AIAA AVIATION Forum
August 2-6, 2021, VIRTUAL EVENT
AIAA AVIATION 2021 FORUM

10.2514/6.2021-2747

A Parallel Symmetric Successive Overrelaxation Method for OVERFLOW

Joseph M. Derlaga,* Charles W. Jackson,† Pieter G. Buning,‡
NASA Langley Research Center, Hampton, Virginia, 23681

The block Jacobi symmetric successive overrelaxation (SSOR) algorithm has been reformulated as a parallelized algorithm for the OVERFLOW structured, overset grid, computational fluid dynamics flow solver. Simple changes to the flow solver required to implement the algorithm are discussed. A series of test cases are presented that demonstrate how the addition of implicit overset boundaries has improved the robustness and nonlinear convergence characteristics of the flow solver.

I. Introduction

Parallelization of a linear solver can be a difficult task. Typical stationary iterative solvers, beyond point Jacobi, are not well suited to a direct parallel extension due to the need to carefully orchestrate the parallel communication and iterative behavior between multiple subdomains. Nonstationary methods, such as Krylov subspace methods, are an attractive alternative as they typically only require a parallel matrix-vector product. Preconditioning for Krylov methods is typically accomplished through a modification (or simplified variant) of a stationary linear solver, and may still be treated in a block Jacobi manner, bypassing parallel communication during the preconditioning operation. In either case, the introduction of a block Jacobi phase to a stationary or nonstationary linear solver can be detrimental to convergence.

Due to the flexibility of overset grids and the use of approximate factorization techniques, OVERFLOW has used a divide-and-conquer solver paradigm since its inception; i.e., the linear system is solved in a block Jacobi manner. Due in part to this design, OVERFLOW has maintained a small memory footprint and high computational efficiency on a per iteration basis. However, the approximate factorization techniques suffer from factorization error and difficult to implement implicit boundary conditions. To address factorization errors and improve robustness, a symmetric successive overrelaxation (SSOR) scheme was introduced in Nichols et al. [1]. Recent work by Derlaga et al. [2] has focused on adding implicit boundary conditions and better numerical approximations to improve time to solution. However, all of the linear solvers have remained in a block Jacobi paradigm that results in artificial explicit boundary conditions between overset subdomains that are only updated at the nonlinear residual level.

To address the issue of explicit boundary condtions between subdomains, OVERFLOW has been modified to allow for a parallel linear solve. This has been done in the context of the SSOR solver to build on the previous convergence improvements. The focus of this work is not on creating a parallel algorithm that matches the behavior of the serial SSOR scheme; there is simply too much flexibility in the overset gridding process to make the orchestration of that process practical. Instead, the goal of this work is to implement a form of the data parallel algorithms proposed in Wissink et al. [3] or Lee and Lee [4] that will work well with OVERFLOW's overset grid paradigm and alleviates some of the problems inherent to explicit overset grid boundaries.

The remainder of this paper will describe the basic SSOR solver scheme, introduce the parallel SSOR solver, and compare the proposed algorithm with other versions of the SSOR scheme available in OVERFLOW in a similar vein to the work of Derlaga et al. [2].

II. Background and Motivation

The SSOR scheme is an iterative matrix inversion process that was introduced to OVERFLOW as a more stable linear solver for the upwind flux discretizations, as compared to the approximate factorization schemes, which were primarily focused on the central-flux finite-difference algorithm. More recent work has focused on increasing the convergence rate of the nonlinear solver by adding implicit physical boundary conditions and improved flux linearizations [2] to an 'improved' SSOR (ISSOR) scheme. While that work greatly improved the convergence characteristics of OVERFLOW

*Research Scientist, Computational AeroSciences Branch, 15 Langley Blvd. MS 128, AIAA Member
†Research Scientist, Computational AeroSciences Branch, 15 Langley Blvd. MS 128, AIAA Member
‡Senior Research Scientist, Computational AeroSciences Branch, 15 Langley Blvd. MS 128, AIAA Associate Fellow

2.3, it did nothing to fix issues related to the explicit boundary conditions at overset regions or the artificial explicit boundaries created by block splitting. In the authors' experiences, OVERFLOW can be sensitive to block splitting, often experiencing degraded convergence due to 'bad' or 'unfortunate' grid splitting as grids are automatically decomposed for MPI load balancing. The same modifications to the SSOR scheme detailed in Ref. [2] that have improved nonlinear convergence have also exacerbated problems caused by grid splitting. Faster convergence of the block Jacobi algorithm results in discontinuities between grid blocks due to the lack of communication during the linear solve. These artificial jumps can then destabilize both the linear and nonlinear solvers resulting in divergence. To address these issues, a parallel SSOR (PSSOR) algorithm is proposed that builds on the developments of the ISSOR scheme. The following sections continue with a primer on basic terminology, and the SSOR schemes, and then continue with a discussion of two forms of a parallel SSOR scheme.

III. The SSOR Algorithm

The concern of a linear solver is the solution of an equation of the form $Ax = b$, where A is some matrix, b is a forcing function in the form of a vector, and x is the solution vector. For OVERFLOW, the matrix A is the left-hand side (LHS) Jacobian matrix formed by a nearest neighbor approximation of the nonlinear residual, while the vector b (RHS) contains the full nonlinear residual. The solution, x, is a vector of solution updates in the form of $\Delta Q/J$, where Q represents the conserved variables and J is the determinant of the metric transformation matrix (and the inverse of the cell volume). For convenience, we will only refer to $\Delta Q/J$ as ΔQ from this point. For reference, the nonlinear solution proceeds from an initial guess for the solution Q, through the formation of the nonlinear residual and solution of the linear problem, and then the application of the solution update to the estimated solution and this process is continued until the residual norm has been sufficiently converged.

