Jack Dongarra's list of freely available software for linear algebra [7], which provides a list of software and associated attributes, and is periodically updated.

Bibliography

[1] P. Arbenz, O. Chinellato, M. Gutknecht, and M. Sala. Software for Numerical Linear Algebra. online, 2006.

[2] W. Arter and R. Akers. ExCALIBUR equations for NEPTUNE Proxyapps. techreport, UK Atomic Energy Authority, 2020.

[3] B. Bigot, T. Bonometti, L. Lacaze, and O. Thual. A simple immersed-boundary method for solid–fluid interaction in constant-and stratified-density flows. *Computers & Fluids*, 97:126–142, 2014.

[4] J. Brown, M. G. Knepley, D. A. May, L. C. McInnes, and B. Smith. Composable linear solvers for multiphysics. In *2012 11th International Symposium on Parallel and Distributed Computing*, pages 55–62. IEEE, 2012.

[5] P. Daniel, A. Ern, I. Smears, and M. Vohralík. An adaptive hp-refinement strategy with computable guaranteed bound on the error reduction factor. *Computers & Mathematics with Applications*, 76(5):967–983, 2018.

[6] P. Deuflhard and M. Weiser. *Adaptive Numerical Solution of PDEs*. deGruyter, 2013.

[7] J. Dongarra. Freely Available Software for Linear Algebra, 2018. URL `http://www.netlib.org/utk/people/JackDongarra/la-sw.html`.

[8] J. Dongarra, R. Barett, M. Berry, T. F. Chan, J. Demmel, J. M. Donato, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, 2014.

[9] B. Dudson, P. Hill, and J. Parker. BOUT++. online repository, 2020. URL `http://boutproject.github.io`.

[10] R. Falgout, A. Barker, T. Kolev, R. Li, S. Osborn, D. Osei-Kuffuor, V. P. Magri, and J. Schroeder. HYPRE: Scalable Linear Solvers and Multigrid Methods. online, 2017. URL `https://computing.llnl.gov/projects/hypre-scalable-linear-solvers-multigrid-methods`.

[11] M. Ferronato. Preconditioning for Sparse Linear Systems at the Dawn of the 21st Century: History, Current Developments, and Future Prospects. *ISRN Applied Mathematics*, 2012:49, October 2012. doi: 10.5402/2012/127647.

[12] G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3. JHU press, 2013.

[13] G. Hager and G. Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. Chapman and Hall/CRC, 2011. Asian Reprint.

[14] HSL Development Team. The HSL Mathematical Software Library, 2015. URL `https://www.hsl.rl.ac.uk`.

[15] R. Kingham and A. Bell. An implicit Vlasov–Fokker–Planck code to model non-local electron transport in 2-D with magnetic fields. *Journal of Computational Physics*, 194(1):1–34, 2004.

[16] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey. libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations. *Engineering with Computers*, 22(3-4):237–254, 2006.

[17] D. Kressner. *Lecture notes for the course on Numerical Methods held by Daniel Kressner in the spring semester 2010 at the ETH Zurich. (Numerische Methoden Vorlesungsskript zur Veranstaltung Numerische Methoden gehalten von Daniel Kressner im FS 2010 an der ETH Zürich)*. ETH Zürich, May 2010.

[18] M. Kumar and G. Natarajan. Unified solver for thermobuoyant flows on unstructured meshes. In *Fluid Mechanics and Fluid Power–Contemporary Research*, pages 569–580. Springer, 2017.

[19] A. Lani, T. Quintino, D. Kimpe, H. Deconinck, S. Vandewalle, and S. Poedts. The COOLFluiD framework: design solutions for high performance object oriented scientific computing software. In *International Conference on Computational Science*, pages 279–286. Springer, 2005.

[20] S. Liu, Q. Tang, and X.-Z. Tang. A parallel cut-cell algorithm for the free-boundary Grad-Shafranov problem. *arXiv preprint arXiv:2012.06015*, 2020.

