ARTICLE TYPE

ILU Smoothers for AMG with Scaled Triangular Factors

Stephen Thomas¹ | Arielle K. Carr*² | Paul Mullowney³ | Katarzyna Święrydowicz⁴ | Marcus S. Day⁵

Summary

ILU smoothers are effective in the algebraic multigrid (AMG) V-cycle for reducing high-frequency components of the residual error. However, direct triangular solves are comparatively slow on GPUs. Previous work by Chow and Patel¹ and Antz et al.² demonstrated the advantages of Jacobi relaxation as an alternative. Depending on the threshold and fill-level parameters chosen, the factors are highly non-normal and Jacobi is unlikely to converge in a low number of iterations. The Ruiz algorithm applies row or column scaling to U in order to reduce the departure from normality. The inherently sequential solve is replaced with a Richardson iteration. There are several advantages beyond the lower compute time. Scaling is performed locally for a diagonal block of the global matrix because it is applied directly to the factor. An ILUT Schur complement smoother maintains a constant GMRES iteration count as the number of MPI ranks increases and thus parallel strong-scaling is improved. The new algorithms are included in hypre, and achieve improved time to solution for several Exascale applications, including the Nalu-Wind and PeleLM³ pressure solvers. For large problem sizes, GMRES+AMG with iterative triangular solves execute at least five times faster than with direct on massively-parallel GPUs.

KEYWORDS: iterative triangular solvers, GPU acceleration, ILU smoother, algebraic multigrid

1 | INTRODUCTION

Incomplete LU factorizations (e.g., ILU(k), ILUT) are employed as smoothers in algebraic multigrid (AMG) methods. However, when the incomplete factors are applied in the AMG solve phase, the requisite direct triangular solves cause a performance bottleneck on massively parallel architectures such as GPUs. Iterative methods can provide an effective and comparatively faster means of approximating the solution, but when the coefficient matrix is highly non-normal or very ill-conditioned, Jacobi may diverge, or converge in few enough iterations to achieve the desired computational speedups.

Fast, accurate solvers are essential to application codes that rely on sophisticated smoothing algorithms for solving the linear systems, and thus it is critical to avoid exacerbating the departure from normality dep(U) = (‖U‖₂ − ‖D‖₂)ᵖ₂ of the triangular factors, where D contains the eigenvalues, produced when using an ILU-type smoother. We mitigate the effects of problematic L and U factors from A = LU that inhibit fast convergence of the iteration, and specifically those situation in which L or U exhibits a large departure from normality or a high condition number. By scaling the factors, it becomes possible to use the much faster sparse matrix-vector (SpMV) products appearing in the iteration on GPU architectures, thus achieving substantial acceleration in the AMG solve phase. To our knowledge, scaling of the L or U factors to reduce the condition number and the departure from normality is a novel approach for an ILU smoother in AMG.
While iterative methods can provide an effective and comparatively faster means of approximating the solution, in cases where the coefficient matrix is highly non-normal or very ill-conditioned, Jacobi may diverge, or may not converge in a few enough iterations to achieve the desired computational speedups\(^2\). Consider a linear system of the form \(Ax = b\), where \(A\) is a sparse \(n \times n\) matrix that is (highly) ill-conditioned. For example, the matrices exported from the PeleLM application considered in this paper have singular values spanning a dynamic range of sixteen orders of magnitude; see \(^3\). Such systems arise in “projection” methods for evolving variable-density incompressible and reacting flows in the low Mach flow regime, particularly when using cut-cell approaches to complex geometries (see Prenter et al.\(^4\) and Jomo et al.\(^5\)), where non-covered cells that are cut by the domain boundary can have arbitrarily small volumes and areas\(^3\). Equilibration strategies are well-known algorithms designed to reduce the condition number of \(A\)\(^10\) and therefore improve the accuracy of the solution to the linear system, especially for direct techniques.\(^1\)

Equilibration has received considerably less attention when used in conjunction with AMG. Indeed, equilibration of \(A\) can lead to highly non-normal and ill-conditioned \(L\) and \(U\) when computing the ILU factorization of the scaled \(A\)\(^12\), which complicates using iterative methods, such as Jacobi, to compute the resulting triangular solves when employing an ILU smoother in the AMG \(V\)-cycle. Furthermore, scaling \(A\) directly introduces a problem: It requires the global \(A\), which is distributed across the MPI ranks in parallel. In the case where equilibration is not applied to \(A\), the \(L\) and \(U\) can still be highly non-normal. In particular, allowing excessive fill or imposing a conservative (i.e. small) drop tolerance – choices that are often made because they can improve the accuracy of the ILU factorization – lead to problematic factors.\(^12\) In these situations, approximately solving the triangular systems is rendered ineffective. By directly scaling the factors, their ill-conditioning and departure from normality decreases, thus facilitating Richardson iteration. For the applications considered in this paper, scaling leads to a significant reduction in both the condition number and departure from normality\(^14\). Scaling also avoids blocking, another technique for handling non-normal factors when iterative methods are employed to solve these systems.\(^7\) To achieve fast convergence, block relaxation requires a reordering algorithm such as reverse Cuthill–McKee algorithm (RCM)\(^7\), but this can also increase the computational cost.

This paper is organized as follows. In §2, the AMG method is reviewed, along with smoothing techniques. The departure from normality is defined, and scaling is discussed in §3. In §4, a method for incorporating scaling into the ILU smoother is proposed. An experimental study of the different parameter choices illustrates how problematic factors arise, and how scaling can address these issues. The AMGToolbox from Joubert and Cullum\(^15\), \(^16\) is employed here.\(^7\) In §5, the hypre implementation of AMG considered in this paper is described, a performance model is provided, and results are presented for linear systems from the PeleLM\(^3\) and Nalu-Wind\(^17\), \(^18\) pressure solvers. Finally, in section §6, conclusions are provided and future work is proposed. Table 1 summarizes the notation used throughout this paper.

| Notation                        | Definition                                      |
|--------------------------------|-------------------------------------------------|
| Lowercase letters, e.g. \( v \) | column vector                                   |
| Uppercase letters, e.g. \( V, A, H, D \) | matrix                                          |
| Lowercase letter with two subscripts, e.g. \( a_{ij} \) | element of matrix \( A \) in row \( i \) and column \( j \) |
| Lowercase letter with superscript, i.e. \( x^{(k)} \) | \( k \)–th iteration                             |
| Uppercase \( L \)               | lower triangular matrix                         |
| Uppercase \( U \)               | upper triangular matrices                       |
| \( U_s \)                       | strictly upper triangular matrix                |
| \( L_s \)                       | strictly lower triangular matrix                |
| \( \| A \|_2 \)                  | the 2-norm of \( A \)                          |
| \( \| A \|_F \)                  | the Frobenius norm of \( A \)                  |
| \( \| A \|_{\infty} \)          | the infinity norm of \( A \)                   |
| \( \kappa(A) \)                 | 2-norm condition number of \( A \)             |
| \( dep(B) \)                    | departure from normality of \( B \)            |

\(^{1}\) The instability of the solution to a linear system with an ill-conditioned coefficient matrix is well-known; Stewart\(^13\) Chapter 4.2

\(^{2}\) It will be made clear throughout the text when AMGToolbox is employed.
2  |  BACKGROUND

2.1  |  Algebraic Multigrid

The AMG method solves the linear system $Ax = b$ in two steps: the setup and solve phases. In the setup phase, the method constructs prolongation and restriction operators allowing transfers between coarse and fine grids. The application of these transfer operators leads to a sequence, or hierarchy, of successively lower dimension matrices, denoted $A_k \in \mathbb{C}^{m_k \times m_k}$, $k = 0, 1, \ldots, m$ where $A_k = R_k A_{k-1} P_k$, $m_k < m_{k-1}$, and $m_0 = n$. In the Galerkin formulation of AMG, $P_k$ is a rectangular matrix with dimensions $m_{k-1} \times m_k$ also referred to as the interpolant and $R_k = P_k^T$. Once the transfer operators are determined, the coarse-matrix representations are computed through sparse triple-product matrix-matrix multiplication.

In classical Ruge-Stüben AMG, the strength of connection threshold determines the coarse points: The point $j$ is strongly connected to $i$ if and only if

$$|a_{ij}| \geq \theta \max_{k \neq i} |a_{ik}|,$$

where $\theta$ is the strength of connection threshold, $0 < \theta \leq 1$. The selected coarse points are retained at the next coarser level, and the remaining fine points are dropped. More information on the strong connection threshold to determine the set of coarse grid points as well as how to form the transfer operators are found. Let $C_k$ and $F_k$ be the coarse and fine points selected at level $k$, and let $m_k$ be the number of grid points at level $k$. Then, $m_k = |C_k| + |F_k|$ and $m_{k+1} = |C_k|$. Here, the coarsening is performed row-wise by interpolating between coarse and fine points and generally attempts to fulfill two contradictory criteria. In order to ensure that a chosen interpolation scheme is well-defined and of good quality, some close neighborhood of each fine point must contain a sufficient amount of coarse points to interpolate from. Hence, the set of coarse points must be rich enough. However, the set of coarse points should be sufficiently small in order to achieve a reasonable coarsening rate. Because the size of the linear systems decreases on each coarser level, the interpolation should lead to a reduction of roughly five times the number of non-zeros at each level of the hierarchy compared with the number of non-zeros in the coefficient matrix for the original linear system.

In the solve phase, the method employs the $V$-cycle, which comprises a relaxation (or smoothing) iteration coupled with a coarse grid correction. Beginning at the finest level, the method moves to the next coarser level by first performing a small number of pre-smoothing iterations to the solution of $A_k x_k = b_k$ at the $k^{th}$ level of the $V$-cycle. It then computes the residual $r_k = b_k - A_k x_k$ and applies the restriction operator as $b_{k+1} = R_k r_k$ to move to the next coarser level. This process is repeated until the coarsest level is reached, using a direct solver. Though, an iterative may be used if $A_m$ is singular. Moving from coarse to fine, the method interpolates the solution to the next finer level by the prolongation operation along with a small number of post-smoothing iterations. Here, the updated solution becomes a correction for the previous approximate solution, $x_k = x_k + P_k x_{k+1}$. AMG methods are optimal for certain linear systems, (i.e. constant work per degree of freedom in $A_m$) through complementary error reductions by the smoother and solution corrections propagated from coarser levels. Consult Algorithm 1 for a description of the multigrid $V$-cycle.