As discussed in Ref. [1] the SSOR solvers decompose the LHS matrix into a banded diagonal structure based on the three grid directions and premultiply the LHS and RHS by the inverse of the main diagonal. This results in a vector, R, for the RHS and a set of six scaled block diagonals $A_{j/k/l}$ and $C_{j/k/l}$ for the sub- and superdiagonal portions of the LHS, one each for the J, K, and L grid directions. Note that the main block diagonal is assumed to be inverted and applied properly. For a loop over the L, K, and J directions, where J is the planar-Jacobi direction, the SSOR scheme takes the form of

$$\begin{aligned}\Delta Q^{n+1}_{j,k,l} = (1-\Omega)\Delta Q^{n}_{j,k,l} + \Omega(R\ldots \\ - A_j \Delta Q^{n}_{j-1,k,l} - C_j \Delta Q^{n}_{j+1,k,l}\cdots \\ - A_k \Delta Q^{m1}_{j,k-1,l} - C_k \Delta Q^{m2}_{j,k+1,l}\cdots \\ - A_l \Delta Q^{m1}_{j,k,l-1} - C_l \Delta Q^{m2}_{j,k,l+1}),\end{aligned} \tag{1}$$

where n loops from 1 to the total number of forward + backward sweeps and wraps the loops over the L/K/J directions, and $m1 = n+1, m2 = n$, for n = odd and $m1 = n, m2 = n+1$, for n = even, with Ω being the relaxation factor. Depending on the choice of the Jacobi direction, the loops over the J, K, and L directions will change order to place the Jacobi direction as the innermost loop, with an appropriate shift of the $m1$ and $m2$ variables.

Until this point, we have not explained why the SSOR solvers work in a block Jacobi manner. Due to the overset grid-assembly process, there is a distinction in OVERFLOW between 'field' points, which support valid solution data and therefore a residual calculation, and 'receiver' points, which hold solution data that are supplied to them by 'donor' points (which are really field points from another grid block). In all previous versions of OVERFLOW, any receiver points were treated as Dirichlet points within the linear solvers; the rows of both the LHS matrix and RHS vector were set to zero and the main diagonal of the LHS was replaced with ones, effectively solving $\Delta Q = 0$ for those points. As an example, only artificial overset boundaries exist for a split block that does not have any physical boundary conditions. Due to these artificial boundaries, field points on the interior of the block evolve based only on the local approximate linearization contained in the LHS as well as the residual calculation contained in the RHS, and would be artificially constrained by frozen boundary values.

Prior to this work, the only way to damp out the errors caused by the frozen boundaries was by interchanging data at the nonlinear level, creating a convergence bottleneck. Because of the iterative nature of the SSOR algorithm, it is possible to interchange the approximate solution of the linear system after each sweep, resulting in an increase in communication at the linear-solver level that can aid global convergence of the nonlinear problem. Due to the fortuitous use of overlapping grids in a donor-receiver paradigm in OVERFLOW, the parallel interchange of ΔQ and incorporation into the SSOR solver is relatively simple to implement.

A. Parallel SSOR: Algorithm 1

The largest change necessary to incorporate a parallel SSOR solve in OVERFLOW is the creation of a synchronization point for a parallel linear solve, as compared to the previous paradigm where grid blocks would only need to synchronize at the start of each nonlinear step. This required only minor refactoring, and OVERFLOW has been modified so that there is now persistent storage for the Jacobian matrices that form the LHS matrix as well as residual storage for the RHS vector when using the parallel SSOR scheme. The SSOR scheme is initialized before the synchronization point, and it is initially assumed that $\Delta Q = 0$ for all points, which generates a forcing function for the linear solver and a new estimate for ΔQ. Once the linear solve synchronization point is reached, donor-receiver points exchange their ΔQ values and store the exchanged values as the forcing function. Now, instead of 'solving' for $\Delta Q = 0$, receiver points solve for $\Delta Q = `received'\Delta Q$. This happens before each symmetric sweep, and due to the first-order linearization, the effect of the interchange is that receiver points, which are next to valid field points, will contribute off-block values to the linear solver. This results in implicit communication between the donor-receiver points in a lagged manner at the linear solver level, rather than at the nonlinear solver level. The parallel SSOR method is essentially similar to the hybrid method proposed in Ref. [3] or the DP-SGS scheme proposed in Ref. [4], and will be referred to as parallel SSOR (PSSOR) Algorithm 1, or the additive Schwarz PSSOR scheme as the updated ΔQ values are unique to each subdomain. Besides the addition of the persistent memory for the SSOR solver, the previously existing SSOR routines could be resused with minor modifications, and the interchange of the ΔQ values were able to use slightly modified versions of the already existing donor-receiver interchange routines available in OVERFLOW. Essentially, no new code was required to implement the parallel SSOR solver, only minor changes to existing routines and code restructuring to allow for a synchronized linear solve.

for $subdomain \leftarrow 1$ **to** $n_{subdomains}$ **do**
  Form LHS matrix & RHS vector;
  Initialize $\Delta Q = 0$;
end
Linear Solve Synchronization Point;
for $sweep \leftarrow 1$ **to** n_{sweeps} **do**
  for $subdomain \leftarrow 1$ **to** $n_{subdomains}$ **do**
    Interchange ΔQ;
  end
  for $subdomain \leftarrow 1$ **to** $n_{subdomains}$ **do**
    Perform One SSOR Update
  end
end

Algorithm 1: Additive Schwarz PSSOR Algorithm.

