

# Preconditioning in parallel for fractional step Navier–Stokes solvers

V. S. Djanali<sup>1</sup>S. W. Armfield<sup>2</sup>M. P. Kirkpatrick<sup>3</sup>S. Norris<sup>4</sup>

(Received 9 January 2012; revised 3 April 2012)

## Abstract

Preconditioning for the Pressure Poisson Equation, used with the fractional step Navier–Stokes solvers, is studied. The Pressure Poisson Equation results from the segregated calculation of the velocity and pressure in the momentum equations, with the divergence of the velocity as the source term. The coefficient matrix of the Pressure Poisson Equation is dependent only upon grid-size, and thus preconditioners need to be constructed only once initially, and used for all subsequent time steps. Several preconditioning techniques are studied, including Jacobi, incomplete matrix decomposition variants, and sparse approximate inverses. The test case is a three dimensional turbulent channel flow, with the domain discretised using a structured nonstaggered grid. Parallel computing is performed on a cluster of processors by message passing, with domain partitioning to avoid the use of global gather

---

<http://journal.austms.org.au/ojs/index.php/ANZIAMJ/article/view/5097> gives this article, © Austral. Mathematical Soc. 2012. Published April 18, 2012. ISSN 1446-8735. (Print two pages per sheet of paper.) Copies of this article must not be made otherwise available on the internet; instead link directly to this URL for this article.

and scatter operations and to minimise the effect of communication bandwidth. The preconditioners are all constructed based on the local grid partition. For the sparse approximate inverse preconditioners, cell dependencies are bounded to limit communication across grid partition boundaries. The effect of a defined sparsity pattern is also investigated. The optimum sparsity pattern and the dependency on the neighbouring cells are found to be influenced by the grid ratios in each axis direction.

## Contents

<b>1</b>	<b>Introduction</b>	<b>C20</b>
<b>2</b>	<b>Preconditioning in fractional step</b>	<b>C21</b>
<b>3</b>	<b>Numerical methods</b>	<b>C23</b>
<b>4</b>	<b>Results</b>	<b>C25</b>
<b>5</b>	<b>Conclusions</b>	<b>C29</b>
	<b>References</b>	<b>C31</b>

## 1 Introduction

The fractional-step method solves the Navier–Stokes equations in a segregated manner, with each momentum equation and a separate pressure equation solved sequentially. The pressure equation is formed as a Poisson equation with the divergence of the velocity as a source term, and is often the most time consuming part in the calculation. To accelerate the convergence of the solver, preconditioning is applied for the pressure equation. Many preconditioning techniques have been developed, and were overviewed by Benzi [1] and Saad [2].

The performance of the preconditioner will generally depend on the case tested, and thus the best preconditioner for a case may fail on another case [2]. This study is a continuation of previous work [3]. Whereas Djanali, Armfield and Kirkpatrick [3] performed the computations sequentially in a single processor, here the work is extended to parallel computing. Several preconditioning approaches are tested, including Jacobi, incomplete matrix decomposition and sparse approximate inverses, and the performances are compared.

The effect of a prescribed sparsity pattern for sparse approximate inverses is also investigated. The sparsity pattern is determined based upon dependencies to neighbouring cells in the finite volume discretisation, in a similar manner to that recently being used on Graphical Processor Units (GPUs) [4, 5]. Our study shows that a sparse approximate inverse with a customised sparsity pattern has a comparable performance to the original arbitrary-sparse approximate inverse, and this performance may also contribute to the development of preconditioning methods on GPUs.

## 2 Preconditioning in fractional step

In the fractional step method, the pressure and velocity terms in the Navier–Stokes equations are decoupled using a divergence-free velocity constraint. This results in two separate equations, one involving momentum terms solved for the velocity and the other for the pressure. Various forms of fractional-step schemes have been developed to achieve better efficiency and time accuracy; Armfield and Street [6] overviewed and compared some schemes. Application of the fractional-step Navier–Stokes method with second order pressure correction (P2) and negligible body force, using Adams–Bashforth and Crank–Nicolson methods for the time discretisation of the advective and

diffusive terms respectively, results in the equations

$$\frac{\mathbf{v}^* - \mathbf{v}^n}{\Delta t} + \left[ \frac{3}{2}\mathbf{H}(\mathbf{v}^n) - \frac{1}{2}\mathbf{H}(\mathbf{v}^{n-1}) \right] = -\mathbf{G}\mathbf{p}^{n-1/2} + \frac{1}{\text{Re}}\mathbf{L}(\mathbf{v}^* + \mathbf{v}^n), \quad (1)$$