2.2  |  Residual Smoothing

The aforementioned residual smoothers are generally defined by an inexpensive iterative method such as Gauss-Seidel, Jacobi, or incomplete factorization. The smoother rapidly reduces high-frequency components of the residual error by approximately solving the system of equations. When the remaining low-frequency error is restricted to a coarser level, it then becomes higher frequency and we can again employ a smoothing technique. The general form of a relaxation scheme for $Ax = b$ is given by

$$M x^{(k+1)} = N x^{(k)} + b$$

where $M = M - N$ is referred to as a matrix splitting. The Gauss-Seidel iteration is based upon the splitting $M = D + L$, and $N = U$, where $L$ is strictly lower and $U$ is upper triangular. The inverse of the matrix $M$ is not formed, but rather direct solvers are typically employed. However, as noted earlier, these are relatively slow on GPU architectures. A polynomial type smoother is derived from the iterative solution of the triangular system for $(D + L)$ in Gauss-Seidel relaxation and then used to solve the linear system, $Ax = b$, with residual $r = b - Ax$, where $D$ is the diagonal of $A$. An alternate formulation is to replace $(D + L)^{-1}$

---

3. While the coarser matrices are technically more dense - as a ratio of non-zeros to the size of the matrix - these matrices have fewer rows and columns.

4. The $V$-cycle is the simplest complete AMG cycle. Other processes may be used in place of the $V$-cycle, such as the $W$- or $F$-cycle.
Algorithm 1 Multigrid single-cycle algorithm ($v = 1$ yields $V$-cycle) for solving $Ax = b$. The hierarchy has $m + 1$ levels.

//Solve $Ax = b$.
Set $x = 0$.
Set $v = 1$ for $V$-cycle.
call Multilevel($A$, $b$, $x$, 0, $v$).

function Multilevel($A_k$, $b$, $x$, $k$, $v$)
    // Solve $A_k x = b$ ($k$ is current grid level)
    // Pre smoothing step
    $x = S_1^k(A_k, b, x)$
    if ($k \neq m$) then
        // $P_k$ is the interpolant of $A_k$
        // $R_k$ is the restrictor of $A_k$
        $r_{k+1} = R_k(b - A_k x)$
        $A_{k+1} = R_k A_k P_k$
        $v = 0$
        for $i = 1 \ldots \mu$ do
            Multilevel($A_{k+1}$, $r_{k+1}$, $v$, $k + 1$, $v$)
        end for
        $x = x + P_k v$
        // Post smoothing step
        $x = S_2^k(A_k, b, x)$
    end if
end function

with $(I + D^{-1} L)^{-1} D^{-1}$ in the preconditioned iteration, and replace the matrix inverse with a truncated Neumann series

$$x^{(k+1)} = x^{(k)} + \sum_{j=0}^{p} (-D^{-1} L)^j D^{-1} r^{(k)}.$$ 

for $p < n$. In practice, the Neumann series converges rapidly (i.e., $p \ll n$) for close to normal matrices where the off-diagonal elements of $L$ decay rapidly to zero. Because the matrix $L$ is a strictly lower triangular, it is nilpotent and the Neumann series is a finite sum.

An ILU smoother is well-suited to handle highly varying matrix coefficients or anisotropic problems and is a generalization of Gauss-Seidel, where the diagonal matrix $D$ represents row scaling of one (or both) of the triangular factors. An ILU factorization is split into symbolic and numeric phases and if the sparsity pattern of the matrix $A$ does not change on the finest level, then the symbolic phase of the factorization is reused. For example, the sparsity pattern of the pressure matrix in the PeleLM model does not change during the fluid time integration when mesh refinement does not occur. This permits re-factorization in the numeric phase in order to save computational time and avoid re-generating storage for the $L$ and $U$. This is only true for the finest level because the coarsening algorithm may change the sparsity pattern on other levels and provides further motivation for a hybrid $V$-cycle with ILU smoothing on the finest level only for computational efficiency.

An ILUT Schur complement smoother maintains a constant Krylov solver iteration count as the number of parallel processes (sub-domains or MPI ranks) increases. The Schur complement system represents the interface degrees of freedom at sub-domain boundaries. These are associated with the column indices corresponding to row indices owned by other MPI ranks in the parallel block partitioning of the global matrix. A single GMRES iteration is employed to solve the Schur complement system for the interface variables, followed by back-substitution for the internal variables. Iterative solves are applied in the latter case. A key observation is that the explicit residual computation $r^{(k)} = b - A x^{(k)}$ is not needed for this single GMRES iteration, resulting in a significant computational cost saving. A hierarchical basis formulation of AMG based on the C-F block matrix partitioning and Schur complements are considered in Chow and Vassilevski. Their proposed method can become expensive as the number of non-zeros in the coefficient matrix increases. A similar increase in cost is observed here, along with an increase in the departure
from normality of $U$, which decreases by limiting the fill level in the ILU factorization. Thus, Richardson iterations are effective, resulting in comparatively inexpensive solution for the internal variables on the GPU.

Following Saad\cite{Saad19} Chapter 14.2, Xu\cite{Xu22} and Falgout et al\cite{Falgout24} in order to derive a Schur complement preconditioner for the partitioned linear system

$$
A \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix},
$$

consider the block $A = LU$ factorization of the coefficient matrix $A$

$$
A = \begin{bmatrix} B & E \\ F & C \end{bmatrix} = \begin{bmatrix} I & 0 \\ FB^{-1} & I \end{bmatrix} \begin{bmatrix} B & E \\ 0 & S \end{bmatrix}
$$

The block matrix $B$ is associated with the diagonal block (sub-domain) of the global matrix distributed across MPI ranks by hypre. The Schur complement is $S = C - FB^{-1}E$, and the reduced system for the interface variables, $y$, is given by

$$
Sy = g - FB^{-1}f
$$

(1)

Then the internal, or local, variables represented by $x$ are obtained by back-substitution according to the expression

$$
x = B^{-1}(f - Ey)
$$

An ILUT Schur complement smoother for one level of the $V$-cycle in hypre is implemented as a single iteration of a GMRES solver for the global interface system \cite{Xu22}. The local systems involving $B^{-1}$ are solved by computing an ILUT factorization of the matrix $B \approx LDU$. Rather than employing a direct triangular solver for these systems, the Richardson iterations described previously in Section 2.3 are employed. It is important to note the residual vector is not required for the Schur complement GMRES solver for a fixed number of iterations. The initial guess is set to $x^{(0)} = 0$ and $r^{(k)} = b$, without a convergence check with the explicit $r^{(1)}$.

The convergence of GMRES+AMG using ILU smoothers is compared with polynomial Gauss-Seidel. In particular, ILU on the finest level is combined with Gauss-Seidel on coarser levels. The choice to apply the ILU smoother on any number of levels – starting from the finest level – is now an option available in hypre\cite{Hypre}. The ILU smoother on all levels and the ILUT Schur complement smoother are evaluated.

### 2.3 Jacobi Iteration

While Jacobi is a residual smoother for AMG, here the method is used to approximately solve the triangular systems associated with an ILU smoother. The notation employed by Antz et al\cite{Antz23} is adopted. The Jacobi iteration for solving an arbitrary linear system $Ax = b$ is often written in the compact form

$$
x^{(k+1)} = Gx^{(k)} + D^{-1}b
$$

(2)

with the regular splitting $A = M - N$, $M = D$ and $N = D - A$. The iteration matrix $G$ is defined as $G = I - D^{-1}A$. Given the preconditioner $M = D$\cite{Antz23}, the non-compact form is given by

$$
x^{(k+1)} = x^{(k)} + D^{-1}(b - A x^{(k)})
$$

(3)

where $D$ is the diagonal part of $A$. For the triangular systems resulting from the ILU factorization $A \approx LU$ (as opposed to the regular splitting $A = D + L + U$), the iteration matrices are denoted $G_L$ and $G_U$ for the lower and upper triangular factors, $L$ and $U$, respectively. Let $D_L$ and $D_U$ be the diagonal parts of the $L$ and $U$ and let $I$ denote the identity matrix. Assume $L$ has a unit diagonal, then

$$
G_L = D_L^{-1}(D_L - L) = I - L,
$$

(4)

$$
G_U = D_U^{-1}(D_U - U) = I - D_U^{-1}U.
$$

(5)

A sufficient condition for the iteration to converge is that the spectral radius of the iteration matrix is less than one. With $G_L$ strictly lower and $G_U$ strictly upper triangular, the spectral radius of both iteration matrices is necessarily zero. Therefore, the iteration converges in the asymptotic sense for any triangular system. However, the convergence (or not) of the method in practice also depends on the departure from normality of the coefficient matrix.
3  |  NON-NORMAL MATRICES AND SCALING

3.1  |  Departure from Normality

A normal matrix $A \in \mathbb{C}^{n \times n}$ satisfies $A^*A = AA^*$, and this property is referred to as normality throughout this paper. Naturally, a non-normal matrix is defined in terms of the difference between $A^*A$ and $AA^*$. For example, one might use

$$\frac{\|A^*A - AA^*\|_F}{\|A^*A\|_F}$$

to indicate the degree of non-normality. In the current paper, Henrici’s definition of the departure from normality of a matrix is employed

$$dep(A) = \sqrt{\|A\|^2_F - \|D\|^2_F},$$

where $D \in \mathbb{C}^{n \times n}$ is the diagonal matrix containing the eigenvalues of $A$. There are several ways to characterize a large departure from normality of a matrix; for example, the definition in (6) generalizes to the distance between the eigenvalues and singular values as

$$dep^2(A) = \sum_i \sigma_i^2(A) - \sum_i |\lambda_i(A)|^2.$$  \hspace{1cm} (7)

Further information on metrics and bounds describing normality of matrices is found in [14, 26–28] and references therein.