B. Parallel SSOR: Algorithm 2

Depending on the order of the reconstruction scheme used for the inviscid flux terms, the amount of grid overlap varies. Only one point of overlap is required for first-order schemes, while three points of overlap are needed for the fifth-order WENO schemes. Because OVERFLOW uses a first-order, nearest neighbor linearization, first-order schemes are limited to PSSOR Algorithm 1, as described above. However, wider stencil operators allow for a second variant of the PSSOR scheme, which results in an overlapping solve. Points, which are not the outermost receiver point in a residual stencil, have the support to form a first-order linearization. These receiver points can receive the forcing function/residual value from their donor points in addition to the ΔQ value. Since the outermost points in the residual stencil do not support a first-order linearization, they only receive a ΔQ value. This creates an overlapping domain solve, a muliplicative Schwarz algorithm, with the overlap resulting in a shifting of the lagged boundary. Furthermore, two variants of PSSOR Algorithm 2 exist; one where ΔQ values are interchanged for all points in the overlap region, which allows for independent evolution during each SSOR sweep but then ties each grid together before the start of the next sweep, and a second where only the outermost points in the stencil receive updated ΔQ values before each sweep, in a manner similar to PSSOR Algorithm 1. This allows receiver points that supported a first-order linearization to maintain the ΔQ values they calculated rather than having them be overwritten. How much data are transferred affects the degree of coupling between overset grids; overwriting all points could potentially increase the transfer of data between subdomains, but only transferring the outermost point could be more efficient from the standpoint of parallel

communication requirements.

```
for subdomain ← 1 to n_subdomains do
    Form LHS matrix & RHS vector;
    Initialize ΔQ = 0;
end
Linear Solve Synchronization Point;
for subdomain ← 1 to n_subdomains do
    Interchange RHS vector;
end
for sweep ← 1 to n_sweeps do
    for subdomain ← 1 to n_subdomains do
        Interchange ΔQ;
    end
    for subdomain ← 1 to n_subdomains do
        Perform One SSOR Update
    end
end
```

Algorithm 2: Multiplicative Schwarz PSSOR Algorithm.

PSSOR Algorithm 2, while implemented, has not yet been studied in detail to determine its stability for practical problems. Instead, the focus of the results section will be on PSSOR Algorithm 1 and how it addresses issues of robustness and improves convergence as compared to other linear solver options.

C. Implementation in OVERFLOW

The PSSOR algorithms described above are currently implemented for the meanflow equations, species convection equations, and 1- and 2-equation turbulence models available in OVERFLOW, and use the same underlying numerics as the ISSOR algorithm. The modifications necessary to implement the PSSOR algorithm in OVERFLOW result in fairly significant changes to the nonlinear solution update behavior. In the block-Jacobi path, the turbulence model, then the species convection equation, and then the meanflow equations are solved in sequential order, with the meanflow being able to use updated values from previously computed quantities e.g., the turbulent eddy viscosity, as it lags behind the other equation sets. In the PSSOR path, the various equations are now updated at the same time across all blocks. While the option still exists to vary the number of symmetric sweeps for the SSOR scheme between each of the equation sets, the PSSOR scheme does not allow for different subdomains to use a different number of sweeps, nor does it allow for 'subcycling,' i.e., taking multiple iterations per time step for the species convection or turbulence model, in order to keep the solution process synchronized. The input files are checked at run-time and appropriate adjustments, with warnings, are made to allow the PSSOR scheme to function if it is selected. As of this writing, users will not notice any change in behavior when using OVERFLOW, except in the case of wall functions, which require an additional update stage that was not previously present and may slightly change results.

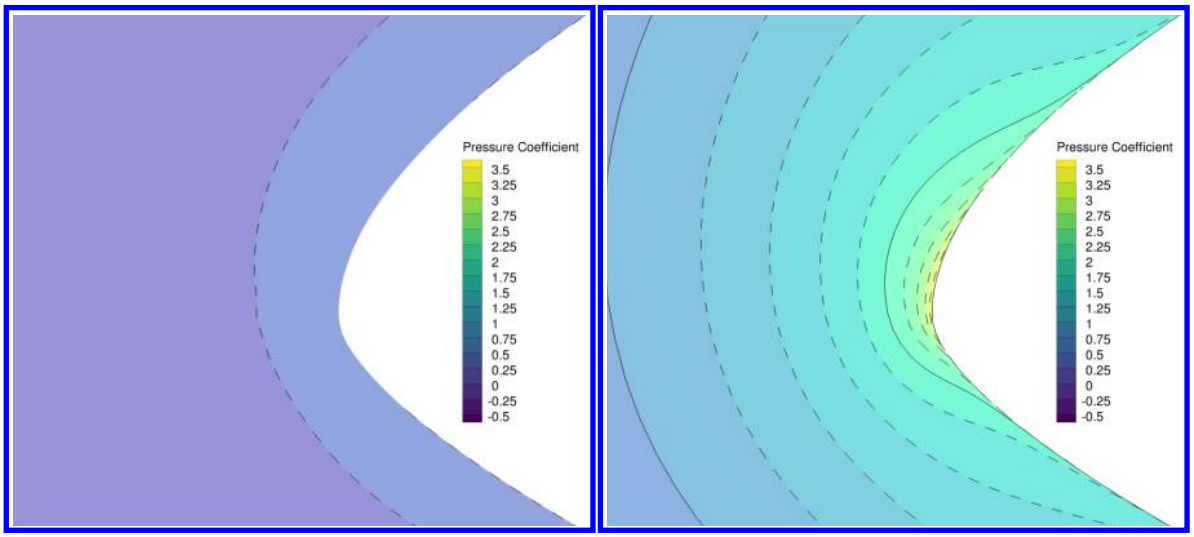
IV. Test Cases

Several test cases are presented below, which compare the PSSOR scheme to the previous study of Derlaga et al. [2]. The intent behind these test cases is to not just demonstrate performance benefits, but to also educate readers on potential issues and solutions to problems that users may experience.

A. S809 Airfoil

To demonstrate the impact of the parallel SSOR solver, a simple 2D, C-grid test case is shown below. One nonlinear step with the original and improved SSOR solvers using 10 symmetric sweeps was taken on a low-speed (Mach=0.1) 2D airfoil without grid splitting, initialized to freestream conditions, with the results shown in Fig. 1. As can be seen, the pressure wave caused by the impulsive start has barely begun to influence the flowfield for the original SSOR scheme, while the improved SSOR scheme introduced in OVERFLOW 2.3 appears to evolve at a much faster rate.