$$\mathbf{L}\pi = \frac{1}{\Delta t}\mathbf{D}\mathbf{v}^*, \quad (2)$$

where  $\pi$  is the pressure correction and  $\mathbf{H}$ ,  $\mathbf{G}$ ,  $\mathbf{L}$  and  $\mathbf{D}$  are the advection, gradient, Laplace and divergence operators, respectively. The discrete velocity is  $\mathbf{v}$ , the discrete pressure  $\mathbf{p}$  and the time level  $\mathbf{n}$ . The intermediate velocity field,  $\mathbf{v}^*$ , is not necessarily divergence-free. Equation (2) is termed the Pressure Poisson Equation (PPE). A correction to  $\mathbf{v}^*$  is then applied using the gradient of  $\pi$  to give a divergence-free velocity,  $\mathbf{v}^{n+1}$ , while  $\pi$  provides an update for the pressure, that is,

$$\mathbf{v}^{n+1} = \mathbf{v}^* - \Delta t\mathbf{G}\pi \quad \text{and} \quad \mathbf{p}^{n+1/2} = \mathbf{p}^{n-1/2} + \pi. \quad (3)$$

Computing the solution of the PPE is often the most time consuming part of the process, particularly in high accuracy simulations [6]. Thus, accelerating the convergence of the PPE will significantly affect the overall efficiency of the Navier–Stokes solver. Since the PPE has a constant coefficient matrix for a given grid spacing, the preconditioner needs to be constructed only once and is then recalled in the subsequent iterations.

Preconditioning techniques are generally classified into two types. The first type is a preconditioning matrix  $\mathbf{M}$  that approximates the original coefficient matrix  $\mathbf{A}$ , or  $\mathbf{M} \approx \mathbf{A}$ . Examples of this class are Jacobi, Gauss–Seidel, Successive Over Relaxation (SOR) and various forms of Incomplete Lower-Upper (ILU) preconditioners. While simple methods like Jacobi, Gauss–Seidel and SOR are often not effective, ILU-variant preconditioners greatly improve the rate of convergence [2]. The Strongly Implicit Procedure (SIP) [7] and Modified Strongly Implicit (MSI) [8] are further development of the ILU-variants specifically on a finite difference grid. The ILU-variants are easy to construct but the implementation requires forward-backward calculations, which means they do not parallelise well. The second type of preconditioner

is the sparse approximate inverse, in which the preconditioner approximates the inverse of the original matrix,  $\mathbf{M} \approx \mathbf{A}^{-1}$ . Several studies developed sparse approximate preconditioners [1]. The most popular are the Sparse Approximate Inverse (SPAI) method [9], which is based on the minimisation of the Frobenius norm, and the factorised sparse Approximate Inverse (AINV) method [10], which is based on the biconjugation algorithm. Both SPAI and AINV use control parameters to limit the number of fills in the preconditioning matrix  $\mathbf{M}$ , while the pattern is left arbitrary. The sparse approximate inverses are expensive in construction but require only matrix-vector products in the implementation. The ease of implementation in the solver encourages the use of sparse approximate inverses, particularly with GPU processing. To further simplify the computation and reduce memory storage, recent studies on GPUs determined the sparsity pattern of the inverses should be dependent only upon the neighbouring cells, with a seven point stencil in three dimensions [4, 5]. This means in the matrix  $\mathbf{M}$  each row contains only seven fills at columns corresponding to the location of the neighbouring cells. In this study the pattern of the sparsity is extended to include more than seven cell dependency with various location of cells.

### 3 Numerical methods

A three dimensional turbulent channel flow is used as a test case, for a Reynolds number of 180 based on the turbulent friction velocity  $u_\tau$  and channel half-height  $h$ . The channel box is  $2h$  in the wall normal ( $y$ ) direction,  $6.3h$  in the streamwise ( $x$ ) direction, and  $3.15h$  in the spanwise ( $z$ ) direction. Wall boundaries are set to no-slip, while the streamwise and spanwise boundaries are set to periodic. To investigate the effect of a prescribed sparsity pattern on the approximate inverses, two structured grids are tested. The first grid contains  $75 \times 125 \times 75$  cells in the  $x$ ,  $y$  and  $z$  directions, respectively, with logarithmic grid spacing used in the  $y$  direction and uniform grid spacing used in other directions. The minimum grid spacing in the  $y$  direction is

$\Delta y = 5.17 \times 10^{-4}$  with a stretching factor of 1.07. The time step used for the first grid is  $10^{-4}$ . The second grid is discretised uniformly in all directions, having  $284 \times 90 \times 140$  cells in the  $x$ ,  $y$  and  $z$  directions, respectively. The second grid has a the time step of  $10^{-5}$ .