In general, $dep(L)$ remains modest, and the ILU factorization computes $L$ such that $L = I + L_s$. Thus, scaling is not applied to $L$. However, the same observations are not true for $U$. In particular, scaling is necessary to produce $U = I + U_s$, with unit diagonal, where $U_s$ is strictly upper triangular. Further, when the number of non-zeros in the factors is limited by larger drop tolerance and smaller fill-in levels, the number of non-zeros in $U_s$ will decrease, and thus $\|U_s\|_F$ will become smaller. At the extreme, when fill-in is not allowed beyond the diagonal, then $U_s = 0$, and

$$dep(U) = \sqrt{\|I + U_s\|^2_F - n} = \sqrt{\|I\|^2_F - n} = 0.$$  \hspace{1cm} (8)

However, restricting the number of non-zeros in the factors this dramatically would produce an ILU factorization that is far too inaccurate to be useful as a smoother. More generally, tuning the ILU parameters results in conservative or small $mnz(U)$ (and thus conservative $mnz(U_s)$), $dep(U)$ cannot grow too large.

The metric defined by (6) (or equivalently, by (7)) motivates an experimental analysis of the parameters provided in section 4.3. It follows directly from the definition of normality that an upper (or lower) triangular matrix cannot be normal unless it is a diagonal matrix (see [23], Lemma 1.13 for a proof). Therefore, some departure from normality is expected in the $L$ and $U$. However, if the departure from normality is too great, the iterations may diverge. In Section 4, an extensive experimental study establishes these connections and bounds on the departure from normality given in (6) when employing the proposed scaling. A brief review of scaling strategies is given below before elaborating on the connection between ILU parameter choices, the departure from normality using the $dep(A)$ metric, and the convergence of the iteration.

3.2  |  Scaling Strategies

Equilibration based on row/column scaling, and row scaling alone are applied to the $U$ factor in an ILU or incomplete LDU factorization. Row and column scaling produces a linear system that has been equilibrated and now takes the form

$$L \ D \ U \ D_c \ x = b$$

and Richardson iteration (introduced in Section 4.4) approximately solves the triangular system with matrix $D_cU D_c$ (in addition to the $L$ factor). Because $D_cU D_c$ has a unit diagonal, the iterations are expressed in terms of a Neumann series, discussed in Section 4.4 Ruiz scales the matrix such that the diagonal entries have magnitude one, and all off-diagonal elements are less than (or equal to) one. The algorithm is provided in Algorithm 2 for ease of reference.

When an incomplete $LDU$ or $LDL^T$ factorization is available, the diagonal matrix $D$ represents row scaling for either the $L$ or $U$ factor or both. Given an ILU factorization, $D$ is written as

$$D = \text{diag}(U),$$

and the scaled $\tilde{U}$ is subsequently defined as

$$\tilde{U} = D^{-1}U$$
Algorithm 2 Ruiz for row and column scaling of a coefficient matrix $A$ and corresponding right hand side $b$.

\[
\begin{aligned}
\text{//Iteratively scale } A \in \mathbb{C}^{n \times n}, b \in \mathbb{C}^n \\
k = 0 \\
\text{Set } A_k = A, b_k = b, D_k = I_n \\
\text{while not converged do} \\
\quad D_r = \text{diag}(\sqrt[n]{\| \tilde{a}_{i,:} \|_\infty} / a_{i,:}) \text{ the row vectors of } A \\
\quad D_c = \text{diag}(\sqrt[n]{\| \tilde{a}_{:,j} \|_\infty} / a_{:,j}) \text{ the col vectors of } A \\
\quad A_{k+1} = D_r^{-1} A_k D_c^{-1} \\
\quad b_{k+1} = D_r b_k \\
\quad D_{k+1} = D_k D_c^{-1} \\
\quad k = k + 1 \\
\text{end while} \\
\text{//After solving } A_k \tilde{x} = b_k, x = D_k^{-1} \tilde{x}
\end{aligned}
\]

To obtain the $LD\tilde{U}$.

In many of the applications highlighted in this paper, Ruiz produces the larger reduction in departure from normality compared with row scaling alone; see Table 6. This may be attributed to the (additional) column scaling in the Ruiz approach. However, in some cases, row scaling leads to a smaller departure. Row scaling reduces $\kappa(U)$ more for a subset of the matrices; see Table 7. Even though symmetry is preserved, it is not guaranteed to provide the minimal condition number compared with other scaling algorithms and it is not easily parallelized. This method also requires access to both the rows and columns of a matrix, which can present problems when storing the matrix in CSR format. It is among several different row and column scaling methods, that also preserves symmetry of the coefficient matrix where necessary.

4 | AN ILU SMOOTHER WITH SCALED FACTORS

A sparse triangular solver is a critical kernel in many scientific computing simulations, and significant efforts have been devoted to improving the performance of a general-purpose sparse triangular solver for GPUs. For example, the traditional parallel algorithm is based upon level-set scheduling, derived from the sparsity structure of the triangular matrix. Independent computations proceed within each level of the elimination tree. Overall, the sparsity pattern of the matrix can result in an extremely deep and narrow tree, thereby limiting the amount of available parallelism for the solver to fully utilize many-core architectures, especially when compared to an SpMV. In other words, there is little (or no) parallel work at most levels of the tree, and thus level scheduling does not provide significant speed-up, if any at all. Adopting an iterative approach leverages the speed of sparse matrix-vector products on GPUs, and by scaling, a finite Neumann sum is obtained (and thus a convergent Richardson iteration). We also explore the subtleties of employing an ILU smoother, in particular, how the parameters chosen for the ILU factorization can affect the departure from normality and convergence, and how scaling can mitigate potential problems.

4.1 | The Algorithm

For ease of reference, an ILU smoother with scaling is provided in Algorithm 3. To employ row scaling only within Algorithm 3, subsequent formation of $D_u$, $D$ is constructed as in (8) and updated $\tilde{U}$ as in (9). Here, only the $U$ factor is scaled. The ILU factorization is assumed to result in a lower triangular matrix with unit diagonal, e.g. $L = I + L_s$ which also generates a finite Neumann sum. Furthermore, in the numerical experiments, $\text{dep}(L)$ is generally small enough that it does not negatively affect the convergence. Therefore, only $U$ is considered in the analysis. However, in the case when $\text{dep}(L)$ becomes problematic, or if $L$ does not have unit diagonal, this approach easily extends to include scaling of $L$.

Scaling the triangular factors of the ILU smoother results in both a finite Neumann series and reduces the departure from normality. Next, we examine the effects of parameter choice in the ILU factorization on each of these.

---

5 This is a reasonable assumption as many factorizations produce such a matrix; e.g. consider the `ilu` function in Matlab.
Algorithm 3 ILU+Richardson smoother for AMG with Ruiz scaling of \( U \).

Given \( A \in \mathbb{C}^{n \times n} \), \( b \in \mathbb{C}^n \)

Define droptol and fill

Compute \( A \approx LU \) with droptol and fill imposed

Define \( m_L \) and \( m_u \), total number of iterations for solving \( L \) and \( U \)

Define \( y = 0 \), \( v = y \)

//Richardson iteration to solve \( Ly = b \)

for \( k = 1 : m_L \) do

\( y = b - L_s y \)

end for

Call Algorithm 2 with \( U \) and \( y \) to obtain scaled \( \tilde{U} \) and \( \tilde{y} \), and \( D_k \)

Let \( D_u = \text{diag}(U) \)

Define \( D = D_u^{-1} \)

//Richardson iteration to solve \( \tilde{U}v = \tilde{y} \)

for \( k = 1 : m_U \) do

\( v = D \tilde{y} - \tilde{U}_s v \)

end for

//Update and unpermute the solution

\( v = D_k v \)

\( x = P^{-1} v \)


4.2 Neumann Series and the Richardson Iteration

We consider the effect of iterating with a scaled ILU factorization (e.g. ILU scaled with row/column, or the row-scaled LDU).

In this case, the iteration matrix (5) simplifies to \( G_U = U_s \), where \( U_s \) is a strictly upper triangular matrix. To solve \( UX = b \) let \( b_s = D_U^{-1} b \), and \( U = I + U_s \). Then replace (5) with a Richardson iteration

\[
x^{(k+1)} = b_s + (I - U) x^{(k)} = b_s - U_s x^{(k)}
\]

where the unit diagonal is removed. After expanding it follows that

\[
x^{(k+1)} = b_s - U_s b_s + U_s^2 b_s - \cdots + (-1)^k U_s^k b_s = (I - U_s + U_s^2 - \cdots + (-1)^k U_s^k) b_s = (I + U_s)^{-1} b_s.
\]

Then, the inverse of \( U \) is expressed as a Neumann series

\[
U^{-1} = (I + U_s)^{-1} = I - U_s + U_s^2 - \cdots = \sum_{i=0}^{n} (-1)^i U_i,
\]

and with \( U_s \) strictly upper triangular and nilpotent, the above sum is necessarily finite.

The series in (12) converges when \( \|U_s\|_2 < 1 \), and in practice, this is true for the ILU(0) and ILUT smoothers for certain drop tolerances (consult Table 2), and in these cases, a convergent Neumann series is guaranteed. Even in cases when \( \|U_s\|_2 \geq 1 \), it is observed that \( \|U_p^p\|_2 < 1 \) for small \( p \), permitting truncation of the Neumann series - and thus the Richardson iteration - to a small number of terms. That \( \|U_p^p\|_2 \) eventually decreases is not unexpected because the number of possible non-zeros in \( U_s^p \) necessarily grows smaller as \( p \) grows larger when \( U_s \) is dense. However, numerical nilpotence is observed for \( p \ll n \) in many cases for sparse \( U_s \), and the size of \( p \) clearly depends on the number of non-zeros allowed in \( U \) and consequently \( U_s \) (either by imposition of small droptol or conservative fill, or both). In other words, \( \|U_p^p\|_2 \) is effectively zero much sooner than the theoretically guaranteed \( \|U_s^n\|_2 \).

Figure 7 displays \( \|U_p^p\|_2 \) when employing Ruiz scaling at the finest level. Here, Matlab’s \texttt{ilu} is used with type ‘ilutp’, threshold 0 (i.e. no pivoting), and various dropt tolerances. For larger dropt tolerances (i.e. droptol = 1.e-2 for both row/column and row scaling), \( \|U_p^p\|_2 \) < 1 for \( p = 1 \), giving convergence of the Neumann series. However, for smaller dropt tolerances this is no longer the case. In some cases \( \|U_p^p\|_2 \) actually increases for the first few values of \( p \), but eventually decreases and falls below 1. Figure
displays $\|U^p_s\|_2$ for matrix dimension $N = 14186$ with row scaling. In some cases, $\|U^p_s\| < 1$ for modest $p$ (e.g. $p = 7$ for \textit{droptol} = $1.e-2$). For smaller drop tolerances, $p$ can be moderately large (e.g. $p = 45$ for \textit{droptol} = $1.e-2$).

