Check for updates

ACM DIGITAL LIBRARY

acm open

Latest updates: https://dl.acm.org/doi/10.1145/3712285.3759882

RESEARCH-ARTICLE

DAS-ILU: A Distributed Asynchronous Parallel ILU Factorization Based on Domain Decomposition

FAN YUAN, National University of Defense Technology China, Changsha, Hunan, China

SHENGGUO LI, National University of Defense Technology China, Changsha, Hunan, China

XIAOJIAN YANG, National University of Defense Technology China, Changsha, Hunan, China

YUNQING HUANG, Xiangtan University, Xiangtan, Hunan, China

HONGXIA WANG, National University of Defense Technology China, Changsha, Hunan, China

CHUANFU XU, National University of Defense Technology China, Changsha, Hunan, China

View all

Open Access Support provided by:

Xiangtan University

National University of Defense Technology China

China Ship Scientific Research Center

PDF Download
3712285.3759882.pdf
22 February 2026
Total Citations: 1
Total Downloads: 1932

Published: 16 November 2025

Citation in BibTeX format

SC '25: The International Conference for High Performance Computing, Networking, Storage and Analysis
November 16 - 21, 2025
MO, St. Louis, USA

Conference Sponsors:
SIGHPC

https://doi.org/10.1145/3712285.3759882
ISBN: 9798400714665



DAS-ILU: A Distributed Asynchronous Parallel ILU Factorization Based on Domain Decomposition

Fan Yuan
National University of Defense Technology (NUDT)
Changsha, China
Xiangtan University
Xiangtan, China
fyuan@smail.xtu.edu.cn

Shengguo Li*
National University of Defense Technology (NUDT)
Changsha, China
nudtlsg@nudt.edu.cn

Xiaojian Yang
National University of Defense Technology (NUDT)
Changsha, China
yangxj@nudt.edu.cn

Yunqing Huang
Xiangtan University
Xiangtan, China
huangyq@xtu.edu.cn

Hongxia Wang
National University of Defense Technology (NUDT)
Changsha, China
wanghongxia@nudt.edu.cn

Chuanfu Xu
National University of Defense Technology (NUDT)
Changsha, China
xuchuanfu@nudt.edu.cn

Dezun Dong
National University of Defense Technology (NUDT)
Changsha, China
dong@nudt.edu.cn

Tiejun Li
National University of Defense Technology (NUDT)
Changsha, China
tjli@nudt.edu.cn

Jianchun Wang
China Ship Scientific Research Center
Wuxi, China
wangjianchun@cssrc.com.cn

Jie Liu
National University of Defense Technology (NUDT)
Changsha, China
liujie@nudt.edu.cn

Abstract

This paper presents DAS-ILU, a Distributed Asynchronous parallel Incomplete LU factorization method based on domain decomposition. DAS-ILU partitions the computational domain into independently processed interior nodes and asynchronously updated separator nodes, thereby reducing cross-processor dependencies and halving the separator size compared to conventional methods. To further improve performance, it employs optimized data exchange patterns to minimize communication overhead and extends support to block-structured sparse matrices via exact block inversions. Comprehensive evaluations on a range of problem types—including structural mechanics, computational fluid dynamics, and reservoir simulation demonstrate the superior performance of DAS-ILU. Compared to state-of-the-art ILU implementations, DAS-ILU achieves solve time speedups of up to 2.07× over Chow-Patel's fine-grained parallel ILU and up to 4.11× over HYPRE's ILU. Moreover, DAS-ILU exhibits strong robustness when applied to challenging nonsymmetric and indefinite systems.

CCS Concepts

• **Mathematics of computing** → **Solvers**; • **Computing methodologies** → **Parallel algorithms**; **Distributed algorithms**.

Keywords

ILU factorization, domain decomposition, asynchronous iteration, parallel preconditioning, sparse linear systems

ACM Reference Format:
Fan Yuan, Shengguo Li, Xiaojian Yang, Yunqing Huang, Hongxia Wang, Chuanfu Xu, Dezun Dong, Tiejun Li, Jianchun Wang, and Jie Liu. 2025. DAS-ILU: A Distributed Asynchronous Parallel ILU Factorization Based on Domain Decomposition. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3712285.3759882

*Corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SC '25, St Louis, MO, USA

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-1466-5/25/11
https://doi.org/10.1145/3712285.3759882

1 Introduction

Incomplete LU (ILU) factorization is a widely used preconditioning technique for solving large sparse linear systems arising in computational fluid dynamics, structural mechanics, and reservoir

simulations [9, 34, 36]. By approximating the LU decomposition while selectively discarding fill-in entries, ILU strikes a balance between numerical robustness and computational efficiency. However, the inherent row-wise dependencies of classical ILU algorithms, rooted in Gaussian elimination, pose significant challenges for parallelization on modern distributed and multicore architectures.

To address these challenges, various parallel ILU strategies have been proposed. Techniques such as level scheduling [19, 36] and multicolor reordering [22, 30], aim to expose parallelism by altering the matrix order. Level scheduling partitions matrix rows into levels that can be processed concurrently, but parallel efficiency diminishes with irregular matrices or deep dependency chains. Multicolor reordering assigns non-interacting colors to rows or columns, enabling simultaneous updates within a color group. However, such reordering often disrupts matrix structure, degrading preconditioner quality and increasing iteration counts—for example, by up to 50% in Laplacian problems [14, Table 5.3].

Domain decomposition methods offer an alternative by dividing the global domain into subdomains, enabling coarse-grained parallelism suitable for distributed memory platforms [4, 17, 25]. While these methods apply ILU locally [39], they introduce coupling challenges at subdomain interfaces, necessitating synchronization or approximation that can hinder scalability and convergence. Recent efforts have explored fine-grained parallelism and asynchronous iterative schemes to circumvent the limitations of traditional ILU. Notably, Chow and Patel [10] reformulated ILU as a nonlinear fixed-point problem, enabling asynchronous updates. Although this approach shows promise for shared-memory systems, it suffers from communication overhead and limited scalability on distributed-memory platforms due to frequent global synchronization [2]. Hybrid methods combining domain decomposition with asynchronous execution remain underexplored, particularly in mitigating the serialization bottleneck at subdomain interfaces.

In this work, we propose **DAS-ILU**, a distributed asynchronous parallel ILU factorization method that combines domain decomposition with a communication-aware asynchronous fixed-point iteration. We introduce a domain decomposition strategy that partitions the computational domain into interior and separator regions, thereby minimizing cross-processor dependencies. Interior rows are processed independently using conventional ILU factorization without requiring communication, while separator rows are approximated through a fixed-point iteration with asynchronous communication. To further reduce communication overhead, we develop a asynchronous Jacobi iterations for forward and backward substitutions, leveraging optimized data exchange patterns. For block-structured sparse matrices arising from PDE-coupled systems, we extend the scalar ILU to a point-block ILU factorization by employing block CSR storage and exact block inversions to account for local couplings. Additionally, DAS-ILU is integrated with algebraic multigrid (AMG) as a smoother, further accelerating convergence for large-scale problems.

Comprehensive experimental evaluations on benchmark matrices from the SuiteSparse collection [13], convection-diffusion PDEs, and the SPE10 reservoir model [11] demonstrate that DAS-ILU delivers superior performance and scalability. Compared to the Chow–Patel and the state-of-the-art ILU implementation in HYPRE [17], DAS-ILU consistently achieves faster solve times, with speedups of up to 1.47× and 1.43× for convection–diffusion problems on distributed systems with more than 3,500 cores. Furthermore, numerical results for the symmetric and nonsymmetric matrices from SuiteSparse [12] show that DAS-ILU achieves solve time speedups of up to 2.07× over Chow–Patel and up to 4.11× over HYPRE. Moreover, DAS-ILU maintains its robustness when applied to challenging nonsymmetric and indefinite systems.

The key contributions include:

(1) **Efficient Domain Decomposition**: Partitions the computational domain into interior and separator nodes, minimizing cross-processor dependencies while reducing separator size by approximately 50%.
(2) **Communication-aware Asynchronous Iteration**: Integrates domain decomposition with asynchronous communication while avoiding explicit Schur complement construction, utilizing parallel factorization for interior rows and fixed-point iterations for separator rows, leveraging optimized data exchange patterns to minimize overhead.
(3) **Block-structured Support**: Extends support to block-structured matrices using exact block inversions, improving efficiency for coupled PDE systems.

Algorithm 1 Conventional ILU Factorizations

for $i = 1, \cdots, n$ **do**
  for $k = 1, \cdots, i-1$ and $(i,k) \in S$ **do**
    $a_{ik} = a_{ik}/a_{kk}$
    for $j = k+1, \cdots, n$ and $(i,j) \in S$ **do**
      $a_{ij} = a_{ij} - a_{ik}a_{kj}$
    end for
  end for
end for

Algorithm 2 Fine-grained parallel ILU

Set unknowns l_{ij} and u_{ij} to initial values
for sweep = 1, 2, ... until convergence **do**
  parallel for $(i,j) \in S$ **do**
    if $i > j$ **then**
      $l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}\right)/u_{jj}$
    else
      $u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}$
    end if
  end for
end for

2 Background

2.1 Conventional ILU algorithm

Given a sparse matrix A, the incomplete LU (ILU) factorization constructs sparse lower and upper triangular matrices L and U through an approximate decomposition process, maintaining $LU \approx A$ under a prescribed sparsity pattern S. The sparsity pattern S can be determined a priori through symbolic analysis or adaptively during numerical computation.

