Science and Technology Facilities Council

Preconditioners for the NEPTUNE Programme: Numerical Results

Technical Report 2068625-TN-01
Deliverable D1.1

Maksims Abaļenkovs* Vassil Alexandrov* Niall Bootland† Anton Lebedev*
Emre Sahin* Sue Thorne**

March 2023 (Revised May 2023)
(Revised version of 2047353-TN-03)

1 Introduction

In simulations relating to plasma physics, and more generally, the dominant component in terms of simulation time is normally the time to solve the underlying linear systems

$$Ax = b.$$

In this report, we will assume that $A \in \mathbb{R}^{n\times n}$ is non-singular, the right-hand side $b \in \mathbb{R}^n$ is provided and we wish to compute an (approximate) solution $x \in \mathbb{R}^n$. These systems are usually solved via an iterative method such a Krylov subspace method. Typically, the nature of these systems mean that a large number of iterations are required to reach the desired level of accuracy. To reduce the number of iterations, we aim to choose a preconditioner $P \in \mathbb{R}^{n\times n}$ such that applying the iterative method to the equivalent system

$$P^{-1}Ax = P^{-1}b$$

results in a reduction in the number of iterations, and a reduction in the solution time (including the time to initialise P): note, in practice, the latter is of higher importance. In the above, we have described left preconditioning. For right preconditioning, one is solving the equivalent system

$$AP^{-1}y = b, \quad x = P^{-1}y.$$

It is also possible to use a combination of left and right preconditioners together:

$$P_1^{-1}AP_2^{-1}y = P_1^{-1}b, \quad x = P_2^{-1}y.$$

2 Software for Plasma Physics Modelling: Overview

In the following, we describe our use of BOUT++ and Nektar++, which have been identified by UKAEA to be the modelling packages of interest for the NEPTUNE Project. As part of the descriptions, we include how preconditioners are currently incorporated into these libraries.

*The authors are with the Hartree Centre, STFC Daresbury Laboratory, Sci-Tech Daresbury, Keckwick, Daresbury, Warrington, WA4 4AD, UK. Email contact: maksims.abalenkovs@stfc.ac.uk

†Niall Bootland is with the Scientific Computing Department, STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, OX11 0QX, UK.

**Sue Thorne is with the Hartree Centre, STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, OX11 0QX, UK. Email contact: sue.thorne@stfc.ac.uk

2.1 BOUT++

BOUT++ [5] is very much designed for matrix-free calculations, which use external packages PETSc [13, 14] or SUNDIALS [6, 7, 9] to solve the more complex problems. In themselves, these external packages use matrix-free implementations by default. This means that custom preconditioners either need to be developed in a matrix-free, operator-based manner and provided via BOUT++ or, for preconditioners that can act in a matrix-free manner but require access to the action of the linear system being solved, it would need implementing within the underlying library.

As part of this project, we used a nonlinear diffusion test case that simulates the effects of heat conduction in a plasma. This test case is provided as part of BOUT++. At the moment, BOUT++ relies on the PETSc package for the preconditioning work within this test case. There are multiple ways to execute the heat conduction example. In the simplest possible form the code is launched with

```
./diffusion-nl
```

IMEX-BDF2 multistep scheme is launched with

```
./diffusion-nl solver:type=imexbdf2
```

Preconditioning is enabled with the following flag

```
./diffusion-nl solver:use_precon=true
```

Finally, Jacobian colouring with IMEX-BDF2 is enabled with

```
./diffusion-nl solver:type=imexbdf2 solver:matrix_free=false
```

The end command used in the experiments is

```
./diffusion-nl solver:type=imexbdf2 solver:matrix_free=false solver:use_precon=true
```

Extracting and saving the system matrix was done via PETSc routines. In order to save the matrix `imex-bdf2.cxx` source code file was modified. The `precon` routine was supplemented with the following commands:

```
1:  PetscViewer viewer;
2:  PetscViewerASCIIOpen(MPI_COMM_WORLD, "A.mtx", &viewer);
3:  PetscViewerPushFormat(viewer, PETSC_VIEWER_ASCII_MATRIXMARKET);
4:  MatView(Jmf, viewer);
```

Figure 1: C++ source code for system matrix extraction

In this report, we also consider the SD1D test case [4], which uses BOUT++ to simulate a plasma fluid in one dimension (along the magnetic field) that interacts with a neutral gas fluid. Unlike the nonlinear diffusion example, this test case provides a preconditioner as part of the model in an operator-based manner.

2.2 Nektar++

Nektar++ [2, 12] is designed to provide discretisation and solution of partial differential equations using the spectral/hp element method.