Problems are solved using the P2 pressure correction method. The time discretisation uses Adams–Bashforth for the advective terms and Crank–Nicolson for the diffusive terms. Variables are stored in a nonstaggered (collocated) grid, with spatial central differencing. The iterative solvers used are Jacobi for the momentum equations, and BICGSTAB with left preconditioning for the PPE equation. The iterative solvers are considered converged when the  $L_2$  norm of the residuals becomes less than  $10^{-6}$ . Preconditioners tested are Jacobi, SOR, SIP, MSI, SPAI, AINV and approximate inverse with defined sparsity.

The computations were run on eight cores on two four-core Intel<sup>®</sup> Core<sup>™</sup> i7 920 2.67 GHz machines. Parallelisation is implemented using the Message Passing Interface (MPI), with the domain partitioned into eight uniform slices in the streamwise ( $x$ ) direction. Each slice boundary contains one ghost cell in the  $x$  direction, in which the value is copied from the next slice boundary value. No actual matrix is constructed, instead the coefficients of the system and scalar variables are stored in compact storage based on the finite volume discretisation.

The sparse approximate inverse preconditioners need care. These preconditioners are built locally in each processor to avoid global gather/scatter operations and minimise the effect of the communication bandwidth, resulting in eight  $M_p$  preconditioner matrices, where  $p = 1, 2, \dots, 8$ . This implies that node dependencies in local matrix  $M_p$  are now bounded in each partitioned slice. For SPAI and AINV, the preconditioning matrix is stored in Compressed Sparse Row (CSR) format to allow arbitrary fills inside each domain slice in the matrix. This format further requires PACK and/or RESHAPE functions to reformat scalar variables, from compact storage to a single vector, in matrix-vector operations. These functions account for about 16% of the

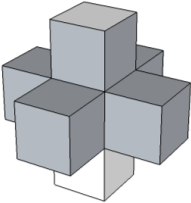
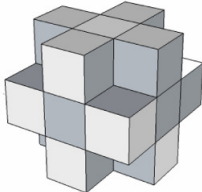
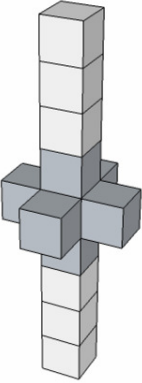
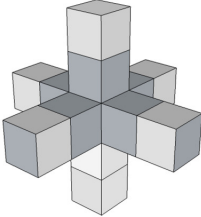
pressure solver time. Reducing the reformating time is the reason for the use of a defined sparsity pattern. With defined sparsity, the approximate inverse need not be stored in CSR format, instead it can be stored in compact storage based on finite differencing, similar to the way the scalar variables are stored. Because the locations of cell dependencies (or fills in a row, in terms of a matrix) are known, the coefficients are stored by indexed value, with indices referring to the positions of the neighbouring cells relative to the point cell. In this study, extended defined sparsity patterns are investigated. Table 1 shows different configurations of the sparsity patterns. All patterns are extended from the basic seven point stencil, and the construction is based on the minimisation of the Frobenius norm.

## 4 Results

Statistical data for the turbulent channel flow has been obtained. Figure 1 shows the time-averaged velocity and velocity fluctuations for the two grids tested, compared to the spectral method of Moser, Kim and Mansour [11]. Previous study showed that preconditioner choice has no effect on the accuracy of the solution with the same convergence criteria [3], and thus the velocity profiles for each preconditioner are not shown here. The stretched grid with sufficiently fine grid spacing in the near wall region shows a good agreement with the reference. Although the uniform grid shows some discrepancy compared to the reference, due to the unresolved near wall region, this result is considered acceptable since the use of the uniform grid is mainly to compare the effect of the sparsity pattern with the approximate inverse preconditioners.