![Graphs showing $\|U^p_s\|_2$ for different values of $p$ and $N$](image)

**FIGURE 1** $\|U^p_s\|_2$ for $p = 1, 2, \ldots, 40$, for $U = I + U_s$ scaled using Ruiz and where $U_s$ is the strictly upper triangular part of $U$. The black, dotted line represents the bound 1.

In particular, when applying Ruiz, a simple upper bound for $\|U_s\|_2$ is derived and is attributed, in part, to the fill level permitted in the ILU factorization. Letting $q$ represent the number of non-zero elements permitted in the $L$ and $U$ of an ILU factorization and recalling that $u_{ij} \leq 1$ for all $i, j$ with Ruiz scaling,

$$\|U_s\|_2 \leq \|U_s\|_F \leq \sqrt{n(q-1)}.$$  \hspace{1cm} (13)

See Table 2 for a comparison of $\|U_s\|_2$ with $\|U_s\|_F$ and the bound given in (13), when varying the amount of fill per row in the ILU factorization. When $\|U_s\|_2 < 1 < \|U_s\|_F$, as is the case for several levels of fill shown in Table 2 and lower and upper bounds are obtained

$$\frac{1}{\sqrt{r}} \leq \|U_s\|_2 < 1,$$  \hspace{1cm} (14)

where $r = \text{rank}(U_s)$. Limiting the number of non-zeros allowed in the triangular factors may result in $U_s$ that is not full rank. In Table 2 observe that, for the PeleLM matrix of size $N = 14186$, the lower bound in (14) and upper bound in (13) remain more or less constant, suggesting that, under certain assumptions, $\text{rank}(U_s)$ does not change significantly as we relax the parameters.
\( \| U_s \|_2 \) for \( p = 1, 2, \ldots, 40 \), for \( U = I + U_s \) using row scaling \( LDU \) and where \( U_s \) is the strictly upper triangular part of \( U \). The black, dotted line represents the bound 1. Matrix dimension \( N = 14186 \). Not displayed in figure: \( \| U_s \|_2 \) drops below 1 at \( p = 45 \).

(i.e., increase the amount of fill in allowed) for the ILU factorization. The implications of \( \text{rank}(U_s) < n \) (or even \( \ll n \)) is part of future work.

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{fill} & 5 & 10 & 20 & 50 & 200 \\
\hline
\| U_s \|_2 & 0.679 & 0.753 & 0.87 & 1.04 & 1.22 \\
\| U_s \|_F & 120.69 & 121.86 & 122.22 & 122.39 & 122.44 \\
\hline
\text{Bound in (13)} & 283.21 & 357.32 & 519.17 & 833.74 & 1.68e+3 \\
\text{Bound in (14)} & 7.91e-5 & 7.34e-5 & 7.17e-5 & 7.12e-5 & 7.11e-5 \\
\hline
\end{array}
\]

TABLE 2 Comparison of \( \| U_s \|_2 \), \( \| U_s \|_F \), and the theoretical bounds (13) and (14) for varying amounts of fill per row permitted in the ILU factorization on the finest level, and after scaling the linear system from PeleLM of dimension \( N = 14186 \).

Next, the work of Chow and Saad\(^{12}\) motivates a parameter study for the ILU factorization and an examination of how the choices for drop tolerance and fill-in can affect the departure from normality, conditioning of the factors, and subsequently the required number of Jacobi iterations to achieve fast convergence of the Krylov method when using AMG as a preconditioner.

### 4.3 Parameter Choice in the ILU factorization

While the existing literature on the choice of parameters for ILU smoothers for AMG is limited, the work in\(^{13}\) provides a framework for studying the effect of parameter choices on an ILU factorization. Here, the relationship between these choices and the resulting sizes of \( \text{dep}(U) \) and \( \kappa(U) \) is examined, and subsequently the number of Jacobi iterations required. The potential inaccuracy when the \( L \) and \( U \) have high condition numbers is first examined. When employing a threshold parameter (or drop tolerance) to limit the number of non-zeros in a row (or column) of the factors, the factorization of a symmetric matrix could be highly nonsymmetric\(^{12}\). One observable indicator is the vertical striping in the sparsity pattern of \( L + U \), which signifies orders of magnitude difference in the entries of a row (or column) of the coefficient matrix \( A \) and is associated with ill-conditioning of the triangular factors\(^{12}\).
Figure 3 displays the sparsity pattern of $L + U$ on the first four levels of AMG using the AMGToolbox for matrix dimension $N = 14186$. Here, the drop tolerance is set to $1.e^{-15}$ and fill limit to 200 per row using the ILUTP implementation with pivoting turned off.\footnote{The term pivoting refers to row or column exchanges employed when a small pivot, or divisor, is encountered in the factorization. With pivoting turned off, the pivot is always the diagonal element.}

Striping is attributed to small pivots when computing the ILU factorization, in large amounts of fill-in, and the obvious choice to mitigate this is to enforce a smaller fill level per row\footnote{Such a small drop tolerance and large fill-in are generally unreasonable choices for an ILU factorization because they can substantially increase the cost associated with computing (and storing) the factors. The data emphasizes the resulting vertical striping associated with poorly chosen parameters.}. However, for a very ill-conditioned $A$, limiting the fill level per row may be insufficient to prevent ill-conditioned $L$ and $U$. Figure 4 displays the non-zero pattern of $L + U$ for the same matrices as in Figure 3 but with the fill level per row now set to 10. The dramatic striping pattern is no longer present, however some remains, indicating the potential for ill-conditioning despite restricting the amount of fill. As shown in the captions of each figure, limiting fill alone is not enough to reduce the high condition number of these particular matrices.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3.png}
\caption{Non-zero patterns of $L + U$ for matrix size $N = 14186$ for the first four levels using AMGToolbox. Drop tolerance is set to $1.e^{-15}$ and fill limit per row set to 200.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure4.png}
\caption{Non-zero patterns of $L + U$ for matrix size $N = 14186$ for the first four levels using AMGToolbox. Drop tolerance is set to $1.e^{-15}$ and fill limit per row set to 10.}
\end{figure}

An example demonstrating the effects of a conservative drop tolerance on the conditioning of $A$ is omitted here, however, by imposing a very small drop tolerance, the resulting triangular solves may be unstable. This is due to the fact that the off diagonal...
elements are much larger than those along the diagonal. See Section 2.2. In fact, the prescription given in is to consider scaling to reduce $\kappa(A)$ with the warning that this will result in non-normal triangular factors.

A similar analysis as it affects $\text{dep}(U)$ is now presented. Specifically, consider the relationship between the drop tolerance imposed for the ILU factorization and the required number of Richardson iterations after performing Ruiz scaling on the $U$ factor; see Table 3. The results displayed in the first four columns rely on Matlab’s built-in $\text{ilu}$ function, with type ‘ilutp’, pivoting turned off (i.e. threshold is set to 0), and drop tolerances varied between $1\text{e}−2$ and $1\text{e}−5$. Matrices with dimension $N = 14186$ are again exported from the PeleLM application.

For larger drop tolerances $1\text{e}−2$ and $1\text{e}−3$, the average number of $\text{nnz}$ per row in $L$ is limited to approximately 13 and 50. For $1\text{e}−4$ and $1\text{e}−5$, this increases to approximately 160 and 360. The average $\text{nnz}$ per row in the corresponding $U$ is more or less the same, except for drop tolerance $1\text{e}−5$, where there are 385 $\text{nnz}$ per row in $U$. For drop tolerance $1\text{e}−5$, the number of iterations remains constant for both $L$ and $U$ in order to achieve a decrease in the GMRES iterations to reach a relative residual error tolerance $1\text{e}−6$. It was found that regardless of drop tolerance, the number of iterations remains constant at 3 for $U$ and 2 for $L$, and the GMRES iterations count is constant at 6.

Because the cost of sparse matrix vector products with the $L$ and $U$ obviously depends on the number of non-zeros in the $L$ and $U$ matrices, limiting the fill level per row is a prudent choice, especially for the drop tolerances $1\text{e}−4$ and $1\text{e}−5$. However, because Matlab’s $\text{ilu}$ does not allow for control of the fill level explicitly, the ILUT implementation in is employed. In the final two columns of Table 3 the fill is limited to 10 per row for these drop tolerances. Even with the low number of iterations for the triangular solves, the GMRES iteration count remains constant at six (6). Thus, to achieve an accurate solution from the iterative solver, a larger drop tolerance with low fill and just a few iterations for the triangular solve are sufficient.

| Totals | ILU droptol | No Fill Specified | Fill = 10 |
|--------|-------------|-------------------|-----------|
| $\text{nnz}(L)$ | $1\text{e}−2$ | $1\text{e}−3$ | $1\text{e}−4$ | $1\text{e}−5$ | $1\text{e}−4$ | $1\text{e}−5$ |
| $197162$ | $702482$ | $2268634$ | $5139896$ | $133552$ | $134543$ |
| $190983$ | $709452$ | $2492995$ | $5473513$ | $141276$ | $141490$ |
| $1.486e+4$ | $1.487e+4$ | $1.487e+4$ | $1.593e+4$ | $1.593e+4$ |
| $1.107e+8$ | $1.163e+8$ | $1.173e+8$ | $1.063e+8$ | $1.063e+8$ |
| $26.71$ | $28.14$ | $28.39$ | $28.49$ | $25.82$ | $25.80$ |
| $3$ | $3$ | $3$ | $3$ | $3$ |
| $2$ | $2$ | $2$ | $2$ | $2$ |
| $6$ | $6$ | $6$ | $6$ | $6$ |

TABLE 3 Iterations for the upper and lower triangular solves versus GMRES iterations to reach tolerance $1\text{e}−6$ for different drop tolerances with row scaling $LDU$ and row/column Ruiz scaling using AMGToolbox. The first four columns use Matlab’s $\text{ilu}$ and fill levels are not specified. The last two columns display $\text{nnz}$ and iterations with fill limit 10 elements per row for drop tolerances $1\text{e}−4$ and $1\text{e}−5$ using the ILUT described in Carr et al. Matrix size $N = 14186$.