Nektar++ accepts `*.xml` or similar formats as inputs, where the input file provides finite-element mesh and the specifications to solve the specific PDE problem. The specification of the mesh format within Nektar++ is hierarchically defined as: 1D edges as connection of vertices, 2D faces as bounding edges and 3D elements as bounding faces. Nektar++'s MultiRegions library derives mapping from these meshes and uses these mappings for different Galerkin projection methods to assemble a global linear system. The global linear system may be solved by (i) using direct methods based on Cholesky or LU factorisations, or (ii) using an iterative preconditioned Krylov subspace method. Nektar++ provides a range of the preconditioners such as the classical Jacobi preconditioner, coarse-space preconditioner, block preconditioner and low-energy preconditioner. It also provides an interface to PETSc for additional solvers and preconditioners. Nektar++ evaluates L_2 and L_{inf} errors against the provided exact solution.

The MCMCMI software that implements the Monte Carlo Markov Chain Matrix Inverse preconditioner [8, 10, 11] is structured in a manner that made it infeasible to follow the steps described in [2] and adapt it as a preconditioner module for the Nektar++ in the timeline of the project. Therefore, Nektar++ is edited accordingly to extract full system matrices as Matrix Market format `*.mtx` to use it separately in the MCMCMI algorithm.

Nektar++ provides an Advection-Diffusion-Reaction solver to solve partial differential equations of the form (1) in either discontinuous or continuous projections of the solution field [12].

$$\alpha\frac{\partial u}{\partial t}+\lambda u+\nu\nabla u+\epsilon\nabla\cdot(D\nabla u) \quad = \quad f \tag{1}$$

However, it is not possible to construct linear systems for the discontinuous Galerkin method. Hence, application cases using the continuous Galerkin method were chosen from 'Nektar++/Solvers/ADRSolver/Tests' and used to generate the prioritised equations as described in [1].

2.3 Testing Environment

Numerical experiments were run on the SKL (Skylake) nodes of the Scafell Pike system which consists of nodes fitted with 2x XEON gold E5-6142 v5 processors resulting in 32 cores per node and, due to HyperThreading, 64 threads per node.

3 Numerical Results for Sparse Approximate Inverse Preconditioners

MCMCMI experiments were performed on four system matrices extracted from the BOUT++ software. These matrices were created for the one-dimensional "non-linear diffusion" test problem called `diffusion-nl`. The main BOUT++ source code was modified to enforce PETSc, powering the solution of `diffusion-nl` problem, to output system matrices into the Matrix Market `*.mtx` format. Matrix orders of the extracted BOUT++ system matrices were $n = 128$, 512, 2048 and 8192. The ever-increasing system matrices were obtained by increasing the resolution over the y axis. Unfortunately, it was not possible to extend the problem even further since PETSc started to crash for $n > 8192$.

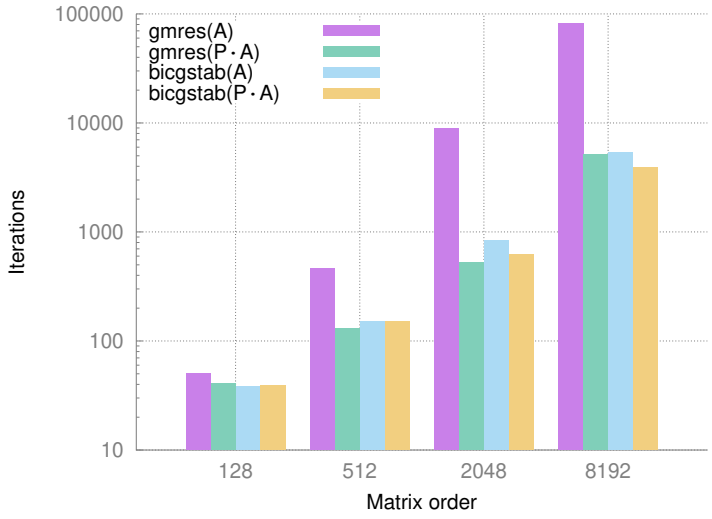

Figure 2: **Matrix order vs GMRES and BiCGStab iteration steps, BOUT++.**

In Figure 2, we compare the matrix order vs GMRES and BiCGStab iteration steps for the BOUT++ examples. In the case of GMRES method and small matrices ($n = 128$, 512), the MCMCMI preconditioner P decreases the number of iterations required for the linear system solution by 18% and 72%, respectively. For the GMRES method applied to larger matrix orders ($n = 2048$, 8192), the MCMCMI preconditioner shows a dramatic improvement of 94%. On the other hand, using the BiCGStab solver, the MCMCMI-based preconditioner provides performance comparable with the reference solution for small matrices ($n = 128$, 512). For larger system matrices ($n = 2048$, 8192), the MCMCMI preconditioner reduces the number of iterations by 26% and 27%, respectively.