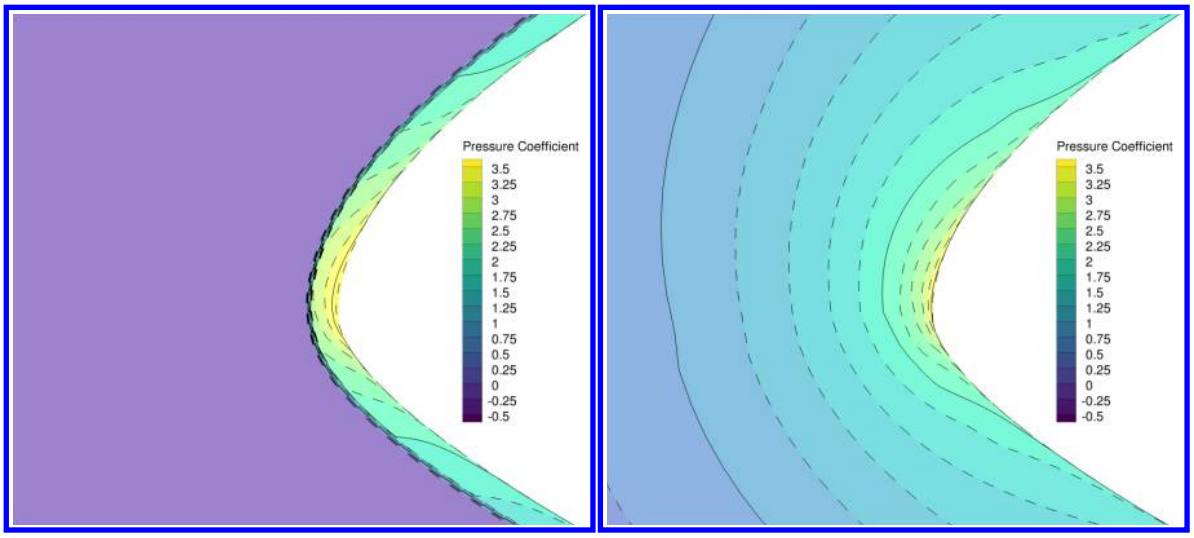
When grid splitting is applied, due to running in parallel with 8 MPI processes, both the original and improved SSOR schemes experience an artificial constraining effect because of the explicit nature of the interblock boundaries, as demonstrated by Fig. 2(a). Not only do grid blocks, which were initially at freestream conditions, remain at freestream



(a) Original SSOR. (b) Improved SSOR.

Fig. 1 One-step solution comparison of original and improved SSOR schemes without block splitting.

conditions, the constraining effect of the artificial boundaries results in an artificially higher pressure peak at the airfoil leading edge, as compared to the run without block splitting. Due to the faster evolution of the improved SSOR scheme, this type of behavior can destabilize the solution process. The result of the PSSOR solver on 8 MPI processes is shown in Fig. 2(b). This solution is a much better match to the serial version of the improved SSOR scheme and does not demonstrate nearly the same level of artificial constraint that the improved SSOR scheme shows with block splitting. While the split grid blocks are not explicitly shown, careful study of Fig. 2(b) does show that there are slight discontinuities that hint at the split grid interfaces, but these are only minor discontinuities. Because there is still a lag caused by the block splitting, the PSSOR solution has not evolved to the same point as the serial improved SSOR case, but the reader is reminded that this is not the goal of the parallel SSOR algorithm. The PSSOR scheme simply attempts to be more implicit across artificial overset boundaries, and clearly succeeds.



(a) Improved SSOR. (b) Parallel SSOR.

Fig. 2 One-step solution comparison of improved and parallel SSOR schemes with block splitting.

For further insight, the case was run with 1, 2, 4, 8, and 16 MPI processes, with the residual histories shown in

Figure 3. Note that for 1 MPI process, both the improved and parallel SSOR scheme converge exactly the same because the grid is not split. As soon as block splitting is introduced with 2 MPI processes, the ISSOR scheme experiences a drastically reduced convergence rate, while the PSSOR scheme is essentially unchanged from the 1 MPI process case, with only a slight reduction in convergence rate as the grid is further decomposed as the number of MPI processes increases.