As shown in Algorithm 1, the conventional approach mimics Gaussian elimination with fill-in restrictions [33]. This sequential process in rows creates inherent dependencies in which each update

of the row requires previously computed entries, making direct parallelization challenging. The subsequent triangular solves are also subject to this sequential dependency.

Common types of ILU factorization include ILU(k) (level-of-fill), ILUT (threshold-based), and ILUP (ILU with pivoting), among others; see [33] for a comprehensive overview.

2.2 Fine-grained parallel ILU

Chow and Patel [10] proposed a fine-grained parallel ILU factorization algorithm in reinterpreting ILU as a nonlinear fixed-point problem. For a given sparse matrix A with sparsity pattern S, the ILU factorization must satisfy the element-wise consistency condition:

$$(LU)_{ij} = A_{ij}, \forall (i,j) \in S. \tag{1}$$

This constraint induces a nonlinear system of equations governing the lower triangular entries l_{ij} and upper triangular entries u_{ij}, with explicit expressions:

$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right) \quad (i > j) \tag{2}$$

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \quad (i \leq j) \tag{3}$$

By vectorizing the unknowns l_{ij} and u_{ij} into $x \in R^m$, where $m = nnz(S)$, the system admits a fixed-point representation

$$x = G(x), \tag{4}$$

where G denotes the nonlinear update operator. Chow and Patel present an asynchronous implementation of this nonlinear fixed-point iteration in Algorithm 2. Each fixed-point iteration updating all the unknowns of x is called a "sweep". Compared to the conventional algorithm shown in Algorithm 1, Algorithm 2 is parallelized across the elements of S, and the elements can also be updated asynchronously. The convergence of this new algorithm is proved under certain conditions in [10], and numerical results show that only a few sweeps of the factorization are sufficient to construct an effective preconditioner. In this work, we implement the Chow–Patel ILU algorithm on a distributed-memory platform within the YHAMG framework [18].

2.3 Domain Decomposition Methods

Domain decomposition (DD) methods partition the computational domain into smaller subdomains that can be processed in parallel, enabling efficient utilization of distributed computing resources for large-scale linear systems. The domain decomposition framework partitions the computational domain into p non-overlapping subdomains, leading to a block-structured matrix representation:

$$A = \begin{pmatrix} A_1 & E_{12} & \cdots & E_{1p} \\ E_{21} & A_2 & \cdots & E_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ E_{p1} & E_{p2} & \cdots & A_p \end{pmatrix}, \tag{5}$$

where A_i denotes the submatrix associated with the i-th subdomain, while E_{ij} represents the coupling between subdomains i and j.

One of the simplest ways to precondition (5) is to use the *one-level* block Jacobi approach, where ILU factorization is applied independently to each subdomain matrix A_i. While computationally efficient, this approach often suffers from poor convergence due to neglecting inter-subdomain couplings.

Two-Level ILU Approaches. To construct more efficient preconditioners, *two-level* approaches further exploit the DD framework. Each subdomain matrix A_i is reordered into interior and interface nodes:

$$A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix}, \tag{6}$$

where B_i contains the interior nodes, C_i contains the interface nodes, and E_i, F_i represent cross-term couplings. Interior nodes are connected only to other nodes within the same subdomain, while interface nodes have connections to nodes in other subdomains.

The global coefficient matrix A can be written as

$$A = \begin{pmatrix} B & F \\ E & C \end{pmatrix} = \begin{pmatrix} I & 0 \\ EB^{-1} & I \end{pmatrix} \begin{pmatrix} B & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I & B^{-1}F \\ 0 & I \end{pmatrix}, \tag{7}$$

where $B = \mathrm{diag}\{B_1, \cdots, B_p\}$ is block diagonal, and $S = C - EB^{-1}F$ is the Schur complement matrix capturing interface interactions [38].

The ILU factorization is applied to approximate each local subdomain matrix B_i:

$$B_i \approx L_i U_i, \tag{8}$$

where L_i and U_i are lower and upper triangular matrices, respectively.

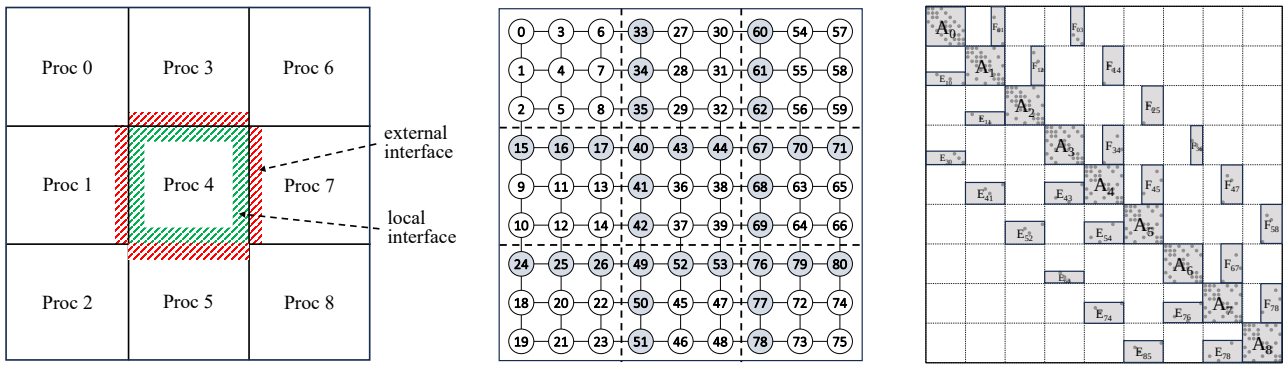
The parallel ILU strategy in HYPRE employs a two-level approach [39], consisting of ILU solves within local subdomain blocks and a global solve involving the Schur complement system. However, the explicit formation and factorization of the Schur complement S can be computationally expensive and memory-intensive, particularly for problems with large interface regions. Other DD-based ILU methods, such as those in [5, 7, 8], also rely on explicit Schur complement construction, leading to similar computational and storage challenges.

3 Distributed Asynchronous ILU Factorization

3.1 Domain Decomposition and Ordering

Our method employs a domain decomposition framework that partitions the computational domain into multiple subdomains using graph partitioning techniques, with each subdomain assigned to a processor such that subdomain numbering aligns with processor rank. As shown in Figure 1(a), the computational domain is evenly distributed on nine processors. In each subdomain, *local interface* points denote points within a local subdomain that couple to an external subdomain, and *external interface* points represent points outside subdomain that couple to a local interface point. For example, in Figure 1(a), the green and red areas are the *local interface* and *external interface* points of Proc 4, respectively.

ILU decomposition employs Gaussian elimination to progressively eliminate the lower triangular entries, and the resulting data dependencies inherently limit inter-processor communication to a lower triangular communication pattern. Specifically, each subdomain receives external interface data only from its neighboring subdomains that are ordered before it, which we designate as lower-rank neighbors. Correspondingly, each subdomain transmits its *local interface* data to the neighboring subdomains that are ordered after it, referred to as higher-rank neighbors.



(a) Domain Decomposition (b) Classification for Grid Points (c) Associated Matrix

Figure 1: Domain decomposition and classification of grid points. In (a), the computational domain is divided into 9 subdomains. In (b), grid points are classified into separator (gray) and interior (white) regions, and sorted accordingly. (c) shows the corresponding matrix structure.

Furthermore, we divide each subdomain into two topological regions: the separator and the interior. The separator (gray grid points in Figure 1(b)) comprising *local interface* points adjacent to lower-rank subdomains, and the interior (white grid points) containing all remaining points. Notably, *local interface* points bordering higher-rank subdomains are classified as interior nodes, whereas their external counterparts constitute the separator. Compared to the conventional DD framework [39], our partitioning approach yields larger interior regions, thereby offering improved parallelism.

Within each subdomain, the grid points are numbered in such a way that the interior points precede the separator points. This ordering strategy ensures two fundamental principles: in each processor, (1) local interior points are not coupled with external interior points, but only with adjacent separator points (local or from higher-rank neighbors), and (2) their indices are always smaller than those of adjacent separator points. The associated pattern-symmetric matrix structure under this ordering is demonstrated in Figure 1(c).

3.2 Distributed Asynchronous ILU

To clarify our distributed asynchronous ILU method, we begin by introducing the local data structure. Figure 2 shows the local structure of the matrix in Figure 1(c). The local matrix for each subdomain can be partitioned into four distinct blocks: L, U, E, and F. Here, L and U denote the lower and upper triangular parts of the local coupling matrix within the subdomain, respectively. The block E corresponds to the couplings with lower-rank neighboring subdomains, while F corresponds to the couplings with higher-rank neighboring subdomains. Crucially, E does not contain nonzero entries in rows corresponding to interior points, as these points have no dependencies on preceding subdomains. R stores the factored rows corresponding to the column of E from lower-rank neighbors.

For interior rows, ILU factorization can proceed independently for these interior rows without communication between subdomains. In contrast, due to the dependencies on lower-rank neighbors, separator rows require sequential processing that follows three phases: (1) Receiving and integrating factored *external interface* rows R from lower-rank neighbors. (2) Eliminating the E block using the pivots from R, and performing local ILU factorization on the separator rows. (3) Packing and sending completed *local interface* rows to higher-rank neighbors. This dependency chain introduces an unavoidable serialization bottleneck, particularly problematic for large-scale distributed systems where asynchronous execution is preferred.