An upper bound on the $\text{dep}(U)$ is derived after Ruiz scaling (or any scaling that results in the diagonal elements being one and the off-diagonal elements being less than or equal to one). Recall that the elements of $U_s$ are such that $u_{i,j} \leq 1$, and with $U = I + U_s$, it follows that

$$\text{dep}(U)^2 = \| U \|_F^2 - \| D \|_F^2 = \| I + U_s \|_F^2 - n$$

$$= \| I + U_s \|_F \| I + U_s \|_F - n \leq (\sqrt{n} + \| U_s \|_F)^2 - n$$

$$= (2\sqrt{n} + \| U_s \|_F)\| U_s \|_F,$$

or more concisely,

$$\text{dep}(U) \leq \sqrt{(2\sqrt{n} + \| U_s \|_F)\| U_s \|_F}.$$  \hspace{1cm} (15)$$

This bound provides a practical guarantee on the departure from normality of scaled $U$: when $\| U_s \|_F$ is modest (e.g. $\mathcal{O}(10)$ or even $\mathcal{O}(10^2)$), a reasonable bound on $\text{dep}(U)$ is expected. The bound also quantifies the earlier observation that fewer non-zeros
in \( U_j \) may lead to smaller \( \text{dep}(U) = \text{dep}(I + U_j) \). In fact, an upper bound on \( \text{dep}(U) \) depends directly on the amount of fill-in allowed by combining (13) with (15) to obtain

\[
\text{dep}(U) \leq \sqrt{n(2\sqrt{q-1} + q-1)}.
\]

This relaxation results, but is computed without knowledge of the size of \( \|U_j\|_F \) and demonstrates the potential for the increase in \( \text{dep}(U) \) when more fill is allowed in the ILU factorization, even after scaling. Table 4 compares \( \text{dep}(U) \) before and after scaling with the bounds given in (15) and (16) for matrices with dimension \( N = 14186 \) from the PeleLM application.

### TABLE 4

| fill = 5 | fill = 10 | fill = 20 | fill = 50 | fill = 200 |
|----------|-----------|-----------|-----------|------------|
| \( \text{dep}(U) \) before | 7.98e+7 | 1.06e+8 | 1.12e+8 | 1.16e+8 | 1.17e+8 |
| \( \text{dep}(U) \) after | 208.12 | 209.48 | 209.89 | 210.08 | 210.14 |
| Bound in (15) | 336.88 | 461.29 | 627.06 | 945.37 | 1.795e+3 |

Previous results have shown the reduction in \( \kappa(U) \) and \( \text{dep}(U) \) when applying scaling to the upper triangular factor on the finest level. In Table 5, the departure from normality and condition number of the factors is displayed for all levels of the \( V \)-cycle. Based on the previous results, a moderate drop tolerance of 1e-2 is chosen and the fill per row is set to 10.

### TABLE 5

| Level \( \ell \) | \( \text{dep}(L) \) | \( \text{dep}(U) \) | \( \kappa_2(A) \) | \( \kappa_2(L) \) | \( \kappa_2(U) \) |
|---------------|-------------|-------------|-------------|-------------|-------------|
| \( \ell = 1 \) | 1.48e+4 | 1.17e+8 | 28.40 | 6.65e+15 | 1.95e+8 |
| \( \ell = 2 \) | 9.29e+3 | 9.03e+7 | 19.66 | 9.91e+15 | 1.27e+8 |
| \( \ell = 3 \) | 5.25e+3 | 1.84e+8 | 11.41 | 4.72e+16 | 1.20e+8 |
| \( \ell = 4 \) | 1.81e+3 | 1.23e+11 | 14.15 | 1.14e+19 | 2.80e+7 |
| \( \ell = 5 \) | 1.36e+3 | 1.89e+11 | 10.15 | 1.43e+18 | 1.00e+7 |
| \( \ell = 6 \) | 1.02e+3 | 3.69e+10 | 5.72 | 8.16e+16 | 5.95e+6 |

From the earlier analysis, both smaller amounts of fill-in and larger drop tolerances are reasonable choices when using an ILU smoother and iteration to solve the triangular linear systems. In Tables 6 and 7, the departure from normality and condition number of the factors, respectively, are compared before and after scaling when using these more modest parameter choices for matrices obtained from the SuiteSparse Matrix Collection (first five rows). Results for three matrices exported from PeleLM (final three rows) are also provided. To generate these results Matlab’s \texttt{ilu} was applied with setup type ‘nofill’ (i.e. ILU(0)). In the cases considered here, \( \text{dep}(L) \) is not very large - and in all cases shown, it is smaller than \( \text{dep}(U) \), motivating the decision to scale only \( U \). The condition numbers of the matrices are listed Table 7.

Next, this strategy extends to much larger linear systems exported from the Nalu-Wind and PeleLM pressure solvers and a parallel performance analysis is provided. The Nalu-Wind matrix has dimension \( N = 21 \) million and is sufficiently large to
Thomas, Carr, Mullowney, Świrydowicz, Day

| Matrix         | Dimension | \(nnz(A)\) | \(nnz(L)\) | \(nnz(U)\) | \(dep(L)\) | \(dep(U)\) | \(dep(D^{-1}U)\) | \(dep(D_r UD_c)\) |
|----------------|-----------|-------------|-------------|-------------|-------------|-------------|--------------------|--------------------|
| \(af_0_0_k101\) | 503625    | 17550675    | 9027150     | 9027150     | 326.95      | 1.84e8      | 326.95             | 320.89             |
| \(af\_shell1\) | 504855    | 17562051    | 9033453     | 9033453     | 386.66      | 1.52e8      | 386.66             | 407.35             |
| \(bundle\_adj\) | 513351    | 20207907    | 10360629    | 10360629    | 8.52e6      | 4.52e11     | 8.52e6             | 438.70             |
| \(F1\)          | 343791    | 26837113    | 13590452    | 13590452    | 335.52      | 7.05e15     | 335.52             | 222.71             |
| \(offshore\)    | 259789    | 4242673     | 222666      | 222666      | 16.99       | 1.09e7      | 16.99              | 9.33               |
| \(PeleLM331\)   | 331       | 5415        | 2873        | 2873        | 1.45e4      | 1.00e6      | 1.45e4             | 26.33              |
| \(PeleLM2110\)  | 2110      | 42422       | 22266       | 22266       | 16.99       | 1.09e7      | 16.99              | 9.33               |
| \(PeleLM14186\) | 14186     | 291068      | 152627      | 152627      | 1.45e4      | 1.00e6      | 1.45e4             | 26.33              |

**TABLE 6** Departure form normality for the \(L\) and \(U\) of an ILU(0) factorization, followed by the departure from normality after row scaling and Ruiz scaling the \(U\) factor. The first five matrices are from\(^{[34]}\) and the last three are exported from PeleLM. Matlab’s `ilu` with type ‘nofill’ was computed for all matrices.

| Matrix         | \(\kappa(A)\) | \(\kappa(L)\) | \(\kappa(U)\) | \(\kappa(D^{-1}U)\) | \(\kappa(D_r UD_c)\) |
|----------------|----------------|----------------|----------------|----------------------|----------------------|
| \(af_0_0_k101\) | 3.60e8         | 156.54         | 1.02e3         | 75.78                | 108.83               |
| \(af\_shell1\) | 1.72e10        | 49.99          | 231.94         | 116.42               | 171.30               |
| \(bundle\_adj\) | 6.10e15        | 4.59e12        | 3.53e14        | 2.93e12              | 2.37e3               |
| \(F1\)          | 3.26e7         | 6.51e3         | 1.34e5         | 2.67e4               | 1.54e4               |
| \(offshore\)    | 2.32e13        | 96.79          | 7.56e10        | 148.35               | 156.99               |
| \(PeleLM331\)   | 3.48e17        | 14.06          | 3.87e9         | 43.06                | 18.72                |
| \(PeleLM2110\)  | 3.21e17        | 13.39          | 4.41e9         | 34.02                | 12.31                |
| \(PeleLM14186\) | 6.64e15        | 1.83e8         | 6.87e12        | 1.74e7               | 9.51                 |

**TABLE 7** Condition number for \(A\), \(L\), and \(U\) for an ILU(0) factorization, followed by the condition numbers after row scaling and Ruiz scaling the \(U\) factor. The first five matrices are from\(^{[34]}\) and the last three are exported from PeleLM. Matlab’s `ilu` function with type ‘nofill’ was computed for all matrices.

#### 5 NUMERICAL RESULTS

Our approach accelerates the solution of linear systems exported from the “nodal projection” component of the time stepping scheme employed in PeleLM\(^{[3]}\) and the pressure continuity equation for the Nalu-Wind\(^{[17]}\) CFD model. PeleLM is an adaptive mesh low Mach number combustion code developed and supported under DOE’s Exascale Computing Program. PeleLM features a variable-density projection scheme to ensure that the velocity field used to advect the state satisfies an elliptic divergence constraint. Physically, this constraint enforces a consistently evolving flow with a spatially uniform thermodynamic pressure across the domain. A key feature of the model is that the fluid density varies considerably across the computational domain, leading to highly ill-conditioned matrices representing the elliptic projection operator. Section 5 indicates that the standard Jacobi and Gauss-Seidel smoothers are less effective in these cases at reducing the residual error at each level of the AMG \(V\)-cycle and may result in very large iteration counts for the Krylov solver. Jomo et al.\(^{[9]}\) compare GMRES+AMG with Jacobi and Gauss-Seidel smoothers with GMRES preconditioned by ILU. However, they did not investigate ILU as a smoother. Results are also reported for linear systems from Nalu-Wind\(^{[18]}\) based on the incompressible Navier-Stokes equations. Nalu-Wind is coupled to AMR-wind through an overset mesh in order to simulate both the near wind-turbine air flow and far-field atmospheric boundary layer. The solver has also been incorporated into the MFIX-Exa CFD-DEM model\(^{[35]}\) for carbon capture and chemically reacting fluid flows. In this case, the solver leads to at least 5× improvement in the computational speed on GPUs.
5.1 | Stopping Criteria

The stopping criteria for Krylov methods is an important consideration and is related to backward error for solving linear systems \( Ax = b \). The most common convergence criterion found in existing iterative solver frameworks is based upon the relative residual, defined by

\[
\frac{\|r^{(k)}\|_2}{\|b\|_2} = \frac{\|b - Ax^{(k)}\|_2}{\|b\|_2} < \text{tol},
\]

where \( r^{(k)} \) and \( x^{(k)} \) represent, respectively, the residual and approximate solution after \( k \) iterations of the iterative solver. An alternative metric commonly employed in direct solvers is the norm-wise relative backward error (NRBE)

\[
\text{NRBE} = \frac{\|r^{(k)}\|_2}{\|b\|_2 + \|A\|_\infty \|x^{(k)}\|_2}.
\]

In numerical experiments, the norm-wise relative backward error for the solution of linear systems with GMRES was sometimes found to be lower than when the right-preconditioned GMRES was employed. Indeed, the latter exhibited false convergence (the implicit GMRES residual norm did not agree with the norm of the explicit residual \( r^{(k)} = b - Ax^{(k)} \) when executed in parallel for highly ill-conditioned problems, \( \kappa(A) = 1e+15 \). Flexible FGMRES was found to be the most effective Krylov solver in with AMG preconditioner and did not exhibit false convergence.