We compare the L^1 norm values of the residual for our approximate solutions from GMRES and BiCGStab for the BOUT++ test problems in Figure 3. The highest values of L^1 norm are provided by the preconditioned linear system solve in the GMRES method. L^1 norm values for non-preconditioned solutions in GMRES and BiCGStab are comparable. Finally, the smallest values of L^1 norm are obtained by the preconditioned solution with the BiCGStab. One exception is the L^1 norm values for the smallest test matrix ($n = 128$).

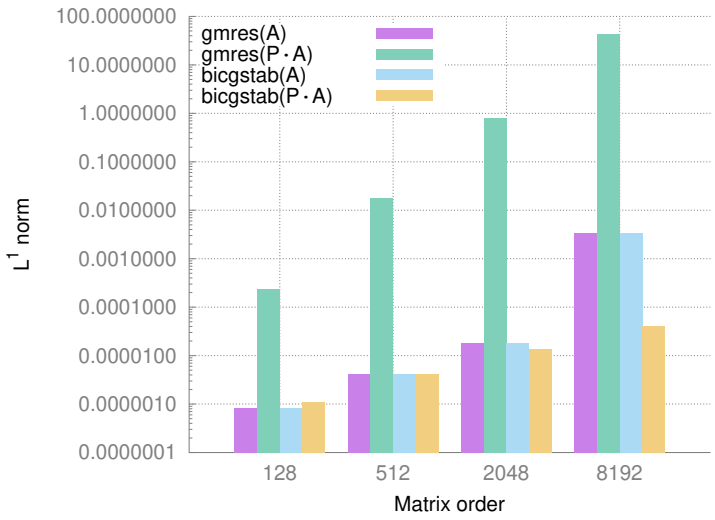

Figure 3: **Matrix order vs L^1 norm values in GMRES and BiCGStab, BOUT++.**

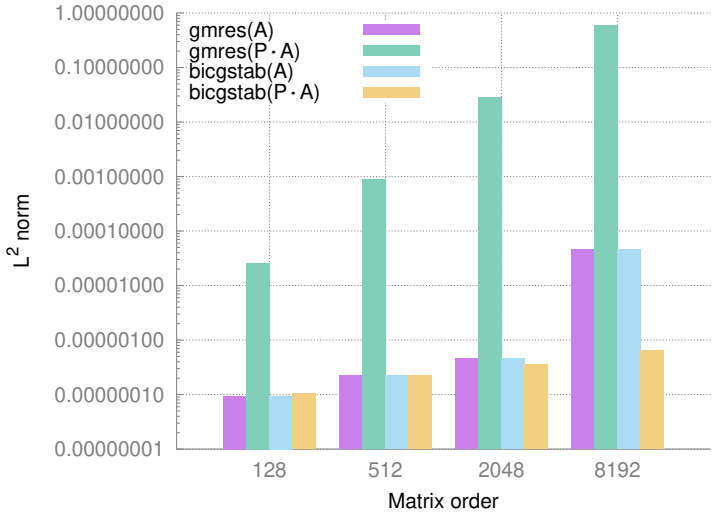

Figure 4: **Matrix order vs L^2 norm values in GMRES and BiCGStab, BOUT++.**

In Figure 4, we compare the L^2 norm values of the residuals for our approximate solutions from GMRES and BiCGStab for the BOUT++ test problems. The L^2 norm values follow the same tendency as the L^1 norms described above (see Fig. 3 for details). The highest values are produced by the preconditioned solution with the GMRES method and the lowest values by the preconditioned solution with the BiCGStab algorithm. The L^∞ norm values are compared in Figure 5. L^∞ norms exhibit behaviour similar to the L^1 and L^2 norms. Overall, the L^∞ norms are the smallest amongst all norms. With rare exceptions the preconditioned system solutions resulted in the highest (GMRES) and the lowest (BiCGStab) norm values. We note that a left preconditioner is being used within the GMRES method and, hence, at each iteration k, the preconditioned residual

$$||P^{-1}(b-Ax_k)||_2$$

is minimimised, where x_k is a member of the Krylov subspace defined by

$$\text{span}\left\{P^{-1}r_0, P^{-1}AP^{-1}r_0, \ldots, \left(P^{-1}A\right)^k P^{-1}r_0\right\}$$

with $r_0 = b - Ax_0$. Now,

$$\frac{||P^{-1}(b-Ax_k)||_2}{||P^{-1}||_2} \leq ||b-Ax_k||_2 \leq ||P||_2||P^{-1}(b-Ax_k)||_2$$

and, hence, left preconditioned GMRES is not minimising the L^2 norm of the residual and the values of $||P||_2$ and $||P^{-1}||_2$ are likely to be such that whilst $||P^{-1}(b-Ax_k)||_2$ can be small, the value of $||b-Ax_k||_2$ can be large. There is no minimisation property for the iterations of BiCGStab but these results lead to the question: would right preconditioned GMRES produce similar results to BiCGStab? In the future, we would like to investigate this with larger problem sizes.