To overcome this limitation, we implement a distributed asynchronous iterative scheme, as shown in Algorithm 3. Initially, each processor performs the conventional ILU factorization for interior rows in parallel. To solve the system for separator rows, we first construct initial approximations using the original matrix entries from A. Subsequently, we employ a block version of the fixed-point iteration scheme to obtain the approximate factorization. In each iteration, processors simultaneously receive the *external interface* rows R from lower-rank neighbors and send current *local interface* rows to higher-rank neighbors. With block E eliminated using the most recently received matrix R, we can perform independent ILU factorization on the separator rows. This process is repeated until convergence.

As shown in Figure 3, process 4 first performs ILU factorization on interior rows 36-39. During iterative processing of separator rows 40-44, the algorithm concurrently receives factorized rows 17, 13, 14 from Proc 1 and rows 35, 29, 32 from Proc 3, while simultaneously transmitting local interface rows 42, 37, 39 to Proc 5 and rows 44, 38, 39 to Proc 7. The received rows serve as pivots for eliminating their corresponding couplings in the E block components identified in Figure 2.

A key advantage of our method is the significant reduction in communication overhead after the initial sweep. The interface points that are also interior points (such as {13, 14, 29, 32, 37, 38, 39} in Figure 3, marked with dotted circles) only participate in communication during the first sweep. This is because interior rows undergo exact ILU factorization at the initial stage. In subsequent iterations, communication is streamlined to focus solely on separator points ({40, 41, 42, 43, 44} in this example), substantially decreasing inter-process data exchange.

The efficiency of our approach is particularly evident in large-scale problems. For instance, in a 3D problem with total size $O(k^3)$, the separator size and first-sweep communication volume are $O(k^2)$,

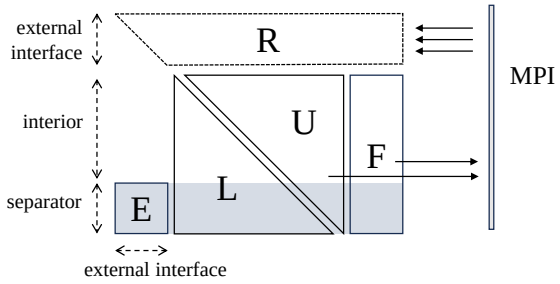


Figure 2: The local ILU data structure contains four matrix blocks: L (lower triangular) and U (upper triangular) representing intra-subdomain couplings, E for couplings with lower-rank neighboring subdomains, and F for couplings with higher-rank neighbors. R stores the factored external interface rows received from lower-rank neighbors.

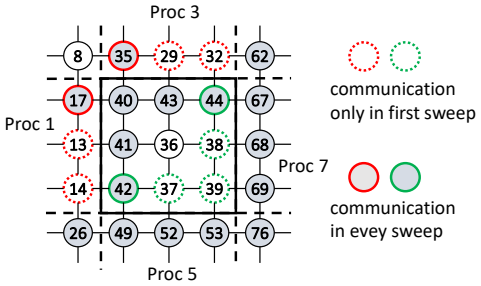


Figure 3: Local grid in Proc 4. Red grid points denote external interface rows received from lower-rank neighbors, while green grid points indicate local interface rows sent to higher-rank neighbors. Interface points with dotted circles represent communication only in the first iteration.

Algorithm 3 Distributed Asynchronous ILU Factorization

Require: Local matrix structure:

$$A_{\text{local}} = \begin{bmatrix} R_{\text{ext}_1} & R_{\text{loc}} & R_{\text{ext}_2} \\ 0 & A_{\text{int}} & F_{\text{int}} \\ E & A_{\text{sep}} & F_{\text{sep}} \end{bmatrix} \quad \begin{array}{l} \triangleright \text{ External interface (from lower-rank neighbors)} \\ \triangleright \text{ Interior rows} \\ \triangleright \text{ Separator rows} \end{array}$$

Ensure: Approximate ILU factors $\tilde{E}, \tilde{L}, \tilde{U}, \tilde{F}$

1: Perform ILU factorization for interior rows: $[\tilde{L}_{\text{int}}, \tilde{U}_{\text{int}}, \tilde{F}_{\text{int}}] = \text{ilu}([A_{\text{int}}\ F_{\text{int}}])$
2: Initialize separator: $\tilde{L}_{\text{sep}}^{(0)}, \tilde{U}_{\text{sep}}^{(0)} \leftarrow \textsc{Split}(A_{\text{sep}})$ and $\tilde{F}_{\text{sep}}^{(0)} \leftarrow F_{\text{sep}}$
3: **for** $k = 1, 2, \ldots$ until convergence **do**
4: Send factorized local interface rows to higher-rank neighbors : $[\tilde{U}_{\text{sep}}^{(k-1)}\ \tilde{F}_{\text{sep}}^{(k-1)}]$, and $[\tilde{U}_{\text{int}}\ \tilde{F}_{\text{int}}]$ for only first sweep
5: Receive external interface data from lower-rank neighbors : $[\tilde{R}_{\text{ext}_1}^{(k-1)}\ \tilde{R}_{\text{loc}}^{(k-1)}\ \tilde{R}_{\text{ext}_2}^{(k-1)}]$
6: Eliminate coupling block: $\tilde{E}^{(k)} \approx E(R_{\text{ext}_1}^{(k-1)})^{-1}$
7: Update separator rows: $[\hat{A}_{\text{sep}}\ \hat{F}_{\text{sep}}] \approx [A_{\text{sep}}\ F_{\text{sep}}^{(k)}] - \tilde{E}^{(k)}[\tilde{R}_{\text{loc}}^{(k-1)}\ \tilde{R}_{\text{ext}_2}^{(k-1)}]$
8: Perform ILU factorization on updated separator rows: $[\tilde{L}_{\text{sep}}^{(k)}, \tilde{U}_{\text{sep}}^{(k)}, \tilde{F}_{\text{sep}}^{(k)}] = \text{ilu}([\hat{A}_{sep}\ \hat{F}_{\text{sep}}])$
9: **end for**

while subsequent sweeps reduce communication volume to $O(k)$. Our extensive testing reveals that typically 2-5 sweeps are required to construct an effective ILU factorization.

3.3 Distributed Asynchronous Triangular Solves

Once the factorization is complete, applying the ILU preconditioner involves solving triangular systems. In each processor, the systems can be expressed as

$$Ly + Ey_{ext} = b, \tag{9}$$

$$Ux + Fx_{ext} = y, \tag{10}$$

where y_{ext} and x_{ext} represent external interface data for the distributed solution process. The vector y_{ext} corresponds to the external interface data aligned with the columns of block E, which is received from lower-rank neighboring processors during the forward substitution phase. Similarly, x_{ext} denotes the external interface data associated with the columns of block F, obtained from higher-rank neighboring processors during the backward substitution phase.

Algorithm 4 Distributed asynchronous solve for $Ly + Ey_{ext} = b$

1: Solve $y^{(0)} = L^{-1}b$ for interior rows.
2: **for** $k = 1, 2, \ldots$ until convergence **do**
3: Receive external interface data y_{ext} from lower-rank neighbors and send local interface data to higher-rank neighbors.
4: Update $y^{(k)} = L^{-1}(b - Ey_{ext}^{(k-1)})$ for separator rows.
5: **end for**

The forward substitution in Equation (9) can be resolved through an iterative scheme. Specifically, we consider a block Jacobi iteration as following:

$$Ly^{k+1} = b - Ey_{ext}^{k} \tag{11}$$

with initial condition $y^0 = 0$. Observing that interior points are fully decoupled from external dependencies (block E has no nonzero in interior rows), they can be solved directly during the initial iteration. So only the separator variables require iterative computation in subsequent steps.

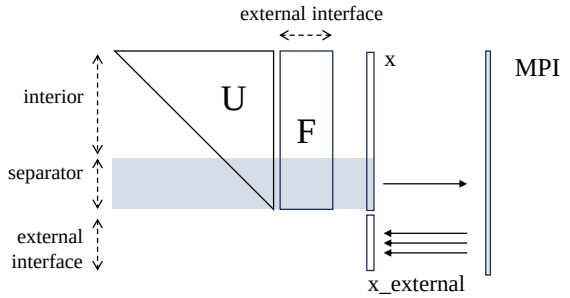


Figure 4: Local data structure and communication pattern for backward substitution. The vector $x_{external}$ stores the externally received values of x from higher-rank neighbors, aligned with the column indices of matrix F.

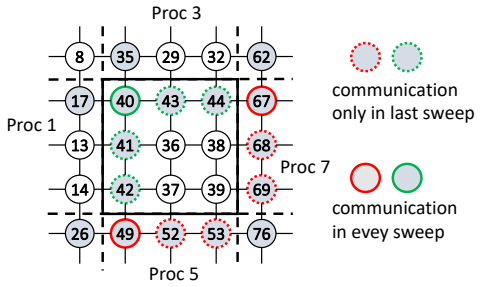


Figure 5: Local grid in Proc 4. Red and green points denote data received from higher-rank neighbors, and data sent to lower-rank neighbors, respectively. Dotted circles indicate communication only required in last sweep.