[21] Y. G. Maneva, A. A. Laguna, A. Lani, and S. Poedts. Multi-fluid modeling of magnetosonic wave propagation in the solar chromosphere: effects of impact ionization and radiative recombination. *The Astrophysical Journal*, 836(2):197, 2017.

[22] 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.

[23] A. Marx and H. Lütjens. Hybrid parallelization of the XTOR-2F code for the simulation of two-fluid MHD instabilities in tokamaks. *Computer Physics Communications*, 212:90–99, 2017.

[24] A. Meister. *Numerics of systems of linear equations (Numerik Linearer Gleichungssysteme)*. Springer Spektrum, 5 edition, 2014. doi: 10.1007/978-3-658-07200-1. With MATLAB exercises.

[25] S. Mijin, A. Antony, F. Militello, and R. J. Kingham. SOL-KiT—Fully implicit code for kinetic simulation of parallel electron transport in the tokamak Scrape-Off Layer. *Computer Physics Communications*, 258:107600, 2021.

[26] MUMPS Development Team. MUMPS: MUltifrontal Massively Parallel sparse direct Solver, 2020. URL `http://mumps.enseeiht.fr/index.php?page=home`.

[27] PaStiX Development Team. PaStiX: Parallel Sparse matriX package, 2019. URL `http://pastix.gforge.inria.fr/files/README-txt.html`.

[28] Z. Peng, Q. Tang, and X.-Z. Tang. An Adaptive Discontinuous Petrov–Galerkin Method for the Grad–Shafranov Equation. *SIAM Journal on Scientific Computing*, 42(5):B1227–B1249, 2020.

[29] PETSc Development Team. PETSc Web page, 2019. URL `https://www.mcs.anl.gov/petsc`.

[30] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*. Springer, 2002.

[31] R. Rabenseifner, A. Meister, et al. Iterative Solvers and Parallelization - Course material for HLRS Course 2016-ITER-S, 2016.

[32] N. V. Roberts, D. Ridzal, P. B. Bochev, and L. Demkowicz. A toolbox for a class of discontinuous Petrov-Galerkin methods using Trilinos. *Technical Report SAND2011-6678, Sandia National Laboratories*, 2011.

[33] A. L. Rossa and A. L. Coutinho. Parallel adaptive simulation of gravity currents on the lock-exchange problem. *Computers & Fluids*, 88:782–794, 2013.

[34] K. Sabelfeld and N. Loshchina. Stochastic iterative projection methods for large linear systems. *Monte Carlo Methods and Applications*, 2010. doi: 10.1515/mcma.2010.020.

[35] S. Sherwin, M. Kirby, C. Cantwell, and D. Moxey. Nektar++. online, 2021. URL `https://www.nektar.info`.

[36] 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*, 1(126):741–770, 2014.

[37] STRUMPACK Development Team. STRUMPACK - STRUctured Matrix PACKage, 2021. URL `https://portal.nersc.gov/project/sparse/strumpack/v5.0.0/index.html`.

[38] Trilinos Project Team. Stokhos Package for Intrusive Stochastic Galerkin Methods, 2021. URL `https://trilinos.github.io/stokhos.html`.

[39] B. Williams and R. Kingham. Hybrid simulations of fast electron propagation including magnetized transport and non-local effects in the background plasma. *Plasma Physics and Controlled Fusion*, 55(12):124009, 2013.

[40] C. S. Woodward, D. R. Reynolds, A. C. Hindmarsh, D. J. Gardner, and C. J. Balos. SUNDIALS: SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers. online, 2021. URL `https://computing.llnl.gov/projects/sundials`.

[41] P. Zhu, C. Sovinec, C. Hegna, A. Bhattacharjee, and K. Germaschewski. Nonlinear ballooning instability in the near-Earth magnetotail: Growth, structure, and possible role in substorms. *Journal of Geophysical Research: Space Physics*, 112(A6), 2007.