The effectiveness of preconditioners is compared in terms of the computing time and number of iterations needed in the pressure solver to achieve convergence. Figure 2 shows the average pressure solver time per time step for the stretched grid case. The abscissa is the average number of fills in each row of matrix  $M$ , or the average number of cell dependencies for each cell, denoted by  $\alpha$ . Higher  $\alpha$  means a higher number of cell dependencies,

Table 1: Defined sparsity pattern for approximate inverses.

Pattern 1	Pattern 2	Pattern 3
includes one or more neighbour cell/s in all $x^+$ , $x^-$ , $y^+$ , $y^-$ , $z^+$ , $z^-$ directions.	includes one neighbour and four corner cells in all $x^+$ , $x^-$ , $y^+$ , $y^-$ , $z^+$ , $z^-$ directions.	includes one neighbour cell in $x^+$ , $x^-$ , $z^+$ , $z^-$ directions and $(1 + n)$ neighbour cells in $y^+$ , $y^-$ directions, with $n$ the number of extension.
		
7 cells	25 cells	$(7 + 2 * n)$ cells
		
13 cells		

or a denser matrix  $M$ . Jacobi and SOR have  $\alpha = 1$ , and SIP and seven point stencil approximate inverse have  $\alpha = 7$ . In SPAI, the number of fills is controlled by varying the residual of the Frobenius norm, with lower residual producing a denser matrix. Similarly, a drop tolerance is set in AINV to limit the number of fills by omitting entries below the drop parameter. For the stretched grid, it can be seen that preconditioner choice significantly affects the convergence rate of the solver. SIP and MSI are the best preconditioners, with speeds about ten times faster than that of Jacobi. Considering MSI has a higher number of coefficients, SIP is the most efficient preconditioner



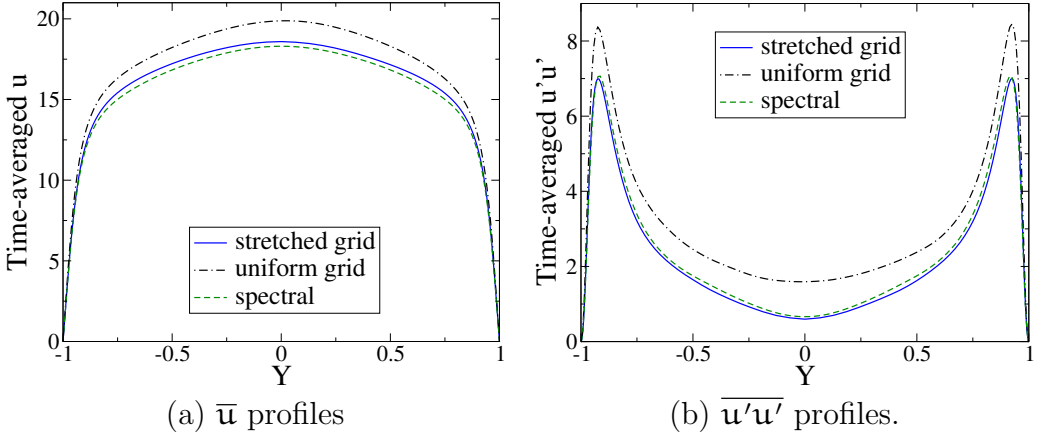


Figure 1: Time averaged velocity profiles compared to DNS spectral method [11].

with lower memory requirement. SOR is better than Jacobi, while SPAI and AINV perform in between Jacobi and SIP with AINV slightly better than SPAI for a denser matrix. The performance of SPAI and AINV may be better than that reported here because the computation time reported includes the reformatting time that accounts for about 16% of the pressure CPU time, although they are still considerably slower than SIP and MSI. Nevertheless, the results presented here may differ from those performed on vector machines or GPUs, in which the benefit of having only matrix-vector operations in the sparse approximate inverse implementation can be fully optimised. Generally, no significant improvement is achieved by varying the number of fills for the sparse approximate inverses. Comparison of defined sparsity patterns shows that Pattern 3, the extended pattern in the  $y$  direction, is proven to be the optimum pattern for this grid configuration with CPU time close to that of the standard arbitrary sparse approximate inverses. This is logical since the grid is finer in the  $y$  direction. The sparse approximate inverse with a higher number of cell dependencies is not necessarily an effective preconditioner; the extended Pattern 1 has a higher computing time even than Jacobi. Figure 3 shows the average number of pressure iterations per