5.2 | AMG Implementations

The hypre-BoomerAMG library was designed for massively-parallel computation and now also supports GPU acceleration of key solver components. Direct interpolation is straightforward to implement on GPUs because the interpolatory set of a fine point \( i \) is just a subset of the neighbors of \( i \), and thus the interpolation weights are determined solely by the \( i \)-th equation. The weights \( w_{ij} \) are computed by solving the local least squares problem

\[
\min \|a_i^T w + a_{i,C_i}^T \|_2 \quad \text{s.t.} \quad w^T f_{C_i} = f_i,
\]

where \( w_i \) is a vector containing \( w_{ij} \), \( C_i \) and denotes strong C-neighbors of \( i \) and \( f \) is a target vector that needs to be interpolated exactly. For elliptic problems where the near null-space is spanned by constant vectors, i.e. \( f = 1 \), the closed-form solution of (5.2) is given by

\[
w_{ij} = -\frac{a_{ij} + \beta_i/n_{C_i}}{a_{ii} + \sum_{k \in N_i^w} a_{ik}}, \quad \beta_i = \sum_{k \in \{f, i, C_i\}} a_{ik},
\]

where \( n_{C_i} \) denotes the number of points in \( C_i \), \( C_i \) the weak C-neighbors of \( i \), \( f \) the F-neighbors, and \( N_i^w \) the weak neighbors.

With minor modifications to the original form, it turns out that the extended interpolation operator is rewritten in terms of standard sparse matrix computations such as matrix-matrix (M-M) multiplication and diagonal scaling with certain FF- and FC-sub-matrices. The coarse-fine C-F splitting of the coarse matrix \( A \) and the full prolongation operator \( P \) are given by

\[
A = \begin{bmatrix} A_{FF} & A_{FC} \\ A_{CF} & A_{CC} \end{bmatrix}, \quad P = \begin{bmatrix} W \\ I \end{bmatrix}
\]

where \( A \) is assumed to be decomposed into \( A = D + A^s + A^w \), the diagonal, the strong part and weak part respectively, and \( A_{FF}^s, A_{FC}^s, A_{FC}^w \) and \( A_{FC}^w \) are the corresponding sub-matrices of \( A^w \) and \( A^s \).

The extended “MM-ext” interpolation takes the form

\[
W = -[(D_{FF} + D_\gamma)^{-1}(A_{FF}^s + D_\beta)](D_\beta^{-1}A_{FC}^s)
\]

with

\[
D_\beta = \text{diag}(A_{FC}^w 1_C) \quad D_\gamma = \text{diag}(A_{FF}^s 1_F + A_{FC}^w 1_C)
\]

The above formulation allows simple and efficient implementations that can utilize optimized sparse kernels on GPUs. Similar approaches that are referred to as “MM-ext+i” modified from the original extended+i algorithm and “MM-ext+e” are also available in BoomerAMG. See for details on the class of M-M based interpolation operators.
5.3 | PeleLM Combustion Model

Pressure linear systems are exported from the ‘nodal projection’ component of the time integrator used in PeleLM. PeleLM features a variable-density projection scheme to ensure that the velocity field used to advect the state satisfies an elliptic divergence constraint. Physically, this constraint enforces that the flow evolves consistently with a spatially uniform thermodynamic pressure across the domain. A key feature of the model is that the fluid density varies considerably across the computational domain. Extremely ill-conditioned problems arise for incompressible and reacting flows in the low Mach flow regime, particularly for discretizations with cut-cell meshes in complex geometries, where non-covered cells that are cut by the domain boundary can have arbitrarily small volumes and areas. The standard Jacobi and Gauss-Seidel smoothers are less effective in these cases at reducing the residual error at each level of the AMG V-cycle and may result in very large iteration counts for the GMRES+AMG solver Jomo et al.

A sequence of three different size problems was examined, based on matrices exported from the PeleLM pressure continuity solver. The first of these is a dimension \( N = 14186 \) linear system, solved with GMRES+AMG using the AMGToolBox by applying ILUT smoothing only on the finest level \( l = 1 \), then ILUT on all levels and polynomial Gauss-Seidel smoothers. Iterative Richardson solvers are employed. Two pre- and post-smoothing sweeps were applied on all V-cycle levels, except for the coarse level direct solve. The AMG strength of connection threshold was set to \( \theta = 0.25 \). The convergence histories are plotted in Figure 5. The lowest iteration count results from using \( ILU(0) \) smoothing on all levels, however, the minimum compute time is obtained by using \( ILU(0) \) only on the finest level and polynomial Gauss-Seidel on the remaining levels. The iterative triangular solves fail to converge unless either a preconditioned Jacobi or Richardson iteration is employed. In the case of \( ILU(0) \), both row and Ruiz row/column scaling exhibit similar convergence histories and thus row scaling is less costly in the set-up phase.

In the case of ILUT as the smoother, there are differences in the achievable GMRES error level between the row and row/column scaling, depending on the drop tolerances and level of fill per row. For the dimension \( N = 14K \) problem, the norm-wise relative backward error (NRBE) is reported in Table 8, for both row and Ruiz row/column scaling, with \( drop = 1e−2 \) and \( lfill = 3 \). The \( L \) solver iterations are fixed at ten (10), whereas the number of \( U \) iterations is varied. Table 9 presents results for \( drop = 1e−3 \) and \( lfill = 5 \), where the NRBE is found to be an order of magnitude lower. For this problem, the slightly lower backward error obtained using two Richardson iterations with the Ruiz scaling versus three iterations using row scaling may not be sufficient to justify the additional set-up cost.

| \( U \) iters. | 5       | 4       | 3       | 2       | 1       |
|---------------|---------|---------|---------|---------|---------|
| row scale     | 3.588e−8 | 3.857e−8 | 3.018e−8 | 2.604e−8 | 9.63e−8 |
| row/col scale | 3.588e−8 | 3.857e−8 | 3.837e−8 | 2.837e−8 | 9.63e−8 |

TABLE 8 GMRES+AMG norm-wise relative backward error (NRBE). PeleLM matrix dimension \( N = 14186 \). \( drop = 1e−2 \), \( lfill = 3 \). Six GMRES iterations.

| \( U \) iters. | 5       | 4       | 3       | 2       | 1       |
|---------------|---------|---------|---------|---------|---------|
| row scale     | 5.85e−10 | 7.3e−10 | 5.94e−10 | 3.84e−10 | 5.35e−8 |
| row/col scale | 5.85e−10 | 7.17e−10 | 5.93e−10 | 3.85e−10 | 1.91e−9 |

TABLE 9 GMRES+AMG norm-wise relative backward error (NRBE). PeleLM matrix dimension \( N = 14186 \). \( drop = 1e−3 \), \( lfill = 5 \). Six GMRES iterations.

Results using Hypre-BoomerAMG for the \( N = 1.4 \) million linear system are plotted in 7. These tests were performed on the NREL Eagle supercomputer with Intel Skylake CPUs and NVIDIA V100 GPUs. The parallel maximum independent set (PMIS) algorithm is applied together with aggressive coarsening and “MM-ext+i” interpolation are employed, with a strength of connection threshold \( \theta = 0.25 \). Because the problem is very ill-conditioned, flexible FGMRES achieves the best convergence rates and the lowest NRBE. Iterative solvers were employed in these tests with ten (10) iterations. The convergence histories are
plotted for hybrid-ILUT, and polynomial Gauss-Seidel smoothers. The ILUT parameters were $droptol = 1e^{-2}$ and $lfil = 10$. The lowest time for a single-GPU, was the ILUT smoother with Richardson iterations which achieved a solve time of 0.11 seconds.

For comparison, a larger PeleLM linear system of dimension $N = 4$ million was run on the ORNL Crusher supercomputer. The machine contains multiple compute nodes consisting of two AMD Epyc CPU sockets and four AMD MI250X GPUs. The same AMG parameters as the dimension 1.4M problem were specified, however, and ILU(0) smoother is applied on the first three $V$-cycle levels. The Richardson iterations are reduced to six (6) upper and five (5) lower per level. To reduce the relative residual to $1e^{-11}$, the solve time is 0.11 seconds with iterative triangular solvers. Whereas, the solve time is 0.16 seconds when a direct solver is employed in the smoother. The speed-up is now reduced to $1.5 \times$ on the AMD MI250X GPU. This may be attributed to a faster implementation of the direct triangular solver by AMD. Despite the increased number of GMRES+AMG iterations for the polynomial Gauss-Sediel smoother, the solve time is 0.16 seconds, which is comparable to the direct solves.
with ILU. However, the number of GMRES iterations has grown from the smaller problem and is expected to increase further at larger problem sizes.