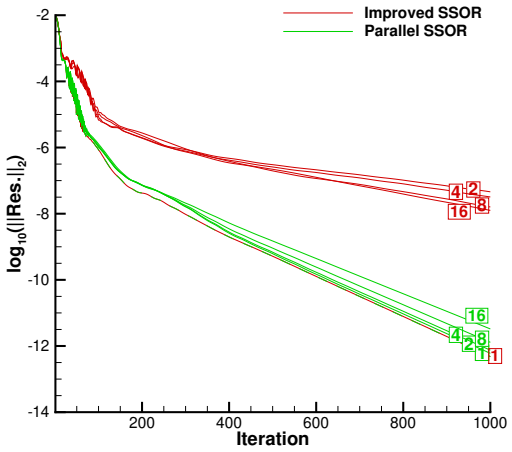


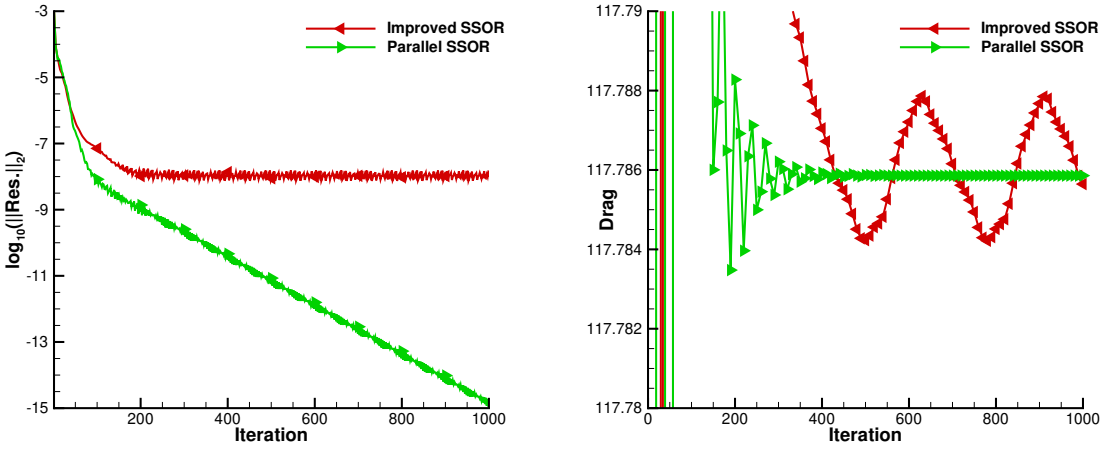
Fig. 3 Comparison of residual convergence for improved and parallel SSOR schemes for a C-grid topology. The number of MPI processes are indicated by the integer symbol.

To further demonstrate the benefits of the PSSOR algorithm, an O-grid topology was run with a partial sweep of MPI processes, and is summarized in Table 1. For an O-grid, the body surface can be split across multiple blocks, so this case more readily demonstrates the interaction of the implicit boundary conditions with grid splitting. A peculiar note is that the improved SSOR scheme converged for 1 and 4 MPI processes but diverged for 2 and 3 processes, victims of 'unfortunate' block splitting. At 8 or more MPI processes, the improved SSOR scheme stagnated. In contrast, for all tested MPI ranks, the parallel SSOR scheme converged without difficulty, demonstrating its increased robustness as compared to the improved SSOR scheme.

Table 1 Convergence across MPI ranks for O-grid test case. Symbols legend: '✓' indicates convergence, '✗' indicates divergence, and '-' indicates stagnated convergence.

	1	2	3	4	5	6	7	8	12	16	20	24
Improved SSOR	✓	✗	✗	✓	✓	✓	✓	-	-	-	-	-
Parallel SSOR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

The convergence behavior for 8 MPI proceses is shown in Figs. 4(a) and 4(b), where the PSSOR scheme converges, while the ISSOR scheme oscillates around the converged PSSOR solution, albeit on the order of a few-thousandths of a drag count. Nonetheless, the PSSOR scheme is convergent across the tested number of MPI processes, an important benefit as compared to the ISSOR scheme. Given that these tests only examine overset communication due to block splitting, the next question to be addressed is whether or not the PSSOR scheme will also improve general overset and dynamic cases.



(a) Residual convergence comparison.

(b) Drag (counts) convergence comparison.

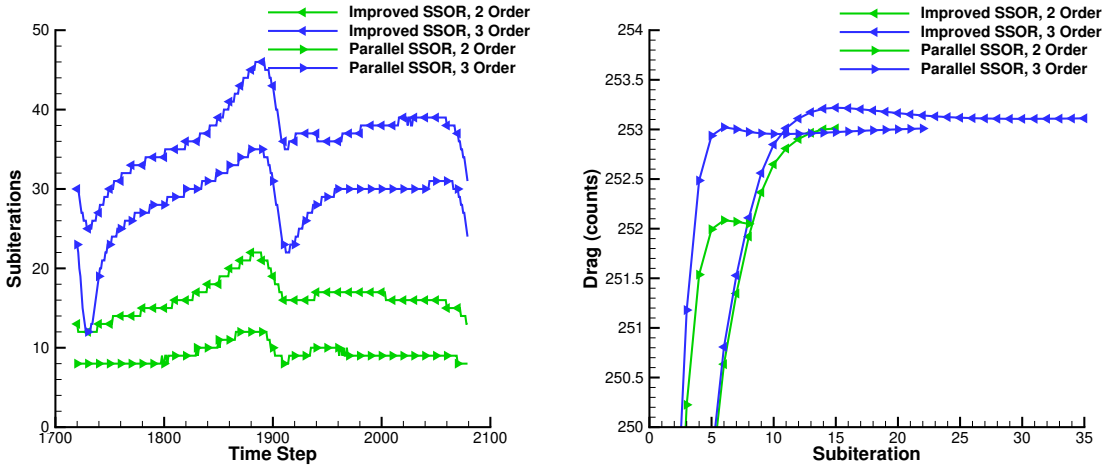
Fig. 4 Solution comparison of improved and parallel SSOR schemes with block splitting for an O-grid topology.

B. NACA 0015 Pitching Airfoil

To further test the ability of the PSSOR scheme to improve convergence, an NACA 0015 airfoil oscillating in pitch [5] between 6.8 and 15.2 degrees at $M = 0.29$ with a chord Reynolds number of 1.95 million has been studied running on 16 MPI processes. The flowfield is initialized by running at 6.8 degrees and then goes through 3 pitch cycles, with results examined on the third pitch cycle. While most of the solution procedure is the same as those in Ref. [2], rather than starting the time accurate iterations from the same set of restart files after two pitch cycles, the linear scheme (ISSOR vs. PSSOR) and subiteration convergence tolerance (2 vs. 3 orders), have been kept constant for each case to demonstrate the effects from starting with different 'base' flows.

Figure 5(a) shows the required number of subiterations to reach both a two and three order-of-magnitude residual drop per time step during the third pitch cycle. For a two order-of-magnitude residual drop, the PSSOR scheme requires approximately 30 – 40% fewer subiterations per timestep than the ISSOR scheme. At a more challenging three order-of-magnitude drop, the PSSOR scheme requires approximately 20% fewer subiterations.