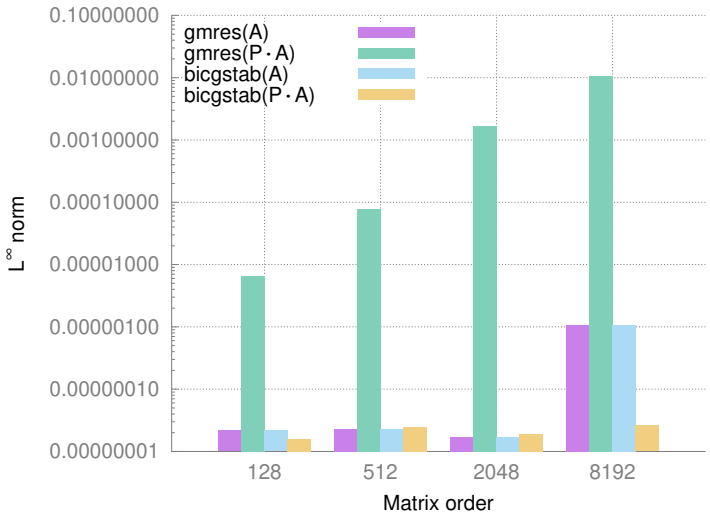

Figure 5: **Matrix order vs L^∞ norm values in GMRES and BiCGStab, BOUT++.**

In Table 1, we provide the details for the matrices that were extracted from Nektar++. We compare the number of GMRES and BiCGStab steps in Figure 6. Since the iterations are higher than non-preconditioned GMRES method, the preconditioner failed for the GMRES method for all except the Helmholtz problems. However, the preconditioner is successful for the BiCGStab method. Higher L^2 and L^∞ values also supports the failure of the preconditioner for the GMRES method, likewise success for the BiCGStab method as shown in the Figures 7 and 8.

In the future, we would like to investigate the use of the MCMCMI preconditioner with test problems that arise from anisotropic problems to see if the performance is markedly different.

Matrix name	Label	n	Mode | Order	Int timestep
Helmholtz	H	1312	2D modal	
Steady Adv-Diff	SAD	2281	2D modal	
Unsteady Adv-Diff 1	UAD1	225	Order 1	001
Unsteady Adv-Diff 2	UAD2	225	Order 1	0001
Unsteady Adv-Diff 3	UAD3	225	Order 2	001
Unsteady Adv-Diff 4	UAD4	225	Order 2	0001