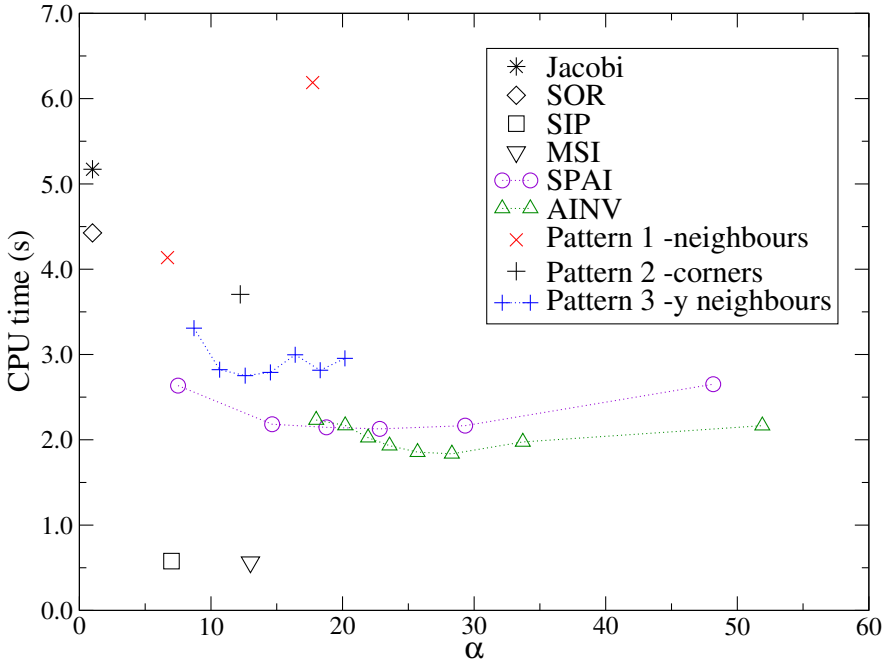


Figure 2: Averaged pressure solver time per time step for stretched grid.

time step. The sparse approximate inverse has a relatively lower number of iterations with a denser matrix  $M$  (a better approximation to  $A^{-1}$ ), but the higher number of cell dependencies results in a slower computing time. However, extended Pattern 1 does not show improvement compared to the basic seven point stencil indicating that increasing the cell dependencies in this pattern does not provide a better matrix  $M$ .

Results for the uniform grid are shown in Figures 4 and 5. Figure 4 shows the average pressure solver time per time step versus  $\alpha$ . As expected, the uniform grid configuration has a lower condition number, and thus the performance has less dependency on the choice of preconditioners. SIP is still the most efficient preconditioner, but Jacobi performs well and may be sufficient to use as an effective preconditioner. SPAI and AINV have higher computation times

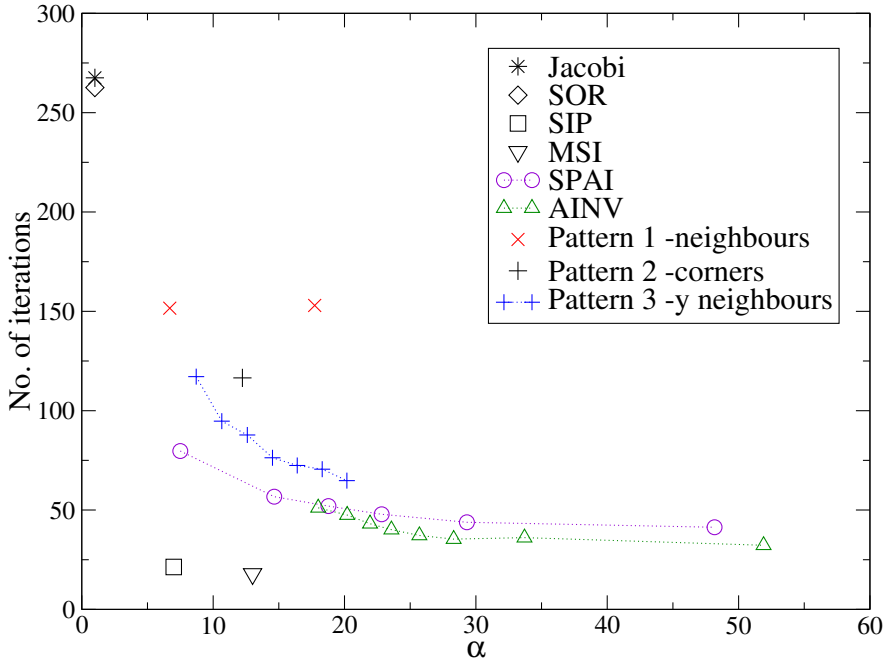


Figure 3: Averaged number of pressure iterations per time step for stretched grid.

compared to Jacobi, although once again these values include reformatting times, which are now increased since the uniform grid has a larger number of cells compared to the stretched grid. For defined sparsity approximate inverses, the basic seven point stencil is the optimum pattern and further extending the pattern is not useful, as shown in Figure 5.