**FIGURE 7** Hypre-BoomerAMG GPU results. Convergence history of (F)GMRES+AMG with polynomial, and ILU smoothers on first 3 levels with iterative solves. Matrix size $N = 4$ million

**FIGURE 8** Hypre-BoomerAMG GPU results. Nalu-Wind. Convergence history of (F)GMRES+AMG with polynomial, and ILUT Schur Complement smoothers with iterative solves. Matrix size $N = 23$ million

### 5.4 Exa-Wind Fluid Mechanics Models

The ExaWind ECP project aims to simulate the atmospheric boundary layer air flow through an entire wind farm on next-generation exascale-class computers. The primary physics codes in the ExaWind simulation environment are Nalu-Wind and AMR-Wind. Nalu-Wind and AMR-Wind are finite-volume-based CFD codes for the incompressible-flow Navier-Stokes governing equations. Nalu-Wind is an unstructured-grid solver that resolves the complex geometry of wind turbine blades and thin
blade boundary layers. AMR-Wind is a block-structured-grid solver with adaptive mesh refinement (AMR) capabilities that captures the background turbulent atmospheric flow and turbine wakes. Nalu-Wind and AMR-Wind models are coupled through overset meshes. The equations consist of the mass-continuity equation for pressure and Helmholtz-type equations for transport of momentum and other scalars (e.g. those for turbulence models). For Nalu-Wind, simulation times are dominated by linear-system setup and solution of the continuity and momentum equations. Both PeleLM and AMR-Wind are built on the AMReX software stack and employs geometric multigrid is the primary solver, however, it has the option of using the hypre library as a solver at the coarsest AMR level.

The NREL 5-MW turbine is a notional reference turbine with a 126 meter rotor that is appropriate for offshore wind studies. The simulations performed here use the model described in Thomas et al. and Sprague et al., but with rigid blades, and they include low- and high-resolution models of a single-turbine. These models use inflow and outflow boundary conditions in the directions normal to the blade rotation and symmetry boundary conditions in other directions. For each simulation, 50 time steps are taken from a cold start with four Picard iterations per time step. The cold start implies that the simulation will undergo an initial transient phase from a non-physical initial solution guess before settling into a quasi-steady solution state. This initial transient phase is challenging for the linear-system solvers and will require more GMRES iterations per equation system. Convergence histories for one such pressure linear system (after reaching steady-state) are displayed in Figure for the ILUT-Schur complement and polynomial Gauss-Seidel smoothers. The former requires half as many iterations to reach the 1e−5 convergence tolerance. Furthermore, a coarse-fine (C-F) ordering of the degrees of freedom results in fewer iterations. The strength of connection parameter was set to θ = 0.57, which contributes to a reduction in the AMG set-up time. In addition, the ILU drop tolerance was droptol = 1.0 × 10^{-2}, with a fill level per row of fill = 2. Sufficient smoothing was achieved with 18 iterations for the lower triangular L solve and 31 for the U solve. Aggressive coarsening was not specified and a single level of ILU smoothing was applied.

5.5 Parallel Performance

A hybrid AMG algorithm is obtained with ILU smoothing on the finest levels of the AMG V-cycle hierarchy (e.g. level 1), followed by polynomial Gauss-Seidel on the remaining levels. In the numerical results reported earlier, the ILU smoothing is used on all levels and also in the hybrid configuration for comparison. The latter requires fewer sparse matrix-vector multiplies and thus is more efficient.

The computational cost of the V-cycle, besides the high set-up cost, is determined by the number of non-zeros in the factors. The ILU smoother requires twenty (20) Richardson iterations and one outer sweep. These are applied during pre- and post-smoothing or twice. Therefore, the total number of flops required for the V-cycle with ILU(0) smoothing on every level is given by the sum below, where an SpMV costs 2 × nnz(A_l) on level l,

\[ \text{flops} = \sum_{l=1}^{N-1} \text{nnz}(A_l) \times 80 \]  

whereas the factor 80 is replaced by 8 for the polynomial smoother of degree two. This makes a compelling case for the hybrid AMG approach with a combination of ILU and polynomial Gauss-Seidel smoothers when the convergence rate is improved and leads to lower compute times. The cost of the coarse grid direct solve is \( O(N_c^2) \), where \( N_c \) is the dimension of the coarsest level matrix \( A_c \), and is small in comparison. The cost of the Krylov iteration is dominated by the SpMV with the matrix \( A \), whose cost is given by 2 × nnz(\( A \)).

The cost of a sparse direct triangular solver on a many-core GPU architecture such as from the NVIDIA cuSparse library is 10 to 25x slower than the SpMV. For the NVIDIA V100 GPU architecture, the SpMV can now achieve on the order of 50 – 100 GigaFlops/sec in double precision floating point arithmetic. The cost of a V-cycle for the third problem with matrix dimension \( N = 14186 \) would be \( 2.3 \times 10^7 \) flops, whereas the SpMV in the Krylov iteration costs \( 5.8 \times 10^5 \) flops. The former would require 0.0005 seconds to execute on the V100 GPU (assuming 50 GigaFlops/sec sustained performance), and the latter would take 0.000012 seconds. The compute time for the GMRES solver is 0.00051 seconds per iteration and six (6) iterations would execute in 0.0034 seconds (which is quite close to the measured time in Table). When the number of GMRES+AMG iterations to achieve the same NRBE remains less than two times larger, then the case for employing the hybrid scheme on GPUs becomes rather compelling.

Finally, the compute times of the GMRES+AMG solver in hypre with an incomplete LDU smoother, and either direct or iterative solvers in the ILU smoother, are compared below. The compute times for a single pressure solve are given in Table.
for the $N = 14186$ dimension matrix. The ILU(0) and ILUT smoothers are included for comparison. Both the CPU and GPU times are reported. In all cases, two Gauss-Seidel and one ILU sweep are employed. The solver time reported again corresponds to when the relative residual decreases below $1e^{-5}$.

| Iterations | Gauss-Seidel | Poly G-S | ILUT direct | ILUT iter | ILU(0) iter |
|------------|--------------|----------|-------------|-----------|-------------|
| CPU (sec)  | 0.037        | 0.025    | 0.025       | 0.035     | 0.038       |
| GPU (sec)  | 0.021        | 0.0067   | 0.032       | 0.0065    | 0.0048      |

TABLE 11 GMRES+AMG compute time. Gauss-Seidel, poly Gauss-Seidel, and ILU smoothers. PeleLM matrix dimension $N = 14186$

First, consider the CPU compute times for a single solve. The results indicate that the GMRES+AMG solver time using the hybrid $V$-cycle with an ILU(0) smoother on the first level, versus a direct solver for the $L$ and $U$ systems, costs less than Gauss-Seidel smoothing on all levels. The PMIS algorithm is employed along with aggressive coarsening on the first $V$-cycle level. One sweep of the Gauss-Seidel smoother is employed in both configurations. The longer time is primarily due to the higher number of Krylov iterations required to converge. The ILUT smoother, on the finest level, with iterative solvers is the more efficient approach on the GPU. Despite only ten (10) SpMV products to solve the $L$ and 15 to solve the $U$ systems, the computational speed of the GPU for the SpMV kernel is more than sufficient to overcome the cost of a direct solve. The compute time estimate of 0.0034 sec for the solver when $N = 14186$ is fairly accurate given six iterations to converge and aggressive coarsening with hypre reduces the $nnz(A)$ per level.

The compute times for a larger dimension problem where $N = 1.4$ million are reported in Table 12. Here it was observed that the ILU(0) compute time on the GPU is lower than with the polynomial Gauss-Seidel smoother. Most notably, the GPU compute time for ILU(0) solves with fifteen (15) SpMV for $U$ are two times faster than ILUT with direct solves. To further explore the parallel strong-scaling behaviour of the iterative and direct solvers within the ILU smoothers, the GMRES+AMG solver was employed to solve a PeleLM linear system of dimensions $N = 11$ million. The $LDU$ form of the factorization with row scaling was again employed and twenty (20) SpMV provide sufficient smoothing for this much larger problem. The linear system solver was tested on the NREL Eagle Supercomputer configured with two NVIDIA Volta V100 GPUs per node. Most notably, the solver with iterative Neumann scheme achieves a faster solve time compared to the direct solver as displayed in Figure 9. The convergence histories of the GMRES+AMG solver with a polynomial Gauss-Seidel and the ILU direct and iterative smoothers are plotted in Figure 10.

The strong-scaling performance of the low resolution NREL 5 MegaWatt single-turbine mesh is displayed in Figure 11. The matrix dimension for this problem is $N = 23$ million. The total setup plus solve time is displayed for (F)GMRES+AMG executing on the NREL Eagle supercomputer using two NVIDIA V100 GPUs per node on up to 20 nodes or 40 GPUs. The solve time is plotted for the polynomial Gauss-Seidel and ILUT Schur complement smoothers. In the latter case, a single iteration of the iterative GMRES solver, without residual computations, results in a steeper decrease in the execution time and improved strong-scaling performance. In addition, the number of GMRES+AMG solver iterations to reach a relative residual tolerance of $1e^{-5}$ remains constant at eleven (11) as the number of compute nodes is increased.
|                | Gauss-Seidel | Poly G-S | ILUT direct | ILUT iter | ILU(0) iter |
|----------------|--------------|----------|-------------|-----------|-------------|
| iterations     | 7            | 9        | 8           | 8         | 4           |
| CPU (sec)      | 9.2          | 9.6      | 4.3         | 6.9       | 6.8         |
| GPU (sec)      | 0.29         | 0.055    | 0.098       | 0.058     | 0.042       |

**TABLE 12** GMRES+AMG compute time. Gauss-Seidel, poly Gauss-Seidel, and ILU smoothers. PeleLM matrix dimension $N = 1.4$ million.

**FIGURE 9** Hypre-BoomerAMG GPU results. Strong-scaling of (F)GMRES+AMG with ILU(0) smoothers using direct and iterative solves. Matrix size $N = 11$ million.

**FIGURE 10** Hypre-BoomerAMG GPU results. Convergence histories of (F)GMRES+AMG with polynomial Gauss-Seidel and ILU(0) smoothers using direct and iterative solves. Nalu-Wind matrix dimension $N = 11$ million.

## 6 | CONCLUSIONS

A novel approach was developed for the solution of sparse triangular systems for the $L$ and $U$ of an ILU smoother for AMG. Previous work by H. Anzt, and E. Chow demonstrated that these factors are highly non-normal, even after appropriate re-ordering.
and scaling of the linear system $Ax = b$ and in this case, Jacobi iterations may diverge. In order to mitigate the effects of a high degree of non-normality, as measured by Henrici’s metric, either a row or row/column scaling is applied to the $U$ factor during the set-up phase of the AMG $V$-cycle. A finite Neumann series multiplied by a vector is then computed, which is equivalent to a Richardson iteration. Our results demonstrate that several orders of magnitude reduction in the departure from normality $\text{dep}(U)$ is possible, thus leading to robust convergence of GMRES+AMG.