Table 1: **Nektar++ matrices used in MCMCMI preconditioner experiments.**

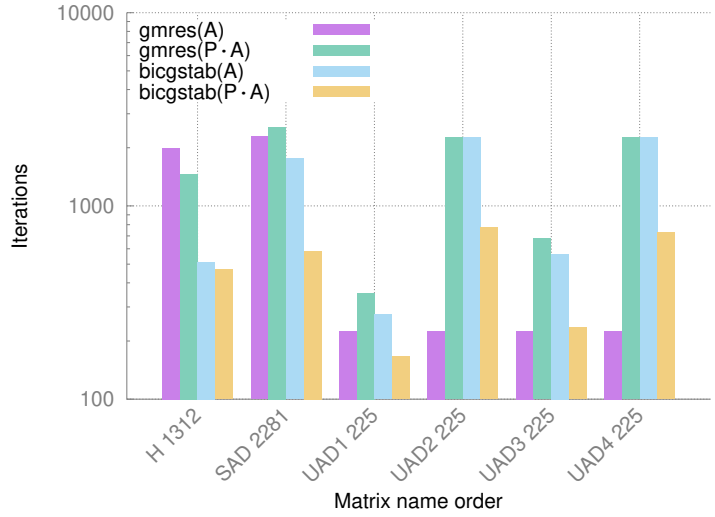

Figure 6: **Matrix order vs GMRES and BiCGStab iteration steps, Nektar++.**

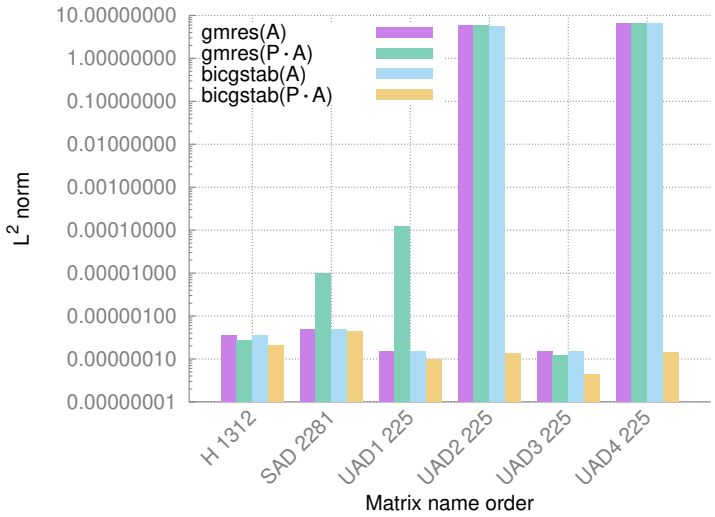

Figure 7: **Matrix order vs L^2 norm values in GMRES and BiCGStab, Nektar++.**

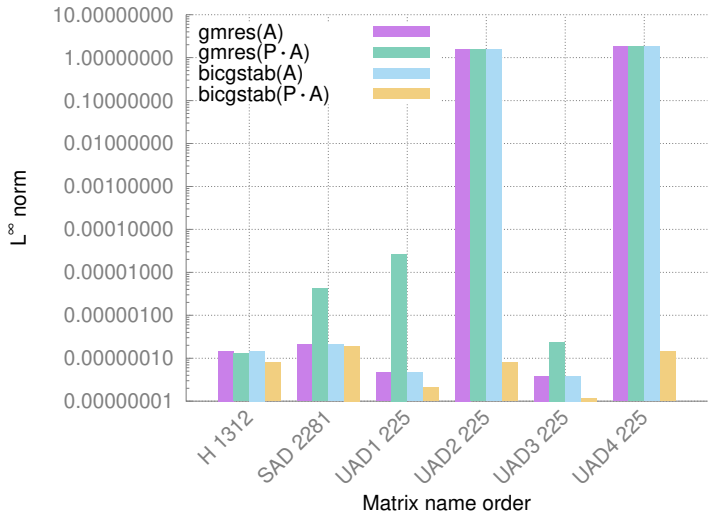

Figure 8: **Matrix order vs L^∞ norm values in GMRES and BiCGStab, Nektar++.**

4 Numerical results for operator-based preconditioners

Here, we focus on operator-based preconditioners. An understanding of the underlying mathematical operators and the numerical properties of the discretised version can provide great insight into the optimal choice of preconditioner. For example, a standard finite-element discretisation of the Laplacian operator will, in general, have condition number that is inversely proportion to h^2 for 2D problems and h^3 for 3D problems, where h is the mesh size and the domain is assumed to be uniformly discretised. Thus, halving h will increase the condition number by factors of 4 or 8, respectively. This can cause a dramatic increase in iterations if no preconditioner, or an ineffective preconditioner, is used.

Suppose we have a problem that couples together 3 different variables, then the matrix A will naturally split into a block 3×3 format:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}. \tag{2}$$

For such problems, it will be natural to use a block-diagonal preconditioner of the form:

$$P = \begin{bmatrix} P_{1,1} & 0 & 0 \\ 0 & P_{2,2} & 0 \\ 0 & 0 & P_{3,3} \end{bmatrix}, \tag{3}$$

where the dimension of each block directly tallies with the dimension of the associated block in (2). Alternatively, P could be block upper or lower triangular, or take a constraint preconditioner format:

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & A_{1,3} \\ P_{2,1} & P_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}.$$

For symmetric A, Wathen [16] provides a good overview of block preconditioners and we note that factorizations of constraint preconditioners can be generated implicitly [3]. Constraint preconditioners have been extended for non-symmetric problems in [15] and, due to time restrictions, are not considered for the test case considered in this report.

Here, we consider the SD1D test case [4], which uses BOUT++ to simulate a plasma fluid in one dimension (along the magnetic field) that interacts with a neutral gas fluid. Unlike the nonlinear diffusion example, this test case provides a preconditioner as part of the model in an operator-based manner. SD1D has a number of different cases. For Case-03, the equations for the plasma density n, pressure p and momentum $m_i n V_{||i}$ are evolved:

$$\begin{aligned}
\frac{\partial n}{\partial t} &= -\nabla\cdot\left(\mathbf{b}V_{||}n\right) + S_n - S && (4)\\
\frac{\partial}{\partial t}\left(\frac{3}{2}p\right) &= -\nabla\cdot\mathbf{q} + V_{||}\partial_{||}p + S_p - E - R && (5)\\
\frac{\partial}{\partial t}\left(m_i n V_{||}\right) &= -\nabla\cdot\left(m_i n V_{||}\mathbf{b}V_{||}\right) - \partial_{||}p - F && (6)\\
j_{||} &= 0 \\
T_i &= T_e = \frac{1}{2}\frac{p}{en} \\
\mathbf{q} &= \frac{5}{2}p\mathbf{b}V_{||} - \kappa_{||e}\partial_{||}T_e.
\end{aligned}$$

Which has a conserved energy:

$$\int_V \left[\frac{1}{2}m_i n V_{||i}^2 + \frac{3}{2}p\right] dV.$$

The heat conduction coefficient $\kappa_{||e}$ is a nonlinear function of temperature T_e:

$$\kappa_{||e} = \kappa_0 T_e^{5/2},$$

where κ_0 is a constant. See [4] for further details. Operators are:

$$\partial_{||}f = \mathbf{b}\cdot\nabla f \qquad \nabla_{||}f = \nabla\cdot\left(\mathbf{b}f\right). \tag{7}$$