## 5 Conclusions

This study applies preconditioning in parallel for the PPE of the fractional step method using MPI. Preconditioner choice has a considerable effect on the

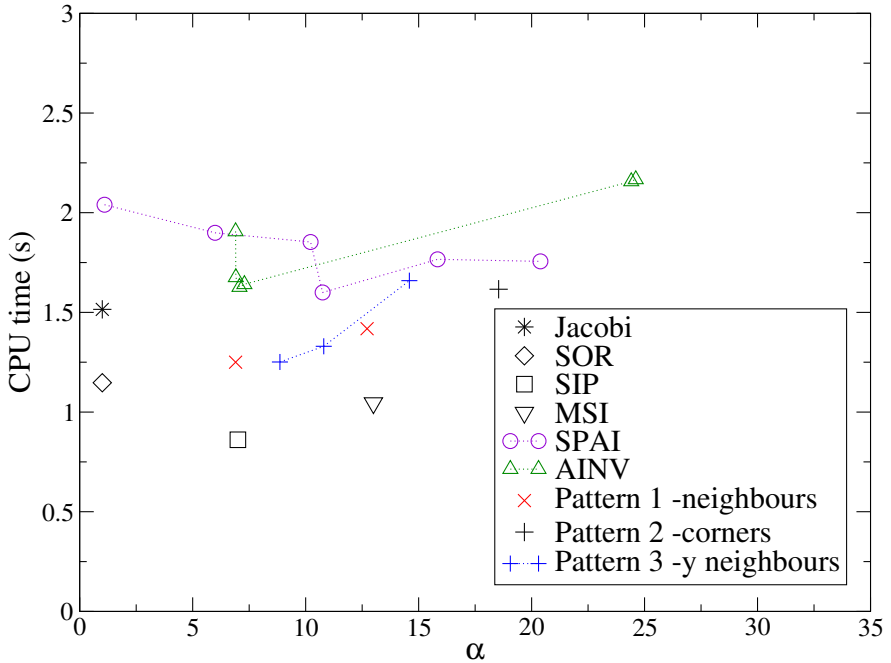


Figure 4: Averaged pressure solver time per time step for uniform grid.

rate of convergence, particularly for difficult problems with high condition numbers. Generally, SIP is the most effective preconditioner tested. However, parallelisation of SIP and MSI implementations can be problematic, while that of sparse approximate inverses is relatively straightforward. Sparse approximate inverses have been shown to perform well, and are expected to perform even better on GPUs, in which the benefit of having only matrix-vector operations in the implementation can be fully optimised. Sparse approximate inverses may be implemented in domain-partitioned parallel computing, although the arbitrary sparsity is bounded in each partition slice. The defined sparsity approximate inverse has comparable performance to SPAI and AINV and is easier to apply with finite volume schemes. The optimum sparsity pattern of the approximate inverse preconditioner is influenced by grid spacing and grid configuration. A simple adaptive sparsity pattern can