Algorithm 5 Distributed asynchronous solve for $Ux + Fx_{ext} = y$

1: Solve $x^{(0)} = U^{-1}y$.
2: **for** k = 1, 2, ... until convergence **do**
3: Receive interface data from higher-rank neighbors $x_{ext}^{(k-1)}$ and send interface data to lower-rank neighbors.
4: Update $x^{(k)} = U^{-1}(y - Fx_{ext}^{(k-1)})$ for separator rows.
5: **end for**
6: Update $x^{(*)} = U^{-1}(y - Fx_{ext}^{(*)})$ for interior rows.

Similarly, we propose a distributed lower triangular solve algorithm, detailed in Algorithm 4. This process also leverages the communication pattern illustrated in Figure 3, where interface points that are also interior points (corresponding to the points marked with dotted circles) only need to communicate once, as their values will not change with iterations.

The backward substitution of Equation (10) employs a similar asynchronous communication pattern, but with reversed data dependency directions. The iterative scheme can be formally expressed as:

$$Ux^{k+1} = y - Fx_{ext}^{k} \tag{12}$$

initialized with $x^0 = 0$. In backward substitution, the separator does not depend on the interior points (it comes after the interior in order). Consequently, interior variables can be resolved in a final parallel step after separator iterations converge.

The distributed asynchronous upper triangular solve algorithm is detailed in Algorithm 5. In particular, the communication pattern illustrated in Figure 4 and Figure 5 reveals optimized data exchange patterns.

3.4 Convergence Analysis

Since DAS-ILU is equivalent to Chow-Patel ILU under our ordering scheme, it inherits the same convergence properties. We provide formal convergence guarantees for both the factorization and triangular solve phases.

Asynchronous Factorization: The DAS-ILU algorithm converges to the traditional ILU factorization within at most $p - 1$ fixed-point iterations, where p is the total number of domain partitions. This can be proven by mathematical induction: partition P_0 has no couplings with lower-rank neighbors, so all its rows converge after the first phase without requiring fixed-point iterations. If partitions $P_0, \ldots, P_k$ have converged after the k-th iteration, then partition P_{k+1} receives converged external interface rows R in the $(k + 1)$-th iteration, making its local computation identical to traditional ILU factorization. By induction, all p partitions converge within $p - 1$ iterations.

Triangular Solve: Both asynchronous forward and backward substitution algorithms converge to the exact triangular solve within at most $p-1$ iterations. For forward substitution, partition P_0 solves its separator components exactly in the first iteration since it has no external dependencies from lower-rank neighbors. By similar inductive reasoning, if partitions $P_0, \ldots, P_k$ have converged after k iterations, then P_{k+1} receives converged data and solves exactly in iteration $k + 1$. The same analysis applies to backward substitution with reversed dependency order.

3.5 Comparison with Other ILU

Our DAS-ILU method leverages domain decomposition techniques to partition the computational domain into interior and separator regions, exploiting their inherent topological properties. This approach enables direct parallel processing of interior rows, while separator rows are computed through an efficient fixed-point iteration scheme to approximate the ILU factors.

Compared to the fine-grained parallel ILU methods [10] proposed by Chow and Patel, DAS-ILU offers significant advantages. It confines the fixed-point iteration and Jacobi iteration for application to a small subset of rows (separators), which typically constitute a minor fraction of the total domain, thus substantially reducing overall computation time. Moreover, after the first sweep, DAS-ILU dramatically decreases communication overhead in subsequent iterations, which is particularly beneficial in large-scale distributed computing environments where communication costs can be a major bottleneck.

In contrast to conventional domain decomposition-based ILU methods [5, 7, 8, 39], DAS-ILU achieves a minimization of the separator region size, resulting in a separator size approximately half that of general domain decomposition ILU methods. More importantly, a key distinguishing feature of DAS-ILU is that it employs asynchronous fixed-point iterations that implicitly handle the interface coupling, eliminating the computational and memory costs

associated with explicit Schur complement construction and storage. These two advantages make DAS-ILU particularly well-suited for problems with large interface regions where traditional Schur complement-based methods become prohibitively expensive.

3.6 Point-block Preconditioner

In coupled PDE systems, the Jacobian matrices inherently exhibit block structures, with each block representing intra-cell variable couplings. Leveraging this structure, we employ point-blocking, a technique that enables exact inversion of small dense blocks. This approach efficiently resolves local coupling between different physical variables at each mesh point, effectively extending scalar PDE iterations to systems of PDEs [23].

To harness these advantages, we adopt a block compressed row storage (Block-CSR) format, which contiguously stores entries within each block, treating them as small dense matrices. This block-wise representation enables algorithm-level optimizations where scalar operations in conventional ILU are systematically elevated to block operations, as detailed in Algorithm 1.

During factorization, diagonal blocks A_{kk} undergo in-place LU decomposition with partial pivoting, retaining both L and U factors, followed by two triangular solve (TRSM) operations to compute $A_{ik} = A_{ik}A_{kk}^{-1}$. Notably, for small dense blocks, manually unrolled loop implementations with explicit vectorization intrinsics may outperform BLAS library calls. Off-diagonal updates $A_{ij} = A_{ij} - A_{ik}A_{kj}$ employ similar vectorization strategies. During triangular solving phases, we achieve exact block matrix inversions through TRSV operations using the stored L and U factors.

4 Experimental Setup

4.1 Evaluation Platforms

We conducted our experiments on a high-performance computing cluster consisting of 64 nodes. Each node is equipped with dual Intel Xeon Gold 6348 processors, totally 56 CPU cores, operating at a base frequency of 2.60 GHz, and 128GB of DDR4-3200 memory. The complete system provides an aggregate compute capacity of 3,584 CPU cores.

The software environment consists of Kylin Linux operating system, MPICH-4.1.2, and the Intel 2023 compiler (with -O3 optimization flag enabled). Domain decomposition and load balancing are implemented using ParMETIS 4.0.3. To ensure reproducibility, each test was repeated five times, and we report the average results.

4.2 Competing Baselines

We implemented our DAS-ILU algorithm in the YHAMG algebraic multigrid package [18, 41] and integrated it as a smoother in the AMG cycle. For comprehensive evaluation, we compare DAS-ILU against three baseline methods:

Serial: A conventional sequential ILU implementation with block structure support.

Chow-Patel: Our MPI-based implementation of the fine-grained parallel ILU by Chow and Patel [10], which employs traditional ILU factorization for on-processor blocks and fixed-point iterations for off-processor blocks, with block structure support.

HYPRE: The open-source HYPRE library (v2.32.0) [17] with domain decomposition-based ILU implementation, supporting ILUK and ILUT variants and algebraic multigrid solver. Block structure support is not implemented.

Table 1: Test matrices from SuiteSparse collection. Janna group matrices are symmetric positive definite with block size 3, while Fluorem group matrices are nonsymmetric and indefinite with block size 7.

Group	Matrix	n	nnz	SPD	n_b
Janna	Hook_1498	1,498,023	59,374,451	yes	3
	Flan_1565	1,564,794	114,165,372	yes	3
	Geo_1438	1,437,960	60,236,322	yes	3
	Emilia_923	923,136	40,373,538	yes	3
Fluorem	PR02R	161,070	8,185,136	no	7
	RM07R	381,689	37,464,962	no	7

The experimental configuration deploys 56 MPI processes per node. Matrix partitioning uses ParMETIS to optimize spatial locality. All baselines employ Reverse Cuthill-McKee (RCM) reordering within local subdomains, while DAS-ILU implements RCM reordering for local interior and separator blocks separately.

For practical problems with complex matrix structures, ILU(0) [33] may exhibit limited convergence, so our implementation uses ILU(1) as the baseline configuration. For severely challenging ill-conditioned matrices, we extend to ILU(2). The filling level is precomputed to determine the sparse pattern structure, followed by static-pattern ILU decomposition. Both reordering and pattern calculations are included in the preconditioner setup time metrics.

4.3 Evaluation Case

We selected sparse matrices from SuiteSparse for evaluation, including six matrices from the Janna and Fluorem groups. The matrices from the Janna group [20, 21] are derived from finite element simulations of gas reservoir dynamics and structural mechanics problems. These matrices are symmetric and positive definite, verifying the effectiveness of the ILU-preconditioned PCG solver. The matrices from the Fluorem group [28] model compressible Navier-Stokes equations. These matrices are nonsymmetric and indefinite, requiring FGMRES [31] or BiCGSTAB [37] solvers. Given the scale of these matrices, we performed tests using 56 processes on a single node.

For multi-node scalability evaluation, we focused on convection-diffusion equations and the SPE10 benchmark model [11] with hundreds of millions of unknowns. These large-scale problems were solved using up to 64 compute nodes, evaluating FGMRES with both ILU and AMG (using ILU smoothing) preconditioners.

The convergence tolerance is set to a relative drop of 1.0e-08 in the relative residual. An iteration ceiling of 5,000 steps acts as a safeguard against non-convergent configurations. For problems without a provided right-hand side, we set b to a vector with all terms equal to 1.