In order to further improve the efficiency of the PeleLM (F)GMRES+AMG pressure solver on many-core GPU architectures, AMG $V$-cycles with ILU smoothing on the finest levels are combined with a polynomial smoother on the remaining coarse levels. It was found that the convergence rates for hybrid AMG are almost identical to using ILU on all levels, thus leading to significant cost reductions. For a large problem $N = 11$ million solved on the NREL Eagle supercomputer, the iterative solve for $LDU$ with row scaling, led to a five times speed-up over the direct solve within the GMRES+AMG $V$-cycles. Furthermore, the strong scaling curve for the solver run time is close to linear.

For the Nalu-Wind pressure continuity equation, an ILUT Schur complement smoother with iterative solves on the local block diagonal systems was applied. Pressure linear systems from NREL 5 MegaWatt reference turbine simulations were employed to assess numerical accuracy and performance. The linear solver exhibits improved parallel strong-scaling characteristics with this new smoother and maintains a constant GMRES iteration count when the number of GPU compute nodes increases. By omitting the residual computations from the single iteration of the GMRES-Schur solver, we are able to reduce the overall execution time. Our future plans include implementing the fixed-point iteration algorithms of Chow and Anzt\cite{17-SC-20-SC} to compute the ILU factorization on GPUs. The Schwarz preconditioners described in Prenter\cite{8} and Jomo et al.\cite{9} could also be adapted to hypre for PeleLM.

ACKNOWLEDGMENTS

This work was authored in part by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Funding was provided by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (ASCR and the NNSA). We also thank the anonymous reviewers for their suggestions, which led to several valuable improvements in this paper.
References

1. Chow E, and Patel A. Fine-Grained Parallel Incomplete LU Factorization. SIAM Journal on Scientific Computing. 2015;37(2):C169–C193.

2. Anzt H, Chow E, and Dongarra J. Iterative sparse triangular solves for preconditioning. In: European conference on parallel processing. Springer; 2015. p. 650–661.

3. Nonaka A, Bell JB, and Day MS. A conservative, thermodynamically consistent numerical approach for low Mach number combustion. I. Single-level integration. Combust Theor Model. 2018;22(1):156–184.

4. Stüben K. Algebraic multigrid (AMG): an introduction with applications. In: Trottenberg U, and Schuller A, editors. Multigrid. USA: Academic Press, Inc.; 2000.

5. Falgout RD, and Meier-Yang U. hypre: A Library of High Performance Preconditioners. In: International Conference on computational science; 2002. p. 632–641.

6. Stüben K. A review of algebraic multigrid. Numerical Analysis: Historical Developments in the 20th Century. 2001;p. 331–359.

7. Chow E, Anzt H, Scott J, and Dongarra J. Using Jacobi iterations and blocking for solving sparse triangular systems in incomplete factorization preconditioning. Journal of Parallel and Distributed Computing. 2018;119:219–230.

8. de Prenter F, Verhoosel CV, van Brummelen EH, Evans JA, Messe C, Benzaken J, et al. Multigrid solvers for immersed finite element methods and immersed isogeometric analysis. Computational Mechanics. 2020;65:807–838.

9. Jomo J, Oztoprak O, de Prenter F, Zander N, Kollmannsberger S, and Rank E. Hierarchical multigrid approaches for the finite cell method on uniform and multi-level hp-refined grids. Computer Methods in Applied Mechanics and Engineering. 2021;386:114075.

10. Van der Sluis A. Condition numbers and equilibration of matrices. Numerische Mathematik. 1969;14(1):14–23.

11. Bauer FL. Optimally scaled matrices. Numerische Mathematik. 1963;5(1):73–87.

12. Chow E, and Saad Y. Experimental study of ILU preconditioners for indefinite matrices. Journal of Computational and Applied Mathematics. 1997;86:387–414.

13. Stewart GW. Matrix Algorithms: Volume 1: Basic Decompositions. SIAM; 1998.

14. Henrici P. Bounds for iterates, inverses, spectral variation and fields of values of non-normal matrices. Numerische Mathematik. 1962;4(1):24–40.

15. Verbeek M, Cullum J, and Joubert W. 2002. AMGToolBox. . Los Alamos National Laboratory.

16. Joubert W, and Cullum J. Scalable Algebraic Multigrid on 3500 processors. Electronic Transactions on Numerical Analysis. 2006;23:105–128.

17. Mullowney P, Li R, Thomas S, Ananthan S, Sharma A, Rood J, et al. Preparing an Incompressible-Flow Fluid Dynamics Code for Exascale-Class Wind Energy Simulations. In: Proceedings of Supercomputing 2021. IEEE/ACM; 2021. p. 1–12.

18. Thomas SJ, Ananthan S, Yellapantula S, Hu JJ, Lawson M, and Sprague MA. A Comparison of Classical and Aggregation-Based Algebraic Multigrid Preconditioners for High-Fidelity Simulation of Wind Turbine Incompressible Flows. SIAM Journal on Scientific Computing. 2019;41(5):S196–S219.

19. Ruge JW, and Stüben K. Algebraic multigrid. In: Multigrid methods. SIAM; 1987. p. 73–130.

20. Falgout RD, Jones JE, and Yang UM. The Design and Implementation of hypre, a Library of Parallel High Performance Preconditioners. In: Bruaset AM, and Tveito A, editors. Numerical Solution of Partial Differential Equations on Parallel Computers. Berlin, Heidelberg: Springer Berlin Heidelberg; 2006. p. 267–294.
21. Chow E, and Vassilevski PS. Multilevel block factorizations in generalized hierarchical bases. Numerical Linear Algebra with Applications. 2003;10(1-2):105–127.

22. Saad Y. Iterative Methods for Sparse Linear Systems, 2nd Ed. SIAM; 2003.

23. Xu T, Li R, and Osei-Kuffuor D. A two-level GPU-accelerated incomplete LU preconditioner for general sparse linear systems. Numer Linear Algebra Appl. 2020; Submitted for publication.

24. Falgout RD, Li R, Sjögreen B, Wang L, and Yang UM. Porting hypre to heterogeneous computer architectures: Strategies and experiences. Parallel Computing. 2021;108:102840. Available from: https://www.sciencedirect.com/science/article/pii/S0167819121000867.

25. Ruhe A. On the closeness of eigenvalues and singular values for almost normal matrices. Linear Algebra and its Applications. 1975;11(1):87–93.

26. Trefethen L, and Embree M. The behavior of nonnormal matrices and operators. Spectra and Pseudospectra. 2005.

27. Ipsen IC. 1998. A note on the field of values of non-normal matrices. North Carolina State University. Center for Research in Scientific Computation.

28. Elsner L, and Paardekooper M. On measures of nonnormality of matrices. Linear Algebra and its Applications. 1987;92:107–123.

29. Eiermann M. Fields of values and iterative methods. Linear Algebra and its Applications. 1993;180:167–197. Available from: https://www.sciencedirect.com/science/article/pii/0024379593905302.

30. Knight PA, Ruiz D, and Uçar B. A symmetry preserving algorithm for matrix scaling. SIAM Journal on matrix analysis and applications. 2014;35(3):931–955.

31. Li R, and Zhang C. Efficient Parallel Implementations of Sparse Triangular Solves for GPU Architecture. In: Proceedings of SIAM Conference on Parallel Proc. for Sci. Comput.; 2020. p. 118–128.

32. Naumov M. 2011. Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the GPU. Tech. Rep. NVR-2011.

33. Ciar A, de Sturler E, and Gugercin S. Preconditioning Parametrized Linear Systems. SIAM Journal on Scientific Computing. 2021;43(3):A2242–A2267.

34. Davis TA, and Hu Y. The University of Florida sparse matrix collection. ACM Transactions on Mathematical Software (TOMS). 2011;38(1):1–25.

35. Musser J, Almgren AS, Fullmer WD, Antepara O, Bell JB, Blaschke J, et al. MFIX-Exa: A path toward exascale CFD-DEM simulations. The International Journal of High Performance Computing Applications. 2022;36(1):40–58.

36. Saad Y, and Schultz MH. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM Journal on scientific and statistical computing. 1986;7(3):856–869.

37. Baker AH, Falgout RD, Kolev TV, and Yang UM. Multigrid Smoothers for Ultraparallel Computing. SIAM J Sci Comput. 2011;33:2864–2887.

38. Baker AH, Falgout RD, Kolev TV, and Meier-Yang U. In: Scaling Hypre’s Multigrid Solvers to 100,000 Cores. London: Springer London; 2012. p. 261–279.

39. Mantuuffel T, McCormick S, Park M, and Ruge J. Operator-based interpolation for bootstrap algebraic multigrid. Numerical Linear Algebra with Applications. 2010;17(2-3):519–537.

40. De Sterck H, Falgout RD, Nolting JW, and Meier-Yang U. Distance-two interpolation for parallel algebraic multigrid. Numerical Linear Algebra with Applications. 2008;15(2-3):115–139.
41. Li R, Sjogreen B, and Meier-Yang U. A new class of AMG interpolation operators based on matrix matrix multiplications. To appear SIAM Journal on Scientific Computing. 2020;

42. Zhang W, and et al. AMReX: A Framework for Block-Structured Adaptive Mesh Refinement. Journal of Open Source Software. 2019;4(37):1370.

43. Jonkman J, Butterfield S, Musial W, and Scott G. 2009. Definition of a 5-MW reference wind turbine for offshore system development. NREL/TP-500-38060. National Renewable Energy Laboratory.

44. Sprague MA, Ananthan S, Vijayakumar G, and Robinson M. ExaWind: A multifidelity modeling and simulation environment for wind energy. Journal of Physics: Conference Series. 2020;1452.

45. Yang UM. On long range interpolation operators for aggressive coarsening. Numerical Linear Algebra with Applications. 2010;17:453–472.

46. Anzt H, Ribizel T, Flegar G, Chow E, and Dongarra J. ParILUT - A Parallel Threshold ILU for GPUs. In: International Conference on Parallel Processing and Applied Mathematics. Springer; 2011. p. 133–142.

**AUTHOR BIOGRAPHY**

**How to cite this article:** Thomas S. J., Carr, A. K., Mullowney, P. Świrydowicz K., and Day, M. (2022), ILU Smoothers for AMG with Scaled Triangular Factors, NAME, VOL.