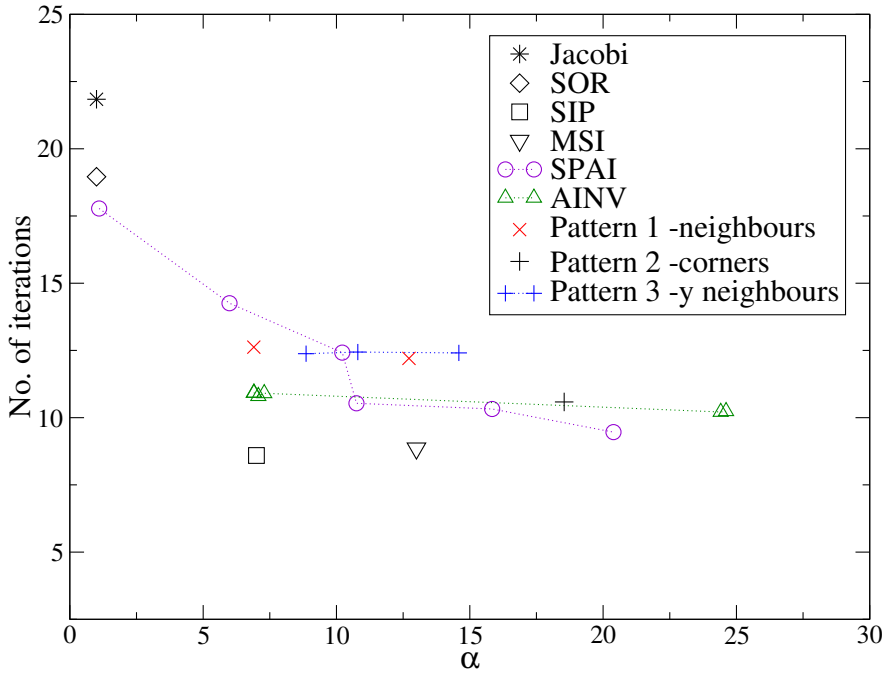


Figure 5: Averaged number of pressure iterations per time step for uniform grid.

be further explored to yield a function of the cell aspect ratios and this may be particularly useful for Poisson preconditioning on GPUs.

## References

- [1] M. Benzi. Preconditioning techniques for large linear systems: A survey. *J. of Comp. Physics*, 182(2):418–477, 2002. doi:10.1006/jcph.2002.7176  
C20, C23

- [2] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA, USA, 2nd edition, 2003. C20, C21, C22
- [3] V. S. Djanali, S. W. Armfield, and M. P. Kirkpatrick. Comparison of approximate inverse preconditioners for the fractional step Navier–Stokes equations. *ANZIAM J.*, 52:C581–C595, 2011. <http://journal.austms.org.au/ojs/index.php/ANZIAMJ/article/view/3890> C21, C25
- [4] M. Ament, G. Knittel, D. Weiskopf, and W. Strasser. A parallel preconditioned conjugate gradient solver for the Poisson problem on a multi-GPU platform. *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference*, 583–592, 2010. doi:10.1109/PDP.2010.51 C21, C23
- [5] S.M. Xu, H.X. Lin, K. Wang, and W. Xue. Utilizing CUDA for preconditioned GMRES solvers. *Proc. 8th Intl Symp. on Distributed Computing and Applications to Business, Engineering and Sciences (DCABES 2009)*. C21, C23
- [6] S. Armfield and R. Street. An analysis and comparison of the time accuracy of fractional step methods for the Navier–Stokes equations on staggered grid. *Int. J. Numer. Methods Fluids*, 38:255–282, 2002. doi:10.1002/fld.217 C21, C22
- [7] H. L. Stone. Iterative solution of implicit approximations of multidimensional partial differential equations. *SIAM J. Numer. Anal.*, 5(3):530–558, 1968. doi:10.1137/0705044 C22
- [8] G. E. Schneider and M. Zedan. A Modified strongly implicit procedure for the numerical solution of field problems. *Numer. Heat Transfer*, 4(1):1–19, 1981. doi:10.1080/01495728108961775 C22
- [9] M. J. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, 18(3):838–853, May 1997. doi:10.1137/S1064827594276552 C23

- [10] M. Benzi and M. Tũma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 19(3):968–994, 1998. doi:10.1137/S1064827595294691 C23
- [11] R. D. Moser, J. Kim, and N. N. Mansour. Direct numerical simulation of turbulent channel flow up to  $Re_\tau = 590$ . *Phys. Fluids*, 11(4):943–945, 1999. doi:10.1063/1.869966 C25, C27

## Author addresses

1. **V. S. Djanali**, School of Aerospace, Mechanical and Mechatronic Engineering, The University of Sydney, Sydney, AUSTRALIA.  
[mailto:vivien\\_s@me.its.ac.id](mailto:vivien_s@me.its.ac.id)
2. **S. W. Armfield**, School of Aerospace, Mechanical and Mechatronic Engineering, The University of Sydney, Sydney, AUSTRALIA.
3. **M. P. Kirkpatrick**, School of Aerospace, Mechanical and Mechatronic Engineering, The University of Sydney, Sydney, AUSTRALIA.
4. **S. Norris**, Department of Mechanical Engineering, The University of Auckland, Auckland, NEW ZEALAND.