5 Experimental results

5.1 Janna Problems

We evaluate the performance of different ILU implementations on four matrices from the Janna group. These matrices are list in Table 1, which are symmetric positive definite, derived from

Table 2: Preconditioner setup time and PCG iterations for Janna problems under different sweep configurations. The solver uses exact methods for triangular solves during preconditioning.

matrix	sweeps	DAS-ILU setup time	DAS-ILU iter	Chow-Patel setup time	Chow-Patel iter
Hook_1498 $\gamma = 0.915$	1	0.1109	5000+	0.1038	5000+
	2	0.1313	896	0.1366	769
	3	0.1470	807	0.1738	771
	4	0.1668	804	0.2061	791
	5	0.1843	804	0.2438	769
	HYPRE	0.3357	1137	0.3357	1137
	Serial	2.0806	738	2.0806	738
Flan_1565 $\gamma = 0.930$	1	0.1989	5000+	0.2003	5000+
	2	0.2646	2434	0.3035	2486
	3	0.3299	2372	0.4165	2435
	4	0.3889	2372	0.5194	2386
	5	0.4644	2372	0.6168	2384
	HYPRE	0.9951	2865	0.9951	2865
	Serial	4.5743	1794	4.5743	1794
Geo_1438 $\gamma = 0.894$	1	0.1123	373	0.1126	5000+
	2	0.13284	353	0.1531	325
	3	0.15552	351	0.1861	323
	4	0.17455	348	0.2198	323
	5	0.19385	348	0.2578	322
	HYPRE	0.3609	360	0.3609	360
	Serial	2.2836	303	2.2836	303
Emilia_923 $\gamma = 0.869$	1	0.07849	5000+	0.07397	5000+
	2	0.0967	5000+	0.10451	5000+
	3	0.10846	1193	0.12797	5000+
	4	0.12141	1190	0.15841	1092
	5	0.13601	1190	0.17981	1090
	HYPRE	0.23653	5000+	0.23653	5000+
	Serial	1.52744	143	1.52744	143

Table 3: Solve time and iterations for Janna problems with different Jacobi steps for triangular solve. 5 fixed-point sweeps is used to setup the preconditioner.

steps	DAS-ILU solve time	DAS-ILU iter	Chow-Patel solve time	Chow-Patel iter
1	10.759	1430	10.351	1400
2	7.0193	809	9.5764	776
3	7.3595	807	13.132	770
4	7.8011	807	16.741	770
5	8.1113	807	20.143	770
HYPRE	28.012	1137	28.012	1137
Serial	104.59	738	104.59	738
1	44.081	3756	41.979	3685
2	32.563	2471	45.301	2398
3	32.662	2372	62.106	2402
4	34.282	2370	79.570	2404
5	35.726	2373	95.482	2402
HYPRE	128.39	2865	128.39	2865
Serial	457.68	1794	457.68	1794
1	6.7594	994	7.1646	1061
2	2.9018	368	3.8515	338
3	2.9207	348	5.0025	322
4	3.0671	348	6.2754	322
5	3.1801	351	7.6512	322
HYPRE	9.6894	360	9.6894	360
Serial	46.795	303	46.795	303
1	18.0645	5000+	17.9475	5000+
2	20.2883	5000+	29.2282	5000+
3	8.70918	2069	25.7177	3327
4	5.1646	1190	10.6674	1139
5	5.38941	1190	12.0291	1089
HYPRE	97.836	5000+	97.836	5000+
Serial	11.328	143	11.328	143

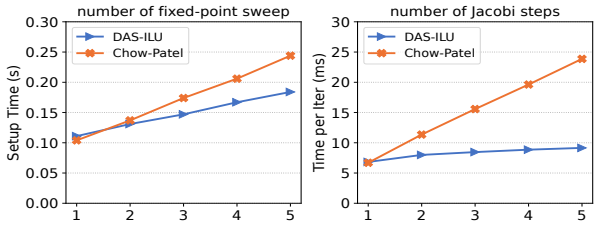


Figure 6: Setup and iteration time of HOOK_1498: Setup time for up to 5 fixed-point sweeps, time per iter in PCG for up to 5 Jacobi steps (right).

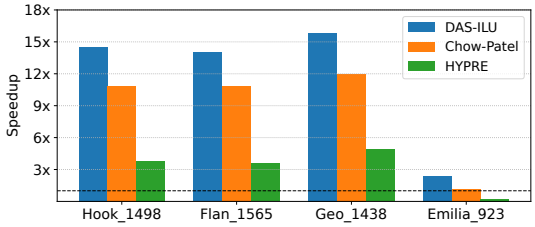


Figure 7: Speedup of ILU implementations relative to serial execution for the four Janna matrices.

finite element simulations of gas reservoir dynamics and structural mechanics problems. They exhibit inherent block structures with a block size of 3, corresponding to three displacement unknowns per grid node. Therefore, block-aware DAS-ILU and Chow-Patel has particular advantages for these problems over HYPRE.

Experiments are conducted with one MPI rank per core on a single node with 56 MPI processes, and the local ILU computations are performed in a single-threaded manner. We evaluate the PCG solver with ILU(1) preconditioner for each implementation.

5.1.1 Setup Time. Table 2 shows the preconditioner setup time and PCG iterations with different fixed-point sweeps configurations. Preconditioning in solver uses exact methods for triangular solving. Performance benchmarks against serial and HYPRE implementations are provided for reference. We introduce the metric $\gamma \in [0, 1]$ to quantify the proportion of interior rows in the DAS-ILU decomposition, defined as:

$$\gamma = \frac{\text{number of interior rows}}{\text{total matrix rows}}.$$

This metric reflects the parallelization potential of the system, with $\gamma \to 1$ indicating optimal domain partitioning where most rows can be processed without cross-node dependencies. In all four problems, the proportion of interior rows reached more than 0.85, which demonstrates that DAS-ILU maintains significant performance advantages over Chow-Patel.

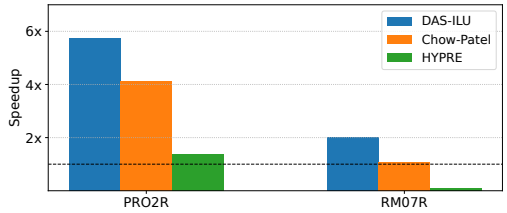


Figure 8: Speedup relative to serial implementation for two matrices from Fluorem group.

From the results of Table 2, DAS-ILU exhibits slightly higher iteration counts than Chow-Patel in most experimental cases (with the exception of Flan_1565), while the serial implementation achieves minimal iteration counts and HYPRE demonstrates the most iterations. This contrast becomes particularly pronounced in the Emilia_923 case, where both DAS and Chow-Patel methods required 7-8 times more iterations than the serial approach, while HYPRE failed to converge within 5,000 iterations. Both DAS-ILU and Chow-Patel preconditioners show stabilized convergence after 3 fixed-point sweeps, though Chow-Patel requires a minimum of 4 sweeps for successful convergence in the Emilia_923 problem. At 5 sweeps, DAS-ILU achieves a 1.32x setup speedup over Chow-Patel and 1.7–2.1x over HYPRE, with >10x acceleration compared to serial implementation.

Figure 6 (left) demonstrates the setup times of DAS-ILU and Chow-Patel with 1 to 5 fixed-point sweeps. The DAS-ILU method demonstrates growing setup time advantages over Chow-Patel as sweep counts increase.

5.1.2 Solve Time. Table 3 presents the triangular solve times for preconditioners constructed with 5 fixed-point sweeps, evaluated across varying numbers of Jacobi steps. The results demonstrate that DAS-ILU's computational advantage over Chow-Patel grows consistently with increasing Jacobi steps. As shown in Figure 6 (right), DAS-ILU maintains near-constant solve time per iteration for the Hook_1498 problem, with marginal increases per additional step. In contrast, Chow-Patel exhibits linear time growth with step count, reaching up to 2.6× longer time at 5 steps than DAS-ILU.

While more steps generally improve convergence, both methods exhibit minimal iteration reductions beyond three steps in most scenarios. Notably, Emilia_923 requires five steps for iteration stabilization under Chow-Patel versus four in the DAS-ILU implementation. Chow-Patel achieves the minimum solve time at 2 Jacobi steps for Hook_1498 and Geo_1438, and at 1 and 4 steps for Flan_1565 and Emilia_923 respectively. For the DAS-ILU method, the first three problems all have the minimum solve time when taking 2 steps, while the Emilia_923 problem has the minimum solve time when taking 4 steps. Under these optimal conditions, DAS-ILU achieves speedups of 1.3–2.07× over Chow-Patel across different problems, with 3.3–4.0× improvements over HYPRE, and 14–16× acceleration relative to serial implementation (except Emilia_923 which shows about 2× due to its ill-conditioned nature).

Figure 7 presents the total speedup (setup + solve time) relative to serial implementation. Despite employing five fixed-point sweeps during preconditioner construction, both Chow-Patel and DAS-ILU maintain computationally efficient setup phases. Note that HYPRE fails to converge for Emillia_923, whereas DAS-ILU converges in 1,190 iterations despite identical block structure. This underscores DAS-ILU's enhanced robustness for ill-conditioned systems.

5.2 Fluorem Problems

The two matrices used for evaluation are listed in Table 1. The equations involved are the compressible Reynolds Averaged Navier-Stokes (RANS) equations. They are nonsymmetric and indefinite, exhibiting a block structure with original block size 7.

We conducted the experiments on a single node with 56 processes. Both matrices present significant solving challenges, requiring at minimum ILU(2) preconditioning to ensure convergence according to our experiments. PR02R can be successfully solved using the FGMRES(100) with ILU(2) preconditioner. However, RM07R proves problematic for FGMRES across all four ILU implementations. BiCGSTAB emerges as the viable alternative, achieving convergence in serial, Chow-Patel, and DAS-ILU implementations. HYPRE fails to converge even with BiCGSTAB.