This nonlinear problem is solved by using CVODE from the SUNDIALS library [9], which uses a Newton method. At the heart of the simulation, a large number of systems of the form (2) are solved, where $A = I - \gamma J$ and J are Jacobian matrices for a computed value of γ : for clarity, we will assume that J has the same block structure as A and $J_{1,j}$ are derived via (4), $J_{2,j}$ are derived via (5), $J_{3,j}$ are derived via (6), $J_{j,1}$ are formed by taking the derivative with respect to n, $J_{j,2}$ are formed by taking the derivative with respect to p and $J_{j,3}$ are formed by taking the derivative with respect to $m_i n V_{||}$. We note that $J_{2,1}$ and $J_{2,2}$ contain terms that include $\partial_{||}^2$.

The preconditioner provided within SD1D takes the following block-diagonal, operator form:

$$P_0 = \begin{bmatrix} I & 0 & 0 \\ 0 & I - \gamma \frac{2}{3} \partial_{||}^2 & 0 \\ 0 & 0 & I \end{bmatrix}. \tag{8}$$

We note that in (5), $\partial_{||}^2$ is applied to $\frac{1}{2} T_e$ and not p, and hence, we propose the following block-diagonal, operator-form preconditioner:

$$P_1 = \begin{bmatrix} I & 0 & 0 \\ 0 & (I - \gamma \frac{1}{3n} \partial_{||}^2) n & 0 \\ 0 & 0 & I \end{bmatrix}. \tag{9}$$

We note that application of the preconditioner P_1 will be more expensive than P_0. Finally, for Case-03, we also try a block lower triangular preconditioner that additionally incorporates the $\partial_{||}$ from (6):

$$P_2 = \begin{bmatrix} I & 0 & 0 \\ 0 & (I - \gamma \frac{1}{3n} \partial_{||}^2) n & 0 \\ 0 & -\gamma \partial_{||} & I \end{bmatrix}. \tag{10}$$

Case-04 from SD1D additionally couples the above plasma equations to a similar set of equations for the neutral gas density, pressure, and parallel momentum. A fixed particle and power source is used here, and a 20% recycling fraction. Exchange of particles, momentum and energy between neutrals and plasma occurs through ionisation, recombination and charge exchange. If we sub-divide A according to the variables that are being evolved, we now have a block 6×6 structure. The provided preconditioner P_0 is now

$$P_0 = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 \\ 0 & I - \gamma \frac{2}{3} \partial_{||}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I - \gamma \frac{2}{3} \partial_{||}^2 & \\ 0 & 0 & 0 & 0 & 0 & I \end{bmatrix}. \tag{11}$$

Using the properties of the neutral, we instead propose

$$P_1 = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 \\ 0 & I - \gamma \frac{2}{3} \partial_{||}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I - \gamma D_n \frac{2}{3} \partial_{||}^2 & \\ 0 & 0 & 0 & 0 & 0 & I \end{bmatrix}, \tag{12}$$

where D_n is a diffusion term. As for Case-03, we also propose a block lower-triangular preconditioner that incorporates the $\partial_{||}$ from (6) and the $\partial_{||}$ from the corresponding equation for neutral momentum:

$$P_2 = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 \\ 0 & I - \gamma \frac{2}{3} \partial_{||}^2 & 0 & 0 & 0 & 0 \\ 0 & -\gamma \partial_{||} & I & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I - \gamma D_n \frac{2}{3} \partial_{||}^2 & \\ 0 & 0 & 0 & 0 & -\gamma \partial_{||} & I \end{bmatrix}. \tag{13}$$

Finally, we consider a preconditioner that is identical to P_2 but the $\partial_{||}$ relating to the neutral momentum equa-

tion is dropped:

$$P_3 = \begin{bmatrix} I & 0 & 0 & 0 & 0 & 0 \\ 0 & I-\gamma\frac{2}{3}\partial_{||}^2 & 0 & 0 & 0 & 0 \\ 0 & -\gamma\partial_{||} & I & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I-\gamma D_n\frac{2}{3}\partial_{||}^2 & \\ 0 & 0 & 0 & 0 & 0 & I \end{bmatrix}. \tag{14}$$

In Figure 9, we compare the number of times the time derivatives (right-hand sides) were computed for Case-03 and Case-04 for the different preconditioners. We also compare what happens if no preconditioner is used for the smaller problems. The smaller matrix orders are from the default problem set-up (i.e., 200 mesh points for each variable). For Case-03, we also compare the preconditioners for the discretization with 1000 mesh points per variable and, for Case-04, 400 mesh points per variable. If we consider the smaller version of Case-03, all of the preconditioners reduce the number of right-hand side evaluations by roughly a factor of 36 with P_2 producing the lowest number of evaluations; for the larger problem, we observe that P_2 also has the lowest number of evaluations with an 18% reduction compared to P_0. In Figure 10, we compare the wall clock time to run the simulations. Just one MPI process was used with one OpenMP thread because of the relatively small problem sizes. For the larger version of Case-03, we see a 10% improvement, with respect to wall-clock time, when using the block lower-triangular preconditioner compared to the original preconditioner. These savings are relatively modest because the density n remains close to uniform throughout the simulation.