The representative C_D convergence, Fig. 5(b), is from the middle of the down pitch and shows greater variation than the results in Ref. [2]. These differences are due to the different 'base' flows computed by each of the schemes leading up to the third pitch cycle. The two- and three-order convergence PSSOR results show a similar convergence behavior to each other, as do the ISSOR results. The main difference is that the PSSOR scheme plateaus closer to its 'converged' force values faster at each time step as compared to the ISSOR scheme, which demonstrates that the PSSOR scheme speeds up both the residual convergence and force convergence.



(a) Subiterations required during final pitch cycle.

(b) Representative C_D convergence.

Fig. 5 NACA 0015 results.

C. 3rd Drag Prediction Workshop

To test a 3D configuration, the 'Boeing-overflow' grids provided on the 3rd AIAA Drag Prediction Workshop website* were used to study Case W1 [6]. All the settings were reused from the previous work [2], except no grid sequencing was used for the PSSOR scheme. Due to the minimal amount of overlap created by the static grid assembly process the number of fringe points varies on coarser levels, sometimes resulting in no overlap. While OVERFLOW will average the solution to fill in hole points at the nonlinear level, there is no such ability to do so at the linear solver level. As such, it was found that the PSSOR scheme could sometimes be unstable when overset subdomains were found to only be partially connected to the other grids on coarser levels.

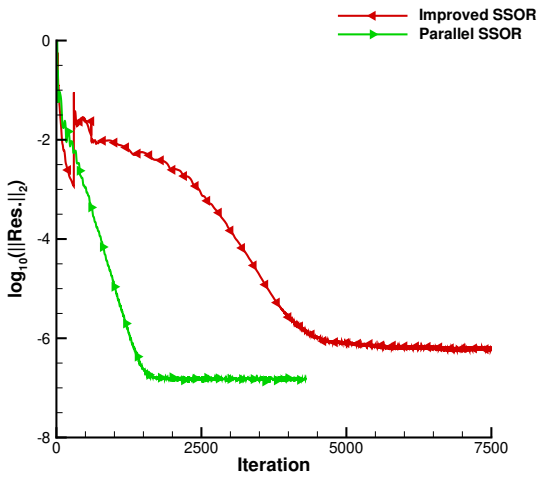
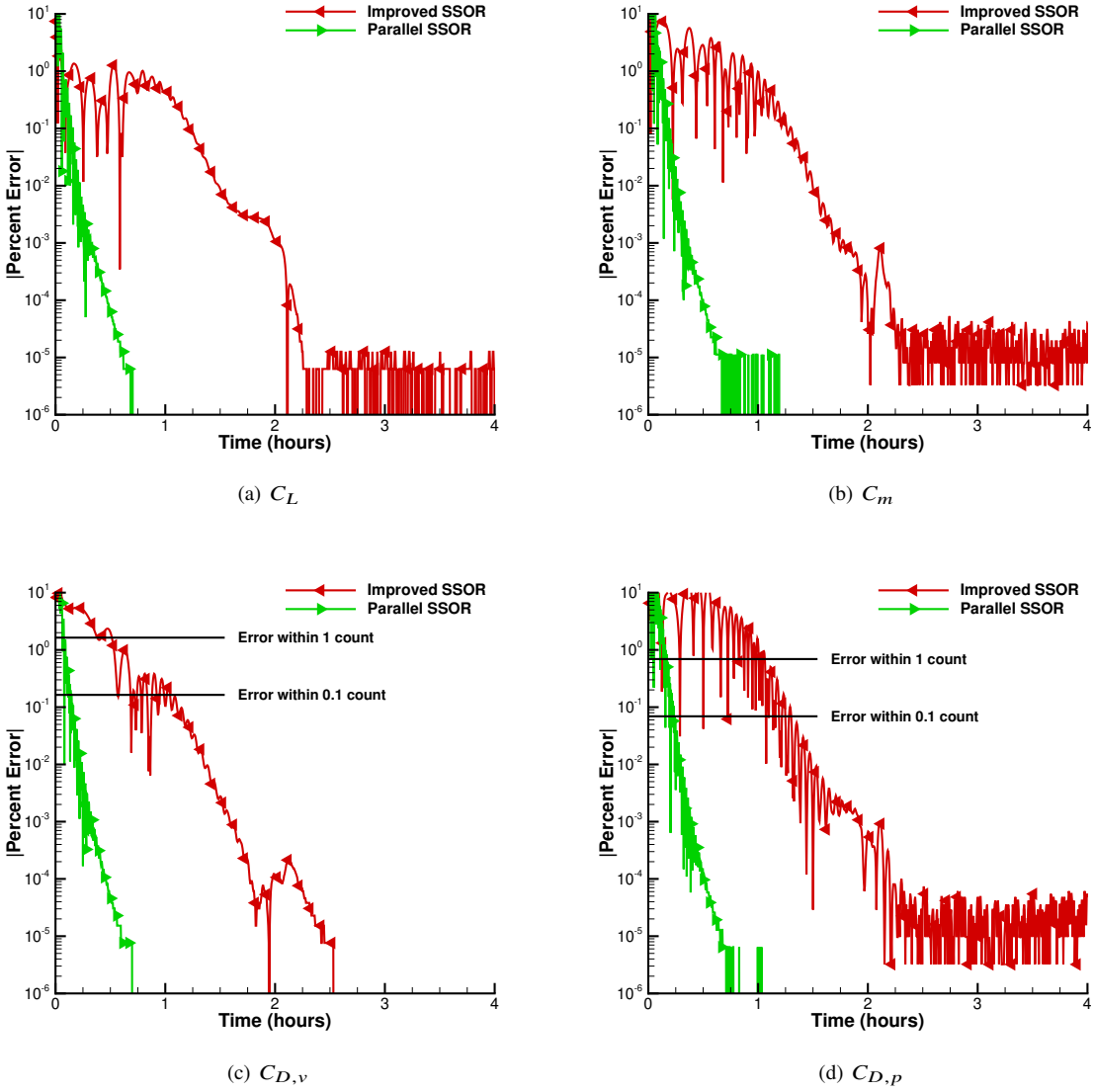


Fig. 6 Comparison of residual convergence for improved and parallel SSOR schemes for DPW3 wing grid.