Table 4 details preconditioner setup times and iteration counts under various fixed-point sweeps. To ensure accurate iteration counts during the solution phase, we employ exact triangular solves when applying the ILU preconditioner. For the RM07R matrix, the low interior proportion $\gamma = 0.591$ results in a large separator region with significant inter-processor dependencies. The sweeps requirements for Chow-Patel implementation are significantly higher, which requires at least 27 sweeps to achieve convergence, while the DAS-ILU requires 6 sweeps. Hypre failed to converge on this problem.

Table 5 evaluates triangular problem solving times using 5-sweep preconditioners with varying Jacobi steps. For the PR02R problem, both DAS-ILU and Chow-Patel use 5 sweeps to setup the preconditioner. While for RM07R, Chow-Patel uses 28 sweeps, and 6 sweeps for DAS-ILU. For both implementations, the solution time is minimized for both problems when the number of Jacobi steps is set to three. While Chow-Patel shows slightly better solve times in some configurations, DAS-ILU demonstrates superior total time performance due to its reduced setup overhead. The total speedup relative to serial implementation is shown in Figure 8. The DAS-ILU demonstrates notable performance advantages across both matrices. For the PR02R problem, DAS-ILU achieves speedup factors of 1.46× against Chow-Patel, 4.11× over HYPRE, and 5.71× compared to the serial implementation. While for RM07R problem, DAS-ILU attains a 1.85× speedup over Chow-Patel and a 2.01× speedup over the serial implementation.

5.3 3D Convection-diffusion Problem

In this section, we test the 3D convection-diffusion equation

$$-\Delta u + \alpha \nabla \cdot u = f, \tag{13}$$

with Dirichlet boundary conditions. The diffusion part is discretized using central finite differences, while upwind finite differences are used for the advection term. This results in an asymmetric problem. The parameter α parameterizes the strength of the advection term. As α is increased, the matrix becomes more nonsymmetric. We tested the solution performance when the advection coefficient was set to $\alpha = 500$, 1500 and 2500 respectively. The global problem size

Table 4: Preconditioner setup time and solver iterations for Fluorem problems with different fixed-point sweeps configurations. The PR02R problem employs the FGMRES solver, while RM07R utilizes BiCGSTAB. Preconditioning in solver uses exact methods for triangular solving. Note that RM07R for Chow-Patel requires at least 27 sweeps.

	DAS-ILU			Chow-Patel		
matrix	sweeps	setup time	iter	sweeps	setup time	iter
PR02R $\gamma = 0.842$	1	0.0441	5000+	1	0.0349	5000+
	2	0.0496	482	2	0.0536	5000+
	3	0.0635	488	3	0.0703	489
	4	0.0771	488	4	0.0912	464
	5	0.0925	488	5	0.1242	464
	HYPRE	0.2401	1074	HYPRE	0.2401	1074
	Serial	0.8041	142	Serial	0.8041	142
RM07R $\gamma = 0.591$	3	2.7357	5000+	26	20.723	5000+
	4	3.5242	5000+	27	21.689	361
	5	4.3554	513	28	22.577	322
	6	5.1459	455	29	23.790	328
	7	5.9529	451	30	24.879	347
	HYPRE	2.8372	5000+	HYPRE	2.8370	5000+
	Serial	11.235	110	Serial	11.235	110

Table 5: Solve times and iteration counts for Fluorem benchmark problems with the different Jacobi steps. PR02R employs the FGMRES solver with 5 sweeps for preconditioner setup. RM07R utilizes BiCGSTAB, with 28 sweeps for the Chow-Patel preconditioner and 6 sweeps for DAS-ILU.

	DAS-ILU		Chow-Patel	
steps	solve time	iter	solve time	iter
1	8.9473	5000+	8.8593	5000+
2	11.006	5000+	14.699	5000+
3	1.1326	487	1.6626	464
4	1.1968	489	1.9505	456
5	1.2724	488	2.2259	459
HYPRE	4.7982	1074	4.7982	1074
Serial	6.2013	142	6.2013	142
1	45.229	5000+	43.523	5000+
2	13.481	774	86.363	5000+
3	10.638	447	9.9606	341
4	12.452	421	14.614	376
5	15.477	436	16.962	346
HYPRE	798.71	5000+	798.71	5000+
Serial	24.223	104	24.223	104

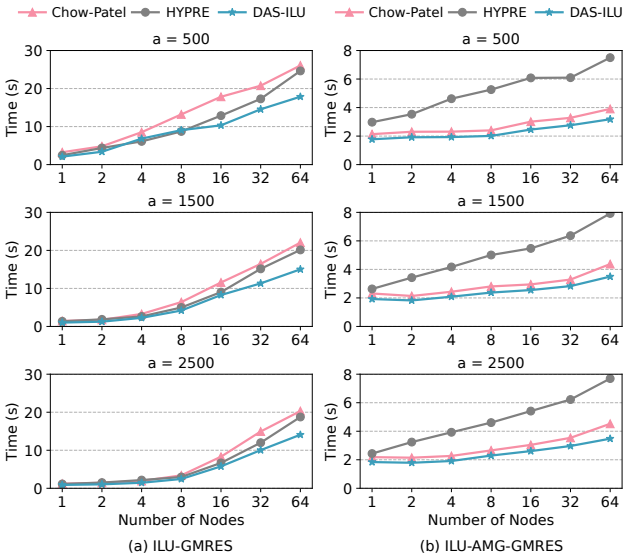


Figure 9: Total time of ILU-FGMRES and AMG-FGMRES solvers applied to 3D convection-diffusion benchmark problems with varying advection coefficients, on a 50 × 50 × 50 cube per MPI process across 1 to 64 nodes.

was set to a cubic area of $800 \times 800 \times 700$. 64 nodes with a total of 3584 cores were used, and each core with an MPI process.

We conduct weak scaling experiments from 1 to 64 nodes, maintaining a constant subdomain size of $50 \times 50 \times 50$ grid points per MPI process. Each node contains 56 MPI processes, resulting in a total of up to 3584 processes for the 64-node configuration. The global problem size scales proportionally with the number of processes.

All methods use FGMRES(40) with ILU(1) preconditioner. We use three fixed-point sweeps for ILU factorization, and three Jacobi steps in preconditioning for DAS-ILU and Chow-Patel. The ILU type in HYPRE is selected as 30 (RAS-ILUK). Our implementation additionally introduces algebraic multigrid (AMG) preconditioning, typically enabling accelerated solution convergence. The DAS-ILU and Chow-Patel implementations use YHAMG as their AMG framework, while the HYPRE implementation calls its BoomerAMG. The AMG configuration employs HMIS coarsening with ext+i interpolation implemented through matrix-matrix multiplication (entries truncated to four elements per row maximum), augmented by one aggressive coarsening stage using a strength threshold of 0.3. The relaxation scheme employs ILU(0) smoothing on the finest level with symmetric Gauss-Seidel relaxation for coarse-level. This configuration exhibited superior computational efficiency for HYPRE in our benchmarks.

Figure 9 summarizes the total computation times across different node counts and advection coefficients. For the ILU-FGMRES method, DAS-ILU consistently outperforms competing approaches across all scaling configurations. At 64 nodes, DAS-ILU achieves speedups of 1.36–1.47× over Chow-Patel and 1.32–1.43× over HYPRE across different advection coefficients. However, under the weak scaling setup, the ILU-FGMRES method exhibits severe convergence degradation as the global problem size increases, with iteration counts rising by 7.27–14.42× when scaling from 1 to 64 nodes.

In contrast, the AMG-FGMRES method demonstrates superior scalability characteristics. At 64 nodes, DAS-ILU with AMG preconditioning achieves speedups of 1.21–1.28× compared to Chow-Patel and 2.29–2.35× relative to HYPRE. AMG-FGMRES exhibits much better convergence stability, with iteration counts increasing by only 25%–50% when scaling from single-node to 64-node configurations.

5.4 SPE10 Problem

Multicomponent flow models involve the coupling of nonlinear PDEs, which are solved implicitly via block-structured Jacobian systems. We evaluate performance using the oil-water SPE10

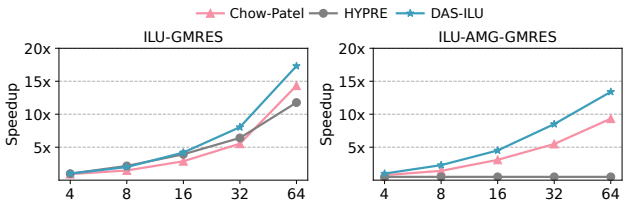


Figure 10: Strong scaling performance of ILU-FGMRES and AMG-FGMRES implementations on the SPE10 benchmark, showing total solution time as a function of node count from 4 to 64 nodes. HYPRE's AMG-FGMRES fails to converge.

benchmark [11], characterized by strong heterogeneity, with the grid refined to a resolution of 480 × 880 × 340. Each grid node contains three displacement unknowns, resulting in approximately 358 million total unknowns. The problem is executed across 4 to 64 compute nodes (224 to 3,584 cores), with one MPI process assigned per core to assess strong scaling behavior.