For Case-04, we observe that all of the preconditioners substantially decrease the number of evaluations and the wall clock time compared to using no preconditioner. For both problem sizes, compared to the original preconditioner, preconditioners P_1, P_2 and P_3 are reducing the number of evaluations and wall clock time by a third. Thus, for the expert user, it is possible to incorporate more sophisticated operator-based preconditioners within BOUT++.

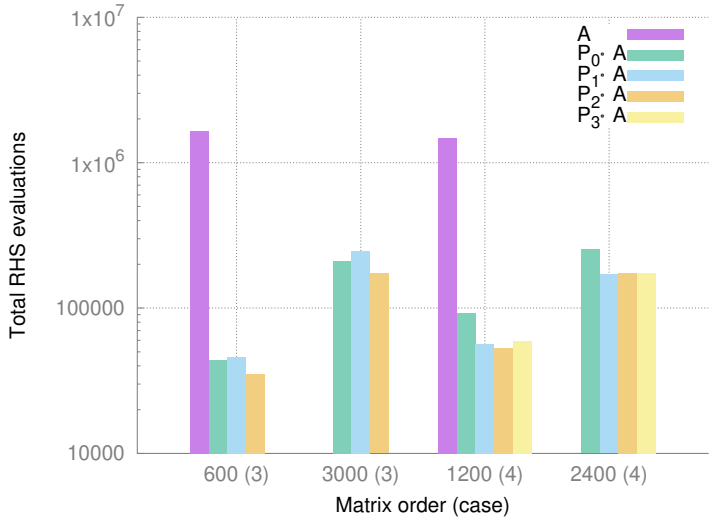

Figure 9: **Total number of right-hand side evaluations performed during the whole simulation for Case-03 and Case-04 with different preconditioners.** For Case-03, results for the default discretization with 200 mesh points per variable (matrix order 600) and a larger problem size with 1000 mesh points per variable (matrix order 3000) are provided. For Case-04, results for the default discretization with 200 mesh points per variable (matrix order 600) and a larger problem size with 1000 mesh points per variable (matrix order 3000) are provided.

5 Proposed roadmap for including new preconditioners within BOUT++ and Nektar++

Prior to including custom new preconditioners into BOUT++ and Nektar++, it would be best to have the exact testing scenarios of interest to UKAEA available and ready. Once the testing scenarios are available and working at scale close to the real problems UKAEA want to solve, it would be the right time to investigate preconditioner deployment. Ideally the new preconditioners need to be reformulated in a matrix-free manner. Once these

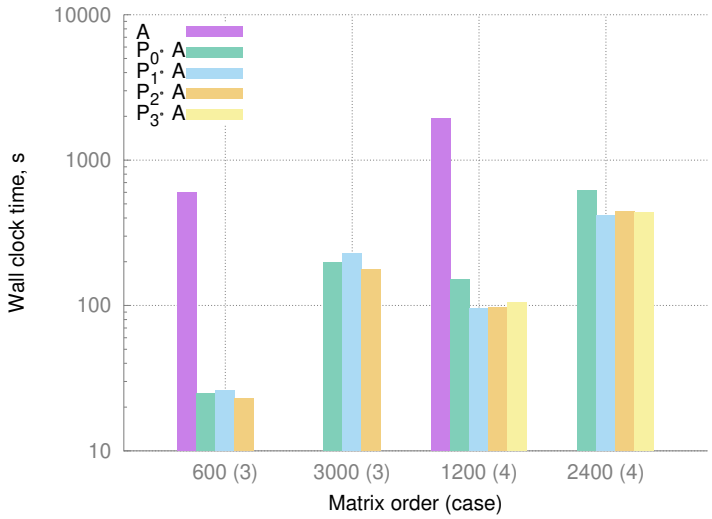

Figure 10: **Total wall clock time (seconds) for the whole simulation for Case-03 and Case-04 with different preconditioners.** For Case-03, results for the default discretization with 200 mesh points per variable (matrix order 600) and a larger problem size with 1000 mesh points per variable (matrix order 3000) are provided. For Case-04, results for the default discretization with 200 mesh points per variable (matrix order 600) and a larger problem size with 1000 mesh points per variable (matrix order 3000) are provided.

formulations are available they could be integrated into PETSc, SUNDIALS and other solver packages powering BOUT++ and Nektar++. This way it would bring more benefit to the user community. The user base of PETSc and SUNDIALS is likely to be much larger than that of BOUT++ and Nektar++. This would also ensure that only minimal changes would be required to BOUT++ and Nektar++ in order to leverage computational advantages of new preconditioners.