Despite not having the benefits of grid sequencing to initialize the flowfield on the finest grid level, the PSSOR scheme required approximately a third of the nonlinear iterations, on the finest grid level, to reach a $log_{10}(|Res|_2) = -6$ as compared to the ISSOR scheme, although, unfortunately, neither scheme is fully convergent for this case.

*https://aiaa-dpw.larc.nasa.gov/Workshop3/workshop3.html

Despite the lack of residual convergence, the PSSOR scheme resulted in 'converged' force and moment coefficients that could be used as 'truth' values, whereas the ISSOR scheme required an averaged 'truth' value to compute the relative errors shown in Fig. 7. In general, the use of the PSSOR scheme resulted in halving the time to solution as compared to the ISSOR scheme. On closer examination, the PSSOR scheme produced reasonably converged results in a fraction of the time of the ISSOR scheme, fifteen minutes vs. an hour and fifteen minutes to reach a drag error of a tenth of a count, a worthwhile time savings when, for instance, a large number of cases are needed to complete a test matrix.



(a) C_L (b) C_m

(c) $C_{D,v}$ (d) $C_{D,p}$

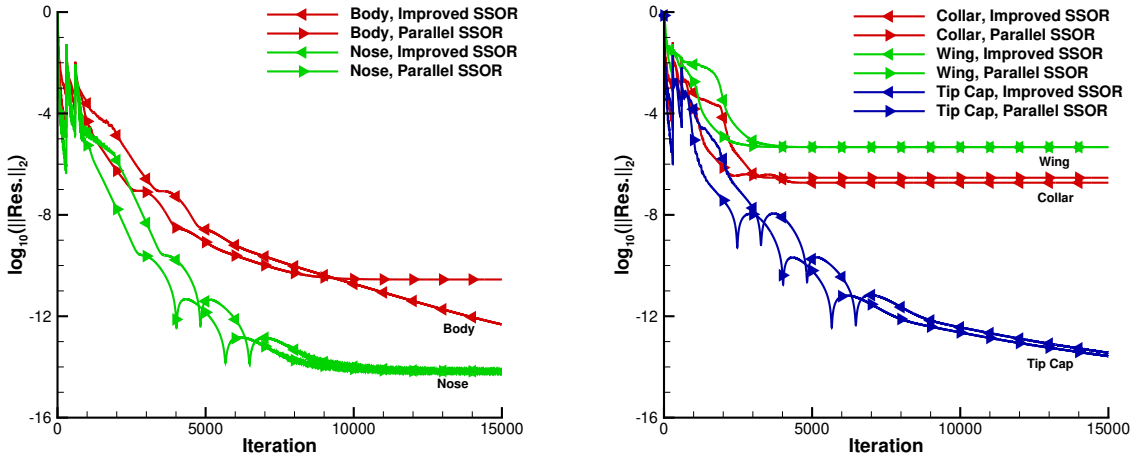
Fig. 7 DPW3 force and moment percent error vs. time.

D. 6th Drag Prediction Workshop

For the final comparison, Case 2A from the 6th AIAA Drag Prediction Workshop [7] was reused with the same settings from the study of Derlaga et al. [2] The PSSOR scheme ran successfully with grid sequencing, indicating that its use should be examined on a case-by-case basis.

For this case, the PSSOR scheme does a superior job of smoothing through the transient behavior. Despite the faster convergence towards steady-state, the PSSOR algorithm exhibits 'bleed-through' to the fuselage grid from the

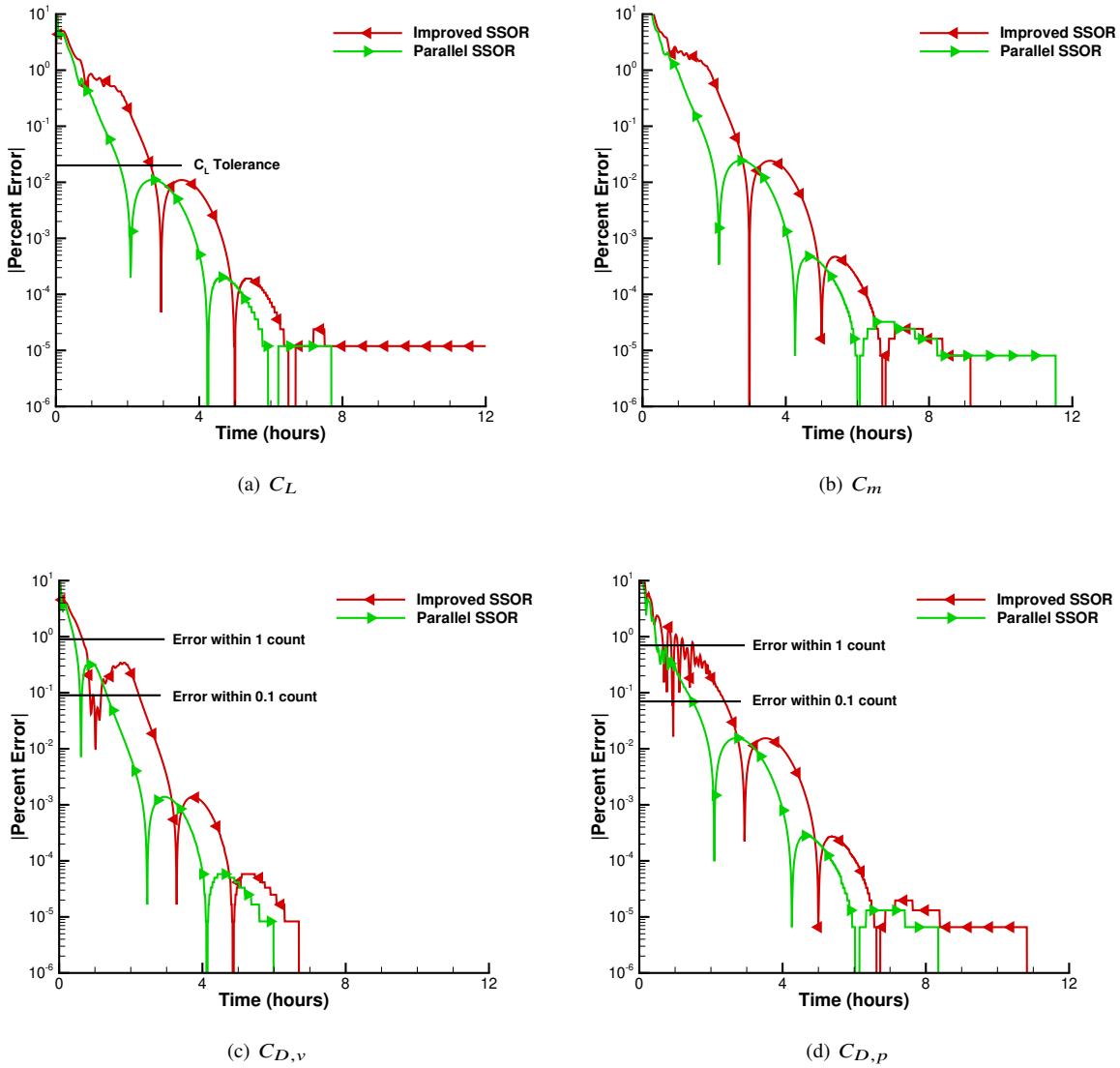
stagnated convergence of the collar grid, as shown in Fig. 8. This is not unexpected, as the PSSOR scheme should result in better communication between grids, but this serves as a reminder that the scheme is not a panacea that will fix all convergence problems. Even so, the better convergence behavior provided by the PSSOR scheme shaves an hour of time to converge drag to within 0.1 count of the final answer, from three hours to two hours, as shown in Fig. 9. Once past the transient behavior, overall convergence of the forces and moments follows the same general trend and the final force and moment values are very similar.