All implementations use FGMRES(40) with three fixed-point sweeps and three Jacobi steps for the ILU preconditioner. We apply ILU(1) for both the DAS-ILU and Chow–Patel ILU implementations. In HYPRE, ILU(1) fails to converge, so ILUT(1000, 0.01) is used instead. We also attempt to solve this problem using AMG-FGMRES. In YHAMG, the AMG configuration employs a block-aggregate multigrid method. The relaxation scheme applies ILU(1) smoothing on the finest level and symmetric Gauss–Seidel relaxation on the coarse levels. The cycle type is set to the more robust K-cycle [27]. In contrast, in our tests, both BoomerAMG and the block-based MGR methods in HYPRE fail to converge for this problem.

Figure 10 presents the strong scaling performance, showing speedups relative to the 4-node DAS-ILU baseline. For the ILU-FGMRES method, DAS-ILU demonstrates excellent strong scaling behavior, achieving a 17.3× speedup when scaling from 4 to 64 nodes. At 64 nodes, DAS-ILU maintains its performance advantage with speedups of 1.21× over Chow-Patel and 1.47× over HYPRE. For the AMG-FGMRES method, DAS-ILU achieves a 13.4× speedup from 4 to 64 nodes and maintaining a 1.2× advantage over Chow-Patel at 64 nodes.

6 Related Works

Incomplete LU (ILU) factorization remains one of the most widely used preconditioning strategies for sparse linear systems [33, 35]. Parallelization of ILU factorization has been a topic of much interest since the 1980s. Early efforts to parallelize ILU focused on graph-based reordering strategies such as level scheduling and multicolor ordering, which aim to expose concurrency by manipulating the matrix structure [22, 28, 36]. Although effective for a certain class of problems, these approaches often degrade the quality of the preconditioner, particularly for nonsymmetric or irregular matrices [14].

To improve scalability, domain decomposition methods have been employed to localize ILU factorizations. These methods partition the global problem into smaller subdomains, allowing independent ILU application on each subdomain. Additive Schwarz-type methods, in particular, are popular in distributed memory environments, where each subdomain can be assigned to a different processor [19, 29]. The HYPRE library [15, 16] incorporates such strategies by offering ILU variants like ILUK and ILUT within additive Schwarz frameworks. However, interface coupling between subdomains introduces synchronization overheads and often necessitates communication-heavy global solves. Saad proposes multilevel ILU methods with ideas that relate to hierarchical or domain decomposed strategies [32].

Algebraic Multigrid (AMG) is a highly efficient algorithm for solving large-scale sparse linear systems [3, 26]. Its core idea involves constructing a hierarchy of grid levels, where coarse grids rapidly eliminate low-frequency errors and fine grids address high-frequency errors, thereby accelerating iterative convergence. Integrating ILU with AMG to improve preconditioning effectiveness has become a significant research direction and has been included in HYPRE [17] and YHAMG [18, 40].

Recent research has explored the combination of ILU with domain decomposition to better balance local computation and communication. Yet, most existing methods either require complex synchronization protocols or fall short in handling highly irregular systems. The Restricted Additive Schwarz (RAS) approach of Cai and Sarkis [6] attempted to address this by minimizing overlap and reducing the volume of communication, but its convergence performance remains sensitive to interface approximations. Point-block ILU preconditioner [23, 24, 35] for SIMD acceleration is studied to solve multiphysics coupled PDE problems.

Our work builds upon these developments by proposing a distributed asynchronous ILU (DAS-ILU) framework that integrates domain decomposition, hierarchical reordering, and lightweight asynchronous communication. Unlike prior methods, DAS-ILU decouples interior and interface computations using a rank-based partitioning strategy and reduces the volume of communicated messages. This design improves scalability and preserves robustness even on indefinite problems such as those arising from compressible flow simulations [29].

7 Conclusion

This paper presents **DAS-ILU**, a distributed asynchronous ILU preconditioner designed to overcome the scalability and synchronization challenges of traditional ILU algorithms on modern multicore and distributed-memory architectures. By tightly integrating domain decomposition with a two-level asynchronous iteration framework, DAS-ILU effectively decouples interior and separator computations, significantly reducing global synchronization overhead while preserving the robustness of ILU-based preconditioners. These contributions collectively enable scalable and efficient preconditioning for both symmetric positive-definite and nonsymmetric indefinite systems.

Experimental validation across over 10 large-scale benchmarks, including indefinite systems from compressible Navier-Stokes and reservoir models with up to 358 million unknowns, demonstrates DAS-ILU's robustness and efficiency. DAS-ILU achieves 1.3–2.07× speedup over Chow-Patel and up to 4.11× speedup over HYPRE across different problem types. On 3,584-core clusters, DAS-ILU maintains 1.2–1.47× performance advantages. Notably, DAS-ILU succeeds where HYPRE fails, particularly for challenging indefinite systems like Emilia_923 and RM07R.

Looking forward, this work opens up promising directions for further research. Future extensions will explore *adaptive fill-level strategies*, *mixed-precision factorizations*, and *GPU-accelerated implementations* to further enhance performance. Moreover, integration with *multiphysics solvers* and *learning-based auto-tuning* of iteration parameters presents new opportunities for expanding DAS-ILU's applicability in exascale scientific computing [1].

Acknowledgments

The authors would like to thank the referees for their valuable comments. This work is supported by National Key R&D Program of China (2023YFB3001702), NSF of China (No. 62032023, 42104078, and 61902411).

References

[1] Hartwig Anzt, Erik Boman, Rob Falgout, Pieter Ghysels, Michael Heroux, Xiaoye Li, Lois Curfman McInnes, Richard Tran Mills, Sivasankaran Rajamanickam, Karl Rupp, et al. 2020. Preparing sparse solvers for exascale computing. *Philosophical Transactions of the Royal Society A* 378, 2166 (2020), 20190053.

[2] Hartwig Anzt, Edmond Chow, and Jack Dongarra. 2018. ParILUT—A new parallel threshold ILU factorization. *SIAM Journal on Scientific Computing* 40, 4 (2018), C503–C519.

[3] William L Briggs, Van Emden Henson, and Steve F McCormick. 2000. *A multigrid tutorial*. SIAM.

[4] Xiao-Chuan Cai, William D Gropp, and David E Keyes. 1994. A comparison of some domain decomposition and ILU preconditioned iterative methods for nonsymmetric elliptic problems. *Numerical Linear Algebra with Applications* 1, 5 (1994), 477–504.

[5] Xiao-Chuan Cai and Marcus Sarkis. 1999. A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems. *SIAM Journal on Scientific Computing* 21, 2 (1999), 792–797. https://doi.org/10.1137/S106482759732678X

[6] Xiao-Chuan Cai and Marcus Sarkis. 1999. A restricted additive Schwarz preconditioner for general sparse linear systems. *Siam journal on scientific computing* 21, 2 (1999), 792–797.

[7] L. M. Carvalho, L. Giraud, and P. Le Tallec. 2001. Algebraic Two-Level Preconditioners for the Schur Complement Method. *SIAM Journal on Scientific Computing* 22, 6 (2001), 1987–2005. https://doi.org/10.1137/S1064827593409809

[8] J. Cerdán, J. Marín, and J. Mas. 2017. A two-level ILU preconditioner for electromagnetic applications. *J. Comput. Appl. Math.* 309 (2017), 371–382. https://doi.org/10.1016/j.cam.2016.03.012

[9] Andrew Chapman, Yousef Saad, and Larry Wigton. 2000. High-order ILU preconditioners for CFD problems. *International Journal for numerical methods in fluids* 33, 6 (2000), 767–788.

[10] Edmond Chow and Aftab Patel. 2015. Fine-Grained Parallel Incomplete LU Factorization. *SIAM Journal on Scientific Computing* 37, 2 (2015), C169–C193. https://doi.org/10.1137/140968896

[11] M. A. Christie and M. J. Blunt. 2001. Tenth SPE Comparative Solution Project: A Comparison of Upscaling Techniques. *SPE Reservoir Evaluation & Engineering* 4, 04 (08 2001), 308–317. https://doi.org/10.2118/72469-PA

[12] T. Davis and Y. Hu. 2011. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.* 38, 1 (2011), 1:1–1:25.

[13] Timothy A Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 1–25.

[14] Iain S Duff and Gerard A Meurant. 1989. The effect of ordering on preconditioned conjugate gradients. *BIT Numerical Mathematics* 29 (1989), 635–657.

[15] Robert D Falgout, Jim E Jones, and Ulrike Meier Yang. 2006. The design and implementation of hypre, a library of parallel high performance preconditioners. In *Numerical solution of partial differential equations on parallel computers*. Springer, 267–294.

[16] Robert D Falgout, Ruipeng Li, Björn Sjögreen, Lu Wang, and Ulrike Meier Yang. 2021. Porting hypre to heterogeneous computer architectures: Strategies and experiences. *Parallel Comput.* 108 (2021), 102840.

[17] Robert D Falgout and Ulrike Meier Yang. 2002. hypre: A library of high performance preconditioners. In *International Conference on computational science*. Springer, 632–641.

[18] Yuan Fan and Li Shengguo. 2022. YHAMG. https://gitee.com/e-level-parallel-algorithm/yhamg

[19] Patricia Gonzalez, Jose C Cabaleiro, and Tomas F Pena. 1999. Parallel incomplete LU factorization as a preconditioner for Krylov subspace methods. *Parallel processing letters* 9, 04 (1999), 467–474.