In summary the following steps are required for a successful deployment of new preconditioners into BOUT++ and Nektar++:

1. Design testing scenarios in native BOUT++ and Nektar++. These scenarios should reflect real-world problems UKAEA solves at the moment and pay attention to the scale at which simulations should be performed.
2. Identify (matrix-free) methods and underlying solver packages (PVODE, CVODE, PETSc, SUNDIALS) that BOUT++ and Nektar++ employ to solve those problems.
3. Reformulate MCMCMI-based preconditioner in the operator (matrix-free) form.
4. Implement MCMCMI preconditioner in operator form and test it thoroughly.
5. Integrate operator-based MCMCMI preconditioner into solver packages identified in Step 2.
6. Solve testing scenarios identified in Step 1 using standard (native) BOUT++ and Nektar++ methods as well as new MCMCMI preconditioners.
7. Compare computational performance of native vs MCMCMI-based preconditioners.

6 Conclusion

Including MCMCMI-based preconditioners into either BOUT++ or Nektar++ simulation software is difficult at the moment. First, the MCMCMI code needs to be reformulated in the matrix-free manner. Then it can be integrated into solver packages (PETSc, SUNDIALS) powering solving abilities of BOUT++ and Nektar++. Reformulation and implementation of MCMCMI in the operator form will take 3–6 months. Integration of operator-based MCMCMI into PETSc and SUNDIALS will take another 3–6 months. *Therefore, to obtain preliminary results of MCMCMI performance over the test problems the STFC team decided to extract the system matrices from the relevant BOUT++ and Nektar++ testing scenarios.* In the process the team discovered (i) it was not possible to extract the matrices for some scenarios due to their inherent matrix-free nature, (ii) some

testing scenarios of interest to UKAEA are not available yet in either BOUT++ or Nektar++. These need to be designed and developed further.

Acknowledgements

The team would like to thank Dr. Benjamin Dudson from the University of York and Dr. Chris Cantwell from the Imperial College London for their time, help and guidance in technical aspects of BOUT++ and Nektar++ functionality.

References

1. Alexandrov, V., Lebedev, A., *et al. Linear systems of equations and preconditioners relating to the NEPTUNE Programme: a brief overview* tech. rep. 2047353-TN-02 (UKAEA, 2021).
2. Cantwell, C. D., Moxey, D., *et al.* Nektar++: An open-source spectral/hp element framework. *Computer physics communications* **192,** 205–219 (2015).
3. Dollar, H. S., Gould, N. I., *et al.* Implicit-factorization preconditioning and iterative solvers for regularized saddle-point systems. *SIAM Journal on Matrix Analysis and Applications* **28,** 170–189 (2006).
4. Dudson, B. *SD1D* online repository. 2016. `https://github.com/boutproject/SD1D` (2021).
5. Dudson, B., Hill, P., *et al. BOUT++* online repository. Mar. 1, 2020. `http://boutproject.github.io` (2021).
6. Gardner, D. J., Reynolds, D. R., *et al.* Enabling new flexibility in the SUNDIALS suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)* (2022).
7. Hindmarsh, A. C., Brown, P. N., *et al.* SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)* **31,** 363–396 (2005).
8. Lebedev, A. & Alexandrov, V. *On Advanced Monte Carlo Methods for Linear Algebra on Advanced Accelerator Architectures* in *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)* (2018), 81–90.
9. S. Woodward, C., R. Reynolds, D., *et al. SUNDIALS: SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers* online. Lawrence Livermore National Laboratory, 2023. `https://computing.llnl.gov/projects/sundials`.
10. Sahin, E., Lebedev, A., *et al. Usability of Markov Chain Monte Carlo Preconditioners in Practical Problems* in *2021 12th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)* (2021), 44–49.
11. Şahin, M. E., Lebedev, A., *et al.* Empirical Analysis of Stochastic Methods of Linear Algebra. *Computational Science–ICCS 2020* **12143,** 539.
12. Sherwin, S., Kirby, M., *et al. Nektar++* online. Jan. 21, 2021. `https://www.nektar.info` (2021).
13. Team, P. D. *PETSc Users Manual* tech. rep. ANL-95/11 - Revision 3.14 (Argonne National Laboratory, 2020). `https://www.mcs.anl.gov/petsc`.
14. Team, P. D. *PETSc Web page* 2019. `https://www.mcs.anl.gov/petsc`.
15. Thorne, S. *Implicit-factorization preconditioners for non-symmetric problems* tech. rep. 2047353-TN-04 (UKAEA, 2021).
16. Wathen, A. Preconditioning. *Acta Numerica* **24,** 329–376 (2015).