(a) Body Grids.

(b) Wing Grids.

Fig. 8 DPW6 convergence comparison.



(a) C_L (b) C_m

(c) $C_{D,v}$ (d) $C_{D,p}$

Fig. 9 DPW6 force and moment percent error vs. time.

V. Conclusions

A method for parallelizing the SSOR solver in OVERFLOW has been proposed, successfully tested, and demonstrates convergence improvements versus the improved SSOR scheme introduced with OVERFLOW 2.3. Parallelization of the linear solver has generally resulted in improving the convergence of cases where the increased explicitness due to block splitting has resulted in slower convergence. In addition, testing appears to indicate that the parallel SSOR algorithm is less sensitive to instabilities caused by block splitting. Despite mostly positive results, the parallel scheme may fail when grid sequencing is used due to a lack of overset support and underlying grid issues may hinder convergence as well. Ideally, the parallel SSOR algorithm will be tested in the future as a preconditioner for a Krylov scheme in order to more fully couple overset blocks together at the linear solver level, and thereby reduce the number of nonlinear iterations for convergence.

Acknowledgments

The authors would like to thank the NASA Revolutionary Vertical Lift Technology (RVLT) Project of the Advanced Air Vehicles Program (AAVP) for supporting this work. The first author would like to thank Robert 'Bobby' Nichols (The University of Alabama at Birmingham) for his discussions on the SSOR solver; Bobby's suggestions led to starting the work on a parallelized SSOR solver for OVERFLOW. In addition, special thanks go to Brian Allan (LaRC) for testing the parallel SSOR solver. His initial tests for 3D, moving body problems helped prove the effectiveness of this work.

References

[1] Nichols, R. H., Tramel, R. W., and Buning, P. G., "Solver and Turbulence Model Upgrades to OVERFLOW 2 for Unsteady and High-Speed Applications," AIAA Paper 2006–2824, 2006. doi:10.2514/6.2006-2824.

[2] Derlaga, J. M., Jackson, C. W., and Buning, P. G., "Recent Progress in OVERFLOW Convergence Improvements," AIAA Paper 2020–1045, 2020. doi:10.2514/6.2020-1045.

[3] Wissink, A. M., Lyrintzis, A. S., and Strawn, R. C., "Parallelization of a Three-Dimensional Flow Solver of Euler Rotorcraft Aerodynamics Predictions," *AIAA Journal*, Vol. 34, No. 11, 1996, pp. 2276–2283. doi:10.2514/3.13391.

[4] Lee, B., and Lee, D. H., "Data Parallel Symmetric Gauss-Seidel Algorithm for Efficient Distributed Computing using Massively Parallel Supercomputers," AIAA Paper 1997–2138, 1997. doi:10.2514/6.1997-2138.

[5] Piziali, R. A., "2-D and 3-D Oscillating Wing Aerodynamics for a Range of Angles of Attack Including Stall," NASA TM-4632, Ames Research Center, Sep. 1994.

[6] Vassberg, J. C., Tinoco, E., Mani, M., Brodersen, O., Eisfeld, B., Wahls, R., Morrison, J. H., Zickuhr, T., Laflin, K., and Mavriplis, D., "Summary of the Third AIAA CFD Drag Prediction Workshop," AIAA Paper 2007–260, 2007. doi:10.2514/6.2007-260.

[7] Tinoco, E. N., Brodersen, O., Keye, S., Laflin, K., Feltrop, E., Vassberg, J., Mani, M., Rider, B., Wahls, R., Morrison, J., Hue, D., Gariepy, M., Roy, C., Mavriplis, D., and Murayama, M., "Summary of Data from the Sixth AIAA CFD Drag Prediction Workshop: CRM Cases 2 to 5," AIAA Paper 2017–1208, 2017. doi:10.2514/6.2017-1208.