[20] Carlo Janna and Massimiliano Ferronato. 2011. Adaptive Pattern Research for Block FSAI Preconditioning. *SIAM Journal on Scientific Computing* 33, 6 (2011), 3357–3380. https://doi.org/10.1137/100810368

[21] Carlo Janna, Massimilano Ferronato, and Giuseppe Gambolati. 2010. A Block FSAI-ILU Parallel Preconditioner for Symmetric Positive Definite Linear Systems. *SIAM Journal on Scientific Computing* 32, 5 (2010), 2468–2484.

[22] Mark T Jones and Paul E Plassmann. 1994. Scalable iterative solution of sparse linear systems. *Parallel Comput.* 20, 5 (1994), 753–773.

[23] Aditya Kashi and Sivakumaran Nadarajah. 2021. An Asynchronous Incomplete Block LU Preconditioner for Computational Fluid Dynamics on Unstructured Grids. *SIAM Journal on Scientific Computing* 43, 1 (2021), C1–C30. https://doi.org/10.1137/19M1301084

[24] Wenpeng Ma and Xiao-Chuan Cai. 2021. Point-block incomplete LU preconditioning with asynchronous iterations on GPU for multiphysics problems. *The International Journal of High Performance Computing Applications* 35, 2 (2021), 121–135. https://doi.org/10.1177/1094342020981153

[25] M Magolu monga Made and Henk A van der Vorst. 2001. A generalized domain decomposition paradigm for parallel incomplete LU factorization preconditionings. *Future Generation Computer Systems* 17, 8 (2001), 925–932.

[26] M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh, V. Sellappan, and R. Strzodka. 2015. AmgX: A Library for GPU Accelerated Algebraic Multigrid and Preconditioned Iterative Methods. *SIAM Journal on Scientific Computing* 37, 5 (2015), S602–S626. https://doi.org/10.1137/140980260

[27] Yvan Notay and Panayot S Vassilevski. 2008. Recursive Krylov-based multigrid cycles. *Numerical Linear Algebra with Applications* 15, 5 (2008), 473–487.

[28] Francois Pacull, Stéphane Aubert, and Martin Buisson. 2011. A Study of ILU Factorization for Schwarz Preconditioners with Application to Computational Fluid Dynamics. *Civil-Comp Proceedings* 95. https://doi.org/10.4203/ccp.95.39

[29] F Pacull, S Aubert, and M Buisson. 2011. A study of ilu factorization for schwarz preconditioners with application to computational fluid dynamics. In *Proceedings of the second international conference on parallel, distributed, grid and cloud computing for engineering, Civil-Comp Press, Stirlingshire, UK*.

[30] Eugene L Poole and James M Ortega. 1987. Multicolor ICCG methods for vector computers. *SIAM J. Numer. Anal.* 24, 6 (1987), 1394–1418.

[31] Youcef Saad. 1993. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific Computing* 14, 2 (1993), 461–469.

[32] Yousef Saad. 1996. ILUM: A multi-elimination ILU preconditioner for general sparse matrices. *SIAM Journal on Scientific Computing* 17, 4 (1996), 830–847.

[33] Yousef Saad. 2003. *Iterative methods for sparse linear systems*. SIAM.

[34] Stefan Sauter. 1991. The ILU method for finite-element discretizations. *Journal of computational and applied mathematics* 36, 1 (1991), 91–106.

[35] Kengo Suzuki, Takeshi Fukaya, and Takeshi Iwashita. 2023. A novel ILU preconditioning method with a block structure suitable for SIMD vectorization. *J. Comput. Appl. Math.* 419 (2023), 114687.

[36] Henk A van der Vorst. 1989. High performance preconditioning. *SIAM J. Sci. Statist. Comput.* 10, 6 (1989), 1174–1185.

[37] Henk A Van der Vorst. 1992. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing* 13, 2 (1992), 631–644.

[38] Tianshi Xu, Vassilis Kalantzis, Ruipeng Li, Yuanzhe Xi, Geoffrey Dillon, and Yousef Saad. 2022. parGeMSLR: A parallel multilevel Schur complement low-rank preconditioning and solution package for general sparse matrices. *Parallel Comput.* 113 (2022), 102956.

[39] Tianshi Xu, Rui Peng Li, and Daniel Osei-Kuffuor. 2023. A two-level GPU-accelerated incomplete LU preconditioner for general sparse linear systems. *The International Journal of High Performance Computing Applications* (2023), 10943420251319334.

[40] Xiaojian Yang, Shengguo Li, Fan Yuan, and Dezun Dong. 2024. DBSR: An Efficient Storage Format for Vectorizing Sparse Triangular Solvers on Structured Grids. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–14.

[41] Fan Yuan, Xiaojian Yang, Shengguo Li, Dezun Dong, Chun Huang, and Zheng Wang. 2024. Optimizing Multi-Grid Preconditioned Conjugate Gradient Method on Multi-Cores. *IEEE Transactions on Parallel and Distributed Systems* 35, 5 (2024), 768–779. https://doi.org/10.1109/TPDS.2024.3372473

Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

A Overview of Contributions and Artifacts

A.1 Paper's Main Contributions

- **C_1** Efficient Domain Decomposition: Partitions the computational domain into interior and separator regions, minimizing cross-processor dependencies while reducing separator size by approximately 50%.
- **C_2** Communication-aware Asynchronous Iteration: Integrates domain decomposition with asynchronous communication, utilizing parallel factorization for interior rows and fixed-point iterations for separator rows.
- **C_3** Block-structured Support: Extends support to block-structured matrices using exact block inversions, improving efficiency for coupled PDE systems.

A.2 Computational Artifacts

A_1 https://doi.org/10.5281/zenodo.15294834

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1, C_2, C_3	Tables 2-6 Figures 1-10

B Artifact Identification

B.1 Computational Artifact A_1

Relation To Contributions

The artifact A_1 implements all three main contributions:

- **C_1** Implements efficient domain partitioning into interior and separator regions using ParMETIS. Results in Tables 2-3 ($\gamma > 0.85$) show reduced separator size and improved load balancing.
- **C_2** Implements asynchronous communication for ILU factorization and triangular solves. Efficiency demonstrated in Figures 7-10.
- **C_3** Supports block-structured matrices with exact block inversions. Evident in SPE10 results (Figure 10) and Fluorem matrices (Tables 5-6).

Expected Results

DAS-ILU demonstrates superior performance, achieving speedups of up to 2.07× over Chow-Patel's fine-grained parallel ILU and up to 4.11× over HYPRE's ILU across various problem types. It exhibits strong scalability on distributed systems with more than 3,500 cores and maintains robustness for challenging nonsymmetric and indefinite systems.

Expected Reproduction Time (in Minutes)

The execution time for this artifact typically ranges from 5 to 20 minutes. The precise duration varies significantly depending on the problem size and the number of processors utilized in parallel. Detailed execution times for various tasks are provided at the conclusion of each experimental run.

Artifact Setup (incl. Inputs)

Hardware. Evaluation was conducted on a high-performance cluster comprising 64 nodes, each equipped with dual Intel Xeon Gold 6348 processors (56 cores/node), operating at a 2.60 GHz base frequency, and 128GB of DDR4-3200 memory.

Software. The artifact requires MPICH 4.1.2 and the Intel 2023 compiler. Hypre (available at github.com/hypre-space/hypre) is also necessary as a competitive baseline for testing and comparison.

Datasets / Inputs. The evaluation utilizes matrices from the SuiteSparse Collection (sparse.tamu.edu), the SPE10 model, and 3D convection-diffusion systems generated using Hypre's GenerateDifConv interface.

Installation and Deployment. The artifact is built using MPICH 4.1.2 and Intel C++ Compiler 2023 (ICC 2023.2.0) with -O3 optimization.

Artifact Execution

The workflow of this artifact is divided into four tasks: T1, T2, T3, T4. Their dependencies are:

$$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4.$$

- **T_1** Read the matrix and right-hand side vector files and distributes the data to each process.
- **T_2** Repartition and redistribute the matrix and right-hand side vector using ParMETIS.
- **T_3** Setup the ILU or AMG preconditioner with the input configuration.
- **T_4** Execute Preconditioned CG/GMRES solver to solve the system.

The numerical solver employs a relative residual norm reduction threshold of 1.0e-08 with a 5,000-iteration safeguard. ILU preconditioning uses fill level 1-2 factorization with RCM reordering for local blocks, applying 2-5 fixed-point sweeps and Jacobi steps. AMG utilizes HMIS coarsening with ext+i interpolation (maximum 4 elements per row), plus one aggressive coarsening stage with 0.3 strength threshold. Relaxation applies ILU(0) smoothing on the finest level and symmetric Gauss-Seidel on coarse levels. Each configuration is repeated five times for statistical validity.

Artifact Analysis (incl. Outputs)

Results demonstrate DAS-ILU's significant performance advantages across various problem types. Tables 2-6 showcase speedups on Janna and Fluorem matrices, with DAS-ILU achieving up to 2.07× improvement over Chow-Patel and 4.11× over HYPRE. Figure 9 illustrates the method's scalability for 3D convection-diffusion problems, demonstrating 1.52-1.59× speedup over Chow-Patel and 1.67-1.77× over HYPRE. Figure 10 presents DAS-ILU's efficiency on the SPE10 model, showing a 1.6× speedup over Chow-Patel and 1.44× over HYPRE.

Artifact Evaluation (AE)

The instructions of how to setup, execute, and analyze the provided artifacts is included in the subsections of the artifact description part above.

.